However, in yet other alternate embodiments, each image file could be organized in "quality level support order" with the data for each analysis array being arranged so that each successive data structure 172 stores the image information needed to increase image quality by one predefined image quality level. Thus, the information in some data structures 172 might represent two, three or more bit planes of information. In this embodiment, an image can be reduced by one quality level by deleting the last data structure 172 from every analysis array data structure 172 in the image file.

Digital Camera State Machines

For the purposes of this explanation, it will be assumed that the digital camera 100 has four predefined image quality levels: High, Very Good +, Very Good -, and Good. It will be further assumed that image files stored at High quality typically occupy about twice as much space as image files stored at Good quality. In other embodiments, more or fewer image quality levels could be used, and the ratio of image file sizes from highest to lowest quality could be larger or smaller than 2:1. For instance, if the camera is capable of taking very high resolution images, such as 2000 x 2000 pixels or even 4000 x 4000 pixels, and at very high fidelity, then it would make sense to provide a large number of quality levels with a ratio of image file sizes from highest to lowest quality of perhaps as high as 64:1.

It is noted that an image file's quality cannot be increased once it has been lowered, unless the original image file or an alternate source thereof remains available, because the information needed to restore the image's quality has been lost.

Referring back to Fig. 1, the digital camera 100 preferably includes data processing circuitry 106 for performing a predefined set of primitive operations, such as performing the multiply and addition operations required to apply a transform to a certain amount of image data, as well as a set of state machines 200-212 for controlling the data processing circuitry so as to perform a set of predefined image handling operations. In one embodiment, the state machines in the digital camera are as follows.

- One or more state machines 200 for transforming, compressing and storing an image received from the camera's image capture mechanism. This image is sometimes called the "viewfinder" image, since the image being processed is generally the one seen on the camera's

image viewer 114. This set of state machines 200 are the ones that initially generate each image file stored in the nonvolatile image memory 108. Prior to taking the picture, the user specifies the quality level of the image to be stored, using the camera's buttons 112. It should noted that in most digital cameras the viewfinder is capable of displaying only a very small and low fidelity version of the captured image, and thus the image displayed in the camera's viewfinder is typically a much lower quality rendition of the captured image than the quality of the "viewfinder" image stored in the image file.

- One or more state machines 202 for decompressing, inverse transforming and displaying a stored image file on the camera's image viewer. The reconstructed image generated by decompressing, inverse transforming and dequantizing the image data is stored in camera's framebuffer 118 so that it can be viewed on the image viewer 114.
- One or more state machines 204 for updating and displaying a count of the number of images stored in the nonvolatile image memory 108. The image count is preferably displayed on the user interface display 116. This set of state machines 204 will also typically indicate what percentage of the nonvolatile image memory 108 remains unoccupied by image files, or some other indication of the camera's ability to store additional images. If the camera does not have a separate interface display 116, this memory status information may be shown on the image viewer 114, for instance superimposed on the image shown in the image viewer 114 or shown in a region of the viewer 114 separate from the main viewer image.
- One or more state machines 206 for implementing a "viewfinder" mode for the camera in which the image currently "seen" by the image capture mechanism 102 is displayed on the image viewer 114 to that the user can see the image that would be stored if the image capture button is pressed. These state machines transfer the image received from the image capture device 102, possibly after appropriate remedial processing steps are performed to improve the raw image data, to the camera's framebuffer 118.
- One or more state machines 208 for downloading images from the nonvolatile image memory 108 to an external device, such as a general purpose computer.
- One or more state machines 210 for uploading images from an external device, such as a general purpose computer, into the nonvolatile image memory 108. This enables the camera to be used as an image viewing device, and also as a mechanism for transferring image files on memory cards.
- One or more state machines 212 for reducing the size of image files in the nonvolatile image memory 108. This will be described in more detail next.

In the context of the present invention, an image file's quality level can be reduced in one of two ways: 1) by deleting from the image file all the analysis arrays associated with one or more transform layers, or 2) by deleting from the image file one or more bit planes of data. In either method, the state machines 212 extract the data structures of the image file that correspond to the new, lower image quality level selected by the user, and then replaces the original image file with one that stores the extracted data structures. Alternately, the original image file is updated by deleting a portion of its contents, thereby freeing some of the memory previously occupied by the file. A feature of the present invention is that the image quality level of an image file can be lowered without having to reconstruct the image and then re-encode it, which would be costly in terms of the computational resources used. Rather, the data structures within the image file are pre-arranged so that the image data in the file does not need to be read, analyzed or reconstructed. The image quality level of an image file is lowered simply by keeping an easily determined subset of the data in the image file and deleting the remainder of the data in the image file, or equivalently by extracting and storing in a new image file a determined subset of the data in the image file and deleting the original image file.

For the purposes of this document, it should be noted that the term "deleting" when applied to a data structure in an image file does not necessarily mean that the information in the data structure is replaced with null values. Rather, what this means is that the image file is replaced with another image file that does not contain the "deleted" data structure. Thus, various data structures in an image file may be deleted simply by copying all the other data structures in the image file into a new image file, and updating all required bookkeeping information in the image file and the image directory for the modified file. The "deleted" data structures may actually remain in memory unchanged until they are overwritten with new information. Alternately, data in an image file may in some implementations be deleted solely by updating the bookkeeping information for the file, without moving any of the image data.

In one embodiment of the present invention, the digital camera lowers the image quality of an image from High quality to "Very Good +" by deleting the two lowest bit planes of the image. Similarly, lowering the image's quality to "Very Good -" is accomplished by deleting two more bit planes of the image, and then lowering the image's quality to Good is accomplished by deleting yet another two bit planes of the image. More generally, each quality level

transition is represented by deleting a certain percentage of the bit planes of the highest quality level representation of the image.

In an alternate embodiment, the transition from High quality to the next highest quality level is accomplished by deleting the analysis arrays for a first transform layer (e.g., the analysis arrays for the HL1, HH1 and LH1 regions of the transformed image in Fig. 3). Subsequent quality level transitions to lower quality levels are accomplished by deleting appropriate numbers of bit planes.

In one embodiment of the present invention the digital camera provides two image file size reduction modes. In a first size reduction mode, the user selects one image (or a specified group of images), uses the camera's buttons to indicate what lower quality level the image is to be stored at, and then the state machine 212 generates a smaller image file and stores the resulting image file in the camera's nonvolatile image memory 108. In the second size reduction mode, the user commands the camera to reduce the size of all image files that are currently stored at quality level A to quality level B. For instance, in this second size reduction mode the user might command the camera to convert all "High" quality image files to "Very Good +" image quality files. This latter size reduction mode is particularly useful for "clearing space" in memory 108 to enable additional pictures to be stored in the memory 108.

In another embodiment, the camera or other device may include one or more automatic image file size reduction modes. For instance, in one such mode the camera could be set to record all pictures at a particular quality level. When the camera's memory is sufficiently filed with images files so that there is insufficient room to store one more image at the current quality level setting, the camera automatically reduces the size of enough of the stored image files so as to create room for one more image at the current quality level. In some embodiments, the quality level setting of the device for future images might be automatically reduced to match the quality level of the highest quality image stored in the camera's memory. In this way, the camera takes and stores the maximum quality images for the space available, and this maximization will occur flexibly and "on-the-fly."

Camera Operation and
Image File Size Reduction

Referring to Fig. 5, the status of a digital camera is represented by status information displayed on the camera's user interface display 116. For example, before the camera's image memory 108 is filled, the camera might indicate to the user that it is currently storing twenty-one pictures at High quality and has enough memory to store three more pictures. The indication of how many more pictures can be stored in the camera's image memory 108 (Fig. 1) depends on the camera's current picture quality setting, which determines the quality of the next picture to be taken.

After the camera has stored three more pictures, the camera's image memory 108 is full (i.e., it has insufficient room to store another picture at the camera's current picture quality setting), and the camera indicates to the user that it is currently storing twenty-four pictures at High quality and has enough memory to store zero more pictures. For the purposes of this example, we will assume that the user wants to take at least ten more pictures, despite the fact that he/she has no more memory cards. To make this possible, the user utilizes the image size reduction feature of the camera.

In this example, the user commands the camera to reduce all "High" quality image files down one quality level to the "Very Good +" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good +" quality level, and the camera has room for seven new images at the High quality image level.

In this example, the user next commands the camera to perform a second size reduction so as to compress all "Very Good +" quality image files down one quality level to the "Very Good -" quality level. The camera accomplishes this by running the size reduction state machine 212 and then updating the status information displayed on the camera's user interface display 116. In this example, the twenty-four images are now shown to be stored in image files having the "Very Good -" quality level, and the camera has room for twelve new images at the High quality image level.

Alternately, if the user had, before capturing the images, switched the quality level for new images to "Very Good +" quality, a single image size reduction step might have been sufficient to create room for at least ten additional pictures.

In another example, the digital camera may be configured to have an automatic image file size reduction mode that is activated only when the camera's memory is full and the user nevertheless presses the image capture button on the camera. In this mode of operation, the camera's image processing circuitry reduces the size of previously stored image files as little as possible so as to make room for an additional image file. If the user continues to take more pictures in this mode, the quality of the stored images will eventually degrade to some user defined or predefined setting for the lowest allowed quality level, at which point the camera will not store any additional image files until the permitted quality level is lowered further or at least some of the previously stored image files are transferred to another device or otherwise deleted.

The image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the image processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image processing procedures suitable for use by a computer system.

Video Image Management System

Referring to Fig. 6, there is shown a conceptual data flow diagram for a video image management system for storing video images at a plurality of image quality levels. The basic structure of the video image management system is the same as shown in Fig. 1. However, when the camera is a digital video camera, successive images F_i are automatically generated at a predefined rate, such as eight, sixteen, twenty-four or thirty frames per second. In a preferred embodiment, the sequence of video images is processed N frames at a time, where N is an integer greater than three, and is preferably equal to four, eight or sixteen; generally N will be determined by the availability of memory and computational resources in the particular system in which the invention is being implemented. That is, each set of N (e.g., sixteen) successive

images (i.e., frames) are processed as a set, as follows. For each set of sixteen frames F_{16x} to F_{16x+15}, all the frames except the first one are replaced with differential frames. Thus, when N=16, fifteen differential frames F_{i+1}-F_i are generated. Then, following the data processing method shown in Fig. 3 and discussed above, the first frame and the fifteen differential frames are each divided into analysis arrays, a wavelet-like or other transform is applied to the analysis arrays, and then the resulting transform coefficients are encoded.

In alternate embodiments, other methodologies could be used for initially transforming and encoding each set of success frames. For instance, a frame by frame decision might be made, based on a measurement of frame similarity or dissimilarity, as to whether or not to replace the frame with a differential frame before applying the transform and encoding steps.

In all embodiments, the image file (or files) representing the set of frames is stored so as to facilitate the generation of smaller image files with minimal computational resources. In particular, the data in the image file(s) is preferably stored using distinct data structures for each bit plane (see Fig. 4B). Furthermore, as explained above with reference to Fig. 4A, the analysis arrays may be adjusted prior to the transform step so that the boundaries between analysis arrays correspond to the boundaries between transform layer coefficients. By so arranging the data stored in the video image files, the generation of smaller, lower quality level video image files is made much easier.

Continuing to refer to Fig. 6, after the video image files for a video frame sequence have been generated at a particular initial quality level, the user of the device (or the device operating in a particular automatic mode) may decide to reduce the size of the video image files while retaining as much image quality as possible. By way of example, in a first video image file size reduction step, the HH1 transform coefficients for the last eight frames of each sixteen frame sequence are deleted. In a second reduction step, the HH1 transform coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. In a third reduction step, the Z (e.g., four) least significant bit planes of the video image files are deleted. In a fourth reduction step, the HL1 and LH1 coefficients are deleted for the last eight frames of each sixteen frame sequence. In a fifth reduction step, the HL1 and LH1 coefficients are deleted for all frames other than the first frame of each sixteen frame sequence. These reduction steps are only examples of the type of file size reduction steps that could be

performed. For instance, in other embodiments, bit planes might be deleted in earlier reduction steps and transform layers (or portions of transform layers) deleted only in later reduction steps. In general, each video image size reduction causes a corresponding decrease in image quality.

Referring to Figs. 7, 8 and 9, in another embodiment, video image sequences are compressed and encoded by performing a time domain, one dimensional wavelet transformation on a set of video frames. In particular, the video frames are divided into groups of N frames, where N is an integer greater than 3, and for every x,y pixel position in the video image, a one dimensional K level wavelet transform is performed on the pixels for a sequence of N+1 frames. For instance, the K level wavelet transform is performed on the 1,1 pixels for the last frame of the previous group and the current group of N frames, as well as the 1, 2 pixels, the 1,3 pixels and so on.

In order to avoid artifacts from the separate encoding of each group of N frames, the frame immediately preceding the current group is included in K level wavelet transform. The wavelet transform uses a short transform filter that extends only one position to the left (backwards in time) of the position for which a coefficient is being generated and extends in the right hand (forward in time) direction only to the right hand edge of the set of N frames.

Furthermore, as shown in Fig. 8, the "right edge" coefficients are saved for each of the first through K-1 level wavelet transforms for use when processing the next group of N frames. In a preferred embodiment, only the rightmost edge coefficient is saved for each of the first through K-1 level transforms; in other embodiments two or more right edge coefficients may be saved and used when processing the next block of N frames. When the second level transform is performed on a block of N frames, the saved layer 1 right edge coefficients for the previous set of N frames are used (i.e., included in the computation of the leftmost computed coefficient(s) for layer 2). By saving the rightmost edge coefficients for each of the 1 through K-1 layers, artifacts that would caused (during regeneration of the video image sequence) by the discontinuities between the last frame of one block and the first frame of the next block are avoided, resulting in a smoother and more visually pleasing reconstructed video image sequence. The wavelet-like transformation and data compression of a video sequence is shown in pseudocode form in Table 1.

Table 1
Pseudocode for Wavelet-Like Transform and
Compression of One Block of Video Frames

```
Repeat for each block of video frames:
{
For each row y (of the images)
        {
        For each column x (of the images)
                {
                Save rightmost edge value for use when processing next block of video frames;

                Apply level 1 wavelet-like transform to time-ordered sequence of pixel values
                at position x,y, including saved edge value from prior block to generate level 1
                L and H coefficients;

                Save rightmost edge L coefficient for use when processing next block of video
                frames;

                Apply level 2 wavelet-like transform to level 1 L coefficients for position x,y,
                including saved level 1 edge value from prior block to generate level 2 L and H
                coefficients;

                Save rightmost edge level 2 L coefficient for use when processing next block of
                video frames;

                ...

                Apply level k-1 wavelet-like transform to level k-2 L coefficients for position
                x,y, including saved level k-2 edge value from prior block to generate level k-1
                L and H coefficients;

                Save rightmost edge level k-1 L coefficient for use when processing next block
                of video frames;

                Apply level k wavelet-like transform to level k-1 L coefficients for position x,y,
                including saved level k-1 edge value from prior block to generate level k L and
                H coefficients;
                }
        }
Quantize coefficients
Encode coefficients
Store coefficients in image data structure(s), creating image file for current block of video
frames
}
```

A more detailed explanation of saving edge coefficient from one block of image data for use while performing a wavelet or wavelet like transforms on a neighboring block of image data is provided in U.S. patent application serial no. 09/358,876, filed 07-22-99, "Memory Saving Wavelet-Like Image Transform System and Method for Digital Camera And Other Memory Conservative Applications," which is hereby incorporated by reference as background information.

Once the wavelet-like transform of each block of video data has been completed, all other aspects of processing the transformed video data are as described above. That is, the transformed data is quantized, stored in image data structures and subject to reductions in image quality, using the same techniques as those applied to still images and video image sequences as described above.

The video image management system and method of the present invention can also be implemented in computer systems and computer controlled systems, using a data processor that executes procedures for carrying of the video frame processing steps discussed above. The present invention can also be implemented as a computer program product (e.g., a CD-ROM or data signal conveyed on a carrier signal) containing image and/or video frame processing procedures suitable for use by a computer system.

Alternate Embodiments

The state machines of the embodiments described above can be replaced by software procedures that are executed by a general purpose (programmable) data processor or a programmable image data processor, especially if speed of operation is not a concern.

Numerous other aspects of the described embodiments may change over time as technology improvements are used to upgrade various parts of the digital camera. For instance, the memory technology used to store image files might change from flash memory to another type of memory, or a camera might respond to voice commands, enabling the use of fewer buttons.

Referring to Fig. 10, the present invention can also be used in a variety of image processing systems other than digital cameras and digital video cameras, including cable television set top

boxes, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. For example, an Internet server 300 can store images and/or video sequences in the wavelet transform compressed data structures of the present invention. Copies of those compressed data structures are transferred to the memory 306 of client computers or other client devices 302, using HTTP or any other suitable protocol via the Internet 304 or other communications network. When appropriate, an image or video sequence is reduced in size so as to fit in the memory available in the client computer or other device (client device). Furthermore, once an image or video sequence has been stored in the memory 306 of a client device, the techniques of the present invention can be used to manage the storage of the image, for instance through gradual reduction of image quality so as to make room for the storage of additional images or video sequences. In the embodiment shown in Fig. 10, the memory 306 of the client computer will have stored therein:

- an operating system 310;
- a browser or other image viewer application 312 for viewing documents and images;
- image files 314;
- image transform procedures 316, such as wavelet or wavelet-like transform procedures for converting a raw image array into wavelet transform coefficients, procedures for compressing and encoding the wavelet transform coefficients, as well as other transform procedures for handling images received in other image formats, such JPEG transform procedures for converting JPEG files into reconstructed image data that is then used as the raw image data by a wavelet or wavelet-like transform procedure;
- an image compression, quality reduction procedure 318 for implementing the image data structure size and quality reduction features of the present invention; and
- image reconstruction procedures 320 for decompressing and reverse transforming image files so as to generate reconstructed image data arrays that are suitable for viewing on the monitor of the client workstation, or for printing or other use.

The client workstation memory 306 will typically include both high speed random access memory and slower non-volatile memory such as a hard disk and/or read-only memory. The client workstation's central processing unit(s) 308 execute operating system procedures and image handling procedures, as well as other applications, thereby performing image processing functions similar to those performed by dedicated circuitry in other embodiments of the present invention.

As indicated above, when the present invention is used in conjunction with, or as part of, a browser application, for management of image storage, some images may be initially received in formats other than "raw" image arrays. For instance, some images may be initially received as JPEG files, or in other proprietary or industry standard formats. To make full use of the capabilities of the present invention, such images are preferably decoded so as to generate reconstructed "raw" image arrays, and then those raw image arrays are wavelet or wavelet-like transformed so as to put the images in a form that enables use of the image quality level management features of the present invention.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic, including data processing circuitry and state machines for storing and processing image data received from the image capture mechanism, the data processing circuitry and state machines including:

image processing circuitry for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression method to the transform image data so as to generate a new image data structure having an associated image quality level selected from the predefined range of image quality levels; the new image data structure being stored in the memory device;

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

2. The image processing apparatus of claim 1, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction circuitry and one or more state machines includes logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

3. The image processing apparatus of claim 1, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

4. Image processing apparatus, for use in conjunction with an image capture mechanism, the image processing apparatus comprising:

a memory device for storing a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

image management logic for storing and processing image data received from the image capture mechanism, including:

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data received from the image capture mechanism, the instructions including:

an initial image processing procedure for applying a predefined transform to image data received from the image capture mechanism to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an

image data structure having an associated image quality level selected from the predefined range of image quality levels;

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

5. The image processing apparatus of claim 4, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

6. The image processing apparatus of claim 4, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at

least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

7. Image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to an array of image data so as to generate transform image data and for applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures stored in the memory device, and forming a reduced size version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure and that has a lower associated image quality level than the image quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by the image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

8. The image processing apparatus of claim 7, further including a communications interface for receiving the image data from another apparatus.

9. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

10. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

11. Image processing apparatus, comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

image management logic, including:

image size reduction circuitry for extracting a subset of the data in a first specified one of the image data structures to form a reduced size image data structure that occupies less space in the memory device than was previously occupied by the first specified

image data structure and that has a lower associated image quality level than the quality level associated with the first specified image data structure; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to any specified one of the image data structures and the reduced size image data structure so as to generate a reconstructed image suitable for display on a display device;

wherein the amount of space occupied by the image data structure in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

12. The image processing apparatus of claim 11, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

13. The image processing apparatus of claim 8, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

14. Image processing apparatus, the image processing apparatus comprising:

a communications interface for receiving an image data structure having an associated image quality level selected from a predefined range of image quality levels that range from a

highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the image data structure and other image data structures representing a set of images;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including instructions for storing and processing image data, the instructions including:

an image size reduction procedure for lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in the first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

15. The image processing apparatus of claim 14, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction instructions include instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

16. The image processing apparatus of claim 14, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is

stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction instructions include instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

17. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a plurality of image data structures,

an image size reduction procedure for accessing image data structures in the memory of the computer system, each of the image data structures containing image transform data, lowering the quality level of a first specified one of the image data structures, including instructions for extracting a subset of the data in a first specified image data structure and forming a lower quality version of the first specified image data structure that occupies less space in the memory device than was previously occupied by the first specified image data structure; and

at least one image reconstruction procedure for successively applying a data decompression procedure and an inverse transform to any specified one of the image data structures stored in the memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

18. The computer program product of claim 17, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes; and

the image size reduction procedure includes instructions for extracting a portion of the first specified image data structure that excludes the image transform data for at least one bit plane and for replacing the first specified image data structure with an image data structure containing the extracted portion.

19. The computer program product of claim 17, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction procedure includes instructions, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

20. The computer program product of claim 17, wherein the image handling procedure includes one or more image processing procedures for applying a predefined transform to raw image data to generate transform image data and for applying a data compression procedure to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels.

21. A method of processing images, comprising:

storing in a memory device a plurality of image data structures that each represent a respective image, each image data structure having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded in the image data structure; the image quality level of each image data structure being a member of predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

reducing the size of a specified one of the image data structures stored in the nonvolatile memory device, including extracting a subset of the data in the specified image data structure and forming a lower quality version of the specified image data structure that

occupies less space in the nonvolatile memory device than was previously occupied by the specified image data structure; and

successively applying a data decompression method and an inverse transform to a specified one of the image data structures stored in the nonvolatile memory device so as to generate a reconstructed image suitable for display on an image viewer;

wherein the amount of space occupied by images stored in the form of image data structures in the nonvolatile memory device can be reduced so as to make room for the storage of additional image data structures in the nonvolatile memory device.

22. The method of claim 21, wherein the method is performed by a digital camera and the method includes applying a predefined transform to image data received from an image capture mechanism in the digital camera to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

23. The method of claim 21, wherein the method includes applying a predefined transform to raw image data to generate transform image data, applying a data compression method to the transform image data so as to generate an image data structure having an associated image quality level selected from the predefined range of image quality levels, and storing the image data structure in the memory device.

24. The method of claim 21, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the size reduction step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure in the nonvolatile memory device with an image data structure containing the extracted portion.

25. The method of claim 21, wherein

each of a subset of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the size reduction step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and replacing the first specified image data structure with an image data structure containing the extracted portion.

26. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels; and

image management logic including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

27. The video image processing apparatus of claim 26, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at

least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

28. The video image processing apparatus of claim 26, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

29. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

30. The video image processing apparatus of claim 29, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

31. The video image processing apparatus of claim 26, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

32. Video image processing apparatus, comprising:

image management logic, including:

image processing circuitry for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated

image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a memory device for storing the set of image data structures;

the image management logic further including:

image size reduction circuitry for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

image reconstruction circuitry for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

33. The video image processing apparatus of claim 32, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the image size reduction circuitry and one or more state machines including logic for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

34. The video image processing apparatus of claim 32, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the image size reduction circuitry and one or more state machines includes logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the

image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

35. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

36. The video image processing apparatus of claim 35, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

37. The video image processing apparatus of claim 32, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

38. Video image processing apparatus, comprising:

a memory device for storing a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

39. The video image processing apparatus of claim 38, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

40. The video image processing apparatus of claim 38, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

41. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

42. The video image processing apparatus of claim 41, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

43. The video image processing apparatus of claim 38, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

44. Video image processing apparatus, comprising:

a memory device for storing image data structures;

a data processor coupled to the memory device;

image management procedures, executable by the data processor, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate a set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels, and for storing the set of image data structures in the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

45. The video image processing apparatus of claim 44, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

46. The video image processing apparatus of claim 44, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

47. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

48. The video image processing apparatus of claim 47, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

49. The video image processing apparatus of claim 44, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

50. A computer program product, for use in conjunction with a computer system having a memory in which image data structures can be stored, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, the computer program mechanism comprising:

an image handling procedure, including instructions for storing in the memory of the computer system a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

a data processor coupled to the memory device;

an image size reduction procedure for extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

at least one image reconstruction procedure for successively applying a data decompression method and an inverse transform to at least a subset of the image data structures so as to generate a reconstructed sequence of video frames suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

51. The computer program product of claim 50, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit

plane and for replacing the image data structure with an image data structure containing the extracted portion.

52. The computer program product of claim 50, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

53. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

54. The computer program product of claim 53, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

55. The computer program product of claim 50, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

56. The computer program product of claim 50, including:

at least one image processing procedure for applying a predefined transform to a sequence of video frames to generate transform image data and for applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory.

57. The computer program product of claim 56, wherein

each image data structure contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the at least one image size reduction procedure includes instructions for extracting a portion of an image data structure that excludes the image transform data for at least one bit plane and for replacing the image data structure with an image data structure containing the extracted portion.

58. The computer program product of claim 56, wherein

the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the at least one image size reduction procedure and one or more state machines include logic, operative when the first specified data structure is a member of the subset of image data structures, for extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer and for replacing the first specified image data structure with an image data structure containing the extracted portion.

59. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

60. The computer program product of claim 59, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

61. The computer program product of claim 56, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

62. A method of processing video images, comprising:

storing in a memory device a set of image data structures representing a sequence of video frames, the set of image data structures having an associated image quality level selected from a predefined range of image quality levels that range from a highest quality level to a lowest quality level and that include at least two distinct quality levels;

extracting a subset of the data in the set of image data structures and forming a lower quality version of the set of image data structures that occupies less space in the memory device than was previously occupied by the set of image data structures; and

successively applying a data decompression method and an inverse transform to the specified set of image data structures so as to generate a reconstructed sequence of video images suitable for display on a display device;

whereby the amount of space occupied by the set of image data structures in the memory device can be reduced so as to make room for the storage of additional image data structures in the memory device.

63. The method of claim 62, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

64. The method of claim 62, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

65. The method of claim 62, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generate transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

66. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

67. The method of claim 66, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

68. The method of claim 62, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

69. The method of claim 62, including applying a predefined transform to a sequence of video images to generate transform image data and applying a data compression method to the transform image data so as to generate the set of image data structures stored in the memory device.

70. The method of claim 69, wherein

each of the image data structures contains image transform data organized on a bit plane basis such that image transform data for at least one bit plane is stored in distinct portions of the image data structure from image transform data for other bit planes;

the extracting step includes extracting a portion of an image data structure that excludes the image transform data for at least one bit plane, and the forming step includes replacing the image data structure with an image data structure containing the extracted portion.

71. The method of claim 69, wherein

the of the image data structures contains image transform data organized on a transform layer basis such that image transform data for at least one transform layer is stored in distinct portions of the image data structure from image transform data for other transform layers; and

the extracting step includes extracting a portion of the first specified image data structure that excludes the image transform data for at least one transform layer, and the forming step includes replacing the first specified image data structure with an image data structure containing the extracted portion.

72. The method of claim 69, after performing the extracting and forming steps, applying the predefined transform to a sequence of additional video images to generatc transform image data and applying the data compression method to the transform image data so as to generate an additional set of image data structures having an associated image quality, and storing the additional set of image data structures in the memory device.

73. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to at least one video frame in each said sub-sequence of N frames.

74. The method of claim 73, wherein the wavelet-like transform is applied to at least one difference frame for each said sub-sequence of N frames, the difference frame representing differences between one frame and a next frame in said sub-sequence of N frames.

75. The method of claim 69, wherein the video frames are divided into sub-sequences of N frames, where N is an integer greater than three, and the predefined transform applied to the sequence of video images is a wavelet-like transform that is applied to separately and in time order to data at each x,y position in the video frames.

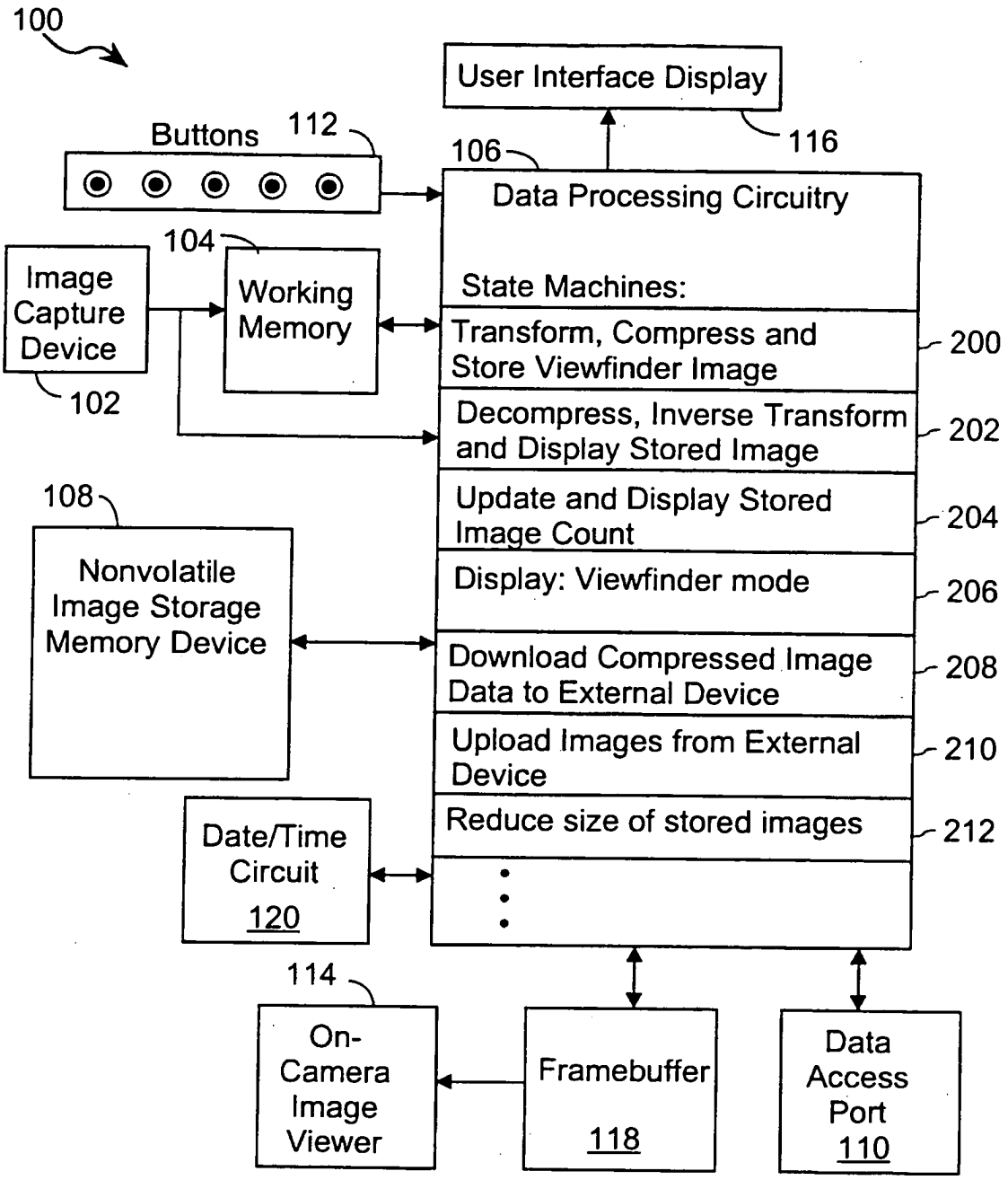

100
User Interface Display
116
Buttons
112
106
Data Processing Circuitry
State Machines:
104
Image Capture Device
Working Memory
102
Transform, Compress and Store Viewfinder Image
200
Decompress, Inverse Transform and Display Stored Image
202
Update and Display Stored Image Count
204
108
Nonvolatile Image Storage Memory Device
Display: Viewfinder mode
206
Download Compressed Image Data to External Device
208
Upload Images from External Device
210
Date/Time Circuit
120
Reduce size of stored images
212
114
On-Camera Image Viewer
Framebuffer
118
Data Access Port
110

FIG. 1

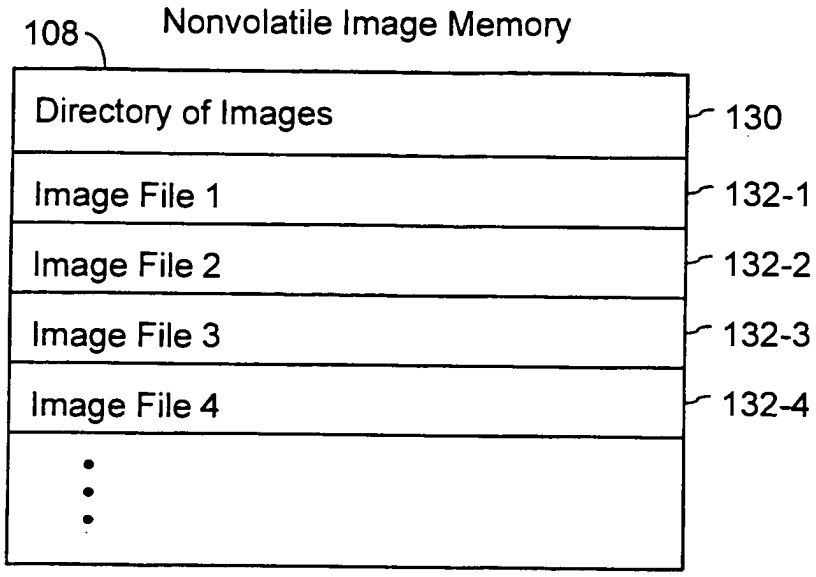

108
Nonvolatile Image Memory
Directory of Images
130
Image File 1
132-1
Image File 2
132-2
Image File 3
132-3
Image File 4
132-4

FIG. 2

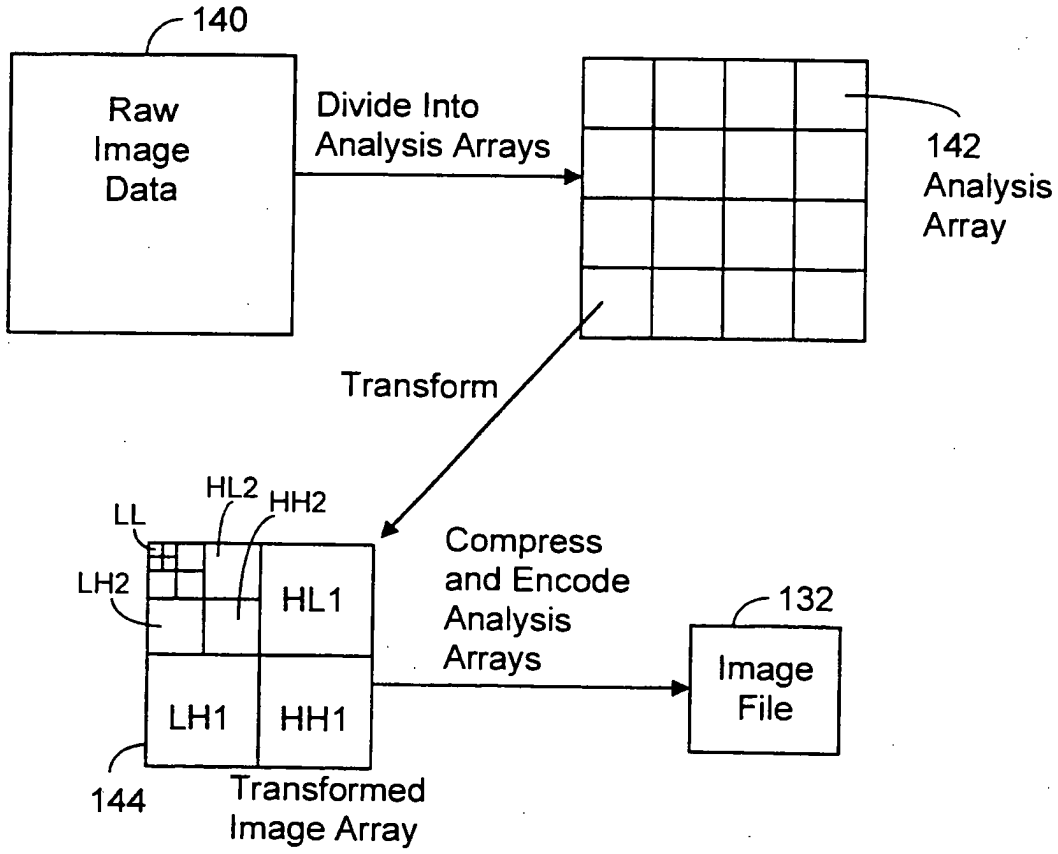

140
Raw Image Data
Divide Into Analysis Arrays
142 Analysis Array
Transform
LL
HL2
HH2
LH2
HL1
LH1
HH1
144
Transformed Image Array
Compress and Encode Analysis Arrays
132
Image File

FIG. 3

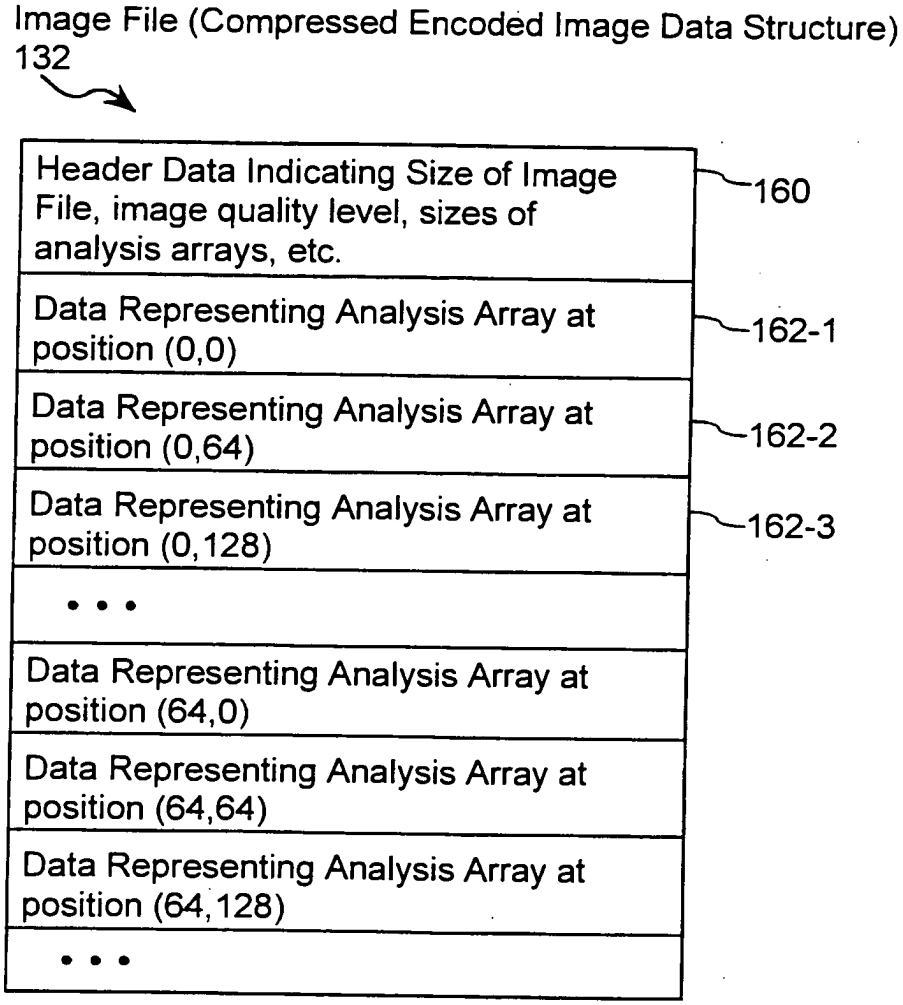


Image File (Compressed Encoded Image Data Structure)
132
Header Data Indicating Size of Image File, image quality level, sizes of analysis arrays, etc.
160
Data Representing Analysis Array at position (0,0)
162-1
Data Representing Analysis Array at position (0,64)
162-2
Data Representing Analysis Array at position (0,128)
162-3
• • •
Data Representing Analysis Array at position (64,0)
Data Representing Analysis Array at position (64,64)
Data Representing Analysis Array at position (64,128)
• • •

FIG. 4A

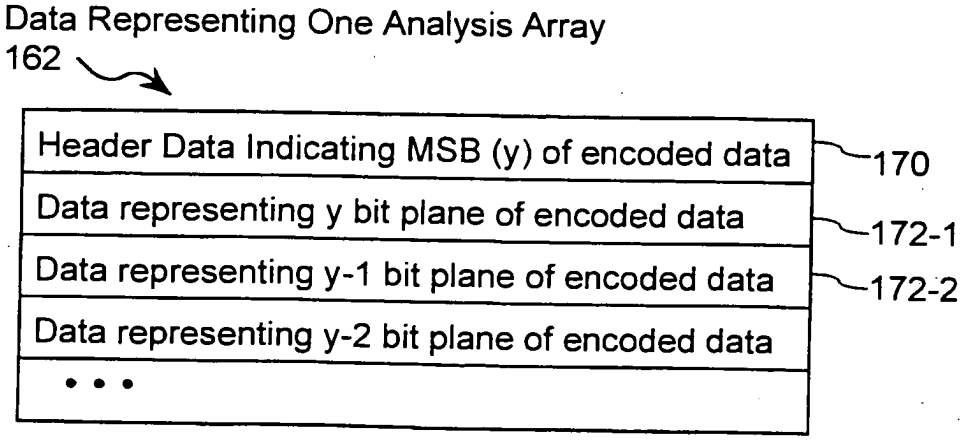

Data Representing One Analysis Array
162
Header Data Indicating MSB (y) of encoded data
170
Data representing y bit plane of encoded data
172-1
Data representing y-1 bit plane of encoded data
172-2
Data representing y-2 bit plane of encoded data
• • •

FIG. 4B

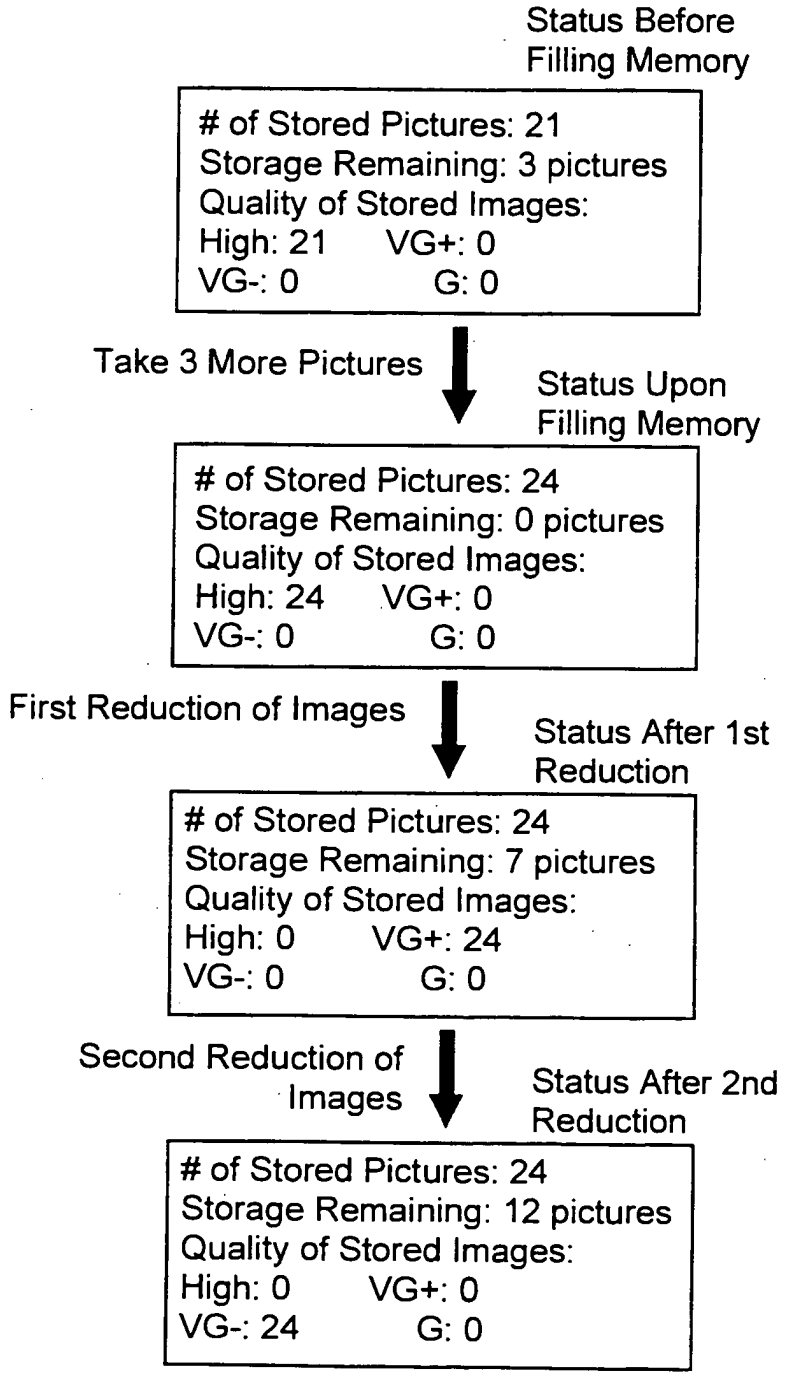

Status Before Filling Memory
# of Stored Pictures: 21
Storage Remaining: 3 pictures
Quality of Stored Images:
High: 21 VG+: 0
VG-: 0 G: 0
Take 3 More Pictures
Status Upon Filling Memory
# of Stored Pictures: 24
Storage Remaining: 0 pictures
Quality of Stored Images:
High: 24 VG+: 0
VG-: 0 G: 0
First Reduction of Images
Status After 1st Reduction
# of Stored Pictures: 24
Storage Remaining: 7 pictures
Quality of Stored Images:
High: 0 VG+: 24
VG-: 0 G: 0
Second Reduction of Images
Status After 2nd Reduction
# of Stored Pictures: 24
Storage Remaining: 12 pictures
Quality of Stored Images:
High: 0 VG+: 0
VG-: 24 G: 0

FIG. 5

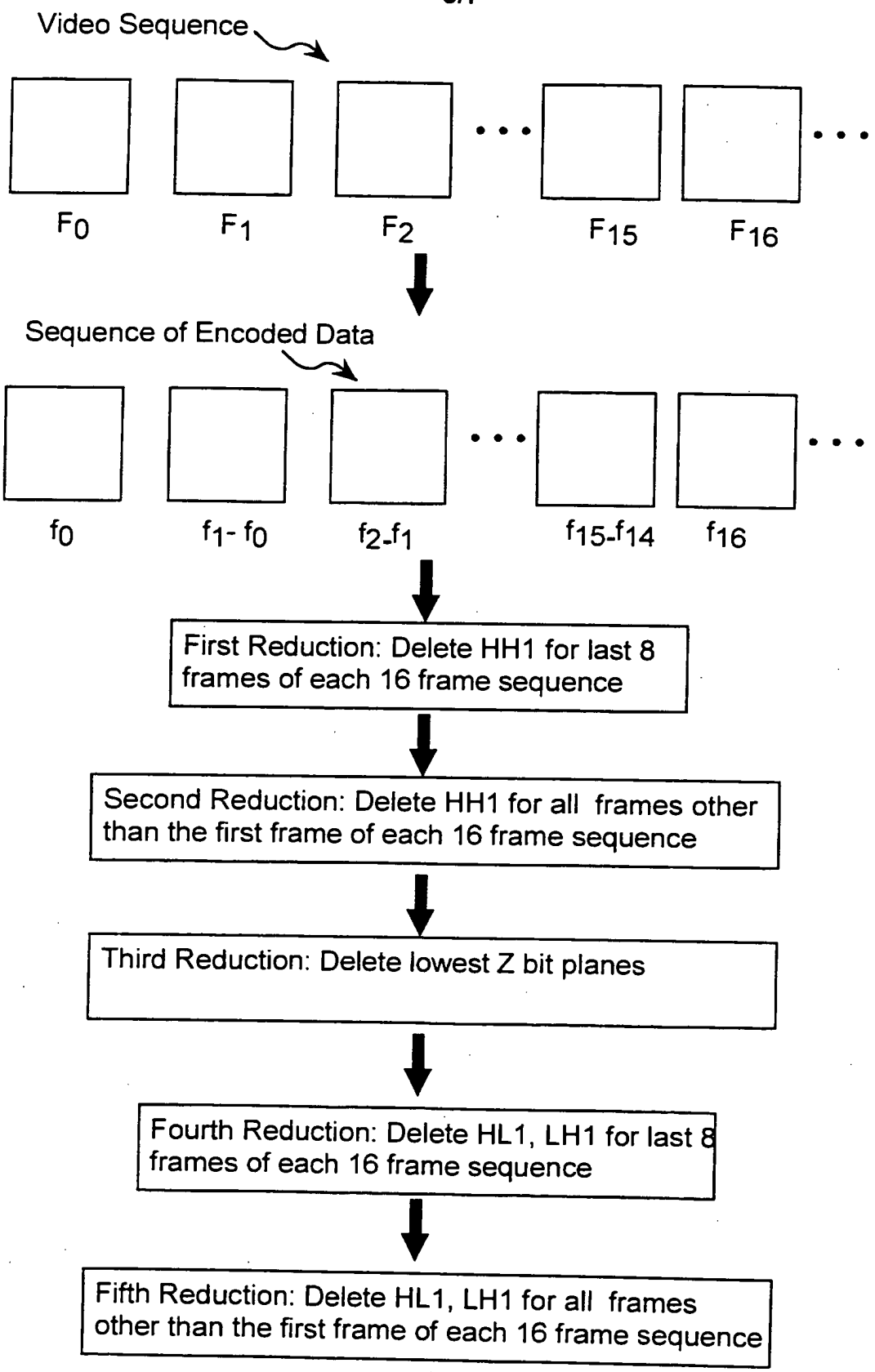

Video Sequence
F0
F1
F2
F15
F16
Sequence of Encoded Data
f0
f1- f0
f2-f1
f15-f14
f16
First Reduction: Delete HH1 for last 8 frames of each 16 frame sequence
Second Reduction: Delete HH1 for all frames other than the first frame of each 16 frame sequence
Third Reduction: Delete lowest Z bit planes
Fourth Reduction: Delete HL1, LH1 for last 8 frames of each 16 frame sequence
Fifth Reduction: Delete HL1, LH1 for all frames other than the first frame of each 16 frame sequence

FIG. 6

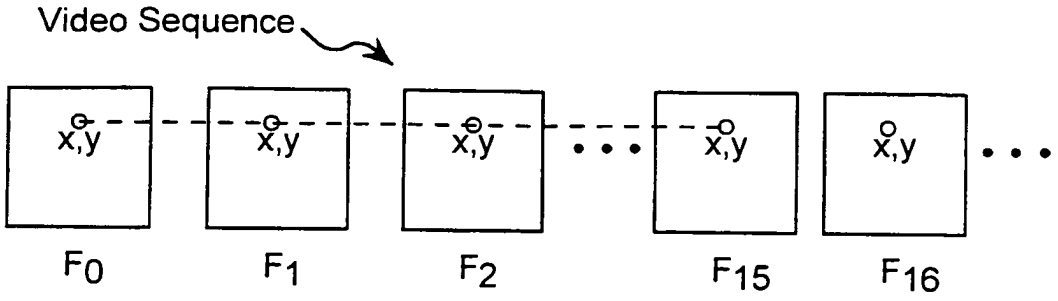


FIG. 7

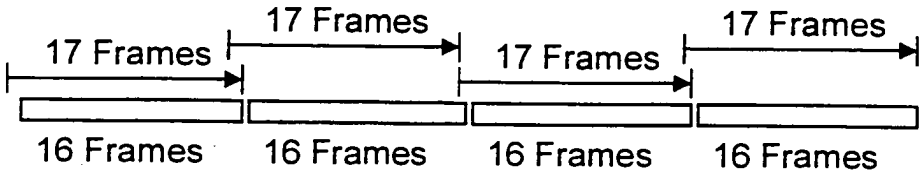


FIG. 8

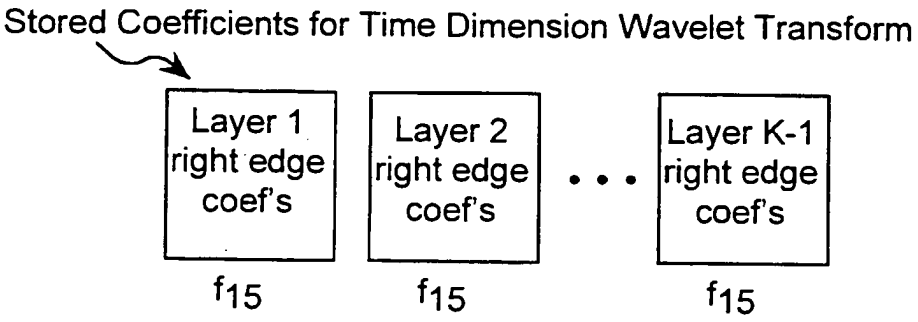


FIG. 9

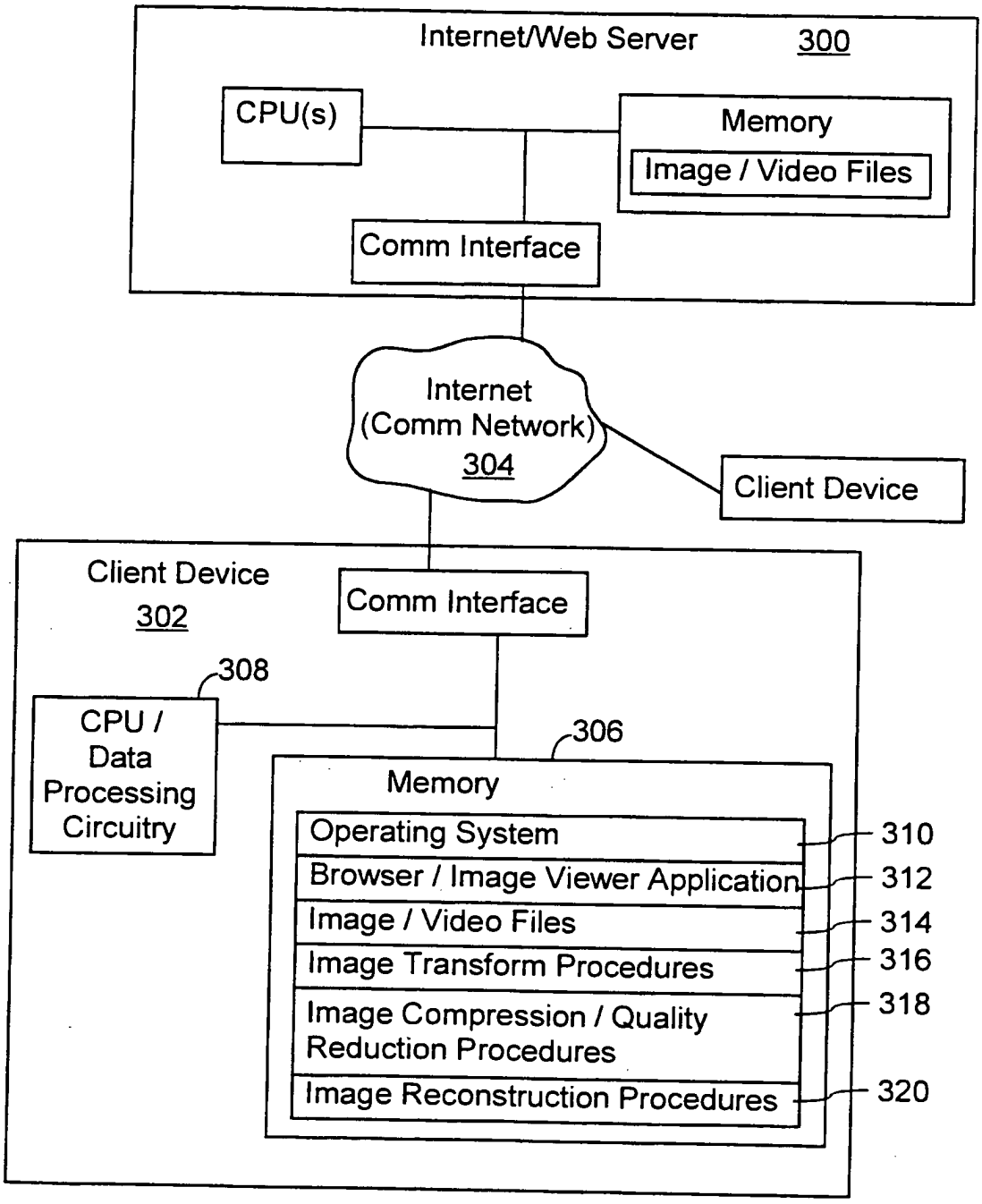

Internet/Web Server
300
CPU(s)
Memory
Image / Video Files
Comm Interface
Internet
(Comm Network)
304
Client Device
Client Device
302
Comm Interface
308
CPU /
Data
Processing
Circuitry
306
Memory
Operating System
Browser / Image Viewer Application
Image / Video Files
Image Transform Procedures
Image Compression / Quality
Reduction Procedures
Image Reconstruction Procedures
310
312
314
316
318
320

FIG. 10

INTERNATIONAL SEARCH REPORT

Interna ..al application No.
PCT/US00/30825

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06K 9/36, 9/46
US CL : 382/232

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : Please See Continuation Sheet

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,867,602 A (ZANDI ET AL.) 02 February 1999 (02.02.99).	1-75
A	US 5,881,176 A (KEITH ET AL.) 09 March 1999 (09.03.1999).	1-75
A	US 5,966,465 A (KEITH ET AL.) 12 October, 1999 (12.10.1999).	1-75

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
04 January 2001 (04.01.2001)	30 MAR 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Jose L. Couso Telephone No. (703)305-8576

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

Inte: ional application No.
PCT/US00/30825

Continuation of B. FIELDS SEARCHED Item 1: 382/232, 233, 234, 240, 244, 246, 247, 248, 260, 263, 264, 276; 341/51, 63, 65, 67, 107; 364/724.011, 724.04, 724.05, 724.13, 724.14, 725.01, 725.02.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 067 556**
B1

(12) # EUROPEAN PATENT SPECIFICATION

(45) Date of publication of patent specification: **18.04.90**

(51) Int. Cl.[5]: **G 06 F 9/46**

(21) Application number: **82302596.0**

(22) Date of filing: **21.05.82**

(60) **Divisional applications 88200917, 88200916, 88200921 filed on 09.05.88.**

(54) **Digital data processing system.**

(30) Priority: **22.05.81 US 266413**
22.05.81 US 266539 **22.05.81 US 266414**
22.05.81 US 266521 **22.05.81 US 266532**
22.05.81 US 266415 **22.05.81 US 266403**
22.05.81 US 266409 **22.05.81 US 266408**
22.05.81 US 266424 **22.05.81 US 266401**
22.05.81 US 266421 **22.05.81 US 266524**
22.05.81 US 266404

(43) Date of publication of application:
22.12.82 Bulletin 82/51

(45) Publication of the grant of the patent:
18.04.90 Bulletin 90/16

(84) Designated Contracting States:
AT BE CH DE FR GB IT LI LU NL SE

(56) References cited:

Hasselmeier, Spruth: "Rechnerstrukturen", 1974, pp 75-103

Klar, Wichmann: "Mikroprogrammierung", June 1975, pp 159-163, 176-179, 185-187, 195-205, 214-215

(73) Proprietor: **DATA GENERAL CORPORATION**
Route 9
Westboro Massachusetts 01581 (US)

(72) Inventor: **Ahlstrom, John K.**
1309 San Domar
Mountain View California 94043 (US)
Inventor: **Bachman, Brett L.**
214 W. Canton Street Suite 4
Boston Massachusetts 02116 (US)
Inventor: **Belgard, Richard A.**
21250 Glenmont Drive
Saratoga California 95070 (US)
Inventor: **Bernstein, David H.**
41 Bay Colony Drive
Ashland Massachusetts 01721 (US)
Inventor: **Bratt, Richard Glenn**
9 Brook Trail Road
Wayland Massachusetts 01778 (US)
Inventor: **Clancy, Gerald F.**
13069 Jaccaranda Center
Saratoga California 95070 (US)
Inventor: **Farber, David A.**
1700 Lakewood Avenue
Durham North Carolina 27707 (US)
Inventor: **Gavrin, Edward S.**
Beaver Pond Road RFD 4
Lincoln Massachusetts 01773 (US)

Courier Press, Leamington Spa, England.

EP 0 067 556 B1

(56) References cited:

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 22, no. 3, august 1979, pages 1286-1289, New York, US D.B. LOMET: "Regions for controlling the propagation of addressability in capability systems"

IBM TECHNICAL DISCLOSURE BULLETIN, vol. 15, no. 9, February 1973, pages 2721-2722 W.L. DUNNE: "Common Compiler/Interpreter for a programming language"

ADVANCES IN COMPUTERS, vol. 14, 1976, Academic Press, New York, US pages 231-272 D.K. HSIAO et al.:"Information Secure Systems"

IEEE,DIGEST OF PAPERS FROM COMPCON FALL MEETING, September 10-12, 1974, Washington, Micros and Minis, pages 15-17 Long Beach, US C.J. NEUHAUSER et al.: "Description of an emulation laboratory"

7th Annual Symposium on COMPUTER ARCHITECTURE, May 6-8, 1980, Conference Proceedings, pages 245-252 New York, US V. BERSTIS: "Security and protection of data in the IBM System/38

VERY LARGE DATA BASES, vol.9, no. 2C, October 1977, Third International Conference Proceedings, pages 507-514 Tokyo, JP D. DOWNS et al.: "A kernel design for a secure data base management system"

Inventor: Gruner, Ronald Hans
112 Dublin Wood Drive
Cary North Carolina 27514 (US)
Inventor: Houseman, David L.
1213 Selwyn Lane
Cary North Carolina 27511 (US)
Inventor: Jones, Thomas M. Jones
300 Reade Road
Chapel Hill North Carolina 27514 (US)
Inventor: Katz, Lawrence H.
10943 S. Forest Ridge Road
Oregon City Oregon 97045 (US)
Inventor: Mundie, Craig James
136 Castlewood Drive
Cary North Carolina (US)
Inventor: Pilat, John F.
1308 Ravenhurst Drive
Raleigh North Carolina 27609 (US)
Inventor: Richmond, Michael S.
Fearrington Post Box 51
Pittsboro North Carolina 27312 (US)
Inventor: Schleimer, Stephen I.
1208 Ellen Place
Chapel Hill North Carolina 27514 (US)
Inventor: Wallach, Steven J.
12436 Green Meadow Lane
Saratoga California 95070 (US)
Inventor: Wallach, Walter A., Jr.
1336 Medfield Road
Raleigh North Carolina 27607 (US)
Inventor: Wells, Douglas M.
106 Robin Road
Chapel Hill North Carolina 27514 (US)

(74) Representative: Pears, David Ashley et al
REDDIE & GROSE 16 Theobalds Road
London WC1X 8PL (GB)

Description

The present invention relates to a digital data processing system and, more particularly, to a multiprocessor digital data processing system suitable for use in a data processing network and having a simplified, flexible user interface and flexible, multileveled internal mechanism.

A general trend in the development of data processing systems has been towards systems suitable for use in interconnected data processing networks. Another trend has been towards data processing systems wherein the internal structure of the system is flexible, protected from users, and effectively invisible to the user and wherein the user is presented with a flexible and simplified interface to the system.

Certain problems and shortcomings affecting the realization of such a data processing system have appeared repeatedly in the prior art and must be overcome to create a data processing system having the above attributes. These prior art problems and limitations include the following topics.

First, the data processing systems of the prior art have not provided a system wide addressing system suitable for use in common by a large number of data processing systems interconnected into a network. Addressing systems of the prior art have not provided sufficiently large address spaces and have not allowed information to be permanently and uniquely identified. Prior addressing systems have not made provisions for information to be located and identified as to type or format, and have not provided sufficient granularity. In addition, prior addressing systems have reflected the physical structure of particular data processing systems. That is, the addressing systems have been dependent upon whether a particular computer was, for example, an 8, 16, 32, 64 or 128 bit machine. Since prior data processing systems have incorporated addressing mechanisms wherein the actual physical structure of the processing system is apparent to the user, the operations a user could perform have been limited by the addressing mechanisms. In addition, prior processor systems have operated as fixed word length machines, further limiting user operations.

Prior data processing systems have not provided effective protection mechanisms preventing one user from effecting another user's data and programs without permission. Such protection mechanisms have not allowed unique, positive identification of users requesting access to information, or of information, nor have such mechanisms been sufficiently flexible in operation. In addition, access rights have pertained to the users rather than to the information, so that control of access rights has been difficult. Finally, prior art protection mechanisms have allowed the use of "Trojan Horse arguments". That is, users not having access rights to certain information have been able to gain access to that information through another user or procedure having such access rights.

Yet another problem of the prior art is that of providing a simple and flexible user's interface to a data processing system. The character of user's interface to a data processing system is determined, in part, by the means by which a user refers to and identifies operands and procedures of the user's programs and by the instruction structure of the system. Operands and procedures are customarily referred to and identified by some form of logical address having points of reference, and validity, only within a user's program. These addresses must be translated into logical and physical addresses within a data processing system each time a program is executed, and must then be frequently retranslated or generated during execution of a program. In addition, a user must provide specific instructions as to data format and handling. As such reference to operands or procedures typically comprise a major portion of the instruction stream of the user's program and requires numerous machine translations and operations to implement. A user's interface to a conventional system is thereby complicated, and the speed of execution of programs reduced, because of the complexity of the program references to operands and procedures.

A data processing system's instruction structure includes both the instructions for controlling system operations and the means by which these instructions are executed. Conventional data processing systems are designed to efficiently execute instructions in one or two user languages, for example, FORTRAN or COBOL. Programs written in any other language are not efficiently executable. In addition, a user is often faced with difficult programming problems when using any high level language other than the particular one or two languages that a particular conventional system is designed to utilize.

Yet another problem in conventional data processing systems is that of protecting the system's internal mechanisms, for example, stack mechanisms and internal control mechanisms, from accidental or malicious interference by a user.

Finally, the internal structure and operation of prior art data processing systems have not been flexible, or adaptive, in structure and operation. That is, the internal structure and operation of prior systems have not allowed the systems to be easily modified or adapted to meet particular data processing requirements. Such modifications may include changes in internal memory capacity, such as the addition or deletion of special purpose subsystems, for example, floating point or array processors. In addition, such modifications have significantly effected the users interface with the system. Ideally, the actual physical structure and operation of the data processing system should not be apparent at the user interface.

It has already been proposed (IBM Technical Disclosure Bulletin Vol. 22 No. 3 Aug. 1979 pp 1286—1289) to maintain such a large address space that every object which is ever created can have a unique identifier. This requires a very large identifier field, e.g. 40 to 50 bits.

The object of the present invention is to implement such a concept so that it may be applied across many computers geographically distributed and without requiring all computers to use the same

programming language.

The system according to the invention is defined in the appended claims.

It is known in virtual address machines to use name tables to provide logical addresses for translation to physical addresses (Klar, Wichmann, "Mikroprogrammierung" June 1975, especially pp. 159—163, 176—179, 185—187, 195—205, 214—215) and this reference also discusses the emulation in software of different target machines.

More specifically, the embodiment of the invention described in detail below provides a data processing system suitable for use in interconnected data processing networks, which internal structure is flexible, protected from users, effectively invisible to users, and provides a flexible and simplified interface to users. The data processing system provides an addressing mechanism allowing permanent and unique identification of all information generated for use in or by operation of the system, and an extremely large address space which is accessible to and common to all such data processing systems. The addressing mechanism provides addresses which are independent of the physical configuration of the system and allow information to be completely identified, with a single address, to the bit granular level and with regard to information type or format. The present invention further provides a protection mechanism wherein variable access rights are associated with individual bodies of information. Information, and users requesting access to information, are uniquely identified through the system addressing mechanism. The protection mechanism also prevents use of Trojan Horse arguments. And, the present invention provides an instruction structure wherein high level user language instructions are transformed into dialect coded, uniform, intermediate level instructions to provide equal facility of execution for a plurality of user languages. Another feature is the provision of an operand reference mechanism wherein operands are referred to in user's programs by uniform format names which are transformed, by an internal mechanism transparent to the user, into addresses. The present invention additionally provides multilevel control and stack mechanisms protecting the system's internal mechanism from interference by users. Yet another feature is a data processing system having a flexible internal structure capable of performing multiple, concurrent operations and comprised of a plurality of separate, independent processors. Each such independent processor has a separate microinstruction control and at least one separate and independent port to a central communications and memory node. The communications and memory node is also an independent processor having separate and independent microinstruction control. The memory processor is internally comprised of a plurality of independently operating, microinstruction controlled processors capable of performing multiple, concurrent memory and communications operations. The present invention also provides further data processing system structural and operational features for implementing the above features.

It is thus advantageous to incorporate the present invention into a data processing system because the present invention provides addressing mechanisms suitable for use in large interconnected data processing networks. Additionally, the present invention is advantageous in that it provides an information protection mechanism suitable for use in large, interconnected data processing networks. The present invention is further advantageous in that it provides a simplified, flexible, and more efficient interface to a data processing system. The present invention is yet further advantageous in that it provides a data processing system which is equally efficient with any user level language by providing a mechanism for referring to operands in user programs by uniform format names and instruction structure incorporating dialect coded, uniform format intermediate level instructions. Additionally, the present invention protects data processing system internal mechanisms from user interference by providing multilevel control and stack mechanisms. The present invention is yet further advantageous in providing a flexible internal system structure capable of performing multiple, concurrent operations, comprising a plurality of separate, independent processors, each having a separate microinstruction control and at least one separate and independent port to a central, independent communications and memory processor comprised of a plurality of independent processors capable of performing multiple, concurrent memory and communications operations.

Other advantages and features of the present invention will be understood by those of ordinary skill in the art, after referring to the following detailed description of the preferred embodiments and drawings wherein.

BRIEF DESCRIPTION OF DRAWINGS

Fig. 1 is a partial block diagram of a computer system incorporating the present invention;

Fig. 2 is a diagram illustrating computer system addressing structure of the present invention;

Fig. 3 is a diagram illustrating the computer system instruction stream of the present invention;

Fig. 4 is a diagram illustrating the control structure of a conventional computer system;

Fig. 4A is a diagram illustrating the control structure of a computer system incorporating the present invention;

Fig. 5 — Fig. A1 inclusive are diagrams all relating to the present invention;

Fig. 5 is a diagram illustrating a stack mechanism;

Fig. 6 is a diagram illustrating procedures, procedure objects, processes, and virtual processors;

Fig. 7 is a diagram illustrating operating levels and mechanisms of the present computer;

Fig. 8 is a diagram illustrating a physical implementation of processes and virtual processors;

Fig. 9 is a diagram illustrating a process and process stack objects;
Fig. 10 is a diagram illustrating operation of macrostacks and secure stacks;
Fig. 11 is a diagram illustrating detailed structure of a stack;
Fig. 12 is a diagram illustrating a physical descriptor;
Fig. 13 is a diagram illustrating the relationship between logical pages and frames in a memory storage space;
Fig. 14 is a diagram illustrating access control to objects;
Fig. 15 is a diagram illustrating virtual processors and virtual processor swapping;
Fig. 16 is a partial block diagram of an I/0 system of the present computer system;
Fig. 17 is a diagram illustrating operation of a ring grant generator;
Fig. 18 is a partial block diagram of a memory system;
Fig. 19 is a partial block diagram of a fetch unit of the present computer system;
Fig. 20 is a partial block diagram of an execute unit of the present computer system;
Fig. 101 is a more detailed partial block diagram of the present computer system;
Fig. 102 is a diagram illustrating certain information structures and mechanisms of the present computer system;
Fig. 103 is a diagram illustrating process structures;
Fig. 104 is a diagram illustrating a macrostack structure;
Fig. 105 is a diagram illustrating a secure stack structure;
Figs. 106 A, B, and C are diagrams illustrating the addressing structure of the present computer system;
Fig. 107 is a diagram illustrating addressing mechanisms of the present computer system;
Fig. 108 is a diagram illustrating a name table entry;
Fig. 109 is a diagram illustrating protection mechanisms of the present computer system;
Fig. 110 is a diagram illustrating instruction and microinstruction mechanism of the present computer system;
Fig. 201 is a detailed block diagram of a memory system;
Fig. 202 is a detailed block diagram of a fetch unit;
Fig. 203 is a detailed block diagram of an execute unit;
Fig. 204 is a detailed block diagram of an I/O system;
Fig. 205 is a partial block diagram of a diagnostic processor system;
Fig. 206 is a diagram illustrating assembly of Figs. 201—205 to form a detailed block diagram of the present computer system;
Fig. 207 is a detailed block diagram of a memory interface controller;
Fig. 209 is a diagram of a memory to I/O system port interface;
Fig. 210 is a diagram of a memory operand port interface;
Fig. 211 is a diagram of a memory instruction port interface;
Fig. 230 is a detailed block diagram of memory field interface unit logic;
Fig. 231 is a diagram illustrating memory format manipulation operations;
Fig. 238 is a detailed block diagram of fetch unit offset multiplexer;
Fig. 239 is a detailed block diagram of fetch unit bias logic;
Fig. 240 is a detailed block diagram of a generalized four way, set associative cache representing name cache, protection cache, and address translation unit;
Fig. 241 is a detailed block diagram of portions of computer system instruction and microinstruction control logic;
Fig. 242 is a detailed block diagram of portions of computer system microinstruction control logic;
Fig. 243 is a detailed block diagram of further portions of computer system microinstruction control logic;
Fig. 244 is a diagram illustrating computer system states of operation;
Fig. 245 is a diagram illustrating computer system states of operation for a trace trap request;
Fig. 246 is a diagram illustrating computer system states of operation for a memory repeat interrupt;
Fig. 247 is a diagram illustrating priority level and masking of computer system events;
Fig. 248 is a detailed block diagram of event logic.
Fig. 249 is a detailed block diagram of microinstruction control store logic;
Fig. 251 is a diagram illustrating a return control word stack word;
Fig. 252 is a diagram illustrating machine control words;
Fig. 253 is a detailed block diagram of a register address generator;
Fig. 254 is a block diagram of interval and egg timers;
Fig. 255 is a detailed block diagram of execute unit control logic;
Fig. 257 is a detailed block diagram of execute unit multiplier data paths and memory;
Fig. 260 is a diagram illustrating operation of an execute unit command queue load and interface to a fetch unit;
Fig. 261 is a diagram illustrating operation of an execute unit operand buffer load and interface to a fetch unit;
Fig. 262 is a diagram illustrating operation of an execute unit storeback or transfer of results and

interface to a fetch unit;

Fig. 263 is a diagram illustrating operation of an execute unit check test condition and interface to a fetch unit;

Fig. 264 is a diagram illustrating operation of an execute unit exception test and interface to a fetch unit;

Fig. 265 is a block diagram of an execute unit arithmetic operation stack mechanism;

Fig. 266 is a diagram illustrating execute unit and fetch unit interrupt handshaking and interface;

Fig. 267 is a diagram illustrating execute unit and fetch unit interface and operation for nested interrupts;

Fig. 268 is a diagram illustrating execute unit and fetch unit interface and operation for loading an execute unit control store;

Fig. 269 is a detailed block diagram and illustration of operation of an I/O system ring grant generator;

Fig. 270 is a detailed block diagram of a fetch unit micromachine of the present computer system;

Fig. 271 is a diagram illustrating a logical descriptor;

Fig. 272 is a diagram illustrating use of fetch unit stack registers;

Fig. 273 is a diagram ilustrating structures controlling event invocations;

Fig. 301 is a diagram illustrating pointer formats;

Fig. 302 is a diagram illustrating an associated address table;

Fig. 303 is a diagram illustrating a namespace overview of a procedure object;

Fig. 304 is a diagram illustrating name table entries;

Fig. 305 is a diagram illustrating an example of name resolution;

Fig. 306 is a diagram illustrating name cache entries;

Fig. 307 is a diagram illustrating translation of S-interpreter universal identifiers to dialect numbers;

Fig. 401 is a diagram illustrating operating systems and system resources;

Fig. 402 is a diagram illustrating multiprocess operating systems;

Fig. 403 is a diagram illustrating an extended operating system and a kernel operating system;

Fig. 404 is a diagram illustrating an EOS view of objects;

Fig. 405 is a diagram illustrating pathnames to universal identifier translation;

Fig. 406 is a diagram illustrating universal identifier detail;

Fig. 407 is a diagram illustrating address translation with an address translation unit, a memory hash table, and a memory;

Fig. 408 is a diagram illustrating hashing in an active subject table;

Fig. 409 is a diagram illustrating logical allocation units and objects;

Fig. 410 is a diagram illustrating an active logical allocation unit table and active allocation units;

Fig. 411 is a diagram illustrating a conceptual logical allocation unit directory structure;

Fig. 412 is a diagram illustrating detail of a logical allocation unit directory entry;

Fig. 413 is a diagram illustrating universal identifiers and active object numbers;

Fig. 416 is a diagram illustrating subject templates, primitive access control list entries, and extended access control list entries;

Fig. 421 is a diagram illustrating an active primitive access matrix and an active primitive access matrix entry;

Fig. 422 is a diagram illustrating primitive data access checking;

Fig. 448 is a diagram illustrating event counters and await entries;

Fig. 449 is a diagram illustrating an await table overview;

Fig. 453 is a diagram illustrating an overview of a virtual processor;

Fig. 454 is a diagram illustrating virtual processor synchronization;

Fig. 467 is a diagram illustrating an overview of a macrostack object;

Fig. 468 is a diagram illustrating details of a macrostack object base;

Fig. 469 is a diagram illustrating details of a macrostack frame;

Fig. 470 is a diagram illustrating an overview of a secure stack;

Fig. 471 is a diagram illustrating details of a secure stack frame; and,

Fig. 472 is a diagram illustrating an overview of procedure object.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description presents the structure and operation of a computer system incorporating a presently preferred embodiment of the present invention. As indicated in the following Table of Contents, certain features of computer system structure and operation will first be described in an Introductory Overview. Next, these and other features will be described in further detail in a more detailed Introduction to the detailed descriptions of the computer system. Following the Introduction, the structure and operation of the computer system will be described in detail. The detailed descriptions will present descriptions of the structure and operation of each of the major subsystems, or elements, of the computer system, of the interfaces between these major subsystems, and of overall computer system operation. Next, certain features of the operation of the individual subsystems will be presented in further detail.

Certain conventions are used throughout the following descriptions to enhance clarity of presentation. First, and with exception of the Introductory Overview, each figure referred to in the following descriptions

will be referred to by a three digit number. The most significant digit represents the number of the chapter in the following descriptions in which a particular figure is first referred to. The two least significant digits represent the sequential number of appearance of a figure in a particular chapter. For example, Figure 319 would be the nineteenth figure appearing in the third chapter. Figures appearing in the Introductory Overview are referred to by a one or two digit number representing the order in which they are referred to in the Introductory Overview. It should be noted that certain figure numbers, for example, Figure 208, do not appear in the following figures and descriptions; the subject matter of these figures has been incorporated into other figures and these figures deleted, during drafting of the following descriptions, to enhance clarity of presentation.

Second, reference numerals comprise a two digit number (00—99) preceded by the number of the figure in which the corresponding elements first appear. For example, reference numerals 31901 to 31999 would refer to elements 1 through 99 appearing in Fig. 319.

Finally, interconnections between related circuitry is represented in two ways. First, to enhance clarity of presentation, interconnections between circuitry may be represented by common signal names or references, rather than by drawn representations of wires or buses. Second, where related circuitry is shown in two or more figures, the figures may share a common figure number and will be distinguished by a letter designation, for example, Figs. 319, 319A, and 319B. Common electrical points between such circuitry may be indicated by a bracket enclosing a lead to such a point and a designation of the form "A—b". "A" indicates other figures having the same common point for example, 319A, and "b" designates the particular common electrical point. In cases of related circuitry shown in this manner in two or more figures, reference numerals to elements will be assigned in sequence through the group of figures; the figure number portion of such reference numerals will be that of the first figure of the group of figures.

INTRODUCTORY OVERVIEW

A. Hardware Overview (Fig. 1)
B. Individual Operating Features (Figs. 2, 3, 4, 5, 6)
    1. Addressing (Fig. 2)
    2. S-Language Instructions and Namespace Addressing (Fig. 3)
    3. Architectural Base Pointer Addressing
    4. Stack Mechanisms (Figs. 4-5)
C. Procedure Processes and Virtual Processors (Fig. 6)
D. CS 101 Overall Structure and Operation (Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15)
    1. Introduction (Fig. 7)
    2. Compilers 702 (Fig. 7)
    3. Binder 703 (Fig. 7)
    4. EOS 704 (Fig. 7)
    5. KOS and Architectural Interface 708 (Fig. 7)
    6. Processes 610 and Virtual Processors 612 (Fig. 8)
    7. Processes 610 and Stacks (Fig. 9)
    8. Processes 610 and Calls (Figs. 10, 11)
    9. Memory References and the Virtual Memory Management System (Fig. 12, 13)
    10. Access Control (Fig. 14)
    11. Virtual Processors and Virtual Processor Swapping (Fig. 15)
E. CS 101 Structural Implementation (Figs. 16, 17, 18, 19, 20)
    1. (IOS) 116 (Figs. 16, 17)
    2. Memory (MEM) 112 (Fig. 18)
    3. Fetch Unit (FU) 120 (Fig. 19)
    4. Execute Unit (EU) 122 (Fig. 20)

1. Introduction (Figs. 101—110)
A. General Structure and Operation (Fig. 101)
    a. General Structure
    b. General Operation
    c. Definition of Certain Terms
    d. Multi-program Operation
    e. Multi-Language Operation
    f. Addressing Structure
    g. Protection Mechanism
B. Computer System 10110 Information Structure and Mechanisms (Figs. 102, 103, 104, 105)
    a. Introduction (Fig. 102)
    b. Process Structures 10210 (Figs. 103, 104, 105)
        1. Procedure Objects (Fig. 103)
        2. Stack Mechanisms (Figs. 104, 105)
        3. FURSM 10214 (Fig. 103)
C. Virtual Processor State Blocks and Virtual Process Creation (Fig. 102)

D. Addressing Structures 10220 (Figs. 103, 106, 107, 108)
1. Objects, UID's, AON's, Names, and Physical Addresses (Fig. 106)
2. Addressing Mechanisms 10220 (Fig. 107)
3. Name Resolution (Figs. 103, 108)
4. Evaluation of AON Addresses to Physical Addresses (Fig. 107)
E. CS 10110 Protection Mechanisms (Fig. 109)
F. CS 10110 Micro-Instruction Mechanisms (Fig. 110)
G. Summary of Certain CS 10110 Features and Alternate Embodiments.

2. Detailed Description of CS 10110 Major Subsystems Figs. 201—206, 207—274
A. MEM 10110 (Figs. 201, 206, 207-237)
a. Terminology
b. MEM 10112 physical Structure (Fig. 201)
c. MEM 10112 General Operation
d. MEM 10112 Port Structure
1. IO Port Characteristics
2. JO Port Characteristics
3. JI Port Characteristics
e. MEM 10112 Control Structure and Operation (Fig. 207)
1. MEM 10112 Control Structure
2. MEM 10112 Control Operation
f. MEM 10112 Operations
g. MEM 10112 Interfaces to JP 10114 and IOS 10116 (Figs. 209, 210, 211, 204)
1. IO Port 20910 Operating Characteristics (Figs 209, 204)
2. JO Port 21010 Operating Characteristics (Fig. 210)
3. JI Port 21110 Operating Characteristics (Fig. 211)
h. FIU 20120 (Figs. 201, 230, 231)
B. Fetch Unit 10120 (Figs. 202, 206, 101, 103, 104, 238)
1. Descriptor Processor 20210 (Figs. 202, 101, 103, 104, 238, 239).
a. Offset Processor 20218 Structure
b. AON Processor 20216 Structure
c. Length Processor 20220 Structure
d. Descriptor Processor 20218 Operation
a.a. Offset Selector 20238
b.b. Offset Multiplexer 20240 Detailed Structure (Fig. 238)
c.c. Offset Multiplexer 20240 Detailed Operation
aaa. Internal Operation
bbb. Operation Relative to DESP 20210
e. Length Processor 20220 (Fig. 239)
a.a. Length ALU 20252
b.b. BIAS 20246 (Fig. 239)
f. AON Processor 20216
a.a. AONGRF 20232
b.b. AON Selector 20248
2. Memory Interface 20212 (Figs. 106, 240)
a.a. Descriptor Trap 20256 and Data Trap 20258
b.b. Name Cache 10226, Address Translation Unit 10228, and Protection Cache 10234 (Fig. 106)
c.c. Structure and Operation of Generalized Cache and NC 10226 (Fig. 240)
d.d. ATU 10228 and PC 10234
3. Fetch Unit Control Logic 20214 (Fig. 202)
a.a. Fetch Unit Control Logic 20214 Overall Structure
b.b. Fetch Unit Control Logic 20214 Operation
a.a.a. Prefetcher 20264, Instruction Buffer 20262, Parser 20264, Operation Code Register 20268, CPC 20270, IPC 20272, and EPC 20274 (Fig. 241)
b.b.b. Fetch Unit Dispatch Table 11010, Execute Unit Dispatch Table 20266, and Operation Code Register 20268 (Fig. 242)
c.c.c. Next Address Generator 24310 (Fig. 243)
cc. FUCTL 20214 Circuitry for CS 10110 Internal Mechanisms (Figs. 244-250)
a.a.a. State Logic 20294 (Figs. 244A—244Z)
b.b.b. Event Logic 20284 (Figs. 245, 246, 247, 248)
c.c.c. Fetch Unit S-Interpreter Table 11012 (Fig. 249)
d.d. CS 10110 Internal Mechanism Control
a.a.a. Return Control Word Stack 10358 (Fig. 251)
b.b.b. Machine Control Block (Fig. 252)
c.c.c. Register Address Generator 20288 (Fig. 253)

a.a. MAS Base 10410 (Fig. 468)
b.b. Per-domain Data Area 46853 (Fig. 468)
c.c. MAS Frame 46709 Detail (Fig. 469)

3. SS 504 (Fig. 470)
   a.a. SS Base 47001 (Fig. 471)
   b.b. SS Frames 47003 (Fig. 471)
      a.a.a. Ordinary SS Frame Headers 10514 (Fig. 471)
      b.b.b. Detailed Structure of Macrostate 10516 (Fig. 471)
      c.c.c. Cross-domain SS Frames 47039 (Fig. 471)
4. Portions of Procedure Object 608 Relevant to Call and Return (Fig. 472)
5. Execution of Mediated Calls
   a.a. Mediated Call SINs
   b.b. Simple Mediated Calls (Figs. 270, 468, 469, 470, 471, 472)
   c.c. Invocations of Procedures 602 Requiring SEBs 46864 (Figs. 270, 468, 469, 470, 471, 472)
   d.d. Cross-Procedure Object Calls (Figs. 270, 468, 469, 470, 471, 472)
   e.e. Cross-Domain Calls (Figs. 270, 408, 418, 468, 469, 470, 471, 472)
   f.f. Failed Cross-Domain Calls (Figs. 270, 468, 469, 470, 471, 472)
6. Neighborhood Calls (Figs. 468, 469, 472)

INTRODUCTORY OVERVIEW

The following overview will first briefly describe the overall physical structure and operation of a presently preferred embodiment of a digital computer system incorporating the present invention. Then certain operating features of that computer system will be individually described. Next, overall operation of the computer system will be described in terms of those individual features.

A. Hardware Overview (Fig. 1)

Referring to Fig. 1, a block diagram of Computer System (CS) 101 incorporating the present invention is shown. Major elements of CS 101 are I/O System (IOS) 116, Memory (MEM) 112, and Job Processor (JP) 114. JP 114 is comprised of a Fetch Unit (FU) 120 and an Execute Unit (EU) 122. CS 101 may also include a Diagnostic Processor (DP), not shown or described in the instant description.

Referring first to IOS 116, a primary function of IOS 116 is control of transfer of information between MEM 112 and the outside world. Information is transferred from MEM 112 to IOS 116 through IOM Bus 130, and from IOS 116 to MEM 112 through MIO Bus 129. IOMC Bus 131 is comprised of bi-directional control signals coordinating operation of MEM 112 and IOS 116. IOS 116 also has an interface to FU 120 through IOJP Bus 132. IOJP Bus 132 is a bi-directional control bus comprised essentially of two interrupt lines. These interrupt lines allow FU 120 to indicate to IOS 116 that a request for information by FU 120 has been placed in MEM 112, and allows IOS 116 to inform FU 120 that information requested by FU 120 has been transferred into a location in MEM 112. MEM 112 is CS 101's main memory and serves as the path for information transfer between the outside world and JP 114. MEM 112 provides instructions and data to FU 120 and EU 122 through Memory Output Data (MOD) Bus 140 and receives information from FU 120 and EU 122 through Job Processor Data (JPD) Bus 142. FU 120 submits read and write requests to MEM 112 through Physical Descriptor (PD) Bus 146.

JP 114 is CS 101's CPU and, as described above, is comprised of FU 120 and EU 122. A primary function of FU 120 is executing operations of user's programs. As part of this function, FU 120 controls transfer of instructions and data from MEM 112 and transfer of results of JP 114 operations back to MEM 112. FU 120 also performs operating system type functions, and is capable of operating as a complete, general purpose CPU. EU 122 is primarily an arithmetic and logic unit provided to relieve FU 120 of certain arithmetic operations. FU 120, however, is capable of performing EU 122 operations. In alternate embodiments of CS 101, EU 122 may be provided only as an option for users having particular arithmetic requirements. Coordination of FU 120 and EU 122 operations is accomplished through FU/EU (FUEU) Bus 148, which includes bi-directional control signals and mutual interrupt lines. As described further below, both FU 120 and EU 122 contain register file arrays referred to respectively as CRF and ERF, in addition to registers associated with, for example, ALUs.

A primary feature of CS 101 is that IOS 116, MEM 112, FU 120 and EU 122 each contain separate and independent microinstruction control, so that IOS 116, MEM 112, and EU 122 operate asynchronously under the general control of FU 120. EU 122, for example, may execute a complex arithmetic operation upon receipt of data and a single, initial command from FU 120.

Having briefly described the overall structure and operation of CS 101, certain features of CS 101 will be individually further described next below.

B. Individual Operating Features (Figs. 2, 3, 4, 5, 6)

1. Addressing (Fig. 2)

Referring to Fig. 2, a diagramic representation of portions of CS 101's addressing structure is shown. CS 101's addressing structure is based upon the concept of Objects. An Object may be regarded as a

container for holding a particular type of information. For example, one type of Object may contain data while another type of Object may contain instructions or procedures, such as a user program. Still another type of Object may contain microcode. In general, a particular Object may contain only one type or class of information. An Object may, for example, contain up to 232 bits of information, but the actual size of a particular Object is flexible. That is, the actual size of a particular Object will increase as information is written into that Object and will decrease as information is taken from that Object. In general, information in Objects is stored sequentially, that is without gaps.

Each Object which can ever exist in any CS 101 system is uniquely identified by a serial number referred to as a Unique Identifier (UID). A UID is a 128 bit value comprised of a serial number dependent upon, for example, the particular CS 101 system and user, and a time code indicating time of creation of that Object. UIDs are permanently assigned to Objects, no two Objects may have the same UID, and UIDs may not be reused. UIDs provide an addressing base common to all CS 101 systems which may ever exist, through which any Object ever created may be permanently and uniquely identified.

As described above, UIDs are 128 bit values and are thus larger than may be conveniently handled in present embodiments of CS 101. In each CS 101, therefore, those Objects which are active (currently being used) in that system are assigned 14 bit Active Object Numbers (AONs). Each Object active in that system will have a unique AON. Unlike UIDs, AONs are only temporarily assigned to particular Objects. AONs are valid only within a particular CS 101 and are not unique between systems. An Object need not physically reside in a system to be assigned an AON, but can be active in that system only if it has been assigned an AON.

A particular bit within a particular Object may be identified by means of a UID address or an AON address. In CS 101, AONs and AON addresses are valid only within JP 114 while UIDs and UID addresses are used in MEM 112 and elsewhere. UID and AON addresses are formed by appending a 32 bit Offset (O) field to that Object's UID or AON. O fields indicate offset, or location, of a particular bit relative to the start of a particular Object.

Segments of information (sequences of information bits) within particular Objects may be identified by means of descriptors. A UID descriptor is formed by appending a 32 bit Length (L) field of a UID address. An AON, or logical descriptor is formed by appending a 32 bit L field to an AON address. L fields identify length of a segment of information bits within an Object, starting from the information bit identified by the UID or AON address. In addition to length information, UID and logical descriptors also contain Type fields containing information regarding certain characteristics of the information in the information segment. Again, AON based descriptors are used within JP 114, while UID based descriptors are used in MEM 112.

Referring to Figs. 1 and 2 together, translation between UID addresses and descriptors and AON addresses and descriptors is performed at the interface between MEM 112 and JP 114. That is, addresses and descriptors within JP 114 are in AON form while addresses and descriptors in MEM 112, IOS 116, and the external world are in UID form. In other embodiments of CS 101 using AONs, transformation from UID to AON addressing may occur at other interfaces, for example at the IOS 116 to MEM 112 interface, or at the IOS 116 to external world interface. Other embodiments of CS 101 may use UIDs throughout, that is not use AONs even in JP 114.

Finally, information within MEM 112 is located through MEM 112 Physical Addresses identifying particular physical locations within MEM 112's memory space. Both IOS 116 and JP 114 address information within MEM 112 by providing physical addresses to MEM 112. In the case of physical addresses provided by JP 114, these addresses are referred to as Physical Descriptors (PDs). As described below, JP 114 contains circuitry to translate logical descriptors into physical descriptors.

2. S-Language Instructions and Namespace Addressing (Fig. 3)

CS 101 is both an S-Language machine and a Namespace machine. That is, operations to be executed by CS 101 are expressed as S-Language Operations (SOPs) while operands are identified by Names. SOPs are of a lower, more detailed, level than user language instructions, for example FORTRAN and COBOL, but of a higher level than conventional machine language instructions. SOPs are specific to particular user languages rather than a particular embodiment of CS 101, while conventional machine language instructions are specific to particular machines. SOPs are in turn interpreted and executed by microcode. There will be an S-Language Dialect, a set of SOPs, for each user languages. CS 101, for example, may have SOP Dialects for COBOL, FORTRAN, and SPL. A particular distinction of CS 101 is that all SOPs are of a uniform, fixed length, for example 16 bits. CS 101 may generally contain one or more sets of microcode for each S-Language Dialect. These microcode Dialect Sets may be completely distinct, or may overlap where more than one SOP utilizes the same microcode.

As stated above, in CS 101 all operands are identified by Names, which are 8, 12, or 16 bit numbers. CS 101 includes one or more "Name Tables" containing an Entry for each operand Name appearing in programs currently being executed. Each Name Table Entry contains information describing the operand referred to by a particular Name, and the directions necessary for CS 101 to translate that information into a corresponding logical descriptor. As previously described, logical descriptors may then be transformed into physical descriptors to read and write operands from or to MEM 112. As described above, UIDs are unique for all CS 101 systems and AONs are unique within individual CS 101 systems. Names, however, are unique only within the context of a user's program. That is, a particular Name may appear in two different

user's programs and, within each program, will have different Name Table Entries and will refer to different operands.

CS 101 may thereby be considered as utilizing two sets of instructions. A first set is comprised of SOPs, that is instructions selecting algorithms to be executed. The second set of instructions are comprised of Names, which may be regarded as entry points into tables of instructions for making references regarding operands.

Referring to Fig. 3, a diagramic representation of CS 101 instruction stream is shown. A typical SIN is comprised of an SOP and may include one or more Names referring to operands. SOPs and Names allow user's programs to be expressed in very compact code. Fewer SOPs than machine language instructions are required to express a user's program. Also, use of SOPs allows easier and simpler construction of compilers, and facilitates adaption of CS 101 systems to new user languages. In addition, use of Names to refer to operands means that SOPs are independent of the form of the operands upon which they operate. This in turn allows for more compact code in expressing user programs in that SOPs specifying operations dependent upon operand form are not required.

3. Architectural Base Pointer Addressing

As will be described further below, a user's program residing in CS 101 will include one or more Objects. First, a Procedure Object contains at least the SINs of the user's programs and a Name Table containing entries for operand Names of the program. The SINs may include references, or calls, to other Procedure Objects containing, for example, procedures available in common to many users. Second, a Static Data Area may contain static data, that is data having an existence for at least a single execution of the program. And third, a Macro-stack, described below, may contain local data, that is data generated during execution of a program. Each Procedure Object, the Static Data Area and the Macro-stack are individual Objects identified by UIDs and AONs and addressable through UID and AON addresses an descriptors.

Locations of information within a user's Procedure Objects, Static Data Area, and Macro-stack are expressed as offsets from one of three values, or base addresses, referred to as Architectural Base Pointers (ABPs). For example, location information in Name Tables is expressed as offsets from one of the ABPs. ABPs may be expressed as previously described.

The three ABPs are the Frame Pointer (FP), the Procedure Base Pointer (PBP), and the Static Data Pointer (SDP). Locations of data local to a procedure, for example in the procedure's Macrostack, are described as offsets from FP. Locations of non-local data, that is Static Data, are described as offsets from SDP. Locations of SINs in Procedure Objects are expressed as offsets from PBP; these offsets are determined as a Program Counter (PC) value. Values of the ABPs vary during program execution and are therefore not provided by the compiler converting a user's high level language program into a program to be executed in a CS 101 system. When the program is executed, CS 101 provides the proper values for the ABPs. When a program is actually being executed, the ABP's values are stored in FU 120's GRF.

Other pointers are used, for example, to identify the top frame of CS 101's Secure Stack (a microcode level stack described below) or to identify the microcode Dialect currently being used in execute the SINs of a procedure. These pointers are similar to FP, SDP, and PBP.

4. Stack Mechanisms (Fig. 4—5)

Referring to Fig. 4 and 4A, diagramic representations of various control levels and stack mechanisms of, respectively, conventional machines and CS 101, are shown. Referring first to Fig. 4, top level of control is provided by User Language Instructions 402, for example in FORTRAN or COBOL. User Language Instructions 402 are converted into a greater number of more detailed Machine Language Instructions 404, used within a machine to execute user's programs. Within the machine, Machine Language Instructions 404 are interpreted and executed by Microcode Instructions 406, that is sequences of microinstructions which in turn directly control Machine Hardware 408. Some conventional machines may include a Stack Mechanism 410 used to save current machine state, that is current microinstruction and contents of various machine registers, if a current Machine Language Instruction 404 cannot be executed or is interrupted. In general, machine state on the microcode and hardware level is not saved. Execution of a current Machine Language Instruction 404 is later resumed at start of the microinstruction sequence for executing that Machine Language Instruction 404.

Referring to Fig. 4A, top level control in CS 101 is by User Language Instructions 412 as in a conventional machine. In CS 101, however, User Language Instructions 412 are translated into SOPs 414 which are of a higher level than conventional machine language instructions. In general, a single User Language Instruction 412 is transformed into at most two or three SOPs 414, as opposed to an entire sequence of conventional Machine Language Instructions 404. SOPs 414 are interpreted and executed by Microcode Instructions 416 (sequences of microinstructions) which directly control CS 101 Hardware 418. CS 101 includes a Macro-stack Mechanism (MAS) 420, at SOPs 414 level, which is comparable to but different in construction and operation from a conventional Machine Language Stack Mechanism 410. CS 101 also includes Micro-code Stack Mechanisms 422 operating at Microcode 416 level, so that execution of an interrupted microinstruction of a microinstruction sequence may be later resumed with the particular microinstruction which was active at the time of the interrupt. CS 101 is therefore more efficient in handling

interrupts in that execution of microinstruction sequences is resumed from the particular point that a microinstruction sequence was interrupted, rather from the beginning of that sequence. As will be described further below, CS 101's Micro-code Stack Mechanisms 422 on microcode level is effectively comprised of two stack mechanisms. The first stack is Micro-instruction Stack (MIS) 424 while the second stack is referred to as Monitor Stack (MOS) 426. CS 101 SIN Microcode 428 and MIS 424 are primarily concerned with execution of SOPs of user's programs. Monitor Microcode 430 and MOS 426 are concerned with operation of certain CS 101 internal functions.

Division of CS 101's microcode stacks into an MIS 424 and a MOS 426 illustrates a further feature of CS 101. In conventional machines, monitor functions may be performed by a separate CPU operating in conjunction with the machine's primary CPU. In CS 101, a single hardware CPU is used to perform both functions with actual execution of both functions performed by separate groups of microcode. Monitor microcode operations may be initiated either by certain SINs 414 or by control signals generated directly by CS 101's Hardware 418. Invocation of Monitor Microcode 430 by Hardware 418 generated signals insures that CS 101's monitor functions may always be invoked.

Referring to Fig. 5, a diagramic representation of CS 101,s stack mechanisms for *a single user's program, or procedure*, is shown. Basically, and with exception of MOS 426, CS 101's stacks reside in MEM 112 with certain portions of those stacks accelerated into FU 120 and EU 122 to enhance speed of operation.

Certain areas of MEM 112 storage space are set aside to contain Macro-Stacks (MASs) 502, stack mechanisms operating on the SINs level, as described above. Other areas of MEM 112 are set aside to contain Secure Stack (SS) 504, operating on the microcode level, as described above and of which MIS 424 is a part.

As described further below, both FU 120 and EU 122 contain register file arrays, referred to respectively as GRF and ERF, in addition to registers associated with, for example, ALUs. Referring to FU 120, shown therein is FU 120's GRF 506. GRF 506 is horizontally divided into three areas. A first area, referred to as General Registers (GRs) 508 may in general be used in the same manner as registers in a conventional machine. A second area of GRF 506 is Micro-Stack (MIS) 424, and is set aside to contain a portion of a Process's SS 504. A third portion of GRF 506 is set aside to contain MOS 426. Also indicated in FU 120 is a block referred to as Microcode Control State (mCS) 510. mCS 510 represents registers and other FU 120 hardware containing current operating state of FU 120 on the microinstruction and hardware level. mCS 510 may include, for example, the current microinstruction controlling operation of FU 120.

Referring to EU 122, indicated therein is a first block referred to as Execute Unit State (EUS) 512 and a second block referred to as SOP Stack 514. EUS 512 is similar to mCS 510 in FU 120 and includes all registers and other EU 122 hardware containing information reflecting EU 122's current operating state. SOP Stack 518 is a portion of EU 122's ERF 516 which has been set aside as a stack mechanism to contain a portion of a process's SS 504 pertaining to EU 122 operations.

Considering first MASs 502, as stated above MASs 502 operate generally upon the SINs level. MASs 502 are used in general to store current state of a process's (defined below) execution of a user's program.

Referring next to MIS 424, in a present embodiment of CS 101 that portion of GRF 506 set aside to contain MIS 424 may have a capacity of eight stack frames. That is, up to 8 microinstruction level interrupts or calls pertaining to execution of a user's program may be stacked within MIS 424. Information stored in MIS 424 stack frames is generally information from GR 508 and MCS 510. MIS 424 stack frames are transferred between MIS 424 and SS 504 such that at least one frame, and no more than 8 frames, of SS 504 reside in GRF 506. This insures that at least the top-most frames of a process's SS 504 are present in FU 120, thereby enhancing speed of operation of FU 120 by providing rapid access to those top frames. SS 504, residing in MEM 112, may contain, for all practical purposes, an unlimited number of frames so that MIS 424 and SS 504 appear to a user to be effectively an infinitely deep stack.

MOS 426 resides entirely in FU 120 and, in a present embodiment of CS 101, may have a capacity of 8 stack frames. A feature of CS 101 operation is that CS 101 mechanisms for handling certain events or interrupts should not rely in its operation upon those portions of CS 101 whose operation has resulted in those faults or interrupts. Among events handled by CS 101 monitor microcode, for example, are MEM 112 page faults. An MEM 112 page fault occurs whenever FU 120 makes a reference to data in MEM 112 and that data is not in MEM 112. Due to this and similar operations, MOS 426 resides entirely in FU 120 and thus does not rely upon information in MEM 112.

As described above, GRs 508, MIS 424, and MOS 426 each reside in certain assigned portions of GRF 506. This allows flexibility in modifying the capacity of GRs 508, MIS 424, and MOS 426 as indicated by experience, or to modify an individual CS 101 for particular purposes.

Referring finally to EU 122, EUS 512 is functionally a part of a process's SS 504. Also as previously described, EU 122 performs arithmetic operations in response to SINs and may be interrupted by FU 120 to aid certain FU 120 operations. EUS 512 allows stacking of interrupts. For example, FU 120 may first interrupt an arithmetic SOP to request EU 122 to aid in evaluation of a Name Table Entry. Before that first interrupt is completed, FU 120 may interrupt again, and so on.

SOP Stack 514, is a single frame stack for storing current state of EU 122 when an interrupt interrupts execution of an arithmetic SOP. An interrupted SOP's state is transferred into SOP Stack 514 and the interrupt begins execution in EUS 512. Upon occurrence of a second interrupt (before the first interrupt is completed) EU's first interrupt state is transferred from EUS 512 to a stack frame in SS 504, and execution

of the second interrupt begins in EUS 512. If a third interrupt occurs before completion of second interrupt, EU's second interrupt state is transferred from EUS 512 to another stack frame in SS 504 and execution of the third interrupt is begun in EUS 512; and so on. EUS 512 and SS 504 thus provide an apparently infinitely deep microstack for EU 122. Assuming that the third interrupt is completed, state of second interrupt is transferred from SS 504 to EUS 512 and execution of second interrupt resumed. Upon completion of second interrupt, state of first interrupt is transferred from SS 504 to EUS 512 and completed. After completion of first interrupt, state of the original SOP is transferred from SOP Stack 514 to EUS 512 and execution of that SOP resumed.

C. Procedure Processes, and Virtual Processors (Fig. 6)

Referring to Fig. 6, a diagramic representation of procedures, processes, and virtual processes is shown. As described above, a user's program to be executed is compiled to result in a Procedure 602. A Procedure 602 includes a User's Procedure Object 604 containing the SOPs of the user's program and a Name Table containing Entries for operand Names of the user's program, and a Static Data Area 606. A Procedure 602 may also include other Procedure Objects 608, for example utility programs available in common to many users. In effect, a Procedure 602 contains the instructions (procedures) and data of a user's program.

A Process 610 includes, as described above, a Macro Stack (MAS) 502 storing state of execution of a user's Procedure 602 at the SOP level, and a Secure Stack (SS) 504 storing state of execution of a user's Procedure 602 at the microcode level. A Process 610 is associated with a user's Procedure 602 through the ABPs described above and which are stored in the MAS 502 of the Process 610. Similarly, the MAS 502 and SS 504 of a Process 610 are associated through non-architectural pointers, described above. A Process 602 is effectively a body of information linking the resources, hardware, microcode, and software, of CS 101 to a user's Procedure 602. In effect, a Process 610 makes the resources of CS 101 available to a user's Procedure 602 for executing of that Procedure 602. CS 101 is a multi-program machine capable of accommodating up to, for example, 128 processes 610 concurrently. The number of Processes 610 which may be executed concurrently is determined by the number of Virtual Processors 612 of CS 101. There may be, for example, up to 16 Virtual Processors 612.

As indicated in Fig. 6, a Virtual Processor 612 is comprised of a Virtual Processor State Block (VPSB) 614 associated with the SS 504 of a Process 612. A VPSB 614 is, in effect, a body of information accessible to CS 101's operating system and through which CS 101's operating system is informed of, and provided with access to, a Process 610 through that process 610's SS 504. A VPSB 614 is associated with a particular Process 610 by writing information regarding that Process 610 into that VPSB 614. CS 101's operating system may, by gaining access to a Process 610 through an associated PSB 614, read information, such as ABP's, from that Process 610 to FU 120, thereby swapping that Process 610 onto FU 120 for execution. It is said that a Virtual Processor 612 thereby executes a process 610; a Virtual Processor 612 may be regarded therefor, as a processor having "Virtual", or potential, existence which becomes "real" when its associated Process 610 is swapped into FU 120. In CS 101, as indicated in Fig. 6, only one Virtual Processor 612 may execute on FU 120 at a time and the operating system selects which Virtual Processor 612 will excecute on FU 120 at any given time. In addition, CS 101's operating system selects which Processes 610 will be associated with the available Virtual Processors 612.

Having briefly described certain individual structural and operating features of CS 101, the overall operation of CS 101 will be described in further detail next below in terms of these individual features.

D. CS 101 Overall Structure and Operation (Figs. 7, 8, 9, 10, 11, 12, 13, 14, 15)

1. Introduction (Fig. 7)

As indicated in Fig. 7, CS 101 is a multiple level system wherein operations in one level are generally transparent to higher levels. User 701 does not see the S-Language, addressing, and protection mechanisms defined at Architectural Level 708. Instead, he sees User Interface 709, which is defined by Compilers 702, Binder 703, and Extended (high level) Operating System (EOS) 704. Compilers 702 translate high-level language code into SINs and Binder 703 translates symbolic Names in programs into UID-offset addresses.

As Fig. 7 shows, Architectural Level 708 is not defined by FU 120 Interface 711. Instead, the architectural resources level are created by S-Language Interpreted SINs when a program is executed; Name Interpreter 715 operates under control of S-Language Interpreters 705 and translates Names into logical descriptors. In CS 101, both S-Language Interpreters 705 and Name Interpreter 715 are implemented as microcode which executes on FU 120. S-Language Interpreters 705 may also use EU 122 to perform calculations. A Kernel Operating System (KOS) provides CS 101 with UID-offset addressing, objects, access checking, processes, and virtual processors, described further below. KOS has three kinds of components: KOS Microcode 710, KOS Software 706, and KOS Tables in MEM 112. KOS 710 components are microcode routines which assist FU 120 in performing certain required operations. Like other high-level language routines, KOS 706 components contain SINs which are interpreted by S-Interpreter Microcode 705. Many KOS High-Level Language Routines 706 are executed by special KOS processes; others may be executed by any process. Both KOS High-Level Language Routines 706 and KOS Microcode 710 manipulate KOS Tables in MEM 112.

FU 120 Interface 711 is visible only to KOS and to S-Interpreter Microcode 705. For the purposes of this discussion, FU 120 may be seen as a processor which contains the following main elements:

A Control Mechanism 725 which executes microcode stored in Writable Control Store 713 and manipulates FU 120 devices as directed by this microcode.

A GRF 506 containing registers in which data may be stored.

A Processing Unit 715.

All microcode which executes on FU 120 uses these devices; there is in addition a group of devices for performing special functions; these devices are used only by microcode connected with those functions. The microcode, the specialized devices, and sometimes tables in MEM 112 make up logical machines for performing certain functions. These machines will be described in detail below.

In the following, each of the levels illustrated in Fig. 7 will be discussed in turn. First, the components at User Interface 709 will be examined to see how they translate user programs and requests into forms usable by CS 101. Then the components below the User Interface 709 will be examined to see how they create logical machines for performing CS 101 operations.

2. Compilers 702 (Fig. 7)

Compilers 702 translate files containing the highlevel language code written by User 701 into Procedure Objects 608. Two components of a Procedure Object 608 are code (SOPs) and Names, previously described. SOPs represent operations, and the Names represent data. A single SIN thus specifies an operation to be performed on the data represented by the Names.

3. Binder 703 (Fig. 7)

In some cases, Compiler 702 cannot define locations as offsets from an ABP. For example, if a procedure calls a procedure contained in another procedure object, the location to which the call transfers control cannot be defined as an offset from the PBP used by the calling procedure. In these cases, the compiler uses symbolic Names to define the locations. Binder 703 is a utility which translates symbolic Names into UID-offset addresses. It does so in two ways: by combining separate Procedure Objects 608 into a single large Procedure Object 608, and then redefining symbolic Names as offsets from that Procedure Object 608's ABPs, or by translating symbolic Names when the program is executed. In the second case, Binder 703 requires assistance from EOS 704.

4. EOS 704 (Fig. 7)

EOS 704 manages the resources that User 701 requires to execute his programs. From User 701's point of view, the most important of these resources are files and processes. EOS 704 creates files by requesting KOS to create an object and then mapping the file onto the object. When a User 701 performs an operation on a file, EOS 704 translates the file operation into an operation on an object. KOS creates them at EOS 704's request and makes them available to EOS 704, which in turn makes them available to User 701. EOS 704 causes a process to execute by associating it a Virtual Processor 612. In logical terms, a Virtual Processor 612 is the means which KOS provides EOS 704 for executing processes 610. As many Processes 610 may apparently execute simultaneously in CS 101 as there are Virtual Processors 612. The illusion of simultaneous execution is created by multiplexing JP 114 among the Virtual processors; the manner in which Processes 610 and Virtual Processors 610 are implemented will be explained in detail below.

5. KOS and Architectural Interface 708 (Fig. 7)

S-Interpreter Microcode 710 and Name Interpreter Microcode 715 require an environment provided by KOS Microcode 710 and KOS Software 706 to execute SINs. For example, as previously explained, Names and program locations are defined in terms of ABPs whose values vary during execution of the program. The KOS environment provides values for the ABPs, and therefore makes it possible to interpret Names and program locations as locations in MEM 112. Similarly, KOS help is required to transform logical descriptors into references to MEM 112 and to perform protection checks.

The environment provided by KOS has the following elements:

A Process 610 which contains the state of an execution of the program for a given User 701.

A Virtual Processor 612 which gives the Process 610 access to JP 114.

An Object Management System which translates UIDs into values that are usable inside JP 114.

A Protection System which checks whether a Process 610 has the right to perform an operation on an Object.

A Virtual Memory Management System which moves those portions of Objects which a Process 610 actually references from the outside world into MEM 112 and translates logical descriptors into physical descriptors.

In the following, the logical properties of this environment and the manner in which a program is executed in it will be explained.

6. Processes 610 and Virtual Processors 612 (Fig. 8)

Processes 610 and Virtual Processors 612 have already been described in logical terms; Fig. 8 gives a high-level view of their physical implementation.

Fig. 8 illustrates the relationship between Processes 610, Virtual Processors 612, and JP 114. In physical terms, a Process 610 is an area of MEM 112 which contains the current state of a user's execution of a program. One example of such state is the current values of the ABPs and a program Counter (PC). Given the current value of the PBP and the PC, the next SOP in the program can be executed; similarly, given the current values of SDP and FP, the program's Names can be correctly resolved. Since the Process 610 contains the current state of a program's execution, the program's physical execution can be stopped and resumed at any point. It is thus possible to control program execution by means of the Process 610.

As already mentioned, a process 610's execution proceeds only when KOS has bound it to a Virtual Processor 612, that is, an area of MEM 112 containing the state required to execute microinstructions on JP 114 hardware. The operation of binding is simply a transfer of Process 610 state from the Process 610's area of MEM 112 to a Virtual Processor 612's area of MEM 112. Since binding and unbinding may take place at any time, EOS 704 may multiplex Processes 610 among Virtual Processors 612. In Fig. 8, there are more Processes 610 than there are Virtual Processors 612. The physical execution of a Process 610 on JP 114 takes place only while the Process 610's Virtual Processor 612 is bound to JP 114, i.e., when state is transferred from Virtual Processor 612's area of MEM 112 to JP 114's registers. Just as EOS 704 multiplexes Virtual Processors 612 among Processes 610, KOS multiplexes JP 114 among Virtual Processors 612. In Fig. 8, only one Process 610 is being physically executed. The means by which JP 114 is multiplexed among Virtual Processors 612 will be described in further detail below.

7. Processes 610 and Stacks (Fig. 9)

In CS 101 systems, a Process 610 is made up of six Objects: one Process Object 901 and Five Stack Objects 902 to 906. Fig. 9 illustrates a Process 610. Process Object 901 contains the information which EOS 704 requires to manage the Process 610. EOS 704 has no direct access to Process Object 901, but instead obtains the information it needs by means of functions provided to it by KOS 706, 710. Included in the information are the UIDs of Stack Objects 902 through 906. Stack Objects 902 to 906 contain the Process 610's state.

Stack Objects 902 through 905, are required by CS 101's domain protection method and comprise Process 610's MAS 502. Briefly, a domain is determined in part by operations performed when a system is operating in that domain. For example, the system is in EOS 704 domain when executing EOS 704 operations and in KOS 706, 710 domain when executing KOS 706, 710 operations. A Process 610 must have one stack for each domain it enters. In the present embodiment, the number of domains is fixed at four, but alternate embodiments may allow any number of domains, and correspondingly, any number of Stack Objects. Stack Object 906 comprises Process 610's Secure Stack 504 and is required to store state which may be manipulated only by KOS 706, 710.

Each invocation made by a Process 610 results in the addition of frames to Secure Stack 504 and to Macro-Stack 502. The state stored in the Secure Stack 504 frame includes the macrostate for the invocation, the state required to bind Process 610 to a Virtual Processor 612. The frame added to Macro-Stack 502 is placed in one of Stack Objects 902 through 905. Which Stack Objects 902 to 905 gets the frame is determined by the invoked procedure's domain of execution.

Fig. 9 shows the condition of a Process 610's MAS 502 and Secure Stack 504 after the Process 610 has executed four invocations. Secure Stack 504 has one frame for each invocation; the frames of Process 610's MAS 502 are found in Stack Objects 902, 904, and 905. As revealed by their locations, Frame 1 is for an invocation of a routine with KOS 706, 710 domain of execution, Frame 2 for an invocation of a routine with the EOS 704 domain of execution, and Frames 3 and 4 for invocations of routines with the User domain of execution. Process 610 has not yet invoked a routine with the Data Base Management System (DBMS) domain of execution. The frames in Stack Objects 902 through 905 are linked together, and a frame is added to or removed from Secure Stack 504 every time a frame is added to Stack Objects 902 through 905. MAS 502 and Secure Stack 504 thereby function as a single logical stack even though logically contained in five separate Objects.

8. Processes 610 and Calls (Figs. 10, 11)

In the CS 101, calls and returns are executed by KOS 706, 710. When KOS 706, 710 performs a call for a process, it does the following:

It saves the calling invocation's macrostate in the top frame of Secure Stack 504 (Fig. 9).

It locates the procedure whose Name is contained in the call. The location of the first SIN in the procedure becomes the new PBP.

Using information contained in the called procedure, KOS 706, 710 creates a new MAS 502 frame in the proper Stack Object 902 through 905 and a new Secure Stack 504 frame in Secure Stack 504. FP is updated to point to the new MAS 502. If necessary, SDP is also updated.

Once the values of the ABPs have been updated, the PC is defined, Names can be resolved, and execution of the invoked routine can commence. On a return from the Invocation to the invoking routine, the stack frames are deleted and the ABPs are set to the values saved in the invoking routine's macrostate. The invoking routine then continues execution at the point following the invocation.

A Process 610 may be illustrated in detail by putting the FORTRAN statement A + B into a FORTRAN routine called EXAMPLE and invoking it from another FORTRAN routine named CALLER. To simplify the

example, it is assumed that CALLER and EXAMPLE both have the same domain of execution. The parts of EXAMPLE which are of interest look like this:

```
        SUBROUTINE EXAMPLE (C)

        INTEGER X,C

        INTEGER A,B

        ...

        A = B
        ...

        RETURN

        END
```

The new elements are a formal argument, C, and a new local variable, X. A formal argument is a data item which receives its value from adata item used in the invoking routine. The formal argument's value thus varies from invocation to invocation. The portions of INVOKER which are of interest look like this:

```
        SUBROUTINE INVOKER

        INTEGER Z

        ...

            CALL EXAMPLE (Z)

        ...

        END
```

The CALL statement in INVOKER specifies the Name of the subroutine being invoked and the actual arguments for the subroutine's formal arguments. During the invocation, the subroutine's formal arguments take on the values of the actual arguments. Thus, during the invocation specified by this CALL statement, the formal argument C will have the value represented by the variable Z in INVOKER.

When INVOKER is compiled, the compiler produces a CALL SIN corresponding to the CALL statement. The CALL SIN contains a Name representing a pointer to the beginning of the called routine's location in a procedure object and a list of Names representing the call's actual arguments. When CALL is executed, the Names are interpreted to resolve the SIN's Names as previously described, and KOS 710 microcode to perform MAS 502 and Secure Stack 504 operations.

Fig. 10 illustrates the manner in which the KOS 710 call microcode manipulates MAS 502 and Secure Stack 504.

Fig. 10 includes the following elements:

Call Microcode 1001, contained in FU 120 Writable Control Store 1014.

PC Device 1002, which contains part of macrostate belonging to the invocation of INVOKER which is executing the CALL statement.

Registers in FU Registers 1014. Registers 1004 contents include the remainder of macrostate and the descriptors corresponding to Names for EXAMPLE's location and the actual argument Z.

Procedure Object 1006 contains the entries for INVOKER and EXAMPLE, their Name Tables, and their code.

Macro-Stack Object 1008 (MAS 502) and Secure Stack Object 1010 (Secure Stack 504) contain the stack frames for the invocations of INVOKER and EXAMPLE being discussed here. EXAMPLE's frame is in the same Macro-Stack object as INVOKER's frame because both routines are contained in the same Procedure Object 1006, and therefore have the same domain of execution.

KOS Call Microcode 1001 first saves the macrostate of INVOKER's invocation on Secure Stack 504. As will be discussed later, when the state is saved, KOS 706 Call Microcode 1001 uses other KOS 706 microcode to translate the location information contained in the macrostate into the kind of pointers used in MEM 112. Then Microcode 1001 uses the descriptor for the routine Name to locate the pointer to EXAMPLE's entry in Procedure Object 1006. From the entry, it locates pointers to EXAMPLE's Name Table and the beginning of EXAMPLE's code. Microcode 1001 takes these pointers, uses other KOS 706 microcode to translate them into descriptors, and places the descriptors in the locations in Registers 1004 reserved for the values of the PBP and NTP. It then updates the values contained in PC Device 1002 so that

when the call is finished, the next SIN to be executed will be the first SIN in EXAMPLE.

CALL Microcode 1001 next constructs the frames for EXAMPLE on Secure Stack 504 and Macro-Stack 502. This discussion concerns itself only with Frame 1102 on Macro-Stack 502. Fig. 11 illustrates EXAMPLE's Frame 1102. The size of Frame 1102 is determined by EXAMPLE's local variables (X, A, and B) and formal arguments (C). At the bottom of Frame 1102 is Header 1104. Header 1104 contains information used by KOS 706, 710 to manage the stack. Next comes Pointer 1106 to the location which contains the value represented by the argument C. In the invocation, the actual for C is the local variable Z in INVOKER. As is the case with all local variables, the storage represented by Z is contained in the stack frame belonging to INVOKER's invocation. When a name interpreter resolved C's name, it placed the descriptor in a register. Call Microcode 1001 takes this descriptor, converts it to a pointer, and stores the pointer above Header 1104.

Since the FP ABP points to the location following the last pointer to an actual argument, Call Microcode 1001 can now calculate that location, convert it into a descriptor, and place it in a FU Register 1004 reserved for FP. The next step is providing storage for EXAMPLE's local variables. EXAMPLE's procedure Object 1006 contains the size of the storage required for the local variables, so Call Microcode 1001 obtains this information from procedure Object 1006 and adds that much storage to Frame 1102. Using the new value of FP and the information contained in the Name Table Entries for the local data, Name Interpreter 715 can now construct descriptors for the local data. For example, A's entry in Name Table specified that it was offset 32 bits from FP, and was 32 bits long. Thus, its storage falls between the storage for X and B in Figure 11.

9. Memory References and the Virtual Memory Managment System (Fig. 12, 13)

As already explained, a logical descriptor contains an AON field, an offset field, and a length field. Fig. 12 illustrates a Physical Descriptor. Physical Descriptor 1202 contains a Frame Number (FN) field, a Displacement (D) field, and a Length (L) field. Together, the Frame Number field and the Displacement field specify the location in MEM 112 containing the data, and the Length field specifies the length of the data.

As is clear from the above, the virtual memory management system must translate the AON-offset location contained in a logical descriptor 1204 into a Frame Number-Displacement location. It does so by associating logical pages with MEM 112 frames. (N.B: MEM 112 frames are not to be confused with stack frames). Fig. 13, illustrates how Macrostack 502 Object 1302 is divided into Logical Pages 1304 in secondary memory and how Logical Pages 1304 are moved onto Frames 1306 in MEM 112. A Frame 1306 is a fixed-size, contiguous area of MEM 112. When the virtual memory management system brings data into MEM 112, it does so in frame-sized chunks called Logical Pages 1308. Thus, from the virtual memory system's point of view, each object is divided into Logical Pages 1308 and the address of data on a page consists of the AON of the data's Object, the number of pages in the object, and its displacement on the page. In Fig. 13, the location of the local variable B of EXAMPLE is shown as it is defined by the virtual memory system. B's location is a UID and an offset, or, inside JP 114, an AON and an offset. As defined by the virtual memory system, B's location is the AON, the page number 1308, and a displacement within the page. When a process references the variable B, the virtual memory management system moves all of Logical Page 1308 into a MEM 112 Frame 1306. B's displacement remains the same, and the virtual memory system translates its Logical Page Number 1308 into the number of Frame 1306 in MEM 112 which contains the page.

The virtual memory management system must therefore perform two kinds of translations: (1) AON-offset addresses into AON-page number-displacement addresses, and (2) AON-page number into a frame number.

10. Access Control (Fig. 14)

Each time a reference is made to an Object, KOS 706, 710 checks whether the reference is legal. The following discusson will first present the logical structure of access control in CS 101, and then discuss the microcode and devices which implement it.

CS 101 defines access in terms of subjects, modes of access, and Object size. A process may reference a data item located in an Object if three conditions hold:

1) If the process's subject has access to the Object.

2) If the modes of access specified for the subject include those required to perform the intended operation.

3) If the data item is completely contained in the Object, i.e., if the data item's length added to the data item's offset do not exceed the number of bits in the Object.

The subjects which have access to an Object and the kinds of access they have to the Object are specified by a data structure associated with the Object called the Access Control List (ACL). An Object's size is one of its attributes. Neither an Object's size nor its ACL is contained in the Object. Both are contained in system tables, and are accessible by means of the Object's UID.

Fig. 14 shows the logical structure of access control in CS 101. Subject 1408 has four components: Principal 1404, Process 1405, Domain 1406, and Tag 1407. Tag 1407 is not implemented in a present embodiment of CS 101, so the following description will deal only with principal 1404, Process 1405, and Domain 1406.

Principal 1404 specifies a user for which the process which is making the reference was created;
Process 1405 specifies the process which is making the reference; and,
Domain 1406 specifies the domain of execution of the procedure which the process is executing when it makes the reference.

Each component of the Subject 1408 is represented by a UID. If the UID is a null UID, that component of the subject does not affect access checking. Non-null UIDs are the UIDs of Objects that contain information about the subject components. Principal Object 1404 contains identification and accounting information regarding system users, Process Object 1405 contains process management information, and Domain Object 1406 contains information about per-domain error handlers.

There may be three modes of accessing an Object 1410: read, write, and execute. Read and write are self-explanatory; execute is access which allows a subject to execute instructions contained in the Object.

Access Control Lists (ACLs) 1412 are made up of Entries 1414. Each entry two components: Subject Template 1416 and Mode Specifier 1418. Subject Template 1416 specifies a group of subjects that may reference the Object and Mode Specifier 1418 specifies the kinds of access these subjects may have to the Object. Logically speaking, ACL 1412 is checked each time a process references an Object 1410. The reference may succeed only if the process's current Subject 1408 is one of those on Object 1410's ACL 1412 and if the modes in the ACL Entry 1414 for the Subject 1408 allow the kind of access the process wishes to make.

11. Virtual Processors and Virtual Processor Swapping (Fig. 15)

As previously mentioned, the execution of a program by a Process 610 cannot take place unless EOS 704 has bound the Process 610 to a Virtual Processor 612. Physical execution of the Process 610 takes place only while the process's Virtual Processor 612 is bound to JP 114. The following discussion deals with the data bases belonging to a Virtual Processor 612 and the means by which a Virtual Processor 612 is bound to and removed from JP 114.

Fig. 15 illustrates the devices and tables which KOS 706, 710 uses to implement Virtual Processors 612. FU 120 WCS contains KOS Microcode 706 for binding Virtual Processors 612 to JP 114 and removing them from JP 114. Timers 1502 and Interrupt Line 1504 are hardware devices which produce signals that cause the invocation of KOS Microcode 706. Timers 1502 contains two timing devices: Interval Timer 1506, which may be set by KOS 706, 710 to signal when a certain time is reached, and Egg Timer 1508, which guarantees that there is a maximum time interval for which a Virtual processor 612 can be bound to JP 114 before it invokes KOS Microcode 706. Interrupt Line 1504 becomes active when JP 114 receives a message from IOS 116, for example when IOS 116 has finished loading a logical page into MEM 112.

FU 120 Registers 508 contain state belonging to the Virtual Processor 612 currently bound to JP 114. Here, this Virtual Processor 612 is called Virtual Processor A. In addition, Registers 508 contain registers reserved for the execution of VP Swapping Microcode 1510. ALU 1942 (part of FU 120) is used for the descriptor-to-pointer and pointer-to-descriptor transformations required when one Virtual Processor 612 is unbound from JP 114 and another bound to JP 114. MEM 112 contains data bases for Virtual Processors 612 and data bases used by KOS 706, 710 to manage Virtual Processors 612. KOS 706, 710 provides a fixed number of Virtual Processors 612 for CS 101. Each Virtual Processor 612 is represented by a Virtual Processor State Block (VPSB) 614. Each VPSB 614 contains information used by KOS 706, 710 to manage the Virtual Processor 612, and in addition contains information associating the Virtual Processor 612 with a process. Fig. 15 shows two VPSBs 614, one belonging to Virtual Processor 612A, and another belonging to Virtual Processor 612B, which will replace Virtual Processor 612A on JP 114. The VPSBs 614 are contained in VPSB Array 1512. The index of a VPSB 614 in VPSB Array 1512 is Virtual Processor Number 1514 belonging to the Virtual Processor 612 represented by a VPSB 614. Virtual Processor Lists 1516 are lists which KOS 706, 710 uses to manage Virtual Processors 612. If a Virtual Processor 612 is able to execute, its Virtual Processor Number 1514 is on a list called the Runnable List; Virtual Processors 612 which cannot run are on other lists, depending on the reason why they cannot run. It is assumed that Virtual Processor 612B's Virtual Processor Number 1514 is the first one on the Runnable List.

When a process is bound to a Virtual Procesor 612, the Virtual Processor Number 1514 is copied into the process's Process Object 901 and the AONs of the process's Process Object 901 and stacks are copied into the Virtual Processor 612's VPSB 614. (AONs are used because a process's stacks are wired active as long as the process is bound to a Virtual Processor 612). Binding is carried out by KOS 706, 710 at the request of EOS 704. In Fig. 15, two Secure Stack Objects 906 are shown, one belonging to the process to which Virtual Processor 612A is bound, and one belonging to that to which Virtual Processor 612B is bound.

Having described certain overall operating features of CS 101, a present implementation of CS 101's structure will be described further next below.

E. CS 101 Structural Implementation (Figs. 16, 17, 18, 19, 20)

1. (IOS) 116 (Figs. 16, 17)

Referring to Fig. 16, a partial block diagram of IOS 116 is shown. Major elements of IOS 116 include an ECLIPSE® Burst Multiplexer Channel (BMC) 1614 and a NOVA® Data Channel (NDC) 1616, an IO Controller (IOC) 1618 and a Data Mover (DM) 1610. IOS 116's data channel devices, for example BMC 1614 and NDC 1616, comprise IOS 116's interface to the outside world. Information and addresses are received from

external devices, such as disk drives, communications modes, or other computer systems, by IOS 116's data channel devices and are transferred to DM 1610 (described below) to be written into MEM 112. Similarly, information read from MEM 112 is provided through DM 1610 to IOS 116's data channel devices and thus to the above described external devices. These external devices are a part of CS 101's addressable memory space and may be addressed through UID addresses.

IOC 1618 is a general purpose CPU, for example an ECLIPSE® computer available from Data General Corporation. A primary function of IOC 1618 is control of data transfer through IOS116. In addition, IOC 1618 generates individual Maps for each data channel device for translating external device addresses into physical addresses within MEM 112. As indicated in Fig. 16, each data channel device contains an individual Address Translation Map (MAP) 1632 and 1636. This allows IOS 116 to assign individual areas of MEM 112's physical address space to each data channel device. This feature provides protection against one data channel device writing into or reading from information belonging to another data channel device. In addition, IOC 1618 may generate overlapping address translation Maps for two or more data channel devices to allow these data channel devices to share a common area of MEM 112 physical address space.

Data transfer between IOS 116's data channel devices and MEM 112 is through DM 1610, which includes a Buffer memory (BUF) 1641. BUF 1641 allows MEM 112 and IOS 116 to operate asychronously. DM 1610 also includes a Ring Grant Generator (RGG) 1644 which controls access of various data channel devices to MEM 112. RGG 1644 is designed to be flexible in apportioning access to MEM 112 among IOS 116's data channel devices as loads carried by various data channel devices varies. In addition, RGG 1644 insures that no one, or group, of data channel devices may monopolize access to MEM 112.

Referring to Fig. 17, a diagramic representation of RGG 1644's operation is shown. As described further in a following description, RGG 1644 may be regarded as a commutator scanning a number of ports which are assigned to various channel devices. For example, ports A, C, E, and G may be assigned to a BMC 1614, ports B and F to a NDC 1616, and ports D and H to another data channel device. RGG 1644 will scan each of these ports in turn and, if the data channel device associated with a particular port is requesting access to MEM 112, will grant access to MEM 112 to that data channel device. If no request is present at a given port, RGG 1644 will continue immediately to the next port. Each data channel device assigned one or more ports is thereby insured opportunity of access to MEM 112. Unused ports, for example indicating data channel devices which are not presently engaged in information transfer, are effectively skipped over so that access to MEM 112 is dynamically modified according to the information transfer loads of the various data channel devices. RGG 1644's ports may be reassigned among IOS 116's various data channel devices as required to suit the needs of a particular CS 101 system. If, for example, a particular CS 101 utilizes NDC 1616 more than a BMC 1614, that CS 101's NDC 1616 may be assigned more ports while that CS 101's BMC 1614 is assigned fewer ports.

2. Memory (MEM) 112 (Fig. 18)

Referring to Fig. 18, a partial block diagram of MEM 112 is shown. Major elements of MEM 112 are Main Store Bank (MSB) 1810, a Bank Controller (BC) 1814, a Memory Cache (MC) 1816, a Field Interface Unit (FIU) 1820, and Memory Interface Controller (MIC) 1822. Interconnections of these elements with input and output buses of MEM 112 to IOS 116 and JP 114 are indicated.

MEM 112 is an intelligent, prioritizing memory having a single port to IOS 116, comprised of IOM Bus 130, MIO Bus 129, and IOMC Bus 131, and dual ports to JP 114. A first JP 114 port is comprised of MOD Bus 140 and PD Bus 146, and a second port is comprised of JPD Bus 142 and PD Bus 146. In general, all data transfers from and to MEM 112 by IOS 116 and JP 114 are of single, 32 bit words; IOM Bus 130, MIO Bus 129, MOD Bus 140, and JPD Bus 142 are each 32 bits wide. CS 101, however, is a variable word length machine wherein the actual physical width of data buses are not apparent to a user. For example, a Name in a user's program may refer to an operand containing 97 bits of data. To the user, that 97 bit data item will appear to be read from MEM 112 to JP 114 in a single operation. In actuality, JP 114 will read that operand from MEM 112 in a series of read operations referred to as a string transfer. In this example, the string transfer will comprise three 32 bit read transfers and one single bit read transfer. The final single bit transfer, containing a single data bit, will be of a 32 bit word wherein one bit is data and 31 bits are fill. Write operations to MEM 112 may be performed in the same manner. If a single read or write request to MEM 112 specifies a data item of less than 32 bits of data, that transfer will be accomplished in the same manner as the final transfer described above. That is, a single 32 bit word will be transferred wherein non-data bits are fill bits.

Bulk data storage in MEM 112 is provided in MSB 1810, which is comprised of one or more Memory Array cards (MAs) 1812. The data path into and out of MA 1812 is through BC 1814, which performs all control and timing functions for MAs 1812. BC 1814's functions include addressing, transfer of data, controlling whether a read or write operation is performed, refresh, sniffing, and error correction code operations. All read and write operations from and to MAs 1812 through BC 1814 are in blocks of four 32 bit words.

The various MAs 1812 comprising MSB 1810 need not be of the same data storage capacity. For example, certain MAs 1812 may have a capacity of 256 kilobytes while other MAs 1812 may have a capacity of 512 kilobytes. Addressing of the MAs 1812 in MSB 1810 is automatically adapted to various MA 1812

configurations. As indicated in Fig. 18, each MA 1812 contains an address circuit (A) which receives an input from the next lower MA 1812 indicating the highest address in that next lower MA 1812. The A circuit on an MA 1812 also receives an input from that MA 1812 indicating the total address space of that MA 1812. The A circuit of that MA 1812 adds the highest address input from next lower MA 1812 to its own input representing its own capacity and generates an output to the next MA 1812 indicating its own highest address. All MAs 1812 of MSB 1810 are addressed in parallel by BC 1814. Each MA 1812 compares such addresses to its input from the next lower MA 1812, representing highest address of that next lower MA 1812, and its own output, representing its own highest address, to determine whether a particular address provided by BC 1814 lies within the range of addresses contained within that particular MA 1812. The particular MA 1812 whose address space includes that address will then respond by accepting the read or write request from BC 1814.

MC 1816 is the data path for transfer of data between BC 1814 and IOS 116 and JP 114. MC 1816 contains a high speed cache storing data from MSB 1810 which is currently being utilized by either IOS 116 or JP 114. MSB 1810 thereby provides MEM 112 with a large storage capacity while MC 1816 provides the appearance of a high speed memory. In addition to operating as a cache, MC 1816 includes a bypass write path which allows IOS 116 to write blocks of four 32 bit words directly into MSB 1810 through BC 1814. In addition, MC 1816 includes a cache write-back path which allows data to be transferred out of MC 1816's cache and stored while further data is transferred into MC 1816's cache. Displaced data from MC 1816's cache may then be written back into MSB 1810 at a later, more convenient time. This write-back path enhances speed of operation of MC 1816 by avoiding delays incurred by transferring data from MC 1816 to MSB 1810 before new data may be written into MC 1816.

MEM 112's FIU 1820 allows manipulation of data formats in writes to and reads from MEM 112 by both JP 114 and IOS 116. For example, FIU 1820 may convert unpacked decimal data to packed decimal data, and vice versa. In addition, FIU 1820 allows MEM 112 to operate as a bit addressable memory. For example, as described all data transfers to and from MEM 112 are of 32 bit words. If a data transfer of less than 32 bits is required, the 32 bit word containing those data bits may be read from MC 1816 to FIU 1820 and therein manipulated to extract the required data bits. FIU 1820 then generates a 32 bit word containing those required data bits, plus fill bits, and provides that new 32 bit word to JP 114 or IOS 116. When writing into MEM 112 from IOS 116 through FIU 1820, data is transferred onto IOM Bus 130, read into FIU 1820, operated upon, transferred onto MOD Bus 140, and transferred from MOD Bus 140 to MC 1816. In read operations from MEM 112 to IOS 116, data is transferred from MC 1816 to MOD Bus 140, written into FIU 1820 and operated upon, and transferred onto MIO Bus 129 to IOS 116. In a data read from MEM 112 to JP 114, data is transferred from MC 1816 onto MOD Bus 140, transferred into FIU 1820 and operated upon, and transferred again onto MOD Bus 140 to JP 114. In write operations from JP 114 to MEM 112, data on JPD Bus 142 is transferred into FIU 1820 and operated upon, and is then transferred onto MOD Bus 140 to MC 1816. MOD Bus 140 is thereby utilized as an MEM 112 internal bus for FIU 1820 operations.

Finally, MIC 1822 provides primary control of BC 1814, MC 1816, and FIU 1820. MIC 1822 receives control inputs from and provides control outputs to PD Bus 146 and IOMC Bus 131. MIC 1822 contains primary microcode control for MEM 112, but BC 1814, MC 1816, and FIU 1820 each include internal microcode control. Independent, internal microcode controls allow BC 1814, MC 1816, and FIU 1820 to operate independently of MIC 1822 after their operations have been initiated by MIC 1822. This allows BC 1814 and MSB 1810, MC 1816, and FIU 1820 to operate independently and asynchronously. Efficiency and speed of operation of MEM 112 are thereby enhanced by allowing pipelining of MEM 112 operations.

3. Fetch Unit (FU) 120 (Fig. 19)

A primary function of FU 120 is to execute SINs. In doing so, FU 120 fetches instructions and data (SOPs and Names) from MEM 112, returns results of operations to MEM 112, directs operation of EU 122, executes instructions of user's programs, and performs the various functions of CS 101's operating systems. As part of these functions, FU 120 generates and manipulates logical addresses and descriptors and is capable of operating as a general purpose CPU.

Referring to Fig. 19, a major element of FU 120 is the Descriptor Processor (DESP) 1910. DESP 1910 includes General Register File (GRF) 506. GRF 506 is a large register array divided vertically into three parts which are addressed in parallel. A first part, AONGRF 1932, stores AON fields of logical addresses and descriptors. A second part, OFFGRF 1934, stores offset fields of logical addresses and descriptors and is utilized as a 32 bit wide general register array. A third portion GRF 506, LENGRF 1936, is a 32 bit wide register array for storing length fields of logical descriptors and as a general register for storing data. Primary data path from MEM 112 to FU 120 is through MOD Bus 140, which provides inputs to OFFGRF 1934. As indicated in Fig. 19, data may be transferred from OFFGRF 1934 to inputs of AONGRF 1932 and LENGRF 1936 through various interconnections. Similarly, outputs from LENGRF 1936 and AONGRF 1932 may be transferred to inputs of AONGRF 1932, OFFGRF 1934, and LENGRF 1936.

Output of OFFGRF 1934 is connected to inputs of DESP 1910's Arithmetic and Logic Unit (ALU) 1942. ALU 1942 is a general purpose 32 bit ALU which may be used in generating and manipulating logical addresses and descriptors, as distinct from general purpose arithmetic and logic operands performed by MUX 1940. Output of ALU 1942 is connected to JPD Bus 142 to allow results of arithmetic and logic operations to be transferred to MEM 112 or EU 122.

Also connected from output of OFFGRF 1934 is Descriptor Multiplexer (MUX) 1940. An output of MUX 1940 is provided to an input of ALU 1942. MUX 1940 is a 32 bit ALU, including an accumulator, for data manipulation operations. MUX 1940, together with ALU 1942, allows DESP 1910 to perform 32 bit arithmetic and logic operations. MUX 1940 and ALU 1942 may allow arithmetic and logic operations upon operands of greater than 32 bits by performing successive operations upon successive 32 bit words of larger operands.

Logical descriptors or addresses generated or provided by DESP 1910, are provided to Logical Descriptor (LD) Bus 1902. LD Bus 1902 in turn is connected to an input of Address Translation Unit (ATU) 1928. ATU 1928 is a cache mechanism for converting logical descriptors to MEM 112 physical descriptors.

LD Bus 1902 is also connected to write input of Name Cache (NC) 1926. NC 1926 is a cache mechanism for storing logical descriptors corresponding to operand Names currently being used in user's programs. As previously described, Name Table Entries corresponding to operands currently being used in user's programs are stored in MEM 112. Certain Name Table Entries for operands of a user's program currently being executed are transferred from those Name Tables in MEM 112 to FU 120 and are therein evaluated to generate corresponding logical descriptors. These logical descriptors are then stored in NC 1926. As will be described further below, the instruction stream of a user's program is provided to FU 120's Instruction Buffer (IB) 1962 through MOD Bus 140. FU 120's Parser (P) 1964 separates out, or parses, Names from IB 1962 and provides those Names as address inputs to NC 1924. NC 1924 in turn provides logical descriptor outputs to LD Bus 1902, and thus to input of ATU 1928. NC 1926 input from LD Bus 1902 allows logical descriptors resulting from evaluation of Name Table Entries to be written into NC 1926. FU 120's Protections Cache (PC) 1934 is a cache mechanism having an input connected from LD Bus 1902 and providing information, as described further below, regarding protection aspects of references to data in MEM 112 by user's programs. NC 1926, ATU 1928, and PC 1934 are thereby acceleration mechanisms of, respectively, CS 101's Namespace addressing, logical to physical address structure, and protection mechanism.

Referring again to DESP 1910, DESP 1910 includes BIAS 1952, connected from output of LENGRF 1936. As previously described, operands containing more than 32 data bits are transferred beteen MEM 112 and JP 114 by means of string transfers. In order to perform string transfers, it is necessary for FU 120 to generate a corresponding succession of logical descriptors wherein length fields of those logical descriptors is no greater than 5 bits, that is, specify lengths of no greater than 32 data bits.

A logical descriptor describing a data item to be transferred by means of a string transfer will be stored in GRF 506. AON field of the logical descriptor will reside in AONGRF 1932, O field in OFFGRF 1934, and L field in LENGRF 1936. At each successive transfer of a 32 bit word in the string transfer, O field of that original logical descriptor will be incremented by the number of data bits transferred while L field will be accordingly decremented. The logical descriptor residing in GRF 506 will thereby describe, upon each successsive transfer of the string transfer, that portion of the data item yet to be transferred. O field in OFFGRF 1934 will indicate increasingly larger offsets into that data item, while L field will indicate successively shorter lengths. AON and O fields of the logical descriptor in GRF 506 may be utilized directly as AON and O fields of the successive logical descriptors of the string transfer. L field of the logical descriptor residing in LENGRF 1936, however, may not be so used as L fields of the successive string transfer logical descriptors as this L field indicates remaining length of data item yet to be transferred. Instead, BIAS 1952 generates the 5 bit L fields of successive string transfer logical descriptors while correspondingly decrementing L field of the logical descriptor in LENGRF 1936. During each transfer, BIAS 1952 generates L field of the *next* string transfer logical descriptor while concurrently providing L field of the *current* string transfer logical descriptor. By doing so, BIAS 1952 thereby increases speed of execution of string transfers by performing pipelined L field operations. BIAS 1952 thereby allows CS 101 to appear to the user to be a variable word length machine by automatically performing string transfers. This mechanism is used for transfer of any data item greater than 32 bits, for example double precision floating point numbers.

Finally, FU 120 includes microcode circuitry for controlling all FU 120 operations described above. In particular, FU 120 includes a microinstruction sequence control store (mC) 1920 storing sequences of microinstructions for controlling step by step execution of all FU 120 operations. In general, these FU 120 operations fall into two classes. A first class includes those microinstruction sequences directly concerned with executing the SOPs of user's programs. The second class includes microinstruction sequences concerned with CS 101's operating systems, including and certain automatic, internal FU 120 functions such as evaluation of Name Table Entries.

As previously described, CS 101 is a multiple S-Language machine. For example, mC 1920 may contain microinstruction sequences for executing user's SOPs in at least four different Dialects. mC 1920 is comprised of a writeable control store and sets of microinstruction sequences for various Dialects may be transferred into and out of mC 1920 as required for execution of various user's programs. By storing sets of microinstruction sequences for more than one Dialect in mC 1920, it is possible for user's programs to be written in a mixture of user languages. For example, a particular user's program may be written primarily in FORTRAN but may call certain COBOL routines. These COBOL routines will be correspondingly translated into COBOL dialect SOPs and executed by COBOL microinstruction sequences stored in mC 1920.

The instruction stream provided to FU 120 from MEM 112 has been previously described with

reference to Fig. 3. SOPs and Names of this instruction stream are transferred from MOD Bus 140 into IB 1962 as they are provided from MEM 112. IB 1962 includes two 32 bit (one word) registers. IB 1962 also includes prefetch circuitry for reading for SOPs and Names of the instruction stream from MEM 112 in such a manner that IB 1962 shall always contain at least one SOPs or Name. FU 120 includes (P) 1964 which reads and separates, or parses, SOPs and Names from IB 1962. As previously described, P 1964 provides those Names to NC 1926, which accordingly provides logical descriptors to ATU 1928 so as to read the corresponding operands from MEM 112.

SOPs parsed by P 1964 are provided as inputs to Fetch Unit Dispatch Table (FUDT) 1904 and Execute Unit Dispatch Table (EUDT) 1966. Referring first to FUDT 1904, FUDT 1904 is effectively a table for translating SOPs to starting addresses in mC 1912 of corresponding microinstruction sequences. This intermediate translation of SOPs to mC 1912 addresses allows efficient packing of microinstruction sequences within mC 1912. That is, certain microinstruction sequences may be common to two or more S-Language Dialects. Such microinstruction sequences may therefore be written into mC 1912 once andy may be referred to by different SOPs of different S-Language Dialects.

EUDT 1966 performs a similar function with respect to EU 122. As will be described below, EU 122 contains a mC, similar to mC 1912, which is addressed through EUDT 1966 by SOPs specifying EU 122 operations. In addition, FU 120 may provide such addresses mC 1912 to initiate EU 122 operations as required to assist certain FU 120 operations. Examples of such operations which may be requested by FU 120 include calculations required in evaluating Name Table Entries to provide logical descriptors to be loaded into NC 1926.

Associated with both FUDT 1904 and EUDT 1966 are Dialect (D) registers 1905 and 1967. D registers 1905 and 1967 store information indicating the particular S-Language Dialect currently being utilized in execution of a user's program. Outputs of D registers 1905 and 1967 are utilized as part of the address inputs to mC 1912 and EU 122's mC.

4. Execute Unit (EU) 122 (Fig. 20)

As previously described, EU 122 is an arithmetic and logic unit provided to relieve FU 120 of certain arithmetic operations. EU 122 is capable of performing addition, subtraction, multiplication, and division operations on integer, packed and unpacked decimal, and single and double precision floating operands. EU 122 is an independently operating microcode controlled machine including Microcode Control (mC) 2010 which, as described above, is addressed by EUDT 1966 to initiate EU 122 operations. mC 2010 also includes logic for handling mutual interrupts between FU 120 and EU 122. That is, FU 120 may interrupt current EU 122 operations to call upon EU 122 to assist an FU 120 operation. For example, FU 120 may interrupt an arithmetic operation currently being executed by EU 122 to call upon EU 122 to assist in generating a logical descriptor from a Name Table Entry.

Similarly, EU 122 may interrupt current FU 120 operations when EU 122 requires FU 120 assistance in executing a current arithmetic operation. For example, EU 122 may interrupt a current FU 120 operation if EU 122 receives an instruction and operands requiring EU 122 to perform a divide by zero.

Referring to Fig. 20, a partial block diagram of EU 122 is shown. EU 122 includes two arithmetic and logic units. A first arithmetic and logic unit (MULT) 2014 is utilized to perform addition, subtraction, multiplication, and division operations upon integer and decimal operands, and upon mantissa fields of single and double precision floating point operands. Second ALU (EXP) 2016 is utilized to perform operations upon single and double precision floating point operand exponent fields in parallel with operations performed upon floating point mantissa fields by MULT 2014. Both MULT 2014 and EXP 2016 include an arithmetic and logic unit, respectively MALU 2074 and EXPALU 2084. MULT 2014 and EXP 2016 also include register files, respectively MRF 2050 and ERF 2080, which operate and are addressed in parallel in a manner similar to AONGRF 1932, OFFGRF 1984 and LENGRF 1936.

Operands for EU 122 to operate upon are provided from MEM 112 through MOD Bus 140 and are transferred into Operand Buffer (OPB) 2022. In addition to serving as an input buffer, OPB 2022 performs certain data format manipulation operations to transform input operands into formats most efficiently operated with by EU 122. In particular, EU 122 and MULT 2014 may be designed to operate efficiently with packed decimal operands. OPB 2022 may transform unpacked decimal operands into packed decimal operands. Unpacked decimal operands are in the form of ASCII characters wherein four bits of each characters are binary codes specifying a decimal value between zero and nine. Other bits of each character are referred to as zone fields and in general contain information identifying particular ASCII characters. For example, zone field bits may specify whether a particular ASCII character is a number, a letter, or punctuation. Packed decimal operands are comprised of a series of four bit fields wherein each field contains a binary number specifying a decimal value of between zero and nine. OPB 2022 converts unpacked decimal to packed decimal operands by extracting zone field bits and packing the four numeric value bits of each character into the four bit fields of a packed decimal number.

EU 122 is also capable of transforming the results of arithmetic operands, for example in packed decimal format, into unpacked decimal format for transfer back to MEM 112 or FU 120. In this case, a packed decimal result appearing at output of MALU 2074 is written into MRF 2050 through a multiplexer, not shown in Fig. 20, which transforms the four bit numeric code fields of the packed decimal results into corresponding bits of unpacked decimal operand characters, and forces blanks into the zone field bits of

those unpacked decimal characters. The results of this operation are then read from MRF 2050 to MALU 2074 and zone field bits for those unpacked decimal characters are read from Constant Store (CST) 2060 to MALU 2074. These inputs from MRF 2050 and CST 2060 are added by MALU 2074 to generate final result outputs in unpacked decimal format. These final results may then be transferred onto JPD Bus 142 through Output Multiplexer (OM) 2024.

Considering first floating point operations, in addition or subtraction of floating point operands it is necessary to equalize the values of the floating point operand exponent fields. This is referred to as prealignment. In floating point operations, exponent fields of the two operands are transferred into EXPALU 2034 and compared to determine the difference between exponent fields. An output representing difference between exponent fields is provided from EXPALU 2034 to an input of floating point control (FPC) 2002. FPC 2002 in turn provides control outputs to MALU 2074, which has received the mantissa fields of the two operands. MALU 2074, operating under direction of FPC 2002, accordingly right or left shifts one operand's mantissa field to effectively align that operand's exponent field with the other operand's exponent field. Addition or subtraction of the operand's mantissa fields may then proceed.

EXPALU 2034 also performs addition or subtraction of floating point operand exponent fields in multiplication or division operations, while MALU 2074 performs multiplication and division of the operand mantissa fields. Multiplication and division of floating point operand mantissa fields by MALU 2074 is performed by successive shifting of one operand, corresponding generation of partial products of the other operand, and successive addition and subtraction of those partial products.

Finally, EU 122 performs normalization of the results of floating point operand operations by left shifting of a final result's mantissa field to eliminate zeros in the most significant characters of the final result mantissa field, and corresponding shifting of the final result exponent fields. Normalization of floating point operation results is controlled by FPC 2002. FPC 2002 examines an unnormalized floating point result output of MALU 2074 to detect which, if any, of the most significant characters of that results contain zeros. FPC 2002 then accordingly provides control outputs to EXPALU 2034 and MALU 2074 to correspondingly shift the exponent and mantissa fields of those results so as to eliminate leading character zeros from the mantissa field. Normalized mantissa and exponent fields of floating point results may then be transferred from MALU 2074 and EXPALU 2034 to JPD Bus 142 through OM 2024.

As described above, EU 122 also performs addition, subtraction, multiplication, and division operations on operands. In this respect, EU 122 uses a leading zero detector in FPC 2002 in efficiently performing multiplication and division operations. FPC 2002's leading zero detector examines the characters or bits of two operands to be multiplied or divided, starting from the highest, to determine which, if any, contain zeros so as not to require a multiplication or division operation. FPC 2002 accordingly left shifts the operands to effectively eliminate those characters or bits, thus reducing the number of operations to multiply or divide the operands and accordingly reducing the time required to operate upon the operands.

Finally, EU 122 utilizes a unique method, with associated hardware, for performing arithmetic operations on decimal operands by utilizing circuitry which is otherwise conventionally used only to perform operations upon floating point operands. As described above, MULT 2074 is designed to operate with packed decimal operands, that is operands in the form of consecutive blocks of four bits wherein each block of four bits contains a binary code representing numeric values of between zero and nine. Floating point operands are similarly in the form of consecutive blocks of four bits. Each block of four bits in a floating point operand, however, contains a binary number representing a hexadecimal value of between zero and fifteen. As an initial step in operating with packed decimal operands, those operands are loaded, one at a time, into MALU 2074 and, with each such operand, a number comprised of all hexadecimal sixes is loaded into MALU 2074 from CST 2060. This CST 2060 number is added to each packed decimal operand to effectively convert those packed decimal operands into hexadecimal operands wherein the four bit blocks contain numeric values in the range of six to fifteen, rather than in the original range of zero to nine. MULT 2014 then performs arithmetic operation upon those transformed operands, and in doing so detects and saves information regarding which four bit characters of those operands have resulted in generation of carries during the arithmetic operations. In a final step, the intermediate result resulting from completion of those arithmetic operations upon those transformed operands are reconverted to packed decimal format by subtraction of hexadecimal sixes from those characters for which carries have been generated. Effectively, EU 122 converts packed decimal operands into "Excess Six" operands, performs arithmetic operations upon those "Excess Six" operands, and reconverts "Excess Six" results of those operations back into packed decimal format.

Finally, as previously descibed FU 120 controls transfer of arithmetic results from EU 122 to MEM 112. In doing so, FU 120 generates a logical descriptor describing the size of MEM 112 address space, or "container", that result is to be transferred into. In certain arithmetic operations, for example integer operations, an arithmetic result may be larger than anticipated and may contain more bits than the MEM 112 "container". Container Size Check Circuit (CSC) 2052 compares actual size of arithmetic results and L fields of MEM 112 "container" logical descriptors. CSC 2052 generates an output indicating whether an MEM 112 "container" is smaller than an arithmetic result.

Having briefly described certain features of CS 101 structure and operation in the above overview, these and other features of CS 101 will be described in further detail next below in a more detailed

introduction of CS 101 structure and operation. Then, in further descriptions, these and other features of CS 101 structure and operation will be described in depth.

1. Introduction (Figs. 101—110)

A. General Structure and Operation (Fig 101)

a. General Structure

Referring to Fig. 101, a partial block diagram of Computer System (CS) 10110 is shown. Major elements of CS 10110 are Dual Port Memory (MEM) 10112, Job Processor (JP) 10114, Input/Output System (IOS) 10116, and Diagnostic Processor (DP) 10118. JP 10114 includes Fetch Unit (FU) 10120 and Execute Unit (EU) 10122.

Referring first to IOS 10116, IOS 10116 is interconnected with External Devices (ED) 10124 through Input/Output (I/O) Bus 10126. ED 10124 may include, for example, other computer systems, keyboard/display units, and disc drive memories. IOS 10116 is interconnected with Memory Input/Output (MIO) Port 10128 of MEM 10112 through Input/Output to Memory (IOM) Bus 10130 and Memory to Input/Output (MIO) Bus 10129, and with FU 10120 through I/O Job Processor (IOJP) Bus 10132.

DP 10118 is interconnected with, for example, external keyboard/CRT Display Unit (DU) 10134 through Diagnostic Processor Input/Output (DPIO) Bus 10136. DP 10118 is interconnected with IOS 10116, MEM 10112, FU 10120, and EU 10122 through Diagnostic Processor (DP) Bus 10138.

Memory to Job Processor (MJP) Port 10140 of Memory 10112 is interconnected with FU 10120 and EU 10122 through Job Processor Data (JPD) Bus 10142. An output of MJP 10140 is connected to inputs of FU 10120 and EU 10122 through Memory Output Data (MOD) Bus 10144. An output of FU 10120 is connected to an input of MJP 10140 through Physical Descriptor (PD) Bus 10146. FU 10120 and EU 10122 are interconnected through Fetch/Execute (F/E) Bus 10148.

b. General Operation

As will be discussed further below, IOS 10116 and MEM 10112 operate independently under general control of JP 10114 in executing multiple user's programs. In this regard, MEM 10112 is an intelligent, prioritizing memory having separate and independent ports MIO 10128 and MJP 10140 to IOS 10116 and JP 10114 respectively. MEM 10112 is the primary path for information transfer between External Devices 10124 (through IOS 10116) and JP 10114. MEM 10112 thus operates both as a buffer for receiving and storing various individual user's programs (e.g., data, instructions, and results of program execution) and as a main memory for JP 10114.

A primary function of IOS 10116 is as an input/output buffer between CS 10110 and ED 10124. Data and instructions are transferred from ED 10124 to IOS 10116 through I/O Bus 10126 in a manner and format compatible with ED 10124. IOS 10116 receives and stores this information, and manipulates the information into formats suitable for transfer into MEM 10112. IOS 10116 then indicates to MEM 10112 that new information is available for transfer into MEM 10112. Upon acknowledgement by MEM 10112, this information is transferred into MEM 10112 through IOM Bus 10130 and MIO Port 10128. MEM 10112 stores the information in selected portions of MEM 10112 physical address space. At this time, IOS 10116 notifies JP 10114 that new information is present in MEM 10112 by providing a "semaphore" signal to FU 10120 through IOJP Bus 10132. As will be described further below, CS 10110 manipulates the data and instructions stored in MEM 10112 into certain information structures used in executing user's programs. Among these structures are certain structures, discussed further below, which are used by CS 10110 in organizing and controlling flow and execution of user programs.

FU 10120 and EU 10122 are independently operating microcode controlled "machines" together comprising the CS 10110 micromachine for executing user's programs stored in MEM 10112. Among the principal functions of FU 10120 are: (1) fetching and interpreting instructions and data from MEM 10112 for use by FU 10120 and EU 10122; (2) organizing and controlling flow of user programs; (3) initiating EU 10122 operations; (4) performing arithmetic and logic operations on data; (5) controlling transfer of data from FU 10120 and EU 10122 to MEM 10112; and, (6) maintaining certain "stack" and "register" mechanisms, described below. FU 10120 "cache" mechanisms, also described below, are provided to enhance the speed of operation of JP 10114. These cache mechanisms are acceleration circuitry including, in part, high speed memories for storing copies of selected information stored in MEM 10112. The information stored in this acceleration circuitry is therefore more rapidly available to JP 10114. EU 10122 is an arithmetic unit capable of executing integer, decimal, or floating point arithmetic operations. The primary function of EU 10122 is to relieve FU 10120 from certain extensive arithmetic operations, thus enhancing the efficiency of CS 10110.

In general, operations in JP 10114 are executed on a memory to memory basis; data is read from MEM 10112, operated upon, and the results returned to MEM 10112. In this regard, certain stack and cache mechanisms in JP 10114 (described below) operate as extensions of MEM 10112 address space.

In operation, FU 10120 reads data and instructions from MEM 10112 by providing physical addresses to MEM 10112 by way of PA Bus 10146 and MJP Port 10140. The instructions and data are transferred to FU 10120 and EU 10122 by way of MJP Port 10140 and MOD Bus 10144. Instructions are interpreted by FU 10120 microcode circuitry, not shown in Fig. 101 but described below, and when necessary, microcode instructions are provided to EU 10122 from FU 10120's microcode control by way of F/E Bus 10148, or by way of JPD Bus 10142.

As stated above, FU 10120 and EU 10122 operate asynchronously with respect to each other's functions. A microinstruction from FU 10120 microcode circuitry to EU 10122 may initiate a selected operation of EU 10122. EU 10122 may then proceed to independently execute the selected operation. FU 10120 may proceed to concurrently execute other operations while EU 10122 is completing the selected arithmetic operation. At completion of the selected arithmetic operation, EU 10122 signals FU 10120 that the operation results are available by way of a "handshake" signal through F/E Bus 10148. FU 10120 may then receive the arithmetic operation results for further processing or, as discussed momentarily, may directly transfer the arithmetic operation results to MEM 10112. As described further below, an instruction buffer referred to as a "queue" between FU 10120 and EU 10122 allows FU 10120 to assign a sequence of arithmetic operations to be performed by EU 10122.

Information, such as results of executing an instruction, is written into MEM 10112 from FU 10120 or EU 10122 by way of JPD Bus 10142. FU 10120 provides a "physical write address" signal to MEM 10112 by way of PA Bus 10146 and MJP Port 10140. Concurrently, the information to be written into MEM 10112 is placed on JPD Bus 10142 and is subsequently written into MEM 10112 at the locations selected by the physical write address.

FU 10120 places a semaphore signal on IOJP Bus 10132 to signal to IOS 10116 that information, such as the results of executing a user's program, is available to be read out of CS 10110. IOS 10116 may then transfer the information from MEM 10112 to IOS 10116 by way of MIO Port 10128 and IOM Bus 10130. Information stored in IOS 10116 is then transferred to ED 10124 through I/O Bus 10126.

During execution of a user's program, certain information required by JP 10116 may not be available in MEM 10112. In such cases as further described in a following discussion, JP 10114 may write a request for information into MEM 10112 and notify IOS 10116, by way of IOJP Bus 10132, that such a request has been made. IOS 10116 will then read the request and transfer the desired information from ED 10124 into MEM 10112 through IOS 10116 in the manner described above. In such operations, IOS 10116 and JP 10114 operate together as a memory manager wherein the memory space addressable by JP 10114 is termed virtual memory space, and includes both MEM 10112 memory space and all external devices to which IOS 10116 has access.

As previously described, DP 10118 provides a second interface between Computer System 10110 and the external world by way of DPIO Bus 10136. DP 10118 allows DU 10134, for example a CRT and keyboard unit or a teletype, to perform all functions which are conventionally provided by a hard (i.e., switches and lights) console. For example, DP 10118 allows DU 10134 to exercise control of Computer System 10110 for such purposes as system initialization and start up, execution of diagnostic processes, and fault monitoring and identification. DP 10118 has read and write access to most memory and register portions within each of IOS 10116, MEM 10112, FU 10120, and EU 10122 by way of DP Bus 10138. Memories and registers in CS 10110 can therefore be directly loaded or initialized during system start up, and can be directly read or loaded with test and diagnostic signals for fault monitoring and identification. In addition, as described further below, microinstructions may be loaded into JP 10114's microcode circuitry at system start up or as required.

Having described the general structure and operation of Computer System 10110, certain features of Computer System 10110 will next be briefly described to aid in understanding the following, more detailed descriptions of these and other features of Computer System 10110.

c. Definition of Certain Terms

Certain terms are used relating to the structure and operation of CS 10110 throughout the following discussions. Certain of these terms will be discussed and defined first, to aid in understanding the following descriptions. Other terms will be introduced in the following descriptions as required.

A *procedure* is a sequence of operational steps, or instructions, to be executed to perform some operation. A procedure may include data to be operated upon in performing the operation.

A *program* is a static group of one or more procedures. In general, programs may be classified as user programs, utility programs, and operating system programs. A user program is a group of procedures generated by and private to one particular user of a group of users interfacing with CS 10110. Utility programs are commonly available to all users; for example, a compiler comprises of a set of procedures for compiling a user language program into an S-language program. Operating system programs are groups of procedures internal to CS 10110 for allocation and control of CS 10110 resources. Operating system programs also define interfaces within CS 10110. For example, as will be discussed further below all operands in a program are referred to by "NAME". An operating system program translates operand NAME into the physical locations of the operands in MEM 10112. The NAME translation program thus defines the interface between operand NAME (name space addresses) and MEM 10112 physical addresses.

A *process* is an independent locus of control passing through physical, logical or virtual address spaces, or, more particularly, a path of execution through a series of programs (i.e., procedures). A process will generally include a user program and data plus one or more utility programs (e.g., a compiler) and operating system programs necessary to execute the user program.

An *object* is a uniquely identifiable portion of "data space" accessible to CS 10110. An object may be regarded as a container for information and may contain data or procedure information or both. An object may contain for example, an entire program, or set of procedures, or a single bit of data. Objects need not

be contiguously located in the data space accessible to CS 10110, and the information contained in an object need not be contiguously located in that object.

A *domain* is a state of operation of CS 10110 for the purposes of CS 10110's protection mechanisms. Each domain is defined by a set of procedures having access to objects within that domain for their execution. Each object has a single domain of execution in which it is executed if it is a procedure object, or used, if it is a data object. CS 10110 is said to be operating in a particular domain if it is executing a procedure having that domain of execution. Each object may belong to one or more domains; an object belongs to a domain if a procedure executing in that domain has potential access to the object. CS 10110 may, for example have four domains: User domain, Data Base Management System (DBMS) domain, Extended Operating System (EOS) domain, and Kernel Operating System (KOS) domain. User domain is the domain of execution of all user provided procedures, such as user or utility procedures. DBMS domain is the domain of execution for operating system procedures for storing, retrieving, and handling data. EOS domain is the domain of execution of operating system procedures defining and forming the user level interface with CS 10110, such as procedures for controlling an executing files, processes, and I/O operations. KOS domain is the domain of execution of the low level, secure operating system which manages and controls CS 10110's physical resources. Other embodiments of CS 10110 may have fewer or more domains than those just described. For example, DBMS procedures may be incorporated into the EOS domain or EOS domain may be divided by incorporating the I/O procedures into an I/O domain. There is no hardware enforced limitation on the number of, of boundaries between, domains in CS 10110. Certain CS 10110 hardware functions and structures are, however, dependent upon domains.

A *subject* is defined, for purposes of CS 10110's protection mechanisms, as a combination of the current principle (user), the current process being executed, and the domain the process is currently being executed in. In addition to principle, process, and domain, which are identified by UIDs, subject may include a Tag, which is a user assigned identification code used where added security is required. For a given process, principle and process are constant but the domain is determined by the procedure currently being executed. A process's associated subject is therefore variable along the path of execution of the process.

Having discussed and defined the above terms, certain features of CS 10110 will next be briefly described.

d. Multi-Program Operation

CS 10110 is capable of concurrently executing two or more programs and selecting the sequence of execution of programs to make most effective use of CS 10110's resources. This is referred to as multiprogramming. In this regard, CS 10110 may temporarily suspend execution of one program, for example when a resource or certain information required for that program is not immediately available, and proceed to execute another program until the required resource or information becomes available. For example, particular information required by a first program may not be available in MEM 10112 when called for. JP 10114 may, as discussed further below, suspend execution of the first program, transfer a request for that information to IOS 10116, and proceed to call and execute a second program. IOS 10116 would fetch the requested information from ED 10124 and transfer it into MEM 10112. At some time after IOS 10116 notifies JP 10114 that the requested information is available in MEM 10112, JP 10114 could suspend execution of the second program and resume execution of the first program.

e. Multi-Language Operation

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language, such as COBOL or FORTRAN, is compiled into a corresponding Soft (S) Language program. That is, in terms of a conventional computer system, each user level language has a corresponding machine language, classically defined as an assembly language. In contrast to classical assembly languages, S-Languages are mid-level languages wherein each command in a user's high level language is replaced by, in general, two or three S-Language instructions, referred to as SINs. Certain SINs may be shared by two or more high level user languages. CS 10110, as further described in following discussions, provides a set, or dialect, of microcode instructions (S-Interpreters) for each S-Language. S-Interpreters interpret SINs and provide corresponding sequences of microinstructions for detailed control of CS 10110. CS 10110's instruction set and operation may therefore be tailored to each user's program, regardless of the particular user language, so as to most efficiently execute the user's program. Computer System 10110 may, for example, execute programs in both FORTRAN and COBOL with comparable efficiency. In addition, a user may write a program in more than one high level user language without loss of efficiency. For example, a user may write a portion of his program in COBOL, but may wish to write certain portions in FORTRAN. In such cases, the COBOL portions would be compiled into COBOL SINs and executed with the COBOL dialect S-Interpreter. The FORTRAN portions would be compiled into FORTRAN SINs and executed with a FORTRAN dialect S-Interpreter. The present embodiment of CS 10110 utilizes a uniform format for all SINs. This feature allows simpler S-Interpreter structures and increases efficiency of SIN interpretation because it is not necessary to provide means for interpreting each dialect individually.

f. Addressing Structure

Each object created for use in, or by operation of, a CS 10110 is permanently assigned a Unique Identifier (UID). An object's UID allows that object to be uniquely identified and located at any time, regardless of which particular CS 10110 it was created by or for or where it is subsequently located. Thus each time a new object is defined, a new and unique UID is allocated, much as social security numbers are allocated to individuals. A particular piece of information contained in an object may be located by a logical address comprising the object's UID, an offset from the start of the object of the first bit of the segment, and the length (number of bits) of the information segment. Data within an object may therefore be addressed on a bit granular basis. As will be described further in following discussions, UID's are used within a CS 10110 as logical addresses, and, for example, as pointers. Logically, all addresses and pointers in CS 10110 are UID addresses and pointers. As previously described and as described below, however, short, temporary unique identifiers, valid only within JP 10114 and referred to as Active Object Numbers are used within JP 10114 to reduce the width of address buses and amount of address information handled.

An object becomes active in CS 10110 when it is transferred from backing store CED 10124 to MEM 10112 for use in executing a process. At this time, each such object is assigned an Active Object Number (AON). AONs are short unique identifiers and are related to the object's UIDs through certain CS 10110 information structures described below. AONs are used only within JP 10114 and are used in JP 10114, in place of UIDs, to reduce the required width of JP 10114's address buses and the amount of address data handled in JP 10114. As with UID logical addresses, a piece of data in an object may be addressed through a bit granular AON logical address comprising the object's AON, an offset from the start of the object of the first bit of the piece, and the length of the piece.

The transfer of logical addresses, for example pointers, between MEM 10112 (UIDA) and JP 10114 (AONs) during execution of a process requires translations between UIDs and AONs. As will be described in a later discussion, this translation is accomplished, in part, through the information structures mentioned above. Similarly, translation of logical addresses to physical addresses in MEM 10112, to physically access information stored in MEM 10112, is accomplished through CS 10110 information structures relating AON logical addresses to MEM 10112 physical addresses.

Each operand appearing in a program is assigned a Name when the program is compiled. Thereafter, all references to the operands are through their assigned Names. As will be described in detail in a later discussion, CS 10110's addressing structure includes a mechanism for recognizing Names as they appear in an Instruction stream and Name Tables containing directions for resolving Names to AON logical addresses. AON logical addresses may then be evaluated, for example translated into a MEM 10112 physical address, to provide actual operands. The use of Names to identify operands in the instructions stream (process) (1) allows a complicated address to be replaced by a simple reference of uniform format; (2) does not require that an operation be directly defined by data type in the instruction stream; (3) allows repeated references to an operand to be made in an instruction stream by merely repeating the operand's Name; and, (4) allows partially completed Name to address translations to be stored in a cache to speed up operand references. The use of Names thereby substantially reduces the volume of information required in the instruction stream for operand references and increases CS 10110 speed and efficiency by performing operands references through a parallel operating, underlying mechanism.

Finally, CS 10110 address structure incorporates a set of Architectural Base Pointers (ABPs) for each process. ABPs provide an addressing framework to locate data and procedure information belonging to a process and are used, for example, in resolving Names to AON logical addresses.

g. Protection Mechanism

CS 10110's protection mechanism is constructed to prevent a user from (1) gaining access to or disrupting another user's process, including data, and (2) interfering with or otherwise subverting the operation of CS 10110. Access rights to each particular active object are dynamically granted as a function of the currently active subject. A subject is defined by a combination of the current principle (user), the current process being executed, and the domain in which the process is currently being executed. In addition to principle, process, and domain, subject may include a Tag, which is a user assigned identification code used where added security is required. For a given process, the principle and process are constant but the domain is determined by the procedure currently being executed. A process's associated subject is therefore variable along the path of execution of the process.

In a present embodiment of CS 10110, procedures having KOS domain of execution have access to objects in KOS, EOS, DBMS, and User domains; procedures having EOS domain of execution have access to objects in EOS, DBMS, and User domains; procedures having DBMS domain of execution have access to objects in DBMS and User domains; and procedures having User domain of execution have access only to objects in User domain. A user cannot, therefore, obtain access to objects in KOS domain of execution and cannot influence CS 10110's low level, secure operating system. The user's process may, however, call for execution a procedure having KOS domain of execution. At this point the process's subject is in the KOS domain and the procedure will have access to certain objects in KOS domain.

In a present embodiment of CS 10110, also described in a later discussion, each object has associated with it an Access Control List (ACL). An ACL contains an Access Control Entry (ACE) for each subject having access to that object. ACEs specify, for each subject, access rights a subject has with regard to that object.

There is normally no relationship, other than that defined by an object's ACL, between subjects and objects. CS 10110, however, supports Extended Type Objects having Extended ACLs wherein a user may specifically define which subjects have what access rights to the object.

In another embodiment of CS 10110, described in a following discussion, access rights are granted on a dynamic basis. In executing a process, a procedure may call a second procedure and pass an argument to the called procedure. The calling procedure will also pass selected access rights to that argument to the called procedure. The passed access rights exist only for the duration of the call.

In the dynamic access embodiment, access rights are granted only at the time they are required. In the ACL embodiment, access rights are granted upon object creation or upon specific request. In either embodiment, each procedure to which arguments may be passed in a cross-domain call has associated with it an Access Information Array (AIA). A procedure's AIA states what access rights a calling procedure (subject) must have before the called procedure can operate on the passed argument. CS 10110's protection mechanisms compare the calling procedure's access rights to the rights required by the called procedure. This ensures that a calling procedure may not ask a called procedure to do what the calling procedure is not allowed to do. Effectively, a calling procedure can pass to a called procedure only the access rights held by the calling procedure.

Having described the general structure and operation and certain features of CS 10110, those and other features of CS 10110 operation will next be described in greater detail.

B. Computer System 10110 Information Structures and Mechanisms (Figs. 102, 103, 104, 105)

CS 10110 contains certain information structures and mechanisms to assist in efficient execution of processes. These structures and mechanisms may be considered as falling into three general types. The first type concerns the processes themselves, i.e., procedure and data objects comprising a user's process or directly related to execution of a user's process. The second type are for management, control, and execution of processes. These structures are generally shared by all processes active in CS 10110. The third type are CS 10110 micromachine information structures and mechanisms. These structures are concerned with the eternal operation of the CS 10110 micromachine and are private to the CS 10110 micro-machine.

a. Introduction (Fig. 102)

Referring to Fig. 102, a pictorial representation of CS 10110 (MEM 10112, FU 10120, and EU 10122) is shown with certain information structures and mechanisms depicted therein. It should be understood that these information structures and mechanisms transcend or "cut across" the boundaries between MEM 10112, FU 10120, EU 10122, and IOS 10116. Referring to the upper portion of Fig. 103 Process Structures 10210 contains those information structures and mechanisms most closely concerned with individual processes, the first and third types of information structures described above. Process Structures 10210 reside in MEM 10112 and Virtual Processes 10212 include Virtual Processes (VP) 1 through N. Virtual Processes 10212 may contain, in a present embodiment of CS 10110, up to 256 VP's. As previously described, each VP includes certain objects particular to a single user's process, for example stack objects previously described and further described in a following description. Each VP also includes a Process Object containing certain information required to execute the process, for example pointers to other process information.

Virtual Processor State Blocks (VPSBs) 10218 include VPSBs containing certain tables and mechanisms for managing execution of VPs selected for execution by CS 10110.

A particular VP is bound into CS 10110 when a Virtual Process Dispatcher, described in a following discussion selects that VP as eligible for execution. The selected VPs Process Object, as previously described, is swapped into a VPSB. VPSBs 10218 may contain, for example 16 or 32 State Blocks so that CS 10110 may concurrently execute to 16 or 32 VPs. When a VP assigned to a VPSB is to be executed, the VP is swapped onto the Information structures and mechanisms shown in FU 10120 and EU 10122. FU Register and Stack Mechanism (FURSM) 10214 and EU Register and Stack Mechanism (EURSM) 10216, shown respectively in FU 10120 and EU 10122, comprise register and stack mechanisms used in execution of VPs bound to CS 10110. These register and stack mechanisms, as will be discussed below, are also used for certain CS 10110 process management functions. Procedure Objects (POs) 10213 contains Procedure Objects (POs) 1 to N of the processes executing in CS 10110.

Addressing Mechanisms (AM) 10220 are a part of CS 10110's process management system and are generally associated with Computer System 10110 addressing functions as described in following discussions. UID/AON Tables 10222 is a structure for relating UID's and AON's, previously discussed. Memory Management Tables 10224 includes structures for (1) relating AON logical addresses and MEM 10112 physical addresses; (2) managing MEM 10112's physical address space; (3) managing transfer of information between MEM 10112 and CS 10110's backing store (ED 10124) and, (4) activating objects into CS 10110; Name Cache (NC) 10226 and Address Translation Cache (ATC) 10228 are acceleration mechanisms for storing addressing information relating to the VP currently bound to CS 10110. NC 10226, described further below, contains information relating operand Names to AON addresses. ATC 10228, also discussed further below, contains information relating AON addresses to MEM 10112 physical addresses.

Protection Mechanisms 10230, depicted below AM 10220, include protection Tables 10232 and Protection Cache (PC) 10234. Protection Tables 10232 contain information regarding access rights to each

object active in CS 10110. PC 10234 contains protection information relating to certain objects of the VP currently bound to CS 10110.

Microinstruction Mechanisms 10236, depicted below PM 10230, includes Micro-code (M Code) Store 10238, FU (Micro-code) M Code Structure 10240, and EU Micro-code (M Code) Structure 10242. These structures contain microinstruction mechanisms and tables for interpreting SINs and controlling the detailed operation of CS 10110. Micro-instruction Mechanisms 10232 also provide microcode tables and mechanisms used, in part, in operation of the low level, secure operating system that manages and controls CS 10110's physical resources.

Having thus briefly described certain CS 10110 information structures and mechanisms with the aid of Fig. 102, those information structures and mechanisms will next be described in further detail in the order mentioned above. In these descriptions it should be noted that, in representation of MEM 10112 shown in Fig. 102 and in other figures of following discussions, the addressable memory space of MEM 10112 is depicted. Certain portions of MEM 10112 address space have been designated as containing certain information structures and mechanisms. These structures and mechanisms have real physical existence in MEM 10112, but may vary in both location and volume of MEM 10112 address space they occupy. Assigning position of a single, large memory to contain these structures and mechanisms allows these structures and mechanisms to be reconfigured as required for most efficient operation of CS 10110. In an alternate embodiment, physically separate memories may be used to contain the structures and mechanisms depicted in MEM 10112, rather than assigned portions of a single memory.

b. Process Structure 10210 (Figs. 103, 104, 105)

Referring to Fig. 103, a partial schematic representation of Process Structures 10210 is shown. Specifically, Fig. 103 shows a Process (P) 10310 selected for execution, and its associated Procedure Objects (POs) in Process Objects (POs) 10213. P 10310 is represented in Fig. 103 as including four procedure objects in POs 10213. It is to be understood that this representation is for clarity of presentation; a particular P 10310 may include any number of procedure objects. Also for clarity of presentation, EURSM 10216 is not shown as EURSM 10216 is similar to FURSM 10214. EURSM 10216 will be described in detail in the following detailed discussons of CS 10110's structure and operation.

As previously discussed, each process includes certain data and procedure object. As represented in Fig. 103 for P 10310 the procedure objects reside in POs 10213. The data objects include Static Data Areas and stack mechanisms in P 10310. POs, for example KOS Procedure Object (KOSPO) 10318, contain the various procedures of the process, each procedure being a sequence of SINs defining an operation to be performed in executing the process. As will be described below, Procedure Objects also contain certain information used in executing the procedures contained therein. Static Data Areas (SDAs) are data objects generally reserved for storing data having an existence for the duration of the process. P 10310's stack mechanisms allow stacking of procedures for procedure calls and returns and for swapping processes in and out of JP 10114. Macro-Stacks (MAS) 10328 to 10334 are generally used to store automatic data (data generated during execution of a procedure and having an existence for the duration of that procedure). Although shown as separate from the stacks in P 10310, the SDAs may be contained with MASs 10328 to 10334. Secure Stack (SS) 10336 stores, in general, CS 10110 micro-machine state for each procedure called. Information stored in SS 10336 allows machine state to be recovered upon return from a called procedure, or when binding (swapping) a VP into CS 10110.

As shown in P 10310, each process is structured on a domain basis. A P 10310 may therefore include, for each domain, one or more procedure objects containing procedures having that domain as their domain of execution, an SDA and an MAS. For example, KOS domain of P 10310 includes KOSPO 10318, KOSSDA 10326, and KOSMAS 10334. P 10310's SS 10336 does not reside in any single domain of P 10310, but instead is a stack mechanism belonging to CS 10110 micromachine.

Having described the overall structure of a P 10310, the individual information structures and mechanisms of a P 10310 will next be described in greater detail.

1. Procedure Objects (Fig. 103)

KOSPO 10318 is typical of CS10110 procedure objects and will be referred to for illustration in the following discussion. Major components of KOSPO 10318 are Header 10338, External Entry Descripter (EED) Area 10340, Internal Entry Descripter (IED) Area 10342, S-Op Code Area 10344, Procedure Environment Descripter (PED) 10348, Name Table (NT) 10350, and Access Information Array (AIA) Area 10352.

Header 10338 contains certain information identifying PO 10318 and indicating the number of entries in EED area 10340, discussed momentarily.

EED area 10340 and IED area 10342 together contain an Entry Descripter (ED) for each procedure in KOSPO 10318. KOSPO 10318 is represented as containing Procedures 1, 2, and 11, of which Procedure 11 will be used as an example in the present discussion. EDs effectively comprise an index through certain all information in KOSPO 10318 can be located. IEDs form an index to all KOSPO 10318 procedures which may be called only from other procedures contained in KOSPO 10318. EEDs form an index to all KOSPO 10318 procedures which may be called by procedures external to KOSPO 10318. Externally callable procedures are distinguished aid, as described in a following discussion of CS 10110's protection mechanisms, in

confirming external calling procedure's access rights.

Referring to ED 11, ED for procedure 11, three fields are shown therein. Procedure Environment Descripter Offset (PEDO) field indicates the start, relative to start of KOSPO 10318, of Procedure 11's PED in PED Area 10348. As will be discussed further below, a procedure's PED contains a set of pointers for locating information used in the execution of that procedure. PED Area 10348 contains a PED for each procedure contained in 10318. In the present embodiment of CS 10110, a single PED may be shared by two or more procedures. Code Entry Point (CEP) field indicates the start, relative to Procedure Base Pointer (PBP) which will be discussed below, of Procedure 11's SIN Code and SIN Code Area 10344. Finally, ED 11's Initial Frame Size (IFS) field indicates the required Initial Frame Size of the KOSMAS 10334 frame storing Procedure 11's automatic data.

PED 11, Procedure 11's PED in PED Area 10348, contains a set of pointers for locating information used in execution of Procedure 11. The first entry in PED 11 is a header containing information identifying PED 11. PED 11's Procedure Base Pointer (PBP) entry is a pointer providing a fixed reference from which other information in PO 10318 may be located. In a specific example, Procedure 11's CEP indicates the location, relative to PBP, of the start of Procedure 11's S-Op code in S-Op Code Area 10344. As will be described further below, PBP is a CS 10110 Architectural Base Pointer (ABP). CS 10110's ABP's are a set of architectural pointers used in CS 10110 to facilitate addressing of CS 10110's address space. PED 11's Static Data Pointer (SDP) entry points to data, in PO 10318, specifying certain parameters of P 10310's KOSSDA 10326. Name Table Pointer (NTP) entry is a pointer indicating the location, in NT 10350, of Name Table Entry's (NTE's) for Procedure 11's operands. NT 10350 and NTE's will be described in greater detail in the following discussion of Computer System 10110's Addressing Structure. PED 11's S-Interpreter Pointer (SIP) entry is a pointer, discussed in greater detail in a following discussion of CS 10110's microcode structure, pointing to the particular S-Interpeter (SINT) to be used in interpreting Procedure 11's SIN Code.

Referring finally to AIA 10352, AIA 10352 contains, as previously discussed, information pertaining to access rights required of any external procedure calling a 10318 procedure. There is an AIA 10352 entry for each PO 10318 procedure which may be called by an external procedure. A particular AIA entry may be shared by one or more procedures having an ED in EED Area 10340. Each EED contains certain information, not shown for clarity of presentation, indicating that that procedure's corresponding AIA entry must be referred to, and the calling procedure's access rights confirmed, whenever that procedure is called.

2. Stack Mechanism (Figs. 104, 105)

As previously described, P10310's stack mechanisms include SS 10336, used in part for storing machine state, and MAS's 10328 to 10334, used to store local data generated during execution of P 10310's procedures. P 10310 is represented as containing an MAS for each CS 10110 domain. In an alternate embodiment of CS 10110, a particular P 10310 will include MAS's only for those domains in which that P 10310 is executing a procedure.

Referring to MAS's 10328 to 10334 and SS 10336, P 10310 is represented as having had eleven procedure calls. Procedure 0 has called Procedure 1, Procedure 1 has called Procedure 2, and so on. Each time a procedure is called, a corresponding stack frame is constructed on the MAS of the domain in which the called procedure is executed. For example, Procedures 1, 2, and 11 execute in KOS domain; MAS frames for Procedures 1, 2, and 11 therefore are placed on KOSMAS 10334. Similarly, Procedures 3 and 9 execute in EOS domain, so that their stack frames are placed on EOSMAS 10332. Procedures 5 and 6 execute in DBMS domain, so that their stack frames are placed on DBMSMAS 10330. Procedures 4, 7, 8, and 10 execute in User domain with their stack frames being placed on USERMAS 10328. Procedure 11 is the most recently called procedure and procedure 11's stack frame on KOSMAS 10334 is referred to as the current frame. Procedure 11 is the procedure which is currently being executed when VP 10310 is bound to CS 10110.

SS 10336, which is a stack mechanism of CS 10110 micromachine, contains a frame for each of Procedures 1 to 11. Each SS 10336 frame contains, in part, CS 10110 operating state for its corresponding procedure.

Referring to Fig. 104, a schematic representation of a typical MAS, for example KOSMAS 10334, is shown. KOSMAS 10334 includes Stack Header 10410 and a Frame 10412 for each procedure on KOSMAS 10334. Each Frame 10412 includes a Frame Header 10414, and may contain a Linkage Pointer Block 10416, a Local Pointer Block 10418, and a Local (Automatic) Data Block 10420.

KOSMAS 10334 Stack Header 10410 contains at least the following information:

(1) an offset, relative to Stack Header 10410, indicating the location of Frame Header 10414 of the first frame on KOSMAS 10334;

(2) a Stack Top Offset (STO) indicating location, relative to start of KOSMAS 10334, of the top of KOSMAS 10334; top of KOSMAS 10334 is indicated by pointer STO pointing to the top of the last entry of Procedure 11 Frame 10412's Local Data Block 10420;

(3) an offset, relative to start of KOSMAS 10334, indicating location of Frame Header 10414 of the current top frame of KOSMAS 10334; in Fig. 104 this offset is represented by Frame Pointer (FP), an ABP discussed further below;

(4) the VP 10310's UID;

(5) a UID Pointer indicating location of certain domain environment information, described further in a

following discussion;

(6) a signaller pointer indicating the location of certain routines for handling certain CS 10110 operating system faults;

(7) a UID pointer indicating location of KOSSDA 10326; and

(8) a frame label sequencer containing pointers to headers of frames in other domains; these pointers are used in executing non-local go-to operations.

KOSMAS 10334 Stack Header 10410 thereby contains information for locating certain important points in KOSMAS 10334's structure, and for locating certain information pertinent to executing procedures in KOS domain.

Each Frame Header 10414 contains at least the following information:

(1) offsets, relative to the Frame Header 10414, indicating the locations of Frame Headers 10414 of the previous and next frames of KOSMAS 10334;

(2) an offset, relative to the Frame Header 10414, indicating the location of the top of that Frame 10412;

(3) information indicating the number of passed arguments contained in that Frame 10412;

(4) a dynamic back pointer, in UID/Offset format, to the previous Frame 10412 if that previous Frame 10412 resides in another domain;

(5) a UID/Offset pointer to the environmental descripter of the procedure calling that procedure;

(6) a frame label sequence containing information indicating the locations of other Frame Headers 10414 in KOSMAS 10334; this information is used to locate other frames in KOSMAS 10334 for the purpose of executing local go-to operations. Frame Headers 10414 thereby contain information for locating certain important points in KOSMAS 10334 structure, and certain data pertinent to executing the associated procedures. In addition, Frame Headers 10414, in combination with Stack Header 10410, contain information for linking the activation records of each VP 10310 MAS, and for linking together the activation records of the individual MAS's.

Linkage Pointer Blocks 10416 contain pointers to arguments passed from a calling procedure to the called procedure. For example, Linkage Pointer Block 10416 of Procedure 11's Frame 10412 will contain pointers to arguments passed to Procedure 11 from Procedure 10. The use of linkage pointers in CS 10110's addressing structure will be discussed further in a following discussion of CS 10110's Addressing Structure. Local Data Pointer Blocks 10418 contain pointers to certain of the associated procedure's local data. Indicated in Fig. 104 is a pointer, Frame Pointer (FP), pointing between top most Frame 10412's Linkage Pointer Block 10416 and Local Data Pointer Block 10418. FP, described further in following discussions, is an ABP to MAS Frame 10412 of the process's current procedure.

Each Frame 10412's Local (Automatic) Data Block 10420 contains certain of the associated procedure's automatic data.

As described above, at each procedure call a MAS frame is constructed on top of the MAS of the domain in which the called procedure is executed. For example, when Procedure 10 calls Procedure 11 a Frame Header 10414 for Procedure 11 is constructed and placed on KOSMAS 10334. Procedure 11's linkage pointers are then generated, and placed in Procedure 11's Linkage Pointer Block 10416. Next Procedure 11's local pointers are generated and placed in Procedure 11's Local Pointer Block 10418. Finally, Procedure 11's local data is placed in Procedure 11's Local Data Block 10420. During this operation, USERMAS 10328's frame label sequence is updated to include an entry pointing to Procedure 11's Frame Header 10414. KOSMAS 10334's Stack Header 10410 is updated with respect to STO to the new top of KOSMAS 10334. Procedure 2's Frame Header 10414 is updated with respect to offset to Frame Header 10414 of Procedure 11 Frame 10412, and with respect to frame label sequence indicating location of Procedure 11's Frame Header 10414. As Procedure 11 is then the current procedure, FP is updated to a point between Linkage Pointer Block 10416 and Local Pointer Block 10418 of Procedure 11's Frame 10412. Also, as will be discussed below, a new frame is constructed on SS 10336 or Procedure 11. CS 10110 will then proceed to execute Procedure 11. During execution of Procedure 11, any further local data generated may be placed on the top of Procedure 11's Local Data Block 10420. The top of stack offset information in Procedure 11's Frame Header 10414 and in KOSMAS 10334 Stack Header 10410 will be updated accordingly.

MAS's 10328 to 10334 thereby provide a per domain stack mechanism for storing data pertaining to individual procedures, thus allowing stacking of procedures without loss of this data. Although structured on a domain basis, MAS's 10328 to 10334 comprise a unified logical stack structure threaded together through information stored in MAS stack and frame headers.

As described above and previously, SS 10336 is a CS 10110 micromachine stack structure for storing, in part, CS 10110 micromachine state for each stacked VP 10310 procedure. Referring to Fig. 105, a partial schematic representation of a SS 10336 Stack Frame 10510 is shown. SS 10336 Stack Header 10512 and Frame Headers 10514 contain information similar to that in MAS Stack Headers 10410 and Frame Headers 10414. Again, the information contained therein locates certain points within SS 10336 structure, and threads together SS 10336 with MAS's 10328 to 10334.

SS 10336 Stack Frame 10510 contains certain information used by the CS 10110 micromachine in executing the VP 10212 procedure with which this frame is associated. Procedure Pointer Block 10516 contains certain pointers including ABPs, used by CS 10110 micromachine in locating information within VP 10310's information structures. Micro-Routine Frames (MRFs) 10518 together comprise Micro-Routine Stack (MRS) 10520 within each SS 10336 Stack Frame 10510. MRS Stack 10520 is associated with the

internal operation of CS 10110 microroutines executed during execution of the VP 10212 procedure associated with the Stack Frame 10510. SS 10336 is thus a dual function CS 10110 micromachine stack. Pointer Block 10516 entries effectively define an interface between CS 10110 micromachine and the current procedure of the current process. MRS 10520 comprise a stack mechanism for the internal operations of CS 10110 micromachine.

Having briefly described Virtual Processes 10212, FURSM 10214 will be described next. As stated above, EURSM 10216 is similar in operation to FURSM 10214 and will be described in following detailed descriptions of CS 10110 structure and operation.

3. FURSM 10214 (Fig. 103)

Referring again to Fig. 103, FURSM 10214 includes CS 10110 micromachine information structures used internally to CS 10110 micromachine in executing the procedures of a P 10310. When a VP, for example P 10310, is to be executed, certain information regarding that VP is transferred from the Virtual Processes 10212 to FURSM 10214 for use in executing that procedure. In this respect, FURSM 10214 may be regarded as an acceleration mechanism for the current Virtual Process 10212.

FURSM 10214 includes General Register File (GRF) 10354, Micro Stack Pointer Register Mechanism (MISPR) 10356, and Return Control Word Stack (RCWS) 10358. GRF 10354 includes Global Registers (GRs) 10360 and Stack Registers (SRs) 10362. GRs 10360 include Architectural Base Registers (ABRs) 10364 and Micro-Control Registers (MCRs) 10366. Stack Registers 10362 include Micro-Stack (MIS) 10368 and Monitor Stack (MOS) 10370.

Referring first to GRF 10354, and assuming for example that Procedure 11 of P 10310 is currently being executed, GRF 10354 primarily contains certain pointers to P 10310 data used in execution of Procedure 11. As previously disussed, CS 10110's addressing structure includes certain Architectural Base Pointers (ABP's) for each procedure. ABPs provide a framework for excessing CS 10110's address space. The ABPs of each procedure include a Frame Pointer (FP), a Procedure Base Pointer (PBP), and a Static Data Pointer (STP). As discussed above with reference to KOSPO 10318, these ABPs reside in the procedure's PEDs. When a procedure is called, these ABP's are transferred from that procedure's PED to ABR's 10364 and reside therein for the duration of that procedure. As indicated in Fig. 103, FP points between Linkage Pointer Block 10416 and Local pointer Blocks 10418 of Procedure 11's Frame 10412 on KOSMAS 10334. PBP points to the reference point from which the elements of KOSPO 10318 are located. SDP points to KOSSDA 10326. If Procedure 11 calls, for example, a Procedure 12, Procedure 11's ABPs will be transferred onto Procedure Pointer Block 10516 of SS 10336 Stack Frame 10510 for Procedure 11. Upon return to Procedure 11, Procedure 11's ABPs will be transferred from Procedure Pointer Block 10516 to ABR's 10364 and execution of Procedure 11 resumed.

MCRs 10336 contain certain pointers used by CS 10110 micromachine in executing Procedure 11. CS 10110 micromachine pointers indicated in Fig. 103 include Program Counter (PC), Name Table Pointer (NTP), S-Interpreter Pointer (SIP), Secure Stack Pointer (SSP), and Secure Stack Top Offset (SSTO). NTP and SIP have been previously described with reference to KOSPO 10318 and reside in KOSPO 10318. NTP and SIP are transferred into MCR's 10366 at start of execution of Procedure 11. PC, as indicated in Fig. 103, is a pointer to the Procedure 11 SIN currently being executed by CS 10110. PC is initially generated from Procedure 11's PBP and CEP and is thereafter incremented by CS 10110 micromachine as Procedure 11's SIN sequences are executed. SSP and SSTO are, as described in a following discussion, generated from information contained in SS 10336's Stack Header 10512 and Frame Headers 10514. As indicated in Fig. 103 SSP points to start of SS 10336 while SSTO indicates the current top frame on SS 10336, whether Procedure Pointer Block 10516 or a MRF 10518 of MRS 10520, by indicating an offset relative to SSP. If Procedure 11 calls a subsequent procedure, the contents of MCR's 10366 are transferred into Procedure 11's Procedure Pointer Block 10516 on SS 10336, and are returned to MCR's 10366 upon return to Procedure 11.

Registers 10360 contain further pointers, described in following detailed discussions of CS 10110 operation, and certain registers which may be used to contain the current procedure's local data.

Referring now to Stack Registers 10362, MIS 10368 is an upward extension, or acceleration, of MRS 10520 of the current procedure. As previously stated, MRS 10520 is used by CS 10110 micromachine in executing certain microroutines during execution of a particular procedure. MIS 10368 enhances the efficiency of CS 10110 micromachine in executing these microroutines by accelerating certain most recent MRFs 10518 of that procedure's MRS 10520 into FU 10120. MIS 10368 may contain, for example, up to the eight most recent MRFs 10518 of the current procedures MRS 10520. As various microroutines are called or returned from, MRS 10520 MRF's 10518 are transferred accordingly between SS 10336 and MIS 10368 so that MIS 10368 always contains at least the top MRF 10518 of MRS 10520, and at most eight MRFs 10518 of MRS 10520. MISPR 10356 is a CS 10110 micromachine mechanism for maintaining MIS 10368. MISPR 10356 contains a Current Pointer, a Previous Pointer, and a Bottom Pointer. Current Pointer points to the top-most MRF 10518 on MIS 10368. Previous Pointer points to the previous MRF 10518 on MIS 10368, and Bottom Pointer points to the bottom-most MRF 10518 on MIS 10368. MISPR 10356's Current, Previous and Bottom Pointers are updated as MRFs 10518 are transferred between SS 10336 and MIS 10368. If Procedure 11 calls a subsequent procedure, all Procedure 11 MRFs 10518 are transferred from MIS 10368 to Procedure 11's MRS 10520 on SS 10336. Upon return to Procedure 11, up to seven of Procedure 11's MRFs 10518

frames are returned from SS 10336 to MIS 10368.

Referring to MOS 10370, MOS 10370 is a stack mechanism used by CS 10110 micromachine for certain microroutines for handling fault or error conditions. These microroutines always run to completion, so that MOS 10370 resides entirely in FM 10120 and is not an extension of a stack residing in a P 10310 in MEM 10112. MOS 10370 may contain, for example, eight frames. If more than eight successive fault or error conditions occur, this is regarded as a major failure of CS 10110. Control of CS 10110 may then be transferred to DP 10118. As will be described in a following discussion, diagnostic programs in DP 10118 may then be used to diagnose and locate the CS 10110 faults or errors. In other embodiments of CS 10110 MOS 10370 may contain more or fewer stack frames, depending upon the degree of self diagnosis and correction capability desired for CS 10110.

RCWS 10358 is a two-part stack mechanism. A first part operates in parallel with MIS 10368 and a second part operates in parallel with MOS 10370. As previously described, CS 10110 is a microcode controlled system. RCWS is a stack for storing the current microinstruction being executed by CS 10110 micromachine when the current procedure is interrupted by a fault or error condition, or when a subsequent procedure is called. That portion of RCWS 10358 associated with MIS 10368 contains an entry for each MRF 10518 residing in MIS 10368. These RCWS 10358 entries are transferred between SS 10336 and MIS 10368 in parallel with their associated MRFs 10518. When resident in SS 10336, these RCWS 10358 entries are stored within their associated MRFs 10518. That portion of RCWS 10358 associated with MOS 10370 similarly operates in parallel with MOS 10370 and, like MOS 10370, is not an extension of an MEM 10112 resident stack.

In summary, each process active in CS 10110 exists as a separate, complete, and self-contained entity, or Virtual Process, and is structurally organized on a domain basis. Each Virtual Process includes, besides procedure and data objects, a set of MAS's for storing local data of that processes procedures. Each Virtual Process also includes a CS 10110 micromachine stack, SS 10336, for storing CS 10110 micromachine state pertaining to each stacked procedure of the Virtual Process. CS 10110 micromachine includes a set of information structures, register 10360, MIS 10368, MOS 10370, and RCWS 10358, used by CS 10110 micromachine in executing the Virtual Process's procedures. Certain of these CS 10110 micromachine information structures are shared with the currently executing Virtual Process, and thus are effectively acceleration mechanisms for the current Virtual Process, while others are completely internal to CS 10110 micromachine.

A primary feature of CS 10110 is that each process' macrostacks and secure stack resides in MEM 10112. CS 10110's macrostack and secure stacks are therefore effectively unlimited in depth.

Yet another feature of CS 10110 micromachine is the use of GRF 10354. GRF 10354 is, in an embodiment of CS 10110, a unitary register array containing for example, 256 registers. Certain portions, or address locations, of GRF 10354 are dedicated to, respectively, GRs 10360, MIS 10368, and MOS 10370. The capacities of GR 10360, MIS 10368, and MOS 10370, may therefore be adjusted, as required for optimum CS 10110 efficiency, by reassignment of GRF 10354's address space. In other embodiments of CS 10110, GRs 10360, MIS 10368, and MOS 10370 may be implemented a functionally separate registers arrays.

Having briefly described the structure and operation of Process Structures 10210, VP State Block 10218 will be described next below.

C. Virtual Processor State Blocks and Virtual Process Creation (Fig. 102)

Referring again to Fig. 102, VP State Blocks 10218 is used in management and control of processes. VP State Blocks 10218 contains a VP State Block for each Virtual Process (VP) selected for execution by CS 10110. Each such VP State Block contains at least the following information: (1) the state, or identification number of a VP;

(2) entries identifying the particular principle and particular process of the VP;

(3) an AON pointer to that VP's secure stack (e.g., SS 10336);

(4) the AON's of that VP's MAS stack objects (e.g., MAS's 10328 to 10334); and,

(5) certain information used by CS 10110's VP Management System.

The information contained in each VP State Block thereby defines the current state of the asociated VP.

A Process is loaded into CS 10110 by building a primitive access record and loading this access record into CS 10110 to appear as an already existing VP. A VP is created by creating a Process Object, including pointers to macro- and secure-stack objects created for that VP, micromachine state entries, and a pointer to the user's program. CS 10110's KOS then generates Macro- and Secure-Stack Objects with headers for that process and, as described further below, loads protection information regarding that process' objects into protection Structures 10230. CS 10110's KOS then copies this primitive machine state record into a vacant VPSB selected by CS 10110's VP Manager, thus binding the newly created VP into CS 10110. At that time a KOS Initializer procedure completes creation of the VP for example by calling in the user's program through a compiler. The newly creatd VP may then be executed by CS 10110.

Having briefly described VP State Blocks 10218 and creation of a VP, CS 10110's Addressing Structures 10220 will be decribed next below.

D. Addressing Structure 10220 (Figs. 103, 106, 107, 108)

1. Objects, UID's, AON's, Names, and Physical Addresses (Fig. 106)

As previously described, the data space accessible to CS 10110 is divided into segments, or containers, referred to as objects. In an embodiment of CS 10110, the addressable data space of each object has a capacity of 2^{32} bits of information and is structured into 2^{18} pages with each page containing 2^{14} bits of information.

Referring to Fig. 106A, a schematic representation of CS 10110's addressing structure is shown. Each object created for use in, or by operation of, a CS 10110 is permanently assigned a unique identifier (UID). An object's UID allows an object to be uniquely identified and located at any future point in time. Each UID is an 80 bit number, so that the total addressable space of all CS 10110's includes 2^{80} objects wherein each object may contain up to 2^{32} bits of information. As indicated in Fig. 106, each 80 bit UID is comprised of 32 bits of Logical Allocation Unit Identifier (LAUID) and 48 bits of Object Serial Number (OSN). LAUIDs are associated with individual CS 10110 systems. LAUIDs identify the particular CS 10110 system generating a particular object. Each LAUID is comprised of a Logical Allocation Unit Group Number (LAUGN) and a Logical Allocation Unit Serial Number (LAUSN). LAUGNs are assigned to individual CS 10110 systems and may be guaranteed to be unique to a particular system. A particular system may, however, be assigned more than one LAUGN so that there may be a time varying mapping between LAUGNs and CS 10110 systems. LAUSNs are assigned within a particular system and, while LAUSNs may be unique within a particular system, LAUSNs need not be unique between systems and need not map onto the physical structure of a particular system.

OSNs are associated with individual objects created by an LAU and are generated by an Architectural Clock in each CS 10110. Architectural clock is defined as a 64 bit binary number representing increasing time. Least significant bit of architectural clock represents increments of 600 picoseconds, and most significant bit represents increments of 127 years. In the present embodiment of CS 10110, certain most significant and least significant bits of architectural clock time are disregarded as generally not required practice. Time indicated by architectural clock is measured relative to an arbitrary, fixed point in time. This point in time is the same for all CS 10110s which will ever be constructed. All CS 10110s in existence will therefore indicate the same architectural clock time and all UIDs generated will have a common basis. The use of an architectural clock for generation of OSNs is advantageous in that it avoids the possibility of accidental duplication of OSNs if a CSC 10110 fails and is subsequently reinitiated.

As stated above, each object generated by or for use in a CS 10110 is uniquely identified by its associated UID. By appending Offset (O) and Length (L) information to an object's UID, a UID logical address is generated which may be used to locate particular segments of data residing in a particular object. As indicated in Fig. 106, O and L fields of a UID logical address are each 32 bits. O and L fields can therefore indicate any particular bit, out of 2^{32-1} bits, in an object and thus allow bit granular addressing of information in objects.

As indicated in Fig. 106 and as previously described, each object active in CS 10110 is assigned a short temporary unique identifier valid only within JP 10114 and referred to as an Active Object Number (AON). Because fewer objects may be active in a CS 10110 than may exist in a CS 10110's address space, AON's are, in the present embodiment of CS 10110, 14 bits in length. A particular CS 10110 may therefore contain up to 2^{14} active objects. An object's AON is used within JP 10114 in place of that object's UID. For example, as discussed above with reference to process structures 10210, a procedure's FP points to start of that procedure's frame on its process' MAS. When that FP is residing in SS 10336, it is expressed as a UID. When that procedure is to be executed, FP is transferred from SS 10336 to ABR's 10364 and is translated into the corresponding AON. Similarly, when that procedure is stacked, FP is returned to SS 10336 and in doing so is translated into the corresponding UID. Again, a particular data segment in an object may be addressed by means of an AON logical address comprising the object's AON plus associated 32 bit Offset (O) and Length (L) fields.

Each operand appearing in a process is assigned a Name and all references to a process's operands are through those assigned Names. As indicated in Fig. 106B, in the present embodiment of CS 10110 each Name is an 8, 12, or 16 bit number. All Names within a particular process will be of the same length. As will be described in a following discussion, Names appearing during execution of a process may be resolved, through a procedure's Name Table 10350 or through Name Cache 10226, to an AON logical address. As described below, an AON logical address corresponding to an operand Name may then be evaluated to a MEM 10112 physical address to locate the operand referred to.

The evaluation of AON logical addresses to MEM 10112 physical addresses is represented in Fig. 106C. An AON logical address's L field is not involved in evaluation of an AON logical address to a physical address and, for purposes of clarity of presentation, is therefore not represented in Fig. 106C. AON logical address L field is to be understood to be appended to the addresses represented in the various steps of the evaluation procedure shown in Fig. 106C.

As described above, objects are 2^{32} bits structured into 2^{18} pages with each page containing a 2^{14} bits of data. MEM 10112 is similarly physically structured into frames with, in the present embodiment of CS 10110, each frame containing 2^{14} bits of data. In other embodiments of CS 10110, both pages and frames may be of different sizes but the translation of AON logical addresses to MEM 10112 physical addresses will be similar to that described momentarily.

An AON logical address O field was previously described as a 32 bit number representing the start, relative to start of the object, of the addressed data segment within the object. The 18 most significant bits of O field represent the number (P) of the page within the object upon which the first bit of the addressed data occurs. The 14 least significant bits of O field represent the offset (O_p), relative to the start of the page, within that page of the first bit of the addressed data. AON logical address O field may therefore, as indicated in Fig. 106C, be divided into an 18 bit page (P) field and a 14 bit offset within page (O_p) field. Since, as described above, MEM 10112 physical frame size is equal to object page size, AON logical address O_p field may be used directly as an offset within frame (O_f) field of the physical address. As will be described below, an AON logical address AON and P fields may then be related to the frame number (FN) of the MEM 10112 frame in which that page resides, through Addressing Mechanisms 10220.

Having briefly described the relationships between UIDs, UID Logical Addresses, Names, AONs, AON Logical Addresses, and MEM 10112 Physical Addresses, Addressing Mechanisms 10220 will be described next below.

2. Addressing Mechanisms 10220 (Fig. 107)

Referring to Fig. 107, a schematic representation of Computer System 10110's Addressing Mechanisms 10220 is shown. As previously described, Addressing Mechanisms 10220 comprise UID/AON Tables 10222, Memory Management Tables 10224, Name Cache 10226, and Address Translation Unit 10228.

UID/AON Tables 10222 relate each object's UID to its assigned AON and include AOT Hash Table (AOTHT) 10710, Active Object Table (AOT) 10712, and Active Object Table Annex (AOTA) 10714.

An AON corresponding to a particular UID is determined through AOTHT 10710. The UID is hashed to provide a UID index into AOTHT 10710, which then provides the corresponding AON. AOTHT 10710 is effectively an acceleration mechanism of AOT 10712 to, as just described, provide rapid translation of UIDs to AONs. AONs are used as indexes into AOT 10712, which provides a corresponding AOT Entry (AOTE). An AOTE as described in following detailed discussions of CS 10110, includes, among other information, the UID corresponding to the AON indexing the AOTE. In addition to providing translation between AONs and UIDs, the UID of an AOTE may be compared to an original UID to determine the correctness of an AON from AOTHT 10710.

Associated with AOT 10712 is AOTA 10714. AOTA 10714 is an extension of AOT 10712 and contains certain information pertaining to active objects, for example the domain of execution of each active procedure object.

Having briefly described CS 10110's mechanism for relating UIDs and AONs, CS 10110's mechanism for resolving operand Names to AON logical addresses will be described next below.

3. Name Resolution (Figs. 103, 108)

Referring first to Fig. 103, each procedure object in a VP, for example KOSPO 10318 in VP 10310, was described as containing a Name Table (NT) 10350. Each NT 10350 contains a Name Table Entry (NTE) for each operand whose Name appears its procedure. Each NTE contains a description of how to resolve the corresponding Name to an AON Logical Address, including fetch mode information, type of data referred to by that Name, and length of the data segment referred to.

Referring to Fig. 108, a representation of an NTE is shown. As indicated, this NTE contains seven information fields: Flag, Base (B), Predisplacement (PR), Length (L), Displacement (D), Index (I), and Inter-element Spacing (IES). Flag Field, in part, contains information describing how the remaining fields of the NTE are to be interpreted, type of information referred to by the NTE, and how that information is to handled when fetched from MEM 10112. L Field, as previously described, indicates length, or number of bits in, the data segment. Functions of the other NTE fields will be described during the following discussions.

In a present embodiment of CS 10110, there are five types of NTE: (1) base (B) is not a Name, address resolution is not indirect; (2) B is not a Name, address resolution is indirect; (3) B is a Name, address resolution is indirect; (4) B is a Name, address resolution is indirect. A fifth type is an NTE selecting a particular element from an array of elements. These five types of NTE and their resolution will be described below, in the order mentioned.

In the first type, B is not a Name and address resolution is not indirect, B Field specifies an ABR 10364 containing an AON plus offset (AON/O) Pointer. The contents of D Field are added to the O Field of this pointer, and the result is the AON Logical Address of the operand. In the second type, B is not a Name and address resolution is indirect, B Field again specifies an ABR 10364 containing an AON/O pointer. The contents of PR Field are added to the O Field of the AON/O pointer to provide an AON Logical Address of a Base Pointer. The Base Pointer AON Logical Address is evaluated, as described below, and the Base Pointer fetched from MEM 10112. The contents of D Field are added to the O Field of the Base Pointer and the result is the AON Logical Address of the operand.

NTE types 3 and 4 correspond, respectively to NTE types 1 and 2 and are resolved in the same manner except that B Field contains a Name. The B Field Name is resolved through another NTE to obtain an AON/O pointer which is used in place of the ABR 10364 pointers referred to in discussion of types 1 and 2.

The fifth type of NTE is used in references to elements of an array. These array NTEs are resolved in the

same manner as NTE types 1 through 4 above to provide an AON Logical Address of the start of the array. I and IES Fields provide additional information to locate a particular element in the array. I Field is always Name which is resolved to obtain an operand value representing the particular element in the array. IES Field provides information regarding spacing between elements of the array, that is the number of bits between adjacent element of the array. IES Field may contain the actual IES value, or it may contain a Name which is resolved to an AON Logical Address leading to the inter-element spacing value. The I and IES values, obtained by resolving the I and IES Fields as just described, are multiplied together to determine the offset, relative to the start of the array, of the particular element referred to by the NTE. This within array offset is added to the O Field of the AON Logical Address of the start of the array to provide the AON Logical Address of the element.

In the current embodiment of CS 10110, certain NTE fields, for example B, D, and Flag fields, always contain literals. Certain other fields, for example, IES, D, PRE, and L fields, may contain either literals or names to be resolved. Yet other fields, for example I field, always contain names which must be resolved.

Passing of arguments from a calling procedure to a called procedure has been previously discussed with reference to Virtual Processes 10212 above, and more specifically with regard to MAS's 10328 to 10334 of VP 10310. Passing of arguments is accomplished through the calling and called procedure's Name Tables 10350. In illustration, a procedure W(a, b, c) may wish to pass arguments a, b, and c to procedure X(u, v, w), where arguments, v and w correspond to arguments a, b, and c. At compilation, NTEs are generated for arguments a, b, and c in procedure W's procedure object, and NTEs are generated for arguments u, v and w in Procedure X's procedure object. Procedure X's NTEs for u, v, and w are constructed to resolve to point to pointers in Linkage Pointer Block 10416 of Procedure X's Frame 10412 in MAS. To pass arguments a, b, and c from Procedure W to Procedure X, the NTEs of arguments a, b, and c are resolved to AON Logical Addresses (i.e., AON/O form). Arguments a, b, and c's AON Logical Addresses are then translated to corresponding UID addresses which are placed in Procedure X's Linkage Pointer Block 10416 at those places pointed to by Procedure X's NTEs for u, v, and w. When Procedure X is executed, the resolution of Procedure X's NTEs for u, v, and w will be resolved to locate the pointers, in Procedure X's Linkage Pointer Block 10416 to arguments a, b, and c. When arguments are passed in this manner, the data type and length information are obtained from the called procedure's NTEs, rather than the calling procedure's NTEs. This allows the calling procedure to pass only a portion of, for example, arguments a, b, or c, to the called procedure and thus may be regarded as a feature of CS 10110's protection mechanisms.

Having briefly described resolution of Names to AON/Offset addresses, and having previously described translation of UID addresses to AON addresses, the evaluation of AON addresses to MEM 10112 physical addresses will be described next below.

4. Evaluation of AON Addresses to Physical Addresses (Fig. 107)

Referring again to Fig. 107, a partial schematic representation of CS 10110's Memory Management Table 10224 is shown. Memory Hash Table (MHT) 10716 and Memory Frame Table (MFT) 10718 are concerned with translation of AON addresses into MEM 10112 physical addresses and will be discussed first. Working Set Matrix (WSM) 10720 and Virtual Memory Manager Request Queue (VMMRQ) 10722 are concerned with management of MEM 10112's available physical address base and will be discussed second. Active Object Request Queue (AORQ) 10728 and Logical Allocation Unit Directory (LAUD) 10730 are concerned with locating inactive objects and management of which objects are active in CS 10110 and will be discussed last.

Translation of AON/O Logical Addresses to MEM 10112 physical addresses was previously discussed with reference to Fig. 106C. As stated in that discussion, objects are divided into pages. Correspondingly, the AON/O Logical Address' O Field is divided into an 18 bit page number (P) Field and a 14 bit offset within a page (O_p) Field. MEM 10112 is structured into frames, each of which in the present embodiment of CS 10110 is equal to a page of an object. An AON/O address' O_p Field may therefore be used directly as an offset within frame (O_f) of the corresponding physical address. The AON and P fields of an AON address must, however, be translated into a MEM 10112 frame represented by a corresponding Frame Number (FN).

Referring now to Fig. 107, an AON address' AON and P Fields are "hashed" to generate an MHT index which is used as an index into MHT 10716. Briefly, "hashing" is a method of indexing, or locating, information in a table wherein indexes to the information are generated from the information itself through a "hashing function". A hashing function maps each piece of information to the corresponding index generated from it through the hashing function. MHT 10716 then provides the corresponding FN of the MEM 10112 frame in which that page is stored. FNs are used as indexes into MFG 10718, which contains, for each FN, an entry describing the page stored in that frame. This information includes the AON and P of the page stored in that MEM 10112 frame. An FN from MHT 10716 may therefore be used as an index into MFT 10718 and the resulting AON/P of MFT 10718 compared to the original AON/P to confirm the correctness of the FN obtained from MHT 10716. MHT 10716 is an effectively acceleration mechanism of MFT 10718 to provide rapid translation of AON address to MEM 10112 physical addresses.

MFT 10718 also stores "used" and "modified" information for each page in MEM 10112. This information indicates which page frames stored therein have been used and which have been modified.

This information is used by CS 10110 in determining which frames may be deleted from MEM 10112, or are free, when pages are to be written into MEM 10112 from backing store (ED 10124). For example, if a page's modified bit indicates that that page has not been written into, it is not necessary to write that page back into backing store when it is deleted from MEM 10112; instead, that page may be simply erased.

Referring finally to ATU 10228, ATU 10228 is an acceleration mechanism for MHT 10716. AON/O addresses are used directly, without hashing, as indexes into ATU 10228 and ATU 10228 correctly provides corresponding FN and O outputs. A CS 10110 mechanism, described in a following detailed discussion of CS 10110 operation, continually updates the contents of ATU 10228 so that ATU 10228 contain the FN's and O_p (O_i) of the pages most frequently referenced by the current process. If ATU 10228 does not contain a corresponding entry for a given AON input, an ATU fault occurs and the FN and O information may be obtained directly from MHT 10716.

Referring now to WSM 10720 and VMMRQ 10722, as previously stated these mechanisms are concerned with the management of MEM 10112's available address space. For example, if MHT 10716 and MFT 10718 do not contain an entry for a page referenced by the current procedure, an MHT/MFT fault occurs and the reference page must be fetched from backing store (ED 10124) and read into MEM 10112. WSM 10720 contains an entry for each page resident in MEM 10112. These entries are accessed by indexes comprising the Virtual Processor Number (VPN) of the virtual process making a page reference and the P of the page being referenced. Each WSM 10720 entry contains 2 bits stating whether the particular page is part of a VP's working set, that is, used by that VP, and whether that page has been referenced by that VP. This information, together with the information contained in that MFT 10718 entries described above, is used by CS 10110's Virtual Memory Manager (VMM) in transferring pages into and out of MEM 10112.

CS 10110's VMM maintains VMMRQ 10722, which is used by VMM to control transfer of pages into and out of MEM 10112. VMMRQ 10722 includes Virtual Memory Request Counter (VMRC) 10724 and a Queue of Virtual Memory Request Entries (VMREs) 10726. As will be discussed momentarily, VMRC 10724 tracks the number of currently outstanding request for pages. Each VMRE 10726 describes a particular page which has been requested. Upon occurrence of a MHT/MFT (or page) fault, VMRC 10724 is incremented, which initiates operation of CS 10110's VMM, and a VMRE 10726 is placed in the queue. Each VMRE 10726 comprises the VPN of the process requesting the page and the AON/O of the page requested. At this time, the VP making the request is swapped out of JP 10114 and another VP bound to JP 10114. VMM allocates MEM 10112 frame to contain the requested page, using the previously described information in MFT 10718 and WSM 10720 to select this frame. In doing so, VMM may discard a page currently resident in MEM 10112 for example, on the basis of being the oldest page, an unused page, or an unmodified page which does not have to be written back into backing store. VMM then requests an I/O operation to transfer the requested page into the frame selected by the VMM. While the I/O operation is proceeding, VMM generates new entries in MHT 10716 and MFT 10718 for the requested page, cleans the frame in MEM 10112 which is to be occupied by that page, and suspends operation. IOS 10116 will proceed to execute the I/O operation and writes the requested page directly into MEM 10112 in the frame specified by VMM. IOS 10116 then notifies CS 10110's VMM that the page now resides in memory and can be referenced. At some later time, that VP requesting that page will resume execution and repeat that reference. Going first to ATU 10228, that VP will take an ATU 10228 fault since VP 10212 has not yet been updated to contain that page. The VP will then go to MHT 10716 and MFT 10718 for the required information and, concurrently, WSM 10720 and ATU 10228 will be updated.

In regard to the above operations, each VP active in CS 10110 is assigned a Page Fault Frequency Time Factor (PFFT) which is used by CS 10110's VMM to adjust that VP's working set so that the interval between successive page faults for that VP lies in an optimum time range. This assists in ensuring CS 10110's VMM is operating most efficiently and allows CS 10110's VMM to be tuned as required.

The above discussions have assumed that the page being referenced, whether from a UID/O address, an AON/O address, or a Name, is resident in an object active in CS 10110. While an object need not have a page in MEM 10112 to be active, the object must be active to have a page in MEM 10112. A VP, however, may reference a page in an object not active in CS 10110. If such a reference is made, the object must be made active in CS 10110 before the page can be brought into MEM 10112. The result is an operation similar to the page fault operation described above. CS 10110 maintains an Active Object Manager (AOM), including Active Object Request Queue (AORQ) 10728, which are similar in operation to CS 10110's VMM and VMMRQ 10722. CS 10110's AOM and AORQ 10728 operate in conjunction with AOTHT 10710 and AOT 10712 to locate inactive objects and make them active by assigning them AON's and generating entries for them in AOTHT 10710, AOT 10712, and AOTA 10714.

Before a particular object can be made active in CS 10110, it must first be located in backing store (ED 10124). All objects on backing store are located through a Logical Allocation Unit Directory (LAUD) 10730, which is resident in backing store. An LAUD 10730 contains entries for each object accessible to the particular CS 10110. Each LAUD 10730 entry contains the information necessary to generate an AOT 10712 entry for that object. An LAUD 10730 is accessed through a UID/O address contained in CS 10110's VMM. A reference to an LAUD 10730 results in MEM 10112 frames being assigned to that LAUD 10730, and LAUD 10730 being transferred into MEM 10112. If an LAUD 10730 entry exists for the referenced inactive object, the LAUD 10730 entry is transferred into AOT 10712. At the next reference to a page in that object, AOT 10712 will provide the AON for that object but, because the page has not yet been transferred into MEM

10112, a page fault will occur. This page fault will be handled in the manner described above and the referenced page transferred into MEM 10112.

Having briefly described the structure and operation of CS 10110's Addressing Structure, including the relationship between UIDs, Names, AONs, and Physical Addresses and the mechanisms by which CS 10110 manages the available address space of MEM 10112, CS 10110's protection structures will be described next below.

E. CS 10110 Protection Mechanisms (Fig. 109)

Referring to Fig. 109, a schematic representation of Protection Mechanisms 10230 is shown. Protection Tables 10232 include Active Primitive Access Matrix (APAM) 10910, Active Subject Number Hash Table (ASNHT) 10912, and Active Subject Table (AST) 10914. Those portions of Protection Mechanism 10230 resident in FU 10120 include ASN Register 10916 and Protection Cache (PC) 10234.

As previously discussed, access rights to objects are arbitrated on the basis of subjects. A subject has been defined as a particular combination of a principle, Process, and Domain (PPD), each of which is identified by a corresponding UID. Each object has associated with it an Access Control List (ACL) 10918 containing an ACL Entry (ACLE) for each subject having access rights to that object.

When an object becomes active in CS 10110 (i.e., is assigned an AON) each ACLE in that object's ACL 10918 is written into APAM 10910. Concurrently, each subject having access rights to that object, and for which there is an ACLE in that object's ACL 10918, is assigned an Active Subject Number (ASN). These ASNs are written into ASNHT 10912 and their corresponding PPDs are written into AST 10914. Subsequently, the ASN of any subject requesting access to that object is obtained by hashing the PPD of that subject to obtain a PPD index into ASNHT 10912. ASNHT 10912 will in turn provide a corresponding ASN. An ASN may be used as an index into AST 10914. AST 10914 will provide the corresponding PPD, which may be compared to an original PPD to confirm the accuracy of the ASN.

As described above, APAM 10910 contains an ACL 10918 for each object active in CS 10110. The access rights of any particular active subject to a particular active object are determined by using that subject's ASN and that object's AON as indexes into APAM 10910. APAM 10910 in turn provides a 4 bit output defining whether that subject has Read (R) Write (W) or Execute (E) rights with respect to that object, and whether that particular entry is Valid (V).

ASN Register 10916 and PC 10234 are effectively acceleration mechanisms of Protection Tables 10232. ASN Register 10916 stores the ASN of a currently active subject while PC 10234 stores certain access right information for objects being used by the current process. PC 10234 entries are indexed by ASNs from ASN register 10916 and by a mode input from JP 10114. Mode input defines whether the current procedure intends to read, write, or execute with respect to a particular object having an entry in PC 10234. Upon receiving ASN and mode inputs, PC 10234 provides a go/nogo output indicating whether that subject has the access rights required to execute the intended operation with respect to that object.

In addition to the above mechanism, each procedure to which arguments may be passed in a cross-domain call has associated with it an Access Information Array (AIA) 10352, as discussed with reference to Virtual Processes 10212. A procedure's AIA 10352 states what access rights a calling procedure (subject) must have to a particular object (argument) before the called procedure can operate on the passed argument. CS 10110's protection mechanisms compare the calling procedures access rights to the rights required by the called procedure. This insures the calling procedure may not ask a called procedure to do what the calling procedure is not allowed to do. Effectively, a calling procedure can pass to a called procedure only the access rights held by the calling procedure.

Finally, PC 10234, APAM 10910, or AST 10914 faults (i.e., misses) are handled in the same manner as described above with reference to page faults in discussion of CS 10110's Addressing Mechanisms 10220. As such, the handling of protection misses will not be discussed further at this point.

Having briefly described structure and operation of CS 10110's Protection Mechanisms 10230, CS 10110's Micro-Instruction Mechanisms 10236 will be described next below.

F. CS 10110 Micro-Instruction Mechanism (Fig. 110)

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language is compiled into a corresponding S-Language program containing instructions expressed as SINs. CS 10110 provides a set, or dialect, of microcode instructions, referred to as S-Interpreters (SINTs) for each S-Language. SINTs interpret SINs and provide corresponding sequences of microinstructions for detailed control of CS 10110.

Referring to Fig. 110, a partial schematic representation of CS 10110's Micro-Instruction Mechanisms 10236 is shown. At system initialization all CS 10110 microcode, including SINTs and all machine assist microcode, is transferred from backing store to Micro-Code Control Store (mCCS) 10238 in MEM 10112. The Micro-Code is then transferred from mCCS 10238 to FU Micro-Code Structure (FUmC) 10240 and EU Micro-Code Structure (EUmC) 10242. EUmC 10242 is similar in structure and operation to FUmC 10240 and thus will be described in following detailed descriptions of CS 10110's structure and operation. Similarly, CS 10110 machine assist microcode will be described in following detailed discussions. The present discussion will concern CS 10110's S-Interpreter mechanisms.

CS 10110's S-Interpreters (SINTs) are loaded into S-Interpret Table (SITT) 11012, which is represented

in Fig. 110 as containing S-Interpreters 1 to N. Each SIT contains one or more sequences of micro-code; each sequence of microcode corresponds to a particular SIN in that S-Language dialect. S-Interpreter Dispatch Table (SDT) 11010 contains S-Interpreter Dispatchers (SDs) 1 to N. There is one SD for each SINT in SITT 11012, and thus a SD for each S-Language dialect. Each SD comprises a set of pointers. Each pointer in a particular SD corresponds to a particular SIN of that SD's dialect and points to the corresponding sequence of microinstructions for interpreting that SIN in that dialect's SIT in SITT 11012. In illustration, as previously discussed when a particular procedure is being executed the SIP for that procedure is transferred into one of mCR's 10366. That SIP points to the start of the SD for the SIT which is to be used to interpret the SINs of that procedure. In Fig. 110, the SIP in mCRs 10366 is shown as pointing to the start of SD2. Each S-Op appearing during execution of that procedure is an offset, relative to the start of the selected SD, pointing to a corresponding SD pointer. That SD pointer in turn points to the corresponding sequence of microinstructions for interpreting that SIN in the corresponding SIT in SITT 11012. As will be described in following discussions, once the start of a microcode sequence for interpreting an SIN has been selected, CS 10110 micromachine then proceeds to sequentially call the microinstructions of that sequence from SITT 11012 and use those microinstructions to control operation of CS 10110.

G. Summary of Certain CS 10110 Features and Alternate Embodiments

The above Introductory Overview has described the overall structure and operation and certain features of CS 101, that is, CS 10110. The above Introduction has further described the structure and operation and further features of CS 10110 and, in particular, the physical implementation and operation of CS 10110's information, control, and addressing mechanisms. Certain of these CS 10110 features are summarized next below to briefly state the basic concepts of these features as implemented in CS 10110. In addition, possible alternate embodiments of certain of these concepts are described.

First, CS 10110 is comprised of a plurality of independently operating processors, each processor having a separate microinstruction control. In the present embodiment of CS 10110, these processors include FU 10120, EU 10122, MEM 10112 and IOS 10116. Other such independently operating processors, for example, special arithmetic processors such as an array processor, or multiple FU 10120's, may be added to the present CS 10110.

In this regard, MEM 10112 is a multiport processor having one or more separate and independent ports to each processor in CS 10110. All communications between CS 10110's processors are through MEM 10112, so that MEM 10112 operates as the central communications node of CS 10110, as well as performing memory operations. Further separate and independent ports may be added to MEM 10112 as further processors are added to CS 10110. CS 10110 may therefore be described as comprised of a plurality of separate, independent processors, each having a separate microinstruction control and having a separate and independent port to a central communications and memory node which in itself is an independent processor having a separate and independent microinstruction control. As will be further described in a following detailed description of MEM 10112, MEM 10112 itself is comprised of a plurality of independently operating processors, each performing memory related operations and each having a separate microinstruction control. Coordination of operations between CS 10110's processors is achieved by passing "messages" between the processors, for example, SOP's and descriptors.

CS 10110's addressing mechanisms are based, first, upon UID addressing of objects. That is, all information generated for use in or by operation of a CS 10110, for example, data and procedures, is structured into objects and each object is assigned a permanent UID. Each UID is unique within a particular CS 10110 and between all CS 10110's and is permanently associated with a particular object. The use of UID addressing provides a permanent, unique addressing means which is common to all CS 10110's, and to other computer systems using CS 10110's UID addressing.

Effectively, UID addressing means that the address (or memory) space of a particular CS 10110 includes the address space of all systems, for example disc drives or other CS 10110s, to which that particular CS 10110 has access. UID addressing allows any process in any CS 10110 to obtain access to any object in any CS 10110 to which it has physical access, for example, another CS 10110 on the other side of the world. This access is constrained only by CS 10110's protection mechanism. In alternate embodiments of CS 10110, certain UIDs may be set aside for use only within a particular CS 10110 and may be unique only within that particular CS 10110. These reserved UIDs would, however, be a limited group known to all CS 10110 systems as not having uniqueness between systems, so that the unique object addressing capability of CS 10110's UID addressing is preserved.

As previously stated, AONs and physical descriptors are presently used for addressing within a CS 10110, effectively as shortened UIDs. In alternate embodiments of CS 10110, other forms of AONs may be used, or AONs may be discarded entirely and UIDs used for addressing within as well as between CS 10110s.

CS 10110's addressing mechanisms are also based upon the use of descriptors within and between CS 10110s.

Each descriptor includes an AON or UID field to identify a particular object, an offset field to specify a bit granular offset within the object, and a length field to specify a particular number of bits beginning at the specified offset. Descriptors may also include a type, or format field identifying the particular format of the data referred to by the descriptor. Physical descriptors are used for addressing MEM 10112 and, in this

case, the AON or UID field is replaced by a frame number field referring to a physical location in MEM 10112.

As stated above, descriptors are used for addressing within and between the separate, independent processors (FU 10120, EU 10122, MEM 10112, and IOS 10116) comprising CS 10110, thereby providing common, system wide bit granular addressing which includes format information. In particular, MEM 10112 responds to the type information fields of descriptors by performing formatting operations to provide requestors with data in the format specified by the requestor in the descriptor. MEM 10112 also accepts data in a format specified in a descriptor and reformats that data into a format most efficiently used by MEM 10112 to store the data.

As previously described, all operands are referred to in CS 10110 by Names wherein all Names within a particular S-Language dialect are of a uniform, fixed size and format. A K value specifying Name size is provided to FU 10120, at each change in S-Language dialect, and is used by FU 10120 in parsing Names from the instruction stream. In an alternate embodiment of CS 10110, all Names are the same size in all S-Language dialects, so that K values, and the associated circuitry in FU 10120's parser, are not required.

Finally, in descriptions of CS 10110's use of SOPs, FU 10120's microinstruction circuitry was described as storing one or more S-Interpreters. S-Interpreters are sets of sequences of microinstructions for interpreting the SOPs of various S-Language dialects and providing corresponding sequences of microinstructions to control CS 10110. In an alternate embodiment of CS 10110, these S-Interpreters (SITT 11012) would be stored in MEM 10112. FU 10120 would receive SOPs from the instruction stream and, using one or more S-Interpreter Base Pointers (that is, architectural base pointers pointing to the SITT 11012 in MEM 10112), address the SITT 11012 stored in MEM 10112. MEM 10112 would respond by providing, from the SITT 11012 in MEM 10112, sequences of microinstructions to be used directly in controlling CS 10110. Alternately, the SITT 11012 in MEM 10112 could provide conventional instructions usable by a conventional CPU, for example, Fortran or machine language instructions. This, for example, would allow FU 10120 to be replaced by a conventional CPU, such as a Data General Corporation Eclipse®.

Having briefly summarized certain features of CS 10110, and alternate embodiments of certain of these features, the structure and operation of CS 10110 will be described in detail below.

2. DETAILED DESCRIPTION OF CS 10110 MAJOR SUBSYSTEMS

(Figs. 201—206, 207—274)

Having previously described the overall structure and operation of CS 10110, the structure and operation of CS 10110's major subsystems will next be individually described in further detail. As previously discussed, CS 10110's major subsystems are, in the order in which they will be described, MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118. Individual block diagrams of MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118 are shown in, respectively, Figures 201 through 205. Figures 201 through 205 may be assembled as shown in Fig. 206 to construct a more detailed block diagram of CS 10110 corresponding to that shown in Fig. 101. For the purposes of the following descriptions, it is assumed that Figs. 201 through 205 have been assembled as shown in Fig. 206 to construct such a block diagram. Further diagrams will be presented in following descriptions as required to convey structure and operation of CS 10110 to one of ordinary skill in the art.

As previously described, MEM 10112 is an intelligent, priortizing memory having separate and independent ports MIO 10128 and MJP 10140 to, respectively, IOS 10116 and JP 10114. MEM 10112 is shared by and is accessible to both JP 10114 and IOS 10116 and is the primary memory of CS 10110. In addition, MEM 10112 is the primary path for information transferred between the external world (through IOS 10116) and JP 10114.

As will be described further below, MEM 10112 is a two-level memory providing fast access to data stored therein. MEM 10112 first level is comprised of a large set of random access arrays and MEM 10112 second level is comprised of a high speed cache whose operation is generally transparent to memory users, that is JP 10114 and IOS 10116. Information stored in MEM 10112, in either level, appears to be bit addressable to both JP 10114 and IOS 10116. In addition, MEM 10112 presents simple interfaces to both JP 10114 and IOS 10116. Due to a high degree of pipe lining (concurrent and overlapping memory operations) MEM 10112 interfaces to both JP 10114 and IOS 10116 appear as if each HP 10116 and IOS 10116 have full access to MEM 10112. This feature allows data transfer rates of up to, for example, 63.6 megabytes per second from MEM 10112 and 50 megabytes per second to MEM 10112.

In the following descriptions, certain terminology used on those descriptions will be introduced first, followed by description of MEM 10112 physical organization. Then MEM 10112 port structures will be described, followed by descriptions of MEM 10112's control organization and control flow. Next, MEM 10112's interfaces to JP 10114 and IOS 10116 will be described. Following these overall descriptions the major logical structures of MEM 10112 will be individually described, starting at MEM 10112's interfaces to JP 10114 and IOS 10116 and proceeding inwardly to MEM 10112's first (or bulk) level of data stored. Finally, certain features of MEM 10112 microcode control structure will be described.

A. MEM 10112 (Figs. 201, 206, 207—237)

a. Terminology

Certain terms are used throughout the following descriptions and are defined here below for reference

by the reader.

A word is 32 bits of data

A byte is 8 bits of data

A block is 128 bits of data (that is, 4 words).

A block is always aligned on a block boundary, that is the low order 7 bits of logical or physical address are zero (see Chapter 1, Sections A.f and D. Descriptions of CS 10110 Addressing).

The term aligned refers to the starting bit address of a data item relative to certain address boundaries. A starting bit address is block aligned when the low order 7 bits of starting bit address are equal to zero, that is the starting bit address falls on a boundary between adjacent blocks. A word align starting bit address means that the low order 5 bits of starting bit address are zero, the starting bit address points to a boundary between adjacent words. A byte aligned starting bit address means that the low order 3 bits of starting bit address are zero, the starting bit address points to a boundary between adjacent bytes.

Bit granular data has a starting bit address falling within a byte, but not on a byte boundary, or the address is aligned on a byte boundary but the length of the data is bit granular, that is not a multiple of 8 bits.

b. MEM 10112 Physical Structure (Fig. 201)

Referring to Fig. 201, a partial block diagram of MEM 10112 is shown. Major functional units of MEM 10112 are Main Store Bank (MSB) 20110, including Memory Arrays (MA's) 20112, Bank Controller (BC) 20114, Memory Cache (MC) 20116, including Bypass Write File (BYF) 20118, Field Isolation Unit (FIU) 20120, and Memory Interface Controller (MIC) 20122.

MSB 20110 comprises MEM 10112's first or bulk level of storage. MSB 20110 may include from one to, for example, 16 MA 20112's. Each MA 20112 may have a storage capacity, for example, 256 K-byte, 512 K-byte, 1 M-byte, or 2 M-bytes of storage capacity. As will be described further below, MA 20112's of different capacities may be used together in MSB 20110. Each MA 20112 has a data input connected in parallel to Write Data (WD) Bus 20124 and a data output connected in parallel to Read Data (RD) Bus 20126. MA's 20112 also have control and address ports connected in parallel to address and control (ADCTL) Bus 20128. In particular, Data Inputs 20124 of Memory Arrays 20112 are connected in parallel to Write Data (WD) Bus 20126, and Data Outputs 20128 of Memory Arrays 20112 are connected in parallel to Read Data (RD) Bus 20130. Control Address Ports 20132 of Memory Arrays 20112 are connected in parallel to Address and Control (ADCTL) Bus 20134.

Data Output 20136 of Bank Controller 20114 is connected to WD Bus 20126 and Data Input 20138 of BC 20114 is connected to RD Bus 20130. Control and Address Port 20140 of BC 20114 is connected to ADCTL Bus 20134. BC 20114's Data Input 20142 is connected to MC 20116's Data Output 20144 through Store Back Data (SBD) Bus 20146. BC 20114's Store Back Address Input 20148 is connected to MC 20116 Store Back Address Output 20150 through Store Back Address (SBA) Bus 20152. BC 20114's Read Data Output 20154 is connected to MC 20116's Read Data Input 20156 through Read Data Out (RDO) Bus 20158. BC 20114's Control Port 20160 is connected to Memory Control (MCNTL) Bus 20164.

MC 20116 has Output 20166 connected to MIO Bus 10131 through MIO Port 10128, and Port 20168 connected to MOD Bus 10144 through MJP Port 10140. Control Port 20170 of MC 20116 is connected to MCNTL Bus 20164. Input 20172 of BYF 20118 is connected to IOM Bus 10130 through MIO Port 10128, and Output 20176 is connected to SBD Bus 20146 through Bypass Write In (BWI) Bus 20178.

Finally, FIU 20120 has an Output 20180 and an Input 20182 connected to, respectively, MIO Bus 10129 and IOM Bus 10130 through MIO Port 10128. Input 20184 and Port 20186 are connected to, respectively, JPD Bus 10142 and MOD Bus 10144 through MJP Port 10140. Control Port 20188 is connected to MCNTL Bus 20164. Referring finally to MIC 20122, MIC 20122 has Control Port 20190 and Input 20192 connected to, respectively, IOMC Bus 10131 and IOM Bus 10130 through MIO Port 10128. Control Port 20194 and Input 20196 are connected, respectively, to JPMC Bus 10147 and Physical Descriptor (PD) Bus 10146 through MJP Port 10140. Control Port 20198 is connected to MCNTL Bus 20164.

c. MEM 10112 General Operation

Referring first to MEM 10112's interface to IOS 10116, this interface includes MIO Bus 10129, IOM Bus 10130, and IOMC Bus 10131. Read and Write Addresses and data to be written into MEM 10112 are transferred from IOS 10116 to MEM 10112 through IOM Bus 10130. Data read from MEM 10112 is transferred to IOS 10116 through MIO Bus 10129. IOMC 10131 is a Bi-directional Control bus between MEM 10112 and IOS 10116 and, as described further below, transfers control signals between MEM 10112 and IOS 10116 to control transfer of data between MEM 10112 and IOS 10116.

MEM 10112's interface to JP 10114 is MJP Port 10140 and includes JPD Bus 10142, MOD Bus 10144, PD Bus 10146, and JPMC Bus 10147. Physical descriptors, that is MEM 10112 physical read and write addresses, are transferred from JP 10114 to MEM 10112 through PD Bus 10146. S Ops, that is sequences of S Instructions and operand names, are transferred from MEM 10112 to JP 10114 through MOD Bus 10144 while data to be written into MEM 10112 from JP 10114 is transferred from JP 10114 to MEM 10112 through JPD Bus 10142. JPMC Bus 10147 is a By-directional Control bus for transferring command and control signals between MEM 10112 and JP 10114 for controlling transfer of data between MEM 10112 and JP 10114. As will be described further below, MJP Port 10140, and in particular MOD Bus 10144 and PD Bus

10146, is generally physically organized as a single port that operates as a dual port. In a first case, MJP Port 10140 operates as a Job Processor Instruction (JI)-Port for transferring S Ops from MEM 10112 to JP 10114. In a second case, MOD 10144 and PD 10146 operate as a Job Processor Operand (JO) Port for transfer of operands, from MEM 10112 to JP 10114, while JPD Bus 10142 and PD Bus transfer operands from JP 10114 to MEM 10112.

Referring to MSB 20110, MSB 20110 contains MEM 10112's first, or bulk, level of storage capacity. MSB 20110 may contain from one to, for example, 16 MA's 20112. Each MA 20112 contains a dynamic, random access memory array and may have a storage capacity of, for example 256 Kilo-bytes, 512 Kilo-bytes, 1 Mega-bytes, or 2 Mega-bytes. MEM 10112 may therefore have a physical capacity of up to, for example, 16 Mega-bytes of bulk storage. As will be described further below. MA 20112's of different capacity may be used together in MSB 20110, for example, four 2 Mega-byte MA 20112's and four 1 Megabyte MA 20112's.

BC 20114 controls operation of MA's 20112 and is the path for transfer of data to and from MA's 20112. In addition, BC 20114 performs error detection and correction on data transferred into and out of MA's 20112, refreshes data stored in MA's 20112, and, during a refresh operations, performs error detection and correction of data stored in MA's 20112.

MC 20116 comprises MEM 10112's second, or cache, level of storage capacity and contains, for example 8 Kilo-bytes of high speed memory. MC 20116, including BYF 20118, is also the path for data transfer between MSB 20110 (through BC 20114) and JP 10114 and IOS 10116. In general, all read and write operations between JP 10114 and IOS 10116 are through MC 20116. IOS 10116 may, however, perform read and write operations of complete blocks by-passing MC 20116. Block write operations from IOS 10116 are accomplished through BYF 20118 while block read operations are performed through a data transfer path internal to MC 20116 and shown and described below. All read and write operations between MEM 10112 and JP 10114, however, must be performed through the cache internal to MC 20116, as will be shown and described further below.

As also shown and described below, FIU 20120 includes write data registers for receiving data to be written into MEM 10112 from JP 10114 and IOS 10116, and circuitry for manipulating data read from MSB 20110 so that MEM 10112 appears as a bit addressable memory. FIU 20120, in addition to providing bit addressability of MEM 10112, performs right and left alignment of data, zero fill of data, sign extension operations, and other data manipulation operations described further below. In performing these data manipulation operations on data read from MEM 10112 to JP 10114, MOD Bus 10144 is used as a data path internal to MEM 10112 for transferring of data from MC 20116 to FIU 20120, and from FIU 20120 to MC 20116. That is, data to be transferred to JP 10114 is read from MC 20116, transferred through MOD Bus IC144 to FIU 20120, manipulated by FIU 20120, and transferred from FIU 20120 to JP 10114 through MOD Bus 10144.

MIC 20122 contains circuitry controlling operation of MEM 10112 and, in particular, controls MEM 10112's interface with JP 10114 and IOS 10116. MIC 20122 receives MEM 10112 read and write request, that is read and write addresses through PD Bus 10146 and IOM Bus 10130 and control signals through JPMC Bus 10147 and IOMC Bus 10131, and provides control signals to BC 20114, MC 20116, and FIU 20120 through MCNTL Bus 20164.

Having described the overall structure and operation of MEM 10112, the structure and operation of MEM 10112's Port, MIO Port 10128, and MJP Port 10140, will be described next, followed by descriptions of MEM 10112's control structure and the control and flow of MEM 10112 read and write requests.

d. MEM 10112 Port Structure

MEM 10112 port structure is designed to provide a simple interface to JP 10114 and IOS 10116. While providing fast and flexible operation in servicing MEM 10112 read and write requests from JP 10114 and IOS 10116. In this regard, MEM 10112, as will be described further below, may handle up to 4 read and write requests concurrently and up to, for example, a 63.6 M-byte per second data rate. In addition MEM 10112 is capable of performing bit granular addressing, block read and write operations, and data manipulations, such as alignment and filling, to enable JP 10114 and IOS 10116 to operate most efficiently.

MEM 10112 effectively services requests from three ports. These ports are MIO Port 10128 to IOS 10116, hereafter referred to as IO Port, and JI and JO Ports, described above, to JP 10114. These three ports share the entire address base of MEM 10112, but IOS 10116, for example, may be limited from making full use of MEM 10112's address space. Each port has a different set of allowed operations. For example, JO Port can use a bit granular addresses but can reference only 32 bits of data on each request. JI Port can make read requests only to word align 32 bit data items. IO Port may reference bit granular data, and, as described further below, may read or write up to 16 bytes on each read or write request. The characteristics of each of these ports will be discussed next below.

1. IO Port Characteristics

IOS 10116 may access MEM 10112 in either of two modes. The first mode is block transfers by-passing or through the cache in MC 20116, and the second is non-block transfer through the cache and MC 20116.

Block by-passes may occur for both read and write operations. A read or write operation is eligible for a block by-pass if the data is on block boundaries, is 16 bytes long, and the read or write request is not accompanied by a control signal indicating that an encache (load into MC 20116's cache) operation is to be

performed. A by-pass operation takes place only if the block address, that is the physical address of the block in MEM 10112 does not address a currently encached block, that is the block is not present in MC 20116's cache. If the block is encached in MC 20116's cache, the read or write transfer is to MC 20116's cache.

Partial block references, that is non-full block transfers will go through MC 20116's cache. If a cache miss occurs, that is the reference data is not present in MC 20116's cache, MEM 10112's control structures transfer the data to or from MSB 20110 and update MC 20116's cache. It should be noted that partial blocks may be as short as one byte, or up to 15 bytes long. A starting byte address may be anywhere within a block, but the partial block's length may not cross a block boundary.

Bit length transfers, that is transfers of data items having a length of 1 to 16 bits and not a multiple of a byte, or where address is not on a byte boundary, go through MC 20116's cache. These operations may cross byte, word, or block boundaries but may not cross page boundaries. These specific operations requested by IO port determines whether a read or write request is a partial block or bit length transfer.

2. JO Port Characteristics

All read or write requests from JO Port must go through MC 20116's cache; by-pass operations may not be performed. The data transferred between MEM 10112 and JP 10114 is always 32 bits in length but, of the 32 bits passed, from zero to 32 bits may be valid data. JP 10114 determines the location of valid data within the 32 bits by referring to certain FIU specification bits provided as part of the read or write request. As will be described further below, FIU specification bits, and other control bits, are provided to MIC 20122 by JP 10114 through JPMC Bus 10147 when each read or write request is made.

While MEM 10112 does not perform block by-pass operations to JP 10114, MEM 10112 may perform a cache read-through operation. Such operations occur on a JP 10114 read request wherein the requested data is not present in MC 20116's cache. If the JP 10114 read request is for a full word, which is word aligned, MEM 10112's Load Manager, discussed below, transfers the requested data directly to JP 10114 while concurrently loading the requested data into MC 20116's cache. This operation is referred to as a "hand-off" operation. These operations may also be performed by IO Port for 16 bit half words aligned on the right hand half word of a 32 bit word, or if a full block is handed left and loaded into MC 20116's cache.

3. JI Port Characteristics

All JI Port requests are satisfied through MC 20116's cache; MEM 10112 does not perform by-pass operations to JI Port. JI Port requests are always read requests for full-word aligned words and are handed off, as described above, if a cache miss occurs. In most other respects, JI Port requests are similar to JO Port requests.

Having described the overall structure and operation of MEM 10112, including MEM 10112's input and output ports to JP 10114 and IOS 10116, MEM 10112's control structure will be described next below.

e. MEM 10112 Control Structure and Operation (Fig. 207)

Referring to Fig. 207, a more detailed block diagram of MIC 20116 is shown. Fig. 207 will be referred to in conjunction with Fig. 201 in the following discussion of MEM 10112's control structure.

1. MEM 10112 Control Structure

Referring first to Fig. 207, MCNTL Bus 20164 is represented as including MCNTL-BC Bus 20164A, MCNTL-MC Bus 20164B, and MCNTL-FIU Bus 20164C. Buses 20164A, 20164B, and 20164C are branches of MCNTL Bus 20164 connected to, respectively, BC 20114, MC 20116, and FIU 20120. Also represented in Fig. 207 are PD Bus 10146 and JPMC Bus 10147 to JP 10114, and IOM Bus 10130 and IOMC Bus 10131 to IOS 10116.

JO Port Address Register (JOPAR) 20710 and JI Port Address Register (JIPAR) 20712 have inputs connected from PD Bus 10146. IO Port Address Register (IOPAR) 20714 has an input connected from IOM Bus 10130. Port Control Logic (PC) 20716 has a bi-directional input/outputs connected from JPC 10147 and IOMC Bus 10131. By-pass Read/Write Control Logic (BR/WC) 20718 has a bidirectional input/output connected from IOMC Bus 10131.

Outputs of JOPAR 20710, JIPAR 20712, and IOPAR 20714 are connected to inputs of Port Request Multiplexer (PRMUX) 20720 through, respectively, Buses 20732, 20734, 20736. PRMUX 20720's output in turn is connected to Bus 20738. Branches of Bus 20738 are connected to inputs of Load Pointers (LP) 20724, Miss Control (MISSC) 20726, and Request Manager (RM) 20722, and to Buses MCNTL-MC 20164B and MCNTL-FIU 20164C.

Outputs of PC 20716 are connected to inputs of JOPAR 20710, JIPAR 20712, IOPAR 20714, PRMUX 20720, and LP 20724 through Bus 20738. Bus 20740 is connected between an input/output of PC 20716 and in input/output of RM 20722.

An output of BR/WC 20718 is connected to MCNTL-MC Bus 20164B through Bus 20742. Inputs of BR/WC 20718 are connected from outputs of RM 20722 and Read Queue (RQ) 20728 through, respectively, Buses 20744 and 20746.

RM 20722 has outputs connected to MCNTL-BC Bus 20164A, MCNTL-FIU Bus 20164C, and input of MISSC 20726, and an input of LP 20724 through, respectively, Buses 20748, 20750, 20752, and 20754.

MISSC 20726's output is connected to MCNTL-BC Bus 20164A. Outputs of LP 20724 are connected to MCNTL-MC Bus 20164B and to an input of LM 20730 through, respectively, Buses 20756 and 20758. RQ 20728's input is connected from MCNTL-MC Bus 20164B through Bus 20760 and RQ 20728 has outputs connected to an input of LP 20724, through Bus 20762, and as previously described to an input of BR/WC 20718 through Bus 20746. Finally, LM 20730's output is connected to MCNTL-MC Bus 20164B through Bus 20764.

Having described the structure of MIC 20716 with reference to Fig. 207, and having previously described the structure of MEM 10112 with reference to Fig. 201, MEM 10112's control structure operation will next be described with reference to both figures 201 and 207.

2. MEM 10112 Control Operation

Referring first to Fig. 207, JOPAR 20710, JIPAR 20712, and IOPAR 20714 are, as previously described, connected from PD Bus 10146 from JP 10114 and IOM Bus 10130 from IOS 10116. JPAR 20710, JIPAR 20712, and IOPAR 20714 receive read and write request addresses from JP 10114 and IOS 10116 and store these addresses for subsequent service by MEM 10112. As will be described further below, these address inputs from JP 10114 and IOS 10116 include FIU information specifying what data manipulation operations must be performed by FIU 20120 before requested data is transferred to the requestor or written into MEM 10112, information regarding the destination data read from MEM 10112 is to be provided to, information regarding the type of operation to be performed by MEM 10112, and information regarding operand length. Request address information received and stored in JOPAR 20710, JIPAR 20712, and IOPAR 20714 is retained therein until MEM 10112 has initiated service of the corresponding requests. MEM 10112 will accept further request address information into a given port register only after a previous request into that port has been serviced or aborted. Address information outputs from JOPAR 20710, JIPAR 20712, and IOPAR 20714 is transferred through PRMUX 20720 to Bus 20738 and from there to RM 20722, MC 20116, and FIU 20120 as service of individual requests is initiated. As will be described below, this address information will be transferred through PRMUX 20720 and Bus 20738 to LP 20724 for use in servicing a cache miss upon occurrence of a MC 20116 miss.

PC 20716 receives command and control signals pertinent to each requested memory operation from JP 10114 and IOS 10116 through JPMC Bus 10147 and IOSC Bus 10131. PC 20716 includes request arbitration logic and port state logic. Request arbitration logic determines the sequence in which IO, JI, JO ports are serviced, and when each port is to be serviced. In determining the sequence of port service, request arbitration logic uses present port state information for each port from the port state logic, information from JPMC Bus 10147 and IOMC Bus 10131 regarding each incoming request, and information from RM 20722 concerning the present state of operation of MEM 10112. Port state logic selects each particular port to be serviced and, by control signals through Bus 20738, enables transfer of each port's request address information from JOPAR 20710, JIPAR 20712, and IOPAR 20714 through PRMUX 20720 to Bus 20738 for use by the remainder of MEM 10112's control logic in servicing the selected port. In addition to request information received from JP 10114 and IOS 10116 through JPMC Bus 10147 and IOMC Bus 10131, port state logic utilizes information from RM 20722 and, upon occurrence of a cache miss, from LM 20730 (for clarity of presentation, this connection is not represented in Fig. 207). Port state logic also controls various port state flag signals, for example port availability signals, signals indicating valid requests, and signals indicating that various ports are waiting service.

RM 20722 controls execution of service for each request. RM 20722 is a microcode controlled "micromachine" executing programs called for by requested MEM 10112 operations. Inputs of RM 20722 include request address information from IOPAR 20714, JIPAR 20212, and JOPAR 20210, including information regarding the type of MEM 10112 operation to be performed in servicing a particular request, interrupt signals from other MEM 10112 control elements, and, for example, start signals from PC 20716's request arbitration logic. RM 20722 provides control signals to FIU 20120, MC 20116, and most other parts of MEM 10112's control structure.

Referring to Fig. 201, MC 20116's cache is, for example, an 8 Kilo-byte, four set associative cache used to provide rapid access to a subset of data stored in MSB 20110. The subset of MSB 20110 data stored in MC 20116's cache at any time is the data most recently used by JP 10114 or IOS 10116. MC 20116's cache, described further below, includes tag store comparison logic for determining encached addresses, a data store containing corresponding encached data, and registers and logic necessary to up-date cache contents upon occurrence of a cache miss. Registers and logic for servicing cache misses includes logic for determining the least recently used cache entry and registers for capture and storage of information regarding missed cache references, for example modify bits and replacement page numbers. Inputs to MC 20116 are provided from RM 20722, LM 20730 (discussed further below), FIU 20120, MSB 20110 (through BC 20114), LP 20724 (described further below) and address information from PRMUX 20720. Outputs of MC 20116 include data and go to FIU 20120 (through MOD Bus 10144), the data requestors (JP 10114 and IOS 10116), and a MC 20116 Write Back File (described further below).

As previously described, FIU 20120 includes logic necessary to make MEM 10112 appear bit addressable. In addition, FIU 20120 includes logic for performing certain data manipulation operations as required by the requestors (JP 10114 or IOS 10116). Data is transferred into FIU 20120 from MC 20116 through that portion of MOD Bus 10144 internal to MEM 10112, is manipulated as required, and is then

transferred to the requestor through MOD Bus 10144 or MIO Bus 10129. In the case of writes requiring read-modify-write of encached data, the data is transferred back to MC 20116 through MOD Bus 10144 after manipulation. In general, data manipulation operations include locating requested data onto selected MOD Bus 10144 or MIO Bus 10139 lines and filling unused bus lines as specified by the requestor. Data inputs to FIU 20120 may be provided from MC 20116 or JP 10114 through MOD Bus 10144 or from IOS 10116 through IOM Bus 10130. Data outputs from FIU 20120 may be provided to MC 20116, JP 10114, or IOS 10116 through these same buses. Control information is provided to FIU 20120 from RM 20722 through Bus 20748 and MCNTL-FIU Bus 20164C. Address information may be provided to FIU 20120 from JOPAR 20710, JIPAR 20712, or IOPAR 20714 through PRMUX 20720, Bus 20738, and MCNTL-FIU Bus 20164C.

Returning to Fig. 207, MISSC 20726 is used in handling MC 20116 misses. In the event of a request referring to data not in MC 20116's cache, MISSC 20726 stores block address of the reference and type of operation to be performed, this information being provided from an address register in MC 20116 and from RM 20722. MISSC 20726 utilizes this information in generating a command to BC 20114, through MCNTL-BC Bus 20164A, for a data read from MSB 20110 to obtain the referenced data. BC 20114 places this command in a queue, or register, and subsequently executes the commanded read operation. MISSC 20726 also generates an entry into RQ 20728 (described further below) indicating the type of operation to be performed when referenced data is subsequently read from MSB 20110.

RQ 20728 is, for example, a three-level deep queue storing information indicating operations associated with data being read from MSB 20110. Two kinds of operation may be indicated: block by-pass reads and cache loads. If a cache load is specified, that is a read and store to MC 20116's cache, is indicated, RM 20722 is interrupted and forced to place other MEM 10112 operations in idle until cache load is completed. A block by-pass read operation results in by-pass read control (described below) assuming control of the data from MSB 20110. Inputs to RQ 20728 are control signals from RM 20752, MISSC 20726, and BC 20114. RQ 20728 provides control outputs to LP 20724 (described below) LM 20730 (described below) RM 20722, and by-pass read control (described below).

LP 20724 is a set of registers for storing information necessary for servicing MC 20116 misses that result in order to load MC 20116's tag store. LM 20730 uses this information when data stored in MSB 20110 and read from MSB 20110 to service a MC 20116 cache miss, becomes available through BC 20114. Inputs to LP 20724 include the address of the missing reference, provided from JOPAR 20710, JIPAR 20712, or IOPAR 20714 through PRMUX 20720 and Bus 20738, commands from RM 20722, and a control signal from RQ 20728. LP 20724 outputs include addresses of missed references to MC 20116, through Bus 20756 and MNCTL-MC 20164B, and command signals to LM 20730 and BR/WC 20718.

LM 20730, referred to above, controls loading of MC 20116's cache with data from MSB 20110 after occurrence of a cache miss. RQ 20728, referred to above, indicates, for each data read from MSB 20110, whether the data read is the result of a MC 20116 cache miss. If the data is read from MSB 20110 as a result of a cache miss, LM 20730 proceeds to issue a sequence of control signals for loading the data from MSB 20110 and its associated address into MC 20116's cache. This data is transferred into MC 20116's cache data store while the block address, from LP 20724 is transferred into the tag store (described in the following discussion) of MC 20116's cache. If the transfer of data into MC 20116's cache replaces data previously resident in that cache, and that previous data is "dirty", that is has been written into so as to be different from an original copy of the data stored on MSB 20110, the modified data resident in MC 20116's cache must be written back into MSB 20110. This operation is performed through a Write Back File contained in MC 20116 and described below. In the event of such an operation, LM 20730 initiates a write back operation by MC 20116 and BC 20114, also as described below.

As will be described further in a following description, all MC 20116 cache load operations are full 4 word blocks. A request resulting in a MC 20116 cache miss may result in a "hand-off", that is a read operation of a full 4 word block. Handoff operations also may be of single 32 bit words wherein a 32 bit word aligned word is transferred from JP 10114 or a 16 bit operand aligned on the right half-word is transferred from IOS 10116. In such a handoff operation, LM 20730 will send a valid request signal to the requesting port and a handoff operation will be performed. Otherwise, a waiting signal will be sent to the requesting port and the request will re-enter the priority queue of PC 20716 for subsequent execution. To accomplish these operations, LM 20730 receives input from RQ 20728, (not shown in Fig. 207 for clarity of presentation) and LP 20724. LM 20730 provides outputs to port state logic of PC 20716, to MC 20116, MC 20116's Write Back File and MC 20116's Write Back Address Register and to BC 20114.

Referring to Fig. 201, as previously discussed IOS 20116 may request a full block write operation directly to MSB 20110. Such a by-pass write request may be honored if the block being transferred is not encached in MC 20116's cache. In such a case, RM 20722 will initiate the transfer setting up By-pass Write Control logic in BR/WC 20718, and may then pass control of the operation over to BR/WC 20718's By-Pass Write Control logic for completion. By-pass Write Control may then accept the remaining portion of the data block from IOS 10116, generating appropriate hand shaking signals through IOMC Bus 10131, and load the data block into BYF 20118 and MC 20116. MISSC 20726 will provide a by-pass write command to BC 20114, through MNCTL-PC Bus 20164A. BC 20114 will then transfer the data block from BYF 20118 and into MA's 20112 and MSB 20110.

As previously described, BYF 20118 receives data from IOM Bus 10130 and provides data output to BC 20114 through BWY Bus 20178 and SBD Bus 20146. BYF 20118 is capable of simultaneously accepting data

from IOM Bus 10130 while reading data out to BC 20114. Control of writing data into BYF 20118 is provided from BR/WC 20718's By-Pass Write Control logic.

IOS 10116 may, as previously described, request a full block read operation by-passing MC 20116's cache. In such a case, BR/WC 20718's by-pass read control handles data transfer to IOS 10116 and generates required hand shaking signals to IOS 10116 through IOMC Bus 10131. The data path for by-pass read operations is through a data path internal to MC 20116, rather than through BYF 20118. This internal data path is RDO Bus 20158 to MIO Bus 10129.

As previously described, BC 20114 manages all data transfers to and from MA's 20112 in MSB 20110. BC 20114 receives requests for data transfers from RM 20722 in an internal queue register. All data transfers to and from MSB 20110 are full block transfers with block aligned addresses. On data write operations, BC 20114 receives data from BWF 20118 or from MC 20116's Write Back File and transfers the data into MA's 20112. During read operations, BC 20114 fetches the data block from MA's 20112 and places the data block on RDO Bus 20158 while signalling to MIC 20122 that the data is available. As described above, MIC 20122 tracks and controls transfer of data and BYF 20118, MC 20116, and MC 20116's Write Back File, and directs data read from MSB 20110 to the appropriate destination, MC 20116's Data Store, JP 10114, or IOS 10116.

In addition to the above operations, BC 20114 controls refresh of MA's 20112 and performs error detection and correction operations. In this regard, BC 20114 performs two error detection and correction operations. In the first, BC 20114 detects single and double bit errors in data read from MSB 20110 and corrects single bit errors. In the second, BC 20114 reads data stored in MA's 20112 during refresh operations and performs single bit error detection. Whenever an error is detected, during either read operations or refresh operations, BC 20114 makes a record of that error in an error log contained in BC 20114 (described further in a following description). Both JP 10114 and IOS 10116 may read BC 20114's error log, and information from BC 20114's error log may be recorded in a CS 10110 maintenance log and to assist in repair and trouble shooting of CS 10110. BC 20114's error log may be addressed directly by RM 20722 and data from BC 20114's error log is transferred to JP 10114 or IOS 10116 in the same manner as data stored in MSB 20110.

Referring finally to MA's 20112, each MA 20112 contains an array of dynamic semiconductor random access memories. Each MA 20112 may contain 256 Kilo-bytes, 512 Kilo-bytes, 1 Mega-bytes, or 2 Mega-bytes of data storage. The storage capacity of each MA 20112 is organized as segments of 256 Kilo-bytes each. In addressing a particular MA 20112, BC 20114 selects that particular MA 20112 as will be described further below. BC 20114 concurrently selects a segment within that MA 20112, and a block of four words within that segment. Each word may comprise 39 bits of information, 32 bits of data and 7 bits of error correcting code. The full 39 bits of each MA 20112 word are transferred between BC 20114 and MA's 20112 during each read and write operation. Having briefly described the general structure and operation of MEM 10112, certain types of operations which may be performed by MEM 10112 will be described next below.

f. MEM 10112 Operations

MEM 10112 may perform two general types of operation. The first type are data transfer operations and the second type are memory maintenance operations. Data transfer operations may include read, write, and read and set. Memory maintenance operations may include read error log, repair block, and flush cache. Except during a flush cache operation, the existence of MC 20116 and its operation is invisible to the requestors, that is JP 10114 and IOS 10116.

A MEM 10112 read operation transfers data from MS 10112 to a requestor, either JP 10114 or IOS 10116. A read data transfer is asynchronous in that the requestor cannot predict elapsed time between submission of a memory operation request and return of requested data. Operation of a requestor in MEM 10112 is coordinated by a requested data available signal transmitted from MEM 10112 to the requestor.

A MEM 10112 write operation transfers data from either JP 10114 or IOS 10116 to MEM 10112. During such operations, JP 10114 is not required to wait for a signal from MEM 10112 that data provided to MEM 10112 from JP 10114 has been accepted. JP 10114 may transfer data to MEM 10112's JO Port whenever a JO Port available signal from MEM 10112 is present; read data is accepted immediately without further action or waiting required of JP 10114. Word write operations from IOS 10116 are performed in a similar manner. On block write operations, however, IOS 10116 is required to wait for a data taken signal from MEM 10112 before sending the 2nd, 3rd and 4th words of a block.

MEM 10112 has a capability to perform "lock bit" operations. In such operations, a bit granular read of the data is performed and the entire operand is transmitted to the requestor. At the same time, the most significant bit of the operand, that is the Lock Bit, is set to one in the copy of data stored in MEM 10112. In the operand sent to the requestor, the lock bit remains at its previous value, the value before the current read and set operation. Test and set operations are performed by performing read and set operations wherein the data item length is specified to be one bit.

As previously described, MEM 10112 performs certain maintenance operations, including error detection. MEM 10112's Error Log in BC 20114 is a 32 bit register containing an address field and an error code field. On a first error to occur, the error type and in some cases, such as ERCC errors on read data stored in MSB 20110, the address of the data containing the error are stored in BC 20114's Error Log Register. An interrupt signal indicating detection of an error is raised at the same that information

regarding the error is stored in the Error Log. If multiple errors occur before Error Log is read and reset, the information regarding the first error will be retained and will remain valid. The Error Log code field will, however, indicate that more than one error has occurred.

JP 10114 may request a read Error Log operation referred to as a "Read Log and Reset" operation. In this operation, MEM 10112 reads the entire contents of Error Log to JP 10114, resets Error Log Register, and resets the interrupt signal indicating presence of an error. IOS 10116, as discussed further below, is limited to reading 16 bits at a time from MEM 10112. It therefore requires two read operations to read Error Log. First read operation to IOS 10116 reads an upper 16 bits of Error Log data and does not reset Error Log. The second read operation is performed in the same manner as a JP 10114 Read Log and Reset operation, except that only the low order 16 bits of Error Log are read to IOS 10116.

MEM 10112 performs repair block operations to correct parity or ERCC errors in data stored in MC 20116's Cache or in data stored in MA's 20112. In a repair block procedure, parity bits for data stored in MC 20116's Cache, or ERCC check bits of data stored in MA's 20112, are modified to agree with the data bits of data stored therein. In this regard, repaired uncorrectible errors, such as two bit errors of data in MA's 20112, will have good ERCC and parity values. Until a repair block operation is performed, any read request directed to bad data, that is data having parity or ERCC check bits indicating invalid data, will be flagged as invalid. Repair block operations therefore allow such data to be read as valid, for example to be used in a data correction operation. Errors are ignored and not logged in BC 20114's Error Log in repair block operations. A write operation into an area containing bad data may be accomplished if MEM 10112's internal operation does not require a read-modified-write procedure. Only byte aligned writes of integral byte length data residing in MC 20116 and word aligned writes of integral word lengths of data in MSP 20110 do not require read-modified-write operation. By utilizing such write operations, it is therefore possible to overwrite bad data by use of normal write operations before or instead of repair block operations.

MEM 10112 performs a cache flush operation in event of a power failure, that is when MEM 10112 goes into battery back-up operation. In such an event, only MA's 20112 and BC 20114 remain powered. Before JP 10114 and IOS 10116 lose power, JP 10114 and IOS 10116 must transfer to MEM 10112 any data, including operating state, to be saved. This is accomplished by using a series of normal write operations. After conclusion of these write operations, both JP 10114 and IOS 10116 transmit a flush cache request to MEM 10112. Upon receiving two flush cache requests, MEM 10112 flushes MC 20116's Cache so that all dirty data encached in MC 20116's Cache is transferred into MA's 20112 before power is lost. If only JP 10114 or IOS 10116 is operating, DP 10118 will detect this fact and will have transmitted an enabling signal (FLUSHOK) to MEM 10112 during system initialization. FLUSHOK enables MEM 10112 to perform cache flush upon receiving a single flush cache request. After a cache flush operation, no further MEM 10112 operations are possible until Dp 10118 resets a power failure lock-out signal to enable MEM 10112 to resume normal operation.

Having described MEM 10112's overall structure and operation and certain operations which may be performed by MEM 10112, MEM 10112's interfaces to JP 10114 and IOS 10116 will be described next below.

g. MEM 10112 Interfaces to JP 10114 and IOS 10116 (Figs. 209, 210, 211, 204)

As previously described, MJP Port 10140 and MIO Port 10128 logically function as three independent ports. These ports are an IO Port to IOS 10116, a JP Operand Port to JP 10114 and a JP Instruction Port to JP 10114. Referring to Figs. 209, 210, and 211, diagramic representations of IO Port 20910, JP Operand (JPO) Port 21010, and JP Instruction (JPI) port 21110 are shown respectively.

IO Port 20910 handles all IOS 10116 requests to MEM 10112, including transfer of both instructions and operands. JPO Port 21010 is used for read and write operations of operands, for example numeric values, to and from JP 10114. JPI Port 21110 is used to read SINs, that is SOPs and operand NAMEs, from MEM 10112 to JP 10114. Memory service requests to a particular port are serviced in the order that the requests are provided to the Port. Serial order is not maintained between requests to different ports, but ports may be serviced in the order of their priority. In one embodiment of the present invention, IO Port 20910 is accorded highest priority, followed by JPO port 21010, and lastly by JPI Port 21110, with requests currently contained in a port having priority over incoming requests. As described above and will be described in more detail in following descriptions, MEM 10112 operations are pipelined. This pipelining allows interleaving of requests from IO Port 20910, JPO Port 21010, and JPI port 21110, as well as overlapping service of requests at a particular port. By overlapping operations it is meant that one operation servicing a particular port begins before a previous operation servicing that port has been completed.

1. IO Port 20910 Operating Characteristics (Figs. 209, 204)

Referring first to Fig. 209, a diagramic representation of IO port 20910 is shown. Signals are transmitted between IO Port 20910 and IOS 10116 through MIO Bus 10129, IOM Bus 10130, and IOMC Bus 10131. MIO Bus 10129 is a unidirectional bus having inputs from MC 20116 and FIU 20120 and dedicated to transfers of data and instructions from MEM 10112 to IOS 10118. IOM Bus 10130 is likewise a unidirectional bus and is dedicated to the transfer, from IOS 10118 to MEM 10112, of read addresses, write addresses, and data to be written into MEM 10112. IOM Bus 10130 provides inputs to BYF 20118, FIU 20120, and MIC 20122. IOMC Bus 10131 is a set of dedicated signal lines for the exchange of control signals between IOS 10118 and MEM

10112.

Referring first to MIO Bus 10129, MIO Bus 10129 is a 36 bit bus receiving read data inputs from MC 20116's Cache and from FIU 20120. A single read operation from MEM 10112 to IOS 10116 transfers one 32 bit word (or 4 bytes) of data (MIO(0—31)) and four bits of odd parity (MIOP(0—3)), or one parity bit per byte.

Referring next to IOM Bus 10130, a single transfer from IOS 10116 to MEM 10112 includes 36 bits of information which may comprise either a memory request comprising a physical address, a true length, and command bits. These memory requests and data are multiplexed onto IOM 10130 by IOS 10116.

Data transfers from IOS 10116 to MEM 10112 each comprise a single 32 bit data word (IOM(0—31)) and four bits of odd parity (IOMP(0—3)) or one parity bit per byte. Such data transfers are received by either BYF 20118 or FIU 20120.

Each IOS 10116 memory request to MEM 10112, as described above, an address field, a length field, and an operation code field. Address and length fields occupy the 32 IOM Bus 10130 lines used for transfer of data to MEM 10112 in IOS 10116 write operations. Length field includes four bits of information occupying bits (IOM(03)) of IOM Bus 10130 and address field contains 27 bits of information occupying bits (IOM(4—31)) of IOM Bus 10130. Together, address and length field specify a physical starting address and true length of the particular data item to be written into or read from MEM 10112. Operation code field specifies the type of operation to be performed by MEM 10112. Certain basic operation codes comprise 3 bits of information occupying bits (IOMP (32—36)) of IOM Bus 10130; as described above. These same lines are used for transfer of parity bits during data transfers. Certain operations which may be requested of MEM 10112 by IOS 10116 are, together with their corresponding command code fields, are;

000 = read,
001 = read and set,
010 = write,
011 = error,
100 = read error log (first half),
101 = read error log (second half) and reset,
110 = repair block, and
111 = flush cache.

Two further command bits may specify further operations to be performed by MEM 10112. A first command bit, indicates to MEM 10112 during write operations whether it is desirable to encache the data being written into MEM 10112 in MC 20116's Cache. IOS 10116 may set this bit to zero if reuse of the data is unlikely, thereby indicating to MEM 10112 that MEM 10112 should avoid encaching the data. IOS 10116 may set this bit to one if the data is likely to be reused, thereby indicating to MEM 10112 that it is preferable to encache the data. A second command bit is referred to a CYCLE. CYCLE command bit indicates to MEM 10112 whether a particular data transfer is a single cycle operation, that is a bit granular word, or a four cycle operation, that is a block aligned block or a byte aligned partial block.

IOMC 10131 includes a set of dedicated lines for exchange of control signals between IOS 10116 and MEM 10112 to coordinate operation of IOS 10116 and MEM 10112. A first such signal is Load IO Request (LIOR) from IOS 10116 to MEM 10112. When IOS 10116 wishes to load a memory request into MEM 10112, IOS 10116 asserts LIOR to MEM 10112. IOS 10116 must assert LIOR during the same system cycle during which the memory request, that is address, length, and command code fields, are valid.

If LIOR and IO Port Available (IOPA) signals, described below, are asserted during the same clock cycle, MEM 10112's port is loaded from IOS 10116 and IOPA is dropped, indicating the request has been accepted. If a load of a request is attempted and IOPA is not asserted, MEM 10112 remains unaware of the request, LIOR remains active, and the request must then be repeated when IOPA is asserted.

IOPA is a signal from MEM 10112 to IOS 10116 which is asserted by MEM 10112 when MEM 10112 is available to accept a new request from IOS 10116. IOPA may be asserted while a previous request from IOS 10116 is completing operation if the address, length, and operation code fields of the previous request are no longer required by MEM 10112, for example in servicing bypass operations.

IO Data Taken (TIOMD) is a signal from MEM 10112 to IOS 10116 indicating that MEM 10112 has accepted data from IOS 10116. IOS 10116 places a first data word on IOM Bus 10130 on the next system clock cycle after a write request is loaded; that is, LIOR has been asserted, a memory request presented, and IOPA dropped. MEM 10112 then takes that data word on the clock edge beginning the next system clock cycle. At this point, MEM 10112 asserts TIOMD to indicate the data has been accepted. On a single word operations TIOMD is not used by IOS 10116 as a first data word is always accepted by MEM 10112 if IO Port 20910 was available. On block operations, a first data word is always taken but a delay may occur between acceptance of first and second words. IOS 10116 is required to hold the second word valid on IOM Bus 10130 until MEM 10112 responds with TIOMD to indicate that the block operation may proceed.

Data Available for IO (DAVIO) is a signal asserted by MEM 10112 to IOS 10116 indicating that data requested by IOS 10116 is available. DAVIO is asserted by MEM 10112 during the system clock cycle in which MEM 10112 places the requested data on MIO Bus 10129. In any single word type transfer, DAVIO is active for a single system clock transfer. In block type transfers, DAVIO is normally active for four consecutive system clock cycles. Upon event of a single cycle "bubble" resulting from detection and

correction of an ERCC error by BC 20114, DAVIO will remain high for four non-consecutive system clock cycles and with a single cycle bubble, a non-assertion, in DAVIO corresponding to the detection and correction of the error.

IO Memory Interrupt (IMINT) is a signal asserted by MEM 10112 to IOS 10116 when BC 20114 places a record of a detected error in BC 20114's Error Log, as described above.

Previous MIO Transfer Invalid (PMIOI) signal is similarly a signal asserted by MEM 10112 to IOS 10116 regarding errors in data read from MEM 10112 to IOS 10116. If an uncorrectible error appears in such data, that is an error in two or more data bits, the incorrect data is read to IOS 10116 and PMIOI signal asserted by MEM 10112. Correctible, or single bit, errors in data do not result in assertion of PMIOI. MEM 10112 will assert PMIOI to IOS 10116 of the next system clock cycle following MEM 10112's assertion of DAVIO.

Having described MEM 10112's interface to IOS 10116, and certain operations which IOS 10116 may request of MEM 10112, certain MEM 10112 operations within the capability of the interface will be described next. First, operand transfers, for example of numeric data, between MEM 10112 and IOS 10116 may be bit granular with any length from one to sixteen bits. Operand transfers may cross boundaries within a page but may not cross physical page boundaries. As previously described, MIO Bus 10129 and IOM Bus 10130 are capable of transferring 32 bits of data at a time. The least significant 16 bits of these buses, that is bits 16 to 31, will contain right justified data during operand transfers. The contents of the most significant 16 bits of these buses is generally not defined as MEM 10112 generally does not perform fill operations on read operations to IO Port 20910, nor does IOS 10116 fill unused bits during write operations. During a read or write operation, only those data bits indicated by length field in the corresponding memory request are of significance. In all cases, however, parity must be valid on all 32 bits of MIO Bus 10129 and IOM Bus 10130.

Referring to Fig. 204, IOS 10116 includes Data Channels 20410 and 20412 each of which will be described further in a following detailed description of IOS 10116. Data Channels 20410 and 20412 each possess particular characteristics defining certain IO Port 20910 operations. Data Channel 20410 operates to read and write block aligned full and partial blocks. Full blocks have block aligned addresses and lengths of 16 bytes. Partial blocks have byte aligned addresses and lengths of 1 to 15 bytes; a partial block transfer must be within a block, that is not cross block boundaries. A full 4 word block will be transferred between IOS 10116 and MEM 10112 in either case, but only those blocks indicated by length of field in a corresponding MEM 10112 request are of actual significance in a write operation. Non-addressed bytes in such operations may contain any information so long as parity is valid for the entire data transfer. Data Channel 20412 preferably reads or writes 16 bits at a time on double byte boundaries. Such reads and writes are right justified on MIO Bus 10129 and IOM Bus 10130. The most significant 16 bits of these buses may contain any information during such operations so long as parity is valid for the entire 32 bits. Data Channel 20412 operations are similar to IOS 10116 operand read and write operations with double byte aligned addresses and lengths of 16 bits. Finally, instructions, for example controlling IOS 10116 operation, are read from MEM 10112 to IOS 10116 a block at a time. Such operations are identical to a full block data read.

Having described the operating characteristics of IO Port 20910, the operating characteristics of JPO Port 21010 will be described next.

2. JPO Port 21010 Operating Characteristics (Fig. 210)

Referring to Fig. 210, a diagramic representation of JPO Port 21010 is shown. As previously described, JPO Port 21010 is utilized for transfer of operands, for example numeric data, between MEM 10112 and JP 10114. JPO Port 21010 includes a request input (address, length, and operation information) to MIC 20122 from 36 bit PD Bus 10146, a write data input to FIU 20120 from 32 bit JPD Bus 10142, a 32 bit read data output from MC 20116 and FIU 20120 to 32 bit MOD Bus 10144, and bi-directional control inputs and outputs between MIC 20122 and JPMC Bus 10147.

Referring first to JPO Port 21010's read data output to MOD Bus 10144, MOD Bus 10144 is used by JPO Port 21010 to transfer data, for example operands, to JP 10114. MOD Bus 10144 is also utilized internal to MEM 10112 as a bidirectional bus to transfer data between MC 20116 and FIU 20120. In this manner, data may be transferred from MC 20116 to FIU 20120 where certain data format operations are performed on the data before the data is transferred to JP 10114 through MOD Bus 10144. Data may also be used to transfer data from FIU 20120 to MC 20116 after a data format operation is performed in a write operation. Data may also be transferred directly from MC 20116 to JP 10114 through MOD Bus 10144. Internal to MEM 10112, MOD Bus 10144 is a 36 bit bus for concurrent transfer of 32 bits of data, MOD Bus 10144 bits (MOD(0—31)), and 4 bits of odd parity, 1 bit per byte, MOD Bus 10144 bits (MODP(0—3)). External to MEM 10112, MOD Bus 10144 is a 32 bit bus, comprising bits (MOD(0—31)); parity bits are not read to JP 10114.

Data is written into MEM 10112 through JPD Bus 10142 to FIU 20120. As just described, data format operations may then be performed on this data before it is transferred from FIU 20120 to MC 20116 through MOD Bus 10144. In such operations, JPD Bus 10142 operates as a 32 bit bus carrying 32 bits of data, bits (JPD (0—31)), with no parity bits. JO Port 21010 generates parity for JPD Bus 10142 data to be written into MEM 10112 as this data is transferred into MEM 10112.

Memory requests are also transmitted to MEM 10112 from JP 10114 through JPD Bus 10142, which operates in this regard as a 40 bit bus. Each such request includes an address field, a length field, an FIU

field specifying data formating operations to be performed, operation code field, and a destination code field specifying destination of data read from MEM 10112. Address field includes a 13 bit physical page number field, (JPPN(0—12)), and a 14 bit physical page offset field, (JPPO(0—13)). Length field includes 6 bits of length information, (JLNG(0—5)), and expresses true length of the data item to be written to or read from MEM 10112.

As JPD Bus 10142 and MOD Bus 10144 are each capable of transferring 32 bits of data in a single MEM 10112 read or write cycle, 6 bits of length information are required to express true length. As will be described in a following description, JP 10114 may provide physical page offset and length information directly to MEM 10112, performs logical page number to physical page number translations, and may perform a Protection Mechanism 10230 check on the resulting physical page number. As such, MEM 10112 expects to receive (JPPN(0—12)) later than (JPPO(0—13)) and (JLNG(0—5)). (JPPO(0—13)) and (JLNG(0—5)) should, however, be valid during the system clock cycle in which a JP 10114 memory request is loaded into MEM 10112.

Operation code field provided to MEM 10112 from JP 10114 is a 3 bit code, (JMCMD(0—2)) specifying an operation to be formed by MEM 10112. Certain operations which JP 10114 may request of MEM 10112, and their corresponding operation codes, are:

000 = read;
001 = read and set;
010 = write;
011 = error;
100 = error;
101 = read error log and reset;
110 = repair block; and,
111 = flush cache.

Two bit FIU field, (JFIU(0—1)) specifies data manipulation operations to be performed in executing read and write operations. Among the data manipulation operations which may be requested by JP 10114, and their FIU fields, are:

00 = right justified, zero fill;
01 = right justified, sign extend;
10 = left justify, zero fill; and,
11 = left justify, blank fill.

For write operations, JPO Port 21010 may respond only to the most significant bit of FIU field, that is the FIU field bit specifying alignment.

Finally, destination field is a two bit field specifying a JP 10114 destination for data read from MEM 10112. This field is ignored for write operations to MEM 10112. A first bit of destination field, JPMDST, identifies the destination to be FU 10120, and the second field, EBMDST, specifies EU 10120 as the destination.

JPMC Bus 10147 includes dedicated lines for exchange of control signals between JPO Port 21010 and JP 10114. Among these control signals is Load JO Request (LJOR), which is asserted by JP 10114 when JP 10114 wishes to load a request into MEM 10112. LJOR is asserted concurrently with presentation of the memory request to MEM 10112 through PD Bus 10146. JO Port Available (JOPA) is asserted by MEM 10112 when JPO Port 21010 is available to accept a new memory request from JP 10114. If LJOR and JOPA are asserted concurrently, MEM 10112 accepts the memory request from JP 10114 and MEM 10112 drops JOPA to indicate that memory request has been accepted. As previously discussed, MEM 10112 may assert JOPA while a previous request is being executed and the PD Bus 10146 information, that is the memory request previously provided concerning the previous request, is no longer required.

If JP 10114 submits a memory request and JOPA is not asserted by MEM 10112, MEM 10112 does not accept the request and JP 10114 must resubmit that request when JOPA is asserted. Because, as described above, JPPN field of a memory request from JP 10114 may arrive late compared to the other fields of the request, MEM 10112 will delay loading of JPPN field for a particular request until the next system clock cycle after the request was initially submitted. MEM 10112 may also obtain this JPPN field at the same time it is being loaded into the port register by by-passing the port register.

JP 10114 may abort a memory request upon asserting Abort JP Request (ABJR). ABJR will be accepted by MEM 10112 during system clock cycle after accepting memory request from JP 10114 and ABJR will result in cancellation of the requested operation. A single ABJR line is provided for both JPO Port 21010 and JPI Port 21110 because, as described in a following description, MEM 10112 may accept only a single request from JP 10114, to either JPO Port 21010 or to JPI port 21110, during a single system clock cycle.

Upon completion of an operand read operation requested through JPO Port 21010 MEM 10112 may assert either of two data available signals to JP 10114. These signals are data available for FA(DAVFA) and data available for EB(DAVEB). As previously described, a part of each read request from JP 10114 includes a destination field specifying the intended destination of the requested data. As will be described further

below, MEM 10112 tracks such destination information for read requests and returns destination information with a corresponding information in the form of DAVFA and DAVEB. DAVFA indicates a destination in FU 10120 while DAVEB indicates a destination in EU 10122. MEM 10112 may also assert signal zero filled (ZFILL) specifying whether read data for JPO Port 21010 is zero filled. ZFILL is valid only when DAVEB is asserted.

For JPO Port 21010 write request, the associated write data word should be valid on same system clock cycle as the request, or one system clock cycle later. JP 10114 asserts Load JP Write Data (LJWD) during the system clock cycle when JP 10114 places valid write data on JPD Bus 10142.

As previously discussed, when MEM 10112 detects an error in servicing a JP 10114 request MEM 10112 places a record of this error in MC 20116's Error Log. When an entry is placed in Error Log for either JPO Port 21010 or IO Port 20910, MEM 10112 asserts an interrupt flag signal indicating a valid Error Log entry is present. DP 10118 detects this flag signal and may direct the flag signal to either JP 10114 or IOS 10116, or both. IOS 10116 or JP 10114, as selected by DP 10118, may then read and reset Error Log and reset the flag. The interrupt flag signal is not necessarily directed to the requestor, JP 10114 or IOS 10116, whose request resulted in the error.

If an uncorrectible MEM 10112 error, that is an error in two or more bits of a single data word, is detected in a read operation the incorrect data is read to JP 10114 and an invalid data signal asserted. A signal, Previous MOD Transfer Invalid (PMODI), is asserted by MEM 10112 on the next system clock cycle following either DAVFA or DAVEB. PMODI is not asserted for single bit errors, instead the data is corrected and the corrected data read to JP 10114.

Having described JPO Port 21010's structure, and characteristics, JPI Port 21110 will be described next below.

3. JPI Port 21110 Operating Characteristics (Fig. 211)

Referring to Fig. 211, a diagramic representation of JPI Port 21110 is shown. JPI port 21110 includes an address input from PD Bus 10146 to FIU 20120, a data output to MOD Bus 10144 from MC 20116, and bi-directional control inputs and outputs from MIC 20122 to JPMC Bus 10147. As previously described, a primary function of JPI Port 21110 is the transfer of SOPs and operand NAMEs from MEM 10112 to JP 10114 upon request from JP 10114. JPI Port thereby performs only read operations wherein each read operation is a transfer of a single 32 bit word having a word aligned address.

Referring to JPI Port 21110 input from PD Bus 10146, read requests to MEM 10112 by JP 10114 for SOPs and operand NAMEs each comprise a 21 bit word address. As described above, each JPI Port 21110 read operation is of a single 32 bit word. As such, the five least significant bits of address are ignored by MEM 10112. For the same reason, a JPI Port 21110 request to MEM 10112 does not include a length field, an operation code field, an FIU field, or a destination code field. Length, operation code, and FIU code fields are not required since JPI Port 21110 performs only a single type of operation and destination code field is not required because destination is inherent in a JPI Port 21110 request.

The 32 bit words read from MEM 10112 in response to JPI Port 21110 requests are transferred to JP 10114 through MC 20116's 32 bit output to MOD Bus 10144. As in the case of JPO 21010 read outputs to JP 10114, JPI Port 21110 does not provide parity information to JP 10114.

Control signals exchange between JP 10114 and JPI Port 21110 through JPMC Bus 10147 include Load JI Request (LJIR) and JI Port Available (JIPA), which operate in the same manner as discussed with reference to JPO Port 21010. As previously described, JPO Port 21010 and JPI Port 21110 share a single Abort JP Request (ABJR) command. Similarly, JPO Port 21010 and JPI Port 21110 share previous MOD Transfer Invalid (PMODI) from MEM 10112. As described above, a JPI port 21110 request does not include a destination field as destination is implied. MEM 10112 does, however, provide a Data Available Signal (DAVFI) to JP 10114 when a word read from MEM 10112 in response to a JPI Port 21110 request is present on MOD Bus 10144 and valid.

Having described the overall structure and operation of MEM 10112, and the structure and operation of MEM 10112's interface to JP 10114 and IOS 10116, the structure and operation of FIU 20120 MEM 10112 will next be described in further detail.

h. FIU 20120 (Figs. 201, 230, 231)

As previously described, FIU 20120 performs certain data manipulation operations, including those operations necessary to make MEM 10112 bit addressable. Data manipulation operations may be performed on data being written into MEM 10112, for example, JP 10114 through JPD Bus 10142 or from IOS 10116 through IOM Bus 10130. Data manipulations operations may also be performed on data being read from MEM 10112 to JPD 10114 or IOS 10116. In case of data read to JP 10114, MOD Bus 10144 is used both as a MEM 10112 internal bus, in transferring data from MC 20116 to FIU 20120 for manipulation, and to transfer manipulated data from MEM 10112 to JP 10114. In case of data read to IOS 10116, MOD Bus 10144 is again used as MEM 10112 internal bus to read data from MC 20116 to FIU 20120 for subsequent manipulation. The manipulated data is then read from FIU 20120 to IOS 10116 through MIO Bus 10129.

Certain data manipulation operations which may be performed by FIU 20120 have been previously described. In general, a data manipulation operation consists of four distinct operations, and FIU 20120 may manipulate data in any possible manner which may be achieved through performing any combination

of these operations. These four possible operations are selection of data to be manipulated, rotation or shifting of that data, masking of that data, and transfer of that manipulated data to a selected destination. Each FIU 20120 data input will comprise a thirty-two bit data word and, as described above, may be selected from input provided from JPD Bus 10142, MOD Bus 10144, and IOM Bus 10130. In certain cases, an FIU 20120 data input may comprise two thirty-two bit words, for example, when a cross word operation is performed generating an output comprised of bits from each of two different thirty-two bit words. Rotation or shifting of a selected thirty-two bit data word enables bits within a selected word to be repositioned with respect to word boundaries. When used in conjunction with the masking operation, described momentarily, rotation and shifting may be reiterably performed to transfer any selected bits in a word to any selected locations in that word. As will be described further below, a masking operation allows any selected bits of a word to be affectively erased, thus leaving only certain other selected bits, or certain selected bits to be forced to predetermined values. A masking operation may be performed, for example, to zero fill or sign extend portions of a thirty-two bit word. In conjunction with a rotation or shifting operation, a masking operation may, for example, select a single bit of a thirty-two bit input word, position that bit in any selected bit location, and force all other bits of that word to zero. Each output of FIU 20120 is a thirty-two bit data word and, as described above, may be transferred on to MOD Bus 10144 or onto MIO Bus 10129. As will be described below, selection of a particular sequence of the above four operations to be performed on a particular data word is determined by control inputs provided from MIC 20122. These control inputs from MIC 20122 are decoded and executed by microinstruction control logic included within FIU 20120.

Referring to Fig. 230, a partial block diagram of FIU 20120 is shown. As indicated therein, FIU 20120 includes Data Manipulation Circuitry (DMC) 23010 and FIU Control Circuitry (FIUC) 23012. Data Manipulation Circuitry 23010 in turn includes FIUIO circuitry (FIUIO) 23014, Data Shifter (DS) 23016, Mask Logic (MSK) 23018, and Assembly Register (AR) logic 23020. Data manipulation circuitry 23010 will be described first followed by FIUC 23012. In describing data manipulation circuitry 23010, FIUIO 23014 will be described first, followed by DS 23016, MSK 23018, and AR 23020, in that order.

Referring to FIUIO 23014, FIUIO 23014 comprises FIU 20120's data input and output circuitry. Job Processor Write Data Register (JWDR) 23022, IO System Write Data Register (IWDR) 23024, and Write Input Data Register (RIDR) 23026 are connected from, respectively, JPD Bus 10142, IOM Bus 10130, and MOD Bus 10144 for receiving data word inputs from, respectively, JP 10114, IOS 10116, and MC 20116. JWDR 23022, IWDR 23024 and RIDR 23026 are each thirty-six bit registers comprised, for example, of SN74S374 registers. Data words transferred into IWDR 23024 and RIDR 23026 are each, as previously described, comprised of a thirty-two data word plus four bits of parity. Data inputs from JP 10114 are, however, as previously described, thirty-two bit data words without parity. Job Processor Parity Generator (JPPG) 23028 associated with JWDR 23022 is connected from JPD Bus 10142 and generates four bits of parity for each data input to JWDR 23022. JWDR 23022's thirty-six bit input thereby comprises thirty-two bits of data, directly from JPD Bus 10142, plus a corresponding four bits of parity from JPPG 23028.

Data words, thirty-two bits of data plus four bits of parity, are transferred into JWDR 23022, IWDR 23024, or RIDR 23026 when, respectively, input enable signals Load JWD (LJWD), Load IWD (LIWD) or Load RID (LRID) are asserted. LJWD is provided from FU 10120 while LIWD and LRID are provided from MIC 20122.

Data words resident in JWDR 23022, IWDR 23024, or RIDR 23026 may be selected and transferred onto FIU 20120's Internal Data (IB) Bus 23030 by output enable signals JWD Enable Output (JWDEO), IWD Enable Output (IWDEO), an RID Enable Output (RIDEO). JWDEO, IWDEO, and RDIEO are provided from FIUC 23012 described below.

As will be described further below, manipulated data words from DS 23016 or AR 23020 will be transferred onto, respectively, Data Shifter Output (DSO) Bus 23032 or Assembly Register Output (ASYRO) Bus 23034 for subsequent transfer onto MOD Bus 10144 or MIO Bus 10129. Each manipulated data word appearing on DSO Bus 23032 or ASYRO Bus 23034 will be comprised of 32 bits of data plus 4 bits of parity. Manipulated data words present on DSO Bus 23032 may be transferred onto MOD Bus 10144 or MIO Bus 10129 through, respectively, DSO Bus To MOD Bus Driver Gate (DSMOD) 23036 or BSO Bus To MIO Bus Driver Gate (DSMIO) 23038. Manipulated data words present on ASYRO Bus 23034 may be transferred onto MOD Bus 10144 or MIO Bus 10129 through, respectively, ASYRO Bus To MOD Bus Driver Gate (ASYMOD) 23040 or ASYRO Bus To MIO Bus Driver Gate (ASYMIO) 23042. DSMOD 23036, DSMIO 23038, ASYMOD 23040, and ASYMIO 23042 are each comprised of, for example, SN74S244 drivers. A manipulated data word on DSO Bus 23032 be transferred through DSMOD 23036 to MOD Bus 10144 when driver gate enable signal Driver Shift To MOD (DRVSHFMOD) to DSMOD 23036 is asserted. Similarly, a manipulated data word on DSO Bus 23032 will be transferred through DSMIO 23038 to MIO Bus 10129 when driver gate enable signal Drive Shift Through MIO Bus (DRVSHFMIO) to DSMIO 23038 is asserted. Manipulated data words present on ASYRO Bus 23034 may be transferred onto MOD Bus 10144 or MIO Bus 10129 when, respectively, driver gate enable signal Drive Assembly To Mod Bus (DRVASYMOD) to ASYMOD 23040 or Drive Assembly To MIO Bus (DRVASYMIO) to ASYMIO 23042 are asserted. DRVSHFMOD, DRVSHFMIO, DRVASYMOD, and DRVASYMIO are provided, as described below, from FIUC 23012.

Registers IARM 23044 and BARMR 23046, which will be described further in a following description of DP 10118, are used by DP 10118 to, respectively, write data words onto IB 23030 and to Read data words

from MOD Bus 10144, for example manipulated data words from FIU 20120. Data word written into IARMR 23044 from DP 10118, that is 32 bits of data and 4 bits of parity, will be transferred onto IB Bus 23030 when register enable output signal IARM enable output (IARMEO) from FIUC 23012 is asserted. Similarly, a data word present on MOD Bus 10144, comprising 32 bits of data plus 4 bits of parity, will be written into BARMR 23046 when load enable signal Load BARMR (LDBARMR) to BARMR 23046 is asserted by MIC 20122. A data word written into BARMR 23046 from MOD Bus 10144 may then subsequently be read to DP 10118. IARMR 23044 and BARMR 23046 are similar to JWDR 23022, IWDR 23024, and IRDR 23026 and may be comprised, for example, of SN74S299 registers.

Referring finally to IO Parity Check Circuit (IOPC) 23048, IOPC 23048 is connected from IB Bus 23030 to receive each data word, that is 32 bits of data plus 4 bits of parity, appearing on IB Bus 23030. IOPC 23048 confirms parity and data validity of each data word appearing on IB Bus 23030 and, in particular, determines validity of parity and data of data words written into FIU 20120 from IOS 10116. IOPC 23048 generates output Parity Error (PER), previously discussed, indicating a parity error in data words from IOS 10116.

Referring to DS 23016, DS 23016 includes Byte Nibble Logic (BYNL) 23050, Parity Rotation Logic (PRL) 23052, and Bit Scale Logic (BSL) 23054. BYNL 23050, PRL 23052, and BSL 23054 may respectively be comprised of, for example, 25S10 shifters. BYNL 23050 is connected from IB Bus 23030 for receiving and shifting the 32 data bits of a data word selected and transferred onto IB Bus 23030. PRL 23052 is a 4 bit register similarly connected from IB Bus 23030 to receive and shift the 4 parity bits of a data word selected and transferred onto IB Bus 23030. Outputs of BYNL 23050 and PRL 23052 are both connected onto DSO Bus 23032, thus providing a 36 bit FIU 20120 data word output directly from BYNL 23050 and PRL 23052. BYNL 23050's 32 bit data output is also connected to BSL 23054's input. BSL 23054's 32 bit output is in turn provided to MSK 23018.

As previously described, DS 23016 performs data manipulation operations involving shifting of bits within a data word. In general, data shift operations performed by DS 23016 are rotations wherein data bits are right shifted, with least significant bits of data word being shifted into most significant bit position and most significant bits being translated towards least significant bit positions. DS 23016 rotation operations are performed in two stages. First stage is performed by BYNL 23050 and PRL 23052 and comprises right rotations on a nibble basis (a nibble is defined as 4 bits of data). That is, BYNL 23050 right shifts a data word by an integral number of 4 bit increments. A right rotation on a nibble by nibble basis may, for example, be performed when RM 20722 asserts FLIPHALF previously described. FLIPHALF is asserted for IOS 10116 half word read operations wherein the request data resides in the most significant 16 bits of a data word from MC 20116. BYNL 23050 will perform a right rotation of 4 nibbles to transfer the desired 16 bits of data into the least significant 16 bits of BYNL 23050's output. Resulting BYNR 23050 output, together PRL 23052's parity bit output would then be transferred through DSO 23050 to MIO Bus 10129. In addition to performing data shifting operations, DS 23016 may transfer a data word, that is the 32 bits of data, directly to MSK 23018 when data manipulation to be performed does not require data shifting, that is shifts of 0 bits may be performed.

Because data bits are shifted by BYNL 23050 on a nibble basis, the relationship between the 32 data bits of a word and the corresponding 4 parity bits may be maintained if parity bits are similarly right rotated by an amount corresponding to right rotation of data bits. This relationship is true if the data word is shifted in multiples of 2 nibbles, that is 8 bits or 1 byte. PRL 23052 right rotates the 4 parity bits of a data word by an amount corresponding to right rotation of the corresponding 32 data bits in BYNL 23050. Right rotated outputs of BYNL 23050 and PRL 23052 therefore comprise a valid data word having 32 bits of data and 4 bits of parity wherein the parity bits are correctly related to the data bits. A right rotated data word output from BYNL 23050 and PRL 23052 may be transferred onto DSO Bus 23032 for subsequent transfer to MOD Bus 10144 or MIO Bus 10129 as described above. DSO 23032 is used as FIU 20120's output data path for byte write operations and "rotate read" operations wherein the required manipulation of a particular data word requires only an integral numer of right rotations by bytes. Amount of right rotation of 32 bits of data in BYNL 23050 and 4 bits of parity in PRL 23052 is controlled by input signal shift (SHFT) (0—2) to BYNL 23050 and PRL 23052. As will be described below, SHFT (0—2) is generated, together with SHFT (3—4) controlling BSL 23054, by FIUC 23012. BYNL 23050 and PRL 23052, like BSL 23054 described below, are parallel shift logic chips and entire rotation operation of BYNL 23050 and PRL 23052 or BSL 23054 may be performed in a single clock cycle.

Second stage of rotation is performed by BSL 23054 which, as described above, receives the 32 data bits of a data word from BYNL 23050. BSL 23054 performs right rotation on a bit by bit basis with the shift amount being selectable between 0—3 bits. Therefore, BSL 23054 may rotate bits through nibble boundaries. BYNL 23050 and BSL 23054 therefore comprise a data shifting circuit capable of performing bit-by-bit right rotation by an amount from 1 bit to a full 32 bit right rotation.

Referring now to MSK 23018, MSK 23018 is comprised of 5 32 bit Mask Word Generators (MWG's) 23056 to 23064. MSK 23018 generates a 32 bit output to AR 23020 by selectively combining 32 bit mask word outputs of MWG's 23056 to 23064. Each mask word generated by one of MWG's 23056 to 23064 is effectively comprised a bit by bit combination of a set of enabling bits and a predetermined 32 bit mask word, generated by FIUC 23012 and MIC 20122. MWG's 23058 to 23064 are each comprised of for example, open collector NAND gates for performing these functions, while NWG 23056 is comprised of a PROM.

As just described, outputs of MWG's 23056 to 23064 are all open collector circuits so that any selected combination of mask word outputs from MWG's 23056 to 23064 may be ORed together to comprise the 32 bit output of MSK 23018.

MWG 23056 to MWG 23064 generate, respectively, mask word outputs Locked Bit Word (LBW) (0—31), Sign Extended Word (SEW) (0—31), Data Mask Word (DMW) (0—31), Blank Fill Word (BWF) (0—31), and Assembly Register Output (ARO) (0—31). Referring first to MWG 23064 and ARO (0—31), the contents of Assembly Register (ASYMR) 23066 in AR 23020 are passed through MWG 23064 upon assertion of enabling signal Assembly Output Register (ASYMOR). ARO (0—31) is thereby a copy of the contents of ASYMR 23066 and MWG 23064 allows the contents of ASYMR 23066 to be ORed with the selected combination of LBW (0—31), SEW (0—31), DMW (0—31), or BFW (0—31).

DMW (0—31) from MWG 23060 is generated by ANDing enable Input Data Mask (DMSK) (0—31) with the 32 bit output of DS 23016. DMSK (0—31) is a 32 bit enabling word generated, as described below, by FIUC 23012. FIUC 23012 may generate 4 different DMSK (0—31) patterns. Referring to Fig. 231, the 4 DMSKs (0—31) which may be generated by FIUC 20132 are shown. DMSKA (0—31) is shown in Line A of Fig. 231. In DMSKA (0—31) all bits to the left of but not including a bit designated by Left Bit Address (LBA) and all bits to the right of and not including a bit designated by Right Bit Address (RBA) are 0. All bits between, and including, those bits designated by LBA and RBA are 1's. DMSKB (0—31) is shown in Line B of Fig. 231 and is DMSKA (0—31) inverted. DMSKC (0—31) and DMSKD (0—31) are shown, respectively, in Lines C and D of Fig. 231 and are comprised of, respectively, all 0's or all 1's. As stated above DMSK (0—31) is ANDed with the 32 bit output of DS 23016. As such, DMSKC (0—31) may be used, for example, to inhibit DS 23016's output while DMSKD (0—31) may be used, for example, to pass DS 23016's output to AR 23020. DMSKA (0—31) and DMSKB (0—31) may be used, for example, to gate selected portions of DS 23016's output to AR 23020 where, for example, the selected portions of DS 23016's output may be ORed with other mask word outputs MSK 23018.

Referring next to MWG 23062, MWG 23062 generates BFW (0—31). BFW (0—31) is used in a particular operation wherein 32 bit data words containing 1 to 4 ASCII blanks are required to be generated wherein 1 bit/byte contains a logic one and remaining bits contain logic zeros. In this case, the ASCII blank bytes may contain logic 1's in bit positions 2, 10, 18, and 26.

Referring again to Fig. 231, Line E therein shows 32 bit right mask (RMSK) (0—31) which may be generated by FIUC 23012. In the most general case, RMSK contains zeros in all bit positions to the left of and including a bit position designated by RBA. When used in a blank fill operation, bit positions 2, 10, 18, and 26 may be selected to contain logic 1's depending upon those byte positions containing logic 1's, that is in those bytes containing ASCII blanks; these bytes to the right of RBA are determined by RMSK (0—31). RMSK (0—31) is enabled through MWG 23062 as BWF (0—31) when MWG 23062 is enabled by blank fill (BLNKFILL) provided from FIU 23012.

As described above, MWG's 23058 to 23064 and in particular MWG's 23060 and MWG 23062 are NAND gate operations. Therefore, the outputs of MWGs 23056 through 23064 are active low signals. The inverted output of ASYMR 23066 is used as an output to ASYRO 23034 to invert these outputs to active high.

MWG 23058, generating SEW (0—31), is used in generating sign extended or filled words. In sign extended words, all bit spaces to the left of the most significant bit of a 32 bit data word are filled with the sign bit of the data contained therein, the left most bits of the 32 bit word are filled with 1's or 0's depending on whether that word's sign bit indicates that the data contained therein is a positive or negative number.

Sign Select Multiplexor (SIGNSEL) 23066 is connected to receive the 32 data bits of a word present on IB Bus 23030. Sign Select (SGNSEL) (0—4) to SIGNSEL 23066 is derived from SBA (0—4), that is from SBA Bus 21226 from PRMUX 20720. As previously described, SBA (0—4) is Starting Bit Address identifying the first or most significant bit of a data word. When a data word contains a signed number, most significant bit contains sign bit of that number. SGNSEL (0—4) input to SIGNSEL 23066 is used as a selection input and, when SIGNSEL is enabled by Sign Extend (SIGNEXT) from FIU 23012, selects the sign bit on IB Bus 23030 and provides that sign bit as an input to MWG 23058.

Sign bit input to MWG 23058 is ANDed with each bit of left hand mask (LMSK) (0—31) from FIUC 23012. Referring again to Fig. 231, LMSK (0—31) is shown on Line F thereof. LMSK (0—31) contains all 0's to the right of and including the bit space identified by LBA and 1's in all bit spaces to the left of that bit space identified by LBA. SEW (0—31) will therefore contain sign bit in all bit spaces to the left of the most significant bit of the data word present on output of MWG 23058. The data word on IB Bus 23030 may then be passed through DS 23016 and subjected to a DMSK operation wherein all bits to the left of the most significant bit are forced to 0. SEW (0—31) and DMW (0—31) outputs of MWG's 23058 and 23060 may then be ORed to provide the desired find extended word output.

LBW (0—31), provided by MWG 23056, is used in locked bit operations wherein the most significant data bit of a data word is in MEM 10112 forced to logic 1. SIGNSEL (0—4) is an address input to MWG 23056 and, as previously described, indicates most significant data bit of a data word present on an IB Bus 23030. MWG 23056 is enabled by input Lock (LOCK) from FIUC 23012 and the resulting LBW (0—31) will contain a single logic 1 in the bit space of the most significant data bit of the data word present on IB Bus 23030. The data word present on IB Bus 23030 may then be passed through DS 23016 and MWG 23060 to be ORed with LBW (0—31) so that that data words most significant data bit is forced to logic 1.

Referring to AR 23020, AR 23020 includes ASYMR 23066, which may be comprised for example of a

SN74S175 registers, and Assembly Register Parity Generator (ASYPG) 23070. As previously described, ASYMR 23066 is connected from MSK 23018 32 bit output. A 32 bit word present on MSK 23018's output will be transferred into ASYMR 23066 when ASYMR 23066 is enabled by Assembly Register Load (ASYMLD) from MIC 20122. The 32 bit word generated through DS 23016 and MSK 23018 will then be present on ASYRO Bus 23034 and may, as described above, then be transferred onto MOD Bus 10144 or MIO Bus 10129. ASYPG 23070 is connected from ASYMR 23066 32 bit output and will generate 4 parity bits for the 32 bit word presently on the 32 data lines of ASYRO Bus 23034. ASYPG 23070's 4 bit parity output is bused on the 4 parity bit lines of ASYRO Bus 23034 and accompany the 32 bit data word present thereon.

Having described structure and operation of Data Manipulation Circuitry 23010, FIUC 23012 will be described next below.

Referring again to Fig. 230, FIUC 23012 provides pipelined microinstruction control of FIU 20120. That is, control signals are received from MIC 20122 during a first clock cycle and certain of the control signals are decoded by microinstruction logic to generate further FIUC 23012 control signals. During the second clock cycle, control signals received and generated during first clock cycle are provided to DMC 23010, some of which are further decoded to provide yet other control signals to control operation of FIUC 23012. FIUC 23012 includes Initial Decode Logic (IDL) 23074, Pipeline Registers (PPLR) 23072, Final Decoding Logic (FDL) 23076, and Enable Signal Pipeline Register (ESPR) 23098 with Enable Signal Decode Logic (ESDL) 23099.

IDL 23074 and Control Pipeline Register (CPR) 23084 of PPLR 23072 are connected from control outputs of MIC 20122 to receive control signals therefrom during a first clock cycle as described above. IDL 23074 provides outputs to control pipeline registers Right Bit Address Register (RBAR) 23086, Left Bit Address Register (LBAR) 23088 and Shift Register (SHFR) 23090 of PPLR 23072. CPR 23084 and SHFR 23090 provide control outputs directly to DMC 23010. As described above these outputs control DMC 23010 during second clock cycle.

CPR 23084, RBAR 23086, and LBAR 23088 provide outputs to FDL 23076 during second clock cycle and FDL 23076 in turn provides certain outputs directly to DMC 23010.

ESPR 23098 and ESDL 23099 receive enable and control signals from MIC 20122 and in turn provide enable and control signals to DMC 23010 and certain other portions of MEM 10112 circuitry.

IDL 23074 and FDL 23076 may be comprised, for example, of PROMs. CPR 23084, RBAR 23086, LBAR 23088, SHFR 23090, and ESPR 23098 may be comprised, for example, of SN74S194 registers. ESDL 23099 may be comprised of, for example, compatible decoders, such as logic gates.

Referring first to IDL 23074, IDL 23074 performs an initial decoding of circuitry control signals from MIC 20122 and provides further control signals used by FIUC 23012 in controlling FIU 20120. IDL 23074 is comprised of read-only memory arrays Right Bit Address Decoding Logic (RBADL) 23078, Left Bit Address Decoding Logic (LBADL) 23080, and Shift Amount Decoding Logic (SHFAMTDL) 23082. RBADL 23078 receives, as address inputs, Final Bit Address (FBA) (0—4), Bit Length Number (BLN) (0—4), and Starting Bit Address (SBA) (0—4). FBA, BLN and SBA define, respectively, the final bit, length, and starting bit of a requested data item as previously discussed with reference to PRMUX 20720. RBADL 23078 also receives chip select enable signals Address Translation Chip Select (ATCS) 00, 01, 02, 03, 04, and 15 from MIC 20122 and, in particular, RM 20722. When FIU 20120 is required to execute certain MSK 23018 operations, inputs FBA (0—4), BLN (0—4), and SBA (0—4), together with an ATCS input, are provided to RBADL 23078 from MIC 20122. RBADL 23078 in turn provides output RBA (Right Bit Address) (0—4), which has been described above with reference to DMSK (0—31) and RMSK (0—31). LBADL 23080 is similar to RBADL 23078 and is provided with inputs BLN (0—4), FBA (0—4), SBA (0—4), and ATCS 06, 07, 08, 09, and 05 from MIC 20122. Again, for certain MSK 23018 operations, LBADL 23080 will generate Left Bit Address (LBA) (0—4), which has been previously discussed above with reference to DMSK (0—31) and LMSK (0—31).

RBA (0—4) and LBA (0—4) are, respectively, transferred to RBAR 23086 and LBAR 23088 at start of second clock cycle by Pipeline Load Enable signal PIPELD provided from MIC 20122. RBAR 23086 and LBAR 23088 in turn respectively provide outputs Register Right Address (RRAD) (0—4) and Register Left Address (RLAD) (0—4) as address inputs to Right Mask Decode Logic (RMSKDL) 23092, Left Mask Decode Logic (LMSKDL) 23094, and FDL 23076 at start of second clock cycle. RRAD (0—4) and RLAD (0—4) correspond respectively to RBA (0—4) and LBA (0—4).

RMSKDL 23092 and LMSKDL 23094 are ROM arrays, having, as just described, RRAD (0—4) and RLAD (0—4) as, respectively, address inputs and Mask Enable (MSKENBL) from CPR 23084 as enable inputs. Together, RMSKDL 23092 and LMSKDL 23094 generate, respectively, RMSK (0—31) and LMSK (0—31) to MSK 23018. RMSK (0—31) and LMSK (0—31) are provided as inputs to Exclusive Or/Exclusive Nor gating (XOR/XNOR) 23096. XOR/XNOR 23096 also receives enable and selection signal Out Mask (OUTMSK) from CPR 23084. RMSK (0—31) and LMSK (0—31) inputs to XOR/XNOR 23096 are used, as selected by OUTMSK from CPR 23084, to generate a selected DMSK (0—31) as shown in Fig. 231. DMSK (0—31) output of XOR/XNOR 23096 is provided, as described above, to MSK 23018.

Referring again to IDL 23074, SHFAMTBL 23082 decodes certain control inputs from MIC 20122 to generate, through SHFR 23090, control inputs SHFT (0—4) and SGNSEL (0—4) to, respectively, DS 23016, SIGNSEL 23068 and MWG 23056. Address inputs to the PROMs comprising SHFAMTBL 23082 include FBA (0—4), SBA (0—4), and FLIPHALF (FLIPHALF) from MIC 20122. FBA (0—4) and SBA (0—4) have been described above. FLIPHALF is a control signal indicating that, as described above, that 16 bits of data

requested by IOS 10116 resides in the upper half of a 32 bit data word and causes those 16 bits to be transferred to the lower half of FIU 20120's output data word onto MIO Bus 10129. MIC 20122 also provides chip enable signals ATCS 10, 11, 12, 13, and 14. Upon receiving these control inputs from MIC 20122, SHFAMTDL 23082 generates an output shift amount (SHFAMT) (0—4) which, together with SBA (0—4) from MIC 20122, is transferred into SHFR 23090 by PIPELD at start of second clock cycle. SHFR 23090 then provides corresponding outputs SHFT (0—4) and SIGNSEL (0—4). As described above, SIGNSEL (0—4) are provided to SIGNSEL 23068 and MWG 23056 and MSK 23018. SHFT (0—4) is provided as SHFT (0—2) and SHFT (3—4) to, respectively, BYNL 23050 and BSL 23054 and DS 23016.

Referring to CPR 23084, as described above certain control signals are provided directly to FIU 20120 circuitry without being decoded by IDL 23074 or FDL 23076. Inputs to CPR 23084 include Sign Extension (SIGNEXT) and Lock (LOCK) indicating, respectively, that FIU 20120 is to perform a sign extension operation through MWG 23058 or a lock bit word operation through MWG 23056. CPR 23084 provides corresponding outputs SIGNEXT and LOCK to MSK 23018 to select these operations. Input Assembly Output Register (ASYMOR) and Blank Fill (BLANKFILL) are passed through CPR 23084 as ASYMOR and BLANKFILL to, respectively, MWG 23064 and MWG 23062 to select the output of ASYMR 23066 as a mask or to indicate that MSK 23018 is to generate a blank filled word through MWG 23062. Inputs OUTMSK and MSKENBL to CPR 23084 are provided, as discussed above, as enable signals OUTMSK and MSKENBL to, respectively, EXOR/ENOR 23096 and RMSKDL 23092 and LMSKBL 23094 and generating RMSK (0—31), LMSK (0—31), and DMSK (0—31) as described above.

Referring finally to ESPR 23098 and ESDL 23099, ESPR 23098 and PPLR 23072 together comprise a pipeline register and ESDL 23099 decoding logic for providing enable signals to FIU 20120 and other MEM 10112 circuitry. ESPR 23098 receives inputs Drive MOD Bus (DRVMOD) (0—1), Drive MIO Bus (DRVMIO) (0—1), and Enable Register (ENREG) (0—1) from MIC 20122 as previously described. DRVMOD (0—1), DRVMIO (0—1), and ENREG (0—1) are transferred into ESPR 23098 by PIPELD as previously described with reference to PPLR 23072. ESPR 23098 provides corresponding outputs to ESDL 23099, which in turn decodes DRVMOD (0—1), DRVMIO (0—1), and ENREG (0—1) to provide enable signals to FIU 20120 and other MEM 10112 circuitry. Outputs DRVSHFMOD, DRVASYMOD, DRVSHFMIO, and DRVASYMIO are provided to DSMOD 23036, DSMIO 23038, ASYMOD 23040, ASYMIO 23042, and FIUIO 23014 to control transfer of FIU 20120 manipulated data words onto MOD Bus 10144 and MIO Bus 10129. Outputs IARMEO, JWDEO, IWDEO, and RIDEO are provided as output enable signals to IARMR 23044, JWDR 23022, IWDR 23024, and RIDR 23026 to transfer the contents of these registers onto IB Bus 23030 as previously described. Outputs DRVCAMOD, DRVAMIO, DRVBYMOD, and DRVBYMIO are provided to MC 20116 for use in controlling transfer of information onto MOD Bus 10144 and MIO Bus 10129.

Having described the structure and operation of MEM 10112 above, the structure and operation of FU 10120 will be described next below.

B. Fetch Unit 10120 (Figs. 202, 206, 101, 103, 104, 238)

As has been previously described, FU 10120 is an independently operating, microcode controlled machine comprising, together with EU 10122, CS 10110's micromachine for executing user's programs. Principal functions of FU 10120 include: (1) Fetching and interpreting instructions, that is SINs comprising SOPs and Names, and data from MEM 10112 for use by FU 10120 and EU 10122; (2) Organizing and controlling flow and execution of user programs; (3) Initiating EU 10122 operations; (4) Performing arithmetic and logic operations on data; (5) Controlling transfer of data from FU 10120 and EU 10122 to MEM 10112; and, (6) Maintaining certain stack register mechanisms. Among these stack and register mechanisms are Name Cache (NC) 10226, Address Translation Cache (ATC) 10228, Protection Cache (PC) 10234, Architectural Base Registers (ABRs) 10364, Micro-Control Registers (mCRs) 10366, Micro-Stack (MIS) 10368, Monitor Stack (MOS) 10370 of General Register File (GRF) 10354, Micro-Stack Pointer Register Mechanism (MISPR) 10356, and Return Control Word Stack (RCWS) 10358. In addition to maintaining these FU 10120 resident stack and register mechanisms, FU 10120 generates and maintains, in whole or part, certain MEM 10112 resident data structures. Among these MEM 10112 resident data structures are Memory Hash Table (MHT) 10716 and Memory Frame Table (MFT) 10718, Working Set Matrix (WSM) 10720, Virtual Memory Management Request Queue (VMMRQ) 10721, Active Object Table (AOT) 10712, Active Subject Table (AST) 10914, and Virtual processor State Blocks (VPSBs) 10218. In addition, a primary function of FU 10120 is the generation and manipulation of logical descriptors which, as previously described, are the basis of CS 10110's internal addressing structure. As will be described further below, while FU 10120's internal structure and operation allows FU 10120 to execute arithmetic and logic operations, FU 10120's structure includes certain features to expedite generation and manipulation of logical descriptors.

Referring to Fig. 202, a partial block diagram of FU 10120 is shown. To enhance clarity of presentation, certain interconnections within FU 10120, and between FU 10120 and EU 10122 and MEM 10112 are not shown by line connections but, as described further below, are otherwise indicated, such as by common signal names. Major functional elements of FU 10120 include Descriptor Processor (DESP) 20210, MEM 10112 Interface Logic (MEMINT) 20212, and Fetch Unit Control Logic (FUCTL) 20214. DSP 20210 is, in general, an arithmetic and logic unit for generating and manipulating entries for MEM 10112 and FU 10120 resident stack mechanisms and caches, as described above, and, in particular, for generation and manipulation of logical descriptors. In addition, as stated above, DSP 20210 is a general purpose Central

Processor Unit (CPU) capable of performing certain arithmetic and logic functions.

DESP 20210 includes AON Processor (AONP) 20216, Offset Processor (OFFP) 20218, Length Processor (LENP) 20220. OFFP 20218 comprises a general, 32 bit CPU with additional structure to optimize generation and manipulation of offset fields of logical descriptors. AONP 20216 and LENP 20220 comprise, respectively, processors for generation and manipulation of AON and length fields of logical descriptors and may be used in conjuction with OFFP 20218 for execution of certain arithmetic and logical operations. DESP 20210 includes GRF 10354, which in turn include Global Registers (GRs) 10360 and Stack Registers (SRs) 10362. As previously described, GR's 10360 includes ABRs 10364 and mCRs 10366 while SRs 10362 includes MIS 10368 and MOS 10370.

MEMINT 20212 comprises FU 10120's interface to MEM 10112 for providing Physical Descriptors (physical addresses) to MEM 10112 to read SINs and data from and write data to MEM 10112. MEMINT 20212 includes, among other logic circuitry, MC 10226, ATC 10228, and PC 10234.

FUCTL 20214 controls fetching of SINs and data from MEM 10112 and provides sequences of microinstructions for control of FU 10120 and EU 10122 in response to SOPs. FUCTL 20214 provides Name inputs to MC 10226 for subsequent fetching of corresponding data from MEM 10112. FUCTL 20214 includes, in part, MISPR 10356, RCWS 10358, Fetch Unit S-Interpreter Dispatch Table (FUSDT) 11010, and Fetch Unit S-Interpreter Table (FUSITT) 11012.

Having described the overall structure of FU 10120, in particular with regard to previous descriptions in Chapter 1 of this description, DESP 20210, MEMINT 20212, and FUCTL 20214 will be described in further detail below, and in that order.

1. Description Processor 20210 (Figs. 202, 101, 103, 104, 238, 239)

As described above, DESP 20210 comprises a 32 bit CPU for performing all usual arithmetic and logic operations on data. In addition, a primary function of DESP 20210 is generation and manipulation of entries for, for example, Name Tables (NTs) 10350, ATC 10228, and PC 10234, and generation and manipulation of logical descriptors. As previously described, with reference to CS 10110 addressing structure, logical descriptors are logical addresses, or pointers, to data stored in MEM 10112. Logical descriptors are used, for example, as architectural base pointers or microcontrol pointers in ABRs 10364 and mCRs 10366 as shown in Fig. 103, or as linkage and local pointers of Procedure Frames 10412 as shown in Fig. 104. In a further example, logical descriptors generated by DESP 20210 and corresponding to certain operand Names are stored in MC 10226, where they are subsequently accessed by those Names appearing in SINs fetched from MEM 10112 to provide rapid translation between operand Names and corresponding logical descriptors.

As has been previously discussed with reference to CS 10110 addressing structure, logical descriptors provided to ATU 10228, from DESP 20210 or NC 10226, are translated by ATU 10228 to physical descriptors which are actual physical addresses of corresponding data stored in MEM 10112. That data subsequently is provided to JP 10114, and in particular to FU 10120 or EU 10122, through MOD Bus 10144.

As has been previously discussed with reference to MEM 10112, each data read to JP 10114 from MEM 10112 may contain up to 32 bits of information. If a particular data item referenced by a logical descriptor contains more than 32 bits of data, DESP 20210 will, as described further below, generate successive logical descriptors, each logical descriptor referring to 32 bits or less of information, until the entire data item has been read from MEM 10112. In this regard, it should be noted that NC 10226 may contain logical descriptors only for data items of 255 bits or less in length. All requests to MEM 10112 for data items greater than 32 bits in length are generated by DESP 20210. Most of data items operated on by CS 10110 will, however, be 32 bits or less in length so that NC 10226 is capable of handling most operand Names to logical descriptor translations.

As described above, DESP 20210 includes AONP 20216, OFFP 20218, and LENP 20220. OFFP 20218 comprises a general purpose 32 bit CPU with additional logic circuitry for generating and manipulating table and cache entries, as described above, and for generating and manipulating offset fields of AON pointers and logical descriptors. AONP 20216 and LENP 20220 comprise logic circuitry for generating and manipulating, respectively, AON and length fields of AON pointers and logical descriptors. As indicated in Fig. 202, GRF 10354 is vertically divided in three parts. A first part resides in ANOP 20216 and, in additon to random data, contains AON fields of logical descriptors. Second and third parts reside, respectively, in OFFP 20218 and LENP 20220 and, in addition to containing random data, respectively contain offset and length fields of logical descriptors. AON, Offset, and length portions of GRF 10354 residing respectively in AONP 20216, OFFP 20218, and LENP 20220 are designated, respectively, as AONGRF, OFFGRF, and LENGRF. AONGRF portion of GRF 10354 is 28 bits wide while OFFGRF and LENGRF portions of GRF 10354 are 32 bits in width. Although shown as divided vertically into three parts, GRF 10354 is addressed and operates as a unitary structure. That is, a particular address provided to GRF 10354 will address corresponding horizontal segments of each of GRF 10354's three sections residing in AONP 20216, OFFP 20218, and LENP 20220.

a. Offset Processor 20218 Structure

Referring first to OFFP 20218, in addition to being a 32 bit CPU and generating and manipulating table and cache entries and offset fields of AON pointers and logical descriptors, OFFP 20218 is DESP 20210's

primary path for receiving data from and transferring data to MEM 10112. OFFP 20218 includes Offset Input Select Multiplexer (OFFSEL) 20238, OFFGRF 20234, Offset Multiplexer Logic (OFFMUX) 20240, Offset ALU (OFFALU) 20242, and Offset ALU A Inputs Multiplexer (OFFALUSA) 20244.

OFFSEL 20238 has first and second 32 bit data inputs connected from, respectively, MOD Bus 10144 and JPD Bus 10142. OFFSEL 20238 has a third 32 bit data input connected from a first output of OFFALU 20242, a fourth 28 bit data input connected from a first output of AONGRF 20232, and a fifth 32 bit data input connected from OFFSET Bus 20228. OFFSEL 20238 has a first 32 bit output connected to input of OFFGRF 20234 and a second 32 bit output connected to a first input of OFFMUX 20240. OFFMUX 20240 has second and third 32 bit data inputs connected from, respectively, MOD Bus 10144 and JPD Bus 10142. OFFMUX 20240 also has a fourth 5 bit data input connected from Bias Logic (BIAS) 20246 and LENP 20220, described further below, and fifth 16 bit data input connected from NAME Bus 20224. Thirty-two bit data output of OFFGRF 20234 and first 32 bit data output of OFFMUX 20240 are connected to, respectively, first and second data inputs of OFFALUSA 20244. A first 32 bit data output of OFFALUSA 20244 and a second 32 bit data output of OFFMUX 20240 are connected, respectively, to first and second data inputs of OFFALU 20242. A second 32 bit data output of OFFALUSA 20244 is connected to OFFSET Bus 20228. A first 32 bit data output of OFFALU 20242 is connected to JPD Bus 10142, to a first input of AON Input Select Multiplexer (AONSEL) 20248 and AONP 20216, and, as described above, to a third input of OFFSEL 20238. A second 32 bit data output of OFFALU 20242 is connected to OFFSET Bus 20228 and third 16 bit output is connected to NAME Bus 20224.

b. AON Processor 20216 Structure

Referring to AONP 20216, a primary function of AONP 20216 is that of containing AON fields of AON pointers and logical descriptors. In addition, those portions of AONGRF 20232 not otherwise occupied by AON pointers and logical descriptors may be used as a 28 bit wide general register area by JP 10114. These portions of AONGRF 20232 may be so used either alone or in conjunction with corresponding portions of OFFGRF 20234 and LENGRF 20236. AONP 20216 includes AONSEL 20248 and AONGRF 20232. As previously described, a first 32 bit data input AONSEL 20248 is connected from a first data output of OFFALU 20242. A second 28 bit data input of AONSEL 20248 is connected from 28 bit output of AONGRF 20232 and from AON Bus 20230. A third 28 bit data input of AONSEL 20248 is connected from logic zero, that is a 28 bit input wherein each input bit is set to logic zero. Twenty-eight bit data output of AONSEL 20248 is connected to data input of AONGRF 20232. As just described, 28 bit data output of AONGRF 20232 is connected to second data input of AONSEL 20248, and is connected to AON Bus 20230.

c. Length Processor 20220 Structure

Referring finally to LENP 20220, a primary function of LENP 20220 is the generation manipulation of length fields of AON pointers and physical descriptors. In addition, LENGRF 20236 may be used, in part, either alone or in conjunction with corresponding address spaces of AONGRF 20232 and OFFGRF 20234, as general registers for storage of data. LENP 20220 includes Length Input Select Multiplexer (LENSEL) 20250, LENGRF 20236, BIAS 20246, and Length ALU (LENALU) 20252. LENSEL 20250 has first and second data inputs connected from, respectively, LENGTH Bus 20226 and OFFSET Bus 20228. LENGTH Bus 20226 is eight data bits, zero filled while OFFSET Bus 20228 is 32 data bits. LENSEL 20250 has a third 32 bit data input connected from data output of LENALU 20252. Thirty-two bit data output of LENSEL 20250 is connected to data input of LENGRF 20236 and to a first data input of BIAS 20246. Second and third 32 bit data inputs of BIAS 20246 are connected from, respectively, Constant (C) and Literal (L) outputs of FUSITT 11012 as will be described further below. Thirty-two bits data output of LENGRF 20236 is connected to JPD Bus 10142, to Write Length Input (WL) input of NC 10226, and to a first input of LENALU 20252. Five bit output of BIAS 20246 is connected to a second input of LENALU 20252, to LENGTH Bus 20226, and, as previously described, to a fourth input of OFFMUX 20240. Thirty-two bit output of LENALU 20252 is connected, as stated above, to third input of LENSEL 20250.

Having described the overall operation and the structure of DESP 20210, operation of DESP 20210 will be described next below in further detail.

d. Descriptor Processor 20210 Operation

a.a. Offset Selector 20238

Referring to OFFP 20218, GRF 10354 includes GR's 10360 and SR's 10362. GR's 10360 in turn contain ABR's 10364, mCR's 10366, and a set of general registers. SR's 10362 include MIS 10368 and MOS 10370. GRF 10354 is vertically divided into three parts. AONGRF 20232 is 28 bits wide and resides in AONP 20216, LENGRF 10354 is 32 bits wide and resides in LENP 20220, and OFFGRF 20234 is 32 bits wide and resides in OFFP 20218. AONGRF 20232, OFFGRF 20234, and LENGRF 20236 may be comprised of Fairchild 93422s.

In addition to storing offset fields of AON pointers and logical descriptors, those portions of OFFGRF 20234 not reserved for ABR's 10365, mCR's 10366, and SR's 10362 may be used as general registers, alone or in conjunction with corresponding portions AONGRF 20232 and LENGRF 20236, when OFFP 20218 is being utilized as a general purpose, 32 bit CPU. OFFGRF 20234 as will be described further below, is addressed in parallel with AONGRF 20232 and LENGRF 20236 by address inputs provided from FUCTL 20214.

OFFSEL 20238 is a multiplexer, comprised for example of SN74S244s and SN74S257s, for selecting data inputs to be written into selected address locations of OFFGRF 20234. OFFSEL 20238's first data input is from MOD Bus 10144 and is the primary path for data transfer between MEM 10112 and DESP 20210. As previously described, each data read from MEM 10112 to JP 10114 is a single 32 bit word where between one and 32 bits may contain actual data. If a data item to be read from MEM 10112 contains more than 32 bits of data, successive read operations are performed until the entire data item has been transferred.

OFFSEL 20238's second data input is from JPD Bus 10142. As will be described further below, JPD Bus 10142 is a data transfer path by which data outputs of FU 10120 and EU 10122 are written into MEM 10112. OFFSEL 20238's input of JPD Bus 10142 thereby provides a wrap around path by which data present at outputs of FU 10120 or EU 10122 may be transferred back into DESP 20210 for further use. For example, as previously stated a first output of OFFALU 20242 is connected to JPD Bus 10142, thereby allowing data output of OFFP 20218 to be returned to OFFP 20218 for further processing, or to be transferred to AONP 20216 or LENP 20220 as will be described further below. In addition, output of LENGRF 20236 is also connected to JPD Bus 10142 so that length fields of AON pointers or physical descriptors, or data, may be read from LENGRF 20236 to OFFP 20218. This path may be used, for example, when LENGRF 20236 is being used as a general purpose register for storing data or intermediate results of arithmetic or logical operations.

OFFSEL 20238's third input is provided from OFFALU 20242's output. This data path thereby provides a wrap around path whereby offset fields or data residing in OFFGRF 20234 may be operated on and returned to OFFGRF 20234, either in the same address location as originally read from or to a different address location. OFFP 20218 wrap around path from OFFALU 20242's output to OFFSEL 20238's third input, and thus to OFFGRF 20234, may be utilized, for example, in reading from MEM 10112 a data item containing more than 32 bits of data. As previously described, each read operation from MEM 10112 to JP 10114 is of a 32 bit word wherein between one and 32 bits may contain actual data. Transfer of a data word containing more than 32 bits is accomplished by performing a succession of read operations from MEM 10112 to JP 10114. For example, if a requested data item contains 70 bits of data, that data item will be transferred in three consecutive read operations. First and second read operations will each transfer 32 bits of data, and final read operation will transfer the remaining 6 bits of data. To read a data item of greater than 32 bits from MEM 10112 therefore, DESP 20210 must generate a sequence of logical descriptors, each defining a successive 32 bit segment of that data item. Final logical descriptor of the sequence may define a segment of less than 32 bits, for example, six bits as in the example just stated. In each successive physical descriptor, offset field must be incremented by value of length field of the preceding physical descriptor to define starting addresses of successive data items segments to be transferred. Length field of succeeding physical descriptors will, in general, remain constant at 32 bits except for final transfer which may be less than 32 bits. Offset field will thereby usually be incremented by 32 bits at each transfer until final transfer. OFFP 20218's wrap around data path from OFFALU 20242's output to their input of OFFSEL 20238 may, as stated above, be utilized in such sequential data transfer operations to write incremented or decremented offset field of a current physical descriptor back into OFFGRF 20234 to be offset field of a next succeeding physical descriptor.

In a further example, OFFP 20218's wrap around path from OFFALU 20242's output to third input of OFFSEL 20238 may be used in resolving Entries in Name Tables 10350, that is Name resolutions. In Name resolutions, as previously described, offset fields of AON pointers, for example Linkage Pointers 10416, are successively added and subtracted to provide a final AON pointer to a desired data item.

OFFSEL 20238's fourth input, from AONGRF 20232's output, may be used to transfer data or AON fields from AONGRF 20232 to OFFGRF 20234 or OFFMUX 20240. This data path may be used, for example, when OFFP 20218 is used to generate AON fields of AON pointers or physical descriptors or when performing Name evaluations.

Finally, OFFSEL 20238's fifth data input from OFFSET Bus 20228 allows offset fields on OFFSET Bus 20228 to be written into OFFGRF 20234 or transferred into OFFMUX 20240. This data path may be used, for example, to copy offset fields to OFFGRF 20234 when JP 10114 is performing a Name evaluation.

Referring now to OFFMUX 20240, OFFMUX 20240 includes logic circuitry for manipulating individual bits of 32 bit words. OFFMUX 20240 may be used, for example, to increment and decrement offset fields by length fields when performing string transfers, and to generate entries for, for example, MHT 10716 and MFT 10718. OFFMUX 20240 may also be used to aid in generating and manipulating AON, OFFSET, and LENGTH fields of physical descriptors and AON pointers.

b.b. Offset Multiplexer 20240 Detailed Structure (Fig. 238)

Referring to Fig. 238, a more detailed, partial block diagram of OFFMUX 20240 is shown. OFFMUX 20240 includes Offset Multiplexer Input Selector (OFFMUXIS) 23810, which for example may be comprised of SN74S373s and SN74S244s and Offset Multiplexer Register (OFFMUXR) 23812, which for example may be comprised of SN74S374s. OFFMUX 20240 also includes Field Extraction Circuit (FEXT) 23814, which may for example be comprised of SN74S257s, and Offset Multiplexer Field Selector (OFFMUXFS) 23816, which for example may be comprised of SN74S257s and SN74S374s. Finally, OFFMUX 20240 includes Offset Scaler (OFFSCALE) 23818, which may for example be comprised of AMD 25S10s, Offset Inter-element Spacing Encoder (OFFIESENC) 23820, which may for example be comprised of Fairchild 93427s

and Offset Multiplexer Output Selector (OFFMUXOS) 23822, which may for example be comprised of AMD 25Ss, Fairchild 93427s, and SN74S244s.

Referring first to OFFMUX 20240's connections to other portions of OFFP 20218, OFFMUX 20240's first data input, from OFFSEL 20238, is connected to a first input of OFFMUXIS 23810. OFFMUX 20240's second input, from MOD Bus 10144, is connected to a second input of OFFMUXIS 23810. OFFMUX 20240's third input, from JPD Bus 10142, is connected to a first input of OFFMUXFS 23816 while OFFMUX 20240's fourth input, from BIAS 20246, is connected to a first input of OFFMUXOS 23822. OFFMUX 20240's fifth input, from NAME Bus 20224, is connected to a second input of OFFMUXFS 23816. OFFMUX 20240's first output, to OFFALUSA 20244, is connected from output of OFFMUXR 23812 while OFFMUX 20240's second output, to OFFALU 20242, is connected from output of OFFMUXOS 23822.

Referring to OFFMUX 20240's internal connections, 32 bit output of OFFMUXIS 23810 is connected to input OFFMUXR 23812 and 32 bit output of OFFMUXR 23812 is connected, as described above, as first output of OFFMUX 20240, and as a third input of OFFMUXFS 23816. Thirty-two bit output of OFFMUXR 23812 is also connected to input of FEXT 23814. OFFMUXFS 23816's first, second and third inputs are connected as described above. A fourth input of OFFMUXFS 23816 is a 32 bit input wherein 31 bits are set to logic zero and 1 bit to logic 1. A fifth input is a 32 bit input wherein 31 bits are set to logic 1 and 1 to logic 0. A sixth input of OFFMUXFS 23816 is a 32 bit literal (L) input provided from FUSITT 11012 and is a 32 bit binary number comprising a part of a microinstruction FUCTL 20214, described below. OFFMUXFS 23816's seventh and eighth input are connected from FEXT 23814. Input 7 comprises FIU and TYPE fields of Name Table Entries which have been read into OFFMUXR 23812. Input 8 is a general purpose input conveying bits extracted from a 32 bit word captured in OFFMUXR 23812. As indicated in Fig. 238, OFFMUXFS 23816's first, third, fourth, fifth, and sixth inputs are each 32 bit inputs which are divided to provide two 16 bit inputs each. That is, each of these 32 bit inputs is divided into a first input comprising bit 0 to 15 of that 32 bit input, and a second input comprising bits 16 to 31.

Thirty-two bit output of OFFMUXFS 23816 is connected to inputs of OFFSCALE 23818 and OFFIESENC 23820. As indicated in Fig. 238, Field Select Output (FSO) of OFFMUXFS 23816 is a 32 bit word divided into a first word including 0 to 15 and a second word including bits 16 to 31. Output FSO of OFFMUXFS 23816, as will be described further below, thereby reflects the divided structure of OFFMUXFS 23816's first, third, fourth, fifth, and sixth inputs.

Logical functions performed by OFFMUXFS 23816 in generating output FSO, and which will be described in further detail in following descriptions, include:

(1) Passing the contents of OFFMUXR 23812 directly through OFFMUXFS 23816;

(2) Passing a 32 bit word on JPD Bus 10142 directly through OFFMUXFS 23816;

(3) Passing a literal value comprising a part of a microinstruction from FUCTL 20214 directly through OFFMUXFS 23816;

(4) Forcing FSO to be literal values 0000 0000;

(5) Forcing FSO to be literal value 0000 001;

(6) Extracting Name Table Entry fields;

(7) Accepting a 32 bit word from OFFMUXR 23812 or JPD Bus 10142, or 32 bits of a microinstruction from FUCTL 20214, and passing the lower 16 bits while forcing the upper 16 bits to logic 0;

(8) Accepting a 32 bit word from OFFMUXR 23812 or JPD Bus 10142, or 32 bits of miroinstruction from FUCTL 20214, and passing the higher 16 bits while forcing the lower 16 bits to logic 0;

(9) Accepting a 32 bit word from OFFMUXR 23812, or JPD Bus 10142, or Name Bus 20224, or 32 bits of a microinstruction from FUCTL 20214, and passing the lower 16 bits while sign extending bit 16 to the upper 16 bits; and,

(10) Accepting a 32 bit word from Name Bus 20224 and passing the lowest 8 bits while sign extending bit 24 to the highest 24 bits.

Thirty-two bit output of OFFSCALE 23818 and 3 bit output of OFFIESENC 23820 are connected, respectively, to second and third inputs of OFFMUXOS 23822. OFFMUXOS 23822's first input is, as described above, OFFMUX 20240's fourth input and is connected from output BIAS 20246. Finally, OFFMUXOS 23822's 32 bit output, OFFMUX (0—31) is OFFMUX 20240's second output and as previously described as connected to a second input of OFFALU 20242.

c.c. Offset Multiplexer 20240 Detailed Operation

a.a.a. Internal Operation

Having described the structure of OFFMUX 20240 as shown in Fig. 238, operation of OFFMUX 20240 will be described below. Internal operation of OFFMUX 20240, as shown in Fig. 238, will be described first, followed by description of OFFMUX 20240's operation with regard to DESP 20210.

Referring first to OFFMUXR 23812, OFFMUXR 23812 is a 32 bit register receiving either a 32 bit word from MOD Bus 10144, MOD (0—31), or a 32 bit word received from OFFSEL 20238, OFFSEL (0—31), and is selected by OFFMUXIS 23810. OFFMUXR 23812 in turn provides those selected 32 bit words from MOD Bus 10144 or OFFSEL 20238 as OFFMUX 20240's first data output to OFFALUSA 20244, as FEXT 23814's input, and as OFFMUXFS 23816's third input. OFFMUXR 23812's 32 bit output to OFFMUXFS 23816 is provided as two parallel 16 bit words designated as OFFMUXR output (OFFMUXRO) (0—15) and (16—31). As described above, OFFMUXFS 23816's output to OFFALUSA 20244 from OFFMUXR 23812 may be right shifted 16

places and the highest 16 bits zero filled.

FEXT 23814 receives OFFMUXRO (0—15) and (16—31) from OFFMUXR 23812 and extracts certain fields from those 16 bit words. In particular, FEXT 23814 extracts FIU and TYPE fields from NT 10350 Entries which have been transferred into OFFMUXR 23812. FEXT 23814 may then provide those FIU and TYPE fields as OFFMUXFS 23816's seventh input. FEXT 23814 may, selectively, extract certain other fields from 32 bit words residing in OFFMUXR 23812 and provide those fields as OFFMUXFS 23816's eighth input.

OFFMUXFS 23816 operates as a multiplexer to select certain fields from OFFMUXFS 23816's eight inputs and provide corresponding 32 bit output words, Field Select Output (FSO), comprised of those selected fields from OFFMUXFS 23816's inputs. As previously described, FSO is comprised of 2, parallel 16 bit words, FSO (0—15) and FSO (16—31). Correspondingly, OFFMUX 20240's third input, from JPD Bus 10142, is a 32 bit input presented as two 16 bit words, JPD (0—15) and JPD (16—31). Similarly, OFFMUXFS 23816's fourth, fifth, and sixth inputs are each presented as 32 bit words comprised of 2, parallel 16 bit words, respectively, "0" (0—15) and (16—31), "1" (0—15) and (16—31), and L (0—15) and (16—31). OFFMUXFS 23816's second input, from NAME Bus 20224, is presented as a single 16 bit word, NAME (16—31), while OFFMUXFS 23816's inputs from FEXT 23814 are each less than 16 bits in width. OFFMUXFS 23816 may, for a single 32 bit output word, select FSO (0—15) to contain one of corresponding 16 bit inputs JPD (0—15), "0" (0—15), "1" (0—15), or L (0—15). Similarly, FSO (16—31) of that 32 bit output word may be selected to contain one of NAME (16—31), JPD (16—31), O (16—31), 1 (16—31), L (16—31), or inputs 7 and 8 from FEXT 23814. OFFMUXFS 23816 therefore allows 32 bit words, comprised of two 16 bit fields, to be generated from selected portions of OFFMUXFS 23816's inputs.

OFFMUXFS 23816 32 bit output is provided as inputs to OFFSCALE 23818 and OFFIESENC 23820. Referring first to OFFIESENC 23820, OFFIESENC 23820 is used, in particular, in resolving, or evaluating, NT 10350 Entries (NTEs) referring to arrays of data words. As indicted in Fig. 108, word D of an NTE contains certain information relating to inter-element spacing (IES) of data words of an array. Word D of an NTE may be read from MEM 10112 to MOD Bus 10144 and through OFFMUX 20240 to input of OFFIESENC 23820. OFFIESENC 23820 then examines word D's IES field to determine whether inter-element spacing of that array is a binary multiple, that is 1, 2, 4, 8, 16, 32, or 64 bits. In particular, OFFIESENC 23820 determines whether 32 bit word D contains logic zeros in the most significant 25 bits and a single logic one in the least significant 7 bits. If inter-element spacing is such a binary multiple, starting addresses of data words of that array may be determined by left shifting of index (IES) to obtain offset fields of physical addresses of words in the array and a slower and more complex multiplication operation is not required. In such cases, OFFIESENC generates a first output, IES Encodeable (IESENC) to FUCTL 20214 to indicate that inter-element spacing may be determined by simple left shifting. OFFIESENC 23820 then generates encoded output, Encoded IES (ENCIES), to OFFMUXOS 23822. ENCIES is then a coded value specifying the amount of left shift necessary to translate index (IES) value into offsets of words in that array. As indicated in Fig. 238, ENCIES is OFFMUXOS 23822's third input.

OFFSCALE 23818 is a left shift shift network with zero fill of least significant bits, as bits are left shifted. Amount of shift by OFFSCALE 23818 is selectable between zero and 7 bits. Thirty-two bit words transferred into OFFSCALE 23818 from OFFSCALE 23818 from OFFMUXFS 23816 may therefore be left shifted, bit by bit, to selectively reposition bits within that 32 bit input word. In conjunction with OFFMUXFS 23816, and a wrap around connection provided by OFFALU 20242's output to OFFSEL 20238, OFFSCALE 23818 may be used to generate and manipulate, for example, entries for MHT 10716, MFT 10718, AOT 10712, and AST 10914, and other CS 10110 data structures.

OFFMUXOS 23822 is a multiplexer having first, second, and third inputs from, respectively, BIAS 20246, OFFSCALE 23818, OFFIESENC 23820. OFFMUXOS 23822 may select any one of these inputs as OFFMUX 20240's second output, OFFMUX (0—31). As previously described, OFFMUX 20240's second output is connected to a second input of OFFALU 20242.

Having described internal of OFFMUX 20240, operation of OFFMUX 20240 with regard to overall operation of DESP 20210 will be described next below.

b.b.b. Operation Relative to Descriptor Processor 20210

OFFMUX 20240's first input, from OFFSEL 20238, allows inputs to OFFSEL to be transferred through OFFMUXIS 23810 and into OFFMUXR 23812. This input allows OFFMUXR 23812 to be loaded, under control of FUCTL 20214 microinstructions, with any input of OFFSEL 20238. In a particular example, OFFALU 20242's output may be fed back through OFFSEL 20238's third input and OFFMUX 20240's first input to allow OFFMUX 20240 and OFFALU 20242 to perform reiterative operations on a single 32 bit word.

OFFMUX 20240's second input, from MOD Bus 10144, allows OFFMUXR 23812 to be loaded directly from MOD Bus 10144. For example, NTEs from a currently active procedure may be loaded into OFFMUXR 23812 to be operated upon as described above. In addition, OFFMUX 20240's second input may be used in conjunction with OFFSEL 20238's first input, from MOD Bus 10144, as parallel input paths to OFFP 20218. These parallel input paths allow pipelining of OFFP 20218 operations by allowing OFFSEL 20238 and OFFGRF 20234 to operate independently from OFFMUX 20240. For example, FU 10120 may initiate a read operation from MEM 10112 to OFFMUXR 23812 during a first microinstruction. The data so requested will appear on MOD Bus 10144 during a second microinstruction and may be loaded into OFFMUXR 23812 through OFFMUX 20240's second input from MOD Bus 10144. Concurrently, FU 10120 may initiate, at start

of second microinstruction, an independent operation to be performed by OFFSEL 20238 and OFFGRF 20234, for example loading output of OFFALU 20242 into OFFGRF 20234. Therefore, by providing an independent path into OFFMUX 20240 from MOD Bus 10144, OFFSEL 20238 is free to perform other, concurrent data transfer operations while a data transfer from MOD Bus 10144 to OFFMUX 20240 is being performed.

OFFMUX 20240's third input, from JPD Bus 10142, is a general purpose data transfer path. For example, data from LENGRF 20236 or OFFALU 20242 may be transferred into OFFMUX 20240 through JPD Bus 10142 and OFFMUX 20240's third input.

OFFMUX 20240's fourth input is connected from BIAS 20246 and primarily used during string transfers as described above. That is, length fields of physical descriptors generated for a string transfer may be transferred into OFFMUX 20240 through OFFMUX 20240's fourth input to increment or decrement, offset fields of those physical descriptors in OFFALU 20242.

OFFMUX 20240's fifth input is connected from NAME Bus 20224. As will be described further below, Names are provided to NC 10226 by FUCTL 20214 to call, from MC 10226, logical descriptors corresponding to Names appearing on MOD Bus 10144 as part of sequences of SINs.

As each Name is presented to NC 10226, that Name is transferred into and captured in Name Trap (NT) 20254. Upon occurrence of an NC 10226 miss, that is NC 10226 does not contain an entry corresponding to a particular Name, that Name is subsequently transferred from NT 20254 to OFFMUX 20240 through NAME Bus 20224 and OFFMUX 20240's fifth input. That Name, which is previously described as an 8, 12, or 16 bit binary number, may then be scaled, that is multiplied by a NTE size. That scaled Name may then be added to Name Table Pointer (NTP) from mCRs 10366 to obtain the address of a corresponding NTE in an NT 10350. In addition, a Name resulting in a NC 10226 miss, or a page fault in the corresponding NT 10350, or requiring a sequence of Name resolves, may be transferred into OFFGRF 20234 from OFFMUX 20240, through OFFALU 20242 and OFFSEL 20238 third input. That Name may subsequently be read, or restored, from OFFGRF 20234 as required.

Referring now to outputs of OFFMUX 20240, OFFMUX 20240's first output, from OFFMUXR 23812, allows contents of OFFMUXR 23812 to be transferred to first input of OFFALU 20242 through OFFALUSA 20244. OFFMUX 20240's second output, from OFFMUXOS 23822, is provided directly to second input of OFFALU 20242. OFFALU 20242 may be concurrently provided with a first input from OFFMUXR 23812 and a second input, for example a manipulated offset field, from OFFMUXOS 23822.

Referring to OFFALUSA 20244, OFFALUSA 20244 is a multiplexer. OFFALUSA 20244 may select either output of OFFGRF 20234 or first output of OFFMUX 20240 to be either first input of OFFALU 20242 or to be OFFP 20218's output to OFFSET Bus 20228. For example, an offset field from OFFGRF 20234 may be read to OFFSET Bus 20228 to comprise offset field of a current logical descriptor, and concurrently read into OFFALU 20242 to be incremented or decremented to generate offset field of a subsequent logical descriptor in a string transfer.

OFFALU 20242 is a general purpose, 32 bit arithmetic and logic unit capable of performing all usual ALU operations. For example, OFFALU 20242 may add, subtract, increment, or decrement offset fields of logical descriptors. In addition, OFFALU 20242 may serve as a transfer path for data, that is OFFALU 20242 may transfer input data to OFFALU 20242's outputs without operating upon that data. OFFALU 20242's first output, as described above, is connected to JPD Bus 10142, to third input of OFFSEL 20238, and to first input of AONSEL 20248. Data transferred or manipulated by OFFALU 20242 may therefore be transferred on to JPD Bus 10142, or wrapped around into OFFP 20218 through OFFSEL 20238 for subsequent or reiterative operations. OFFALU 20242's output to AONSEL 20248 may be used, for example, to load AON fields of AON pointers or physical descriptors generated by OFFP 20218 into AONGRF 20232. In addition, this data path allows FU 10120 to utilize AONGRF 20232 as, for example, a buffer or temporary memory space for intermediate or final results of FU 10120 operations.

OFFALU 20242's output to OFFSET Bus 20228 allows logical descriptor offset fields to be transferred onto OFFSET Bus 20228 directly from OFFALU 20242. For example, a logical descriptor offset field may be generated by OFFALU 20242 during a first clock cycle, and transferred immediately onto OFFSET Bus 20228 during a second clock cycle.

OFFALU 20242's third output is to NAME Bus 20224. As will be described further below, NAME Bus 20224 is address input (ADR) to NC 10226. OFFALU 20242's output to NAME Bus 20224 thereby allows OFFP 20218 to generate or provide addresses, that is Names, to NC 10226.

Having described operation of OFFP 20218, operation of LENP 20220 will be described next below.

e. Length Processor 20220 (Fig. 239)

Referring to Fig. 202, a primary function of LENP 20220 is generation and manipulation of logical descriptor length fields, including length fields of logical descriptors generated in string transfers. LENP 20220 includes LENGRF 20236, LENSEL 20250, BIAS 20246, and LENALU 20252. LENGRF 20236 may be comprised, for example, of Fairchild 93422s. LENSEL 20250 may be comprised of, for example, SN74S257s, SN74S157s, and SN74S244s, and LENALU 20252 may be comprised of, for example, SN74S381s.

As previously described, LENGRF 20236 is a 32 bit wide vertical section of GRF 10354. LENGRF 20236 operates in parallel with OFFGRF 20234 and AONGRF 20232 and contains, in part, length fields of logical descriptors. In addition, also as previously described, LENGRF 20236 may contain data.

LENSEL 20250 is a multiplexer having three inputs and providing outputs to LENGRF 20236 and first input of BIAS 20246. LENSEL 20250's first input is from Length Bus 20226 and may be used to write physical descriptor or length fields from LENGTH Bus 20226 into LENGRF 20236 or into BIAS 20246. Such length fields may be written from LENGTH Bus 20226 to LENGRF 20236, for example, during Name evaluation or resolve operations. LENSEL 20250's second input is from OFFSET Bus 20228. LENSEL 20250's second input may be used, for example, to load length fields generated by OFFP 20218 into LENGRF 20236. In addition, data operated upon by OFFP 20218 may be read into LENGRF 20236 for storage through LENSEL 20250's second input.

LENSEL 20250's third input is from output of LENALU 20252 and is a wrap around path to return output of LENALU 20252 to LENGRF 20256. LENSEL 20250's third input may, for example, be used during string transfers when length fields of a particular logical descriptor is incremented or decremented by LENALU 20252 and returned to LENGRF 20236. This data path may also, for example, be used in moving a 32 bit word from one location in LENGRF 20236 to another location in LENGRF 20236. As stated above, LENSEL 20250's output is also provided to first input BIAS and allows data appearing at first, second, or third inputs of LENSEL 20250 to be provided to first input of BIAS 20246.

BIAS 20246, as will be described in further detail below, generates logical descriptor length fields during string transfers. As described above, no more than 32 bits of data may be read from MEM 10112 during a single read operation. A data item of greater than 32 bits in length must therefore be transferred in a series, or string, of read operations, each read operation transferring 32 bits or less of data. String transfer logical descriptor length fields generated BIAS 20246 are provided to LENGTH Bus 20226, to LENALU 20252 second input, and to OFFMUX 20240's fourth input, as previously described. These string transfer logical descriptor length fields, referred to as bias fields are provided to LENGTH Bus 20226 by BIAS 20246 to be length fields of the series of logical descriptors generated by DESP 20210 to execute a string transfer. These bias fields are provided to fourth input OFFMUX 20240 to increment or decrement offset fields of those logical descriptors, as previously described. These bias fields are provided to second input of LENALU 20252, during string transfers, to correspondingly decrement the length field of a data item being read to MEM 10112 in a string transfer. BIAS 20246 will be described in greater detail below, after LENALU 20252 is first briefly described.

a.a. Length ALU 20252

LENALU 20252 is a general purpose, 32 bit arithmetic and logic unit capable of executing all customary arithmetic and logic operations. In particular, during a string transfer of a particular data item LENALU 20252 receives that data items length field from LENGRF 20236 and successive bias fields from BIAS 20246. LENALU 20252 then decrements that logical descriptor's current length field to generate length field to be used during next read operation of the string transfer, and transfers new length field back into LENGRF 20236 through LENSEL 20250's third input.

b.b. BIAS 20246 (Fig. 239)

Referring to Fig. 239, a partial block diagram of BIAS 20246 is shown. BIAS 20246 includes Bias Memory (BIASM) 23910, Length Detector (LDET) 23912, Next Zero Detector (NXTZRO) 23914, and Select Bias (SBIAS) 23916. Input of LDET 23912 is first input of BIAS 20246 and connected from output of LENSEL 20250. Output of LDET 23912 is connected to data input of BIASM 23910, and data output of BIASM 23910 is connected to input of NXTZRO 23914. Output of NXTZRO 23914 is connected to a first input of SBIAS 23916. A second input of SBIAS 23916 is BIAS 20246's second input, L8, and is connected from an output of FUCTL 20214. A third input of SBIAS 23916 is BIAS 20246's third input, L, and is connected from yet another output of FUCTL 20214. Output of SBIAS 23916 is output of BIAS 20246 and, as described above, is connected to LENGTH Bus 20226, to a second input of LENALU 20252, and to fourth input of OFFMUX 20240.

BIASM 23910 is a 7 bit wide random access memory having a length equal to, and operating and addressed in parallel with, SR's 10362 of GRF 10354. BIASM 23910 has an address location corresponding to each address location of SR's 10362 and is addressed concurrently with those address locations in SR's 10362. BIASM 23910 may be comprised, for example, of AMD 27S03As.

BIASM 23910 contains a bias value of each logical descriptor residing in SR's 10362. As described above, a bias value is a number representing number of bits to be read from MEM 10112 in a particular read operation when a data item having a corresponding logical descriptor, with a length field stored LENGRF 20236, is to be read from MEM 10112. Initially, bias values are written into BIASM 23910, in a manner described below, when their corresponding length fields are written into LENGRF 20236. If a particular data item has a length of less than 32 bits, that data item's initial bias value will represent that data items actual length. For example, if a data item has a length of 24 bits the associated bias value will be a 6 bit binary number representing 24. That data item's length field in LENGRF 20236 will similarly contain a length value of 24. If a particular item has a length of greater than 32 bits for example, 70 bits as described in a previous example, that data item must be read from MEM 10112 in a string transfer operation. As previously described, a string transfer is a series of read operations transferring 32 bits at a time from MEM 10112, with a final transfer of 32 bits or less completing transfer of that data item. Such a data item's initial length field entry in LENGRF 20236 will contain, using the same example as previously described, a value of 70.

That data item's initial bias entry written into a corresponding address space of BIASM 23910 will contain a bias value of 32. That initial bias value of 32 indicates that at least the first read operation required to transfer that data item from MEM 10112 will transfer 32 bits of data.

When a data item having a length of less than 32 bits, for example 24 bits, is to be read from MEM 10112, that data item's bias value of 24 is read from BIASM 23910 and provided to LENGTH Bus 20226 as length field of logical descriptor for that read operation. Concurrently, that bias value of 24 is subtracted from that data items length field read from LENGRF 20236. Subtracting that bias value from that length value will yield a result of zero, indicating that no further read operations are required to complete transfer of that data item.

If a data item having, for example, a length of 70 bits is to be read from MEM 10112, that data item's initial bias value of 32 is read from BIASM 23910 to LENGTH Bus 20226 as length field of first logical descriptor of a string transfer. Concurrently, that data item's initial length field is read from LENGRF 20236. That data item's initial bias value, 32, is subtracted from that data item's initial length value, 70, and LENALU 20252. The result of that subtraction operation is the remaining length of data item to be transferred in one or more subsequent read operations. In this example, subtracting initial bias value from initial length value indicates that 38 bits of that data item remain to be transferred. LENALU 20252's output representing results of this subtraction, for example 38, are transferred to LENSEL 20250's third input to LENGRF 20236 and written into address location from which that data item's initial length value was read. This new length field entry then represents remaining length of that data item. Concurrently, LDET 23912 examines that residual length value being written into LENGRF 20236 to determine whether remaining length of that data item is greater than 32 bits or is equal to or less than 32 bits. If remaining length is greater than 32 bits, LDET 23912 generates a next bias value of 32, which is written into BIASM 23910 and same address location that held initial bias value. If remaining data item length is less than 32 bits, LDET 23912 generates a 6 bit binary number representing actual remaining length of data item to be transferred. Actual remaining length would then, again, be written into BIASM 23910 address location originally containing initial bias value. These operations are also performed by LDET 23912 in examining initial length field and generating a corresponding initial bias value. These read operations are continued as described above until LDET 23912 detects that remaining length field is 32 bits or less, and thus that transfer of that data item will be completed upon next read operation. When this event is detected, LDET 23912 generates a seventh bit input into BIASM 23910, which is written into BIASM 23910 together with last bias value of that string transfer, indicating that remaining length will be zero after next read operation. When a final bias value is read from BIASM 23910 at start of next read operation of that string transfer, that seventh bit is examined by NXTZRO 23914 which subsequently generates a test condition output, Last Read (LSTRD) to FUCTL 20214. FUCTL 20214 may then terminate execution of that string transfer after that last read operation, if the transfer has been successfully completed.

As previously described, the basic unit of length of a data item in CS 10110 is 32 bits. Accordingly, data items of 32 or fewer bits may be transferred directly while data items of more than 32 bits require a string transfer. In addition, transfer of a data item through a string transfer requires tracking of the transferred length, and remaining length to be transferred, of both the data item itself and the data storage space of the location the data item is being transferred to. As such, BIAS 20246 will store, and operate with, in the manner described above, length and bias fields of the logical descriptors of both the data item and the location the data item is being transferred to. FUCTL 20214 will receive an LSTRD test condition if bias field of source descriptor becomes zero before or concurrently with that of the destination, that is a completed transfer, or if bias field of destination becomes zero before that of the source, and may provide an appropriate microcode control response. It should be noted that if source bias field becomes zero before that of the destination, the remainder of the location that this data item is being transferred to will be filled and padded with zeros. If the data item is larger than the destination storage capacity, the destination location will be filled to capacity and FUCTL 20214 notified to initiate appropriate action.

In addition to allowing data item transfers which are insensitive to data item length, BIAS 20246 allows string transfers to be accomplished by short, tight microcode loops which are insensitive to data item length. A string transfer, for example, from location A to location B is encoded as:

(1) Fetch from A, subtract length from bias A, and update offset and length of a; and,

(2) Store to B, subtract length from bias B, and branch to (1) if length of B does not go to zero or fall through (end transfer) if length of B goes to zero. Source (A) length need not be texted as the microcode loop continues until length of B goes to zero; as described above, B will be filled and padded with zeros if length of A is less than length of B, or B will be filled and the string transfer ended if length of A is greater than or equal to length of B.

LDET 23912 and NXTZRO 23914 thereby allow FUCTL 20214 to automatically initiate a string transfer upon occurrence of a single microinstruction from FUCTL 20214 initiating a read operation by DESP 20210. That microinstruction initiating a read operation will then be automatically repeated until LSTRD to FUCTL 20214 from NXTZRO 23914 indicates that the string transfer is completed. LDET 23912 and NXTZRO 23914 may, respectively, be comprised for example of S74S260s, SN74S133s, SN74S51s, SN74S00s, SN74S00s, SN74S04s, SN74S02s, and SN74S32s.

Referring finally to SBIAS 23916, SBIAS 23916 is a multiplexer comprised, for example, of SN74S288s, SN74S374s, and SN74S244s. SBIAS 23916, under microinstruction control from FUCTL 20214, selects BIAS

20246's output to be one of a bias value from BIASM 23910, L8, or L. SBIAS 23916's first input, from BIASM 23910, has been described above. SBIAS 23916's second input, L8, is provided from FUCTL 20214 and is 8 bits of a microinstruction provided from FUSITT 11012. SBIAS 23916's second input allows microcode selection of bias values to be used in manipulation of length and offset fields of logical descriptors by LENALU 20252 and OFFALU 20242, and for generating entries to MC 10226. SBIAS 23916's third input, L, is similarly provided from FUCTL 20214 and is a decoded length value derived from portions of microinstructions in FUSITT 11012. These microcode length values represent certain commonly occurring data item lengths, for example length of 1, 2, 4, 8, 16, 32, and 64 bits. An L input representing a length of 8 bits, may be used for example in reading data from MEM 10112 on a byte by byte basis.

Having described operation of LENP 20220, operation of AONP 20216 will be described next below.

f. AON Processor 20216

a.a. AONGRF 20232

As described above, AONP 20216 includes AONSEL 20248 and AONGRF 20232. AONGRF 20232 is a 28 bit wide vertical section of GRF 10354 and stores AON fields of AON pointers and logical descriptors. AONSEL 20248 is a multiplexer for selecting inputs to be written into AONGRF 20232. AONSEL 20248 may be comprised, for example of SN74S257s. AONGRF 20232 may be comprised of, for example, Fairchild 93422s.

As previously described, AONGRF 20232's output is connected onto AON Bus 20230 to allow AON fields of AON pointers and logical descriptors to be transferred onto AON Bus 20230 from AONGRF 20232. AONGRF 20232's output, together with a bi-directional input from AON Bus 20230, is connected to a second input of AONSEL 20248 and to a fourth input of AONSEL 20238. This data path allows AON fields, either from AONGRF 20232 or from AON Bus 20230, to be written into AONGRF 20232 or AONGRF 20234, or provided as an input to OFFMUX 20240.

b.b. AON Selector 20248

AONSEL 20248's first input is, as previously described, connected from output of OFFALU 20242 and is used, for example, to allow AON fields generated or manipulated by OFFP 20218 to be written into AONGRF 20232. AONSEL 20248's third input is a 28 bit word wherein each bit is a logical zero. AONSEL 20248's third input allows AON fields of all zeros to be written into AONGRF 20232. An AON field of all zeros is reserved to indicate that corresponding entries in OFFGRF 20234 and LENGRF 20236 are neither AON pointers nor logical descriptors. AON fields of all zeros are thereby reserved to indicate that corresponding entries in OFFGRF 20234 and LENGRF 20236 contain data.

In summary, as described above, DESP 20210 includes AONP 20216, OFFP 20218, and LENP 20220. OFFP 20218 contains a vertical section of GRF 10354, OFFGRF 20234, for storing offset fields of AON pointers and logical descriptors, and for containing data to be operated upon by DESP 20210. OFFP 20218 is principal path for transfer of data from MEM 10112 to JP 10114 and is a general purpose 32 bit arithmetic and logic unit for performing all usual arithmetic and logic operations. In addition, OFFP 20218 includes circuitry, for example OFFMUX 20240, for generation and manipulation of AON, OFFSET, and LENGTH fields of logical descriptors and AON pointers. OFFP 20218 may also generate and manipulate entries for, for example, NC 10226, ATU 10228, PC 10234, AOT 10712, MHT 10716, MFT 10718, and other data and address structures residing in MEM 10112. LENP 20220 includes a vertical section of GRF 10354, LENGRF 20236, for storing length fields of logical descriptors, and for storing data. LENP 20220 further includes BIAS 20246, used in conjunction with LENGRF 20236 and LENALU 20252, for providing length fields of logical descriptors for MEM 10112 read operations and in particular automatically performing string transfers. AONP 20216 similarly includes a vertical section of GRF 10354, AONGRF 20232. A primary function AONGRF 20232 is storing and providing AON fields of AON pointers and logical descriptors.

Having described structure and operation of DESP 20210, structure and operation of Memory Interface (MEMINT) 20212 will be described next below.

2. Memory Interface 20212 (Figs. 106, 240)

MEMINT 20212 comrises FU 10120's interface to MEM 10112. As described above, MEMINT 20212 includes Name Cache (NC) 10226, Address Translation Unit (ATU) 10228, and Protection Cache (PC) 10234, all of which have been previously briefly described. MEMINT 20212 further includes Descriptor Trap (DEST) 20256 and Data Trap (DAT) 20258. Functions performed by MEMINT 20212 includes (1) resolution of Names to logical descriptors, by NC 10226; (2) translation of logical descriptors to physical descriptors, by ATU 10228; and (3) confirmation of access writes to objects, by PC 10234.

As shown in Fig. 202, NC 10226 adress input (ADR) is connected from NAME Bus 20224. NC 10226 Write Length Field Input (WL) is connected from LENGRF 20236's output. NC 10226's Write Offset Field Input (WO) and Write AON Field Input (WA) are connected, respectively, from OFFSET Bus 20228 and AON Bus 20230. NC 10226 Read AON Field (RA), Read Offset Field (RO), and Read Length Field (RL) outputs are connected, respectively, to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

DEST 20256's bi-directional AON (AON), Offset (OFF), and Length (LEN) ports are connected by bi-directional buses to and from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

PC 10234 has AON (AON) and Offset (OFF) inputs connected from, respectively, AON Bus 20230 and

OFFSET Bus 20228. PC 10234 has a Write Entry (WEN) input connected from JPD Bus 10142. ATU 10228 has AON (AON), Offset (OFF), and Length (LEN) inputs connected from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226. ATU 10228's output is connected to physical Descriptor (PD) Bus 10146.

Finally, DAT 20258 has a bi-directional port connected to and from JPD Bus 10142.

a.a. Description Trap 20256 and Data Trap 20258

Referring first to DST 20256 and DAT 20258, DST 20256 is a register for receiving and capturing logical descriptors appearing on AON Bus 20230, OFFSET Bus 20228, and Length Bus 20226. Similarly, DAT 20258 is a register for receiving and capturing data words appearing on JPD Bus 10142. DST 20256 and DAT 20258 may subsequently return captured logical descriptors or data words to, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226, and to JPD Bus 10142.

As previously described, many CS 10110 operations, in particular MEM 10112 and JP 10114 operations, are pipelined. That is, operations are overlapped with certain sets within two or more operations being executed concurrently. For example, FU 10120 may submit read request to MEM 10112 and, while MEM 10112 is accepting and servicing that request, submit a second read request. DEST 20256 and DAT 20258 assist in execution of overplapping operations by providing a temporary record of these operations. For example, a part of a read or write request to MEM 10112 by FU 10120 is a logical descriptor provided to ATU 10228. If, for example the first red request just referred to results in a ATU 10228 cache miss or a protection violation, the logical descriptor of that first request must be recovered for subsequent action by CS 10110 as previously described. That logical descriptor will have been captured and stored in DEST 20256 and thus is immediately available, so that DESP 20210 is not required to regenerate that descriptor. DAT 20258 serves a similar purpose with regard to data being written into MEM 10112 from JP 10114. That is, DAT 20258 receives and captures a copy of each 32 bit word transferred onto JPD Bus 10142 by JP 10114. In event of MEM 10112 being unable to accept a write request, that data may be subsequently reprovided from DAT 20258.

b.b. Name Cache 10226, Address Translation Unit 10228, and Protection Cache 10234 (Fig. 106)

Referring to NC 10226, ATU 10228, and PC 10234, these elements of MEMINT 20212 are primarily cache mechanisms to enhance the speed of FU 10120's interface to MEM 10112, and consequently of CS 10110's operation. As described previously, NC 10226 contains a set of logical descriptors corresponding to certain operand names currently appearing in a process being executed by CS 10110. NC 10226 thus effectively provides high speed resolution of certain operand names to corresponding logical descriptors. As described above with reference to string transfers, NC 10226 will generally contain logical descriptors only for data items of less than 256 bits length. NC 10226 read and write addresses are names provided on NAME Bus 20224. Name read and write addresses may be provided from DESP 20210, and in particular from OFFP 20218 as previously described, or from FUCTL 20214 as will be described in a following description of FUCTL 20214. Logical descriptors comprising NC 10226 entries, each entry comprising an AON field, an Offset field, a Length field, are written into NC 10226 through NC 10226 inputs WA, WO, and WL from, respectively, AON Bus 20230, OFFSET Bus 20228, and LENGRF 20236's output. Logical descriptors read from NC 10226 in response to names provided to NC 10226 ADR input are provided to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226 from, respectively, NC 10226 outputs RA, RO, and RL.

ATU 10228 is similarly a cache mechanism for providing high speed translation of logical to physical descriptors. In general, ATU 10228 will contain, at any given time, a set of logical to physical page number mappings for MEM 10112 read and write requests which are currently being made, or anticipated to be made, to MEM 10112 by JP 10114. As previously described, each physical descriptor is comprised of a Frame Number (FN) field, and Offset Within Frame (O) fields, and a Length field. As discussed with reference to string transfers, a physical descriptor length field, as in a logical descriptor length field, specify a data item of less than or equal to 32 bits length. Referring to Fig. 106C, as previously discussed a logical descriptor comprised of a 14 bit AON field, a 32 bit Offset field, and Length field, wherein 32 bit logical descriptor Offset field is divided into a 18 bit Page Number (P) field and a 14 bit Offset within Page (O) field. In translating a logicl into a physical descriptor, logical descriptor Length and O fields are used directly, as respectively, physical descriptor length and O fields. Logical descriptor AON and P fields are translated into physical descriptor FN field. Because no actual translation is required, ATU 10228 may provide logical descriptor L field and corresponding O field directly, that is without delay, to MEM 10112 as corresponding physical descriptor O and Length fields. ATU 10228 cache entries are thereby comprised of physical descriptor FN fields corresponding to AON and P fields of those logical descriptors for which ATU 10228 has corresponding entries. Because physical descriptor FN fields are provided from ATU 10228's cache, rather than directly as in physical descriptor O and Length fields, a physical descriptor's FN field will be provided to MEM 10112, for example, one clock cycle later than that physical descriptors O and Length fields, as has been previously discussed.

Referring to Fig. 202, physical descriptor FN fields to be written into ATU 10228 are, in general, generated by DESP 20210. FN fields to be written into ATU 10228 are provided to ATU 10228 Data Input (DI) through JPD Bus 10142. ATU 10228 read and write addresses are comprised of AON and P fields of logical

descriptors and are provided to ATU 10228's AON and OFF inputs from, respectively, AON Bus 20230 and OFFSET Bus 20228. ATU 10228 read and write addresses may be provided from DESP 20210 or, as described further below, from FUCTL 20214. ATU 10228 FN outputs, together with O and Length fields comprising a physical descriptor, are provided to PD Bus 10146.

PC 10234 is a cache mechanism for confirming active procedure's access rights to objects identified by logical descriptors generated as a part of JP 10114 read or write requests to MEM 10112. As previously described access rights to objects are arbitrated on the basis of subjects. A subject has been defined as a particular combination of a principal, process, and domain. A principal, process, and domain are each identified by corresponding UIDs. Each subject having access rights to an object is assigned an Active Subject Number (ASN) as described in a previous description of CS 10110's Protection Mechanism. The ASN of a subject currently active in CS 10110 is stored in ASN Register 10916 in FU 10120. Access rights of a currently active subject to currently active objects are read from those objects Access Control Lists (ACL) 10918 and stored in PC 10234. If the current ASN changes, PC 10234 is flushed of corresponding access right entries and new entries, corresponding to the new ASN, are written into PC 10234. The access rights of a particular current ASN to a particular object may be determined by indexing, or addressing, PC 10234 with the AON identifying that object. Addresses to write entries into or read entries from PC 10234 are provided to PC 10234 AON input from AON Bus 20230. Entries to be written into PC 10234 are provided to PC 10234's WEN input from JPD Bus 10142. PC 10234 is also provided with inputs, not shown in Fig. 202 for purposes of clarity, from FUCTL 20214 indicating the current operation to be perfomed by JP 10114 with respect to an object being presently addressed by FU 10120. Whenever FU 10120 submits a read or write request concerning a particular object to MEM 10112, AON field of that request is provided as an address to PC 10234. Access rights of the current active subject to that object are read from corresponding PC 10234 entry and compared to FUCTL 20214 inputs indicating the particular operation to be performed by JP 10114 with respect to that object. The operation to be performed by JP 10114 is then compared to that active subject's access rights to that object and PC 10234 provides an output indicating whether that active subject possesses the rights required to perform the intended operation. Indexing of PC 10234 and comparison of access rights to intended operation is performed concurrently with translation of the memory request logical descriptor to a corresponding physical descriptor by ATU 10228. If PC 10234 indicates that that active subject has the required access rights, the intended operation is executed by JP 10114. If PC 10234 indicates that that active subject does not have the required access rights, PC 10234 indicates that a protection mechanism violation has occurred and interrupts execution of the intended operation.

c.c Structure and Operation of a Generalized Cache and NC 10226 (Fig. 240)

Having described overall structure and operation of NC 10226, ATU 10228, and PC 10234, structure and operation of these caches will be described in further detail below. Structure and operation of NC 10226, ATU 10228, and PC 10234 are similar, except that NC 10226 is a four-way set associative cache, ATU 10228 is a threeway set associative cache and PC 10234 is a two-way set associative cache.

As such, the structure and operation of NC 10226, ATU 10228, and PC 10234 will be described by reference to and description of a generalized cache similar but not necessarily identical to each of NC 10226, ATU 10228, and PC 10234. Reference will be made to NC 10226 in the description of a generalized cache next below, both to further illustrate structure and operation of the generalized cache, and to describe differences between the generalized cache and NC 10226. ATU 10228 and PC 10234 will then be described by description of differences between ATU 10228 and PC 10234 and the generalized cache.

Referring to Fig. 240, a partial block diagram of a generalized four-way, set associative cache is shown. Tag Store (TS) 24010 is comprised of Tag Store A (TSA) 24012, Tag Store B (TSB) 24014, Tag Store C (TSC) 24016, and Tag Store D (TSD) 24018. Each of the cache's sets, represented by TSA 24012 to TSD 24018, may contain, for example as in NC 10226, up to 16 entries, so that TSA 24012 to TSD 24018 are each 16 words long.

Address inputs to a cache are divided into a tag field and an index field. Tag fields are stored in the cache's tag store and indexed, that is addressed to be read or written from or to tag store by index field of the address. A tag read from tag store in response to index field of an address is then compared to tag field of that address to indicate whether the cache contains an entry corresponding to that address, that is, whether a cache hit occurs. In, for example, NC 10226, a Name syllable may be comprised of an 8, 12, or 16 bit binary word, as previously described. The four least significant bits of these words, or Names, comprise NC 10226's index field while the remaining 4, 8, or 12 most significant bits comprise NC 10226's tag field. TSA 24012 to TDS 24018 may each, therefore, be 12 entry wide memories to store the 12 bit tag fields of 16 bit names. Index (IND) or address inputs of TSA 24012 to TSD 24018, would in NC 10226, be connected from four least significant bits of NAME Bus 20224 while Tag Inputs (TAGI) of TSA 24012 to TSD 24018 would be connected from the 12 most significant bits of NAME Bus 20224.

As described above, tag outputs of TS 24010 are compared to tag fields of addresses presented to the cache to determine whether the cache contains an entry corresponding to that address. Using NC 10226 as an example 12 bit Tag Outputs (TAGOs) of TSA 24012 to TSD 24018 are connected to first inputs of Tag Store Comparators (TSC) 24019, respectively to inputs of Tag Store Comparitor A (TSCA) 24020, Tag Store Comparitor B (TSCB) 24022, Tag Store Comparitor D (TSCD) 24024, and Tag Store Comparitor E (TSCE) 24026. Second 12 bit inputs of TSCA 24020 to TSCE 24026 may be connected from the 12 most significant

bits of NAME Bus 20224 to receive tag fields of NC 10226 addresses. TAS 24020 to TSCE 24026 compare tag field of an address to tag outputs read from TSA 24012 to TSE 24018 in response to index field of that address, and provide four bit outputs indicating which, if any, of the possible 16 entries and their associated tag store correspond to that address tag field. TSCA 24020 to TSCE 24026 may be comprised, for example, of Fairchild 93S46s.

Four bit outputs of TSCA 24012 to TSCE 24026 are connected in the generalized cache to inputs of Tag Store Pipeline Registers (TSPR) 24027; respectively to inputs of Tag Store Pipeline Register A (TSPRA) 24028, Tag Store Pipeline Register B (TSPRB) 24030, Tag Store Pipeline Register C (TSPRC) 24032, and Tag Store Pipeline Register D (TSPRD) 24034. ATU 10228 and PC 10234 is pipelined with a single cache access operation being executed in two clock cycles. During first clock cycle tag store is addressed and tags store therein compared to tag field of address to provide indication of whether a cache hit has occurred, that is whether cache contains an entry corresponding to a particular address. During second clock cycle, as will be described below, a detected cache hit is encoded to obtain access to a corresponding entry in cache data store. Pipeline operation over two clock cycles is provided by cache pipeline registers which include, in part, TSPRA 24028 to TSPRD 24034. NC 10226 is not pipelined and does not include TSPRA 24028 to TSPRD 24034. In NC 10226, outputs of TSCA 24012 to TSCD 24024 are connected directly to inputs of TSHEA 24036 to TSHED 24042, described below.

Outputs of TSPRA 24028 to TSPRD 24034 are connected to inputs of Tag Store Hit Encoders (TSHE) 24035, respectively to Tag Store Hit Encoder A (TSHEA) 24036, Tag Store Hit Encoder B (TSHEB) 24038, Tag Store Hit Encoder C (TSHEC) 24040, and Tag Store Hit Encoder D (TSHED) 24042. TSHEA 24036 to TSHED 24042 encode, respectively, bit inputs from TSPRA 24028 to TSPRD 24034 to provide single bit outputs indicating which, if any, set of the cache's four sets includes an entry corresponding to the address input.

Single bit outputs of TSHEA 24036 to TSHED 24042 are connected to inputs of Hit Encoder (HE) 24044. HE 24044 encodes single bit inputs from TSHEA 24036 to TSHED 24042 to provide two sets of ouputs. First outputs of HE 24044 are provided to Cache Usage Store (CUS) 24046 and indicate in which of the cache's four sets, corresponding to TSA 24012 to TSD 24018, a cache hit has occurred. As described previously with reference to MC 20116, and will be described further below, CUS 24046 is a memory containing information for tracking usage of cache entries. That is, CUS 24046 contains entries indicating whether, for a particular Index, Set A, Set B, Set C or Set D of the cache's four sets has been most recently used and which has been least recently used. CUS 24046 entries regarding Sets A, B, C, and D are stored in, respectively, memories CUSA 24088, CUSB 24090, CUSC 24092, and CUSD 24094. Second output of HE 24044, as described further below, is connected to selection input of Data Store Selection Multiplexer (DSSMUX) 24048 to select an output from Data Store (DS) 24050 to be provided as output of the cache when a cache hit occurs.

Referring to DS 24050, as previously described a cache's data store contains the information, or entries, stored in that cache. For example, each entry in NC 10226's DS 24050 is a logical descriptor comprised of an AON, and Offset, and Length. A cache's data store parallels, in structure and organization, that cache's tag store and entries therein are identified and located through that cache's tag store and associated tag store comparison and decoding logic. In NC 10226, for example, for each Name having an entry in NC 10226 there will be an entry, the tag field of that name, stored in TS 24010 and a corresponding entry, a logical descriptor corresponding to that Name, in DS 24050. As described above, NC 10226 is a four-way, set associative cache so that TS 24010 and DS 24050 will each contain four sets of data. Each set was previously described as containing up to 16 entries. DS 24050 is therefore comprised of four 16 word memories. Each memory is 65 bits wide, accommodating 28 bits of AON, 32 bits of offset, and 5 bits of length. These four component data store memories of DS 24050 are indicated in Fig. 240 as Data Store A (DSA) 24052, Data Store B (DSB) 24054, Data Store C (DSC) 24056, and Data Store D (DSD) 24058. DSA 24052, DSB 24054, DSC 24056 and DSD 24058 correspond, respectively, in structure, contents, and operation to TSA 24012, TSB 24014, TSC 24016 and TSD 24018.

Data Inputs (DIs) of DSA 24052 to DSD 24058 are, in NC 10226 for example, connected from AON Bus 20230, OFFSET Bus 20228, LENGTH Bus 20226 and comprise inputs WA, WO, WL respectively of NC 10226. DSA 24052 to DSD 24058 DIs are, in NC 10226 as previously described, utilized in writing NC 10226 entries into DSA 24052 to DSD 24058. Address inputs of DSA 24052 to DSD 24058 are connected from address outputs of Address Pipeline Register (ADRPR) 24060. As will be described momentarily, except during cache flush operations, DSA 24052 to DSD 24058 address inputs are comprised of the same index fields of cache addresses as are provided as address inputs to TS 24010, but are delayed by one clock cycle and ADRPR 24060 for pipelining purposes. As described above, NC 10226 is not pipelined and does not have the one clock cycle delay. An address input to the cache will thereby result in corresponding entries, selected by index field of that address, being read from TSA 24012 to TSD 24018 and DSA 24052 to DSD 24058. The four outputs of DSA 24052 to DSD 24058 selected by a particular index field of a particular address are provided as inputs to DSSMUX 24048. DSSMUX 24048 is concurrently provided with selection control input from HE 24044. As previously described, this selection input to DSSMUX 24048 is derived from TS 24010 tag entries and indicates which of DSA 24052 to DSD 24058 entries corresponds to an address provided to the cache. In response to that selection control input, DSSMUX 24048 selects one of DS 24050's four logical descriptor outputs as the cache's output corresponding to that address. DSSMUX 24048's output is then provided, through Buffer Driver (BD) 24062 as the cache's output, for example in NC 10226 to AON Bus 20230, OFFSET Bus 20228, and LENGTH Bus 20226.

Referring to ADRMUX 24062, ADRMUX 24062 selects one of two sources to provide address inputs to DS 24050, that is to index to DS 24050. As described above, a first ADRMUX 24062 input is comprised of the cache's address index fields and, for example in NC 10226, is connected from the four least significant bits of NAME Bus 20224. During cache flush operations, DS 24050 address inputs are provided from Flush Counter (FLUSHCTR) 24066, which in the example is a four bit counter. During cache flush operations, FLUSHCTR 24066 generates sequential bit addresses which are used to sequentially address DSA 24052 to DSD 24058. Selection between ADRMUX 24062 first and second inputs, respectively the address index fields and from FLUSHCTR 24066, is controlled by Address Multiplexer Select (ADRMUXS) from FUCTL 20214.

Validity Store (VALS) 24068 and Dirty Store (DIRTYS) 24070 are memories operating in parallel with, and addressed in parallel with TS 24010. VALS 24068 contains entries indicating validity of corresponding TS 24010 and DS 24050 entries. That is, VALS 24068 entries indicate whether corresponding entries have been written into corresponding locations in TS 24010 and DS 24050. In the example, VALS 24068 may thereby be a 16 word by 4 bit wide memory. Each bit of a VALS 24068 word indicates validity of a corresponding location in TSA 24012 and DSA 24052, TSB 24014 and DSB 24054, TSC 24016 and DSC 24056, and TSD 24018 and DSD 24058. DIRTYS 24070 similarly indicates whether corresponding entries in corresponding locations of TS 24010 and DS 24050 have been written over, or modified. Again, DIRTYS 24070 will be a sixteen word by four bit wide memory.

Address inputs of VALS 24068 and DIRTYS 24070 are, for example in NC 10226, connected from least significant bits of NAME Bus 20224 and are thus addressed by index fields of NC 10226 addresses in parallel with TS 24010. Outputs of VALS 24068 are provided to TSCA 24020 to TSEE 24026 to inhibit outputs of TSCA 24020 through TSCE 24026 upon occurrence of an invalid concurrence between a TS 24010 entry and a NC 10226 address input. Similar outputs of DIRTYS 24070 are provided to FUCTL 20214 for use in cache flush operations to indicate which NC 10226 entries are dirty and must be written back into an MT 10350 rather than disgarded.

Outputs of VALS 24068 and DIRTYS 24070 are also connected, respectively, to inputs of Validity Pipeline Register (VALPR) 24072 and Dirty Pipeline Register (DIRTYPR) 24074. VALPR 24072 and DIRTYPR 24074 are pipeline registers similar to TSPRA 24028 to TSPRD 24034 and are provided for timing purposes as will be described momentarily. Outputs of VALPR 24072 and DIRTYPR 24074 are connected to inputs of, respectively, Validity Write Logic (VWL) 24076 and Dirty Write Logic (DWL) 24078. As described above, NC 10226 is not a pipelined cache and does not include VALPR 24072 and DIRTYPR 24074; outputs of VALS 24068 and DIRTYS 24070 are connected directly to inputs of VWL 24076 and DWL 24078. Outputs of VWL 24076 and DWL 24078 are connected, respectively, to data inputs of VALS 24068 and DIRTYS 24070. Upon occurrence of a write operation to TS 24010 and DS 24050, that is writing in or modifying a cache entry, corresponding validity and dirty word entries are read from VALS 24068 and DIRTYS 24070 by index field of the caches input address. Outputs to VALS 24068 DIRTYS 24070 are received and stored in, respectively, VALPR 24070 and DIRTYPR 24074. At start of next clock cycle, validity and dirty words in VALPR 24072 and DIRTYPR 24074 are read into, respectively, VWL 24076 and DWL 24078. VWL 24076 and DWL 24078 respectively modify those validity or dirty word entries from VALS 24068 and DIRTYS 24070 in accordance to whether the corresponding entries in TS 24010 and DS 24050 are written into or modified. These modified validity and dirty words are then written, during second clock cycle, from VWL 24076 and DWL 24078 into, respectively, VALS 24068 and DIRTYS 24070. Control inputs of VWL 24076 and DWL 24078 are provided from FUCTL 20214.

Referring finally to Least Recent Used Logic (LRUL) 24080, LRUL 24080 tracks usage of cache entries. As previously described, the generalized cache of Fig. 240 is a four way, set associative cache with, for example, up to 16 entries in each of NC 10226's sets. Entries within a particular set are identified, as described above, by indexing the cache's TS 24010 and DS 24050 may contain, concurrently, up to four individual entries identified by the same index but distiguished by having different tags. In this case, one entry would reside in Set A, comprising TSA 24012 and DSA 24052, one in Set B, comprising TSB 24014 and DSB 24054, and so on. Since the possible number of individual entries having a common tag is greater than the number of cache sets, it may be necessary to delete a particular cache entry when another entry having the same tag is to be written into the cache. In general, the cache's least recently used entry would be deleted to provide a location in TS 24010 and DS 24050 for writing in the new entry. LRUL 24080 assists in determining which cache entries are to be deleted when necessary in writing in a new entry by tracking and indicating relative usage of the cache's entries. LRUL 24080 is primarily comprised of a memory, LRU Memory (MLRU) 24081, containing a word for each cache set. As described above, NC 10226, for example, includes 16 sets of 4 frames each, so that LRUL 24080's memory may correspondingly be, for example, 16 words long. Each word indicates relative usage of the 4 frames in a set and is a 6 bit word.

Words are generated and written into LRUL 24080's MLRU 24081, through Input Register A, B, C, D (RABCD) 24083, according to a write only algorithm executed by HE 24044, as described momentarily. Each bit of each six word pertains to a pair of frames within a particular cache set and indicates which of those two frames was more recently used than the other. For example, Bit 0 will contain logic 1 if Frame A was used more recently than Frame B and a logic zero if Frame B was used more recently than Frame A. Similarly, Bit 1 pertains to Frames A and C, Bit 2 to Frames A and D, Bit 3 to Frames B and C, Bit 4 to Frames B and D, and Bit 5 to Frames C and D. Initially, all bits of a particular LRUL 24080 word are set to zero.

Assuming, for example, that the frames of a particular set are used in the sequence Frame A, Frame D, Frame B; Bits 0 to 5 of that LRUL 24080 word will initially contain all zeros. Upon a reference to Frame A, Bits 0, 1, and 2, referring respectively to Frames A and B, Frames A and C, and Frames A and D, will be written as logic 1's. Bits 3, 4, and 5, referring respectively to Frames B and C, Frames B an D, and Frames C and D, will remain logic 0. Upon reference to Frame D, Bits 0 and 1, referring respectively to Frames A and B and Frames A and C, will remain logic 1's. Bit 2, referring to Frames A and D, will be changed from logic 1 to logic 0 to indicate that Frame D has been referred to more recently than Frame A. Bit 3, referring to Frames B and C, will remain logic 0. Bits 4 and 5, referring respectively to Frames B and D and Frames C and D, will be written as logic 0, although they are already logic zeros, to indicate respectively that Frame D has been used more recently than Frame B or Frame C. Upon reference to Frame B, Bit 0, referring to Frames A and B, will be written to logic 0 to indicate that Frame B has been used more recently than Frame A. Bits 1 and 2, referring resectively to Frames A and C and Frames A and D, will remain respectively as logic 1 and logic 0. Bits three and four, referring respectively to Frames B and C and Frames B and D, will be written as logics 1's to indicate respectively that Frame B has been used more recently than Frame C or Frame D. Bit five will remain logic 0.

When it is necessary to replace a cache entry in a particular frame, the LRUL 24080 word referring to the cache set containing that frame will be read from LRUL 24080's MLRL 24081 through LRU Register (RLRU) 24085 and decoded by LRU Decode Logic (LRUD) 24087 to indicate which is least recently used frame. This decoding is executed by means of a Read Only Memory operating as a set of decoding gating.

Having described the structure and operation of a generalized cache as shown in Fig. 240, with references to NC 10226 for illustration and to point out differences between the generalized cache and NC 10226, structure and operation of ATU 10228 and PC 10234 will be described next below. ATU 10228 and PC 10234 will be described by describing the differences between ATU 10228 and PC 10234 and the generalized cache and NC 10226. ATU 10228 will be described first, followed by PC 10234.

d.d. Address Translation Unit 10228 and Protection Cache 10234

ATU 10228 is a three-way, set associative cache of 16 sets, that is contains 3 frames for each set. Structure and operation of ATU 10228 is similar to the generalized cache described above. Having 3 rather than 4 frames per set, ATU 10228 does not include a STD 24018, ATSCE 24026, ATSPRD 24034, ATSHED 24042, or ADSD 24058. As previously described ATU 10228 address inputs comprise AON and O fields of logical descriptors. AON fields are each 28 bits and O fields comprise the 18 most significant bits of logical descriptor offset fields, so that ATU 10228 address inputs are 48 bits wide. Four least significant bits of O fields are used as index. AON fields and the 14 most significant bits of O field comprise ATU 10228's tags. ATU 10228 tags are thereby each 42 bits in width. Accordingly, TSA 24012, TSB 24014, and TSC 24016 of ATU 10228's TS 24010 are each 16 words long by 42 bits wide.

DSA 24052, DSB 24054, and DSC 24056 of ATU 10228 are each 16 bits long. ATU 10228 outputs are, as previously described, physical descriptor Frame Number (FN) fields of 13 bits each. ATU 10228's DSA 24052, DSB 24054, DSC 24056 are thereby each 13 bits wide.

ATU 10228's LRUL 24080 is similar in structure and operation to that of the generalized cache. ATU 12028's LRUL 24080 words, each corresponding to an ATU 10228 set, are each 3 bits in width as 3 bits are sufficient to indicate relative usage of frames within a 3 frame set. In ATU 10228, Bit 1 of an LRUL 24080 word indicates whether Frame A was used more recently than Frame B, Bit 2 whether Frame A was used more recently than Frame C, and Bit 3 whether Frame B was used more recently than Frame C. In all other respects, other than as stated above, ATU 10228 is similar in structure and operation to the generalized cache.

Referring to PC 10234, PC 10234 is a two-way, set associative cache of 8 sets, that is has two frames per set. Having 2 rather than 4 frames, PC 10234 will not include a TSC 24016, a TSD 24018, a TSCC 24024, a TSCD 24026, a TSPRC 24032, a TSPRD 24034, a TSHEC 24040, a TSHED 24042, a DSC 24056, or a DSD 24058.

Address inputs of PC 10234 are the 28 bit AON fields of logical descriptors. The 3 least significant bits of those AON fields are utilized as indexes for addressing PC 10234's TS 24010 and DS 24050. The 25 most significant bits of those AON field address inputs are utilized as PC 10234's tags, so that PC 10234's TSA 24012 and TSB 24014 are each 8 word by 25 bit memories.

Referring to PC 10234's LRUL 24080, a single bit is sufficient to indicate which of the two frames in each of PC 10234's sets was most recently accessed. PC 10234's LRUL 24080's memory is thereby 8 words, or sets long, one bit wide.

As previously described, PC 10234 entries comprise information regarding access rights of certain active subjects to certain active objects. Each PC 10234 entry contains 35 bits of information. Three bits of this information indicate whether a particular subject was read, write, or execute rights relative to a particular object. The remaining 32 bits effectively comprise a length field indicating the volume or portion, that is the number of data bits, of that object to which those access rights pertain.

Referring again to Fig. 240, PC 10234 differs from the generalized cache and from NC 10226 and ATU 10228 in further including Extent Check Logic (EXTCHK) 24082 and Operation Check Logic (OPRCHK) 24084. PC 10234 entries include, as described above, 3 bits identifying type of access rights a particular subject has to a particular object. These 3 bits, representing a Read (R), Write (W), or Execute (E) right, are provided to a

first input of OPRCHK 24084. A second input of OPRCHK 24084 is provided from FUCTL 20214 and specifies whether JP 10114 intends to perform a Read (RI), a Write (WI), or Execute (EI), operation with respect to that object. OPRCHK 24084 compares OPRCHK 24084 access right inputs from DS 24050 to OPRCHK 24084's intended operation input from FUCTL 20214. If that subject does not possess the rights to that object which are required to perform the operation intended by JP 10114, OPRCHK 24084 generates an Operation Violation (OPRV) indicating that a protection violation has occurred.

Similarly, the 32 bits of a PC 10234 entry regarding extent rights is provided as an input (EXTENT) to EXTCHK 24082. As stated above, EXTENT field of PC 10234 entry indicates the length or number of data bits, within an obect, to which those access rights pertain. EXTENT field from PC 10234 entry is compared, by EXTCHK 24082, to offset field of the logical descriptor of the current JP 10114 request to MEM 10112 for which a current protection mechanism check is being made. If comparison of extent rights and offset field indicate that the current memory request goes beyond the object length to which the corresponding rights read from DS 24050 apply, EXTCHK 24082 generates an Extent Violation (EXTV) output. EXTV indicates that a current memory request by JP 10114 refers to a portion of an object to which the PC 10234 entry read from BS 24050 does not apply. As described previously, each read from or write to MEM 10112, even as part of a string transfer, is a 32 bit word. As such, EXTCHK 24082 will generate an EXTV output when OFFSET field of a current logical descriptor describes a segment of an object less than 32 bits from the limit defined by EXTENT field of the PC 10234 entry provided in response to that logical descriptor. EXTV and OPRV are gated together, by Protection Violation Gate (PVG) 24086 to generate Protection Violation (PROTV) output indicating that either an extent or an operation violation has occurred.

Having described the structure and operation of MEMINT 20212, and previously the structure and operation of DESP 20210, structure and operation of FUCTL 20214 will be described next below.

3. Fetch Unit Control Logic 20214 (Fig. 202)

The following descriptions will provide a detailed description of FU 10120's structure and operation. Overall operation of FU 10120 will be described first, followed by description of FU 10120's structure, and finally by a detailed description of FU 10120 operation.

As previously described, FUCTL 20214 directs operation of JP 10114 in executing procedures of user's processes. Among the functions performed by FUCTL 20214 are, first, maintenance and operation of CS 10110's Name Space, UID, and AON based addressing system, previously described; second, interpretation of SOPs of user's processes to provide corresponding sequences of microinstructions to FU 10120 and EU 10122 to control operation of JP 10114 in execution of user's processes, previously described; and, third, control of operation of CS 10110's internal mechanisms, for example CS 10110's stack mechanisms.

As will be described in further detail below, FUCTL 20214 includes prefetcher (PREF) 20260 which generates a sequence of logical addresses, each logical address comprising an AON and an offset field, for reading S-Instructions (SINs) of a user's program from MEM 10112. As previously described, each SIN may be comprised of an S-Operation (SOP) and one or more operand Names and may occupy one or more 32 bit words. SINs are read from MEM 10112 as a sequence of single 32 bit words, so that PREF 20260 need not specify a length field in a MEM 10112 read request for an SIN. SINs are read from MEM 10112 through MOD Bus 10144 and are captured and stored in Instruction Buffer (INSTB) 20262. PARSER 20264 extracts, or parses, SOPs and operand Names from INSTB 20262. PARSER 20264 provides operand Names to NC 10226 and SOPs to FUS Intrepreter Dispatch Table (FUSDT) 11010 and to EU Dispatch Table (EUSDT) 20266 through Op-Code Register (OPCODEREG) 20268. Operation of INSTB 20262 and PARSER 20264 is controlled by Current program Counter (CPC) 20270, Initial Program Counter (IPC) 20272, and Executed program Counter (EPC) 20274.

As previously described, FUSDT 11010 provides, for each SOP received from OPCODEREG 20268, a corresponding S-Interpreter Dispatch (SD) Pointer, or address, to FUSITT 11012 to select a corresponding sequence of microinstructions to direct operation of JP 10114, in particular FU 10120. As previously described, FUSITT 11012 also contains sequences of microinstructions for controlling and directing operation of CS 10110's internal mechanisms, for example those mechanisms such as RCWS 10358 which are involved in swapping of processes. EUSDT 20266 performs an analogous function with respect to EU 10122 and provides SD Pointers to EU S-Interpreter Tables (EUSITTs) residing in EU 10122.

Micro-Program Counter (mPC) 20276 provides sequential addresses to FUSITT 11012 to select individual microinstructions of sequences of microinstructions. Branch and Case Logic (BRCASE) 20278 provides addresses to FUSITT 11012 to select microinstructions sequences for microinstructions branches and and cases. Repeat Counter (REPCTR) 20280 and Page Number Register (PNREG) 20282 provide addresses to FUSITT 11012 during FUSITT 11012 load operations.

As previously described, FUSITT 11012 is a writable microinstruction control store which is loaded with selected S-Interpreters (SINTs) from MEM 10112.

FUSITT 11012 addresses are also provided by Event Logic (EVENT) 20284 and by JAM input from NC 10226. As will be described further below, EVENT 20284 is part of FUCTL 20214's circuitry primarily concerned with operation of CS 10110's internal mechanisms. Input JAM from NC 10226 initiates certain FUCTL 20214 control functions for CS 10110's Name Space addressing mechanisms, and in particular NC 10226. Selection between the above discussed address inputs to FUSITT 11012 is controlled by S-

Interpreter Table Next Address Generator Logic (SITTNAG) 20286.

Other portions of FUCTL 20214's circuitry are concerned with operation of CS 10110's internal mechanisms. For example, FUCTL 20214 includes Return Control Word Stack (RCWS) 10358, previously described with reference to CS 10110's Stack Mechanisms. Register Address Generator (RAG) 20288 provides pointers for addressing of GRF 10354 and RCWS 10358 and includes Micro-Stack Pointer Registers (MISPR) 10356.

As previously described, MISPR 10356 mechanism provides pointers for addressing Micro-Stack (MIS) 10368. As will be described further below, actual MIS 10368 Pointers pointing to current, previous, and bottom frames of MIS 10368 reside in Micro-Control Word Register 1 (MCW1) 20290. MCW1 20290 and Micro-Control Word Zero Register (MCWO) 20292 together contain certain information indicating the current execution environment of a microinstruction sequence currently being executed by FU 10120. This execution information is used in aide of execution of these microinstruction sequences. State Registers (STATE) 20294 capture and store certain information regarding state of operation of FU 10120. As described further below, this information, referred to as state vectors, is used to enable and direct operation of FU 10120.

Timers (TIMERS) 20296 monitor elapsed time since occurrence of the events requiring servicing by FU 10120. If waiting time for these events exceeds certain limits, TIMERS 20296 indicate that these limits have been exceeded so that service of those events may be initiated.

Finally, Fetch Unit to E Unit Interface Logic (FUEUINT) 20298 comprises the FU 10120 portion of the interface between FU 10120 an EU 10122. FUEUINT 20298 is primary path through which operation of FU 10120 and EU 10122 is coordinated.

Having described overall operation of FU 10120, structure of FU 10120 will be described next below with aide of Fig. 202, description of FU 10120's structure will be followed by a detailed description of FU 10120 wherein further, more detailed, diagrams of certain portions of FU 10120 will be introduced as required to enhance clarity of presentation.

a.a. Fetch Unit Control Logic 20214 Overall Structure

Referring again to Fig. 202, as previously described Fig. 202 includes a partial block diagram of FUCTL 20214. Following the same sequence of description as above, PREF 20260 has a 28 bit bi-directional port connected to AON Bus 20230 and 32 bit bi-directional port directed from OFFSET Bus 20228. A control input of PREF 20260 is connected from control output of INSTB 20262.

INSTB 20262 32 bit data input (DI) is connected from MOD Bus 10144. INSTB 20262's 16 bit output (DO) is connected to 16 bit bi-directional input of OPCODEREG 20268 and to 16 bit NAME Bus 20224. OPCODEREG 20268's input comprises 8 bits of SINT and 3 bits of dialect selection. As previously described, NAME Bus 20224 is connected to 16 bit bi-directional port of Name Trap (NT) 20254, to address input ADR of NC 10226, and to inputs and outputs of OFFP 20228. Control inputs of INSTB 20262 and PARSER 20264 are connected from a control output of CPC 20270.

Thirty-two bit input of CPC 20270 is connected from JPD Bus 10142 and CPC 20270's 32 bit output is connected to 32 bit input of IPC 20272. Thirty-two bit output of IPC 20272 is connected to 32 bit input of EPC 20274 and to JPD Bus 10142. EPC 20274's 32 bit output is similarly connected to JPD Bus 10142.

Eleven bit outputs of OPCODEREG 20268 are connected to 11 bit address inputs of FUSDT 11010 and EUSDT 20266. These 11 bit address inputs to FUSDT 11010 and EUSDT 20266 each comprise 3 bits of dialect selection code and 8 bits of SINT code. Twelve bit SDT outputs of EUSDT 20266 is connected to inputs of Microinstruction Control Store in EU 10122, as will be described in a following description of EU 10122. FUSDT 11010 has, as described further below, two outputs connected to address (ADR) Bus 20298. First output of FUSDT 11010 are six bit SDT pointers, or addresses, corresponding to generic SINTs as will be described further below. Second output of FUSDT 11010 are 15 bit SDT pointers, or addresses, for algorithm microinstruction sequences, again as will be described further below.

Referring to RCWS 10358, RCWS 10358 has a first bidirectional port connected from JPD Bus 10142. Second, third, and fourth bi-directional ports of RCWS 10358 are connected from, respectively, a bi-directional port of MCW1 20290, a first bi-directional port EVENT 20284, and a bi-directional port of mPC 20276. An output of RCWS 10358 is connected to ADR Bus 20298.

An input of mPC 20276 is connected from ADR Bus 20298 and first and second outputs of mPC 20276 are connected to, respectively, an input of BRCASE 20278 and to ADR Bus 20298. An output of BRCASE 20278 is connected to ADR Bus 20298.

As described above, a first bi-directional port of EVENT 20284 is connected to RCWS 10358. A second bidirectional port of EVENT 20284 is connected from MCWO 20292. An output of EVENT 20284 is connected to ADR Bus 20298.

Inputs of RPCTR 20280 and PNREG 20282 are connected from JPD Bus 10142. Outputs of RPCTR 20280 and PNREG 20282 are connected to ADR Bus 20298.

ADR Bus 20298, and an input from a first output of FUSITT 11012, are connected to inputs of SITTNAG 20286.

Output of SITTNAG 20286 is connected, through Control Store Address (CSADR) Bus 20299, to address input of FUSITT 11012. Data input of FUSITT 11012 is connected from JPD Bus 10142. Control outputs of FUSITT 11012 are connected to almost all elements of JP 10114 and thus, for clarity of presentation, are not

shown in detail by drawn physical connections but are described in following descriptions.

As described above, MCW0 20292 and MCW1 20290 have bi-directional ports connected to, respectively, bidirectional ports of EVENT 20284 and to a second bidirectional port of RCWS 10358. Outputs of MCW0 20292 and MCW1 20290 are connected to JPD Bus 10142. Other inputs of MCW0 20292 and MCW1 20290, as will be described further below, are connected from several other elements of JP 10114 and, for clarity of presentation, are not shown herein in detail but are described in the following text. STATE 20294 similarly has a large number of inputs and outputs connected from and to other elements of JP 10114, and in particular FU 10120. Inputs and outputs of STATE 20294 are not indicated here for clarity of presentation and will be described in detail below.

RAG 20288 has an input connected from JPD Bus 10142 and other inputs connected, for example, from MCW1 20290. RAG 20288, including MISPR 10356, provides outputs, for example, as address inputs to RCWS 10358 and GRF 10354. Again, for clarity of presentation, inputs and outputs of RAG 20288 are not shown in detail in Fig. 202 but will be described in detail further below.

TIMERS 20296 receive inputs from EVENT 20284 and FUSITT 11012 and provide outputs to EVENT 20284. For clarity of presentation, these indications are not shown in detail in Fig. 202 but will be described further below.

FUINT 20298 receives control inputs from FUSITT 11012 and EU 10122. FUINT 20298 provides outputs to EU 10122 and to other elements of FUCTL 20214. For clarity of presentation, connections to and from FUINT 20298 are not shown in detail in Fig. 202 but will be described in further detail below.

Having described the overall operation, and structure, of FUCTL 20214, operation of FUCTL 20214 will be described next below. During the following descriptions further diagrams of certain portions of FUCTL 20214 will be introduced as required to disclose structure and operation of FUCTL 20214 to one of ordinary skill in the art. FUCTL 20214's operation with regard to fetching and interpretation of SINs, that is SOPs and operand Names, will be described first, followed by description of FUCTL 20214's operation with regard to CS 10110's internal mechanisms.

b.b. Fetch Unit Control Logic 20214 Operations

Referring first to those elements of FUCTL 20214 directly concerned with control of JP 10114 in response to SOPs and Name syllables, those elements include: (1) PREF 20260; (2) INSTB 20262; (3) PARSER 20264; (4) CPC 20270, IPC 20272, and EPC 20274; (5) OPCODEREG 20268; (6) FUSDT 11010 and EUSDT 20266; (7) mPC 20276; (8) BRCASE 20278; (9) REPCTR 20280 and PNREG 20282; (10) a part of RCWS 10358; (11) SITTNAG 20286; (12) FUSITT 11012; and, (13) NT 20254. These FUCTL 20214 elements will be described below in the order named.

a.a.a. Prefetcher 20260, Instruction Buffer 20262, Parser 20264, Operation Code Register 20268, CPC 20270, IPC 20272, and EPC 20274 (Fig. 241)

As described above, PREF 20260 generates a series of addresses to MEM 10112 to read SINs of user's programs from MEM 10112 to FUCTL 20214, and in particular to INSTB 20262. Each PREF 20260 read request transfers one 32 bit word from MEM 10112. Each SIN may be comprised of an SOP and one or more Name syllables. Each SOP may comprise, for example, 8 bits of information while each Name syllable may comprise, for example, 8, 12, or 16 bits of data. In general, and as will be described in further detail in a following description of STATE 20294, PREF 20260 obtains access to MEM 10112 on alternate 110 nanosecond system clock cycles. PREF 20260's access to MEM 10112 is conditional upon INSTB 20262 indicating that INSTB 20262 is ready to receive an SIN read from MEM 10112. In particular, INSTB 20262 generates control output Quiry Prefetch (QPF) to PREF 20260 to enable PREF 20260 to submit a request to MEM 10112 when, as described further below, INSTB 20262 is ready to receive an SIN read from MEM 10112.

PREF 20260 is a counter register comprised, for example of SN74S163s.

Bi-directional inputs and outputs of PREF 20260 are connected to AON Bus 20230 and OFFSET Bus 20228. As PREF 20260 reads only single 32 bit words, PREF 20260 is not required to specify a LENGTH field as part of an SIN read request, that is an AON and an OFFSET field are sufficient to define a single 32 bit word. At start of read of a sequence of SINs from MEM 10112, address (AON and OFFSET fields) of first 32 bit word of that SIN sequence are provided to MEM 10112 by DESP 20210 and concurrently loaded, from AON Bus 20230 and OFFSET Bus 20228, into PREF 20260. Thereafter, as each successive thirty-two bit word of the SIN's sequence is read from MEM 10112, the address residing in PREF 20260 is incremented to specify successive 32 bit words of that SIN's sequence. The successive single word addresses are, for all words after first word of a sequence, provided to MEM 10112 from PREF 20260.

As described above, INSTB 20262 receives SINs from MEM 10112 through MOD Bus 10144 and, with PARSER 20264 and operating under control of CPC 20270, provides Name syllables to NAME Bus 20224 and SINs to OPCODEREG 20268. INSTB 20262 is provided, together with PREF 20260 to increase execution speed of SINS.

Referring to Fig. 241, a more detailed block diagram of INSTB 20262, PARSER 20264, CPC 20270, IPC 20272, EPC 20274 as shown. INSTB 20262 is shown as comprising two 32 bit registers having parallel 32 bit inputs from MOD Bus 10144. INSTB 20262 also receives two Write Clock (WC) inputs, one for each 32 bit register of INSTB 20262, from Instruction Buffer Write Control (INSTBWC) 24110. INSTB 20262's outputs

are structured as eight, eight bit Basic Syllables (BSs), indicated as BS0 to BS7. BS0, BS2, BS4, and BS6 are ORed to comprise eight bit Basic Syllable, Even (BSE) of INSTB 20262 while BS0, BS3, BS5, and BS7 are similarly ORed to comprise Basic Syllable, Odd (BSO) of INSTB 20262. BSO and BSE are provided as inputs of PARSER 20264.

PARSER 20264 receives a first control input from Current Syllable Size Register (CSSR) 24112, associated with CPC 20270. A second control input of PARSER 20264 is provided from Instruction Buffer Syllable Decode Register (IBSDECR) 24114, also associated with CPC 20270. PARSER 20264 provides an eight bit output to NAME Bus 20224 and to input of OPCODEREG 20268.

Referring to INSTBWC 24110, INSTBWC 24110 provides, as described further below, control signals pertaining to writing of SINs into INSTB 20262 from MOD Bus 10144. INSTBWC 24110 also provides control signals pertaining to operation of PREF 20260. In addition to WC outputs to INSTB 20262, INSTBWC 24110 provides control output QPF to PREF 20260, control output Instruction Buffer Hung (IBHUNG) to EVENT 20284, and control signal Instruction Buffer Wait (IBWAIT) to STATE 20294. INSTBWC 24110 also receives a control input BRANCH from BRCASE 20278 and an error input from TIMERS 20296.

Referring to CPC 20270, IPC 20272, and EPC 20274, IPC 20272 and EPC 20274 are represented in Fig. 241 as in Fig. 202. Further FUCTL 20214 circuitry is shown as associated with CPC 20270. CPC 20270 is a twenty-nine bit register receiving bits one to twety-five (CPC(1—25)) from bits one to twenty-five of JPD Bus 10142. CPC 20270 Bit 0 (CPC0) is provided from CPC0 CPC0 Select (CPC0S) 24116. Inputs of CPC0S 24116 are Bit 1 output from CPC 20270 (CPC1) and Bit 0 from JPD Bus 10142. Bits twentysix, twenty-seven, and twenty-eight of CPC 20270 (CPC(2628)) are provided from CPC Multiplexer (CPCMUX) 24118. CPCMUX 24118 also provides an input to IBSDECR 24114. Inputs of CPCMUX 24118 are bits twenty-five, twenty-six, and twenty-eight from JPD Bus 10142 and a three bit output of CPC Arithmetic and Logic Unit (CPCALU) 24120. A first input of CPCALU 24120 is connected from output bits 26, 27, and 28 of CPC 20270. Second input of CPCALU 24120 is connected from CSSR 24112. CSSR 24112's input is connected from JPD Bus 10142.

As described above, INSTB 20262 is implemented as a sixty-four bit wide register. INSTB 20262 is organized as two thirty-two bit words, referred to as Instruction Buffer Word 0 (IB0) and Instruction Buffer Word 1 (IB1), and operates as a two word, first-in-first-out buffer memory. PREF 20260 loads one of IB0 or IB1 on each memory reference by PREF 20260. Only PREF 20260 may load INSTB 20262, and INSTB 20262 may be loaded only from MOD Bus 10144. Separate clocks, respectively Instruction Buffer Write Clock 0 (IBWC0) and Instruction Buffer Write Clock 1 (IBWC1), are provided from INSTBWC 24110 to load, respectively, IBW0 and IBW1 into INSTB 20262. IBWC0 and IBWC1 are each a gated 110 nano-second clock. An IBW0 or an IBW1 is written into INSTB 20262 when, respectively, IBWC0 or IBWC1 is enabled by INSTBWC 24110. IBWC0 and IBWC1 will be enabled only when MEM 10112 indicates that data for INSTB 20262 is availabe by asserting interface control signal DAVI as previously discussed.

INSTBWC 24110 is primarily concerned with control of FU 10120 with respect to writing of SINs into INSTB 20262. As described above, INSTBWC 24110 provides IBWC0 and IBWC1 to INSTB 20262. IBWC0 and IBWC1 are enabled by INSTBWC 24110's input DAVI from MEM 10112. Selection between IBWC0 and IBWC1 is controlled by INSTBWC 24110's input from CPC 20270. In particular, and as will be described further below, Bit 26 (CPC 26) of CPC 20270's twenty-nine bit word indicates whether IBW0 or IBW1 is written into INSTB 20262.

In addition to controlling writing of IBW0 and IBW1 into INSTB 20262, INSTBWC 24110 provides control signals to elements of FU 10120 to control reading of SINs from MEM 10112 to INSTB 20262. In this regard, INSTBWC 24110 detects certain conditions regarding status of SIN words in INSTB 20262 and provides corresponding control signals, described momentarily, to other elements of FU 10120 so that INSTB 20262 would generally always contain at least one valid SOP or Name syllable. First, if INSTB 20262 is not full, that is either IBW0 or IBW1 or both is invalid, for example because IBW0 has been read from INSTB 20262 and executed, INSTBWC 24110 detects this condition and provides control signal QPF to PREF 20262 to initiate a read from MEM 10112. INSTBWC 24110 currently enables either IBW0 or IBW1 portion of INSTB 20262 to receive the word read from MEM 10112 in response to PREF 20260's request. As stated above, this operation will be initiated when INSTBWC 24110 detects and indicates, by generating a validity flag, that either IBW0 or IBW1 is invalid. In this case, IBW0 or IBW1 will be indicated as invalid when read from INSTB 20262 by PARSER 20264. As will be described further below, INSTBWC 24110 validity flags for IBW0 and IBW1 are generated by INSTBWC 24110 control inputs comprising Bits 26 to 28 (CPC 26—28) from CPC 20270 and by current syllable size or value, flag (K) input from CSSR 24112. Secondly, INSTBWC 24110 will detect when INSTB 20262 is empty, that is when both IBW0 and IBW1 are invalid, as just described, or when only a half of a sixteen bit Name syllable is present in INSTB 20262. In response to either condition, INSTBWC 24110 will generate control signal IBWAIT to STATE 20294. As will be described further below, IBWAIT will result in suspension of execution of microinstructions referencing INSTB 20262. PREF 20260 requests to MEM 10112 will already have been initiated, as described above unless certain other corditions, described momentarily, occur. Thirdly, INSTBWC 24110 will detect when INSTB 20262 is empty and PREF 20262 is hung, that is unable to submit requests to MEM 10112, and a current microinstruction is attempting to parse a syllable from INSTB 20262. In this case, INSTBWC 24110 will generate control signal Instruction Buffer Hung (IBHUNG) to EVENT 20284. As will be described further below, IBHUNG will result in initiation of a microinstruction sequence to restore flow of words to INSTB 20262. Fourthly, INSTBWC 24110 will detect, through microinstruction control signals provided from FUSITT 11012, when a branch in

a microinstruction sequence provided by FUSITT 11012 in response to an SOP occurs. In this case, both IBW0 and IBW1 will be flaged as invalid. INSTBWC 24110 will then ignore SIN words being read from MEM 10112 in response to a previously submitted PREF 20260 request, but not yet received at the time the branch occurs. This prevents INSTB 20260 from receiving invalid SIN words; PREF 20260 and INSTB 20262 will then proceed to request and receive valid SIN words of the branch.

As described above, PARSER 20264, operating under control of CPC 20270 and CPC 20270 associated circuitry, reads Name syllables and SOPs from INSTB 20262 to, respectively, NAME Bus 20224 and OPCODEREG 20268. PARSER 20264 operates as a multiplexer with associated control logic.

As previously described, INSTB 20262 is internally structured as eight, eight bit words, BS0 to BS7. IBW0 comprises BS0 to B3 while IBW1 comprises BS4 to BS7. Each SOP is comprised of eight bits of data and thus comprises one Basic Syllable while each Name syllable comprises 8, 12, or 16 bits of data and thus comprises either one or two Basic Syllables. Name syllable size, as previously stated, is indicated by Current Syllable Size Value K stored in CSSR 24112.

BS0 and BS4 are loaded into INSTB 20262 from MOD Bus 10144 bits zero to seven while BS2 and BS6 are loaded from MOD Bus 10144 bits sixteen to twenty-three. BS1 and BS5 are loaded from MOD Bus 10144 bits eight to fifteen while BS3 and BS7 are loaded from MOD Bus 10144 bits twenty-four to thirty-one. Odd numbered Basic Syllable outputs BS1, BS3, BS5, and BS7 are ORed to comprise eight bit Basic Syllable, Odd output BS0 of INSTB 20262. Even numbered Basic Syllable outputs BSo, BS2, BS4 and BS6 of INSTB 20262 are similarly ORed to comprise eight bit Basic Syllable, Even output BSE. At any time, one odd numbered Basic Syllable output and one even numbered Basic Syllable output of INSTB 20262 are selected as inputs to PARSER 20264 by Instruction Buffer Read Enable (IBORE) enable and selection signals provided to INSTB 20262 by IBSDECR 24114. IBSDECR 24114 includes decoding circuitry. Input to IBSDECR 24114's decoding logic is comprised of three bits (RCPC(26—28)) provided from CPCMUX 24118. As indicated in Fig. 241, CPC (26—28) may be provided from JPD Bus 10142 bits 25 to 28 or from output of CPCALU 24120. One input CPCALU 24120 is CPC (26—28) from CPC 20270. Operation of CPC 20270 and CPC 20270's associated circuitry will be described further below. RCPC (26—28) is decoded by IBSDECR 24114 to generate IBORE (0—7) to INSTB 20262. RCPC 26 and RCPC 27 are decoded to select one of the four odd numbered Basic Syllable outputs (that is BS1, BS3, BS5 or BS7) of INSTB 20262 as the odd numbered basic syllable input to PARSER 20264. RCPC 28 selects either the preceding or the following even numbered Basic Syllable output of INSTB 20262 as the even numbered Basic Syllable input to PARSER 20264. The eight decoded bits of IBORE (0—7) generated by IBDECR 24114 decoding logic are loaded into IBSDECR 24114 eight bit register and subsequently provided to INSTB 20262 as IBORE (0—7).

PARSER 20264 selects BS0, or BSE, or both BSO and BSE, as PARSER 20264s output to NAME Bus 20224 or to OPCODEREG 20268. In the case of an SOP or an eight bit Name syllable, either BSO or BSE will be selected as PARSER 20264's output. In the case of a twelve or sixteen bit Name syllable, both BS0 and BSE may be selected as PARSER 20264's output. PARSER 20264 operation is controlled by microinstruction control outputs from FUSITT 11012.

Program counters IPC 20272, EPC 20274, and CPC 20270 are associated with control of fetching and parsing of SINs. In general, IPC 20272, EPC 20274, and CPC 20270 operate under microinstruction control from FUSITT 11012.

CPC 20270 is Current Program Counter and contains 28 bits pointing to the current syllable in INSTB 20272. Bits 29 to 31 of CPC 20270 are not provided, so the bits 29 to 31 of CPC 20270's output are zero, which guarantees byte boundaries for SOPs. Contents of CPC 20270 are thereby also a pointer which is a byte align offset into a current procedure object. Initial Program Counter (IPC) 20272 is a buffer register connected from output of CPC 20270 and provided for timing overlap. IPC 20272 may be loaded only from CPC 20270 which, as previously described, is 29 bits wide, that does not contain bits 29, 30, and 31 which are forced to zero in IPC 20272. IPC 20272 may be read onto JPD Bus 10142 as a start value in an unconditional branch.

EPC 20274 is a thirty-two bit register usually containing a pointer to the current SOP being executed. Upon occurrence of an SOP branch, the pointer in EPC 20274 will point to the SOP from which the branch was executed. The pointer residing in EPC 20274 is an offset into a current procedure object. EPC 20274 may be loaded only from IPC 20272, and may be read onto JPD Bus 10142.

Referring again to CPC 20270, as described above CPC 20270 is a current syllable counter. CPC 20270 contains a pointer to the next SOP syllable, or Base Syllable, to be parsed by PARSER 20264. As SOPs are always on byte boundaries, CPC 20270 pointer is 29 bits wide, CPC (0—28). The three low order bits of CPC 20270's pointer, that is CPC (29—31), do not physically exist and are assumed to be always zero. CPC 20270's pointer to next instruction syllable to be parsed thereby always points to byte boundaries.

CPC 20270 bits 26 to 28, CPC (26—28), indicate, as described above, a particular Base Syllable in INSTB 20262. Bits 0—25 (CPC(0—25)) of CPC 20270 indicate 32 bit words, read into INSTB 20262 as IBW0 and IBW1, of a sequence of SINs. CPC 20270 pointer is updated each time a parse operation reading a Base Syllable from INSTB 20262 is executed. As previously described, these parsing operations are performed under microinstruction control from FUSITT 11012.

Conceptually, CPC 20270 is organized as a twenty-six bit counter, containing CPC (0—25), with a three bit register appended on the low order side, as CPC (26—28). This organization is used because CPC (26—28) counts INSTB 20262 Base Syllables parsed and must be incremented dependant upon current

Name Syllable Size K stored CSSR 24112. CPC (0—25), however, counts successive thirty-two bit words of a sequence of SINs and may thereby be implemented as a binary counter. As shown in Fig. 241, CPC (26—28) is loaded from output of CPCMUX 24118. A first input of CPCMUX 24118 is connected from bits 29 to 31 of JPD Bus 10142. This input to CPC (26—28) from JPD Bus 10142 is provided to allow CPC 20270 to be loaded from JPD Bus 10142, for example when loading CPC 20270 with an initial pointer value. Second input of CPCMUX 24118 is from output of CPCALU 24120 and is the path by which CPC (26—28) is incremented as successive Base Syllables are parsed from INSTB 20262. A first input of CPCALU 24120 is CPC (26—28) from CPC 20270. Second input of CPALU 24120 is a dual input from CSSR 24112. First input from CSSR 24112 is logic 1 in the least significant bit position, that is in position corresponding to CPC (28). This input is used when single Base Syllables are parsed from INSTB 20262, for example in an eight bit SOP or an eight bit Name syllable. CSSR 24112's first input to CPCALU 24120 increments CPC (0—32) by eight, that is one to CPC (26—28), each time a single Base Syllable is parsed from INSTB 20262. Second input to CPCALU 24120 from CSSR 24112 is K, that is current Name Syllable size. As previously described, K may be eight, twelve, or sixteen. CPC (26—28) is thereby incremented by one when K equals eight and is incremented by two when K equals twelve or sixteen. As shown in Fig. 241, K is loaded into CSSR 24112 from JPD Bus 10142.

CPC (0—25), as described above, operates as a twenty-six bit counter which is incremented each time CPC (26—28) overflows. CPC (0—25) is incremented by carry output of CPCALU 24120. In actual implementation, CPC 20270 is organized to reduce the number of integrated circuits required. CPC (1—25) is constructed as a counter and inputs of CPC (1—25) counter are connected from bits 1 to 24 of JPD Bus 10142 to allow loading of an initial value of CPC 20270 pointer. CPC (0) and CPC (26—28) are implemented as a four bit register. Operation of CPC (26—28) portions of this register have been described above. Input of CPC (0) portion of this register is connected from output of CPCOS 24116. CPCOS 24116 is a multiplexer having a first input connected from bit 0 of JPD Bus 10142. This input from JPD Bus 10142 is used, for example, when loading CPC 20272 with an initial pointer value. Second input of CPCOS 24116 is overflow output of CPC (1—25) counter and allows CPC (0) portion of the four bit register and CPC (1—25) counter to operate as a twenty-six bit counter.

Finally, as shown in Fig 241, output of CPC 20270 may be loaded into IPC 20272. An initial CPC 20270 pointer value may therefore be written into CPC 20270 from JPD Bus 10142 and subsequently copied into IPC 20272.

Referring again to PARSER 20264, as described above PARSER 20264 reads, or parses, basic syllables from INSTB 20262 to NAME Bus 20224. Input of PARSER 20264 is a sixteen bit word comprised of an eight bit odd numbered Base Syllable, BSO, and an eight bit even numbered Base Syllable, BSE. Depending upon whether PARSER 20264 is parsing an eight bit SOP, an eight bit Name syllable, a twelve bit Name syllable, or sixteen bit Name syllable, PARSER 20264 may select BSO, BSE, or both BSO and BSE, as output onto NAME Bus 20224.

If PARSER 20264 is parsing Name syllables and K is not equal to eight, that is equal to twelve or sixteen, PARSER 20264 transfers both BSO and BSE onto NAME Bus 20224 and determines which of BSO or BSE is most significant. The decision as to whether BSO or BSE is most significant is determined by CPC (28). If CPC (28) indicates BSO is most significant, BSO is transferred onto NAME Bus 20224 bits 0 to 7 (NAME(0—7)) and BSE onto NAME Bus 20224 bits eight to fifteen (NAME(8—15)). If CPC (28) indicates BSE is most significant, BSE is transferred onto NAME (0—7) and BSO onto NAME (8—15). This operation insures that Name syllables are parsed onto NAME Bus 20224 in the order in which occur in the SIN stream.

If PARSER 20264 is parsing Name syllables of Syllable Size K = 8, PARSER 20264 will select either BSO or BSE, as indicated by CPC (28), as output to NAME (0—7). PARSER 20264 will place 0's on NAME (8—15).

If PARSER 20264 is parsing SOPs of eight bits, PARSER 20264 will select BSO or BSE as output to NAME (0—7) as selected by CPC (28). PARSER 20264 will place 0's onto NAME (8—15). Concurrently, PARSER 20264 will generate OPREGE to OPCODEREG 20268 to enable transfer of NAME (0—7) into OPCODEREG 20268. OPCODEREG 20268 is not loaded when PARSER 20264 is parsing Name syllables. The microinstruction input from FUSITT 11012 which controls PARSER 20264 operation also determines whether PARSER 20264 is parsing an SOP or a Name syllable and controls generation of OPREGE.

Operation of NC 10226, which receives Name syllables as address inputs from NAME Bus 20224, has been discussed previously with reference to MEMINT 20212. Name Trap (NT) 20254 is connected from NAME Bus 20224 to receive and capture Name syllables parsed onto NAME Bus 20224 by PARSER 20264. Operation of NT 20254 has been also previously discussed with reference to MEMINT.

b.b.b. Fetch Unit Dispatch Table 11010, Execute Unit Dispatch Table 20266 and Operation Code Register 20268 (Fig. 242)

As previously described, CS 10110 is a multiple language machine. Each program written in a high level user language is compiled into a corresponding S-Language program containing S-Language Instructions referred to as SOPs. CS 10110 provides a set or dialect, of microcode instructions, referred to as S-Interpreters (SINTs) for each S-Language. SINTs interpret SOPs to provide corresponding sequences of microinstructions for detailed control of CS 10110 operations. CS 10110's SINTs for FU 10120 and EU 10122 operations are stored, respectively, in FUSITT 11012 and in a corresponding control store memory in EU 10122, described in a following description of EU 10122. Each SINT comprises one or more sequences

of microinstructions, each sequence of microinstructions corresponding to a particular SOP in a particular S-Language dialect. Fetch Unit S-Interpreter Dispatch Table (FUSDT) 11010 and Execute Unit S-Interpreter Dispatch Table (EUSDT) 20266 contain an S-Interpreter Dispatcher (SD) for each S-Language dialect. Each SD is comprised of a set of SD Pointers (SDPs) wherein each SDP in a particular SD corresponds to a particular SOP of that SD dialect. Each SDP is an address pointing to a location, in FUSITT 11012 or EUSITT, of the start of the corresponding sequence of microinstructions for interpreting the SOP corresponding to that SDP. As will be described further below, SOPs received and stored in OPCODEREG 20268 are used to generate addresses into FUSDT 11010 and EUSDT 20266 to select corresponding SDPs. Those SDPs are then provided to FUSITT 11012 through ADR 20202, or to EUSITT through EUDIS Bus 20206, to select corresponding sequences of microinstructions from FUSITT 11012 and EUSITT.

Referring to Fig. 242, a more detailed block diagram of OPCODEREG 20268, FUSDT 11010, and EUSDT 20266 is shown. As shown therein, OPCODEREG 20268 is comprised of OP-Code Latch (LOPCODE) 24210, Dialect Register (RDIAL) 24212, Load Address Register (LADDR) 24214, and Fetch Unit Dispatch Encoder (FUDISENC) 24216. Data inputs of LOPCODE 24010 are connected from NAME Bus 20224 to receive SOPs parsed from INSTB 20262. Load inputs of RDIAL 24212 are connected from Bits 28 to 31 of JPD Bus 10142. Outputs of LOPCODE 24210, RDIAL 24212 and LADDR 24214 are connected to inputs of FUDISENC 24216. Outputs of FUDISENC 24216 are connected to address inputs of FUSDT 11010 and EUSDT 20266.

FUSDT 11010 is comprised of Fetch Unit Dispatch File (FUDISF) 24218 and Algorithm File (AF) 24220. Address inputs of FUDISF 24218 and AF 24220 are connected, as previously described, from address outputs of FUDISENC 24216. Data load inputs of FUDISF 24218 and AF 24220 are connected from, respectively, Bits 10 to 15 and Bits 16 to 31 of JPD Bus 10142. SDP outputs of FUDISF 24218 and AF 24220 are connected to ADR Buses 20202.

EUSDT 20266 is comprised of Execute Unit Dispatch File (EUDISF) 24222 and Execute Unit Dispatch Selector (EUDISS) 24224. Address inputs of EUDISF 24222 are, as described above, connected from outputs of FUDISENC 24216. Data load inputs of EUDISF 24222 are connected from Bits 20 to 31 of JPD Bus 10142. Inputs of EUDISS 24224 are connected from SDP output of EUDISF 24222, from Bits 20 to 31 of JPD Bus 10142, and from Microcode Literal (mLIT) output of FUSITT 11012. SDP outputs of EUDISS 24224 are connected to EUDIS Bus 20206.

As previously described, OPCODEREG 20268 provides addresses, generated from SOPs loaded into OPCODEREG 20268, to FUSDT 11010 and EUSDT 20266 to select SDPs to be provided as address inputs to FUSITT 11012 and EUSITT. LOPCODE 24210 receives and stores eight bit SOPs parsed from INSTB 20262 as described above. OPCODEREG 20268 also provides addresses to FUSDT 11010 and EUSDT 20266 to load FUSDT 11010 and EUSDT 20266 with SDs for S-Language dialects currently being utilized by CS 10110. LOPCODE 24210 and RDIAL 24212, as described below, provide addresses to FUSDT 11010 and EUSDT 20266 when translating SOPs to SDPs and ADDR 24214 provides addresses when FUSDT 11010 and EUSDT 20266 are being loaded with SDs.

Referring first to LADDR 24214, LADDR 24214 has an eight bit counter. Addresses are provided to FUSDT 11010 and EUSDT 20266 from LADDR 24214 only when FUSDT 11010 and EUSDT 20266 are being loaded with SDs, that is groups of SDPs for S-Language dialects currently being utilized by CS 10110. During this operation, output of LADDR 24214 is enabled to FUSDT 11010 and EUSDT 20266 by microcode control signals (not shown for clarity of presentation) from FUSITT 11012. Selection between FUDISF 24218, AF 24220, and EUDISF 24222 to receive addresses is similarly provided by microinstruction enable signals (also not shown for clarity of presentation) provided from FUSITT 11012. These FUSDT 11010 and EUSDT 20266 address enable inputs may select, at any time, any or all of FUDISF 24218, AF 24220, or EUDSF 24222 to receive address inputs. SDPs to be loaded into FUDISF 24218, AF 24220, and EUDISF 24222 are provided, respectively, from Bits 10 to 15 (JPD(10—15)), Bits 16 to 31 (JPD(16—31)), and Bits 20 to 31 (JPD(20—31)) of JPD Bus 10142. Address contents of LADDR 24214 are successively incremented by one as successive SDPs are loaded into FUSDT 11010 and EUSDT 20266. Incrementing of LADDR 24214 is, again, controlled by microinstruction control inputs from FUSITT 11012.

Address inputs to FUSDT 11010 and EUSDT 20266 during interpretation of SOPs are provided from LOPCODE 24210 and RDIAL 24212. LOPCODE 24210 is a register counter having, as described above, data inputs connected from NAME Bus 20224 to receive SOPs from PARSER 20264. In a first mode, LOPCODE 24210 may operate as a latch, loaded with one SOP at a time from output of PARSER 20264. In a second mode, LOPCODE 24210 operates as a clock register to receive successive eight bit inputs from low order eight bits of NAME Bus 20224 (NAME(8—15)). Loading of LOPCODE 24210 is contolled by microinstruction control outputs (not shown for clarity of presentation) from FUSITT 11012.

As will be described further below, eight bit SOPs stored in LOPCODE 24210 are concatenated with the output of RDIAL 24212 to provide addresses to FUSDT 11010 and EUSDT 20266 to select SDPs corresponding to particular SOPs. That portion of these addresses provided from LOPCODE 24210, that is the eight bit SOPs, selects particular SDPs within a particular SD. Particular SDs are selected by that portion of these addresses which is provided from the contents of RDIAL 24212.

RDIAL 24212 receives and stores four bit Dialect Codes indicating the particular S-Language dialect currently being used by CS 10110 and executing the SOPs of a user's program. These four bit Dialect Codes are provided from JPD Bus 10142, as JPD (28—31). Loading of RDIAL 24212 with four bit Dialect Codes is controlled by microinstruction control signals provided form FUSITT 11012 (not shown for clarity of

presentation).

Four bit Dialect Codes in RDIAL 24212 define partitions in FUDISF 24218, AF 24220 and EUDISF 24222. Each partition contains SDPs for a different S-Language dialect, that is contains a different SD. FUDISF 24218, AF 24220 and EUDISF 24222 may contain, for example, eight 128 word partitions or four 256 word partitions. A single bit of Dialect Code, for example Bit 3, defines whether FUDISF 24218, AF 24220, and EUDISF 24222 contain four or eight partitions. If FUSDT 11010 and EUSDT 20266 contain four partitions, the two most significant bits of address into FUSDT 11010 and EUSDT 20266 are provided from Dilect Code Bits 1 and 2 and determine which partition is addressed. The lower order eight bits of address are provided from LOPCODE 24210 and determine which word in a selected partition is addressed. If FUSDT 11010 and EUSDT 20266 contain eight partitions, the three most significant bits of address into FUSDT 11010 and EUSDT 20266 are provided from Bits 0 to 2 of Dialect Code, to select a particular partition, and the lower seven bits of address are provided from LOPCODE 24210 to select a particular word in the selected partition.

As described above, LOPCODE 24210 eight bit output and RDIAL 24212's four bit output are concatenated together, through FUDISENC 24216, to provide a ten bit address input to FUSDT 11010 and EUSDT 20266. FUDISENC 24216 is an encoding circuit and will be described further below with reference to FUDISF 24218. As previously described, selection of FUDISF 24218, AF 24220, and EUDISF 24222 to receive address inputs from RDIAL 24212 and LOPCODE 24210 is controlled by microinstruction control enable inputs provided from FUSITT 11012 (not shown for clarity of presentation).

Referring to FUSDT 11010, both FUDISF 24218 and AF 24220 provide SDPs to FUSITT 11012, but do so for differing purposes. In general, microinstruction control operations may be regarded as falling into two classes. First, there are those microinstruction operations which are generic, that is general in nature and used by or applying to a broad variety of SOPs of a particular dialect or even of many dialects. An example of this class of microinstruction operation is fetches of operand values. FUDISF 24218 provides SDPs for this class of microinstruction operations. As described below, FUDISF 24218 is a fast access memory allowing a single microinstruction control output of FUSITT 11012 to parse an SOP from INSTB 20262 into LOPCODE 24210, and a corresponding SDP to be provided from FUDISF 24218. That is, an SOP of this generic class may be parsed from INSTB 20262 and a corresponding SDP provided from FUDISF 24218 during a single system clock cycle. Operation of FUDISF 24218 thereby enhances speed of operation of JP 10114, in particular at the beginning of execution of new SOPs.

The second class of microinstruction operations are those specific to particular SINTs or to particular groups of SINTs. These groups of SINTs may reside entirely within a particular dialect, for example FORTRAN, or may exist within one or more dialects. SDPs for this class of microinstruction operation are provided by AF 24220. As described further below, AF 24220 is slower than FUDISF 24218, but is larger. In general, AF 24220 contains SDPs of microinstruction sequences specific to particular SINTs. In general, generic microinstruction operations are performed before those operations specific to particular SINTs, so that SDPs are required from AF 24220 at a later time than those from FUDISF 24218. SDPs for specific SINT operations may therefore be provided from lower speed AF 24220 without a penalty in speed of execution of SOPs.

Referring again to FUDISF 24218, FUDISF 24218 is a 1,024 word by 6 bit fast access by polar memory. Each word contained therein, as described above, is an SDP, or address to start of a corresponding sequence of microinstructions in FUSITT 11012. As will be described further below, FUSITT is an 8K (8192) word memory. SDPs provided by FUDISF 24218 are each, as described above, 6 bits wide and may thus address a limited, 32 word area of FUSITT 11012's address space. FUDISF 24218 is enabled to provide SDPs to FUSITT 11012 by microinstruction control signals (not shown for clarity of presentation) from FUSITT 11012. FUDISF 24218 six bit SDPs are encoded by FUDISENC 24219 to address FUSITT 11012 address space in increments of 4 microinstructions, that is in increments of 4 address locations. FUDISF 24218 SDPs thereby address 4 microinstructions at a time from FUSITT 11012's microinstruction sequences. As will be described further below, mPC 20276 generates successive microinstruction addresses to FUSITT 11012 to select successive microinstructions of a sequence following an initial microinstruction selected by an SDP from FUSDT 11010. An FUDISF 24218 SDP will thereby select the first microinstruction of a 4 microinstruction block, and mPC 20276 will select the following 3 microinstructions of that 4 microinstruction sequence. A 4 microinstruction sequence may therefore be executed in line, or sequentially, for each FUDISF 24218 SDP provided in response to a generic SOP. FUDISENC 24219 encodes FUDISF 24218 six bit SDPs to select these 4 microinstruction sequences so that the least signifficant bit of these SDPs occupies the 24 bit of FUSITT 11012 address inputs, and so on. The two least significant bits of an FUSITT 11012 address, or SDP, provided from FUDISF 24218 are forced to 0 while the ninth and higher bits may be hard-wired to define any particular block of 128 addresses in FUSITT 11012. This hard-wiring of the most significant bits of FUSITT 11012 addresses from FUDISF 24218 allows a set of generic microinstruction sequences selected by FUDISF 24218 to be located as desired within FUSITT 11012's address space. FUDISENC 24219 is comprised of a set of driver gates.

As previously described, SDPs for generic microinstructions currently being utilized by CS 10110 in executing user's programs are written into FUDISF 24218 from Bits 10 to 15 of JPD Bus 10142 (JPD(10—15)). Addresses for loading SDPs into FUDISF 24218 are provided, as previously described, from LADDR 24214. LADDR 24214 is enabled to provide load addresses, and FUDISF 24218 is enabled to be

written into, by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

Referring to AF 24220, as previously described AF 24220 is of larger capacity than FUDISF 24218, but has slower access time. AF 24220 is a 1,024 word by 15 bit memory. In general, 2 clock cycles are required to obtain a DSP from AF 24220. During first clock cycle, an SOP is loaded into LOPCODE 24210 and, during second clock cycle, AF 24220 is addressed to provide a corresponding SDP. SDPs provided by AF 24220 are each 15 bits in width and thus capable of addressing a larger address space than that of FUSITT 11012. As previously described, FUSITT 11012 is an 8K word memory. If FUSITT 11012 is addressed by an AF 24220 SDP referring to an address location outside of FUSITT 11012's address space, FUSITT 11012 will generate a microinstruction Not In Control Store output to EVENT 20284 as described further below. An AF 24220 SDP resulting in this event will then be used to address certain microinstruction sequences stored in MEM 10112. These microinstructions will then be executed from MEM 10112, rather than from FUSDT 11010. This operation allows certain microinstruction sequences, for example rarely used microinstruction sequences, to remain in MEM 10112, thus freeing AF 24220 and FUSITT 11012's address spaces from more frequently used SOPs.

As previously described AF 24220 is loaded, with SDPs, for SINTs currently being used by CS 10110 in executing user's programs, from Bits 16—31 of JPD Bus 10142 (JPD(16—31)). Also as previously discussed, addresses to load SDPs into AF 24220 are provided from LADDR 24214. LADDR 24214 is enabled to provide load addresses and AF 24220 to receive SDPs, by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

Referring finally to EUSDT 20266, SDPs may be provided to EU 10122 from 3 sources. EU 10122 SDPs may be provided from EUDISF 24222, from JPD Bus 10142 or from literal fields of microinstructions provided from FUSITT 11012. EUDISF 24222's SDPs are each 12 bits in width and comprise 9 bits of address into EUSITT and 3 bits of operand format information.

EUDISF 24222 is 1,024 word by 12 bit memory. As previously described addresses to read SDPs from EUDISF 24222 are provided from OPCODEREG 20268 by concatenating a 4 bit Dialect Code from RDIAL 24212 and an 8 bit SOP from LOPCODE 24210. SDPs provided by EUDISF 24222 are provided as a first input to EUDISS 24224.

EUDISS 24224 is a multiplexer. As just described, a first input of EUDISS 24224 are SDPs from EUDISF 24222. A second 12 bit input of EUDISS 24224 is provided from Bits 20 to 31 of JPD Bus 10142 (JPD(20—31)). A third input of EUDISS 24224 is a 12 bit input provided from a literal field of an FUSITT 11012 microinstruction output. EUDISS 20224 selects one of these 3 inputs to be transferred on EUDIS Bus 20206 to be provided as an execute unit SDP to EUSITT. Selection between EUDISS 20224's inputs is provided by microinstruction control signals (not shown for clarity of presentation) provided from FUSITT 11012.

As previously described, EUDISF 24222 is loaded, with SDPs for S-Language dialects currently being used by CS 10110, from Bits 20 to 31 of JPD Bus 10142 (JPD(20—31)). Addresses to load SDPs into EUDISF 24222 are provided, as previously described, from LADDR 20214. FUSITT 11012 provides enable signals (not shown for clarity of presentation) to LADDR 24214 and EUDISF 24222 to enable writing of SDPs into EUDISF 24222.

The structure and operation of FUCTL 20214 circuitry for fetching and parsing SINs from MEM 10112 to provide Name syllables and SOPs, and for interpreting SOP to provide SDPs to FUSITT 11012 and EUSITT from FUSDT 11010 and EUSDT 20266, have been described above. As described above, SDPs provided by FUSDT 11010 and EUSDT 20266 are initial, or starting, addresses pointing to first microinstructions of sequences of microinstructions. Addresses for microinstructions following those initial microinstructions are provided by FUCTL 20214's next address generator circuitry which may include mPC 20276, BRCASE 20278, REPCTR 20280 and PNREG 20282, EVENT 20284 and SITTNAG 20286. mPC 20276, BRCASE 20278, REPCTR 20280 and PNREG 20282, and SITTNAG 20286 are primarily concerned with generation of next addresses during execution of microinstruction sequences in response to SOPs and will be described next below. EVENT 20284 and other portions of FUCTL 20214's circuitry are more concerned with generation of microinstruction sequences with regard to CS 10110's internal mechanisms operations and will be described in a later description. EU 10122 also includes next address generation circuitry and this circuitry will be described in a following description of EU 10122.

c.c.c. Next Address Generator 24310 (Fig. 243)

As stated above, in FU 10120 first, or initial, microinstuctions of microinstruction sequences for interpreting SOPs are provided by FUSDT 11010. Subsequent addresses of microinstructions within these sequences are, in general, provided by mPC 20276 and BRCASE 20278. mPC 20276, as described further below, provides sequential addresses for selecting sequential microinstructions of microinstruction sequences. BRCASE 20278 provides addresses for selecting microinstructions when a microinstruction Branch or microinstruction Case operation is required. REPCTR 20280 and PNREG 20282 provide addresses for writing, or loading, of microinstruction sequences into FUSITT 11012. Other portions of FUCTL 20214 circuitry, for example EVENT 20284, provides microinstruction sequence selection addresses to select microinstruction sequences for controlling operation of CS 10110's internal mechanisms. SITTNAS 20286 selects between these microinstruction address sources to provide to FUSITT 11012 those addresses

required to select microinstructions of the operation to be currently executed by CS 10110.

Referring to Fig. 243, a partial block diagram of FU 10120's Next Address Generator (NAG) 24310 is shown. In addition to FUSDT 11010, NAG 24310 includes mPC 20276, BRCASE 20278, EVENT 20284, REPCTR 20280 and PNREG 20282, a part of RCWS 10358, and SITTNAS 20286. EVENT 20284 is, as described above, primarily concerned with execution of microinstruction sequences for controlling CS 10110 internal mechanisms. EVENT 20284 as shown herein only to illustrate its relationships to other portions of NAG 24310. EVENT 20284 will be described further in a following description of FUCTL 20214's circuitry' controlling CS 10110's internal mechanisms. Similarly, operation of RCWS 10358 will be described in part in the present description of NAG 24310, and in part in a following description of control of CS 10110's internal mechanisms.

Referring first to NAG 24310's structure, interconnections of FUSDT 11010, RCWS 10358, mPC 20276, BRCASE 20278, REPCTR 20280, PNREG 20282, EVENT 20284, and SITTNAS 20286 have been previously described with reference to Fig. 202. NAG 24310's structure will be described below only wherein Fig. 243 differs from Fig. 202.

Referring first to SITTNAS 20286, SITTNAS 20286 is shown as comprised of EVENT Gate (EVNTGT) 24310 and Next Address Select Multiplexer (NASMUX) 24312. NASMUX 24312 is comprised of NAS Multiplexer A (NASMUXA) 24314, NASMUXB 24316, NASMUXC 24318, and NASMUXD 24320. Outputs of EVNTGT 24310 and NASMUXA 24314 to NASMUXD 24320 are ORed to CSADR 20204 to provide microinstruction selection addresses to FUSITT 11012.

ADR 20202 is shown in Fig. 243 as comprised of nine buses, Address A (ADRA) Bus 24322 to Address I (ADRI) Bus 24338. Output of EVENT 20284 is connected to input of EVNTGT 24310 by ADRA Bus 24322. Outputs of REPCTR 20280 and PNREG 20282 and output of AF 24220 are connected to inputs of NASMUXA 24314 by, respectively, ADRB Bus 24324 and ADRC Bus 24326. Outputs of RCWS 10358 and FUDISENC 24219 are connected to inputs of NASMUXB 24316 by, respectively, ADRD Bus 24328 and ADRE Bus 24330. Outputs of BRCASE 20278 and second output of mPC 20276 are connected to inputs of NASMUXC 24318 by, respectively, ADRF Bus 24332 and ADRG Bus 24334. Second output of mPC 20276 and JAM output of NC 10226 are connected to inputs of NASMUXD 24320 by, respectively, ADRH Bus 24336 and ADRI Bus 24338. ADR 20202 thus comprises a set of buses connecting microinstruction address sources to inputs of SITTNAS 20286.

Referring to mPC 20276, mPC 20276 is comprised of Micro-Program Counter Counter (mPCC) 24340 and Micro-Program Counter Arithmetic and Logic Unit (mPCALU) 24342. Data input of mPCC 24340 is connected from CSADR Bus 20204. Output of mPCC 24340 is connected to a first input of mPCALU 24342 and is mPC 20276's third output to BRCASE 20278. Second input of mPCALU 24342 is a fifteen binary number set, for example by hard-wiring, to be binary one. Output of mPCALU 24342 comprises mPC 20276's first output, to RCWS 10358, and mPC 20276's second output, to inputs of NASMUXC 24318 and NASMUXD 24320.

BRCASE 20278 is shown in Fig. 243 as comprising Mask and Shift Multiplexer (MSMUX) 24344, Case Mask and Shift Logic (CASEMS) 24346, Branch and Case Multiplexer (BCMUX) 24348 and Branch and Case Arithmetic and Logic Unit (BCALU) 24350. A first input of MSMUX 24344 (AONBC, not previously shown) is connected from output of AONGRF 20232. A second input of MSMUX 24344 (OFFMUXR, not previously shown) is connected from output of OFFMUXR 23812. Output of MSMUX 24344 is connected to input CASEMS 24346, and output of CASEMS 24346 is connected to a first input of BCMUX 24348. A second input of BCMUX 24348, BLIT is connected from a literal field output of FUSITT 11012's microinstruction output. Output of BCMUX 24348 and third output of mPC 20276, from output of mPCC 24340, are connected, respectively, to first and second inputs of BCALU 24350. Output of BCALU 24350 comprises BRCASE 20278 outputs to NASMUXC 24318.

An address to select a next microinstruction may be provided to FUSITT 11012 by SITTNAS 20286 from any of eight sources. First source is output of mPC 20276. Output of mPC 20276 is referred to as Micro-Program Count Plus 1 (mPC+1) and is fifteen bits of address. Second source is from EVENT 20284 and is comprised of five bits of address. Third source is output of FUDISP 24218 and FUDISENC 24219 and, as previously described, is comprised of six bits of address. Fourth source is output of AF 24220 and, as previously described, is comprised of fifteen bits of address. Fifth source is output of BRCASE 20278. Output of BRCASE 20278 is referred to as Branch and Case Address (BRCASEADR) and comprises fifteen bits of address. Sixth source is an output of RCWS 10358. Output of RCWS 10358 is referred to as RCWS Address (RCWSADR) and is comprised of fifteen bits of address. Seventh source is REPCTR 20280 and PNREG 20282 whose outputs (REPPN) together comprise fifteen bits of address. Finally, eighth source is JAM input from NC 10226, which comprises five bits of address. These address sources differ in number of bits of address that they provide, but a microinstruction address gated onto CSADR Bus 20202 by SITTNAS 20286 always comprises fifteen bits of address. If a particular source applies fewer than fifteen bits, that address is extended to fifteen bits by SITTNAS 20286. In general, extension of address bits may be performed by hard-wiring of additional address input bits to SITTNAS 20286 from each of these sources and will be described further below.

Referring to mPC 20276, mPCC 24340 is a fifteen bit register and mPCALU 24342 is a fifteen bit ALU. mPCC 24340 is, as described above, connected from CSADR Bus 20204 and is sequentially loaded with a microinstruction address currently being presented to FUSITT 11012. mPCC 24340 will thus contain the

address of the currently executing microinstruction. mPCALU 24342 is dedicated to incrementing the address contained in mPCC 24340 by one. mPC+1 output of mPCALU 24342 will thereby always be address of next sequential microinstruction. mPC+1 is, as described above, a fifteen bit address and is thus not extended in SITTNAS 20286.

Referring to BRCASE 20278, as described above BRCASE 20278 provides next microinstruction addresses for mPC 20276 Relative Branches and for Case Branches. Next microinstruction addresses for microprogram Relative Branches and for Case Branches are both generated as addresses relative to address of currently executing microinstruction as stored in mPCC 24340, but differ in the manner in which these relative addresses are generated. Considering first Case Branches, Case Branch addresses relative to a currently executing microinstruction address are generated, in part, by MSMUX 24344 and CASEMS 24346. As described above, MSMUX 24344 which is a multiplexer receives two inputs. First input is AONBC from output of AONGRF 20232 and second input is OFFMUXR from output of OFFMUXR 23812. Each of these inputs is eight bits, or one byte, in width. Acting under control of microinstruction output from FUSITT 11012, MSMUX 24344 selects either input AONBC or input OFFMUXR as an eight bit output to input of CASEMS 24346. CASEMS 24346 is a Mask and Shift circuit, similar in structure and operation to that of FIU 20116 but operating upon bytes rather than thirty-two bit words. CASEMS 24346, operating under microinstruction control from FUSITT 11012, manipulates eight bit input from MSMUX 24344 by masking and shifting to provide eight bit Case Value (CASEVAL) output to BCMUX 24348. CASEVAL represents a microinstruction address displacement relative to address of a currently executing microinstruction and, being an eight bit number, may express a displacement of 0 to 255 address locations in FUSITT 11012.

BCMUX 24348 is an eight bit multiplexer, similar in structure and operation to MSMUX 24344, and is controlled by microinstruction inputs provided from FUSITT 11012. In executing a case operation, BCMUX 24348 selects CASEVAL input to MCMUX 24348's output to first input of BCALU 24350. BCALU 24350 is a sixteen bit arithmetic and logic unit. Second input of BCALU 24350 is fifteen bit address of currently executing microinstruction from mPCC 24340. BCALU 24350 operates under microinstruction control provided from FUSITT 11012 and, in executing a Case operation, adds CASEVAL to the address of a currently executing microinstruction. During a Case operation, carry input of BSALU 24350 is forced, by microinstruction control from FUSITT 11012, to one so that BCALU 24350's second input is effectively mPC+1, or address of currently executing microinstruction plus 1. Output BRCASEADR of BCALU 24350 will thereby be fifteen bit Case address which is between one and 256 FUSITT 11012 address locations higher than the address location of the currently executing microinstruction. The actual case value address displacement from the address of the currently executing microinstruction is determined by either input AONBC or input OFFMUXR to MSMUX 24344, and these mask and shift operations are performed by CASEMS 24346.

Case operations as described above may be used, for example, in interpreting and manipulating CS 10110 table entries. For example, Name Table Entries of Name Tables 10350 contain flag fields carrying information regarding certain operations to be performed in resolving and evaluating those Name Table Entries. These operations may be implemented as Case Branches in microinstruction sequences for resolving and evaluating those Name Table Entries. In the present example, during resolve of a Name Table Entry the microinstruction sequence for performing that resolve may direct a byte of that Name Table Entry's flag field to be read from AONGRF 20232, or OFFMUXR 23812, and through MSMUX 24344 to CASEMS 24346. That microinstruction sequence will then direct CASEMS 24346 to shift and mask that flag field byte to provide a CASEVAL. That CASEVAL will have a value dependent upon the flags within that flag field byte and, when added to mPC+1, will provide a FUSITT 11012 microinstruction address for a microinstruction sequence for handling that Name Table Entry in accordance with those flag bits.

As described above, BRCASE 20278 may also generate microinstruction addresses for Branches occurring within execution of a given microinstruction sequence. In this case, microinstruction control signals from FUSITT 11012 direct BCMUX 24348 to select BCMUX 24348's second input as output to BCALU 24350. BCMUX 24348's second input is Branch Literal (BLIT). As described above, BLIT is provided from a literal field of a microinstruction word from FUSITT 11012's microinstruction output. BLIT output of BCMUX 24348 is added to address of currently executing microinstruction from mPCC 24340, and BCALU 24350, to provide fifteen bit BRCASEADR of a microinstruction address branched to from the address of the currently executing microinstruction. BRCASEADR may represent, for example, any of four Branch Operations. Possible Branch Operations are: first, a Conditional Short Branch; second, a Conditional Short Call; third, a Long Go To; and, fourth, a Long Call. In each of these possible Branch Operations, BLIT is treated as the twos complement of the desired branch value, that is the microinstruction address offset relative to the address of the currently executing microinstruction. BLIT field may therefore be, effectively, added to or subtracted from the address of the currently executing microinstruction, to provide a microinstruction address having a positive or negative displacement from the address of the currently executing microinstruction. In a Conditional Short Branch or a Conditional Short Call, the fourteen bit literal field is a sign extended eight bit number. Both Conditional Short Branch and Conditional Short Call microinstruction addresses may therefore point to an address within a range of +127 to −128 FUSITT 11012 address locations of the address of the currently executing microinstruction. In the case of a Long Go To or Long Call, the BLIT field is a fourteen bit number representing displacement relative to the address of the currently executing microinstruction. BRCASEADR may, in these cases, represent a FUSITT 11012

microinstruction address within a range of +8191 to −8192 FUSITT 11012 address locations of the address of the currently executing microinstruction. BRCASE 20278 thereby provides FU 10120 with capability of executing a full range of microinstruction sequence Case and Branch operations.

Referring to RCWS 10358, as previously described RCWS 10358 stores information regarding microinstruction sequences whose execution has been halted. RCWS 10358 allows execution of those microinstruction sequences to be resumed at a later time. A return control word (RCW) may be written onto RCWS 10358 during any microinstruction sequence that issues a Call to another microinstruction sequence. The calling microinstruction sequence may, for example, be aborted to service an event, as described further in a following description, or may result in a Jam. A Jam is a call for a microinstruction sequence which is forced by operation of CS 10110 hardware, rather than by a microinstruction sequence. RCWS 10358 operation with regard to CS 10110's internal mechanisms will be described in a following description of EVENT 20284, STATE 20294, and MCW1 20290 and MCWO 20292. For purposes of the present discussion, that portion of a RCW concerned with interpretation of SOPs contains, first, certain state information from FUSITT 11012 and, second, a return address into FUSITT 11012. State that FUSITT 11012 state is provided from STATE 20294, as described below, and that portion of a RCW containing FUSITT 11012 state information will be described in a following description. Microinstruction address portions of RCWs are provided from output of mPCALU 24342. This microinstruction address is the address of the microinstruction to which FU 10120 is to return upon return from a Call, Event, or Jam. Upon occurrence of a Call or Jam, the microinstruction return address is mPC+1, that is the address of the microinstruction after the microinstruction issuing the Call or Return. For aborted microinstruction sequences, the microinstruction return address is mPC, that is the address of the microinstruction executing at the time abort occurs.

Upon return from a call, service of an event, or service of a jam, FU 10120 state flag portion of RCW is loaded into STATE 20294. Microinstruction return address is provided by RCWS 10358 as fifteen bit RCWSADR to SITTNAS 20286 and is gated onto CSADR 20204. RCWSADR is provided to FUSITT 11012 to select the next microinstruction and is loaded into mPCC 24340 from CSADR 20204.

As previously described, RCWS 10358 is connected to JPD Bus 10142 by a bi-directional bus. RCWs may be written into RCWS 10358 from JPD Bus 10142, or read from RCWS 10358 to JPD Bus 10142. The fifteen bit next microinstruction address portion, and the single bit FUSITT 11012 state portion of RCW is written from or read to Bits 16 to 31 of JPD Bus 10142. FU 10120 may write Present Bottom RCW or Previous RCW into RCWS 10358 from JPD Bus 10142 and may read Present Bottom RCW, or Previous RCW, or another selected RCW, onto JPD Bus 10142. RCWS 10358 thereby provides a means for storing and returning microinstruction addresses of microinstruction sequences whose execution has been suspended, and a means for writing and reading microinstruction address, and FUSITT 11012 state flags, from and to JPD Bus 10142.

As previously described, REPCTR 20280 and PNREG 20282 provide microinstruction addresses for writing of microinstructions into FUSITT 11012. REPCTR 20280 is an eight bit counter and PNREG 20282 is a seven bit register. Eight bit output of REPCTR 20280 is left concatenated with seven bit output of PNREG 20282 to provide fifteen bit microinstruction addresses REPPN. That is, REPCTR 20280 provides the eight low order bits of microinstruction address while PNREG 20282 provides the seven most significant bits of address.

REPCTR may be loaded from Bits 24—31 of JPD Bus 10142, and may be read to Bits 24—31 of JPD Bus 10142. In addition, the eight bits of microinstruction address in REPCTR 20280 may be incremented or decremented as microinstructions are written into FUSITT 11012.

As described above, PNREG 20282 contains the seven most significant bits of microinstruction address. These address bits may be written into PNREG 20282 from Bits 17—23 of JPD Bus 10142. Contents of PNREG 20282 may not, in general, be read to JPD Bus 10142 and may not be incremented or decremented.

Referring to JAM input to SITTNAS 20286 from NC 10226, certain Name evaluate or resolve operations may result in jams. A Jam functions as a call to microinstruction sequences for servicing Jams and are forced by FU 10120 hardware circuitry involved in Name syllable evaluates and resolves.

JAM input to SITTNAS 20286 is comprised of six Jam address bits. Three bits are provided by NC 10226 and three bits are provided from FUSITT 11012's microinstruction output as part of microinstruction sequences for correcting Name syllable evaluates and resolves. The three bits of address from NC 10226 form the most significant three bits of JAM address. One of these bits gates JAM address onto CSADR Bus 20204 and is thus not a true address bit. Output of FUSITT 11012 provides the three least significant bits of JAM address and specifies the particular microinstruction sequence required to service the particular Jam which has occurred. Therefore, during Name evaluate or resolves, the microinstruction sequences provided by FUSITT 11012 to perform Name evaluates or resolves specifies what microinstruction sequences are to be initiated if a Jam occurs. The three bits of JAM address provided by NC 10226 determine, first, that a Jam has occurred and, second, provide two bits of address which, in combination with the three bits of address from FUSITT 11012, specify the particular microinstruction sequence for handling that Jam. JAM address inputs from NC 10226 and from FUSITT 11012 thereby provide six of the fifteen bits of JAM address. The remaining nine bits of JAM address are provided, for example, by hard-wired inputs to NASMUXD 24320. These hard-wired address bits force JAM address to address FUSITT

11012 in blocks of 4 microinstruction addresses, in a manner similar to address inputs to FUDISF 24218 and FUDISENC 24219.

Address inputs provided to SITTNAS 20286 from FUSDT 11010 have been previously described with respect to description of FUCTL 20214 fetch, parse, and dispatch operations. Address inputs provided by EVENT 20284 will be described in a following description of FUCTL 20214's operations with regard to CS 10110's internal mechanisms.

Referring finally to SITTNAS 20286, as previously described SITTNAS 20286 is comprised of EVNTGT 24310 and NASMUX 24312. Inputs are provided to NASMUX 24312, as described above, from FUSDT 11010, mPC 20276, BRCASE 20278, RCWS 10358, REPCTR 20280 and PNREG 20282, and by JAM input. These inputs are, in general, provided with regard to FUCTL 20214's operations in fetching, parsing, and interpreting SOPs and Name syllables. These operations are thereby primarily directly concerned with execution of user's programs, that is the execution of sequences of SINs. NASMUX 24312 selects between these inputs and transfers selected address inputs onto CSADR 20204 as microinstruction addresses to FUSITT 11012 under microinstruction control from microinstruction outputs of FUSITT 11012. Microinstruction address outputs are provided to SITTNAS 20286 from EVENT 20284 in response to Events, described further below, occurring in CS 10110's operations in executing user's programs. These microinstruction addresses from EVENT 20284 are gated onto CSADR 20204, to select appropriate microinstruction sequences, by EVNTGT 24310. EVNTGT 24310 is separated from NASMUX 24312 to allow EVNTGT 24310 to over-ride NASMUX 24312 and provide microinstruction address to EVENT 20284 while NASMUX 24312 is inhibited due to occurrence of certain Events. These Events are, in general, associated with operation of CS 10110's internal mechanisms and structure and operation of EVENT 20284, together with STATE 20294, MCW1 20290, and MCWO 20292, and other portions of RCWS 10358, will be described next below.

c.c. FUCTL 20214 Control Circuitry for CS 10110 Internal Mechanisms (Figs. 244—249)

Certain portions of FUCTL 20214's Control Circuitry are more directly concerned with operation of CS 10110's internal mechanisms, for example CS 10110 Stack Mechanisms. This circuitry may include STATE 20294, EVENT 20284, MCW1 20290 and MCWO 20292, portions of RCWS 10358, REG 20288, and Timers 20296. These FUCTL 20214 control elements will be described next below, beginning with STATE 20294.

a.a.a. State Logic 20294 (Figs. 244A—244Z)

In general, all CS 10110 operations, including execution of microinstructions, are controlled by CS 10110's Operating State. CS 10110 has a number of Operating States, hereafter referred to as States, each State being defined by certain operations which may be performed in that State. Each of these States will be described further below. Current State of CS 10110 is indicated by a set of State Flags stored in a set of registers in STATE 20294. Each State is entered from previous State and is exited to a following State. Next State of CS 10110 is detected by random logic gating distributed throughout CS 10110 to detect certain conditions indicating which State CS 10110 will enter next. Outputs of these Next State Detection gates are provided as inputs to STATE 20294's registers. A particular State register is set and provides a State Flag output when CS 10110 enters the State associated with that particular register. State Flag outputs of STATE 20294's state registers are provided as enable signals throughout CS 10110 to enable initiation of operations allowed within CS 10110's current State, and to inhibit initiation of operations which are not allowed within CS 10110's current State.

Certain of CS 10110's States, and associated STATE 20294 State Registers and State Flag outputs, are:

(1) MO: the initial State of any microinstruction.

State MO is always entered as first data cycle of every microinstruction. During MO, CS 10110's State may not be changed, thus allowing a microinstruction to be arbitrarily aborted and restarted from State MO. In normal execution of microinstructions, State MO is followed by State M1, described below, that is, State MO is exited to State M1. State M0 may be entered from State M0 and from State M1, State AB, State LR, State NR, or State MS, each of which will be described below.

(2) EP: Enable Pause State. State EP is entered when State MO is entered for the first time in a microinstruction. If that microinstruction requests a pause, that microinstruction will force State MO to be re-entered for one clock cycle. If State M0 lasts more than one clock cycle, State EP is entered on each extension of State M0 unless the extension is a result of a pause request.

(3) SR: Source GRF State. SR State is active for one clock cycle wherein SR State register enables loading of a GRF 10354 output register. State SR is re-entered on every State M0 cycle except a State M0 cycle generated by a microinstruction requesting extension of State M0. When all STATE 20294 State Registers are cleared, DP 20218 may set state SR register alone, for purposes of reading from GRF 10354.

(4) M1: Final state of normal microinstruction execution. State M1 is the exit State of normal microinstruction execution. FUSITT 11012 microinstruction register, described below, is loaded with a next microinstruction upon exit from State M1. In addition, State M1 Flag output of STATE 20294 enables all CS 10110 registers to receive data on their inputs, that is data on inputs of these registers are clocked to outputs of these registers. State M1 may be entered from State M1, or from State M0, State MW, State MWA, or State WB.

(5) LA: Load Accumulator Enable State. State LA is entered, upon exit from State M1, by

microinstructions which read data from MEM 10112 to OFFMUXR 23812. As previously described, OFFMUXR 23812 serves as a general purpose accumulator for DESP 20210. STATE LA overlaps into execution of next microinstruction, and persists until data is returned from MEM 10112 in response to a request to MEM 10112. When MEM 10112 signals data is available, by asserting DAVFA, LA State Flag enables loading of data into OFFMUXR 23812. If the next microinstruction references OFFMUXR 23812, that microinstruction execution is deferred until a read to OFFMUXR 23812 is completed, as indicated by CS 10110 exiting from State LA.

(6) RW: Load GRF 10354 Wait State. State RW is entered from State M1 of microinstructions which read data from MEM 10112 to GRF 10354. RW Fag Inhibits initiation of a next microinstruction, that is prevents entry to State M0, and persists through the CS 10110 clock cycle during which data is returned from MEM 10112 in response to a request. State RW initiates Load GRF Enable State, described below.

(7) LR: Load GRF Enable State. State LR is entered in parallel with State RW, on last clock cycle of RW, and persists for one CS 10110 clock cycle. LR Flag enables writing of MEM 10112 output data into GRF 10354.

(8) MR: Memory Reference Trailer State. State MR is entered on transition to State MO whenever a previous microinstruction makes a logical or physical address reference to MEM 10112. MR Flag enables recognition of any MEM 10112 reference Events, described below, which may occur. State MR persists for one clock cycle. If an MEM 10112 memory reference Event occurs, that Event forces exit from State MR to States AB and MA, otherwise State MR has no effect upon selection next state.

(9) SB: Store Back Enable State. State SB is entered during State M0 of a microinstruction following a microinstruction which generated a store back of a result of a EU 10122 operation. SB Flag gates that result to be written into MEM 10112 through JPD Bus 10142.

(10) AB: Microinstruction Abort State. State AB is entered from first MO State after an Event request is recognized, as described in a following description.

State AB may be entered from State MO or from State AB and suppresses an entry into State M1. If there has been an uncompleted reference to MEM 10112, that is, the reference has not been aborted and data has not returned from MEM 10112, JP 10114 remains in State AB until the MEM 10112 reference is completed. Should an abort have occurred due to a MEM 10112 reference Event, State AB lasts two clock cycles only. As will be described in a following description of EVENT 20284, State MO of a first microinstruction of a Handler for an Event causing an abort is entered from State AB. AB Flag gates the Handler address of the highest priority recognized Event onto CSADR Bus 20204 to select a corresponding Event Handler microinstruction sequence. EVENT 20284 is granted control of CSADR Bus 20204 during all State AB clock cycles.

(11) AR: Microinstruction Abort Reset State. State AR is entered in parallel with first clock cycle of State AB and persists for one clock cycle. AR Flag resets various STATE 20294 State Registers when an abort occurs. If there are no uncompleted MEM 10112 references, next State AB clock cycle is the last. On uncompleted MEM 10112 references, State AR is entered, but State AB remains active until reference is complete. Should a higher priority Event request service and be recognized while JP 10114 is in State AB, State AR is reentered. State AB will thereby be active for two clock cycles during all honored Event requests.

(12) MA: MEM 10112 Reference Abort. State MA is entered in parallel with State AB if a MEM 10112 reference is aborted, as indicated by asserted ABORT control signal output from MEM 10112. State MA persists for one clock cycle and State AB flag generates a MEM 10112 Reference Abort Flag which, as described below, results in a repeat of the MEM 10112 reference. AB Flag also resets MEM 10112 Trailer States, described below.

(13) NW: Nano-interrupt Wait State. State NW is entered from State M0 of a microinstruction which issues a Nano-interrupt Request to EU 10122 for an EU 10122 operation. FU 10120 remains in State NW until EU 10122 acknowledges that interrupt. Various EU 10122 Events may make requests at this time. State NW is exited into State AB or State M1.

(14) FM: First Microinstruction of a SIN. State FM is entered in parallel with State M0 on first microinstruction of each SIN and persists for one clock cycle. FM Flag inhibits premature use of AF 24220 and enables recognition of SIN Entry Events. State FM is re-entered upon return from all aborts taken during State M0 of the first microinstruction of an SIN.

(15) SOP: Original Entry to First SIN. State SOP is entered upon entry to State M0 of the first microinstruction of an SOP and is exited from upon any exit from that microinstruction. State SOP is entered only once for each SOP. SOP Flag may be used, for example, for monitoring performance of JP 10114.

(16) EU: EU 10122 Operand Buffer Unavailable. State EU is entered from State M0 of a microinstruction which attempts to read data to EU 10122 Operand Buffer, described in a following description, wherein EU 10122 Operand Buffer is full. When a new SOP is entered, three fetches of data from MEM 10112 may be performed before EU 10122 Operand Buffer is full; two fetches will fill EU 10122 Operand Buffer but EU 10122 may take one operand during a second fetch, thereby clearing EU 10122 Operand Buffer space for a third operand.

(17) NR: Long Pipeline Read. Entry into State NR disables overlap of MEM 10112 reads and disables execution of the next microinstruction. A following microinstruction does not enter State M0 until

requested data is returned from MEM 10112. State NR is entered from State NR or from State M1.

(18) NS: Nonpipeline Store Back. State NS is entered in parallel with State SB whenever a microinstruction requesting a pipeline store back, or a write to MEM 10112, occurs. State NS flag generates entry into State M0 of a following microinstruction upon exit from State SB.

(19) WA: Load Control Store State A. State WA is entered from State M0 of a microinstruction which directs loading of microinstruction into FUSITT 11012. WA State Flag controls selection of addresses to CSADR Bus 20204 for writing into FUSITT 11012, and generates a write enable pulse to FUSITT 11012 to write microinstructions into FUSITT 11012.

(20) WB: Load Control Store State B. State WB is entered from State WA and is used to generate an appropriate timing interval for writing into FUSITT 11012. State WB also extends State M1 to 2 clock cycles to ensure a valid address input to FUSITT 11012 when a next microinstruction is to be read from FUSITT 11012.

Having described certain CS 10110 states, and operations which may be performed within those states, state sequences for certain CS 10110 operations will be described next below with aid of Figs. 244A to 244Z. Fig. 244A to Fig. 244Z represent those state timing sequences necessary to indicate major features of CS 10110 state timing. All state timing shown in Figs. 244A to 244V assumes full pipelining of CS 10110 operations, for example pipelining of reads from and writes to MEM 10112 by JP 10114. Pipelining is not assumed in Figs. 244W to 244Z. Referring to Figs. 244A to 244Z, these figures are drawn in the form of timing diagrams, with time increasing from left to right. Successive horizontally positioned "boxes" represents successive CS 10110 states during successive CS 10110 110 nano-second clock cycles. Vertically aligned "boxes" represent alternate CS 10110 states which may occur during a particular clock cycle. Horizontally extended dotted lines connecting certain states represented in Fig. 244A to 244Z represent an indeterminate time interval which is an integral multiple of 110 nano-second CS 10110 clock cycles.

Referring to Fig. 244A to 244Z in sequence, State Timing Sequences shown therein represent:

(1) Fig. 244A; state timing for execution of a normal microinstruction with no Events occurring and no MEM 10112 references.

(2) Fig 244B execution of a normal microinstruction, with no Events occurring, no MEM 10112 references, and a hold in State M0 for one clock cycle.

(3) Fig. 244C; a microinstruction requests an extension of State M0 for one clock cycle, with no Events occurring and no MEM 10112 references.

(4) Fig. 244D; a write to MEM 10112 from DESP 20210, for example from GRF 10354 or from OFFALU 20242. MEM 10112 port is available and MEM 10112 reference is made during first sequential occurrence of States M0 and M1.

(5) Fig. 244E; a write to MEM 10112 from DESP 20210 as described above. MEM 10112 port is unavailable for an indeterminate number of clock cycles. A MEM 10112 reference is made during first sequential occurrence of States M0 and M1.

(6) Fig. 244F; writing of an EU 10122 result back into MEM 10112. MEM 10112 is available and a write operation is initiated during first sequential occurrence of States M0 and M1.

(7) Fig. 244G; writing back of an EU 10122 result to MEM 10112 as described above. MEM 10112 port is unavailable for an undetermined number of clock cycles, or EU 10122 does not have a result ready to be written into MEM 10112. Write operation is initiated during first sequential occurrence of States M0 and M1.

(8) Fig. 244H; a read of an EU 10122 result into FU 10120. EU 10122 result is not available for an undetermined number of clock cycles.

(9) Fig. 244I; a read from MEM 10112 to OFFMUXR 23812, with no delays. The microinstruction following the microinstruction initiating a read from MEM 10112 does not reference OFFMUXR 23812.

(10) Fig. 244J; a read from MEM 10112 to OFFMUXR 23812 with data from MEM 10112 being delayed by an indeterminate number of clock cycles. The next following microinstruction from that initiating the read from MEM 10112 does not reference OFFMUXR 23812.

(11) Fig. 244K; a read from MEM 10112 to OFFMUXR 23812. The next microinstruction following the microinstruction initiating the read from MEM 10112 references OFFMUXR 23812.

(12) Fig. 244L; a read from MEM 10112 to GRF 10354. The read to GRF 10354 is initiated by the first sequentially occurring States M0 and M1.

(13) Fig. 244M; a read from MEM 10112 to GRF 10354 and to OFFMUXR 23812. In this case, read operations may not be overlapped.

(14) Fig. 244N; JP 10114 honors an Event request and initiates a corresponding Event Handler microinstruction sequence, no MEM 10112 references occur.

(15) Fig. 2440; JP 10114 honors an Event request as stated above. MEM 10112 references are made during the first sequential occurrence of States M0 and M1 and a MEM 10112 reference Event occurs. In case of an MEM 10112 reference event, State MA is entered from one clock cycle. This occurs only if a MEM 10112 reference is made and aborted.

(16) Fig. 244P; an Event occurs in a MEM 10112 reference made during the first sequential occurrence of States M0 and M1. The MEM 10112 reference does not result in a memory reference Event. CS 10110 remains in State AB until the MEM 10112 reference is completed by return of data from MEM 10112.

(17) Fig. 244Q; a read of data from MEM 10112 or JPD Bus 10114 to EU 10122 Operand Queue. EU 10122 Operand Queue is not full.

(18) Fig. 244R; a read of MEM 10112 or JPD Bus 10142 data to EU 10122 Operand Queue. EU 10122 Operand Queue is full when the microinstruction initiating the read is issued.

(19) Fig. 244S; a request for a "nano-interrupt" to EU 10122 by FU 10120 with no Events occurring.

(20) Fig. 244T; FU 10120 submits a "nano-interrupt" request to EU 10122 and an EU 10122 State Overflow, described further in a following description, occurs. No other Events are recognized, as described in a following description of EVENT 20284.

(21) Fig. 244U; FU 10120 submits a "nano-interrupt" request to EU 10122. Another Event is recognized during State M0 and an abort results. First abort state is entered for the non-EU 10122 event. All aborts recognized in State M0 are taken or acknowledged, before entrance into State M0. Therefore, on retry at State M0 of the original microinstruction entered from State M0, next abort recognized is for EU 10122 Stack Overflow Event since EU 10122 Stack Overflow has higher priority.

(22) Fig. 244V; a load of a 27 bit microinstruction segment into FUSITT 11012.

In Figs. 244A to 244V, pipelining MEM 10112 reads and writes, and of JP 10114 operations, has been assumed. In Figs. 244W to 244Z, non-overlapping operation of JP 10114 is assumed.

(23) Fig. 244W; a read of data from MEM 10112 to OFFMUXR 23812.

(24) Fig. 244X; a read of data from MEM 10112 to EU 10122 Operand Queue.

(25) Fig. 244Y; a write of an EU 10122 result into MEM 10112.

(26) Fig. 244Z; a read of a 32 bit SIN word from MEM 10112 in response to a prefetch or conditional prefetch request.

Having described the general structure and operation of STATE 20294, and the operating states and operations of CS 10110, structure and operation of EVENT 20284 will be described next below.

b.b.b. Event Logic 20284 (Figs. 245, 246, 247, 248)

An Event is a request for a change in sequence of execution of microinstructions which is generated by CS 10110 circuitry, rather than by currently executing microinstructions. Occurrence of an Event will result in provision of a microinstruction sequence, referred to as an Event Handler, by FUSITT 11012 which modifies CS 10110's operations in accordance with the needs of that Event. Event request signals may be generated by CS 10110 circuitry internal to JP 10114, that is from FU 10120 or EU 10122 or CS 10110 circuitry external to JP 10114, for example from IOP 10116 or from MEM 10112. Event request signals are provided as inputs to EVENT 20284. As will be described further below, EVENT 20284 masks Event Requests to determine which Events will be recognized during a particular CS 10110 Operating State, assigns priorities for servicing multiple Event Requests, and fabricates Handler addresses to FUSITT 11012 for microinstruction sequences for servicing requests. EVENT 20284 then provides those Handler microinstruction addresses to FUSITT 11012 through EVNTGT 24310, to initiate execution of selected Event Handler microinstruction sequences.

Certain terms and expressions are used throughout the following description. The following paragraphs define these usages and provide examples illustrating these terms. An Event "makes a request" when a condition in CS 10110 hardware operation results in a Event Request signal being provided to EVENT 20284. As will be described further below, these Event Request signals are provided to EVENT 20284 combinatorial logic which determines the validity of those "requests".

An Event Request "is recognized" if it is not masked, that is inhibited from being acted upon. Masking may be explicit, using masks generated by FUSITT 11012, or may be implicit, resulting from an improper CS 10110 State or invalid due to other considerations. That is, certain Events are recognized only during certain CS 10110 States even though those requests may be recognized during certain other states. Any number of requests, for example up to 31, may be simultaneously recognized.

An Event Request is "honored" if it is the highest priority Event Request occurring. When a request is honored, a corresponding address, of a corresponding microinstruction sequence in FUSITT 11012, for its Handler microinstruction sequence is gated onto CSADR Bus 20204 by EVENT 20284. A request is honored when CS 10110 enters State AB. State AB gates the selected Event Handler microinstruction address on CSADR Bus 20284.

To summarize, a number of Events may request service by JP 10114. Of these Events, all, some, or none, may be recognized. Only one Event Request, the highest priority Event Request, will be honored when JP 10114 enters State AB. Microinstruction control of CS 10110 will then transfer to that Event's Handler microinstruction sequence. A necessary condition for entering State AB is that an Event Request has been made and recognized.

A microinstruction sequence "completes", "is completed", or reaches "completion" when CS 10110 exits State M1 while that microinstruction sequence is active. A microinstruction sequence may, as described above, be aborted in State M0 an indefinite number of times before, if ever, reaching completion.

A MEM 10112 reference "completes", "is completed", or reaches "completion" when requested data is returned to the specified destination, that is read from MEM 10112 to the requestor, or MEM 10112 accepts data to be written into MEM 10112.

"Trace Traps" are an inherent feature of microinstructions being executed. Trace Traps occur on every microinstruction of a given type (if not masked), for example during a sequence of microinstructions to perform a Name evaluate or resolve, and occur on each microinstruction of the sequence. In general, a Trace Trap Event must be serviced before execution of the next microinstruction. Trace Traps are distinct

from Interrupts in that an Interrupt, described below, does not occur on execution of each microinstruction of a microinstruction sequence, but only on those microinstructions where certain other conditions must be considered.

"Interrupts" are the largest class of events in JP 10114. Occurrence of an Interrupt may not, in general, be predicted for a particular execution of a particular microinstruction in a particular instance. Interrupts may require service before execution of the next microinstruction, before execution of the current microinstruction can complete, or before beginning of the next SIN. An Interrupt may be unrelated to execution of any microinstruction, and is serviced before beginning of the next microinstruction.

A "Machine Check" is an Event that JP 10114 may not handle alone, or whose occurrence makes further actions by JP 10114 suspect. These events are captured in EVENT 20284 Registers and result in a request to DP 10118 to stop operation of JP 10114 for subsequent handling.

In summary, three major classes of Events in CS 10110 are Trace Traps, Interrupts, and Machine Checks. Each of these class of events will be described in further detail below, beginning with Trace Traps.

The State of all possible Trace Trap Event Requests, whether requesting or not requesting, is loaded into EVENT 20284 Registers at completion of State M1 and at completion of State AB. That is, since Trap Requests are a function of the currently executing microinstruction, the State of a Trap Request will be loaded into EVENT 20284 Trace Trap Registers at end of State M1 of each currently executing microinstruction. Similarly, if any Trap Requests are recognized, State AB will be entered at the end of the first clock cycle of the next following State M0 and their State loaded at end of the State AB.

Recognized, or unmasked, Trap Requests may be pushed onto RCWS 10358 as Pending Requests. Unrecognized, or masked, Trace Trap Requests may be pushed onto RCWS 10358 as Not Pending Requests and are subsequently disregarded. Subsequently, when a microinstruction sequence ends in a return to a calling microinstruction sequence, the Trace Trap Request bits in an RCWS 10358 may be used to generate Trace Trap Event Requests.

Upon exit from State AB, all Trace Trap Requests, except Micro-Break-Point and Microinstruction Trace Traps, described below, are loaded into corresponding EVENT 20284 Trace Trap Request Registers as not requesting. Micro-Break-Point and Microinstruction Trace Traps, are, in general, always latched as requesting at completion of State AB. Trace Traps may be explicitly masked by a Trace Mode Mask, an Indivisibility Mode Mask, and by a Trace Enable input, all generated by FUSITT 11012 as described below. Micro-Break-Point Trap may also be masked by clearing a Trace Enable bit in a Trace Enable field of certain microinstructions containing Trace Traps. In general, masking is effective from State M0 of the microinstruction which generates the mask, through completion of a microinstruction which clears the mask Trace Traps generated by a microinstruction which clears a mask are taken so as to abort a following microinstruction during its M0 State.

Referring to Fig. 245, CS 10110 state timing for a typical Trap Request, and generation of a microinstruction address to a corresponding Trace Trap Handler microinstruction sequence by EVENT 20284 is shown. Fig. 245 is drawn using the same conventions as described above with reference to Fig. 244A to 244Z. In Fig. 245, a microinstruction executing in States M0 and M1 causes a Trace Trap Request but does not generate an MR (Memory Reference) Trailer State. Trace Trap Request to EVENT 20284 is signaled by Time A. This Trace Trap Request is latched into EVENT 20284 Trace Trap Event Registers, and an Abort Request is provided to STATE 20294. At Time B, FU 10120 enters States AB and AR. The microinstruction address for a Handler microinstruction sequence of the highest priority Event present in EVENT 20284 is presented to FUSITT 11012 and execution of the addressed microinstruction sequence begins. At Time C, FU 10120 exits States AB and AR and enters State AB. State AB will be exited at end of the next 110 nanosecond clock cycle. Address of the selected Event Handler microinstruction sequence will remain on CSADR Bus 20204 for duration of State AB. At Time D, a pointer into RCWS 10358, described in a following description, is incremented, thereby effectively pushing the first microinstruction's return control word, that is the microinstruction executing at first State M0, onto RCWS 10358. First microinstruction of the Trace Trap Event Handler microinstruction sequence is provided by FUSITT 11012. Execution of Handler microinstruction sequence will begin at start of the third State M0 of the state timing sequence shown in Fig. 245. EVENT 20284's Trace Trap Register for this event is now latched in nonrequesting state and will remain so until transition out of second State M1 shown in Fig. 245. At this time, EVENT 20284 Registers will latch new Trap Requests. Finally, at Time E, Trace Trap Event Registers of EVENT 20284 are latched with new Trap Requests arising from execution of the microinstruction being executed in States M0 and M1 occurring between Times D and E. Traps due to the microinstruction that was executed in States M0 and M1 before Time A, but were not serviced, are requested again when the previously pushed RCW described above is returned from RCWS 10358 upon return from the Trace Trap Event Handler microinstruction sequence initiated at Time D. All Trace Trap Requests which have been serviced are explicitly cleared in RCWS 10358 RCWs by their Event Handler microinstruction sequences to prevent recurrence of those Trap Requests. Since Trace Trap Event Requests arising from reads or writes to MEM 10112 will recur if those requests are repeated, EVENT 20284 generates memory repeat Interrupts after all aborted MEM 10112 read and write requests to insure that these Traps will eventually be serviced. Event Handler microinstruction sequences for these read and write Trace Trap Events explicitly disable serviced Trace Trap Event Requests by clearing bits in the logical descriptor of the aborted memory read and write requests.

Having described overall structure and operation of Trace Trap Events, certain specific Trace Trap Events will be described in greater detail below. Trace Trap Events occurring in CS 10112 may include Name Trace Traps, SOP Trace Traps, Microinstruction Trace Traps, Micro-Break-Point Trace Traps, Logical Write Trace Traps, Logical Read Trace Traps, UID Read Trace Traps, and UID Write Trace Traps. These Trace Traps will be described below in the order named.

A Name Trace Trap is requested upon every microinstruction sequence that contains an evaluate or resolve of a Name syllable. Name Trace Traps are provided by decoding certain microinstruction fields of those microinstruction sequences. Name Trace Trap field is masked by either Trace Mask, Indivisibility Mask, or Trace Enable, as described above. All of these masks are set and cleared by microinstruction control signals provided during microinstruction sequences calling for resolves or evaluates of Name syllables.

A SOP Trace Trap may be requested whenever FU 10120 enters State FM (First Microinstruction of an SOP). SOP Trace Traps may be masked by Trace Mask, Indivisibility Mask, or Trace Trap Enable, again provided by microinstruction control outputs of FUSITT 11012. In general, the first microinstruction of such a microinstruction sequence interrupting such SOPs is not completed before a Trace Trap is taken.

Microinstruction Trace Traps may be requested upon completion of microinstructions which do not contain a Return Command, that is those microinstructions which do not return microinstruction control of CS 10110 to the calling microinstruction sequence. For microinstruction sequences containing Return Commands, state of microinstruction Trace Trap Request in a corresponding RCW is used. Every microinstruction for which a Microinstruction Trace Trap is not masked is aborted during State M0 of execution of that microinstruction. Microinstruction Trace Traps may be masked by Trace Mask, Indivisibility Mask, or Trace Enable from FUSITT 11012. A Micro-Break-Point Trap may be requested upon execution of microinstructions which do not contain Return Commands, but in which a Trace Enable bit in a microinstruction is asserted. A Micro-Break-Point Trap may be masked by Trace Mask, Indivisibility Mask, or Trace Enable. In addition, a Trace Enable bit of a microinstruction field in these microinstruction sequences controls recognition of Micro-Break-point Traps. Micro-Break-Point Traps are thereby requested whenever a microinstruction Trace Trap is requested, but have additional enabling conditions expressed in the microinstructions. Since only recognized Traps are pushed onto RCWS 10358 in a RCW, a Microinstruction Trace Trap and a Micro-Break-Point Trap having different request states may be present in RCWS 10358 concurrently.

Logical Write Trace Traps may be requested when enabled by a bit set in a logical descriptor during a microinstruction sequence submitting a write request to MEM 10112 and using logical descriptors to do so. Logical Write Trace Traps are recognized only if they occur during a state which will be immediately followed by State MR (Memory Reference Trailer). A Logical Write Trace Trap will result in the MEM 10112 write request being aborted. Logical Write Trace Traps may be masked by Trace Masks, Indivisibility Mask, or Trace Trap Enable. A further condition for recognition of a Logical Write Trace Trap is determined by the state of certain bits in a logical descriptor of the memory write request. Logical Write Trace Traps are, in general, not pushed onto RCWS 10358 as part of a RCW since aborted MEM 10112 requests are re-generated so that Logical Write Trace Traps may be repeated.

Logical Read Trace Traps are similar in all respects to the Logical Write Trace Traps, but occur during MEM 10112 read requests. Generation of Logical Read Trace Traps is controlled again in part by certain bits in logical descriptors of MEM 10112 read requests.

In certain implementations of CS 10110, UID Trace Traps may be requested when FU 10120 requests an MEM 10112 read operation based upon a UID address or pointer. UID Read Trace Traps are recognized if requested and there is, in general, no explicit masking of UID Read Trace Traps. Generation of UID Read Trace Traps is controlled by certain bits in MEM 10112 read request logical descriptors. UID Read Trace Trap Requests result in the MEM 10112 read requests being aborted and CS 10110 entering State AB. Handler microinstruction sequences for UID Read Trace Traps will, in general, reset the trapped enable bit in the MEM 10112 read request logical descriptor before re-issuing the MEM 10112 read request.

UID Write Trace Traps are similar to UID Read Trace Traps, and are controlled by bits in the logical descriptor in MEM 10112 write request based upon UID addresses or pointers.

Having described above structure and operation of Trace Trap Events, CS 10110 Interrupt Events will be described next below.

As previously described, Interrupts form the largest class of CS 10110 Events. Interrupts may be regarded as falling into one or more of several classes. First, Memory Reference Repeat Interrupts are those Interrupt Events associated, in general, with read and write requests to MEM 10112 in which a read or write request is submitted to MEM 10112, and an Interrupt Event results. That Interrupt Event is handled, and the MEM 10112 request repeated. Second, Deferred Service Interrupts are those Interrupts wherein CS 10110 defers service of an Interrupt until entry to a new SIN. Fourth, Microinstruction Service Interrupts occur when a currently executing microinstruction requires assistance of an Event Handler microinstruction sequence to be completed. Finally, Asynchronous Interrupt Events may occur at any time and must be serviced before CS 10110 may exit State M0 of the next microinstruction. These Interrupt Events will be described next below in the order named.

A Memory Reference Repeat Interrupt is requested, for example, if a microinstruction executes a command, and a corresponding RCW read from RCWS 10358 indicates that a memory reference was

aborted before entrance to the microinstruction sequence from which return was executed. This type of Interrupt Event occurs for all aborted memory references. If an event is honored, that is abort state is entered, for any event and there is a memory reference outstanding, not aborted, the memory reference completes before State AB is exited. No memory Repeat Interrupt Request will be written into the RCW written onto RCWS 10358. Conversely, if a memory reference is aborted, even if the event honored is not that event which aborted the memory reference, a Memory Repeat Interrupt Request will be written into a RCW pushed onto a RCWS 10358.

There are two state timing sequences for execution of Memory Repeat Interrupts. In the first case, there are no MEM 10112 references in the microinstruction executing a Return Command. In the second case, a microinstruction executing a Return Command executes a return and also makes a MEM 10112 reference. Referring to Fig. 246, a CS 10110 State Timing Diagram for the first case is shown. Fig. 246 is drawn using the same conventions as used in Fig. 244 and 245. As described above, in the first case a microinstruction executing a Return Command is executed in States M0 and M1 following Time D. An aborted MEM 10112 reference was made in States M0 and M1 preceding Time A. An MEM 10112 Reference Abort Request is made upon CS 10110's entry into State MR following Time A. Since a Memory Repeat Interrupt is requested only from a RCW provided by RCWS 10358, a Memory Repeat Interrupt is indicated only if a microinstruction executes a Return Command resulting in RCWS 10358 providing such an RCW. Therefore, a Memory Repeat Interrupt Request Register of EVENT 20284 is loaded with "not requesting" at this time. At Time B, CS 10110 enters State AB, State AR, and State MA. At this time, a Memory Reference Abort Request is asserted and written into an RCW when State AB is exited just before Time D. At Time D, CS 10110 exits State AR and State MA. As just described, CS 10110 will remain in State B until Time D. At Time D, Memory Reference Abort Request is written into RCWS 10358 as part of an RCW and, as described further below, various RCWS 10358 Stack Pointers are incremented to load that RCW into RCWS 10358. At this time, EVENT 20284's Interrupt Request Register receives "no request" as state of Memory Repeat Interrupt. First microinstruction of Memory Repeat Interrupt Handler microinstruction sequence is provided by FUSITT 11012. At Time E, the last microinstruction of the Memory Repeat Interrupt Handler microinstruction sequence is provided by FUSITT 11012 and a Return Command is decoded. RCWS 10358 Previous Stack Pointer, previously described, is selected to address RCWS 10358 to provide the previously written RCW as output to EVENT 20284's Memory Repeat Interrupt Event Register. At Time F, EVENT 20284's Memory Repeat Interrupt Register is loaded from output of RCWS 10358 and RCWS 10358's Stack Register Pointers are decremented. At this time, Memory Repeat Interrupt Request is made and, as described below, is written into the current Return Control Word, whether honored or not. JP 10114 then repeats the aborted MEM 10112 reference.

In the second case, a State Timing Sequence wherein the microinstruction executing a return also makes a MEM 10112 reference, CS 10110 State Timing is identical up to Time F. At Time F, MEM 10112 Repeat request is not recognized and the state of Memory Repeat Interrupt written into the current Return Control Word is "not requesting" unless a current MEM 10112 reference is aborted. The previous MEM 10112 Repeat Interrupt Request is disregarded as it is assumed that it is no longer required. Thus, there are two ways to avoid, or cancel a Memory Repeat Interrupt Request. First, that portion of a RCW receiving a MEM 10112 Repeat Interrupt Request may be rewritten as "not requesting". Second, an aborted MEM 10112 reference may be made in the same microinstruction that returns from a Handler servicing the aborted MEM 10112 reference.

Certain CS 10110 Events result in aborting a MEM 10112 read or write references and may result in repeat of MEM 10112 references. These events may include:

(1) Logical read and write Traps and, in certain implementations of CS 10110, UID read and write Traps, previously discussed;

(2) A PC 10234 miss;

(3) Detection of a protection Violation by PC 10234;

(4) A Page Crossing in a MEM 10112 read or write request;

(5) A Long Address Translation, that is an ATU 10228 miss requiring JP 10114 to evaluate a logical descriptor to provide a corresponding physical descriptor;

(6) Detection of a reset dirty bit flag from ATU 10228 upon a MEM 10112 write request as previously described;

(7) An FU 10122 stack overflow;

(8) An FU 10122 Illegal Dispatch;

(9) A Name Trace Trap event as previously described;

(10) A Store Back Exception, as will be described below;

(11) EU 10122 Events resulting in aborting of a Store Back, that is a write request to MEM 10112 from EU 10122;

(12) A read request to a non-accelerated Stack Frame, that is a Stack Frame presently residing in MEM 10112 rather than accelerated to JP 10114 Stack Mechanisms; and,

(13) Conditional Branches in SIN sequences resulting outstanding MEM 10112 read reference from PREF 20260; and,

Of these Events, Logical Read and Write Traps, UID Read and Write Traps, and Name Trace Traps have been previously described. Other Events listed above will be described next below in further detail.

A PC 10234 Miss Interrupt may be requested upon a logical MEM 10112 reference, that is when a logical descriptor is provided as input to ATU 10228 and a protection state is not encached in PC 10234. PC 10234 will, as previously described, indicate that a corresponding PC 10234 entry is not present by providing a Event Protection Violation (EVENTPVIOL) output to EVENT 20284. PC 10234 will concurrently assert an Abort output (ABORT) to force CS 10110 into State AB and thus abort that MEM 10112 reference.

A Page Crossing MEM 10112 Reference Interrupt is requested if a logical MEM 10112 reference, that is a logical descriptor, specifies an operand residing on two logical pages of MEM 10112. An output of ATU 10228 will abort such MEM 10112 references by asserting an Abort output (ABORT).

A Protection Violation Interrupt is requested if a logical MEM 10112 reference does not possess proper access rights, a mode violation, or if that reference appears to refer to an illegal portion of that object, an extent violation. Again, PC 10234 will indicate occurrence of a Protection Violation Event, which may be disabled by a microinstruction control output of FUSITT 11012.

A Long Address Translation Event may be requested upon a logical MEM 10112 reference for which ATU 10228 does not have an encached entry. ATU 10228 will abort that MEM 10112 reference by asserting outputs ABORT and Long Address Translation Event (EVENTLAT).

A Dirty Bit Reset Event Interrupt may be requested when JP 10114 attempts to write to an MEM 10112 page having an encached entry in ATU 10228 whose dirty bit is not set. ATU 10228 will abort that MEM 10112 write request by asserting outputs ABORT and Write Long Address Translation Event (EVENTWLAT).

An FU 10120 User Stack Overflow Event may be requested if the distance between a Current Frame Pointer and a Bottom Frame Pointer, previously described with reference to CS 10110 Stack Mechanisms, is greater than a given value. As previously described, in CS 10110 this value is eight. A User Stack Overflow Event will continue to be requested until either Current Frame Pointer or Bottom Frame Pointer changes value so that the difference limit defined above is no longer violated. A User Stack Overflow Event may be masked by a Trace Mask, an Indivisibility Mask, or by enable outputs of a microinstruction from FUSITT 11012. A Handler microinstruction sequence for User Stack Overflow Events must be executed with one or more of these masks set to prevent recursion of these events. CS 10110 is defined to be running on Monitor Stack (MOS) 10370 when User Stack Overflow Events are masked. User Stack Overflow Events are not loaded into any of EVENT 20284's Event Registers, nor are these events written into a RCW to be written onto RCWS 10358.

Illegal EU 10122 Dispatch Events are requested by EUSDT 20266 if FU 10120 attempts to dispatch, or provide an initial microinstruction sequence address, to EU 10122 to a EUSITT address which is not accessable to a user's program. Illegal EU 10122 Dispatch Events are, in general, not masked. Illegal EU 10122 Dispatch Event Requests are cleared upon CS 10110 exits from State AB. The Handler microinstruction sequence for Illegal EU 10122 Dispatch Events should, in general, reset Illegal EU 10122 Dispatch Event entries in RCWs to prevent recursion of these events.

EU 10122 will indicate a Store Back Exception Event if any one of a number of exceptional conditions arise during arithmetic operations. These events are recognized when CS 10110 enters State SB and are ignored except during Store Back to MEM 10112 of EU 10122 results. These Events may be disabled by microinstruction output of FUSITT 11012 but are, in general, not masked. Store Back Exception Events may be written into RCWs, to be stored in RCWS 10358, and are cleared upon CS 10110's exit from State AB. Again, a Store Back Exception Event Handler microinstruction sequence should reset Store Back Exception Events written into RCWs to prevent recursion of these events.

As described above, the next major class of Interrupt Events are Deferred Service Interrupts. CS 10110 defers service of Deferred Service Interrupts until entry of a new SOP Deferred Service Interrupts which have been recognized will be serviced before completion of execution of the first microinstruction of that new SOP. Deferred Service Interrupts include Nonfatal MEM 10112 Errors, Interval Timer Overflows, and Interrupts from IOS 10116. These Interrupts will be described below, in the order named.

A Nonfatal MEM 10112 Interrupt is signaled by MEM 10112 upon occurrence of a correctable (single bit) MEM 10112 error. Nonfatal Memory Error Interrupts are recognized only during State M0 of the first microinstruction of an SOP. MEM 10112 will continue to assert Nonfatal Memory Error Interrupt until JP 10114 issues an acknowledgement to read MEM 10112's Error Log.

An Interval Timer Overflow Interrupt is indicated by TIMERS 20296 when, as described below, an Interval Timer increments to zero, thus indicating lapse of an allowed time limit for execution of an operation. Interval Timer Overflow Interrupts are recognized during State M0 of the first microinstruction of a SOP. TIMERS 20296 will continue to request such interrupts until cleared by a microinstruction output of FUSITT 11012.

IOS 10116 will indicate an IOS 10116 Interrupt to indicate that an inter-processor message from IOS 10116 to JP 10114 is pending. IOS 10116 will continue to assert an IOS 10116 Interrupt Request, which is stored in a register, until cleared by a microinstruction control output of FUSITT 11012. IOS 10116 Interrupts are recognized during State M0 of the first microinstruction of an SOP.

The next major class of CS 10110 events are Interrupts due to the requirement by microinstruction sequences to be serviced in order to complete execution. These Interrupts must be serviced before a microinstruction sequence may be completed. Microinstruction Service Interrupts include Illegal SOP Events, Microinstructions Not Present in FUSITT 11012 Events, an attempted parse of a hung INSTB 20262, underflow of an FU 10120 Stack, an NC 10226 Cache Miss, or an EU 10122 Stack Overflow. Each of these

events will be described below, in the order named.

An Illegal SOP Event is indicated by FUSDT 11010 to indicate that a current SOP Code is a Long Code, that is greater than eight bits, while the current dialect (S-Language) expects only Short Operation Codes, that is eight bit SOPs. An Illegal SOP Interrupt is not detected for unimplemented SOPs within the proper code length range. Illegal SOP Events are, in general, not masked. FUSDT 11010 continues to indicate an Illegal SOP Event until a new SOP is loaded into OPCODEREG 20268. Illegal SOP Events are recognized during the first microinstruction of an SOP, that is during State FM. Should a Handler microinstruction sequence for a higher priority event change contents of OPCODEREG 20268, a previous Illegal SOP Event will be indicated again when the aborted SOP is retried.

Absence of a Microinstruction in FUSITT 11012 is indicated by FUSITT 11012 asserting a Control Store Address Invalid (CSADVALID). This FUSITT 11012 output indicates that that particular microinstruction address points outside of FUSITT 11012's address space. Output of FUSITT 11012 in such event is not determined and parity checking, described below, of microinstruction output is inhibited. The Handler microinstruction sequence for these Events will load FUSITT 11012 address zero with the required microinstruction from MEM 10112, as previously described, and return to the original microinstruction sequence.

An attempted parse of a hung INSTB 20262 is indicated by INSTBWC 24110 when a parse operation is attempted, INSTB 20262 is empty, and PREF 20260 is not currently requesting SINs from MEM 10112. In general, these Events are not masked. If a higher priority Event is serviced, these Events are indicated again when the aborted microinstruction is retried if the original conditions still apply.

An FU 10120 Stack Underflow Event is requested when a current microinstruction references a Previous Stack Frame which is not in an accelerated stack, that is, the Current Stack Pointer equals Bottom Stack Pointer. FU 10120 Underflow Events are, in general, not masked and are requested again on a retry if the microinstruction is aborted and this event has not been serviced.

An NC 10226 Miss Interrupt occurs on a MEM 10112 read or write operation when a load or read of NC 10226 is attempted and there is no valid NC 10226 block corresponding to that Name syllable. An NC 10226 Miss Event does not result in a request for a Name evaluate or resolve. In general, these Events are not masked and result in a request being issued again if the microinstruction resulting in that Event is retried and has not been serviced.

An EU 10122 Stack Overflow Event is requested from EU 10122 to indicate that EU 10122 is currently already servicing at least one level of Interrupt an FU 10122 is requesting another. As will be described in a following description of EU 10122, EU 10122 contains a one level deep stack for handling of Interrupts. EU 10122 Stack Overflow Events are enabled during State NW. All previously pending events will have been serviced before EU 10122 Stack Overflow Event requests are recognized. These Events will be serviced immediately upon entry into a following State M0, being the highest priority interrupt event. EU 10122 Stack Overflow Events may, in general, not be masked and once recognized are the next honored event.

Finally, the third major class of CS 10110 Interrupt Events are Asychronous Events. Asynchronous Events must, in general, be serviced before exiting State M0 of a microinstruction after they are recognized. Asychronous Events include Fatal Memory Error Events, AC Power Failure Events, Egg Timer Overflow Events, and EU 10122 Stack Underflow Events. CS 10110 Egg Timer is a part of TIMERS 20296 and will be discussed as part of TIMERS 20296. These events will be described below, in the order referred to.

Fatal MEM 10112 Error Events are requested by MEM 10112 by assertion of control signal output PMODI, previously described, when last data read from MEM 10112 contains a noncorrectable error. Fatal MEM 10112 Error Events are recognized on first State M0 after occurrence. Fatal MEM 10112 Error Events are stored in an EVENT 20284 Event Register and are cleared upon entry into its service microinstruction sequence. In general, Fatal MEM 10112 Error Events may not be masked.

AC Power Failure Events are indicated by DP 10118 by assertion of output signal ACFAAIL when DP 10118 detects a failure of power to CS 10110. Recognition of AC Power Failure Events is disabled upon entry to AC Power Failure Event Handler microinstruction sequence. No further AC Power Failure Events will be recognized until DP 10118 reinitiates JP 10114 operation.

As will be described further below, FUCTL 20214's Egg Timer is a part of TIMERS 20296. Egg Timer Overflow Events are indicated by TIMERS 20296 whenever TIMERS 20296's Egg Timer indicates overflow of Egg Timer Counter. Egg Timer Overflow Events may be masked as described in a following description.

Finally, EU 10122 Stack Underflow Events are signaled by EU 10122 when directed to read a word from EU 10122 Stack Mechanism and there is no accelerated stack frame present. EU 10122 will continue to assert this Event Interrupt until acknowledged by JP 10114 by initiation of a Handler microinstruction sequence.

The above descriptions of CS 10110 events have stated that recognition of certain of those Events may be masked, that is inhibited to allow recognition of other Events having higher priority. Certain of these masking operations were briefly described in the above descriptions and will be described in further detail next below. In general, recognition of Events may be masked in five ways, four of which are properly designated as masks. These four masks are generated by microinstruction control from FUSITT 11012 and include Asychronous Masks for, in general, Asychronous Events. Monitor Masks are utilized for those CS 10110 operations being performed on Monitor Stack (MOS) 10370, as previously described with reference to CS 10110 Stack Mechanisms. Trace Mask is utilized with reference to Trace Trap Events. Indivisible Mask

is generated or provided by FUSITT 11012 as an integral or indivisible part of certain microinstructions and allow recognition of certain selected events during certain single microinstructions. Certain other Events, for example Logical Read and Write Traps and UID Read and Write Traps, are recognized or masked by flag bits in logical descriptors associated with those operations. Finally, certain microinstructions result in FUSITT 11012 providing microinstruction control outputs enabling or inhibiting recognition of certain events, but differ from Indivisible Masks in not being associated with single particular microinstructions.

Referring to Fig. 247, the relative priority level and applicable masks of certain CS 10110 Events are depicted therein in three vertical columns. Information regarding priority and masking of particular Events is shown in horizontal entries, each comprising an entry in each of these three vertical columns. Left hand column, titled Priority Level, states relative priority of each Event entry. Second column, titled EVENT, specifies which Event is referred to in that table entry. A particular Event will yield priority to all higher priority Events and will take presidence over all lower priority Events. Fig. 247's third column, titled Masked By, specifies for each entry which masks may be used to mask the corresponding Event. A indicates use of Asychronous Masks, M use of Monitor Mask, T use of Trace Trap Mask, and I represents that Indivisible Mask may be used. DES indicates that an Event is enabled or masked by flag bits of logical descriptors, while MCWD indicates that a particular Event may be masked by microinstruction control signal outputs provided by FUSITT 11012. NONE indicates that a particular Event may, in general, not be masked.

The final major class of CS 10110 event was described above as Machine Check Events. In general, if any of these Events are detected by logic gating in EVENT 20284, EVENT 20284 will provide a Check Machine signal to DP 10118. DP 10118 will then stop operation of JP 10114 and Machine Check Event Handler microinstruction sequences will be initiated. Among these Machine Check Events are wherein FU 10120 is etterning to store back an EU 10122 result to MEM 10112 and EU 10122 signals a parity error in EU 10122's Control Store. These events are stored in EVENT 20284 Event Registers and recogized when FU 10120 enters State AB. EU 10122 will have previously ceased operation until a corrective microinstruction sequence may be initiated. The same Event will occur if FU 10120 attempts to use an EU 10122 arithmetic operation result or test operation result having a parity error in EU 10122's Control Store. Should MOS 10370 overflow or underflow, this event will be detected, FU 10120 operations stopped, and corrective microinstruction sequences initiated. MOS 10370 overflow or underflow occurs whenever a previous MOS 10370 Stack Frame is referenced, whenever MOS 10370 Stack Pointer equals MOS 10370 Bottom Stack Pointer, or the difference between MOS 10370 Current and Bottom Stack Pointers is greater than sixteen. Underflows result in a transfer of operation to MIS 10368, while overflows are handled by DP 10118. Finally, a Machine Check Event will be requested when a parity error is detected in a microinstruction currently being provided by FUSITT 11012 during State M0 of that microinstruction.

Having described general operation of EVENT 20284, the structure and operation of EVENT 20284 will be described briefly next below.

Referring to Fig. 248, a partial block diagram of EVENT 20284 is shown. EVENT 20284 includes Event Detector (EDET) 24810, Event Mask and Register Circuitry (EMR) 24812, and Event Handler Selection Logic (EHS) 24814. EDET 24810 is comprised of random logic gating and, as previously described, receives inputs representing event conditions from other portions of CS 10110's circuitry. EDET 24810 detects occurrences of CS 10110 operating conditions indicating that Events have occurred and provides outputs to EMR 24812 indicating what Events are requested.

EMR 24812 includes a set of registers, for example SN74S194s, comprising EVENT 20284's Event Registers. These registers are enabled by mask inputs, described momentarily, to enable masking of those Events which are latched in EVENT 20284's Event Registers. Certain Events, as previously described, are not latched and logic gating having mask enable inputs is provided to enable masking of those events which are not latched. EMR 24812 mask inputs are Asychronous, Monitor, Trace Trap, and Indivisible Masks, respectively AMSK, MMSK, TMSK, and ISMK, provided from FUSITT 11012. Mask inputs derived from FUSITT 11012 microinstruction outputs (mWRD) are provided from microinstruction control outputs of FUSITT 11012. EMR 24812 provides outputs representing mask and unmask events which have been requested to EHS 24814.

EHS 24814 is comprised of logic gating detecting which of EHS 24814's unmasked Event Requests is of highest priority. EHS 24814 selects the highest priority unmasked Event Request input and provides a corresponding Event Handler microinstruction address to EVNTGT 24310 through ADRA Bus 24322. These address outputs of EHS 24814 are five bit addresses selecting the initial microinstruction of the Event Handler microinstruction sequence of the current highest priority unmasked Event. As previously described with reference to NASMUX 24312, certain inputs of ENTGT 24310 are hard-wired to provide a full fifteen bit address output from EVNTGT 24310. EVENT 20284 also provides, from EHS 24814, an Event Enable Select (EES) output to SITTNAS 20286 to enable EVNTGT 24310 to provide microinstruction addresses to CSADR Bus 20204 when EVENT 20284 must provide a microinstruction address for handling of a current Event.

Having described the structure and operation of FUCTL 20214's circuitry providing microinstruction addresses to FUSITT 11012, FUSITT 11012 will be described next below.

c.c.c. Fetch Unit S-Interpreter Table 11012 (Fig. 249)

Referring to Fig. 249, a partial block diagram of FUSITT 11012 is shown. Address (ADR) and Data

(DATA) inputs of Micro-Instruction Control Store (mCS) 24910 are connected, respectively, from CSADR Bus 20204 through Address Driver (ADRDRV) 24912 and from JPD Bus 10142 through Data Driver (DDRV) 24194. mCS 24910 comprises a memory for storing sequences of microinstructions currently being utilized by CS 10110. mCS 24910 is an 8K (8192) word by 80 bit wide memory. That is, mCS 24910 may contain, for example, up to, 8192 80 bit wide microinstructions. Microinstructions to be written into mCS 24910 are provided, as previously described, to mCS 24910 DATA input from JPD Bus 10142 through DDRV 24914. Addresses of microinstructions to be written into or read from mCS 24910 are provided to mCS 24910 ADR input from CSADR Bus 20204 through ADRDRV 24912. ADRDRV 24912 and DDRV 24914 are buffer drivers comprised, for example, of SN74S240s and SN74S244s.

Also connected from output of ADRDRV 24912 is input of Nonpresent Micro-Instruction Logic (NPmIS) 24916. NPmIS 24916 is comprised of logic gating monitoring read addresses provided to mCS 24910. When a microinstruction read address present on CSADR Bus 20204 refers to an address location not within mCS 24910's address space, that is of a non-present microinstruction, NPmIS 24916 generates an Event Request output indicating this occurrence. As previously described FUCTL 20214 will then call, and execute, microinstructions so addressed from MEM 10112.

As indicated in Fig. 249, mCS 24910 provides three sets of outputs. These outputs are Direct Output (DO), Direct Decoded Output (DDO), and Buffered Decode Output (BDO). In general, control information within a particular microstruction word is used on next clock cycle after the address of that particular microinstruction word has been provided to mCS 24910 ADR input. That is, during a first clock cycle a microinstruction's address is provided to mCS 24910 ADR input. That selected microinstruction appears upon mCS 24910's DO, DDO, BDO outputs during that clock cycle and are used, after decoding, during next clock cycle. Outputs DO, DDO, BDO differ in delay time before decoded microinstruction outputs are available for use.

mCS 24910 DO output provides certain bits of microinstruction words directly to particular destinations, or users, through Direct Output Buffer (DOB) 24918. These microinstructions bits are latched and decoded at their destinations as required. DOB 24918 may be comprised, for example, of SN74SO4s.

mCS 24910's DDO output provides decoded microinstruction control outputs for functions requiring the presence of fully decoded control signals at the start of the clock cycle in which those decoded control signals are utilized. As shown in Fig. 249, mCS 24910's DDO output is connected to input of Direct Decode Logic (DDL) 24920. DDL 24920 is comprised of logic gating for decoding certain microinstruction word bits during same clock cycle in which those bits are provided by mCS 24910's DDO. These microinstruction bits are provided, as described above, during the same clock cycle in which a corresponding address is provided to mCS 24910's ADR input. During this clock cycle, DDL 24920 decodes mCS 24910's DDO microinstruction bits to provide fully decoded outputs by end of this clock cycle. Outputs of DDL 24920 are connected to inputs of Direct Decode Register (DDR) 24922. DDR 24922 is a register comprised, for example, of SN74S374s. DDL 24920's fully decoded outputs are loaded into DDR 24922 at the end of the clock cycle during which, as just described, an address is provided to mCS 24910's ADR input and mCS 24910's corresponding DDO output is decoded by DDL 24920. Fully decoded microinstruction control outputs corresponding to mCS 24910's DDO outputs are thereby available at start of the second clock cycle. Microinstruction control outputs of DDR 24922 are thereby available to FU 10120 at start of the second clock cycle for those FU 10120 operations requiring immediate, that is undelayed, microinstruction control signal outputs from FUSITT 11012.

Finally, mCS 24910's BDO is provided for those FU 10120 operations not requiring microinstruction control signals immediately at the start of the second clock cycle. As shown in Fig. 249, mCS 24910's BDO is connected to inputs of Buffered Decode Register (BDR) 24924. Microinstruction word output bits from mCS 24910's BDO are provided to inputs of BDR 24924 during the clock cycle in which a corresponding address is provided to mCS 24910's ADR input. mCS 24910's BDO outputs are loaded into BDR 24924 at end of this clock cycle. BDR 24924's outputs are connected to inputs of Buffered Decode Logic (BDL) 24926. BDL 24926 is comprised of logic gating for decoding outputs of BDR 24924. BDL 24926 thereby provides decoded microinstruction control outputs to FU 10120 at some delayed time after start of the second clock cycle. Microinstruction control outputs from BDL 24926 are thereby delayed in time from the appearance of microinstruction control outputs of DDR 24922 but, as BDR 24924 stores microinstruction word bits rather than decoded microinstruction word bits, BDR 24924 is required to store proportionately fewer bits than DDR 24922.

Finally, as shown in Fig. 249 outputs of DDR 24922 and BDR 24924, are connected to inputs of Microinstruction Word Parity Checker (mWPC) 24928. mWPC 24928 is comprised of logic gating for checking parity of outputs of DDR 24922 and BDR 24924. A failure in parity of either output of DDR 24922 and BDR 24924 indicates a possible error in microinstruction output from mCS 24910. When such an error is detected by mWPC 24928, mWPC 24928 generates a corresponding Microinstruction Word Parity Error (mWPE).

d.d. CS 10110 Internal Mechanism Control

Associated with SR's 10362, the stack mechanism area of GRF 10354, are two CS 10110 control structures primarily associated with operation of CS 10110's internal mechanisms. A first of these referred to as Machine Control Block, describes current execution environment of JP 10114 microprograms, that is,

JP 10114 microinstruction sequences. Machine Control Block is comprised of two information words residing in MCW1 20290 and MCW0 20292. These Machine Control Words contain all control state information necessary to execute JP 10114's current microprogram. Second control structure is a portion of RCWS 10358, which as previously described parallels the structure of SR's 10362. Each register frame on MIS 10368 or MOS 10370 has, with exception of Top (Current) Register Frame, associated with it a Return Control Word (RCW) residing in RCWS 10358. RCWs are created when MIS 10362 or MOS 10370 register frames are pushed, that is moved onto MIS 10368 or MOS 10370 due to creation of a new Current Register Frame. A current RCW does not exist in a present embodiment of CS 10110.

RCWS 10358 will be described first next below, followed by Machine Control Block.

a.a.a. Return Control Word Stack 10358 (Fig. 251)

Referring to Fig. 251, a diagramic representation of a RCWS 10358 RCW is shown. As previously described, RCWS 10358 RCWs contain information necessary to reinitiate or continue execution of a microinstruction sequence if execution of that sequence has been discontinued.

Execution of a microinstruction sequence may be discontinued due to a requirement to service a CS 10110 Event, as described above, or if that microinstruction sequence has called for execution of another microinstruction sequence, as in a Branch or Case Operation.

As shown in Fig. 251, each RCW may contain, for example, 32 bits of information. RCW Bits 16 to 31 inclusive are primarily concerned with storing current microinstruction address of microinstruction sequences which have been discontinued, as described above. Bits 17 to 31 inclusive contain microinstruction sequence return address. Return address is, as previously described, address of the microinstruction currently being executed of a microinstruction sequence whose execution has been discontinued. When JP 10114 returns from servicing of an Event or execution of a called microinstruction sequence, return address is provided from RCWS 10358 to SITTNAS 20286 and through CSADR Bus 20204 to FUSITT 11012 as next microinstruction address to resume execution of that microinstruction sequence. Bit 16 of an RCW contains a state bit indicating whether the particular microinstruction referred to by return address field is the first microinstruction of a particular SOP. That is, Bit 16 of an RCW stores CS 10110 State FM.

Bits 8 to 15 inclusive of an RCW contain information pertaining to current condition code of JP 10114 and to pending Interrupt Requests. In particular, Bit 8 contains a condition code bit which, as previously described indicates whether a particular test condition has been met. RCW Bit 8 is thereby, as previously described, a means by which JP 10114 may pass results of a particular test from one microinstruction sequence to another Bits 9 to 15 inclusive of an RCW contain information regarding currently pending Interrupts. These Interrupts have been previously discussed, in general, with reference to EVENT 20284. In particular, RCW Bit 9 contains pending state of Illegal EU 10122 Dispatch Interrupt Requests; RCW Bit 10 contains pending state of Name Trace Trap Request; RCW Bit 11 contains pending state of Store Back Interrupt Request; RCW Bit 12 contains pending state of Memory Repeat Interrupt Request; RCW Bit 13 contains pending state of SOP Trace Trap Request; RCW Bit 14 contains pending state of Microtrace Trap Request; and, RCW Bit 15 contains pending state of Micro-Break Point Trap Request. Interrupt Handling microinstruction sequence which require use of CS 10110 mechanisms containing information regarding pending Interrupts must, in general, save and store that information. This save and restore operation is accomplished by use of Bits 9 to 15 of RCWS 10358's RCWs. Upon entry to an Interrupt Handling microinstruction sequence, these bit flags are set to indicate Interrupts which were outstanding at time of entry to that microinstruction sequence. Because these bits are used to initiate Interrupt Request upon returns, pending Interrupts may be cancelled by resetting appropriate bits of Bits 9 to 15 upon return. This capability may be used to implement Microinstruction Trace Traps, previously described.

As indicated in Fig. 251, RCW Bits 0 to 7 are not utilized in a present embodiment of CS 10110. RCW bits 0 to 7 are not implemented in a present embodiment of CS 10110 but are reserved for future use.

As previously described, RCWs may be written into or read from RCWS 10358 from JPD Bus 10142. This allows contents of RCWS 10358 to be initially written as desired, or read from RCWS 10358 to MEM 10112 and subsequently restored as required for swapping of processes in CS 10110.

b.b.b. Machine Control Block (Fig. 252)

As described above, FUCTL 20214's Machine Control Block is comprised of a Machine Control Word 1 (MCW1) and a Machine Control Word 0 (MCW0). MCW1 and MCW0 reside, respectively, in Registers MCW1 20290 and MCW0 20292. MCW1 and MCW0 described the current execution environment of FUCTL 20214's current microprogram, that is the microinstruction sequence currently being executed by JP 10114.

Referring to Fig. 252, diagramic representations of MCW0 and MCW1 are shown. As indicated therein, MCW0 and MCW1 may each contain, for example, 32 bits of information regarding current microprogram execution environment.

Referring to MCW0, MCW0 includes 6 execution environment subfields. Bits 0 to 3 inclusive contain a Top Of Stack Counter (TOSCNT) subfield which is a pointer to Current Frame of accelerated Microstack (MIS) 10368. TOSCNT field is initially set to point to Frame 1 of MIS 10368. Bits 4 to 7 inclusive comprise a Top of Stack -1 Counter (TOS-1CT) subfield which is a pointer to Previous Frame of accelerated MIS 10368, that is to the MIS 10368 frame proceeding that pointed by TOSCNT subfield. TOSICNT subfield is initially

set to Frame 0 of MIS 10368. Bits 8 to 11 inclusive comprise a Bottom of Stack Counter (BOSCNT) subfield which is a pointer to Bottom Frame of accelerated MIS 10368. BOSCNT subfield is initially set to point to Frame 1 of MIS 10368. TOSCNT, TOS-1CNT, and BOSCNT subfields of MCW0 may be read, written, incremented and decremented under microprogram control as frames are transferred between MIS 10368 and a SS 10336.

Bits 17 to 23 inclusive and Bits 24 to 31 inclusive of MCW0 comprise, respectively, Page Number Register (PNREG) and Repeat Counter (REPCTR) subfields which, together, comprise a microinstruction address pointing to a microinstruction currently being written into FUSITT 11012.

Bits 12 to 15 inclusive of MCW0 comprise an Egg Timer (EGGT) subfield which will be described further below with respect to TIMERS 20296. Bit 16 of MCW0 is not utilized in a present embodiment of CS 10110.

Referring to MCW1, MCW1 is comprised of four subfields. Of the 32 bits comprising MCW1, Bits 0 to 15 inclusive and Bits 24 and 25 are not utilized in a present embodiment of CS 10110. Bit 16 is comprised of a Condition Code (CC) subfield indicating results of certain test conditions in JP 10114. As previously described CC subfield is automatically saved and restored in RCWS 10358 RCW's.

Bits 17 to 19 inclusive of RCW1 comprise an Interrupt Mask (IM) subfield. The three bits of IM subfield are utilized to indicate a hierarchy of non-interruptible JP 10114 microinstruction control operating states. That is, a three bit code stored therein indicates relative power to interrupt between three otherwise noninterruptible JP 10114 operating states. Bits 20 to 23 inclusive comprise an Interrupt Request (IR) subfield which indicate Interrupt Request. These Interrupt Requests may include, for example, Egg Timer Overflow, Interval Timer Overflow, or Non-Fatal Memory Error, as have been previously described. Finally, Bits 26 to 31 inclusive comprise a Trace Trap Enable (TTR) subfield indicating which Trace Trap Events, previously described, are currently enabled. These enables may include Name Trace Enable, Logical Retrace Enable, Logical Write Trace Enable, SOP Trace Enable, Microinstruction Enable, and Microinstruction Break point Enable.

MCW0 and MCW1 has been described above as if residing in registers having individual, discrete existence, that is MCW1 20290 and MCW0 20292. In a present embodiment of CS 10110, MCW1 20290 and MCW0 20292 do not exist as a unified, discrete register structure but are instead comprised of individual registers having physical existence in other portions of FUCTL 20214. MCW1 20290 and MCW0 20292, and MCW1 and MCWO, have been so described to more distinctly represent the structure of information contained therein. In addition, this approach has been utilized to illustrate the manner by which current JP 10114 execution state may be controlled and monitored through JPD Bus 10142. As indicated in Fig. 202, MCW1 20290 and MCW0 20292 have outputs connected to JPD Bus 10142, thus allowing current execution state of JP 10114 to be read out of FUCTL 20214. Individual bits or subfields of MCW0 and MCW1 may, as previously described, be written by microinstruction control provided by FUSITT 11012. In a present physical embodiment of CS 10110, those registers of MCW0 20292 containing subfields TOSCNT, TOS-1CNT, and BOSCNT reside in RAG 20288. Those portions of MCW0 20292 containing subfield EGGT reside in TIMERS 20296. MCW0 20292 registers contain PNREG and REPCTR subfields are physically comprised of REPCTR 20280 and PNREG 20282. In MCW1 20290, CC subfield exists as output of FUCTL 20214 test circuits. Those MCW1 20290 registers containing IM, IR, and TTE subfields reside within EVENT 20284.

Having described FUCTL 20214 structure and operation as regards RCWS 10358, MCW1 20290 and MCW0 20292, FUCTL 20214, RAG 20288 will be described next below.

c.c.c. Register Address Generator 20228 (Fig. 253)

Referring to Fig. 253, a partial block diagram of RAG 20228, together with diagramic representation of GRF 10354, BIAS 20246 and RCWS 10358, is shown. As previously described, JP 10114 register and stack mechanisms include General Register File (GRF) 10354. BIAS 20246, and RCWS 10358. GRF 10354 is, in a present embodiment of CS 10110, a 256 word by 92 bit wide array of registers. GRF 10354 is divided horizontally to provide Global Registers (GRs) 10360 and Stack Registers (SRs) 10362, each of which contains 128 of GRF 10354's 256 registers. GRF 10354, that is both GRs 10360 and SRs 10362, is divided vertically into three vertical sections designated as AONGRF 20232, OFFGRF 20234, and LENGRF 20236. AONGRF 20232, OFFGRF 20234, and LENGRF 20236 are, respectively, 28 bits, 32 bits, and 32 bits wide. GRs 10360 is utilized as an array of 128 individual registers, each register containing one 92 bit word. SRs 10362 is structured and utilized as an array of 16 register frames wherein each frame contains eight registers and each register contains one 92 bit wide word. Eight of SR 10362's frames are utilized as Microstack (MIS) 10362 and the remaining eight of SR 10362's frames are utilized as Monitor Stack (MOS) 10370. For addressing purposes only, as described further below, GRs 10360 is regarded as being structured in the same manner as SRs 10362, that is as 16 frames of eight registers each.

BIAS 20246, as previously described, is a register array within BIAS 20246. BIAS 20246 contains 128 six bit wide registers, or words, and operates in parallel with and is addressed in parallel with SR 10362 portion of GRF 10354. RCWS 10358 is, as previously described, an array of 16 registers, or words, wherein each register contains one 32 bit RCW. RCWS 10358 is structured and operates in parallel with SRs 10362 with each RCWS 10358 register corresponding to a SR 10362 frame of eight registers. As described below, RCWS 10358 is addressed in parallel with SR 10362's frames.

Source and Destination Register Addresses (SDAR) for selecting a GRF 10354 register to be,

respectively, read from or written to are provided by RAG 20288. As described above BIAS 20246 operates and is addressed in parallel with SR 10362 portion of GRF 10354, that is parallel with SRs 10362. BIAS 20246 registers are thereby connected to and in parallel with address inputs of SRs 10362 and are addressed concurrently with GRs 10360. Registers RCWS 10358 also operate and are addressed in parallel with SRs 10362. Address inputs of RCWS 10358's registers are thereby connected in parallel with address inputs of SR 10362's registers.

RAG 20288's address inputs to GRF 10354, and to BIAS 20246 and RCWS 10358, may select registers therein to be either source registers, that is registers providing data, or destination registers, that is registers receiving data. RAG 20288's address outputs are designated as output Source and Destination Register Address (SDADR) of RAG 20288. RAG 20288's SDADR output is connected to address input of register comprising GRF 10354, BIAS 20246, and RCWS 10358. As described above, SRs 10362 are structured as 16 frames of 8 registers per frame and RCWS 10358 is structured as a corresponding 16 frames of one register per frame. GRF 10354 and BIAS 20246 are structured and utilized as single registers but, for addressing purposes, are regarded as being comprised of 16 frames of 8 registers per frame. Each SDADR output of RAG 20288 is an 8 bit word wherein the most significant bit indicates whether the addressed register, either a Source or a Destination Register, reside in GRs 10360 or within SRs 10362, BIAS 20246, and RCWS 10358. The four next most significant bits comprise a frame select field for selecting one of 16 frames within GRs 10360 or within SRs 10362, BIAS 20246, and RCWS 10358. The three least significant bits comprise a register select field selecting a particular register within the frame selected by frame select field.

Within a single system clock cycle, SDADR output of RAG 20288 may select a source register and data may be read from that source register, or SDADR output may select a destination register and data may be written into that destination register. As previously described, each JP 10114 microinstruction requires a minimum of two-system clock cycles for execution, that is at first clock cycle in State MO and a second clock cycle in State M1. During a single microinstruction therefore, a source register may be selected and data read from that source register, and a destination register selected and data written into that destination register. Certain operations, however, may require more than one microinstruction for execution. For example, a read-modify-write operation wherein data is read from a particular register, modified, and written back into that register may require two or more microinstructions for execution.

Referring first to RAG 20288 structure, RAG 20288 includes MISPR 10356. MISPR 10356 includes Top Of Stack Counter (TOSCNT) 25310, Top Of Stack-1 Counter (TOS-1CNT) 25312, and Bottom Of Stack Counter (BOSCNT) 25314. Contents of TOSCNT 25310, TOS-1CNT 25312 and BOSCNT 25314 are respectively, pointers to Current, Previous, and Bottom frames of SRs 10362, that is, to MIS 10368. As will be described below, these pointers are also utilized to address MOS 10370. TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314 are each four bit binary counters comprised, for example, of SN74S163s.

Data inputs of TOSCNT 25310 to BOSCNT 25314 are connected from JPD Bus 10142. Control inputs of TOSCNT 15310 to BOSCNT 25314 areconnected from microinstruction control outputs of FUSITT 11012. Data outputs of TOSCNT 25310 to BSOCNT 25314 are connected to data inputs of Source Register Address Multiplexer (SRCADR) 25316 and to data inputs of Destination Register Address Multiplexer (DSTADR) 25318. Data outputs of TOSCNT 25310 and BOSCNT 25314 are connected to inputs of Stack Event Monitor Logic (SEM) 25320.

Source and destination frame addresses are selected, as will be described further below, by SRCADR 25316 and DSTADR 25318 respectively. In addition to data inputs from TOSCNT 25310 and BOSCNT 25314, data inputs of SRCADR 25316 and DSTADR 25318 are connected from microinstruction word CONEXT subfield output from FUSITT 11012. Control inputs of SRCADR 25316 and DSTADR 25318 are connected from, respectively, microinstruction word RS and RD subfield outputs from FUSITT 11012. Source Frame Address Field (SRCFADR) output of SRCADR 25316 and Destination Frame Address Field (DSTFADR) output of DSTADR 25318 are connected to inputs of Source and Destination Register Address Multiplexer (SDADRMUX) 25322. SRCFADR and DSTFADR comprise frame select fields of RAG 20288, SDADR outputs for, respectively, source and destination registers.

In addition to SRCFADR and DSTFADR outputs of ADRSRC 25316 and DSTADR 25318, SDADRMUX 25322 receives microinstruction word SRC and DST subfield inputs from microinstruction outputs of FUSITT 11012. As previously described, SRC subfield is a 3 bit number designating a source register, that is, a source register within a frame selected by SRCFADR. DST is similarly a 3 bit number selecting a destination register within a frame indicated by DSTFADR. SRC subfield input to SDADRMUX 25322 is concatenated with SRCADR 25316 to respectively comprise, as described above, register and frame fields of a source register SDADR output of SDADRMUX 25322. Similarly, DST subfield is concatenated with DSTFDADR output of DSTADR 25318 to comprise, respectively, register and frame subfields of a destination register SDADR output of SDADRMUX 25322. Selection between source and destination register address inputs to SDADRMUX 25322, to generate a corresponding source of destination register SDADR output of SDADRMUX 25322 is controlled by microinstruction control inputs (not shown for clarity of presentation) connected to control inputs of SDADRMUX 25322. RDWS 25324 is a PROM decoding MD field from microinstruction words during reads from MEM 10112 and provides register select field of destination register address and selects one of the pointers as frame select field.

An Event output of SEM 25320 is connected to an input of EVENT 20284, previously described.

SRCADR 25316, DSTADR 25318, and SDADRMUX 25322, as will be described further below, operate as multiplexers and may be comprised, for example, of SN74S153s.

Having described structure and organization of GRF 10354, BIAS 20246, and RCWS 10358, and structure of RAG 20288, operation of RAG 20288 to generate Source of Destination Register Address outputs SDADR will be described next below. Addressing of JP 10114's stack mechanism, comprising SRs 10362 and RCWS 10358, will be described first, followed by addressing of GRs 10360 and BIAS 20246.

SR 10362 portion of GRF 10354, RCWS 10358, and BIAS 20246 are addressed by Current, Previous, and Bottom Frame Pointers contained, respectively, in TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314. Current, Previous, and Bottom Pointers comprise frame select fields of SDADRMUX 25322. As previously described, Current, Previous and Bottom Pointer outputs of TOSCNT 25310 to BOSCNT 25314 are provided as inputs of SRCADR 25316 and DSTADR 25318. Microinstruction word RS subfield to control input of SRCADR 25316 selects either Current, Previous or Bottom Pointer input of SRCADR 25316 to comprise SRCFADR output of SRCADR 25316, that is to be frame select field of source register address. Similarly, microinstruction word RD subfield to control input of DSTADR 25318 concurrently selects either Current, Previous, or Bottom Pointer inputs of DSTADR 25318 to comprise DSTADR 25318's concurrently selects either Current, Previous, or Bottom Pointer inputs of DSTADR 25318 to comprise DSTADR 25318s DSTFADR output, that is frame select field of destination register address. As described above, SRCFADR and DSTFADR are provided as inputs to SDADRMUX 25322. Microinstruction word SRC and DST subfield inputs to SDADRMUX 25322 concurrently determine, respectively, source and destination registers within source and destination frames specified by SRCFADR and DSTFADR. SDADRMUX 25322 then, operating under microinstruction control, selects either SRCFADR and SRC to comprise SDADR output to SR 10362 as a source register address or selects DSTFADR and DST as SDADR output specifying a destination register address. By microinstruction control of SRCADR 25316, DSTADR 25318, and SDADRMUX 25322, a CS 10110 microprogram may select a source frame and register within SR 10362 and simultaneously specify a possible different destination frame and register within SR 10362. All possible combinations of source frame and register and destination frame and register in GRs 10360, SRs 10362, BIAS 20246, and RCWS 10358 are valid.

Control of SRCADR 25316, DSTADR 25318, and SDADRMUX 25322 in addressing SR 10362 portion of GRF 10354, and RCWS 10358, is controlled, in part, by current CS 10110 state. Pertinent CS 10110 operating states, previously described, are State MI and State RW. When CS 10110 is in neither State RW nor State M1, SR 10362 is addressed through SRCADR 25316 and microinstruction word SRC subfield, that is SR 10362 and RCWS 10358 are provided with source register addresses when CS 10110 is in neither RW nor M1 States. When CS 10110 enters State M1, SR 10362 and RCWS 10358 is addressed through DSTADR 25318 and by microinstruction word DST subfield. That is, SR 10362 and RCWS 10358 are provided with destination register addresses during State M1. Similarly, SR 10362 and RCWS 10358 are provided with destination register addresses when CS 10110 is operating in State RW, that is when data is being read from MEM 10112 and written into SR 10362 or RCWS 10358. In this case, however, low order 3 bits of destination register address, that is register select field, are provided by RDS 25324, which decodes microinstruction word subfield MD (Memory Destination). RDS 25324 also provides a control input that DSTADR 25318 to select one of Current, Previous, or Bottom pointers from MISPR 10356 to comprise frame select field of destination register address.

As stated above, frame select field of source and destination register addresses are provided from TOSCNT 25310, TOS-1CNT 25312, and BOSCNT 25314. As described above, the most significant bit of source and destination register address are forced to logic 1 or logic 0, depending upon whether GR 10360 or SR 10362, BIAS 20246, and RCWS 10358 are being addressed. Contents of TOSCNT 25310 to BOSCNT 25314, that is Current, Previous, and Bottom Pointers, are controlled by microinstruction control outputs of FUSITT 11012. Current and Previous Pointers change as stacks are "pushed" or "popped" to and from MIS 10368 as JP 10114 performs, respectively, calls and returns. Similarly, Current, Previous and Bottom Pointers will be incremented or decremented as MIS 10368 frames are transferred between MIS 10368 and MEM 10112, as previously described with respect to CS 10110's Stack Mechanisms.

Referring first to Current and Previous Pointer operation, Current and Previous Pointers in TOSCNT 25310 and TOS-1CNT 25312 are initially set, respectively, to point to Frames 1 and 0 of MIS 10368 by being loaded from JPD Bus 10142. TOSCNT 25310 and TOS-1CNT 25312 are enabled to count when two conditions are met. First condition is dependent upon current operating state of CS 10110. TOSCNT 25310 and TOS-1CNT 25312 will be enabled to count during last system clock cycle of CS 10110 operating States M1 or AB. Second condition is dependant upon whether JP 10114 is to execute a call or return. TOSCNT 25310 and TOS-1CNT 25312 may be enabled to count if a current microinstruction indicates JP 10114 is to execute a call or return, or if CS 10110 is exiting State AB as exit from State AB is an implied call operation. Both a call and an implied call, that is exit from State AB, will cause TOSCNT 25310 and TOS-1CNT 25312 to be incremented. A return will cause TOSCNT 25310 and TOS1CNT 25312 to be decremented.

Referring to BOSCNT 25314, Bottom Frame Pointer is initially loaded from JPD Bus 10142 to point to MIS 10368 Frame 1. Again, incrementing or decrementing of BOSCNT 25314 is dependant upon CS 10110 operating state and operation to be performed. BOSCNT 25314 is enabled to count upon exiting from State M1. In addition, DEVCMD subfield of a current microinstruction word must indicate that BOSCNT 25314 is to be incremented or decremented. BOSCNT 25314 will be incremented or decremented upon exit from

State M1 as indicated by microinstruction word DEVCMD subfield.

SEM 25320 monitors relative values of Current and Bottom Pointers residing in TOSCNT 25310 and BOSCNT 25314 and provides outputs to EVENT 20284 for purposes of controlling operation of MI 10368 and MOS 10370. SEM 25320 is comprised of a Read Only Memory, for example 93S427s, receiving Current and Bottom Pointers as inputs. SEM 25320 detects 3 Events occurring in operation of TOSCNT 25310 and BOSCNT 25314, and thus in operation of MIS 10368 and MOS 10370. First, SEM 25320 detects an MIS 10368 Stack Overflow. This Event is indicated if the present value of Current Frame Pointer is greater than 8 larger than the present value of Bottom Frame Pointer. Second, SEM 25320 detects when MIS 10368 contains only one frame of information. This event is indicated if the value of Current Frame Pointer is equal to the value of Bottom Frame Pointer. In this case, the previous frame of MIS 10368 resides in MEM 10112 and must be fetched from MEM 10112 before a reference to the previous stack frame may be made. Third, SEM 25320 detects when MIS 10368 and MOS 10370 are full. This Event is indicated if the present value of Current Frame Pointer is 16 larger than the present value of Bottom Frame Pointer. When this Event occurs, any further attempt to write a frame onto MIS 10368 or MOS 10370 will result in a MOS 10370 Stack Overflow. EVENT 20284 responds to these Events indicated by SEM 25320 by initiating execution of an appropriate Event Handling microinstruction sequence, as previously decribed. It should be noted that MIS 10368 and MOS 10370 are addressed in the same manner, that is through use of Current, Previous and Bottom Frame Pointers and certain microinstruction word subfields. Primary difference between operation of MIS 10368 and MOS 10370 is in the manner in which stack overflows are handled. In the case of MIS 10368, stack frames are transferred between MIS 10368 and MEM 10112 so that MIS 10368 is effectively a bottomless stack. MOS 10370, however, contains a maximum of 8 stack frames, in a present embodiment of CS 10110, so that no more than eight Events may be pushed onto MOS 10370 at a given time.

GR 10360 is addressed in a manner similar to SR 10362, BIAS 20246, and RCWS 10358, that is through ADRSRC 25316, DSTADR 25318, and SDADRMUX 25322. Again, register select fields of source and destination register addresses are provided by microinstruction word SRC and DST subfields. Frame select field of source and destination register addresses is, however, specified by microinstruction word CONEXT subfield. In this case, microinstruction word RS and RD subfields specify that frame select fields of source and destination register addresses are to be provided by CONEXT subfield. Accordingly, ADRSRC 25316 and DSTADR 25318 provide CONEXT subfield as SRCFADR and DSTFADR inputs to SDADRMUX 25322.

Having described structure and operation of RAG 20288, TIMERS 20296 will be described next below.

Referring to Fig. 254, a partial block diagram of TIMERS 20296 is shown. As indicated therein, TIMERS 20296 includes Interval Timer (INTTMR) 25410, Egg Timer (EGGTMR) 25412, and Egg Timer Clock Enable Gate (EGENB) 25416.

d.d.d. Timers 20296 (Fig. 254)

Referring first to INTTMR 25410, a primary function of INTTMR 25410 is to maintain CS 10110 architectural time as previously described with reference to Fig. 106A and previous descriptions of CS 10110 UID addressing. As described therein, a portion of all UID addresses generated by all CS 10110 systems is an Object Serial Number (OSN) field. OSN field uniquely defines each object created by operation of or for use in a particular CS 10110. OSN field of an object's UID is, in a particular CS 10110, generated by determining time of creation of that object relative to an arbitrary historic starting time common to all CS 10110 systems. That time is maintained within a MEM 10112 storage space, or address location, but is measured by operation of INTTMR 25410.

INTTMR 25410 is a 28 bit counter clocked by a 110 Nano-Second Clock (110NSCLK) input and is enabled to count by a one MHZ Clock Enable input (CLK1MHZENB). INTTMR 25410 may thereby be clocked at a one MHZ rate to measure one microsecond intervals. Maximum time interval which may be measured by INTTMR 25410 is thereby 268.435 seconds.

As indicated in Fig. 254, INTTMR 25410 may be loaded from and read to JPD Bus 10142. In normal operation, the MEM 10112 location containing architectural time for a particular CS 10110 will be loaded with current architectural time at time of start up of that particular CS 10110. INTTMR 25410 will concurrently be loaded with all zeros. Thereafter, INTTMR 25410 will be clocked at one microsecond intervals. Periodically, when INTTMR 25410 overflows, architectural time stored in MEM 10112 will be accordingly updated. At any time, therefore, current architectural time may be determined, down to a one microsecond increment, by reading architectural time from the previous updated architectural time stored in MEM 10112 and elapsed interval since last update of architectural time from INTTMR 25410. In the event of a failure of CS 10110, architectural time in MEM 10112 and INTTMR 25410 may be saved in MEM 10112 by reading elapsed intervals since last architectural time update. When normal CS 10110 operation resumes, INTTMR 25410 may be reloaded with a count reflecting current architectural time. As indicated in Fig. 254, INTTMR 25410 is loaded from JPD Bus 10142 when INTTMR 25410 is enabled by a Load Enable input (LDE) provided from DP 10118.

Referring to EGGTMR 25412, certain CS 10110 Events, in particular Asychronous Events previously described with reference to EVENT 20284, are received or acknowledged by EVENT 20284 only at conclusion of State M1 of first microinstruction of an SOP. As certain CS 10110 microinstructions have long execution times, these Asynchronous Events may be subjected to an extended latency, or waiting, interval

before being serviced. EGGTMR 25412, in effect, measures latency time of pending Asychronous Events and provides an output to EVENT 20284 if a predetermined maximum latency time is exceeded.

As indicated in Fig. 254, EGGTMR 25412 is clocked by a 110 Nano-Second Clock input (110NSCLK). EGGTMR 25412 is initially set to zero by load input (LDZRO) at end of State MI of the first microinstruction of each SOP executed by CS 10110, or when specifically instructed so by DEVCMD subfield of a microinstruction word. EGGTMR 25412 is incremented when enabled by Clock Enable (CLKENB) input from EGGENB 25416. There are two conditions necessary for EGGTMR 25412 to be incremented. First condition is occurrence of an Asychronous Event, which is indicated by input ASYEVNT to EGGENB 25416 from EVENT 20284 Second condition is that 16 or more microseconds have elapsed since last increment of EGGTMR 25412. This interval is measured by an output from fourth bit of INTTMR 25410 which, as shown in Fig. 254, is connected to an input of EGGENB 25416. EGGTMR 25412 is a four bit counter and will thereby overflow and generate output OVRFLW to EVENT 20284 256 microseconds after beginning of an SOP if an Asychronous Event has occurred and if at least 16 microseconds have elapsed since start of that SOP EGGTMR 25412 thereby insures a maximum service latency of 256 microseconds for Asychronous Events.

e.e.e. Fetch Unit 10120 Interface to Execute Unit 10122

Finally, as previously described FU 10120's interface to EU 10122 is primarily comprised of EUDIS Bus 20206, for providing EUDPs to EU 10122's EUSITT, and FUINT 20298. Operation of EUSDT 20266 and EUDIS Bus 20206 has been previously described and will be described further in a following description of EU 10122. FUINT 20298 is primarily concerned with generating Event Requests for conditions signalled from EU 10122 so that these Events may be serviced. In this regard, FUINT 20298 is primarily comprised of gates receiving Event Requests from EU 10122 and providing corresponding outputs to EVENT 20284. Another interface function performed by FUINT 20298 is generation of a "transfer complete" signal generated by FU 10122 and provided to EU 10122 to assert that a EU 10122 result read from EU 10122 to FU 10120 has been received. This transfer complete signal indicates to EU 10122 that EU 10122's result register, described in a following description of EU 10122, is available for further use by EU 10122. This transfer complete signal is generated by an output of FUSITT 11012 as part of microinstruction sequences for transferring data from EU 10122 to FU 10120 or MEM 10112.

Having described structure and operation of FU 10120, including DESP 20210, MEMINT 20212, and FUCTL 20214, the structure and operation of EU 10122 will be described next below.

C. Execute Unit 10122 (Figs. 203, 255—268)

As previously described, EU 10122 is an arithmetic processor capable of executing integer, packed and unpacked decimal, and single and double precision floating point arithmetic operations. A primary function of EU 10122 is to relieve FU 10120 of certain arithmetic operations, thus enhancing efficiency of CS 10110.

Transfer of operands from MEM 10112 to EU 10122 is controlled by FU 10120, as is transfer of results of arithmetic operations from EU 10122 to FU 10120 or MEM 10112. In addition, EU 10122 operations are initiated by FU 10120 by EU 10122 Dispatch Pointers invited to EU 10122 by EUSDT 20266. EU 10122 Dispatch Pointers may initiate both arithmetic operations required for execution of SINs and certain EU 10122 operations assisting in handling of CS 10110 events. As previously described, EU 10122 Dispatch Pointers are translated into sequences of microinstructions for controlling EU 10122 by EU 10122's EUSITT which is similar in structure and operation to FUSITT 11012. As will be described further below, EU 10122 includes a command queue for receiving and storing sequences of EU 10122 Dispatch Pointers from FU 10120. In addition, EU 10122 includes a general register file, or scratch pad memory, similar to GRF 10354. EU 10122's general register file is utilized, in part, in EU 10122 Stack Mechanisms similar to FU 10120's SR's 10362.

Referring to Fig. 203, a partial block diagram of EU 10122 is shown. EU 10122's general structure and operation will be described first with reference to Fig. 203. Then EU 10122's structure and operation will be described in further detail with aid of subsequent figures which will be presented as required.

As indicated in Fig. 203, major elements of EU 10122 include Execute Unit Control Logic (EUCL) 20310, Execute Unit IO Buffer (EUIO) 20312, Multiplier Logic (MULT) 20314, Exponent Logic (EXP) 20316, Multiplier Control Logic (MULTCNTL) 20318, and Test and Interface Logic (TSTINT) 20320. EUCL 20310 receives Execute Unit Dispatch Pointers (EUDP's) from EUSDT 20266 and provides corresponding sequences of microinstructions to control operation of EU 10122.

EUIO 20312 receives operands, or data, from MEM 10112, translates those operands into certain formats most efficiently used by EU 10122. EUIO 20312 receives results of EU 10122's operations and translates those results into formats to be returned to MEM 10112 or FU 10120, and presents those results to MEM 10112 and FU 10120.

MULT 20314 and EXP 20316 are arithmetic units for performing arithmetic manipulations of EU 10122 operations. In particular, EXP 20316 performs operations with respect to exponent fields of single and double precision floating point operations. MULT 20314 performs arithmetic manipulations with respect to mantissa fields of single and double precision floating point operations, and arithmetic operations with regard to integer and packed decimal operations. MULTCNTL 20318 controls and coordinates operation of

MULT 20314 and EXP 20316 and prealignment and normalization of mantissa and exponent fields in floating point operations. Finally, TSTINT 20320 performs certain test operations with regard to EU 10122's operations, and is the interface between EU 10122 and FU 10120.

a. General Structure of EU 10122

1. Execute Unit I/O 20312

Referring first to EUIO 20312, EUIO 20312 includes Operand Buffer (OPB) 20322, Final Result Output Multiplexer (FROM) 20324, and Exponent Output Multiplexer (EXOM) 20326. OPB 20322 has first and second inputs connected, respectively, from MOD Bus 10144 and JPD Bus 10142. OPB 20322 has a first output connected to a first input of Multiplier Input Multiplexer (MULTIM) 20328 and MULT 20314. A second output of OPB 20322 is connected to first inputs of Inputs Selector A (INSELA) 20330 and Exponent Execute Unit General Register File Input Multiplexer (EXRM) 20332 in EXP 20316.

FROM 20324 has an output connected to JPD Bus 10142. A first input of FROM 20324 is connected from output of Multiplier Execute in General Register File Input Multiplexer (MULTRM) 20334 and MULT 20314. A second input of FROM 20324 is connected from output of Final Result Register (RFR) 20336 of MULT 20314. EXOM 20326 has an output connected to JPD Bus 10142. EXOM 20326 is a first input connected from output of Scale Register (SCALER) 20338 of EXP 20316. EXOM 20326 has second and third inputs connected from outputs of, respectively, Next Address Generator (NAG) 20340 and Command Queue (COMQ) 20342 of EUCL 20310.

2. Execute Unit Control Logic 20310

Referring to EUCL 20310, EUCL 20310 includes NAG 20340, COMQ 20342, Execute Unit S Interpreter Table (EUSITT) 20344, and Microinstruction Control Register and Decode Logic (mCRD) 20346. COMQ 20342 has an input connected from EUDIS Bus 20206 for receiving SDPs from EUSDT 20266. COMQ 20342 has, as described above, a first output connected to a third input of EXOM 20326, and has a second output connected to an input of NAG 20340. NAG 20340 has, as described above, a first output connected to second input of EXOM 20326. NAG 20340 has a second output connected to a first input of EUSITT 20344. As previously described, EUSITT 20344 corresponds to FUSITT 11012 and stores sequences of microinstructions for controlling operation of EU 10122 in response to EU 10122 Dispatch Pointers from FU 10120. EUSITT 20344 has a second input connected from JPD Bus 10142 and has an output connected to input of mCRD 20346. mCRD 20346 includes a register and logic for receiving and decoding microinstructions provided by EUSITT 20344. In addition to an input from EUSITT 20344, mCRD 20346 has first outputs providing decoded microinstruction control signals to all parts of EU 10122. mCRD 20346 also has a second output connected to a first input of Input Selecter B (INSELB) 20348 and EXP 20316.

3. Multiplexer Logic 20314

Referring to MULT 20314, MULT 20314 includes two parallel arithmetic operation paths for performing addition, subtraction, multiplication, and division operations on packed decimal numbers, integer numbers, and mantissa portions of single and double precision floating point numbers. MULT 20314 also includes a related portion of EU 10122's general register file, a memory for storing constants used in arithmetic operations, and certain input data selection circuits. That portion of EU 10122's GRF residing in MULT 20314 is comprised of Multiplier Register File (MULTRF) 20350. Output of MULTRF 20350 is connected to a second input of MULTIM 20328. A first input of MULTRF 20350 is connected from output of RFR 20336 and a second input of MULTRF 20350 is connected from output of MULTRM 20334. First and second inputs of MULTRM 20334 are in turn connected, respectively, from output of RFR 20336 and from output of Container Size Logic (CONSIZE) 20352 of TSTINT 20320.

MULTIM 20328 selects the data inputs to MULT 20314's arithmetic circuits and has, as previously described, first and second inputs connected respectively from first output of OPB 20322 and from output of MULTRF 20350. Output of MULTIM 20328 is connected through Multiplier (MULT) Bus 20354 to input of Multiplier Quotient Register (MQR) 20356 and to input of Nibble Shifter (NIBSHF) 20358. Another input to MQR 20356 and NIBSHF 20358 is provided by Constant Store (CONST) 20360. CONST 20360 is a memory for storing constant values used n MULT 20314 operations. Output of CONST 20360 is connected to MULT Bus 20354. MULT 20314's arithmetic circuits may thereby be provided with inputs from OPB 20322, MULTRF 20350, and CONST 20360.

MULT 20314's arithmetic circuitry is comprised of two, parallel arithmetic operation paths having, as common inputs, outputs of MULTIM 20328 and CONST 20360. Common termination of these parallel arithmetic operation paths is Final Register Shifter (FRS) 20362. A first arithmetic operation path is provided through NIBSHF 20358, whose input is connected from MULT Bus 20354. NIBSHF 20358's output is connected to a first input of FRS 20362 and a control input of NRBSHF 20358 is connected from an output of Multiplier Control Logic (MULTCNT) 20364 and MULTCNTL 20318.

MULT 20314's second arithmetic operation path is provided through MQR 20356. As described above, MQR 20356's input is connected from MULT Bus 20354. MQR 29356's output is connected to first and

second inputs of Times 1 And Times 2 Multiply Shifter (MULTSHFT12) 20366 and Times 4 And Times 8 Multiply Shifter (MULTSHFT48) 20368. Outputs of MULTSHFT12 and MULTSHFT8 are connected, respectively, to first and second inputs of First Multiplier Arithmetic and Logic Unit (MULTALU1) 20370. MULTALU1 20370's output is connected to input of Multiplier Working Register (MWR) 20372. Output of MWR 20372 is connected to a first input of Second Multiplier Arithmetic and Logic Unit (MULTALU2) 20374. A second input of MULTALU2 20374 is connected from output of RFR 20336. Output of MULTALU2 is connected to a second input of FRS 20362. As described above, first input of FRS 20362 is connected from output of NIBSHF 20368. Output of FRS 20362 is connected to input of RFR 20336.

As described above, output of RFR 20336 is connected to second input of MULTALU2 20374, to first input of MULTRF 20350, to first input of MULTRM 20334, and to second input of FROM 20324. Output of RFR 20336 is also connected to input of Leading Zero Detector (LZD) 20376 of MULTCNTL 20318, and to inputs of Exception Logic (ECPT) 20378, CONSIZE 20352, and TSTINT 20320.

4. Exponent Logic 20316

Referring to EXP 20316, as previously described EXP 20316 performs certain operations with respect to exponent fields of single and double precision floating point number in EU 10122 floating point operations. EXP 20316 includes a second portion of EU 10122's general register file, shown herein as Exponent Register File (EXPRF) 20380. Although indicated as individual register files, MULTRF 20350 and EXPRF 20380 comprise, as in GRF 10354, a unitary register file structure with common, parallel addressing of corresponding registers therein.

Output of EXPRF 20380 is connected to a second input of INSELA 20330. A first input of EXPRF 20380 is connected from output of EXRM 20332. As previously described, a first input of EXRM 20332 is connected from second output of OPB 20322 through EXPQ Bus 20325. A second input of EXRM 20332 is connected from output Scale Register (SCALER) 20338. A second input of EXPRF 20380 is connected from output of Sign Logic (SIGN) 20382. Input of SIGN 20382 is connected from second output of SCALER 20338.

INSELA 20330, INSELB 20348, Exponent ALU (EXPALU) 20384 and SCALER 20338 comprise EXP 20316's arithmetic circuitry for manipulating exponent fields of floating point numbers. INSELA 20330 and INSELB 20348 select, respectively, first and second inputs to EXPALU 20384. As previously described, a first input of INSELA 20330 is connected from second output of OPB 20322 through EXPQ Bus 20325. Second input of INSELA 20330 is connected from output of EXPRF 20380. Output of INSELA 20330 is connected to first input of EXPALU 20384. First input of INSELB 20348 is, as previously described, connected from a second output of mCRD 20346. Second input of INSELB 20348 is connected from output of OPB 20322 through EXPQ Bus 20325. Third input of INSELB 20348 is connected from output of SCALER 20338 and fourth input of INSELB 20348 is connected from output of LZD 20376. Output of INSELB 20348 is connected to second input of EXPALU 20348. Output of EXPALU 20348 is connected to input of SCALER 20338.

As previously described, second output of SCALER 20338 is connected with input of SIGN 20382 and first output is connected to second input of EXRM 20332 and to third input of INSELB 20348. First output of SCALER 20338 is also connected to EXPQ Bus 20325, to first input of EXOM 20326, and to a second input of MULTCNT 20364.

5. Multiplier Control 20318

As previously described, MULTCNTL 20318 provides certain control signals and information for controlling and coordinating operation of EXP 20316 and MULT 20314 in performing arithmetic operations on floating point numbers. MULTCNTL 20318 includes LZD 20376 and MULTCNT 20364. Input of LZD 20376 is connected from output of RFR 20336 through FR Bus 20337. Output of LZD 20376 are connected to a second input of MULTCNT 20364 and to fourth input of INSELB 20348. A second input of MULTCNT 20364 is connected from output of SCALER 20338. As previously described, control output of MULTCNT 20364 is connected to control inputs of NIBSHF 20358.

6. Test and Interface Logic 20320

Finally, TSTINT 20320 includes ECPT 20378, CONSIZE 20352, and Testing Condition Logic (TSTCON) 20386. Input of ECPT 20378 and first input of CONSIZE 20352 are connected from output of RFR 20336 through FR Bus 20337. A second input of CONSIZE 20352 is connected from LENGTH Bus 20226. An output of CONSIZE 20352 is connected, together with other inputs from EU 10122 (not shown for clarity of presentation) to TSTCON 20386. Output of TSTCON 20386 (not shown for clarity of presentation) are connected to NAG 20340. TSTCON 20386 and ECPT 20378 have outputs to and inputs from FU 10120's FUINT 20298.

Having described the overall structure of EU 10122 above, operation of EU 10122 will be described next below with aid of further diagrams which will be introduced as required. Finally, operation of TSTINT 20320 will be described, including a description of the detailed control signal interface between EU 10122 and FU 10120 through TSTINT 20320 and FUINT 20298. In addition to defining the interface between EU 10122 and FU 10120, certain features of EU 10122 operation will be described wherein those operations are executed

in cooperation with MEM 10112 and FU 10120. For example, EU 10122's Stack Mechanisms, comprising in part portions of MULTRF 20350 and EXPRF 20380, resides partly in MEM 10112 so that operation of EU 10122's Stack Mechanisms requires cooperative operations by EU 10122, MEM 10112 and FU 10120.

b. Execute Unit 10122 Operation (Fig. 255)

1. Execute Unit Control Logic 20310 (Fig. 255)

Referring to Fig. 255, a more detailed block diagram of EUCL 20310 is shown. As described above, EUCL 20310 receives EU 10122 Dispatch Pointers through EUDIS Bus 20206 from EUSDT 20266 and FUCTL 20214. EU 10122 Dispatch Pointers select certain EU 10122 microinstruction sequences for executing EU 10122 arithmetic operations as required to execute user's programs, that is SOPs, and to assist in handling JP 10114 Events. As described above, major elements of EUCL 20310 include COMQ 20342, EUSITT 20344, mCRD 20346, and NAG 20340.

a.a. Command Queue 20342

Inputs of COMQ 20342 are connected from EUDIS Bus 20206 to receive and store EU 10122 Dispatch pointers provided from EUSDT 20266. Each such EU 10122 Dispatch Pointer is comprised of two information fields. A first information field contains a 10 bit starting address of a corresponding sequence of microinstructions residing in EUSITT 20344. Second field of each EU 10122 Dispatch Pointer is a 6 bit field containing certain control information, such as information identifying data format of corresponding operands to be operated upon. In this case unit dispatch pointer control field bits specify whether operands to be operated upon comprise signed or unsigned integer, packed or unpacked decimal, or single or double precision floating point numbers.

COMQ 20342 is comprised of two one word wide by two word deep register files. A first of these register fields is comprised of SOP Command Queue Control Store (CQCS) 25510 and SOP Command Queue Address Store (CQAS) 25512. Together, CQCS 25510 and CQAS 25512 comprise a one word wide by two word deep register file for receiving and storing EU 10122 Dispatch Pointers corresponding to SOPs, that is Dispatch Pointers for initiating EU 10122 operations directly concerned with executing a user's program. Address fields of these SOPs are received in CQAS 25512, while control fields are received and stored in CQCS 25510. COMQ 20342 is thereby capable of receiving and storing up to two sequential EU 10122 Dispatch Pointers corresponding to user program SOPs These SOP derived Dispatch Pointers are executed in the order received from FU 10120. EU 10122 is thereby capable of receiving and storing one currently executing SOP Dispatch Pointer and one pending SOP Dispatch Pointer. Further SOP Dispatch Pointers may be read into COMQ 20342 as previous SOPs are executed.

b.b. Command Queue Event Control Store 25514 and Command Queue Event Address Control Store 25516

Command Queue Event Control Store (CQCE) 25514 and command Queue Event Address Control Store (CQAE) 25516 are similar in function and operation to, respectively, CQCS 25510 ad CQAS 25512. CQCE 25514 and CQAE 25516 receive and store, however, EU 10122 Dispatch Pointers initiating EU 10122 operations requested by FU 10120 as required to handle JP 10114 Events. Again, CQCE 25514 and CQAE 25516 comprise a one word wide by two word deep register file. CQAE 25516 receives and stores address fields of Event Dispatch Pointers, while CQCE 25514 receives and stores corresponding control fields of Event Dispatch Pointers. Again, COMQ 20342 is capable of receiving and storing up to two sequential Event Dispatch Pointers at a time.

As indicated in Fig. 255, outputs of CQAS 25512 and CQAE 25516, that is address fields of EU 10122 Dispatch Pointers are provided as inputs to Select Case Multiplexer (SCASE) 25518 and Starting Address Select Multiplexer (SAS) 25520 and NAG 20340, which will be described further below. Control field outputs of CQCS 25510 and CQCE 25514 are provided as inputs to OPB 20322, described further below.

c.c. Execute Unit S-Interpreter Table 20344

Referring to EUSITT 20344, as described above EUSITT 20344 is a memory for storing sequences of microinstructions for controlling operation of EU 10122 in response to EU 10122 Dispatch Pointers received from FU 10120. These microinstruction sequences may, in general, direct operation of EU 10122 to execute arithmetic operations in response to SOPs of user's programs, or aid direct execution of EU 10122 operations required to service JP 10114 Events. EUSITT 20344 may be, for example, a 60 bit wide by 1,280 word long memory structured as pages of 128 words per page. A portion of EUSITT 20344's pages may be contained in Read Only Memory, for example for storing sequence of microinstructions for handling JP 10114 Events. Remaining portions of EUSITT 20344 may be constructed of Random Access Memory, for example for storing sequences of microinstructions for executing EU 10122 operations in response to user program SOPs. This structure allows EU 10122 microinstruction sequences concerned with operation of JP 10114's internal mechanisms, for example handling of JP 10114 Events, to be effectively permanently

stored in EUSITT 20344. That portion of EUSITT 20344 constructed of Random Access Memory may be used to store sequences of microinstructions for executing SOPs. These Random Access Memories may be used as writable control store to allow sequences of microinstructions for executing SOPs of one or more S-Languages currently being utilized by CS 10110 to be written into EUSITT 20344 from MEM 10112 as required.

As previously described, EUSITT 20344's second input is a Data (DATA) input connected from JPD Bus 10142. EUSITT 20344's data input is utilized to write sequences of microinstructions into EUSITT 20344 from MEM 10112 through JPD Bus 10142. EUSITT 20344's first input is an address (ADR) input connected from output of Address Driver (ADRD) 25522 and NAG 20340. Address inputs provided by ADRD 25522 select word locations within EUSITT 20344 for writing of microinstructions into EUSITT 20344, or for reading of microinstructions from EUSITT 20344 to mCRD 20346 to control operation of EU 10122. Generation of these address inputs to EUSITT 20344 by NAG 20340 will be described further below.

d.d. Microcode Control Decode Register 20346

Output of EUSITT 20344 is connected to input of mCRD 20346. As previously described, mCRD 20346 is a register for receiving microinstructions from EUSITT 20344, and decoding logic for decoding those microinstructions and providing corresponding control signals to EU 10122. As indicated in Fig. 255, Diagnostic processor Micro-Program Register (DPmR) 25524 is a 60 bit register connected in parallel with output of EUSITT 20344 to input of mCRD 20346. DPmR 25524 may be loaded with 60 bit microinstructions by DP 10118. Diagnostic microinstructions may thereby be provided directly to input of mCRD 20346 to provide direct microinstruction by microinstruction control of EU 10122.

Outputs of mCRD 20346 are provided, in general, to all portions of EU 10122 to control detailed operations of EU 10122. Certain outputs of mCRD 20346 are connected to inputs of Next Address Source Select Multiplexer (NASS) 25526 and Long Branch Page Address Gate (LBPAG) 25528 and NAG 20340. As will be described further below, these outputs of mCRD 23046 are used in generating address inputs to EUSITT 20344 when particular microinstructions sequences call for Jumps or Long Branches to other microinstruction sequences. Outputs of mCRD 20346 are also connected in parallel to inputs of Execution Unit Micro-Instruction Parity Check Logic (EUmIPC) 25530. EUmIPC 25530 checks parity of all microinstruction outputs of mCRD 20346 to detected errors in mCRD 20346's outputs.

e.e. Next Address Generator 20340

As described above, read and write addresses to EUSITT 20344 provided by NAG 20340 through ADRD 25522. Address inputs to ADRD 25522 are provided from either NASS 25526 or Diagnostic Processor Address Register (DPAR) 25532. In normal operation, address inputs to EUSITT 20344 are provided from NASS 25526 as will be described momentarily. DP 10118, however, may load EUSITT 20344 addresses into DPAR 25532. These addresses may then be read from DPAR 25532 through ADRD 25522 to individually select address locations within EUSITT 20344. DPAR 25532 may be utilized, in particular, to provide addresses to allow stepping through of EU 10122 microinstruction sequences microinstruction by microinstruction.

As described above, NASS 25526 is a multiplexer having inputs from three NAG 20340 address sources. NASS 25526's first address input is from Jump (JMP) output of mCRD 20346 and LBPAG 25528. These address inputs are utilized, in part, when a current microinstruction calls for a Jump or Long Branch to another microinstruction or microinstruction sequence. Second address source is provided from SAS 25520 and, in general, is comprised of starting addresses of microinstruction sequences. SAS 25520 is a multiplexer having a first input from COAS 25512 and COAE 25516, that is starting addresses of microinstruction sequences corresponding to SOPs or for servicing JP 10114 Events. A second SAS 25520 input is provided from Sub-routine Return Address Stack (SUBRA) 25534. In general, and as will be described further below, SUBRA 25534 operates as a stack mechanism for storing current microinstruction addresses of interrupted microinstruction sequences. These stored addresses may subsequently be utilized to resume execution of those interrupted microinstruction sequences. Third address source to NASS 25526 is provided from Sequential and Case Address Generator (SCAG) 25536. In general, SCAG 25536 generates address to select sequential microinstructions within particular microinstruction sequences. SCAG 25536 also generates microinstruction address for microinstruction Case operations. As indicated in Fig. 255, outputs of SCAG 25536 and of SAS 25520 are bused together to comprise a single NASS 25526 input. Selection between outputs of SCAG 25536 and SAS 25520 are provided by control inputs (not shown for clarity of presentation) to SCAG 25536 and SAS 25520. Selection between NASS 25526's address inputs is controlled by Next Address Source Select Control Logic (NASSC) 25538, which provides control inputs to NASS 25526. NASSC 25538 is effectively a multiplexer receiving control inputs from TSTCON 20386 and TSTINT 20320. As will be described further below, TSTCON 20386 monitors certain operating conditions or states within EU 10122 and provides corresponding inputs to NASSC 25538. NASSC 25538 effectively decodes these control inputs from TSTCON 20386 to provide selection control input to NASS 25526.

Having described overall structure and operation of NAG 20340, operation of NAG 20340 will be

described in further detail next below.

Referring first to NASS 25526's address inputs provided from JMP output of mCRD 20346 and LBPAG 25528, this address source is provided to allow selection of a next microinstruction by a current microinstruction. JMP output of mCRD 20346 allows a current microinstruction to direct a Jump to another microinstruction within the same page of EUSITT 20344. NASS 25526's input through LBPAG 25528 is provided from another portion of mCRD 20346's output specifying pages within EUSITT 20344. This input through LBPAG 25528 allows execution of Long Branch operations, that is jumps from a microinstruction in one page of EUSITT 20344 to a microinstruction in another page. In addition, NASS 25526's input from JMP output of mCRD 20346 and through LBPAG 25528 is utilized to execute an Idle, or Standby, routine when EU 10122 is not currently executing a microinstruction sequence requested by FU 10120. In this case, Idle routine directs TSTCON 20386 to monitor EU 10122 Dispatch Pointer inputs to EU 10122 from FU 10120. If no EU 10122 Dispatch Pointers are present in COMQ 20342, or none are pending, TSTCON 20386 will direct NASSC 25538 to provide control inputs to NASS 25526 to select NASS 25526's input from mCRD 20346 and LBPAG 25528. Idle routine will continually test for EU 10122 Dispatch pointer inputs until such a Dispatch Pointer is received into COMQ 20342. At this time, TSTCON 20386 will detect the pending Dispatch Pointer and direct NASS 25538 to provide control outputs to NASS 25526 to select NASS 25526's input from, in general, SAS 25520. TSTCOND 20386 and NASSC 25538 will also direct NASS 25526 to select inputs from SAS 25520 upon return from a called microinstruction to a previously interrupted microinstruction sequence.

As described above, SAS 25520 receives starting addresses from COMQ 20342 and from SUBRA 25534. SAS 25520 will select the output of COAS 25512 or of COAE 25516 as the input to NASS 25526 when a new microinstruction sequence is to be initiated to execute a user's program SOP or to service a JP 10114 Event. SAS 25520 will select an address output of SUBRA 25534 upon return from a called sub-routine to a previously executing but interrupted sub-routine. SUBRA 25534, as described above, is effectively a stack mechanism for storing addresses of currently executing microinstructions when those microinstruction sequences are interrupted. SUBRA 25534 is an 11 bit wide by 8 word deep register with certain registers dedicated for use in stacking Event Handling microinstruction sequences. Other portions of SUBRA 25534 are utilized for stacking of microinstruction sequences for executing SOPs, that is for stacking microinstruction sequences wherein a first microinstruction sequence calls for a second microinstruction sequence. SUBRA 25534 is not operated as a first-in-first out stack, but as a random access memory wherein address inputs selecting registers and SUBRA 25534 are provided by microinstruction control outputs of mCRD 20346. Operations of SUBRA 25534 as a stack mechanism is thereby controlled by the microinstruction sequences stored in EUSITT 20344. As indicated in Fig. 255, addresses of current microinstructions of interrupted microinstruction sequences are provided to data input of SUBRA 25534 from output of SCAG 25536, which will be described next below.

As described above, SCAG 25536 generates sequential addresses to select sequential microinstructions within microinstruction sequences and to generate microinstruction addresses for Case operations. SCAG 25536 includes Next Address Register (NXTR) 25540, Next Address Arithmetic and Logic Unit (NAALU) 25542, and SCASE 25518. NAALU 25542 is a 12 bit arithmetic and logic unit. A first eleven bit input of NAALU 25542 is connected from output of ADRD 25522 and is thereby current address provided to EUSITT 20344. A second four bit input to NAALU 25542 is provided from output of SCASE 25518. During sequential execution of a microinstruction sequence, output of SCASE 25518 is binary zeros and carry input of NAALU is forced to 1. Output of NAALU 25542 will thereby be and address one greater than the current microinstruction address provided to EUSITT 20344 and will thereby be the address of the next sequential microinstruction. As indicated in Fig. 255, SCASE 25518 receives an input from output of SCALER 20338. This input is utilized during Case operations and allows a data sensitive number to be selected as SCASE 25518's output into second input of NAALU 25542. SCASE 25518's input from SCALER 20338 thereby allows NAG 20340 to perform microinstruction Case operations wherein Case Values are determined by the contents of SCALER 20338.

Next address outputs of NAALU 25542 are loaded into NXTR 25540, which is comprised of tri-state output registers. Next address outputs of NXTR 25540 are connected, in common with outputs of SAS 25520, to second input of NASS 25526 as described above. During normal execution of microinstruction sequences, therefore, SCAG 25536 will, through NASS 25526 and ADRD 25522, select sequential microinstructions from EUSITT 20344. SCAG 25536 may also, as just described, provide next microinstruction addresses in microinstruction Case operations.

In summary, NAG 20340 is capable of performing all usual microinstruction sequence addressing operations. For example, NAG 20340 allows selection of next microinstructions by current microinstructions, either for Jump operations or Long Branch operations, through NASS 25526's input from mCRD 20346's JMP or through LBPAG 25528. NAG 20340 may provide microinstruction sequence starting addresses through COMQ 20342 and SAS 25520, or may provide return addresses to interrupted and stacked microinstruction sequences through SUBRA 25534 and SAS 25520. NAG 20340 may sequentially address microinstructions of a particular microinstruction sequence through operation of SCAG 25536, or may perform mcroinstruction Case operations through SCAG 25536.

2. Operand Buffer 20322

Having described structure and operation of EUCL 20310, structure and operation of OPB 20322 will be described next below. As previously described, OPB 20322 receives operands, that is data, from MEM 10112 and FU 10120 through MOD Bus 10144 and JPD Bus 10142. OPB 20322 may then perform certain operand format translations to provide data to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. As previously described, EU 10122 may perform arithmetic operations on integer, packed and unpacked decimal, and single or double precision floating point numbers.

In summary, therefore, OPB 20322 is capable of accepting integer, single and double precision floating point, and packed and unpacked decimal operands from MEM 10112 and FU 10120 and providing appropriate fields of those operands to MULT 20314 and EXP 20316 in the formats most efficiently utilized by MULT 20314 and EXP 20316. In doing so, OPB 20322 extracts exponent and mantissa fields from single and double precision floating point operands to provide exponent and mantissa fields of these operands to, respectively, EXP 20316 and MULT 20314, and also unpacks, or converts, unpacked decimal operands to packed decimal operands most efficiently utilized by MULT 20314.

Having described structure and operation of OPB 20322, structure and operation of MULT 20314 will be described next below.

3. Multiplier 20314 (Figs. 257, 258)

MULT 20314, as previously described, performs addition, subtraction, multiplication, and division operations on mantissa fields of single and double precision floating point operands, integer operands, and decimal operands. As described above with reference to OPB 20322, OPB 20322 converts unpacked decimal operands to packed decimal operands to be operated upon by MULT 20314. MULT 20314 is thereby effectively capable of performing all arithmetic operations on unpacked decimal operands.

a.a. Multiplier 20314 Data Paths and Memory (Fig. 257)

Referring to Fig. 257, a more detailed block diagram of MULT 20314's data paths and memory is shown. As previously described, major elements of MULT 20314 include memory elements comprised of MULTRF 20350 and CONST 20360, operand input and result output multiplexing logic including MULTIM 20328 and MULTRM 20334, and arithmetic operation logic. MULT 20314's operand input and result output multiplexing logic and memory elements will be described first, followed by description of MULT 20314's arithmetic operation logic.

As previously described, input data, including operands, is provided to MULT 20314's arithmetic operation logic through MULTIN Bus 20354. MULTIN Bus 20354 may be provided with data from three sources. A first source is CONST 20360 which is a 512 word by 32 bit wide Read Only Memory. CONST 20360 is utilized to store constants used in arithmetic operations. In particular, CONST 20360 stores zone fields for unpacked decimal, that is ASCI character, operands. As previously described, unpacked decimal operands are received by OPB 20322 and converted to packed decimal operands for more efficient utilization by MULT 20314. As such, final result outputs generated by MULT 20314 from such operands are in packed decimal format. As will be described below, MULT 20314 may be utilized to convert these packed decimal results into unpacked decimal results by insertion of zone fields. As indicated in Fig. 257, address inputs are provided to CONST 20360 from EXPQ Bus 20325 and from output of mCRD 20346. Selection between these address inputs is provided through CONST Address Multiplexer (CONSTAM) 25710. CONST 20360 addresses will, in general, be provided from EUCL 20310 but alternately may be provided from EXPQ Bus 20325 for special operations.

Operand data is provided to MULTIN Bus 20354 through MULTIM 20328, which is a dual input, 64 bit multiplexer. A first input of MULTIM 20328 is provided from OPQ Bus 20323 and is comprised of operand information provided from OPB 20322. OPQ Bus 20323 is a 56 bit wide bus and operand data appearing thereon may be comprised of 32 bit integer operands; 32 bit packed decimal operands, either provided directly from OPB 20322 or as a result of OPB 20322's conversion of an unpacked decimal to a packed decimal operand; 24 bit single precision operand mantissa fields; or 56 bit double precision floating point operand mantissa fields. As previously described, certain OPQ Bus 20323 may be zero or sign extension filled, depending upon the particular operand.

Second input of MULTIM 20328 is provided from MULTRF 20350. MULTRF 20350 is a 16 word by 64 bit wide random access memory. As indicated in Figs. 203 and 257, MULTRF 20350 is connected between output of RFR 20336, through FR Bus 20337, and to input of MULT 20314's arithmetic operation logic through MULTIM 20328 and MULTIN Bus 20354. MULTRF 20350 may therefore be utilized as a scratch pad memory for storing intermediate results of arithmetic operations, including reiterative arithmetic operations. In addition, a portion of MULTRF 20350 is utilized, as in GRF 10354, as an EU 10122 Stack Mechanism similar to MIS 10368 and MOS 10370 in FU 10120. Operation of EU 10122 Stack Mechanism will be described in a following descripion of EU 10122's interfaces to MEM 10112 and FU 10120. Address Inputs (ADR) of MULTRF 20350 are provided from Multiplier Register File Address Multiplexer (MULTRFAM) 25712.

MULTRFAM 25712 is a dual four bit multiplexer comprised, for example, of SN74S258s. In addition to address inputs to MULTRF 20350, MULTRFAM 25712 provides address inputs to EXPRF 20380. As previously described, MULTRF 20350 and EXPRF 20380 together comprise an EU 10122 general register file similar to GRF 10354 and FU 10120. As such, MULTRF 20350 and EXPRF 20380 are addressed in parallel to read and write parallel entries from and to MULTRF 20350 and EXPRF 20380. Address inputs to MULTRFAM 25712 are provided, first, from outputs of mCRD 20346, thus providing microinstruction control of addressing of MULTRF 20350 and EXPRF 20380. Second address input to MULTRFAM 25712 is provided from output of Multiplier Register File Address Counter (MULTRFAC) 25714.

MULTRFAC 25714 is a four bit counter and is used to generate sequential addresses to MULTRF 20350 and EXPRF 20380. Initial addresses are loaded into MULTRFAC 25714 from Multiplier Register File Address Counter Multiplexer (MULTRFACM) 25716. MULTRFACM 25716 is a dual four bit multiplexer. Inputs to MULTRFACM 25716 are provided, first, from outputs of mCRD 20346. This input allows microinstruction selection of an initial address to be loaded into MULTRFAC 25714 to be subsequently used and generating sequential MULTRF 20350 and EXPRF 20380 addresses. Second address input to MULTRFACM 25716 is provided from OPQ Bus 20323. MULTRFACM 25716's input from OPQ Bus 20323 allows a single address, or a starting address of a sequence of addresses, to be selected through JPD Bus 10142 or MOD Bus 10144, for example from MEM 10112 or FU 10120.

Intermediate and final result outputs of MULT 20314 arithmetic logic are provided to data inputs of MULTRF 20350 directly from FR Bus 20337 and from MULTRM 20334. Inputs to MULTRM 20334, in turn, are provided from FR Bus 20337 and from output of CONSIZE 20352 and TSTINT 20320.

FR Bus 20337 is a 64 bit bus connected from 64 bit output of RFR 20336 and carries final and intermediate results of MULT 20314 arithmetic operations. As will become apparent in a following description of MULT 20314 arithmetic operation logic, RFR 20336 output, and thus FR Bus 20337, are 64 bits wide. Sixty-four bits are provided to insure retention of all significant data bits of certain MULT 20314 arithmetic operation intermediate results, in particular operations involving double precision floating point 64 bit mantissa fields. In addition, as will be described momentarily and has been previously stated, MULT 20314 may convert a final result in packed decimal format into a final result in unpacked decimal format. In this operation, a single 32 bit, or one word, packed decimal result is converted into a 64 bit, or two word, unpacked decimal format by insertion of zone fields.

As described above, two parallel data paths are provided to transfer information from FR Bus 20337 into MULTRF 20350. First path is directly from FR Bus 20337 and second path is through Unpacked Decimal Multiplexer (UPDM) 25718 of MULTRM 20334. Direct path is utilized for thirty-two bits of information comprising bits 0 to 23 and bits 56 to 63 of FR Bus 20337. Data path through UPDM 25718 may comprise either bits 24 to 55 of FR Bus 20337, which are connected into a first input of UPDM 25718, or bits 40 through 55 which are connected to a second input of UPDM 25718. Single precision floating point numbers are 32 bit numbers plus two or more guard bits and are thus written into MULTRF 20350 through bits 0 to 23 of the direct path into MULTRF 20350 and through first input (bits 24 to 55) of UPDM 25718. Double precision floating point numbers are 5 bits wide, plus guard bits, and thus utilize the direct path into MULTRF 20350 and the path through first input of UPDM 25718. Bits 56 to 63 of direct path are utilized for guard bits of double precision floating point numbers. Both integer and packed decimal numbers utilize bits 24 through 55 of FR Bus 20337, and are thus written into MULTRF 20350 through first input of UPDM 25718. As previously described, bits 0 to 23 of these operands are filled by sign extension.

a.a.a. Container Size Check

As stated above, MULTRM 20334 has an Input from CONSIZE 20352. As will be described below with reference to TSTINT 20320, CONSIZE 20352 performs a "container size" check upon each store back of results from EU 10122 to MEM 10112. CONSIZE 20352 compares the number of significant bits in a result to be stored back to the logical descriptor describing the MEM 10112 address space that result is to be written into. Where reiterative write operations to MEM 10112 are required to transfer a result into MEM 10112, that is a string transfer, container size information may read from CONSIZE 20352 through Container Size Driver (CONSIZED) 25720 and MULTRM 20334 and written into MULTRF 20350. This allows EU 10122, using container size information stored in MULTRM 20350, to perform continuous container size checking during a string transfer of result from EU 10122 to MEM 10112. In addition, as will be described momentarily, container size information may be read from CONSIZE 20352 to JPD Bus 10144.

b.b.b. Final Result Output Multiplexer 20324

Referring finally to FROM 20324, as previously described FROM 20324 is utilized to transfer, in general, results of EU 10122 arithmetic operations onto JPD Bus 10142 for transfer to MEM 10112 or FU 10120. As indicated in Fig. 257, FROM 20324 is comprised of 24 bit Final Result Bus Driver (FRBD) 25722 and Result Bus Driver (RBR) 25724. Input of FRBD 25722 is connected from FR Bus 20337 and allows data appearing thereon to be transferred onto JPD Bus 10142. In particular, FRBD 25722 is utilized to transfer 24 bit mantissa fields of single precision floating point results onto JPD Bus 10142 in parallel with a corresponding exponent field from EXP 20316. RBR 25724 input is connected from RSLT Bus 20388 to allow

output of UPDM 25718 to be transferred onto JPD Bus 10142. RBR 25724, RSLT Bus 20388, and UPDM 25718 are used, in general, to transfer final results of EU 10122 operations from output of MULT 20314 onto JPD Bus 10142. Final results transferred by this data path include integer, packed and unpacked decimal results, and mantissa fields of double precision floating point results. Both unpacked decimal numbers and mantissa fields of double precision floating point numbers are comprised of two 32 bit words and are thus transferred onto JPD Bus 10142 in two sequential transfer operations.

Having described structure and operation of MULT 20314's memory elements and input and output circuitry, MULT 20314's arithmetic operation logic will be described next below.

4. Test and Interface Logic 20320 (Figs. 260—268)

As previously described, TSTINT 20320 includes CONSIZE 20352, ECPT 20328, TSTCOND 20384, and INTRPT 20388. CONSIZE 20352, as previously described, performs "container size" check operations when results of EU 10122 operations are to be written into MEM 10112. That is, CONSIZE 20352 compares size or number of significant bits, of an EU 10122 result to the capacity, or container size, of the MEM 10112 location that EU 10122 result is to be written into. As indicated, in Fig. 203, CONSIZE 20352 receives a first input, that is the results of EU 10122 operations, from FR Bus 20337. A second input of CONSIZE 20352 is connected to LENGTH Bus 20226 to receive length field of logical descriptors identifying MEM 10112 address space into which those EU 10122 results are to be written. CONSIZE 20352 includes logic circuitry, for example a combination of Read Only Memory and Field Programmable Logic Arrays, for examining EU 10122 operation results appearing on FR Bus 20337 and determining the number of bits of data in those results. CONSIZE 20352 compares EU 10122 result size to logical descriptor length field and, in particular, if result size exceeds logical descriptor length, provides an alarm output to ECPT 20328, described below.

TSTCOND 20384, previously described and which will be described further below, is an interface circuit between FU 10120 and EU 10122. TSTCOND 20384 allows FU 10120 to specify and examine results of certain test operations performed by EU 10122 with respect to EU 10122 operations.

ECPT 20328 monitors certain EU 10122 operations and provides outputs indicating when certain "exceptions" have occurred. These exceptions include attempted divisions by zero, floating point exponent underflow or overflow, and integer container size fault.

INTRPT 20388 is again an interface between EU 10122 and FU 10120 allowing FU 10120 to interrupt EU 10122 operations. INTRPT 20388 allows FU 10120 to direct EU 10122 to execute certain operations to aid in handling of certain FU 10120 events previously described.

Operation of CONSIZE 20352, ECPT 20328, TSTCOND 20384, INTRPT 20388, and other features of EU 10122's interface with FU 10120 will be described further below in the following description of operation of that interface and of operation of certain EU 10122 internal mechanisms, such as FU 10120 Stack Mechanisms.

a.a. FU 10120/EU 10122 Interface

As previously described, EU 10122 and FU 10120 are asychronous processors, each operating under its own microcode control. EU 10122 and FU 10120 operate simultaneously and independently of each other but are coupled, and their operations coordinated, by interface signals described below. Should EU 10122 not be able to respond immediately to a request from FU 10120, FU 10120 will idle until EU 10122 becomes available; conversely, should EU 10122 not receive, or have present, operands or a request for operations from FU 10120, EU 10122 will remain in idle state until operands and requests for operations are received from FU 10120.

In normal operation, EU 10122 manipulates operands under control of FU 10120, which in turn is under control of SOPs of a user's program. When FU 10120 requires arithmetic or logical manipulation of an operand, FU 10120 dispatches a command, that is an Execute Unit Dispatch Pointer (EUDP) to EU 10122. As previously described, an EUDP is basically an initial address into EUSITT 20344. An EUDP identifies starting location of a EU 10122 microinstruction sequence performing the required operation upon operands. Operands are fetched from MEM 10112 under FU 10120 control, as previously described, and are transferred into OPB 20322. Those operands are then called from OPB 20322 by EU 10122 and transferred into MULT 20314 and EXP 20316 as previously described. After the required operation is completed, FU 10120 is notified that a result is ready. At this point, FU 10120 may check certain test conditions, for example through TSTCOND 20384, such as whether an integer or decimal carry bit is set or whether a mantissa sign bit is set or reset. This test operation is utilized by FU 10120 for conditional branching and synchronization of FU 10120 and EU 10122 operations. Exception checking, by ECPT 20328, is also performed at this time. Exception checking determines, for example, whether division by zero was attempted or if a container size fault has occurred. In general, FU 10120 is not informed of exception errors until FU 10120 requests exception checking. After results are transferred into FU 10120 or MEM 10112 by EU 10122, EU 10122 goes to idle operation until a next operation is requested by FU 10120.

Having briefly described overall interface operation between FU 10120 and EU 10122, operation of that interface, referred to as handshaking, will be described in greater detail next below. In general, handshaking operation between EU 10122 and FU 10120 during normal operation may be regarded as following into six operations. These operations may include, for example, loading of COMQ 20342, loading of OPB 20322, storeback or transfer of results from EU 10122 to FU 10120 or MEM 10112, check of test

conditions, exception checking, and EU 10122 idle operation. Handshaking between FU 10120 and EU 10122 will be described below for each of these classes of operation, in the order just referred to.

a.a.a. Loading of Command Queue 20342 (Fig. 260)

Referring to Fig. 260, a schematic representation of EU 10122's interface with FU 10120 for purposes of loading COMQ 20342 as shown. During normal SOP directed JP 10114 operation, 8 bit operation (OP) codes are parsed from the instruction stream, as previously described, and concatenated with dialect information to address EUSDT 20266 also as previously described. EUSDT 20266 provides corresponding addresses, that is EUDPs, to EUSITT 20344.

Dialect information specifies the S-Language currently being executed and, consequently, the group of microinstruction sequences available in EUSITT 20344 for that S-Language. As previously described, FU 10120 may specify four S-Language dialects with up to 256 EU 10122 microinstruction sequences per dialect, or 8 dialects with up to 128 microinstruction sequences per dialect.

EUDPs provided by EUSDT 20266 are comprised of a 9 bit address field, a 2 bit operand information field, and a 1 bit flag field, as previously described. Address field is starting address of a microinstruction sequence in EUSITT 20344 and EU 10122 will perform the operation directed by that microinstruction sequence. EUSITT 20344 requires 11 bits of address field and the 9 bit address field of EUDPs are mapped into an 11 bit address field by left justification and zero filling.

FU 10120 may also dispatch, or select, any EU 10122 microinstruction controlled operation from JPD Bus 10142. Such EUDPs are provided from JPD Bus 10142 to data input of EUSITT 20344 and passed directly through to mCRD 20346. Before a EUDP may be provided from JPD Bus 10142, however, FU 10120 provides a check operation comparing that EUDP to a list of legal, or allowed, EUSITT 20344 addresses stored in MEM 10112. A fault will be indicated if an EUDP provided through JPD Bus 10142 is not a legal EUSITT 20344 address. Alternately, FU 10120 may effectively provide an EUDP, or EUSITT 20344 addresses, from a literal field in a FU 10120 microinstruction word. Such a FU 10120 microinstruction word literal field may be effectively utilized as an SOP into EUSDT 20266.

Handshaking between EU 10122 and FU 10120 during load COMQ 20342 operations may proceed as illustrated in Fig. 260. A twelve bit EUDP may be placed on EUDIS Bus 20206 and Control Signal Load Command Queue (LDCMQ) asserted. If COMQ 20342 is full, EU 10122 raises control signal Command Hold (CMDHOLD) which causes FU 10120 to remain in State M0 until there is room in COMQ 20342. As previously described, COMQ 20342 is comprised of two, two word buffers wherein one buffer is utilized for normal SOP operation and the other utilized for control of FU 10120 and EU 10122 internal mechanism operation.

EUDPs are loaded into COMQ 20342 when state timing signals M1CPT and M1 are asserted. If a EUDP being transferred into COMQ 20342 concerns a double precision floating point operation, control signal Set Double Precision (SETDP) is asserted. SETDP is utilized to control OPB 20322, and because single precision and precision floating point operations otherwise utilize the same SOP and thus would otherwise refer to same EUSITT 20344 microinstruction sequence.

At this point, a EUDP has been loaded into COMQ 20342 and will be decoded to control FU 10120 operation by EUCL 20310 as previously described. Each particular EUDP will be cleared by that EUDPs EUSITT 20344 microinstruction sequence after the requested microinstruction sequence has been executed.

b.b.b. Loading of Operand Buffer 20320 (Fig. 261)

Referring to Fig. 261, a diagramic representation of the interface and handshaking between EU 10122, FU 10120 and MEM 10112 for loading OPB 20322 is shown. Control signal Clear Queue Full (CLQF) from EU 10122 must be asserted by EU 10122 before FU 10120 initiates a request to MEM 10112 for an operand to be transferred to EU 10122. CLQF clears and "EU 10122's OPB 20322 Full" condition in FU 10120. CLQF indicates, thereby, that there is room in OPB 20322 to receive operands. If FU 10120 is in a "EU 10122's OPB 20322 Full" condition and further operand is required to be transferred to EU 10122, FU 10120 will remain in State M1 until CLQF is asserted.

At the beginning of execution of a particular SOP, FU 10120 may transfer two operands to OPB 20322 without "EU 10122's OPB 20322 Full" condition occurring. This is because EU 10122 is idle at the beginning of an SOP execution and generally immediately unloads a first operand from OPB 20322 before a second operand arrives.

Control signal Job Processor Operand (JPOP) provided from FU 10120 must be non-asserted for operands to be transferred from MEM 10112 to OPB 20322 through MOD Bus 10144. This is the normal condition of JPOP. If JPOP is asserted, OPB 20322 is loaded with data from JPD Bus 10142. Data is strobed into OPB 20322 from JPD Bus 10142 by control signals M1CPT and JPOP. Operands read from MEM 10112, however, are transferred into OPB 20322 through MOD Bus 10144 when MEM 10112 asserts DAVEB to indicate that valid data from MEM 10112 is available on MOD Bus 10144. DAVEB is also utilized to strobe data on MOD Bus 10144 into OPB 20322. If control signal ZFILL from MEM 10112 is asserted at this point, ZFILL is interpreted during integer operand operations to indicate that those operands are unsigned and

should be left zero filled, rather than sign extended. If data is being provided from JPD Bus 10142 rather than from MEM 10112, that is if JPOP is asserted, bit 11 of current EUDP may be utilized to perform the same function as ZFILL during loading of OPB 20322 from MOD Bus 10144.

Loading of OPB 20322 is controlled, in part, by bits 9 and 10 of EUDPs provided from FU 10120 through EUDIS Bus 20206. Bit 9 indicates length of a first operand while bit 10 indicates length of a second operand. Operand length, together with operand type specified in address portion of a EUDP, determines how a particular operand is unloaded from OPB 20322 and transferred into MULT 20314 and EXP 20316.

At this point, both COMQ 20342 and OPB 20322 have been loaded with, respectively, EUDPs and operands. It should be noted that operands are generally not transferred into OPB 20322 before a corresponding EUDP is loaded into COMQ 20342. Operands and EUDPs may, however, be simultaneously transferred into EU 10122. If other operands are required for a particular operation, those operands are loaded into OPB 20322 as described above.

c.c.c. Storeback (Fig. 262)

Referring to Fig. 262, a diagramic representation of a storeback, or transfer, of results to MEM 10112 from EU 10122 and handshaking performed therein is shown. When a final result of a EU 10122 operation is available, EU 10122 asserts control signal Data Ready (DRDY). FU 10120 thereupon responds with control signal Transfer to JPD Bus 10142 (XJPD), which gates EU 10122's result onto JPD Bus 10142. In normal operation, that is execution of SOPs, FU 10120 causes EU 10122's result to be stored back into a destination in MEM 10112, as selected by a physical descriptor provided from FU 10120. Alternately, a result may be transferred into FU 10120, 32 bits, or one word, at a time.

FU 10120 may, as described above and described further below, check EU 10122 test conditions during storeback of results. FU 10120 generates control signal Transfer Complete (XFRC) once the storeback operation is completed. XFRC also indicates to EU 10122 that EU 10122's results and test conditions have been accepted by FU 10120, so that EU 10122 need no longer assert these results and test conditions.

d.d.d. Test Conditions (Fig. 263)

Referring to Fig. 263, a diagramic representation of checking of EU10122 test conditions by FU 10120, and handshaking therein, is shown. As previously described, test results indicating certain conditions and operations of EU 10122 are sampled and stored in TSTCOND 20384 and may be examined by FU 10120. When DRDY is asserted by EU 10122, FU 10120 may select, for example, one of 8 EU 10122 conditions to test, as well as transferring results as described above. EU 10122 conditions which may be tested by FU 10120 are listed and described below. Such conditions, as whether a final result is positive, negative, or zero, may be checked in order to facilitate conditional branching of FU 10120 operations as previously described. FU 10120 specifies a condition to be tested through Test Condition Select signals (TEST(24)). FU 10120 asserts control signal EU Test Enable (EUTESTEN) to EU 10122 to gate the selected test condition. That selected test condition then appears as Data Signal Test Condition (TC) from EU 10122 to FU 10120. A TC of logic 1 may, for example, indicate that the selected condition is false while a TC of logic 0 may indicate that the selected condition is true. FU 10120 indicates that FU 10120 has sensed the requested test condition, and that the test condition need no longer be asserted by EU 10122, by asserting control signal XFRC.

e.e.e. Exception Checking (Fig. 264)

Referring to Fig. 264, a diagramic representation of exception checking of EU 10122 exceptions by FU 10120, and handshaking therein, is shown. As previously described, any EU 10122 exception conditions may be checked by FU 10120 as FU 10120 is initiating storeback of EU 10122 results. Exception checking may detect, for example, attempted division by zero, floating point exponent underflow or overflow, or a container size fault. An attempted division by zero or floating point underflow or overflow may be checked before storeback, that is without specific request by FU 10120.

As previously described, a container size fault is detected by CONSIZE 20352 by comparing length of result with size of destination container in MEM 10112. Container size exception checking occurs during store back of EU 10122 results, that is while FU 10120 is in State SB. Container size is automatically performed by EU 10122 hardware, that is by CONSIZE 20352, only on results of less than 33 bits length. Size checking of larger results, that is larger integers and BCD results, is performed by a microcode routine, using CONSIZE 20352's output, as transfer of such larger results is executed as string transfer. It is unnecessary to perform container size check for either single or double precision floating point results as these data types always occupy either 32 or 64 bits. Destination container size is provided to CONSIZE 20352 through LENGTH Bus 20226.

Control signal Length to Memory AON or Random Signals (LMAONRS) is generated by FU 10120 from Type field of the logical descriptor corresponding to a particular EU 10122 result. LMAONRS indicates that the results data type is an unsigned integer. LMAONRS determines the manner in which a required container size of the EU 10122 result is determined. After receivng this information from LMAONRS, EU 10122 determines whether destination container size in MEM 10112 is sufficiently large to contain the EU

10122 result. If that destination container size is not sufficiently large, a container size fault is detected by CONSIZE 20352, or through an EU 10122 microinstruction sequence.

Container size faults, as well as division by zero and exponent underflow and overflow faults, are signaled to FU 10120 when FU 10120 asserts control signal Check Size (CKSIZE). At this time, EU 10122 asserts control signal Exception (EXCPT) if any of the above faults has occurred. If a fault has occurred, an Event request to FU 10120 results. When an Event request is honored by FU 10120, FU 10120 may interrupt EU 10122 and dispatch EU 10122 to a microinstruction routine that transfers those exception conditions onto JPD Bus 10142. If a container size fault has caused that exception condition, EU 10122 may transfer to FU 10120 the required container size through JPD Bus 10142.

f.f.f. Idle Routine

Finally, when a current EU 10122 operation is completed, EU 10122 goes into an idle loop microinstruction routine. If necessary, FU 10120 may assert control signal Excute Unit Abort (EUABORT) to force EU 10122 into idle loop microinstruction routine until EU 10122 is required for further operations.

g.g.g. EU 10122 Stack Mechanism (Figs. 265, 266, 267)

As previously described, EU 10122 may perform either of two classes of operations. First, EU 10122 may perform arithmetic operations in execution of SOPs of user's programs. Second, EU 10122 may operate as an arithmetic calculator assisting operation of FU 10120's internal mechanisms and operations, referred to as kernel operations.

In kernel operation, EU 10122 acts as an arithmetic calculator for FU 10120 during address generation, address translation, and other kernel functions. In kernel mode, EU 10122 is executing microinstruction sequences at request of FU 10120 kernel microinstruction sequences, rather than at request of an SOP. In general, these kernel operations are vital to operation of JP 10114. FU 10120 may interrupt EU 10122 operations with regard to SOPs and initiate EU 10122 microinstruction sequences to perform kernel operations.

When interrupted, EU 10122 saves EU 10122's current operating state in a one level deep stack. EU 10122 may then accept an EUDP from that portion of COMQ 20342 utilized to receive and store EUDPs regarding FU 10120's and EU 10122's internal, or kernel, operations. When requesting kernel operations by EU 10122, FU 10120 generally transfers operands to OPB 20322 through JPD Bus 10142, and receives EU 10122 final results through JPD Bus 10142. Operands may also be provided to EU 10122 through MOD Bus 10144. After EU 10122 has completed a requested kernel operation, EU 10122 reloads operating state from its internal stack and continues normal operation from the point normal operation was interrupted.

Should another interrupt from FU 10120 occur while a prior interrupt is being executed, EU 10122 moves current state and data, that is of first interrupt, to MEM 10112. EU 10122 requests FU 10120 store state and date of first interrupt in MEM 10112 by requesting an "EU 10122 Stack Overflow" Event. EU 10122's "normal" state, that is state and data pertaining to the operation EU 10122 is executing at time of occurrence of first interrupt, is stored in an EU 10122 internal stack and remains there. EU 10122 then begins executing second interrupt. When EU 10122 has completed operations for second interrupt, state from first interrupt is reloaded from MEM 10112 by EU 10122 requesting a "EU 10122 Stack Underflow" Event to FU 10120. EU 10122 then completes execution of first interrupt and reloads state and resumes execution of normal operation, that is the operation being executed before the first interrupt.

EU 10122 is therefore capable of handling interrupts from FU 10120 during two circumstances. First interrupt circumstance is comprised of interrupts occurring during normal operation, that is while executing SOPs of user's programs. Second circumstance arises when interrupts occur during kernel operations, that is during execution of microinstruction sequences for handling interrupts. EU 10122 operation will be described next below for each of these circumstances, and in the order referred to.

Referring to Fig. 265, a diagramic representation of EU 10122's stack mechanisms, previously described, is shown. Those portions of EU 10122's stack mechanisms residing within EU 10122 are comprised of EU 10122's Current State Registers (EUCSRs) 26510 and EU 10122's Internal Stack (EUIS) 26512. EUCSR 26510 is comprised of EU 10122's internal registers which contain data and state of current EU 10122 operation. EUCSR 26510 may be comprised, for example, of mCRD 20346, registers of TSTINT 20320, and the previously described registers within MULT 20314 and EXP 20316.

State and data contained in EUCSR 26510 is that of the operation currently being executed by EU 10122. This current state may, for example, be that of a SOP currently being executed by EU 10122, or that of an interrupt, for example a fourth interrupt of a nested sequence of interrupts, requested by FU 10120.

EUIS 26512 is comprised of certain registers of MULTRF 20350 and EXPRF 20380. EUIS 26512 is utilized to store and save current state of an SOP operation currently being executed by EU 10122 and which has been interrupted. State and data of that SOP operation will remain stored in EUIS 26512 regardless of the number of interrupts which may occur on a nested sequence of interrupts requested by FU 10120. State and data of the interrupted SOP operation will be returned from EUIS 26512 to EUCSR 26510 when all interrupts have been completed.

Final portion of EU 10122's stack mechanism is that portion of EU 10122's internal stack (EUES) 26514

residing in MEM 10112. EUES 26514 is comprised of certain MEM 10112 address locations used to store state and data of successive interrupt operations of sequences of nested interrupts. That is, if a sequence of four interrupts is requested by FU 10120, state and data of fourth interrupt will reside in EUCSR 26510 while state and data of first, second, and third interrupts have been transferred, in sequence, into EUES 26514. In this respect, and as previously described operation of EU 10122's stack mechanisms is similar to that of, for example, MIS 10368 and SS 10336 previously described with reference to Fig. 103.

As described above, an interrupt may be requested of EU 10122 by FU 10120 either during EU 10122 normal operation, that is during execution of SOPs by EU 10122, or while EU 10122 is executing a previous interrupt requested by FU 10120. Operation of EU 10122 and FU 10120 upon occurrence of an interrupt during EU 10122 normal operation will be described next below.

Referring to Fig. 266, a diagramic representation of handshaking between EU 10122 and FU 10120 during an interrupt of EU 10122 while EU 10122 is operating in normal mode is shown and should be referred to in conjunction with Fig. 265. For purposes of the following discussions, interrupts of EU 10122 operations by FU 10120 are referred to as nanointerrupts to distinguish from interrupts internal to FU 10120.

FU 10120 interrupts normal operation of EU 10122 by assertion of control signal Nano-Interrupt (NINTP) during State M0 of FU 10120 operation. NINTP may be masked by EU 10122 during certain critical EU 10122 operations, such as arithmetic operations. If NINTP is masked by EU 10122, FU 10120 will remain in State NW until EU 10122 acknowledges the interrupt.

Upon receiving NINTP from FU 10120, EU 10122s transfers state and data of current SOP operation from EUCSR 26510 to EUIS 26512. EU 10122 then asserts control signal Nano-Interrupt Acknowledge (NIACK) to FU 10120 to acknowledge availability of EU 10122 to accept a nanointerrupt. FU 10120 will then enter State M1 and place an EUDP on EUDIS Bus 20206. Loading of COMQ 20342 then proceeds as previously described, with EU 10122 loading nanointerrupt EUDPs into the appropriate registers of COMQ 20342. COMQ 20342 is loaded as previously described and, if JPOP is asserted, data transferred into OPB 20322 from JPD Bus 10142. If JPOP is not asserted, data is taken into OPB 20322 from MOD Bus 10144. EU 10122 then proceeds to execute the required nanointerrupt operation and storing back of results and checking of test conditions proceeds as previously described for EU 10122 normal operation. In general, exception checking is not performed. When EU 10122 has completed execution of the nanointerrupt operation, EU 10122 transfers state and data of the interrupted SOP operation from EUIS 26512 to EUCSR 26510 and resumes execution of that SOP. At this point, EU 10122 asserts control signal Nano-Interrupt Trap Enable (NITE). NITE is received and tested by FU 10120 to indicate end of nanointerrupt processing.

Referring to Fig. 267, a diagramic representation of interfaces between EU 10122, FU 10120, and MEM 10112 during nested, or sequential, EU 10122 interrupts for kernel operations, and handshaking therein, is shown. During the following discussion, it is assumed that EU 10122 is already processing a nanointerrupt for a kernel operation submitted to EU 10122 by FU 10120. FU 10120 may then submit a second, third, or fourth, nanointerrupt to EU 10122 for a further kernel operation. FU 10120 will assert NINTP to request a nanointerrupt of EU 10122. EU 10122's normal mode state and data from a previously executing SOP operation has been stored in, and remains in, EUIS 26512. Current state and data of currently executing nanointerrupt operation in EUCSR 26510 will be transferred to EUES 26514 in MEM 10112 to allow initiation of pending nanointerrupt. EU 10122 will at this time assert NIACK and control signal Execute Unit Event (EXEVT). EXEVT to FU 10120 informs FU 10120 that an EU 10122 Event has occurred, specifically, and in this case, EXEVT requests FU 10120 service of an EU 10122 Stack Overflow. FU 10120 is thereby trapped to an "EU 10122 Stack Overflow" Event Handler microinstruction sequence. This handler transfers current state and data of interrupted nanointerrupt previously executing in EU 10122 into EUES 26514. State and data of interrupted nanointerrupt is transferred to EUES 26514, one 32 bit word at a time. FU 10120 asserts control signals XJPD to gate each of these state and data words onto JPD Bus 10142 and controls transfer of these words into EUES 26514.

Processing of new nanointerrupt proceeds as described above with reference to interrupts occurring during normal operation. If any subsequent nanointerrupts occur, they are handled in the same manner as just described; FU 10120 signals a nanointerrupt to FU 10120, current EU 10122 state and data is saved by FU 10120 in EUES 26514, and new nanointerrupt is processed. After a nested nanointerrupt, that is a nanointerrupt of a sequence of nanointerrupts, has been serviced, EU 10122 asserts control signal EU 10122 Trap (ETRAP) to FU 10120 to request a transfer of a previous nanointerrupt's state and data from EUES 26514 to EUCSR 26510. FU 10120 will retrieve that next previous nanointerrupt state and data from EUES 26514 through MOD Bus 10144 and will transfer that data and state onto JPD Bus 10142. This state and data is returned, one 32 bit word at a time, and is strobed into EU 10122 by JPOP from FU 10120. Processing of that prior nanointerrupt will then resume. The servicing of successively prior nanointerrupts will continue until all previous nanointerrupts have been serviced. Original state and data of EU 10122, that is that of SOP operation which was initially interrupted, is then returned to EUCSR 26510 from EUIS 26512 and execution of that SOP resumed. At this time, EU 10122 asserts NITE to indicate end of EU 10122 kernel operations in regard to nanointerrupts.

Having described structure and operation of EU 10122, FU 10120 and MEM 10112, with respect to servicing of kernel operation nanointerrupts by EU 10122, loading of EU 10122's EUSITT 20344 with microinstruction sequences will be described next below.

h.h.h.h Loading of Execute Unit S-Interpreter Table 20344 (Fig. 268)

Referring to Fig. 268, a diagramic representation of interface and handshaking between EU 10122, FU 10120, MEM 10112, and DP 10118 during loading of microinstructions into EUSITT 20344 is shown. As previously described, EUSITT 20344 contains all microinstructions required for control of EU 10122 in executing kernel nanointerrupt operations and in executing arithmetic operations in response to SOPs of user's programs. EUSITT 20344 may store microinstruction sequences for interpreting arithmetic SOPs of user's programs for, for example, up to 4 different S-Language Dialects. In general, a capacity of storing microinstruction sequences for arithmetic operations in up to 4 S-Language Dialects is sufficient for most requirements, so that EUSITT 20344 need be loaded with microinstruction sequences only at initialization of CS 10110 operation. Should microinstruction sequences for arithmetic operations of more than 4 S-Language Dialects be required, those microinstruction sequences may be loaded into EUSITT 20344 in the manner as will be described below.

As previously described, a portion of the microinstructions stored in EUSITT 20344 is contained in Read Only Memories and is thus permanently stored in EUSITT 20344. Microinstruction sequences permanently stored in EUSITT 20344 are, in general, those required for execution of kernel operations. Microinstruction sequences permanently stored in EUSITT 20344 include those used to assist in writing other EU 10122 microinstruction sequences into EUSITT 20344 as required. Certain microinstruction sequences are stored in a Random Access Memory, referred to as the Writeable Control Store (WCS) portion of EUSITT 20344, and include these for interpreting arithmetic operation SOPs of various S-Language Dialects.

Writing of microinstruction sequences into EU 10122 is initialized by forcing EU 10122 into an Idle state. Initialization of EU 10122 is accomplished by FU 10120 asserting EUABORT or by DP 10118 asserting control signal clear (CLEAR). Either EUABORT or CLEAR will clear a current operation of EU 10122 and force EU 10122 into Idle state, wherein EU 10122 waits for further EUDPs provided from FU 10120. FU 10120 then dispatches a EUDP initiating loading of EUSITT 20344 to EU 10122 through EUDIS Bus 20206. Load EUSITT 20344 EUDP specifies starting address of a two step microinstruction sequence in the PROM portion of EUSITT 20344. This two step microinstruction sequence first loads zeros into SCAG 25536, which as previously described provides read and write addresses to EUSITT 20344. EUSITT 20344 load microinstruction sequence then reads a microinstruction from EUSITT 20344 to mCRD 20346. This microinstruction specifies conditions for handshaking operations with FU 10120 so that loading of EUSITT 20344 may begin. At this time, and from this microinstruction word, EU 10122 asserts control signal DRDY to FU 10120 to indicate that EU 10122 is ready to accept EUDPs from FU 10120 for directing loading of EUSITT 20344. This initial microinstruction also generates a write enable control signal for the WCS portion of EUSITT 20344, inhibits loading of mCRD 20346 from EUSITT 20344, and inhibits normal loading operations of NXTR 25540 and SCAG 25536. This first microinstruction also directs NASS 25526 to accept address inputs from SCAG 25536 and, finally, causes NITE to FU 10120 to be asserted to unmask nanointerrupts from FU 10120.

FU 10120 then generates a read request to MEM 10112, and MEM 10112 transfers a first 32 bit word of a EU 10122 microinstruction word onto JPD Bus 10142. Each such 32 bit word from MEM 10112 comprises one half of a 64 bit microinstruction word of EU 10122. When FU 10120 receives DRDY from EU 10122, FU 10120 generates control signal Load Writeable Control Store (LDWCS). LDWCS in turn transfers a 32 bit word on JPD Bus 10142 into a first address of the WCS portion of EUSITT 20344. A next 32 bit half word of a EU 10122 microinstruction word is then read from MEM 10112 through JPD Bus 10142 and transferred into the second half of that first address within the WCS portion of EUSITT 20344. The address in SCAG 25536 is then incremented to select a next address within EUSITT 20344 and the process just described repeated automatically, including generation of DRDY and LDWCS, until loading of EUSITT 20344 is completed.

After loading of EUSITT 20344 is completed, the loading process is terminated when FU 10120 asserts NINTP, or DP 10118 asserts Control Signal Load Complete (LOADCR). Either NINTP or LOADCR releases control of operation of NAG 20340 to allow EU 10122 to resume normal operation.

The above descriptions have described structure and operation of EU 10122, including: execution of various arithmetic operations utilizing various operand formats; operation of EU 10122, FU 10120, and MEM 10112 with regard to handshaking; loading of EUDPs and operands; storeback of results; checking of test conditions and exceptions; EU 10122 Stack Mechanisms during normal and kernel operations; and loading of EU 10122 microinstruction sequences into EUSITT 20344. IOS 10116 and DP 10118 will be described next below, in that order.

D. I/O System 10116 (Figs. 204, 206, 269)

Referring to Fig. 204, a partial block diagram of IOS 10116 is shown. As previously described, IOS 10116 operates as an interface between CS 10110 and the external world, for example, ED 10124. A primary function of IOS 10116 is the transfer of data between CS 10110, that is MEM 10112, and the external world. In addition to performing transfers of data, IOS 10116 controls access between various data sources and sinks of ED 10124 and MEM 10112. As previously described, IOS 10116 directly addresses MEM 10112's physical address space to write data into or read data from MEM 10112. As such, IOS 10116 also performs address translation, a mapping operation required in transferring data between MEM 10112's physical

address space and address spaces of data sources and sinks in ED 10124.

As shown in Fig. 204, IOS 10116 includes Data Mover (DMOVR) 20410, Input/Output Control Processor (IOCP) 20412, and one or more data channel devices. IOS 10116's data channel devices may include ECLIPSE® Burst Multiplexer Channel (EBMC) 20414, NOVA Data Channel (NDC) 20416, and other data channel devices as required for a particular configuration of a CS 10110 system. IOCP 20412 controls and directs transfer of data between MEM 10112 and ED 10124, and controls and directs mapping of addresses between ED 10124 and MEM 10112's physical address space. IOCP 20412 may be comprised, for example, of a general purpose computer, such as an ECLIPSE® M600 computer available from Data General Corporation of Westboro, Massachusetts.

EBMC 20414 and NDC 20416 comprise data channels through which data is transferred between ED 10124 and IOS 10116. EBMC 20414 and NDC 20416 perform actual transfers of data to and from ED 10124, under control of IOCP 20412, and perform mapping of ED 10124 addresses to MEM 10112 physical addresses, also under control of IOCP 20412. EBMC 20414 and NDC 20416 may respectively be comprised, for example, of an ECLIPSE® Burst Multiplexer Data Channel and a NOVA® Data Channel, also available from Data General Corporation of Westboro, Massachusetts.

DMOVR 20410 comprises IOS 10116's interface to MEM 10112. DMOVR 20410 is the path through which data and addresses are transferred between EBMC 20414 and NDC 20416 and MEM 10112. Additionally, DMOVR 20410 controls access between EBMC 20414, NDC 20416, and other IOS 10116 data channels, and MEM 10112.

ED 10124, as indicated in Fig. 204, may be comprised of one or more data sinks and sources. ED 10124 data sinks and sources may include commercially available disc drive units, line printers, communication lengths, tape units, and other computer systems, including other CS 10110 systems. In general, ED 10124 may include all such data devices as are generally interfaced with a computer system.

a. I/O System 10116 Structure (Fig. 204)

Referring first to the overall structure of IOS 10116, data input/output of ECLIPSE® Burst Multiplexer Channel Adapter and Control Circuitry (BMCAC) 20418 of EBMC 20414 is connected to bi-directional BMC Address and Data (BMCAD) Bus 20420. BMCAD Bus 20420 in turn is connected to data and address inputs and outputs of data sinks and sources of ED 10124.

Similarly, data and address inputs and outputs of NOVA® Data Channel Adapter Control Circuits (NDCAC) 20422 in NDC 20416 is connected to bi-directional NOVA® Data Channel Address and Data (NDCAD) Bus 20424. NDCAD Bus 20424 in turn is connected to address and data inputs and outputs of data sources and sinks of ED 10124. BMCAD Bus 20420 and NDCAD Bus 20424 are paths for transfer of data and addresses between data sinks and sources of ED 10124 and IOS 10116's data channels and may be expanded as required to include other IOS 10116 data channel devices and other data sink and source devices of ED 10124.

Within EBMC 20414, bi-directional data input and output of BMCAC 20418 is connected to bi-directional input and output of BMC Data Buffer (BMCDB) 20426. Data inputs and data outputs of BMCDB 20426 are connected to, respectively, Data Mover Output Data (DMOD) Bus 20428 and Data Mover Input Data (DMID) Bus 20430. Address outputs of BMCAC 20418 are connected to address inputs of Burst Multiplexer Channel Address Translation Map (BMCATM) 20432 and address outputs of BMCATM 20432 are connected onto DMID Bus 20430. A bi-directional control input and output of BMCATM 20432 is connected from bi-directional IO Control Processor Control (IOCPC) Bus 20434.

Referring to NDC 20416, as indicated in Fig. 204 data inputs and outputs of NDCAC 20422 are connected, respectively, from DMOD Bus 20428 and to DMID Bus 20430. Address outputs of NDCAC 20422 are connected to address inputs of NOVA® Data Channel Address Translation Map (NDCATM) 20436. Address outputs of NDCATM 20436 are, in turn, connected onto DMID Bus 20430. A bi-directional control input and output of NDCATM 20436 is connected from IOCPC Bus 20434.

Referring to IOCP 20412, a bi-directional control input and output of IOCP 20412 is connected from IOCPC Bus 20434. Address and data output of IOCP 20412 is connected to NDCAD Bus 20424. An address output of IOCP Address Translation Map (IOCPATM) 20438 within IOCP 20412 is connected onto DMID Bus 20430. Data inputs and outputs of IOCP 20412 are connected, respectively, to DMOD Bus 20428 and DMID Bus 20430. A bi-directional control input and output of IOCP 20412 is connected to a bi-directional control input and output of DMOVR 20410.

Referring finally to DMOVR 20410, DMOVR 20410 includes Input Data Buffer (IDB) 20440, Output Data Buffer (ODB) 20442, and Priority Resolution and Control (PRC) 20444. A data and address input of IDB 20440 is connected from DMID Bus 20430. A data and address output of IDB 20440 is connected to IOM Bus 10130 to MEM 10112. A data output of ODB 20442 is connected from MIO Bus 10129 from MEM 10112, and a data output of ODB 20442 is connected to DMOD Bus 20428. Bi-directional control inputs and outputs of IDB 20440 and ODB 20442 are connected from bi-directional control inputs and outputs of PRC 20444. A bi-directional control input and output of PRC 20444 is connected from a bi-directional control input and output of IOCP 20412 as described above. Another bi-directional control input and output of PRC 20444 is connected to and from IOMC Bus 10131 and thus from a control input and output of MEM 10112. Having described overall structure of IOS 10116, operation of IOS 10116 will be described next below.

b. I/O System 10116 Operation (Fig. 269)

1. Data Channel Devices

Referring first to EBMC 20414, BMCAC 20418 receives data and addresses from ED 10124 through BMCAD Bus 20420. BMCAC 20418 transfers data into BMCDB 20426, where that data is held for subsequent transmission to MEM 10112 through DMOVR 20410, as will be described below. BMCAC 20418 transfers addresses received from ED 10124 to BMCATM 20432. BMCATM 20432 contains address mapping information correlating ED 10124 addresses with MEM 10112 physical addresses. BMCATM 20432 thereby provides MEM 10112 physical addresses corresponding to ED 10124 addresses provided through BMCAC 20418.

When, as will be described further below, EBMC 20414 is granted access to MEM 10112 to write data into MEM 10112, data stored in BMCDB 20426 and corresponding addresses from BMCATM 20432 are transferred onto DMID Bus 20430 to DMOVR 20410. As will be described below, DMOVR 20410 then writes that data into those MEM 10112 physical address locations. When data is to be read from MEM 10112 to ED 10124, data is provided by DMOVR 20410 on DMOD Bus 20428 and is transferred into BMCDB 20426. BMCAC 20418 then reads that data from BMCDB 20426 and transfers that data onto BMCAD Bus 20420 to ED 10124. During transfers of data from MEM 10112 to ED 10124, MEM 10112 does not provide addresses to be translated into ED 10124 addresses to accompany that data. Instead, those addresses are generated and provided by BMCAC 20418.

NDC 20416 operates in a manner similar to that of EBMC 20414 except that data inputs and outputs of NDCAC 20422 are not buffered through a BMCDB 20426.

As previously described, MEM 10112 has capacity to perform block transfers, that is sequential transfers of four 32 bit words at a time. In general, such transfers are performed through EBMC 20414 and are buffered through BMCDB 20426. That is, BMCDB 20426 allows single 32 bit words to be received from ED 10124 by EBMC 20414 and stored therein until a four word block has been received. That block may then be transferred to MEM 10112. Similarly, a block may be received from MEM 10112, stored in BMCDB 20426, and transferred one word at a time to ED 10124. In contrast, NDC 20416 may generally be utilized for single word transfers.

As indicated in Fig. 204, EBMC 20414, NDC 20416, and each data channel device of IOS 10116 each contain an individual address translation map, for example BMCATM 20432 in EBMC 20414 and NDCATM 20436 in NDC 20416. Address translation maps stored therein are effectively constructed and controlled by IOCP 20412 for each data channel device. IOS 10116 may thereby provide an individual and separate address translation map for each IOS 10116 data channel device. This allows IOS 10116 to insure that no two data channel devices, nor two groups of data sinks and sources in ED 10124, will mutually interfere by writing into and destroying data in a common area of MEM 10112 physical address space. Alternately, IOS 10116 may generate address translation maps for two or more data channel devices wherein those maps share a common, or overlapping, area of MEM 10112's physical address space. This allows data stored in MEM 10112 to be transferred between IOS 10116 data channel devices through MEM 10112, and thus to be transferred between various data sink and source devices of ED 10124. For example, a first ED 10124 data source and a first IOS 10116 data channel may write data to be operated upon into a particular area of MEM 10112 address space. The results of CS 10110 operations upon that data may then be written into a common area shared by that first data device and a second data device and read out of MEM 10112 to a second ED 10124 data sink by that second data channel device. Individual mapping of IOS 10116's data channel devices thereby provides total flexibility in partitioning or sharing of MEM 10112's address space through IOS 10116.

2. I/O Control Processor 20412

As described above, IOCP 20412 is a general purpose computer whose primary function is overall direction and control or data transfer between MEM 10112 and ED 10124. IOCP 20412 controls mapping of addresses between IOS 10116's data channel devices and MEM 10112 address space. In this regard, IOCP 20412 generates address translation maps for IOS 10116's data channel devices, such EBMC 20414 and NDC 20416. IOCP 20412 loads these address translation maps into and controls, for example, BMCATM 20432 of EBMC 20414 and NDCATM 20436 and NDC 20416 through IOCPC Bus 20434. IOCP 20412 also provides certain control functions to DMOVR 20410, as indicated in Fig. 204. In addition to these functions, IOCP 20412 is also provided with data and addressing inputs and outputs. These data addressing inputs and outputs may be utilized, for example, to obtain information utilized by IOCP 20412 in generating and controlling mapping of addresses between IOS 10116's data channel devices and MEM 10112. Also, these data and address inputs and outputs allow IOCP 20412 to operate, in part, as a data channel device. As previously described, IOCP 20412 has data and address inputs and outputs connected from and to DMID Bus 20430 and DMOD Bus 20428. IOCP 20412 thus has access to data being transferred between ED 10124 and MEM 10112, providing IOCP 20412 with direct access to MEM 10112 address space. In addition, IOCP 20412 is provided with control and address outputs to NDCAD Bus 20424, thus allowing IOCP 20412 partial control of certain data source and sink devices in ED 10124.

3. Data Mover 20410 (Fig. 269)

a.a. Input Data Buffer 20440 and Output Data Buffer 20442

As described above, DMOVR 20410 comprises an interface between IOS 10116's data channels and MEM 10112. DMOVR 20410 performs actual transfer of data between IOS 10116's data channel devices and MEM 10112, and controls access between IOS 10116's data channel devices and MEM 10112. IDB 20440 and ODB 20442 are data and address buffers allowing asynchronous transfer of data between IOS 10116 and MEM 10112. That is, ODB 20442 may accept data from MEM 10112 as that data becomes available and then hold that data until an IOS 10116 data channel device, for example EBMC 20414, is ready to accept that data. IDB 20440 accepts data and MEM 10112 physical addresses from IOS 10116's data channel devices. IDB 20440 holds that data and addresses for subsequent transmission to MEM 10112 when MEM 10112 is ready to accept data and addresses. IDB 20440 may, for example, accept a burst, or sequence, of data from EBMC 20414 or single data words from NDC 20416 and subsequently provide that data to MEM 10112 in block, or four word, transfers as previously described. Similarly, ODB 20442 may accept one or more block transfers or data from ODB 20442 and subsequently provide that data to NDC 20416 as single words, or to DMID 20430 as a data burst. In addition, as previously described, a block transfer from MEM 10112 may not appear as four sequential words. In such cases, ODB 20442 accepts the four words of a block transfer as they appear on MIO Bus 10129 and assembles those words into a block comprising four sequential words for subsequent transfer to ED 10124.

Transfer of data through IDB 20440 and ODB 20442 is controlled by PRC 20444, which exchanges control signals with IOCP 20412 and has an interface, previously described, to MEM 10112 through IOMC Bus 10131.

b.b. Priority Resolution and Control 20444 (Fig. 269)

As previously described, PRC 20444 controls access between IOS 10116 data channel devices and MEM 10112. This operation is performed by means of a Ring Grant Access Generator (RGAG) within PRC 20444.

Referring to Fig. 270, a diagramic representation of PRC 20444's RGAG is shown. In general, PRC 20444's RGAG is comprised of a Ring Grant Code Generator (RGCG) 26910 and one or more data channel request comparators. In Fig. 269, PRC 20444's RGAC is shown as including ECLIPSE® Burst Multiplexer Channel Request Comparator (EBMCRC)26912, NOVA® Data Channel Request Comparator (NDCRC) 26914, Data Channel Device X Request Comparator (DCDXRC) 26916, and Data Channel Device Z Request Comparator (DCDZRC) 26918. PRC 20444's RGAG may include more or fewer request comparators as required by the number of data channel devices within a particular IOS 10116.

As indicated in Fig. 269, Request Grant Code (RGC) outputs of RGCG 26910 are connected in parallel to first inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918. Second inputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are connected from other portions of PRC 20444 and receive indications that, respectively, EBMC 20414, NDC 20416, DCDX, or DCDZ has submitted a request for a read or write access to MEM 10112.

Request Grant Outputs (GRANT) of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are in turn connected to other portions of PRC 20444 circuitry to indicate when read or write access to MEM 10112 has been granted in response to a request by a particular IOS 10116 data channel device. When indication of such a grant is provided to those other portions of PRC 20444, PRC 20444 proceeds to generate appropriate control signals to MEM 10112, through IOMC Bus 10131 as previously described, to IDB 20440 and ODB 20442, and to IOCP 20412. PRC 20444's control signals initiate that read or write request to that IOS 10116 data channel device. Grant outputs of EBMCRC 26912, NDCRC 26914, DCDXRC 26916, and DCDZRC 26918 are also provided as inputs to RGCG 26910 to indicate, as described further below, when a particular IOS 10116 has requested and been granted access to MEM 10112.

As indicated in Fig. 269, a diagramic figure above RGCG 26910, RGCG generates a repeated sequence of unique RGCs. Herein indicated as numeric digits 0 to 15. Each RGC identifies, or defines, a particular time slot during which a IOS 10116 data channel device may be granted access to MEM 10112. Certain RGCs are, effectively, assigned to particular IOS 10116 data channel devices. Each such data channel device may request access to MEM 10112 during its assigned RGC identified access slots. For example, EBMC 20414 is shown as being allowed access to MEM 10112 during those access slots identified by RGCs 0, 2, 4, 6, 8, 10, 12, and 14. NDC 20416 is indicated as being allowed access to MEM 10112 during RGC slots 3, 7, 11, and 15. DCDX is allowed access during slots 1 and 9, and DCDZ is allowed access during RGC slots 5 and 13.

As described above, RGCG generates RGCs 0 to 15 in a repetitive sequence. During occurrence of a particular RGC, each request comparator of PRC 20444's RGAG examines that RGC to determine whether its associated data channel device is allowed access during that RGC slot, and whether that associated data channel device has requested access to MEM 10112. If that associated data channel device is allowed access during that RGC slot, and has requested access, that data channel device is granted access as indicated by that request comparator's GRANT output. The request comparators GRANT output is also provided as an input to RGCG 26910 to indicate to RGCG 26910 that access has been granted during that RGC slot.

If a particular data channel device has not claimed and has not been granted access to MEM 10112

during that RGC slot, RGCG 26910 will go directly to next RGC slot. In next RGC slot, PRC 20444's RGAG again determines whether the particular data channel device allowed access during that slot has submitted a request, and will grant access if such a request has been made. If not, RGCG 26910 will again proceed directly to next RGC slot, and so on. In this manner, PRC 20444's RGAG insures that each data channel device of IOS 10116 is allowed access to MEM 10112 without undue delay. In addition, PRC 20444's RGAG prevents a single, or more than one, data channel device from monopolizing access to MEM 10112. As described above, each data channel device is allowed access to MEM 10112 at least once during a particular sequence of RGCs. At the same time, by not pausing within a particular RGC in which no request for access to MEM 10112 has occurred, PRC 20444's RGAG effectively automatically skips over those data channel devices which have not requested access to MEM 10112. PRC 20444's RGAG thereby effectively provides, within a given time interval, more frequent access to those data channel devices which are most busy. In addition, the RGCs assigned to particular IOS 10116 data channel devices may be reassigned as required to adapt a particular CS 10110 to the data input and output requirements of a particular CS 10110 configuration. That is, if EBMC 20414 is shown to require less access to MEM 10112 then NDC 20416, certain RGCs may be reassigned from EBMC 20414 to NDC 20416. Access to MEM 10112 by IOS 10116's data channel devices may thereby be optimized as required.

Having described structure and operation of IOS 10116, structure and operation of DP 10118 will be described next below.

E. Diagnostic Processor 10118 (Figs. 101, 205)

Referring to Fig. 101, as previously described, DP 10118 is interconnected with IOS 10116, MEM 10112, FU 10120, and EU 10122 through DP Bus 10138. DP 10118 is also interconnected, through DPIO Bus 10136, with the external world and in particular with DU 10134. In addition to performing diagnostic and fault monitoring and correction operations, DP 10118 operates, in part, to provide control and display functions allowing an operator to interface with CS 10110. DU 10134 may be comprised, for example, of a CRT and keyboard unit, or a teletype, and provides operators of CS 10110 with all control and display functions which are conventionally provided by a hard console, that is a console containing switches and lights. For example, DU 10134, through DP 10118, allows an operator to exercise control of CS 10110 for such purposes as system initialization and startup, execution of diagnostic processes, fault monitoring and identification, and control of execution of programs. As will be described further below, these functions are accomplished through DP 10118's interfaces with IOS 10116, MEM 10112, FU 10120, and EU 10122.

DP 10118 is a general purpose computer system, for example a NOVA® 4 computer of Data General Corporation of Westboro, Massachusetts. Interface of DP 10118 and DU 10134, and mutual operation of DP 10118 and DU 10134, will be readily apparent to one of ordinary skill in the art. DP 10118's interface and operation, with IOS 10116, MEM 10112, FU 10120, and EU 10122 will be described further next below.

DP 10118, operating as a general purpose computer programmed specificially to perform the functions described above, has, as will be described below, read and write access to registers of IOS 10116, MEM 10112, FU 10120 and EU 10122 through DP Bus 10138. DP 10118 may read data directly from and write data directly into those registers. As will be described below, these registers are data and instruction registers and are integral parts of CS 10110's circuitry during normal operation of CS 10110. Access to these registers thereby allows DP 10118 to directly control or effect operation of CS 10110. In addition, and as also will be described below, DP 10118 provides, in general, all clock signals to all portions of CS 10110 circuitry and may control operation of that circuitry through control of these clock signals.

For purposes of DP 10118 functions, CS 10110 may be regarded as subdivided into groups of functionally related elements, for example DESP 20210 in FU 10120. DP 10118 obtains access to the registers of these groups, and control of clocks therein, through scan chain circuits, as will be described next below. In general, DP 10118 is provided with one or more scan chain circuits for each major functional sub-element of CS 10110.

Referring to Fig. 205, a diagramic representation of DP 10118 and a typical DP 10118 scan chain is shown. As indicated therein, DP 10118 includes a general purpose Central Processor Unit, or computer, (DPCPU) 27010. A first interface of DPCPU 27010 is with DU 10134 through DPIO Bus 10136. DPCPU 27010 and DU 10134 exchange data and control signals through DPIO Bus 10136 in the manner to direct operations of DPCPU 27010, and to display the results of those operations through DU 10134.

Associated with DPCPU 27010 is Clock Generator (CLKG) 27012. CLKG 27012 generates, in general, all clock signals used within CS 10110.

DPCPU 27010 and CLKG 27012 are interfaced with the various scan chain circuits of CS 10110 through DP Bus 10138. As described above, CS 10110 may include one or more scan chains for each major sub-element of CS 10110. One such scan chain, for example DESP 20210 Scan Chain (DESPSC) 27014 is illustrated in Fig. 205.

Interface between DPCPU 27010 and CLKG 27012 and, for example, DESPSC 27014 is provided through DP Bus 10138. As indicated in Fig. 205, DESPSC 27014 includes Scan Chain Clock Gates (SCCG) 27016 and one or more Scan Chain Registers (SCRs) 27018 to 27024.

SCCG 27016 receives clock signals from CLKG 27012 and control signals from DPCPU 27010 through DP Bus 10138. SCCG 27016 in turn provides appropriate clock signals to the various registers and circuits

of, for example, DESP 20210. Clock control signals provided by DPCPU 27010 to SCCG 27016 control, or gate, the various clock signals to these registers and circuits of DESP 20210, thereby effectively allowing DPCPU 27010 to control of DESP 20210.

SCRs 27018 to 27024 are comprised of various registers within DESP 20210. For example, SCRs 27018 to 27024 may include the output buffer registers of AONGRF 20232, OFFGRF 20234, LENGRF 20236, output registers of OFFALU 20242 and LENALU 20252, and registers within OFFMUX 20240 and BIAS 20246. Such registers are indicated in the present description, as previously described, by arrows appended to ends of those registers, with a first arrow indicating an input and a second an output. In normal CS 10110 operations, as previously described, SCRs 27018 to 27024 operate as parallel in, parallel out buffer registers through which data and instructions are transferred. SCRs 27018 to 27024 are also capable of operating as shift registers and, as indicated in Fig. 205, are connected together to comprise a single shift register circuit having an input from DPCPU 27010 and an output to DPCPU 27010. Control inputs to SCRs 27018 to 27024 from DPCPU 27010 control operation of SCRs 27018 to 27024, that is whether these registers shall operate as parallel in, parallel out registers, or as shift registers of DESPSC 27014's scan chain. The shift register scan chain comprising SCRs 27018 to 27024 allows DPCPU 27010 to read the contents of SCRs 27018 to 27024 by shifting the content of these registers into DPCPU 27010. Conversely, DPCPU 27010 may write into SCRs 27018 to 27024 by shifting information generated by DPCPU 27010 from DPCPU 27010 and through the shift register scan chain to selected locations within SCRs 27018 to 27024.

Scan chain clock generator circuits and scan chain registers of each scan chain circuit within CS 10110 thereby allow DP 10118 to control operation of each major sub-element of CS 10110. For example, to read information from the scan chain registers therein, and to write information into those scan chain registers as required for diagnostic, monitoring, and control functions.

Having described structure and operation of each major element of CS 10110, including MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, certain operations of, in particular, FU 10120 will be described further next below. The following descriptions will further disclose operational features of JP 10114, and in particular FU 10120, by describing in greater detail certain operations therein by further describing microcode control of JP 10114.

F. CS 10110 Micromachine Structure and Operation (Figs. 270—274)

a. Introduction

The preceding descriptions have presented the hardware structures and operation of FU 10120 and EU 10122. The following description will describe how devices in FU 10120, and certain EU 10122 devices, function together as a microprogrammable computer, henceforth termed the FU micromachine. The FU micromachine performs two tasks: it interprets SINs, and it responds to certain signals generated by devices in FU 10120, EU 10122, MEM 10112, and IOS 10116. The signals to which the FU micromachine responds are termed Event signals. In terms of structure and operation, the FU micromachine is characterized by the following:

- Registers and ALUs specialized for the handling of logical descriptors.
- Registers organized as stacks for invocations of microroutines (microinstruction sequences).
- Mechanisms allowing microroutine invocations by means of event signals from hardware.
- Mechanisms which allow an invoked microroutine to return either to the microinstruction following the one which resulted in the invocation or to the microinstruction which resulted in the invocation.
- Mechanisms which allow the contents of stack registers to be transferred to MEM 10112, thereby creating a virtual microstack of limitless size.
- Mechanisms which guarantee response to an event signal within a predictable length of time.
- The division of the devices comprising the micromachine into two groups: those devices which may be used by all microcode and those which may be used only by KOS (Kernel Operating System, previously described) microcode.

These devices and mechanisms allow the FU micromachine to be used in two ways: as a virtual micromachine and as a monitor micromachine. Both kinds of micromachine use the same devices in FU 10120, but perform different functions and have different logical properties. In the following discussion, when the FU micromachine is being used as a virtual micromachine, it is said to be in virtual mode, and when it is being used as a monitor micromachine, it is said to be in monitor mode. Both modes are introduced here and explained in detail later.

When the FU micromachine is being used in virtual mode, it has the following properties:

- It runs on an essentially infinite micromachine stack belonging to a Process 610.
- It can respond to any number of event signals in the M0 cycle (state) of a single microinstruction.
- A page fault may occur on the invocation of any microroutine or on return from any microroutine.
- When the FU micromachine is in virtual mode, any microroutine may not run to completion, i.e., complete its execution in a predictable length of time, or complete it at all.
- It is executing a Process 610.

The last four properties are consequences of the first: Event signals result in invocations, and since the micromachine stack is infinite, there is no limit to the number of invocations. The infinite micromachine stack is realized by placing micromachine stack frames on Secure Stack 10336 belonging to a Process 610,

and the virtual micromachine therefore always runs on a micromachine stack belonging to some Process 610. Furthermore, if the invocation of a microroutine or a return from a microroutine requires micromachine frames to be transferred from Secure Stack 10336 to the FU micromachine, a page fault may result, and Process 610 which is executing the microroutine may be removed from JP 10114, thereby making the time required to execute the microroutine unpredictable. Indeed, if process 610 is stopped or killed, the execution of the microroutine may never finish. As will be seen in descriptions below, the Virtual Processor 612 is the means by which the virtual micromachine gains access to a Process 610's micromachine stack.

When in monitor mode, the FU micromachine has the following properties:

- It has a micromachine stack of fixed size, the stack is always available to the FU micromachine, and it is not associated with a Process 610.
- It can respond to only a fixed number of events during the M0 cycle of a single microinstruction.
- In monitor mode, invocation of a microroutine or return from a microroutine will not cause a page fault.
- Microroutines executing on the FU micromachine when the micromachine is in monitor mode are guaranteed to run to completion unless they themselves perform an action which causes them to give up JP 10114.
- Microroutines executing in monitor mode need not be performing functions for a Process 610.

Again, the remaining properties are consequences of the first: because the monitor micromachine's stack is of fixed size, the number of events to which the monitor micromachine can respond is limited; furthermore, since the stack is always directly accessible to the micromachine, microroutine invocations and returns will not cause page faults, and microroutines running in monitor mode will run to completion unless they themselves perform an action which causes them to give up JP 10114. Finally, the monitor micromachine's stack is not associated with a Process 610's Secure Stack 10336, and therefore, the monitor micromachine can both execute functions for Processes 610 and execute functions (which are related to no Process 610, for example,) the binding and removal of Virtual Processors 612 from JP 10114.

The description which follows first gives an overview of the devices which make up the micromachine, continues with descriptions of invocations on the micromachine and micromachine programming, and concludes with detailed discussions of the virtual and monitor modes and an overview of the relationship between the micromachine and CS 10110 subsystems. The manner in which the micromachine performs specific operations such as SIN parsing, Name resolution, or address translation may be found in previous descriptions of CS 10110 components which the micromachine uses to perform the operations.

b. Overview of Device Comprising FU Micromachine (Fig. 270)

Fig. 270 presents an overview of the devices comprising the micromachine. Fig. 270 is based on Fig. 201, but has been simplified to improve the clarity of the discussion. Devices and subdivisions of the micromachine which appear in Fig. 201 have the numbers given them in that figure. When a device in Fig. 270 appears in two subdivisions, it is shared by those subdivisions.

Fig. 270 has four main subdivisions. Three of them are from Fig. 201: FUCTL 20214, which contains the devices used to select the next microinstruction to be executed by the micromachine, DESP 20210, which contains stack and global registers and ALUs for descriptor processing; and MEMINT 20212, which contains the devices which translate Names into logical descriptors and logical descriptors into physical descriptors. The fourth subdivision, EU Interface 27007, represents those portions of EU 10122 which may be manipulated by FU 10120 microcode.

Fig. 270 further subdivides FUCTL 20214 and MEMINT 20212. FUCTL 20214 has four subdivisions:

- I-Stream Reader 27001, which contains the devices used to obtain SINs and parse them into SOPs and Names.
- SOP Decoder 27003, which translates SOPs into locations in FU microcode (FUSITT 11012), and in some cases EU microcode (EUSITT 20344), which contain the microcode that performs the corresponding SINs.
- Microcode Addressing 27013, which determines the location of the next microinstruction to be executed in FUSITT 11012.
- Register Addressing 27011, which contains devices which generate addresses for GRF 10354 registers.

MEMINT 20212 also has three subdivisions:

- Name Translation Unit 27015, which contains devices which accelerate the translation of Names into logical descriptors.
- Memory Reference Unit 27017, which contains devices which accelerate the translation of logical descriptors into physical descriptors.
- Protection Unit 27019, which contains devices which accelerate primitive access checks on memory references made with logical descriptors.

Fig. 270 also simplifies the bus structure of Fig. 202 by combining LENGTH Bus 20226, OFFSET Bus 20228, and AONR Bus 20230 into a single structure, Descriptor Bus (DB) 27021. In addition, internal bus connections have been reduced to those necessary for explaining the logical operation of the

micromachine. The following discussion first describes those devices used by most microcode executing on FU 10120, and then describes devices used to perform special functions, such as Name translation or protection checking.

1. Devices used by Most Microcode

The subdivisions of the micromachine which contain devices used by most microcode are Microcode Addressing 27013, Register Addressing 27011, DESP 20210, and EU Interface 27007. In addition, most microcode uses MOD Bus 10144, JPD Bus 10142, and DB Bus 27021. The discussion begins with the buses and then describes the other devices in the above order.

a.a. MOD Bus 10144, JPD Bus 10142, and DB Bus 27021

MOD Bus 10144 is the only path by which data may be obtained from MEM 10112. Data on MOD Bus 10144 may have as its destination Instruction Stream Reader 27001, DESP 20210, or EU Interface 27007. In the first case, the data on MOD Bus 10144 consists of SINs; in the second, it is data to be processed by FU 10120, and in the third, it is data to be processed by EU 10122. In the present embodiment, data to be processed by FU 10120 is generally data which is destined for internal use in FU 10120, for example in Name Cache 10226. Data to be processed by EU 10122 is generally operands represented by Names in SINs.

JPD Bus 10142 has two uses: it is the path by which data returns to MEM 10112 after it has been processed by JP 10114, and it is the path by which data other than logical descriptors moves between the subdivisions of the micromachine. For example, when CS 10110 is initialized, the microinstructions which are loaded into FUSITT 11012 are transferred from MEM 10112 to DESP 20210 via MOD Bus 10144, and from DESP 20210 to FUSITT 11012 via JPD Bus 10142.

DB 27021 is the path by which logical descriptors are transferred in the micromachine. DB 27021 connects Name Translation Unit 27015, DESP 20210, Protection Unit 27019, and Memory Reference Unit 27017. Typically, a logical descriptor is obtained from Name Translation Unit 27015, placed in a register in DESP 20210, and then presented to Protection Unit 27019 and Memory Reference Unit 27017 whenever a reference is made using a logical descriptor. However, DB 27021 is also used to transmit cache entries fabricated in DESP 20210 to ATU 10228, Name Cache 10226 and Protection Cache 10234.

b.b. Microcode Addressing

As discussed here, microcode addressing is comprised of the following devices: Timers 20296, Event Logic 20284, RCWS 10358, BRCASE 20278, mPC 20276, MCW0 20292, MCW1 20290, SITTNAS 20286, and FUSITT 11012. All of these devices have already been described in detail, and they are discussed here only as they affect microcode addressing. Other devices contained in Fig. 202, State Registers 20294, Repeat Counter 20280, and PNREG 20282 are not directly relevant to microcode addressing, and are not discussed here.

As has already been described in detail, devices in Microcode Addressing 27013 are loaded from JPD Bus 10142. The microcode addresses provided by these devices and by FUSDT 11010 are transmitted among the devices and to FUSITT 11012 by CSADR Bus 20204. There are six ways in which the next microcode address may be obtained:

- Most commonly, the value in mPC 20276 is incremented, by 1 by a special ALU in mPC 20276, thus yielding the address of the microinstruction following the current microinstruction.
- If a microinstruction specifies a call to a microroutine or a branch, the microinstruction contains a literal which an ALU in BRCASE 20278 adds to the value in mPC 20276 to obtain the location of the next microinstruction.
- If a microinstruction specifies the use of a case value to calculate the location of the next microinstruction, BRCASE 20278 adds a value calculated by DESP 20210 to the value in mPC 20716. The value calculated by DESP 20210 may be obtained from a field of a logical descriptor, thus allowing the micromachine to branch to different locations in microcode on the basis of type information contained in the logical descriptor. On return from an invocation of a microroutine, the location at which execution of the microroutine in which the invocation occurred is to continue is obtained from RCWS 10358.
- At the beginning of the execution of an SIN, the location at which the microcode for the SIN begins is obtained from the SIN's SOP by means of FUSDT 11010.
- Certain hardware signals cause invocations of microroutines. There are two classes of such signals: Event signals, which Event Logic 20284 transforms into invocations of certain microroutines, and JAM signals, which are translated directly into locations in microcode.

The addresses obtained as described above are transmitted to SITTNAS 20286, which selects one of the addresses as the location of the next microinstruction to be executed and transmits the location to FUSITT 11012. As the location is transmitted to FUSITT 11012, it is also stored in mPC 20276. All addresses except those for Jams are tranferred to SITTNAS 20286 via CSADR Bus 20204. Addresses obtained from

JAM signals are transferred by separate lines to SITTNAS 20286.

As will be explained in detail below, microroutine calls and returns also involve pushing and popping micromachine stack frames and saving state contained in MCW1 20290.

Register Addressing 27011 controls access to micromachine registers contained in GRF 10354. As explained in detail below, GRF 10354 contains both registers used for the micromachine stack and global registers, that is, registers that are always accessible to all microroutines. The registers are grouped in frames, and individual registers are addressed by frame number and register number. Register Addressing 27011 allows addressing of any frame and register in the GRs 10360 of GRF 10354, but allows addressing of registers in only three frames of the SR's 10362: the current (top) frame, the previous frame (i.e., the frame preceding the top frame), and the bottom frame, that is, the lowest frame in a virtual micromachine stack which is still contained in GRF 10354. The values provided by Register Addressing 27011 are stored in MCW0 20292. As will be explained in the discussion of microroutine invocations which follows, current and previous are incremented on each invocation and decremented on each return.

c.c. Description Processor 20210 (Fig. 271)

DESP 20210 is a set of devices for storing and processing logical descriptors. The internal structure of DESP 20210's processing devices has already been explained in detail; here, the discussion deals primarily with the structure and contents of GRF 10354. In a present embodiment of CS 10110, GRF 10354 contains 256 registers. Each register may contain a single logical descriptor. Fig. 271 illustrates a Logical Descriptor 27116 in detail. In a present embodiment of CS 10110, a Logical Descriptor 27116 has four main fields:

- RS Field 27101, which contains various flags which are explained in detail below.
- AON Field 27111, which contains the AON portion of the address of the data item represented by the Logical Descriptor 27116.
- OFF Field 27113, which contains the offset portion of the address of the data item represented by Logical Descriptor 27116.
- LEN Field 27115, which contains the length of the data item represented by the Logical Descriptor 27116.

RS Field 27101 has subfields as follows:

- RTD Field 27103 and WTD Field 27105 may be set by microcode to disable certain Event signals provided for debuggers by CS 10110. For details, see a following description of debugging aids in CS 10110.
- FIU Field 27107 contains two bits. The fields are set from information in the Name Table Entry used to construct the Logical Descriptor 27116. The bits determine how the data specified by the Logical Descriptor 27116 is to be justified and filled when it is fetched from MEM 10112.
- TYPE Field 27109's four bits are also obtained from the Name Table Entry used to construct the Logical Descriptor 27116. The field's settings vary from S-Language to S-Language, and are used to communicate S-Language-specific type information to the S-Language's S-Interpreter microcode.

The four fields of a Logical Descriptor 27116 are contained in three separately-accessible fields in a GRF 10354 register: one containing RS Field 27101 and AON Field 27111, one containing OFF Field 27113, and one containing LEN Field 27115. In addition, each GRF 10354 register may be accessed as a whole. GRF 10354 is further subdivided into 32 frames of eight registers each. An individual GRF 10354 register is addressed by means of its frame number and its register number within the frame. In a present embodiment of CS 10110, half of the frames in GRF 10354 belong to SR's 10362 and are used for micromachine stacks, and half belong to GRs 10360 for storing "global information". In SR's 10362, each GRF 10354 frame contains information belonging to a single invocation of a microroutine. As previously explained, Register Addressing 27011 allows addressing of only three GRF 10354 frames in SR's tack 10362, the current top frame in the stack, the previous frame, and the bottom frame. Registers are accessed by specifying one of these three frames and a register number.

The global information contained in GRs 10360, is information which is not connected with a single invocation. There are three broad categories of global information:

- Information belonging to Process 610 whose Virtual Processor 612 is currently bound to JP 10114. Included in this information are the current values of Process 610's ABPs and the pointers which KOS uses to manage Process 610's stacks.
- Information required for the operation of KOS. Included in this information are such items as pointers to KOS data bases which occupy fixed locations in MEM 10112.
- Constants, that is, fixed values required for certain frequently performed operations in FU 10120.

Remaining registers are available to microprogrammers as temporary storage areas for data which cannot be stored in a microroutine's stack frame. For example, data which is shared by several microroutines may best be placed in a GR 10360. Addressing of registers in the GRs 10360 of GRF 10354 requires two values: a value of 0 through 15 to specify the frame and a value of 0 through 7 to specify the register in the frame.

As previously discussed in detail, each of the three components AONP 20216, OFFP 20218, and LENP 20220 of DESP 20210 also contains ALUs, registers, and logic which allows operations to be performed on individual fields of GRF 10354 registers. In particular, OFFP 20218 contains OFFALU 20242, which may be

used as a general purpose 32 bit arithmetic and logical unit. OFFALU 20242 may further serve as a source and destination for JPD Bus 10142, the offset portion of DB 27021, and NAME Bus 20224, and as a destination for MOD Bus 10144. Consequently, OFFALU 20242 may be used to perform operations on data on these buses and to transfer data from one bus to another. For example, when an SIN contains a literal value used in address calculation, the literal value is transferred via NAME Bus 20224 to OFFALU 20242, operated on, and output via the offset portion of DB 27021.

d.d. EU 10122 Interface

FU 10120 specifies what operation EU 10122 is to perform, what operands it is to perform it on, and when it is finished, what is to be done with the operands. FU 10120 can use two devices in EU 10122 as destinations for data, and one device as a source for data. The destinations are COMQ 20342 and OPB 20322. COMQ 20342 receives the location in EUSITT 20344 of the microcode which is to perform the operation desired by the FU 10120. COMQ 20342 may receive the location in microcode either from an FU 10120 microroutine or from an SIN's SOP. In the first case, the location is transferred via JPD Bus 10142, and in the second, it is obtained from EUSDT 20266 and transferred via EUDIS Bus 20206. OPB 20322 receives the operands upon which the operation is to be performed. If the operands come directly from MEM 10112, they are transferred to OPB 20322 via MOD Bus 10144; if they come from registers or devices in FU 10120, they are transferred via JPD Bus 10142.

Result Register 27013 is a source for data. After EU 10122 has completed an operation, FU 10120 obtains the result from Result Register 27013. FU 10120 may then place the result in MEM 10112 or in any device accessible from JPD Bus 10142.

2. Specialized Micromachine Devices

Each of the groups of specialized devices serves one of CS 10110's subsystems. I-Stream Reader 27001 is part of the S-Interpreter subsystem, Name Translation Unit 27015 is part of the Name Interpreter subsystem, Memory Reference Unit 27017 is part of the Virtual Memory Management System, and Protection Unit 27019 is part of the Access Control System. Here, these devices are explained only in the context of the micromachine; for a complete understanding of their functions within the subsystems to which they belong, see previous descriptions of the subsystems.

a.a. I-Stream Reader 27001

I-Stream Reader 27001 reads and parses a stream of SINs (termed the I-Stream) from a Procedure Object 604, 606, 608. The I-Stream consists of SOPs (operation codes), Names, and literals. As previously mentioned, in a present embodiment of CS 10110, the I-Stream read from a given Procedure 602 has a fixed format: the SOPs are 8 bits long and the Names and literals all have a single length. Depending on the procedure, the length may be 8, 12, or 16 bits. I-Stream Reader 27001 parses the I-Stream by breaking it up into its constituent SOPs and Names and passing the SOPs and Names to appropriate parts of the micromachine. I-Stream Reader 27001 contains two groups of devices:

— PC Values 27006, which is made up of three registers which contain locations in the I-Stream. When added to ABP PBP, the values contained in these registers specify locations in Procedure Object 901 containing the Procedure 602 being executed. CPC 20270 contains the location of the SOP or Name currently being interpreted; IPC 20272 contains the location of the beginning of the SIN currently being executed; EPC 20274, finally, is of interest only at the beginning of the execution of an SIN; at that time, it contains the location of the last SIN to be executed.

— Parsing Unit 27005, which is made up of INSTB 20262, PARSER 20264, and PREF 20260. The micromachine uses PREF 20260 to create Logical Descriptors 27116 for the I-Stream, which are then placed on DB Bus 27021 and used in logical memory references. The data returned from these references is placed in INSTB 20262, and parsed by PARSER 20264.

SOPs, Names, and literals obtained by PARSER 20264 are placed on NAME Bus 20224, which connects PARSER 20264, SOP Decoder 27003, Name Translation Unit 27015, and OFFALU 20242.

b.b. SOP Decoder 27003

SOP Decoder 27003 decodes SOPs into locations in FU 10120 and EU 10122 microcode. SOP Decoder 27003 comprises FUSDT 11010, EUSDT 20266, Dialect Register (RDIAL) 24212, and LOPDCODE 24210. FUSDT 11010 are further comprised of FUDISP 24218 and FALG 24220. The manner in which these devices translate SOPs contained in SINs into locations in FUSITT 11012 and EUSITT 20344 has been previously described.

c.c. Name Translation Unit 27015

Name Translation Unit 27015 accelerates the translation of Names into Logical Descriptors 27116. This operation is termed name resolution. It is comprised of two components: NC 10226 and Name Trap 20254. NC 10226 contains copies of information from a Procedure Object 604's Name Table 10350, and thereby makes it possible to translate Names into Logical Descriptors 27116 without referring to Name Table 10350. When a Name is presented to Name Translation Unit 27015, it is latched into Name Trap 20254 for later use by Name Translation Unit 27015 if required. As will be explained in detail later, in the present embodiment,

Name translation always begins with the presentation of a Name to NC 10226. If the Name has already been translated, the information required to construct its Logical Descriptor 27116 may be contained in NC 10226. If there is no information for the Name in NC 10226, Name Resolution Microcode obtains the Name from Name Trap 20254, uses information from Name Table 10350 for the procedure being executed to translate the Name, places the required information in NC 10226, and attempts the translation again. When the translation succeeds, a Logical Descriptor 27116 corresponding to the Name is produced from the information in Name Cache 10115, placed on DB Bus 27021, and loaded into a GRF 10354 register.

d.d. Memory Reference Unit 27017

Memory Reference Unit 27017 performs memory references using Logical Descriptors 27116. Memory Reference Unit 27017 receives a command for MEM 10112 and a Logical Descriptor 27116 describing the data upon which the command is to be performed. In the case of a write operation, Memory Reference Unit 27017 also receives the data being written via JPD Bus 10142. Memory Reference Unit 27017 translates Logical Descriptor 27116 to a physical descriptor and transfers the physical descriptor and the command to MEM 10112 via PD Bus 10146. A Memory Reference Unit 27017 has four components: ATU 10228, which contains copies of information from KOS virtual memory management system tables, and thereby accelerates logical-to-physical descriptor translation; Descriptor Trap 20256, which traps Logical Descriptors 27116, Command Trap 27018, which traps memory commands; and Data Trap 20258, which traps data on write operations. When a logical memory reference is made, a Logical Descriptor 27116 is presented via DB Bus 27021 to ATU 10228, and at the same time, Logical Descriptor 27116 and the memory command are trapped in Descriptor Trap 20256 and Command Trap 27018. On write operations, the data to be written is trapped in Data Trap 20258. If the information needed to form the physical descriptor is present in ATU 10228, the physical descriptor is transferred to MEM 10112 via PD Bus 10146. If the information needed to form the physical descriptor is not present in ATU 10228, an Event Signal from ATU 10228 invokes a microroutine which retrieves Logical Descriptor 27116 from Descriptor Trap 20256 and uses information contained in KOS virtual memory management system tables to make an entry in ATU 10228 for Logical Descriptor 27116. When the microroutine returns, the logical memory reference is repeated using Logical Descriptor 27116 from Descriptor Trap 20256, the memory command from Command Trap 27018, and on write operations, the data in Data Trap 20258. As will be described in detail in the discussion of virtual memory management, if the data referenced by a logical memory reference is not present in MEM 10112, the logical memory reference causes a page fault.

e.e. The Protection Unit 27019

On each logical memory reference, Protection Unit 27019 checks whether the subject making the reference has access rights which allow it to perform the action specified by the memory command on the object being referenced. If the subject does not have the required access rights, a signal from Protection Unit 27019 causes MEM 10112 to abort the logical memory reference. Protection Unit 27019 consists of Protection Cache 10234, which contains copies of information from KOS Access Control System tables, and thereby speeds up protection checking, and shares Descriptor Trap 20256, Command Trap 27018, and Data Trap 20258 with Memory Reference Unit 27017. When a logical memory reference is made, the AON and offset portions of the logical descriptor are presented to Protection Cache 10234. If Protection Cache 10234 contains protection information for the object specified by the AON and offset and the subject performing the memory reference has the required access, the memory reference may continue; if Protection Cache 10234 contains protection information and the subject does not have the required access, a signal from Protection Cache 10234 aborts the memory reference. If Protection Cache 10234 does not contain the required access information, a signal from Protection Cache 10234 aborts the memory reference and invokes a microroutine which obtains the access information from KOS Access Control System tables and places it in Protection Cache 10234. When Protection Cache 10234 is ready, the memory access is repeated, using the logical descriptor from Descriptor Trap 20256, the memory command from Command Trap 27018, and in the case of write operations, the data in Data Trap 20258.

f.f. KOS Micromachine Devices

As mentioned in the above introduction to the micromachine, the devices making up the micromachine may be divided into two classes: those which any microcode written for the micromachine may manipulate, and those which may be manipulated exclusively by KOS microcode. The latter class consists of certain registers in GRs 10360 of GRF 10354, the bottom frame of the portion of the virtual micromachine stack in the stack portion (Stack Registers 10362) of GRF 10354, and the devices contained in Protection Unit 27019 and Memory Reference Unit 27017. Because Protection Unit 27019 and Memory Reference Unit 27017 may be manipulated only by KOS microcode, non-KOS microcode may not use Descriptor Trap 20256 or Command Trap 27018 as a source or destination, may not load or invalidate registers in ATU 10228 or Protection Cache 10234, and may not perform physical memory references, i.e., memory references which place physical descriptors directly on PD Bus 10146, instead of presenting logical

descriptors to Memory Reference Unit 27017 and Protection Unit 27019. Similarly, non-KOS microcode may not specify KOS registers in the GRs 10360 of GRF 10354 or the bottom frame of the stack portion of GRF 10354 when addressing GRF 10354 registers. Further, in embodiments allowing dynamic loading of FUSITT 11012, only KOS microcode may manipulate the devices provided for dynamic loading.

In a present embodiment of CS 10110, the distinction between KOS devices and registers and devices and registers accessible to all microprograms is maintained by the microbinder. The microbinder checks all microcode for microinstructions which manipulate devices in Protection Unit 27019, or Memory Reference Unit 27017, or which address GRF 10354 registers reserved for KOS use. However, it is characteristic of the micromachine that KOS devices are logically and physically separate from devices accessible to all microprograms and, consequently, other embodiments of CS 10110 may use hardware devices to prevent non-KOS microprograms from manipulating KOS devices.

c. Micromachine Stacks and Microroutine Calls and Returns (Figs. 272, 273)

1. Micromachine Stacks (Fig. 272)

As previously mentioned, the FU micromachine is a stack micromachine The properties of the FU micromachine's stack depends on whether the FU micromachine is in virtual or monitor mode. In virtual mode, the micromachine stack is of essentially unlimited size; if it contains more frames than allowed for inside FU 10120, the top frames are in GRF 10354 and the remaining frames are in Secure Stack 10336 belonging to Process 610 being executed by the FU micromachine. In the following, the virtual mode micromachine stack is termed the virtual micromachine stack. In monitor mode, the micromachine stack consists of a fixed amount of storage; in a present embodiment of CS 10110, the monitor mode micromachine stack is completely contained in the stack portion, SRs 10362, of GRF 10354; in other embodiments of CS 10110, part or all of the monitor mode micromachine stack may be contained in an area of MEM 10112 which has a fixed size and a fixed location known to the monitor micromachine. In yet other embodiments of CS 10110, monitor mode micromachine stack may be of flexible depth in a manner similar to the virtual micromachine stack. In either mode, microroutines other than certain KOS microroutines which execute state save and restore operations may access only two frames of GRF 10354 stack: the frame upon which the microroutine is executing, called the current frame, and the frame upon which the microroutine that invoked that microroutine executed, called the previous frame. KOS microroutines which execute state save and restore operations may in addition access the bottom frame of that portion of the virtual micromachine stack which is contained in GRF 10354.

Fig. 272 illustrates stacks for the FU micromachine. Those portions of the micromachine stack which are contained in the FU are contained in SR's 10362 (of GRF 10354) and in RCWS 10358. Each register of RCWS 10358 is permanently associated with a GRF frame in SRs 10362 of GRF 10354, and the RCWS 10358 register and the GRF frame together may contain one frame of a micromachine stack. As previously describe, each register of GRF 10354 contains three fields: one for an AON and other information, one for an offset, and one for a length. As illustrated in Fig. 251, each register in RCWS 10358 contains four fields:

- A one bit field which retains the value of the Condition Code register in MCW1 20290 at the time that the invocation which created the next frame occurred.
- A field indicating what Event Signals were pending at the time that the invocation to which the RCWS register belongs invoked another microroutine.
- A flag indicating whether the microinstruction being executed when the invocation occurred was the first microinstruction in an SIN.
- The address at which the execution of the invoking microroutine is to continue.

The uses of these fields will become apparent in the ensuing discussion.

The space available for micromachine stacks in SRs 10362 and RCWS 10358 is divided into two parts: Frames 27205 reserved for MOS 10370 and Frames 27206 available for the MIS 27203. Frames 27206 may contain no MIS Frames 27203, or be partially or completely occupied by MIS Frames 27203. Space which contains no MIS Frames is Free Frames 27207. The size of the space reserved for Monitor Micromachine Stack Frames 27205 is fixed, and Spaces 27203, 27205, and 27207 always come in the specified order. Register Addressing 27011 handles addressing in Stack Portion 27201 of GRF 10354 and RCWS 10358 in such fashion that the values for the locations of current, previous, and bottom frames specifying registers in RCWS 10358 or frames in Stack Portion 27201 automatically "wrap around" when they are incremented beyond the largest index value allowed by the sizes of the registers or decremented below the smallest index value. Thus, though Spaces 27203, 27205, and 27207 always have the same relative order, their GRF 10354 frames and RCWS registers may be located anywhere in Stack Portion 27201 and RCWS 10358.

2. Microroutine Invocations and Returns

In CS 10110, microroutines may be invoked by other microroutines or by signals from CS 10110 hardware. The methods of invocation aside, microroutine invocations and returns resemble invocations of and returns from procedures written in high-level languages. In the following, the general principles of microroutine invocations and returns are discussed, and thereafter, the specific methods by which microroutines may be invoked in CS 10110. The differences between invocations in monitor mode and

invocations in virtual mode are explained in the detailed discussions of the two modes.

The microroutine which is currently being executed runs on the frame specified by Current Pointer 27215. When an invocation occurs, either because the executing microroutine performs a call, or because a signal which causes invocations has occurred, JP 10114 hardware does three things:

- It stores state information for the invoking microroutine in the RCWS 10358 register associated with the current frame. The state information includes the location at which execution of the invoking microroutine will resume, as well as other state information.
- It increments Current Pointer 27215 and Previous Pointer 27213, thereby providing a frame for the new invocation.
- It begins executing the first instruction of the newly invoked microroutine.

Because the newly-invoked microroutine can access registers of the invoking microroutine's frame, the invoking microroutine can pass "arguments" to the invoked microroutine by placing values in registers in its frame used by the invoked microroutine. However, the invoking microroutine cannot specify which registers contain "arguments" on an invocation, so the invoked microroutine must know which registers of the previous frame are used by the invoking microroutine. Since the only "arguments" which a microroutine has access to are those in the previous frame, a microroutine can pass arguments which it received from its invoker to a microroutine which it invokes only by copying the arguments from its invoker's frame to its own frame, which then becomes the newly-invoked routine's previous frame.

The return is the reverse of the above: Current Pointer 27215 and Previous Pointer 27213 are decremented, thereby "popping off" the finished invocation's frame and returning to the invoker's frame. The invoker then resumes execution at the location specified in the RCWS 10358 register and using the state saved in the RCWS 10358. The saved state includes the value of the Condition Code in MCW1 20290 at the time of the invocation and flags indicating various pending Events. The Condition Code field in MCW1 20290 is set to the saved value, and the pending event flags may cause Events to occur as described in detail below.

3. Means of Invoking Microroutines

In the micromachine, invocations may be produced either by commands in microinstructions or by hardware signals. In the following, invocations produced by commands in microinstructions are termed Calls, while those produced by hardware signals are termed Event invocations and Jams. Invocations are further distinguished from each other by the locations to which they return. Calls and Jams return to the microinstruction following the microinstruction in which the invocation occurs; Event invocations return to that microinstruction, which is then repeated.

In terms of implementation, the different return locations are a consequence of the point in the micromachine cycle at which Calls, Jams, and Event invocations save a return location and transfer control to the called routine. With Calls and Jams, these operations are performed in the M1 cycle; with Event invocations, on the other hand, the Event signal during the M0 cycle causes the M0 cycle to be followed by a MA cycle instead of the M1 cycle, and the operations are performed in the MA cycle. In the M1 cycle, the value in mPC 20276 is incremented; in the MA cycle, it is not. Consequently, the return value saved in RCWS 10358 on a Call or Jam is the incremented value of mPC 20276, while the return value saved on an Event invocation is the unincremented value of mPC 20276. The following discussion will deal first with Calls and Jams, and then with Event invocations.

A Call command in a microinstruction contains a literal value which specifies the offset from the microinstruction containing the Call at which execution is to continue after the Call. When the microinstruction with the Call command is executed in micromachine cycle M1, BRCASE 20278 adds the offset contained in the command to the current value of mPC 20276 in order to obtain the location of the invoked microroutine and sets SITTNAS 20286 to select the location provided by BRCASE 20278 as the location of the next microinstruction. Then the Call command increments mPC 20276 and stores the incremented value of mPC 20276 in the RCWS 10358 register associated with the current frame in SRs 10362 and increments Current Pointer 27215 and Previous Pointer 27213 to provide a new frame in SRs 10362. The Jam works exactly like the Call, except that a hardware signal during micromachine cycle M1 causes the actions associated with the invocation to occur and provides the location of the invoked microroutine directly to SITTNAS 20286.

With Events, Event Logic 20284 causes an invocation to occur during cycle M0 and provides the location of the invoked microroutine via CSADR 20299. Since the Event occurs during cycle M0, the location stored in RCWS 10358 is the unincremented value of mPC 20276, and SITTNAS 20286 selects the location provided by Event Logic 20284 as the location of the next microinstruction. Since the return from the Event causes the microinstruction during which the Event occurred to be re-executed, the microinstruction and the microroutine to which it belongs may be said to be "unaware" of the Event's occurrence. The only difference between the execution of a microinstruction during which an Event occurs and the execution of the same microinstruction without the Event is the length of time required for the execution.

4. Occurrence of Event Invocations (Fig. 273)

As described previously, Event invocations are produced by Event Logic 20284. The location in microcode to which Event Logic 20284 transfers control is determined by the following:

— The operation being commenced by FU 10120. Certain Event invocations may occur only at the beginning of certain FU 10120 operations.
— The state of Event signal lines from hardware and internal registers in Event Logic 20284.
— The state of certain registers visible via MCW1 20290. Some of these registers enable Events and others mask Events. Of the registers which enable Events, some are set by Event signals and others by the microprogram.
— On returns from invocations of microroutines, the settings of certain bits in the RCWS 10358 register belonging to the micromachine frame for the invocation that is being returned to.

Microprograms may use these mechanisms to disable Event signals and to delay an Event invocation from an Event signal for a single microinstruction or an indefinite period, and FU 10120 uses them to automatically delay Event invocations resulting from certain Event signals. Using traditional programming terminology, the mechanisms allow a differential masking of Event signals. An Event signal may be explicitly masked for a single microinstruction, it may be masked for a sequence of microinstructions; it may be automatically masked until a certain operation occurs, or it may be automatically masked for a certain maximum length of time. Event signals which occur while they are masked are not lost. In some cases, the Event signal continues until it is serviced; in others, a register is set to retain the fact that the Event signal occurred. When the Event signal is unmasked, the set register causes the Event signal to reoccur. In some cases, finally, the Event signal is not retained, but recurs when the microinstruction which caused it is repeated.

In the following, the relationship between FU 10120 operations and Event signals is first presented, and then a detailed discussion of the enabling registers in MCW1 20290 and of the bits in RCWS 10358 registers which control Event invocations.

FU 10120 allows Event invocations resulting from Event signals to be inhibited for a single microinstruction; it also delays certain Event invocations for certain Event signals until the first microinstruction of an SIN. Other Event signals occur only at the beginning of an SIN, at the beginning of a Namespace Resolve or Evaluate operation, or at the beginning of a logical memory reference.

Event invocations may be delayed for a single microinstruction by setting a field of the microinstruction itself. Setting this field delays almost all Event invocations, and thereby guarantees that an Event invocation will not occur during the microinstruction's M0 cycle.

Event signals relating to debugging occur at the beginnings of certain micromachine operations. Such Event signals are called Trace Event signals. As will be explained in detail, in the discussion of the debugger, Trace Event signals can occur on the first microinstruction of an SIN, at the beginning of an Evaluate or Resolve operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. IPM interrupt signals and Interval Timer Overflow Event signals are automatically masked until the beginning of the next SIN or until a maximum amount of time has elapsed, which ever occurs first. The mechanisms involved here are explained in detail in the discussion of interrupt handling in the FU 10120 micromachine.

Turning now to the registers used to mask and enable Event signals, Fig. 273 is a representation of the masking and enabling registers in MCW1 20290 and of the field in RCWS 10358 registers which controls Event invocations. Beginning with the registers in MCW1 20290, there are three registers which control Event invocations: Event Mask Register (EM) 27301, Events Pending Register (EP) 27309, and Trace Enable Register (TE) 27319. Bits in EM 27301 mask certain Event signals as long as they are set; bits in EP Register 27309 record the occurrence of certain Event signals while they are masked; when bits in TE Register 27319 are set, Trace Event signals occur before certain FU 10120 operations.

EM 27301 contains three one bit fields: Asynchronous Mask Field 27303, Monitor Mask Field 27305, and Trace Event Mask Field 27307. As explained in detail in the discussion of FU 10120 hardware, these bits establish a hierarchy of Event masks. If Asynchronous Mask Field 27303 is set, only two Event signals are masked: that resulting from an overflow of EGGTMR 25412 and that resulting from an overflow of EU 10122's stack. If Monitor Mask Field 27305 is set, those Events are masked, and additionally, the FU Stack Overflow Event signal is masked. As will be explained in detail later, when the FU 10120 Stack Overflow Event signal is masked, the FU micromachine is executing in monitor mode. If Trace Event Mask Field 27307 is set, Trace Trap Event signals are masked in addition to the above signals. Each of the fields in EM 27301 may be individually set and cleared by the microprogram.

Four Event signals set fields in EP 27309: the EGGTMR 25412 Runout signal sets ET Field 27311, the INTTMR 25410 Runout signal sets IT Field 27313, the Non-Fatal Memory Error signal sets ME Field 27315, and the Inter-Process Message signal sets IPM Field 27317. Event invocations for all of these Event signals but the Egg Timer Runout signal occur at the beginning of an SIN; in these cases the fields in EP 27309 retain the fact that the Event signal has occurred until that time; the Event invocation for the Egg Timer Runout signal occurs as soon after the signal as the settings of mask bits in EM 27301 allow. The bit in ET Field 27311 retains the fact of the Egg Timer Runout signal until the masking allows the Event invocation to occur. All of the fields in EP 27309 but ME Field 27315 may be reset by microcode. The microroutines invoked by the Events must reset the appropriate fields; otherwise, they will be reinvoked when they

return. ME Field 27315 is automatically reset when the memory error is serviced.

TE Register Field 27319 enables tracing. Each bit in the register enables a kind of Trace Event signal when it is set. Depending on the kind of tracing, the Trace Event signal occurs at the beginning of an SIN, at the beginning of a Resolve or Evaluate operation, at the beginning of a logical memory reference, or at the beginning of a microinstruction. For details, see the following description of debugging.

Turning now to the registers contained in RCWS 10358, each RCWS Register 27322 contains eight fields which control Event signals. The first field is FM Field 27323. FM Field 27323 reflects the value of a register in Event Logic 20284 when the invocation to which RCWS Register 27322 belongs occurs. The register in Event Logic 20284 is set only when the microinstruction currently being executed is the first microinstruction of an SIN. Thus, FM Field 27323 is set only in RCWS Registers 27322 belonging to Event invocations which occur in the M0 cycle of the first microinstruction in the SIN, i.e., at the beginning of the SIN. The value of the register in Event Logic 20284 is saved in FM Field 27323 because several Event invocations may occur at the beginning of a single SIN. The Event invocations occur in order of priority: when the one with the highest priority returns, the fact that FM Field 27323 is set causes the register in Event Logic 20284 to again be set to the state which it has on the first microinstruction of an SIN. The register's state, thus set, causes the next Event invocation which must occur at the beginning of the SIN to take place. After all such invocations are finished, the first microinstruction enters its M1 cycle and resets the register in Event Logic 20284. In its reset state, the register inhibits all Event invocations which may occur only at the beginning of an SIN. It is again set at the beginning of the next SIN.

The remaining fields in RCWS Register 27322 which control Event invocations are the fields in Return Signals Field 27331. These fields allow the information that an Event signal has occurred to be retained through Event invocations until the Event signal's Event invocation takes place. When an invocation occurs, these fields are set by Event Logic 20284. On return from the invocation, the values of the fields are input into Event Logic 20284, thereby producing Event signals. The Event signal with the highest priority results in an Event invocation, and the remaining Event signals set fields in Return Signals Field 27331 belonging to RCWS Register 27322 belonging to the invocation which is being executed when the Event signals occur. Because the fields in Return Signals Field 27330 are input into Event Logic 20284, microcode invoked as a consequence of Event signals which sets one of these fields must reset the field itself. Otherwise, the return from the microcode will simply result in a reinvocation of the microcode.

The seven fields in Return Signals Field 27330 have the following significance:

- When EG Field 27333 is set, an EU 10122 dispatch operation produced an illegal location in EU 10122 microcode EUSITT 20344.
- When NT Field 27335, ST Field 27341, mT Field 27343, or mB Field 27345 is set, a trace signal has occurred. These are explained in detail in the discussion of debugging.
- When ES Field 27337 is set, an EU 10122 Storeback Exception has occurred, i.e., an error occurred when EU 10122 attempted to store the result of an operation in MEM 10112.
- When MRR Field 27339 is set, a condition such as an ATU 10228 miss or a Protection Cache 10234 miss has occurred, and it is necessary to reattempt a memory reference.

d. Virtual Micromachines and the Monitor Micromachine

As previously described, microcode being executed on FU 10120's micromachine can run in either monitor mode or virtual mode. In this portion of the discussion, the distinguishing features and applications of the two modes are explained in detail.

1. Virtual Mode

As previously mentioned, the chief distinction between virtual mode and monitor mode is MIS 10368. The fact that MIS 10368 is of essentially unlimited size has the following consequences for microroutines which execute in virtual mode.

- An invocation of a microroutine executing in virtual mode may have as its consequence further invocations to any depth.
- Any invocation of or return from a microroutine executing in virtual mode may cause a page fault.

The FU micromachine is in virtual mode when all bits in the Event Masks portion of MCW1 20290 are cleared. In this state, no enabled Event signals are masked, and Event invocations may occur in any microinstruction which does not itself mask them.

Because invocations may occur to any depth in virtual mode, microroutines executing in this mode may be recursive. Such recursive microroutines are especially useful for the interpretation of Names. Often, as previously described, the Name Table Entry for a Name will contain Names which resolve to other Names, and the virtual micromachine's limitless stack allows the use of recursive Name Resolution microroutines in such situations. Recursive microroutines may also be used for complex SINs, such as Calls.

Because invocations can occur to any depth, any number of Events may occur while a microroutine is executing in monitor mode. This in turn greatly simplifies Event handling. If an Event signal occurs while an Event with a given priority is being handled and the Event being signalled has a higher priority than the one

being handled, the result is simply the invocation of the new Event's handler. Thus, the order in which the Event handlers finish corresponds exactly to the priorities of their Events: those with the highest finish first.

A page fault may occur on any microinvocation or return executed in virtual mode because an invocation in virtual mode which occurs when there are no more Free Frames 27207 on SRs 10362 causes an Event signal which invokes a microroutine running in monitor mode. The microroutine transfers MIS Frames 27203 from GRF 10354 to Secure Stack 10336 in MEM 10112, and the transfer may cause a page fault. Similarly, when a microreturn takes place from the last frame on MIS Frames 27203 on SRs 10362, an Event signal occurs which invokes a microroutine that transfers additional frames from Secure Stack 10336 to GRF 10354, and this transfer, too, may cause a page fault.

The fact that page faults may occur on microinvocations or microreturns in virtual mode has two important consequences: microroutines which cannot tolerate page faults other than those explicitly generated by the microroutine itself cannot execute in virtual mode, and because unexpected page faults cause execution to become indeterminate, microroutines which must run to completion cannot execute in virtual mode. For example, if the microroutine which handles page faults executed in virtual mode, its invocation could cause a page fault, which would cause the microroutine to be invoked again, which would cause another page fault, and so on through an infinite series of recursions.

2. Monitor Micromachine

As previously described, the essential feature of monitor mode is MOS 10370. In a present embodiment of CS 10110, this stack has a fixed minimum size, and is always contained in GRF Registers 10354. The nature of MOS 10370 has four consequences for microroutines which execute in monitor mode:

- When the micromachine is in monitor mode, the depth of invocations is limited; recursive microroutines therefore cannot be executed in monitor mode, and Event invocations must be limited.
- Invocations of microroutines or returns from microroutines in monitor mode never result in page faults.
- Microroutines executing in monitor mode are guaranteed to run to completion if they do not suspend the Process 610 which they are executing or perform a Call to software.
- When the micromachine is executing in monitor mode, it is guaranteed to return to virtual mode within a reasonable period of time, either because a microroutine executing in monitor mode has run to completion, or because the microroutine has suspended the Process 610 which it is executing, or has made a Call to software. The result in both cases is the execution of a new sequence of SOPs, and thus a return to virtual mode.

In a present embodiment of CS 10110, the FU micromachine is in monitor mode when a combination of masking bits in MCW1 20290 is set which results in the masking of the FU Stack Overflow Event and the Egg Timer Overflow Event. As previously described, these Events are masked if Fields 27303, 27305, or 27307 is set. These Events and the consequences of masking them are explained in detail below.

The event signal for the FU Stack Overflow Event occurs on microinvocations for which there is no frame available in MIS Frames 27203. If the Event signal is not masked, it causes the invocation of a microroutine which moves MIS Frames from MIS Frames 27203 onto a Process 610's Secure Stack 10336. When the FU Stack Overflow Event is masked, all frames in SRs 10362 of GRs 10360 are available for microroutine invocations and microroutine invocations will not result in page faults, but if the capacity of SRs 10362 is exceeded, FU 10120 ceases operation.

The Egg Timer Overflow event signal occurs when Egg TMR 25412 runs out. As will be explained in detail later, Egg TMR 25412 ensures that an Interval Timer Runout, an Inter-processor Message, or a Non-fatal Memory Error will be serviced by JP 10114 within a reasonable amount of time. If an Interval Timer Runout Event signal or an Inter-processor Message Event signal occurs at a time when it is inefficient for the FU micromachine to handle the Event, Egg TMR 25412 begins running. When Egg TMR 25412 runs out, the Event is handled unless the micromachine is in monitor mode. If the Egg TMR 25412 Runout Event signal occurs while the FU micromachine is in monitor mode, i.e., while the Event is masked, the Event signal sets Field 27311 in MCW1 20290. When the FU micromachine reverts to virtual mode, i.e., when all Event Mask bits in MCW1 20290 are cleared, the Egg TMR 25412 Runout Event occurs, and the Interval Timer Runout Inter-processor Message Event handlers are invoked by Event Logic 20284.

e. Interrupt and Fault Handling

1. General Principles

Any computer system must be able to deal with occurrences which disrupt the normal execution of a program. Such occurrences are generally divided into two classes: faults and interrupts. A fault occurs as a consequence of an attempt to execute a machine instruction, and its occurrence is therefore synchronous with the machine instruction. Typical faults are floating point overflow faults and page faults. A floating point overflow fault occurs when a machine instruction attempts to perform a floating point arithmetic operation and the result exceeds the capacity of the CS 10110's floating point hardware, that is EU 10122. A page fault occurs when a machine instruction in a computer system with virtual memory attempts to reference data which is not presently available in the computer system's primary memory, that is MEM

10112. Since faults are synchronous with the execution of machine instructions and in many cases the result of the execution of specific machine instructions, their occurrence is to some extent predictable.

The occurrence of an interrupt is not predictable. An interrupt occurs as a consequence of some action taken by the computer system which has no direct connection with the execution of a machine instruction by the computer system. For example, an I/O interrupt occurs when data transmitted by an I/O device (IOS 10116) reaches the central processing unit (FU 10120), regardless of the machine instruction the central processing unit is currently executing.

In conventional systems, interrupts and faults have been handled as follows: if an interrupt or fault occurs, the computer system recognizes the occurrence before it executes the next machine instruction and executes an interrupt-handling microroutine or Procedure 602 instead of the next machine instruction. If the interrupt or fault cannot be handled by the Process 610 in which it occurs, the interrupt or fault results in a process swap. When the interrupt handling routine is finished, Process 610 which faulted or was interrupted can be returned to the CPU if it was removed and the next machine instruction executed.

While the above method works well with faults, the fact that interrupts are asynchronous causes several problems:

— Machine instructions cannot require an indefinite amount of time to execute, since interrupts cannot be handled until the machine instruction during which they occur is finished.

— It must be possible to remove a Process 610 from the CPU at any time, since the occurrence of an interrupt is not predictable. This requirement greatly increases the difficulty of process management.

The method used for interrupt and fault handling in a present embodiment of CS 10110 is described below.

2. Hardware Interrupt and Fault Handling in CS 10110

In CS 10110, there are two levels of interrupts: those which may created and dealt with completely by software, and those which may created by hardware signals. The former class of interrupts is dealt with in the discussion of Processes 610; the latter, termed hardware interrupts, is discussed below.

In CS 10110, hardware interrupts and faults begin as invocations of microroutines in FU 10120. The invocations may be the result of Event signals or may be made by microprograms. For example, when IOS 10116 places data in MEM 10112 for JP 10114, an Inter-processor Message Event signal results, and the signal causes the invocation of Inter-processor Message Interrupt handler microcode. On the other hand, a Page Fault begins as an invocation of Page Fault microcode by LAT microcode. The actions taken by the microcode which begins handling the fault or interrupt depend on whether the fault or interrupt is handled by the Process 610 which was being executed when the fault or Event occurred or by a special KOS Process 610.

In the first case, the Event microcode may perform a Microcode-to-Software Call to a high-level language procedure which handles the Event. An example of an Event handled in this fashion is a floating point overflow: when FU 10120 microcode determines that a floating point overflow has occurred, it invokes microcode which may invoke a floating point overflow procedure provided by the high-level language whose S-Language was being executed when the overflow occurred. In alternate embodiments of CS 10110, the overflow procedure may also be in microcode.

In the second case, the microcode handling the fault or interrupt puts information in tables used by a KOS Process 610 which handles the fault or interrupt and then causes the KOS Process 610 to run at some later time by advancing an Event Counter awaited by the Process 610. Event Counters and the operations on them are explained in detail in a following description of Processes 610. Since the tables and Event Counters manipulated by microcode are always present in MEM 10112, these operations do not cause page faults, and can be performed in monitor mode. For example, when IOS 10116 transmits an IPM Event signal to JP 10114 after IOS 10116 has loaded data into MEM 10112, the Event resulting from the Event signal invokes microcode which examines a queue containing messages from IOS 10116. The messages in the queue contain Event Counter locations, and the microcode which examines the queue advances those Event counters, thereby causing Processes 610 which were waiting for the data returned by the I/O operation to recommence execution.

3. The Monitor Mode, Differential Masking and Hardware Interrupt Handling

FU 10120 micromachine's monitor mode and differential masking facilities allow a method of hardware interrupt handling which overcomes two problems associated with conventional hardware interrupt handling: an interrupt can be handled in a predictable amount of time regardless of the amount of time required to execute an SIN, and if the microcode which handles the interrupt executes in monitor mode, the interrupt may be handled at any time without unpredictable consequences. There are two sources of hardware interrupts in CS 10110: an Inter-Processor Message (IPM) and an Interval Timer 25410 Runout. An IPM occurs when IOS 10116 completes an I/O task for JP 10114 and signals completion of the task via IOJP Bus 10132. An Interval Timer Runout occurs when a preset time at which CS 10110 must take some action is reached. For example, a given Process 610 may have a limit placed on the amount of time it may execute on JP 10114. As is explained in a following description of process synchronization, the virtual processor management system sets Interval Timer 25412 to run out when Process 610 has used all of the time available to it.

Both IPMs and Interval Timer Runouts begin as Event signals. The immediate effect of the Event signal is to set a bit in EP Field 27309 of MCW1. In principle, the set bit can cause invocation of the event microcode for the Event on the next M0 cycle in which the FU 10120 micromachine is in virtual mode. Since microroutines running in monitor mode are guaranteed to return the micromachine to virtual mode within a reasonable length of time, and the Event invocation will occur when this happens, the Event is guaranteed to be serviced in a reasonable period of time. The microroutines invoked by the Events themselves execute in monitor mode, thereby guaranteeing that no page faults will occur while they are executing and that Process 610 which is executing on JP 10114 when the hardware interrupt occurs need not be removed from JP 10114.

While hardware interrupts are serviced in principle as described above, considerations of efficiency require that as many hardware interrupts as possible be serviced when the size of the FU micromachine's stack is at a minimum, i.e., at the beginning of an SIN's execution. This requirement is achieved by means of Egg TMR 25412 and ET Flag 27311 in MCW1 20290. As described above, when an IPM interrupt or an Interval Timer 25410 Runout interrupt occurs, Field 27317 or 27313 respectively is set in MCW1 20290. At the same time, Egg TMR 25412 begins running. If the current SIN's execution ends before Egg TMR 25412 runs out, the set Field in MCW1 20290 causes the Interval Timer Runout or Inter-processor Message Event invocations to occur on the first microinstruction for the next SIN. If, on the other hand, the current SIN's execution does not end before Egg TMR 25412 runs out, the Egg Timer Runout causes an Event signal. The immediate result of this signal is the setting of ET bit 27311 in MCW1 20290, and the setting of ET bit 27311 in turn causes the Interval Timer Runout Event invocation and/or IPM Event invocation to take place on the next M0 cycle to occur while the micromachine is in virtual mode. The above mechanism thus guarantees that most hardware interrupts will be handled at the beginning of an SIN, but that hardware interrupts will always be handled within a certain amount of time regardless of the length of time required to execute an SIN.

g. FU Micromachine and CS 10110 Subsystems

The subsystems of CS 10110, such as the object subsystem, the process subsystem, the S-Interpreter subsystem, and the Name Interpreter subsystem, are implemented all or in part in the micromachine. The description of the micromachine therefore closes with an overview of the relationship between these subsystems and the micromachine. Detailed descriptions of the operation of the subsystems have been presented previously.

The subsystems fall into three main groups: KOS subsystems, the Name Interpreter subsystem, and the S-Interpreter subsystem. The relationship between the three is to some extent hierarchical: the KOS subsystems provide the environment required by the Name Interpreter subsystem, and the Name Interpreter subsystem provides the environment required by the S-Interpreter subsystem. For example, the S-Interpreter subsystem interprets SINs consisting of SOPs and Names; the Name Interpreter subsystem translates Names into logical descriptors, using values called ABPs to calculate the locations contained in the logical descriptors. The KOS subsystems calculate the values of the ABPs, translate Logical Descriptors 27116 into physical MEM 10112 addresses, and check whether a Process 610 has access to an object which it is referencing.

In a present embodiment of CS 10110, the Name Interpreter subsystem and the S-Interpreter subsystem are implemented completely in the micromachine; in other embodiments, they could be implemented in high-level languages or in hardware. The KOS subsystems are implemented in both the micromachine and in high-level language routines. In alternate embodiments of CS 10110, KOS subsystems may be embodied entirely in microcode, or in high-level language routines. Some high-level language routines may execute in any Process 610, while others are executed only by special KOS Processes 610. The KOS subsystems also differ from the others in the manner in which the user has access: with the S-Interpreter subsystem and the Name Interpreter subsystem, the subsystems come into play only when SINs are executed; the subsystems are not directly visible to users of the system. Portions of the KOS subsystems, on the other hand, may be explicitly invoked in high-level language programs. For example, an invocation in a high-level language program may cause KOS to bind a Process 610 to a Virtual Processor 612.

The following will first list the functions performed by the subsystems, and then relate the subsystems to the monitor and virtual micromachine modes and specific micromachine devices. KOS subsystems perform the following functions:

- Virtual memory management;
- Virtual processor management;
- Inter-processor communication;
- Access Control;
- Object management; and,
- Process management.

The Name Interpreter performs the following functions:

- Fetching and parsing SOPs, and
- Interpreting Names.

The S-Interpreter, finally, dispatches SOPs, i.e., locates the FU 10120 and EU 10122 microcode which

executes the operation corresponding to a given SOP for a given S-Language.

Of these subsystems, the S-Interpreter, the Name Interpreter, and the microcode components of the KOS process and object manager subsystems execute on the virtual micromachine; the microcode components of the remaining KOS subsystems execute on the monitor micromachine. As will be seen in the discussions of these subsystems, subsystems which execute on the virtual micromachine may cause Page Faults, and may therefore reference data located anywhere in memory; subsystems which execute on the monitor micromachine may not cause Page Faults, and the data bases which these subsystems manipulate must therefore always be present at known locations in MEM 10112.

The relationship between subsystems and FU 10120 micromachine devices is the following: Microcode for all subsystems uses DESP 20210, Microcode Addressing 27013, and Register Addressing 27011, and may use EU Interface 27007. S-Interpreter microcode uses SOP Decoder 27003, and Name Interpreter Microcode uses Instruction Stream Reader 27001, Parsing Unit 27005, and Name Translation Unit 27015. KOS virtual memory management microcode uses Memory Reference Unit 27017, and Protection Microcode uses Protection Unit 27019.

Having described in detail the structure and operation of CS 10110's major subsystems, MEM 10112, FU 10120, EU 10122, IOS 10116, and DP 10118, and the CS 10110 micromachine, CS 10110 operation will be described in further detail next below. First, operation of CS 10110's Namespace, S-Interpreter, and Pointer Systems will be described. Then, operation of CS 10110 will be described in further detail with respect to CS 10110's Kernel Operating System.

3. Namespace, S-Interpreters, and Pointers (Figs. 301—307, 274)

The preceding chapters have presented an overview of CS 10110, examined its hardware in detail, and explained how the FU 10120 hardware functions as a micromachine which controls the activities of other CS 10110 components. In the remaining portions of the specification, the means are presented by which certain key features of CS 10110 are implemented using the hardware, the micromachine, tables in memory, and high-level language programs. The present chapter presents three of these features: the Pointer Resolution System, Namespace, and the S-interpreters.

The Pointer Resolution System translates pointers, i.e., data items which contain location information, into UID-offset addresses. Namespace has three main functions:

— It locates SINs and fetches them from CS 10110's memory into FU 10120.
— It parses SINs into SOPs and Names.
— It translates Names into Logical Descriptors 27116 or values.

The S-interpreters decode S-operations received from namespace into locations in microcode contained in FUSITT 11012 and EUSITT 20344 and then execute that microcode. If the S-operations require operands, the S-interpreters use Namespace to translate the operands into Logical Descriptors 27116 or values as required by the operations.

Since Namespace depends on the Pointer Resolution System and the S-interpreters depend on Namespace, the discussion of the systems begins with pointers and then deals with namespace and S-interpreters.

A. Pointers and Pointer Resolution (Figs. 301, 302)

A pointer is a data item which represents an address, i.e., in CS 10110, a UID-offset address. CS 10110 has two large classes of pointers: resolved pointers and unresolved pointers. Resolved pointers are pointers whose values may be immediately interpreted as UID-offset addresses; unresolved pointers are pointers whose values must be interpreted by high level language routines or microcode routines to yield UID-offset addresses. The act of interpreting an unresolved pointer is called resolving it. Since the manner in which an unresolved pointer is resolved may be determined by a high-level language routine written by a system user, unresolved pointers provide a means by which users of the system may define their own pointer types.

Both resolved and unresolved pointers have subclasses. The subclasses of resolved pointers are UID pointers and object relative pointers. UID pointers contain a UID and offset, and can thus represent any CS 10110 address; object-relative pointers contain only an offset; the address's UID is assumed to be the same as that of the object containing the object-relative pointer. An object-relative pointer can therefore only represent addresses in the object which contains the pointer.

The subclasses of unresolved pointers are ordinary unresolved pointers and associative pointers. The difference between the two kinds of unresolved pointers is the manner in which they are resolved. Ordinary unresolved pointers are always resolved by high-level language routines, while associative pointers are resolved the first time they are used in a Process 610 and a domain by high-level language routines, but are subsequently resolved by means of a table called the Associated Address Table (AAT). This table is accessible to microcode, and associative pointers may therefore be more quickly resolved than ordinary unresolved pointers.

The following discussion will first explain the formats used by all CS 10110 pointers, and will then explain how pointers are processed in FU 10120.

a. Pointer Formats (Fig. 301)

Figure 301 represents a CS 10110 pointer. The figure has two parts: a representation of General Pointer Format 30101, which gives an overview of the fields which appear in all CS 10110 pointers, and a detailed presentation of Flags and Format Field 30105, which contains the information by which the kinds of CS 10110 pointers are distinguished.

Turning first to General Pointer Format 30101, all CS 10110 pointers contain 128 bits and are divided into three main fields:

- Offset Field 30103 contains the offset portion of a UID-offset address in resolved pointers and in associative pointers; in other unresolved pointers, it may contain an offset from some point in an object or other information as defined by the user.
- Flags and Format Field 30105 contains flags and format codes which distinguish between kinds of pointers. These flags and format codes are explained in detail below.
- UID field 30115 contains a UID in UID pointers and in some associative pointers; in objectrelative pointers, and other associative pointers, its meaning is undefined, and in ordinary unresolved pointers, it may contain information as defined by the user.

Flags and Format Field 30105 contains four subfields:

- Fields 30107 and 30111 are reserved and must be set to 0.
- NR Field 30109 indicates whether a pointer is resolved or unresolved. In resolved pointers, the field is set to 0, and in unresolved pointers, it is set to 1.
- Format Code Field 30113 indicates the kind of resolved or unresolved pointers. Format codes for the present embodiment are explained below.

The values of Format Code Field 30113 may range from 0 to 31. If Format Code Field 30113 has the value 0, the pointer is a null pointer, i.e., a pointer which neither directly nor indirectly indicates an address. The meanings of the other format codes depend on the value of NR Field 30109:

NR Field Value	Format Code Value	Meaning
0	1	UID pointer
0	2	Object-relative pointer
0	all other codes	Illegal
1	1	UID associative pointer
1	2	Object-relative associative pointer
1	all other codes	Ordinary unresolved pointer

As indicated by the above table, the present embodiment has two kinds of associative pointer, UID associative pointers and object-relative associative pointers. Like a UID pointer, a UID associative pointer contains a UID and an offset, and like an object-relative pointer, an object-relative associative pointer contains an offset and takes the value of the UID from the object to which it belongs. However, as will be explained in detail later, the UID and offset which the associative pointers contain or represent are not used as addresses. Instead, the UID and offset are used as tags to locate entries in the AAT, which associates an associative pointer with a resolved pointer.

b. Pointers in FU 10120 (Fig. 302)

When a pointer is used as an address in FU 10120, the address information in the pointer must be translated into a Logical Descriptor 27116 consisting of an AON, an offset, and a length field of 0; when a Logical Descriptor 27116 in FU 10120 is used to form a pointer value in memory, the AON must be converted back to a UID. The first conversion is termed pointer-to-descriptor conversion, and the second descriptor-to-pointer conversion. Both conversions are accomplished by microcodes executing in FU 10120.

What is involved in the translation depends on the kind of pointer: if the pointer is a UID pointer, the UID must be translated into an AON; if the pointer is an object-relative pointer, the AON required to fetch the pointer is the pointer's AON, so no translation is necessary. If the pointer is an unresolved pointer, it must first be translated into a resolved pointer and then into a Logical Descriptor 27116. If the pointer is associative, the translation to a resolved pointer may be performed by means of the ATT.

In the present embodiment, when other FU 10120 microcode calls pointer-to-descriptor microcode, the calling microcode passes Logical Descriptor 27116 for the location of the pointer which is to be translated as an argument to the pointer-to-description translation microcode. The pointer-to-descriptor microcode returns a Logical Descriptor 27116 produced from the value of the pointer at the location specified by

Logical Descriptor 27116 which the pointer-to-descriptor microcode received as an argument.

The pointer-to-descriptor microcode first uses Logical Descriptor 27116 given it as an argument to fetch the value of the pointer's Offset Field 30103 from memory. It then saves Logical Descriptor 27116's offset in the output register belonging to OFFALU 20242 and places the value of the pointer's Offset Field 30103 in the offset field of Logical Descriptor 27116 which it received as an argument. The pointer-to-descriptor microcode then saves Logical Descriptor 27116 indicating the pointer's location by storing Logical Descriptor 27116's AON and offset (obtained from OFFALU 20242) in a register in the GRF 10354 frame being used by the invocation of the pointer-to-descriptor microcode. Next, the microcode adds 40 to the offset stored in OFFALU 20242, thereby obtaining the address of NR Field 30109, and uses the address to fetch and read NR Field 30109 and Format Code Field 30113. The course of further processing is determined by the values of these fields. If NR Field 30109 indicates a resolved pointer, there are four cases, as determined by the value of Format Code Field 30113:

- Format code field = 0: The pointer is a null pointer.
- Format code field = 1: The pointer is a UID pointer.
- Format code field = 2: The pointer is an intra-object pointer.
- Any other value of the format code field: The pointer is invalid.

In the first case, the microcode sets all fields of the argument to 0; in the second, it fetches the value of UID Field 30115 from memory and invokes LAR microcode (explained in the discussion of objects), which translates the UID to the AON associated with it. The AON is then loaded into the argument's AON field. In the third case, the AON of Logical Descriptor 27116 for the pointer's location and the pointer's AON are the same, so the argument already contains the translated pointer. In the fourth case, the microcode performs a call to a pointer fault-handling Procedure 602 which handles invalid pointer faults, passing saved Logical Descriptor 27116 for the pointer as an argument. Procedure 602 which handles the fault must return a resolved pointer to the microcode, which then converts it to a Logical Descriptor 27116 as described above.

c. Descriptor to Pointer Conversion

Descriptor to pointer conversion is the reverse of pointer to descriptor conversion with resolved pointers. The operation must be performed whenever a resolved pointer is moved from an FU 10120 register into MEM 10112. The operation takes two arguments: a Logical Descriptor 27116 which specifies the address to which the pointer is to be written, and a Logical Descriptor 27116 whose AON and offset fields specify the location contained in the pointer. There are two cases: intra-object pointers and UID pointers. Both kinds of pointers have values in Offset Field 30103, so the descriptor-to-pointer microcode first writes the second argument's offset to location specified by the first argument's Logical Descriptor 27116. The next step is to determine whether the pointer is an intra-object pointer or a UID pointer. To do so, the microcode compares the arguments' AONs. If they are the same, the pointer points to a location in the object which contains it, and is therefore an intra-object pointer. Since UID Field 30115 of an intra-object pointer is meaningless, the only step remaining for intra-object pointers is to set Flags and Format Field 30105 to the binary representation of 2, which sets all bits but bit 46 to 0, and thereby identifies the pointer as a resolved intra-object pointer.

With UID pointers, the descriptor-to-pointer microcode sets Flags and Format Field 30105 to 1, thereby identifying the pointer as a resolved UID pointer, and calls a KOS LAR microroutine (explained in detail in the discussion of objects) which converts the first argument's AON to a UID and places the result UID in the current frame. When the KOS AON to UID conversion microroutine returns, the descriptor-to-pointer microcode writes the UID to the converted pointer's UID Field 30115.

B. Namespace and the S-Interpreters (Figs. 303—307)

Namespace and the S-Interpreter both interpret information contained in Procedure Objects 608. Consequently, the discussion of these components of CS 10110 begins with an overview of those parts of Procedure Object 606 relevant to Namespace and the S-interpreters, and then explains Namespace and the S-interpreters in detail.

a. Procedure Object 606 Overview (Fig. 303)

Figure 303 represents those portions of Procedure Object 608. Fig. 303 expands information contained in Fig. 103; Fields which appear in both Figures have the number of Fig. 103. Portions of Procedure Object 608 which are not discussed here are dealt with later in the discussion of Calls and Returns. The most important part of a Procedure Object 608 for these systems is Procedure Environment Descriptor (PED) 30303. A Procedure 602's PED 30303 contains the information required by Namespace and the S-interpreter to locate and parse Procedure 602's code and interpret its Names. A number of Procedures 602 in a Procedure Object 608 may share a PED 30303. As will be seen in the discussion of Calls, the fact that a Procedure 602 shares a PED 30303 with the Procedure 602 that invokes it affects the manner in which the Call is executed.

The fields of PED 30303 which are important to the present discussion are three fields in Header 30304: K Field 30305, LN Field 30307, and SIP Field 30309, and three of the remaining fields: NTP Field 30311, SDPP Field 30313, and PBP Field 30315.

— K Field 30305 indicates whether the Names in the SINs of Procedures 602 which share PED 30303 have 8, 12, or 16 bits.
— LN Field 30307 contains the Name which has the largest index of any in Procedure 602's Name Table 10350.
— SIP Field 30309 is a UID pointer to the object which contains the S-interpreter for Procedure 602's S-Language.
— NTP Field 30311 is an object-relative pointer to the beginning of Procedure 602's Name Table 10350.
— SDPP Field 30313 is a pointer which is resolved to the location of static data used by Procedures 602 to which PED 30303 belongs when one of Procedures 602 is invoked by a given Process 610. The resolved pointer corresponding to SDPP 30313 is the SDP ABP.
— PBP Field 30315 contains the PBP ABP for invocations of Procedures 602 to which PED 30303 belongs. The PBP ABP is used to calculate locations inside Procedure Object 608.

Other areas of interest in Procedure Object 608 are Literals 30301 and Static Data Prototype (SDPR) 30317. Literals 30301 contains literal values, i.e., values in Procedure 602 which are known at compile time and will not change during program execution. SDPR 30317 may contain any of the following: pointers to external routines and to static data contained in other objects, information required to create static data for a Procedure 602, and in some cases, the static data itself. Pointers in SDPR 30317 may be either resolved or non-resolved.

In the present embodiment, Binder Area 30323 is also important. Binder Area 30323 contains information which allows unresolved pointers contained in Procedure Object 608 to be resolved. Unresolved pointers other than SDPP 30313 in Procedure Object 608 all contain locations in Binder Area 30323, and the specified location contains the information required to resolve the pointer.

Fig. 303 contains arrows showing the locations in Procedure Object 608 pointed to by NTP Field 30311, SDPP Field 30313, and PBP Field 30315. NTP Field 30311 points to the beginning of Name Tables 10350, and thus a Name's Name Table Entry can be located by adding the Name's value to NTP Field 30311. PBP Field 30315 points to the beginning of Literals 30301, and consequently, the locations of Literals and the locations of SINs may be expressed as offsets from the value of PBP Field 30315. SDPP Field 30313 points to the beginning of SDPR 30317. As will be explained in detail in the discussion of Calls, when a procedure 602 has static data, the SDP ABP is derived from SDPP Field 30313.

b. Namespace

The Namespace component of CS 10110 locates SINs belonging to a procedure and fetches them from memory to FU 10120, parses SINs into SOPs and Names, and performs Resolve and Evaluation operations on Names. The Resolve operation translates a Name into a Logical Descriptor 27116 for the data represented by the Name, while the Evaluation operation obtains the data itself. The Evaluation operation does so by performing a Resolve operation and then using the resulting Logical Descriptor 27116 to fetch the data. Since the Evaluation and Resolve operations are the most complicated, the discussion begins with them.

1. Name Resolution and Evaluation

Name Resolution and Evaluation translate Names into Logical Descriptors 27116 by means of information contained in the Names' NTEs, and the NTEs define locations in terms of Architectural Base Registers. Consequently, the following discussion will first describe Name Table Entries and Architectural Base Pointers and then the means by which Namespace translates the information contained in the Name Table Entries and Architectural Base Pointers into Logical Descriptors 27116.

2. The Name Table (Fig. 304)

As previously mentioned, Name Tables 10350 are contained in Procedure Objects 608. Name Tables 10350 contain the information required to translate Names into Logical Descriptors 27116 for the operands represented by the Names. Each Name has as its value the number of a Name Table Entry. A Name's Name Table Entry is located by multiplying the Name's value by the size of a short Name Table Entry and adding the product to the value in NTP Field 30311 of PED 30303 belonging to Procedure 602 which contains the SIN.

The Name Table Entry contains length and type information for the data item specified by the Name, and represents the data item's location as a displacement from a known location, termed the base. The base may be a location specified by an ABP, a location specified by another Name, or a location specified by a pointer. In the latter case, the pointer's location may be specified in terms of an ABP or as a Name.

Fig. 304 is a detailed representation of a Name Table Entry (NTE) 30401. There are two kinds of NTEs 30401: Short NTEs 30403 and Long NTEs 30405. Short NTEs 30403 contain 64 bits; Long NTEs 30405 contain 128 bits. Names that represent scaler data items whose displacements may be expressed in 16 bits have Short NTEs 30403; Names that represent scaler data items whose displacements require more than 16 bits and Names that represent array elements have Long NTEs 30405.

A Short NTE 30403 has four main fields, each 16 bits in length:

— Flags and Format Field 30407 contains flags and format information which specify how Namespace is to interpret NTE 30401.

— Base Field 30425 indicates the base to which the displacement is to be added to obtain the location of the data represented by the Name. Base Field 30425 may represent the location in four ways: by means of an ABP by means of a Name, by means of a pointer located by means of an ABP, and by means of a pointer located by means of a Name.
— Length Field 30435 represents the length of the data. The length may be a literal value or a Name. If it is a Name, the Name resolves to a location which contains the data item's length.
— Displacement Field 30437 contains the displacement of the beginning of the data from the base specified in Field 30425. The displacement is a signed integer value.

Long NTEs 30405 have four additional fields, each 16 bits long: Two of the fields, Index Name Field 30441 and IES Field 30445 are used only in NTEs 30401 for Names that represent arrays.

— Displacement Extension Field 30439 is used in all Long NTEs 30405. If the displacement value in Field 30437 has less than 16 bits, Displacement Extension Field 30439 contains sign bits, i.e., the bits in the field are set to 0 when the displacement is positive and 1 when the displacement is negative. When the displacement value has more than 16 bits, Displacement Extension Field 30439 contains the most significant bits of the displacement value as well as sign bits.
— Index Name Field 30441 contains a Name that represents a value used to index an element of an array.
— Field 30443 is reserved.

IES Field 30445 contains a Name or Literal that specifies the size of an element in an array. The value represented by this field is used together with the value represented by Index Name Field 30441 to locate an element of an array.

As may be seen from the above, the following fields may contain names: Base Field 30425, Length Field 30435, Index Name Field 30441, and IES Field 30445.

Two fields in NTE 30401 require further consideration: Flags and Format Field 30407 and Base Field 30425. Flags and Format Field 30407 has three subfields: Flags Field 30408, FM Field 30421, and Type Field 30423. Turning first to Flags Field 30408, the six flags in the field indicate how Namespace is to interpret NTE 30401. The flags have the following meanings when they are set:

— Long NTE Flag 30409: NTE 30401 is a Long NTE 30405.
— Length is a Name Flag 30411: Length Field 30435 contains a Name.
— Base is a Name Flag 30413: Base Field 30425 contains a Name instead of the number of an ABP.
— Base Indirect Flag 30415: Base Field 30425 represents a pointer, and the location represented by NTE 30401 is to be calculated by obtaining the pointer's value and adding the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the pointer's offset.
— Array Flag 30417: NTE 30401 represents an array.
— IES is a Name Flag 30419: IES Field 30445 contains a Name that represents the IES value.

Several of these flags may be set in a given NTE 30401. For example, an entry for an array element that was referenced via a pointer to the array which in turn was represented by a Name, and whose IES value was represented by a Name, would have Flags 30409, 30413, 30415, 30417, and 30419 set.

FM Field 30421 indicates how the data represented by the Name is to be formated when it is fetched from memory. The value of FM Field 30421 is placed in FIU Field 27107 of Logical Descriptor 27116 produced from NTE 30401. The two bits allow for four possibilities:

Setting	Meaning
00	right justify, zero fill
01	right justify, sign fill
10	left justify, zero fill
11	left justify, ASCII space fill

The four bits in Type Field 30423 are used by compilers for language-specific type information. The value of Type Field 30423 is placed in Type Field 27109 of Logical Descriptor 27116 produced from NTE 30401.

Base Field 30425 may have either Base is an ABP Format 30427 or Base is a Name Format 30432. The manner in which Base Field 30425 is interpreted depends on the setting of Base is a Name Flag 30413 and Base Indirect Flag 30415. There are four possibilities:

Field Settings

Base is a Name	Base Indirect	Meaning
0	0	ABP Format locates base directly.
0	1	ABP Format locates a pointer which is the base.
1	0	Base is Name Format locates base when Name is resolved.
1	1	Base is Name Format locates a pointer when Name is resolve and the pointer is the base.

As indicated by the above table, Base Field 30425 is interpreted as having Base is ABP Format 30427 when Base is a Name Flag 30411 is not set. In Base is ABP Format 30427, Base Field 30425 has two subfields: ABP Field 30429 and Pointer Locator Field 30431. The latter field has meaning only when Base Indirect Flag 30415 is set. ABP Field 30429 is a two-bit code which indicates the ABP. The settings and their meanings are the following:

Setting	APB
00	FP
01	Unused
10	SDP
11	PBP

The ABPs are discussed below. When Base Indirect Flag 30415 is set to 1 and Base is a Name Flag 30413 is set to 0, the remaining 14 bits of the Base Field in ABP Format are interpreted as Pointer Locator Field 30413. When so interpreted, Pointer Locator Field 30413 contains a signed integer, which, when multiplied by 128, gives the displacement of a pointer from the ABP specified in ABP Field 30429. The value of this pointer is then the base to which the displacement is added.

Base Field 30425 is interpreted as having Base is a Name Format 30432 when Base is a Name Flag 30413 is set to 1. In Base is a Name Format 30432, Base Field 30425 contains a Name. If Base Indirect Flag 30415 is not set, the Name is resolved to obtain the Base. If Base Indirect Flag 30415 is set, the name is evaluated to obtain a pointer value, and that pointer value is the Base.

3. Architectural Base Pointers (Figs. 305, 306)

If Base is a Name Flag 30413 belonging to a NTE 30401 is not set, Base Field 30425 specifies one of the three ABPs in CS 10110:

- PBP specifies a location in Procedure Object 608 to which displacements may be added to obtain the locations of Literals and SINs.
- SDP specifies a location in a Static Data Block for an invocation of a Procedure 602 to which displacements may be added to obtain the locations of static data and linkage pointers to Procedures 602 contained in other Procedure Objects 608 and static data.
- FP specifies a location in the MAS frame belonging to Procedure 602's current invocation to which displacements may be added to obtain the location of local data and linkage pointers to arguments.

Each time a Process 610 invokes a Procedure 602, Call microcode saves the current values of the ABPs on Secure Stack 10336, calculates the values of the ABPs for the new invocation, and places the resulting Logical Descriptors 27116 in FU 10120 registers, where they are accessible to Namespace microcode.

Call microcode calculates the ABPs as follows: PBP is obtained directly from PBP Field 30315 in PED 30303 belonging to the Procedure 602 being executed. All that is required to make it into a Logical Descriptor 27116 is the addition of the AON for Procedure Object 608's UID.

SDP is obtained by performing a pointer-to-descriptor translation on SDPP Field 30313. FP, finally, is provided by the portion of Call microcode which creates the new MAS 502 frame for the invocation. As is described in detail in the discussion of Call, the Call microcode copies linkage pointers to the invocation's actual arguments onto MAS 502, sets FP to point to the location following the last actual argument, and then allocates storage for the invocation's local data. Positive displacements from FP thus specify locations

in the local data, while negative offsets specify linkage pointers.

a.a. Resolving and Evaluating Names (Fig. 305)

The primary operations performed by Namespace are resolving names and evaluating them. A Name has been resolved when Namespace has used the ABPs and information contained in the Name's NTE 30401 to produce a Logical Descriptor 27116 for the Name; a name has been evaluated when Namespace has resolved the Name, presented the resulting Logical Descriptor 27116 for the Name to memory, and obtained the value of the data represented by the Name from memory.

The resolve operation has three parts, which may be performed in any order:

— Obtaining the Base from Base Field 30425 of the Name's NTE 30401.
— Obtaining the displacement.
— Obtaining the length from Length Field 30435.

Obtaining the length is the simplest of the operations: if Length in a Name Flag 30411 is set, the length is the value obtained by evaluating the Name contained in Length Field 30435; otherwise, Length Field 30435 contains a literal value and the length is that literal's value.

There are four ways in which the Base may be calculated. Which is used depends on the settings of Base is a Name Flag 30413 and Base Indirect Flag 30415:

— Both Flags 0: the ABP specified in ABP Field 30429 is the Base.
— Base is a Name Flag 30413 0 and Base Indirect Flag 30415 1: The Base is the location contained in the pointer specified by ABP Field 30429 and pointer Locator Field 30431.
— Base is a Name Flag 30413 1 and Base Indirect Flag 30415 0: The Base is the location obtained by resolving the Name in Base Field 30425.
— Both Flags 1: The Base is the location obtained by evaluating the Name in Base Field 30425.

The manner in which Namespace calculates the displacement depends on whether NTE 30401 represents a scaler data item or an array data item. In the first case, Namespace adds the value contained in Displacement Field 30437 and Displacement Extension Field 30439 to the location obtained for the Base; in the second case, Namespace evaluates Index Name Field 30441 and IES Field 30445, multiplies the resulting values together, and adds the product to the value in Displacement Field 30437 in order to obtain the displacement.

If any field of a NTE 30401 contains a Name, Namespace obtains the value or location represented by the Name by performing a Resolve or Evaluation operation on it as required. As mentioned in the discussion of NTEs 30401, flags in Flags Field 30408 indicate which fields of an NTE 30401 contain Names. Since the NTE 30401 for a Name used in another NTE 30401 may itself contain Names, Namespace performs the Resolve and Evaluation operations recursively.

b.b. Implementation of Name Evaluation and Name Resolve in CS 10110

In the present embodiment, the Name Evaluation and Resolve operations are carried out by FU 10120 microcode Eval and Resolve commands. Both commands require two pieces of information: a register in the current frame of SR portion 10362 of GRF 10354 for receiving Logical Descriptor 27116 produced by the operation, and the source of the Name which is to be resolved or evaluated. Both Resolve and Eval may choose between three sources: Parser 20264, Name Trap 20254, and the low-order 16 bits of the output register for OFFALU 20242. Resolve may specify current frame registers 0, 1, or 2 for Logical Descriptor 27116, and Eval may specify current frame registers 0 or 1. At the end of the Resolve operation, Logical Descriptor 27116 for the data represented by the Name is in the specified SR 10362 register and at the end of the Evaluation operation, Logical Descriptor 27116 is in the specified SR 10362 register and the data's value has been transferred via MOD Bus 10114 to EU 10122's OPB 20322.

The execution of both Resolve and Eval commands always begin with the presentation of the Name to Name Cache 10226. The Name presented to Name Cache 10226 is latched into Name Trap 20254, where it is available for subsequent use by Name Resolve microcode.

If there is an entry for the Name in Name Cache 10226, a name cache hit occurs. For Names with NTEs 30401 fulfilling three conditions, the Name Cache 10226 entry for the Name is a Logical Descriptor 27116 for the data item represented by the Name. The conditions are the following:

— NTE 30401 contains no Names.
— Length Field of NTE 30401 specifies a length of less than 256 bits.
— If Base is Indirect Flag 30415 is set, Pointer Displacement Field 30431 must have a negative value, indicating that the base is a linkage pointer.

Logical Descriptor 27116 can be encached in this case because neither the location nor the length of the data represented by the Name can change during the life of an invocation of Procedure 602 to which the Name belongs. If the Name Cache 10226 entry for the Name is a Logical Descriptor 27116, the hit causes Name Cache 10226 to place Logical Descriptor 27116 in the specified SR 10362 register. In all other cases, the Name Cache 10226 entry for the Name does not contain a Logical Descriptor 27116, and a hit causes Name Cache 10226 to emit a JAM signal. The JAM signal invokes microcode which uses information stored in Name Cache 10226 to construct Logical Descriptor 27116 for the data item represented by the Name. JAMS are explained in detail below.

If there is no entry for the Name in Name Cache 10226, a Name Cache Miss occurs, and Name Cache 10226 emits a cache miss JAM signal. The Name Resolve microroutine invoked by the cache miss JAM

signal constructs an entry in Name Cache 10226 from the Name's NTE 30401, using FU 10120's DESP 20210 to perform the necessary calculations. When it is finished, the cache miss microcode leaves a Logical Descriptor 27116 for the Name in the specified SR 10362 register and returns.

The Resolve operation is over when Logical Descriptor 27116 has been placed in the specified GRF 10354 register; the Evaluation operation continues by presenting Logical Descriptor 27116 to Memory Reference Unit 27017, which reads the data represented by Logical Descriptor 27116 from memory and places it on OPB 20322. The memory reference may result in Protection Cache 10234 misses and ATU 10228 misses, as well as protection faults and page faults, but these are handled by means of event signals and are therefore invisible to the Evaluation operation.

Name Cache 10226 produces 15 different JAM signals. The signal produced by a JAM depends on the following: whether the operation is a Resolve or an Eval, which register Logical Descriptor 27116 is to be placed in, whether a miss occurred, and in the case of a hit, which register in the Name Cache 10226 entry for the Name was loaded last. From the point of view of the behavior of the microcode invoked by the JAM, the last two factors are the most important. Their relation to the microcode is explained in detail below.

In the present embodiment, all entries in Name Cache 10226 are invalidated when a Procedure 602 calls another Procedure 602. The invalidation is required because Calls always change the value of FP and may also change the values of SDP and PBP, thereby changing the meaning of NTEs 30401 using displacements from ABPs. Entries for Names in invoked Procedure 602 are created and loaded into Name Cache 10226 when the Names are evaluated or resolved and a cache miss occurs.

The following discussion will first present Name Cache 10226 as it appears to the microprogrammer and then explain in detail how Name Cache 10226 is used to evaluate and resolve Names, how it is loaded, and how it is flushed.

c.c. Name Cache 10226 Entries (Fig. 306)

The structure and the physical behavior of Name Cache 10226 was presented in the discussion of FU 10120 hardware; here, the logical structure of Name Cache 10226 entries as they appear to the microprogrammer is presented. To the microprogrammer, Name Cache 10226 appears as a device which, when presented a Name on NAME Bus 20224, always provides the microprogrammer with a Name Cache 10226 entry for the Name consisting of four registers. The microprogrammer may read from or write to any one of the four registers. When the microprogrammer writes to the four registers, the action taken by Name Cache 10226 when a hit occurs on the Name associated with the four registers depends on which of the registers has most recently been loaded. The means by which Name Cache 10226 associates a Name with the four registers, and the means by which Name Cache 10226 provides registers when it is full are invisible to the microprogrammer.

Fig. 306 illustrates Name Cache Entry 30601 for a Name. The four Registers 30602 in Name Cache Entry 30601 are numbered 0 through 3, and each Register 30602 has an AON, offset, and length field like those in GRF 10354 registers, except that some flag bits in GRF 10354 register AON fields are not included in Register 30602 fields, and the length field in Register 30602 is 8 bits long. As is the case with GRF 10354 registers, the microprogrammer can read or write individual fields of Register 30602 or entire Register 30602. Name Cache Entry 30601 is connected via DB 27021 to DESP 20210, and consequently, the contents of a GRF 10354 register may be obtained from or transferred to a Register 30602 or viceversa. When the contents of a Register 30602 have been transfered to a GRF 10354 register, the contents may be processed using OFFALU 20242 and other arithmetic-logical devices in DESP 20210.

d.d. Name Cache 10226 Hits

When a Name is presented to Name Cache 10226 and Name Cache 10226 has a Name Cache Entry 30601 containing information about the Name, a name cache hit occurs. On a hit, Name Cache 10226 hardware always loads the contents of Register 30602 0 of the Name's Name Cache Entry 30601 into the GRF 10354 register specified in the Resolve or Eval microcommand. In addition, a hit may result in the invocation of microcode via a JAM:

- The JAM may invoke special microcode for resolving Names of array elements whose NTEs 30401 allow certain hardware accelerations of index calculations.
- The JAM may invoke general name resolution microcode which produces a Logical Descriptor 27116 from the contents of Name Cache Entry 30601.

Whether the hit produces a JAM, and the kind of JAM it produces, are determined by the last Register 30602 to be loaded when Name Cache Entry 30601 was created by Name Cache Miss microcode. If Register 30602 0 was the last to be loaded, no JAM occurs; if Register 30602 1 was loaded last, the JAM for special array Name resolution occurs; if Register 30602 2 or 3 was loaded last, the JAM for general Name resolution occurs.

As may be inferred from the above, Name Cache 10226 hardware defines the manner in which Name Cache Entries 30601 are loaded for the first two cases. In the first case, Name Cache Register 30602 O must contain Logical Descriptor 27116 for the Name's data. As already mentioned, the Name's NTE 30401 must therefore describe data whose location and length does not change during an invocation and whose length is less than 256 bits. Name Cache 10226 hardware also determines the form of Name Cache Entries 30601 for encachable arrays. An encachable array NTE 30401 is an array NTE 30401 which fills the following

conditions:

- The only Name contained in array NTE 30401 is in Index Name Field 30441.
- NTE 30401 for the index Name fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.
- The value in IES Field 30445 is no greater than 128 and a power of 2.
- Array NTE 30401 otherwise fills the conditions for scaler NTEs 30401 for which Logical Descriptors 27116 may be encached.

In the present embodiment, the encachable array entry uses registers 0, 1, and 2 of Name Cache Entry 30601 for the name:

Register	Contents		
	AON	OFFSET	LENGTH
0	Logical Descriptor 27116 for the index Name		
1	0	IES power of 2	unused
2	Logical Descriptor 27116 for the array		

When a hit for this type of entry occurs, the resulting JAM signal does two things: it invokes encachable array resolve microcode and it causes the index Name's Logical Descriptor 27116 to be presented to Memory Reference Unit 27017 for a read operation which returns the value of the data represented by the index Name to an accumulator in OFFALU 20242. The encachable array resolve microroutine then uses the Name that caused the JAM, latched into Name Trap 20254, to locate Register 30602 2 of Name Cache Entry 30601 for the Name, writes the contents of Register 30602 2 into the GRF register specified by the Resolve or Eval microcommand, obtains the product of the IES value and the index value by shifting the index value left the number of times specified by the IES exponent in Register 30602 1, adds the result to the offset field of the GRF 10354 register containing the array's Logical Descriptor 27116, thus obtaining Logical Descriptor 27116 for the desired array element, and returns.

For the other cases, the manner in which Name Cache Entries 30601 are loaded and processed to obtain Logical Descriptors 27116 is determined by the microprogrammer. The JAM signal which results if a Name Cache Entry 30601 is neither a Logical Descriptor 27116 nor an encachable array entry merely invokes a microroutine. The microroutine uses the Name latched into Name Trap 20254 to locate the Name's Name Cache Entry 30601 and then reads tag values in Name Cache Entry 30601 to determine how the information in Name Cache Entry 30601 is to be translated into a Logical Descriptor 27116. The contents of Name Cache Entries 30601 for the other cases have two general forms: one for NTEs 30401 with Base is Indirect Flag 30415 set, and one for NTEs in which it is not set. The first general form looks like this:

Register	Contents		
	AON	OFFSET	LENGTH
0	ABP AON	tag/length	unused
1	0	index name/IES	unused
2	0	unused	unused
3	0	data displacement from loc. specified by pointer	unused

Register 30602 0 contains the AON of the ABP. Register 30602 0's offset field contains two items: the tag, which contains Flags Field 30408 of NTE 30401 along with other information, and which determines how Name Resolve microcode interprets the contents of Name Cache Entry 30601, and a value or Name for the length of the data item. Register 30602 1 is used only if the Name represents a data item in an array. It then contains the Name from Index Field 30441 and the Name or value from IES Field 30445. The offset field of Register 30602 3 contains the sum of the offset indicated by NTE 30401's ABP and of the displacement indicated by NTE 30401.

The second format, used for NTEs 30401 whose bases are obtained from pointers or by resolving a Name, looks like this:

Registers	Contents		
	AON	OFFSET	LENGTH
0	0	tag/length	unused
1	0	index name/IES	unused
2	0	FM and type bits/ base field	unused
3	0	data displacement from loc. specified by pointer or name	unused

In this form, the location of the Base must be obtained either by evaluating a pointer or resolving a Name. Hence, there is no field specifying the Base's AON. Otherwise, Registers 30602 0 and 1 have the same contents as in the previous format. In Register 30602 2, the offset field contains Name Table Entry 30401's FM Field 30421 and Type Field 30423 and Base Field 30425. The Offset Field of Register 30602 2 contains the value of Name Table Entry 30401 Displacement Fields 30437 and 30439.

As in Name Table Entries 30401, the index must be represented by a Name, and length, IES, and Base may be represented by Names. If a field of Name Cache Entry 30601 contains a Name, a flag in the tag indicates that fact, and Name Resolve microcode performs an Eval or Resolve operation on it as required to obtain the value or location represented by the name.

The microcode which resolves Name Cache Entries 30601 of the types just described uses the general algorithms described in the discussion of Name Table Entries 30401, and is therefore not discussed further here.

e.e. Name Cache 10226 Misses

When a Name is presented to Name Cache 10226 and there is no Name Cache Entry 30601 for the Name, a name cache miss occurs. On a miss Name Cache 10226 hardware emits a JAM signal which invokes name cache miss microcode. The microcode obtains the Name which caused the miss from Name Trap 20254 and locates the Name's NTE 30401 by adding the Name to the value of NTP 30311 from PED 30303 for Procedure 602 being executed. As will be explained in detail later, when a Procedure 602 is called, the Call microcode places the AON and offset specifying the NTP's location in a register in GR's 10360. Using the information contained in the Name's NTE 30401, the Cache Miss microcode resolves the Name and constructs a Name Cache Entry 30601 for it. As described above, the microcode determines the method by which it resolves the Name and the form of the Name's Name Cache Entry 30601 by reading Flags Field 30408 in the Name's NTE 30401. Since the descriptions of the Resolve operation, the micromachine, Name Cache 10226, and the formats of Name Cache Entries 30601 are sufficient to allow those skilled in the art to understand the operations performed by Cache Miss microcode, no further description of the microcode is provided.

f.f. Flushing Name Cache 10226

As described in the discussion of Name Cache 10226 hardware, hardware means, namely VALS 24068, exist which allow Name Cache Entries 30601 to be invalidated. Name Cache Entries 30601 may be invalidated singly, or all entries in Name Cache 10226 may be invalidated by means of a single microcommand. The latter operation is termed name cache flushing. In the present embodiment, Name Cache 10226 must be flushed when Process 610 whose Virtual Processor 612 is bound to JP 10114 executes a Call or a Return and whenever Virtual Processor 612 NO is unbound from JP 10114. Flushing is required on Call and Return because Calls and Returns change the values of the ABPs and other pointers needed to resolve Names. At a minimum, a Call produces a new MAS Frame 10412, and a Return returns to a previous Frame 10412, thereby changing the value of FP. If the called Procedure 602 has a different PED 30303 from that of the calling Procedure 602, the Call or Return may also change PBP, SDP, and NTP. Flushing is required when a Virtual Processor 612 is unbound from JP 10114 because Virtual Processor 612 which is next bound to JP 10114 is bound to a different Process 610, and therefore cannot use any information belonging to Process 610 bound to the Previous Virtual Processor 612.

g.g. Fetching the I-Stream

As explained in the discussion of FU 10120 hardware, SINs are fetched from memory by Prefetcher 20260. PREF 20260 contains a Logical Descriptor 27116 for a location in Code 10344 belonging to Procedure 602 which is currently being executed. On any MO cycle, PREF 20260 can place Logical Descriptor 27116 on DB 27021, cause Memory Reference Unit 27017 to fetch 32 bits at the location specified by Logical

Descriptor 27116, and write them into INSTB 20262. When INSTB 20262 is full, PREF 20260 stops fetching SINs until Namespace parsing operations, described below, have processed part of the contents of INSTB 20262, thereby creating space for more SINs.

The fetching operation is automatic, and requires intervention from Namespace only when a SIN causes a branch, i.e., causes the next SIN to be executed to be some other SIN than the one immediately following the current SIN. On a branch, Namespace must load PREF 20260 with the location of the next SIN to be executed and cause PREF 20260 to begin fetching SINs at that location. The operation which does this is specified by the load-prefetch-for-branch microcommand. The microcommand specifies a source for a Logical Descriptor 27116 and transfers that Logical Descriptor 27116 via DB 27021 to PREF 20260. After PREF 20260 has thus been loaded, it begins fetching SINs at the specified location. Since any SINs still in INSTB 20262 have been rendered meaningless by the branch operation, the first SINs loaded into INSTB 20262 are simply written over INSTB 20262's prior contents. Fig. 274 contains an example of the use of the load-prefetch-for-branch microcommand.

h.h. Parsing the I-Stream

The I-stream as fetched from MEM 10112 and stored in INSTB 20262 is a sequence of SOPs and Names. As already mentioned, the I-stream has a fixed format: in the present embodiment, SOPs are always 8 bits long, and Names may be 8, 12, or 16 bits long. The length of Names used in a given procedure is fixed, and is indicated by the value in K Field 30305 in the Procedure 602's PED 30303. The Namespace parsing operations obtain the SOPs and Names from the I-stream and place them on NAME Bus 20224. The SOPs are transferred via this bus to the devices in SOP Decoder 27003, while the Names are transferred to Name Trap 20254 and Name Cache 10226 for Resolve and Evaluation operations as described above. As the parsing operations obtain SOPs and Names, they also update the three program counters CPC 20270, EPC 20274, and IPC 20272. The values in these three counters are offsets from PBP which point to locations in Code 10344 belonging to Procedure 602 being executed. CPC 20270 points to the I-stream syllable currently being parsed, so it is updated on every parsing operation. EPC 20274 points to the beginning of the last SIN executed by JP 10114, and IPC 20272 points to the beginning of the current SIN, so these program counters are changed only at the beginning of the execution of an SIN, i.e., when a SOP is parsed.

As described in the discussion of FU 10120 hardware, in the current implementation, parsing consists physically of reading 8 or 16 bits of data from a location in INSTB 20262 identified by a pointer for INSTB 20262 which is accessible only to the hardware. As data is read, the hardware increments the pointer by the number of bits read, wrapping around and returning to the beginning of INSTB 20262 if it reaches the end. At the same time that the hardware increments the pointer, it increments CPC 20270 by the same number of bits. As previously mentioned, CPC 20270 contains the offset from PBP of the SOP or Name being currently parsed, thus coordinating the reading of INSTB 20262 with the reading of Procedure 602's Code 10344.

The number of bits read depends on whether Parser 20264 is reading an SOP or a Name, and in the latter case, by the syllable size specified for the Name. The syllable size is contained in CSSR 24112. On a Call to a Procedure 602 which has a different PED 30303 from that of the calling procedure, the Call microcode loads the value contained in K Field 30305 into CSSR 24112.

Namespace's parsing operations are performed by separate microcommands for parsing SOPs and Names. There is a single microcommand for parsing S-operations: parse-op-stage. The microcommand obtains the next eight bits from INSTB 20262, places the bits onto NAME Bus 20224, and latches them into LOPCODE Register 24212. It also updates EPC 20274 and IPC 20272 as required at the beginning of an SIN: EPC 20274 is set to IPC 20272's former value, and IPC 20272 is set to CPC 20270's value. At the end of the operation, CPC 20270 is incremented by 8. Since the parsing of an SOP always occurs as the first operation in the interpretation of an SIN, the parse-op-stage command is generally combined with a dispatch fetch command. As will be explained below, the latter command interprets the S-operation as an address in FDISP 24218, and FDISP 24218 in turn produces an address in FUSITT 11012. The latter address is the location of the beginning of the SIN microcode for the SIN.

There are two microcommands for parsing Names:
parse_k_load_epc and parse_k_dispatch_ebox. Both commands obtain a number of bits from INSTB 20262 and place them on NAME Bus 20214. With both microcommands, the syllable size, K, stored in CSSR 24112, determines the number of bits obtained from INSTB 20262. Both commands also increment CPC by the value stored in CSSR 24112. In addition, parse_k_load_epc sets EPC to IPC's value, while parse_k_dispatch_ebox also dispatches EU 10122, i.e., interprets the SOP saved in LOPCODE 24210 as an address in EDISP 24222, which in turn contains an address in EU EUSITT 20344. The EU EUSITT 20344 address is passed via EUDIS Bus 20206 to COMQ 20342 in EU 10122.

c. The S-Interpreters (Fig. 307)

CS 10110 does not assign fixed meanings to SOPs. While all SOPs are 8 bits long, a given 8 bit SOP may have one meaning in one S-Language and a completely different meaning in another S-Language. The semantics of an S-Language's S-operations are determined completely by the S-interpreter for the S-Language. Thus, in order to correctly interpret an S-operation, CS 10110 must know what S-interpreter it is to use. The S-interpreter is identified by a UID pointer with offset 0 in SIP Field 30309 of PED 30303 for

Procedure 602 that CS 10110 is currently executing. In the present embodiment, the UID is the UID of a microcode object which contains FU 10120 microcode. When loaded into FUSITT 11012, the microcode interprets SOPs as defined by the S-Language to which the SOP belongs. In other embodiments, the UID may be the UID of a Procedure Object 608 containing Procedures 602 which interpret the S-Language's SOPs, and in still others, the S-interpreter may be contained in a PROM and the S-interpreter UID may not specify an object, but may serve solely to identify the S-interpreter.

When a Procedure 602 executes an SIN on JP 10114, CS 10110 must translate the value of SIP Pointer 30309 for Procedure 602 and the S-instruction's SOP into a location in the microcode or high-level language code which makes up the S-interpreter. The location obtained by the translation is the beginning of the microcode or high-level language code which implements the SIN. The translation of an SOP together with SIP Pointer 30309 into a location in the S-interpreter is termed dispatching. Dispatching in the present embodiment involves two primary components: a table in memory which translates the value of SIP Pointer 30309 into a small integer called the Dialect Number, and S-operation Decoder Portion 27003 of the FU 10120 micromachine. The following discussion will first present the table and explain how an SIP Pointer 30309 is translated into a Dialect Number, and then explain how the Dialect Number and the SOP together are translated into locations in FUSITT 11012 and EUSITT 20344.

1. Translating SIP into a Dialect Number (Fig. 307)

In the present embodiment, all S-interpreters in CS 10110 are loaded into FUSITT 11012 when CS 10110 begins operation and each S-interpreter is always placed in the same location. Which S-interpreter is used to interpret an S-Language is determined by a value stored in dialect register RDIAL 24212. Consequently, in the present embodiment, a Call to a Procedure 602 whose S-interpreter differs from that of the calling Procedure 602 must translate the UID pointer contained in SIP Field 30309 into a Dialect Number.

Fig. 307 represents the table and microcode which performs this translation in the present embodiment. S-interpreter Translation Table (STT) 30701 is a table which is indexed by small AONs. Each STT Entry (STTE) 30703 has two fields: an AON Field 30705 and a Dialect Number Field 30709. Dialect Number Field 30709 contains the Dialect Number for the S-interpreter object whose AON is in AON Field 30705.

When CS 10110 begins operation, each S-interpreter object is wired active and assigned an AON small enough to serve as an index in STT 30701. By convention, a given S-interpreter object is always assigned the same AON and the same Dialect Number. The AON is placed in AON Field 30705 of STTE 30703 indexed by the AON, and the Dialect Number is placed in Dialect Number Field 30709. Since the S-interpreter objects are wired active, these AONs will never be reassigned to other objects.

On a Call which requires a new S-interpreter, Call microcode obtains the new SIP from SIP Field 30309, calls KOS LAR microcode to translate its UID to its AON, uses the AON to locate the S-interpreter's STTE 30703, and places the value of Dialect Number Field 30709 into RDIAL 21242.

Other embodiments may allow S-interpreters to be loaded into FUSITT 11012 at times other than system initialization, and allow S-interpreters to occupy different locations in FUSITT 11012 at different times. In these embodiments, STT 30701 may be implemented in a manner similar to the implementations of AST 10914 or MHT 10716 in the present embodiment.

2. Dispatching

Dispatching is accomplished by Dispatch Files 27004. These files translate the values provided by RDIAL 24212 and the SOP of the S-instruction being executed into the location of microcode for the SIN specified by the S-operation in the S-interpreter specified by the value of RDIAL 24212. The present embodiment has three dispatch files: FDISP 24218, FALG 24220, and EDISP 24222. FDISP 24218 and FALG 24220 translate S-operations into locations of microcode which executes on FU 10120; EDISP 24222 translates S-operations into locations of microcode which executes on EU 10122. The difference between FDISP 24218 and FALG 24220 is one of speed: FDISP 24218 can translate an SOP in the same microinstruction which performs a parse_op_stage command to load the SOP into LOPCODE 24210. FALG 24220 must perform the translation on a cycle following the one in which the SOP is loaded into LOPCODE 24210. Typically, the location of the first portion of the microcode to execute an S-operation is contained in an FDISP 24218 register, the location of portions executed later is contained in an FALG 24220 register, and the location of microcode for the S-operation which executes on EU 10122 is contained in EDISP 24222.

In the present embodiment, the registers accomplish the translation from S-operation to microcode location as follows: As mentioned in the discussion of FU 10120 hardware, each Dispatch File contains 1024 registers. Each register may contain an address in an S-interpreter. As will be seen in detail later, the address may be an address in an S-interpreter's object, or it may be the address in FUSITT 11012 or EUSITT 20344 of a copy of microcode stored at an S-interpreter address. The registers in the Dispatch Files may be divided into sets of 128 or 256 registers. Each set of registers translates the SOPs for a single S-Language into locations in microcode. Which set of registers is used to interpret a given S-operation is decided by the value of RDIAL 24212; which register in the set is used is determined by the value of the S-operation. The value contained in the specified register is then the location of microcode which executes the S-instruction specified by the S-operation in the S-Language specified by RDIAL 24212.

Logically, the register addressed by the concatenated value in turn contains a 15 bit address which is

the location in the S-interpreter of the first microinstruction of microcode used to execute the S-instruction specified by the S-operation in the S-Language specified by the contents of RDIAL 24212. In the present embodiment, the microcode referred to by the address may have been loaded into FUSITT 11012 and EUSITT 20344 or it may be available only in memory. Addresses of microcode located in FUSITT 11012 and EUSITT 20344 are only eight bits long. Consequently, if a Dispatch File 27004 contains an address which requires more bits than that, the microcode specified by the address is in memory. As described in the discussion of FU 10112 hardware, addresses larger than 8 bits produce an Event Signal, and microcode invoked by the event signal fetches the microinstruction at the specified address in the S-interpreter from memory and loads it into location 0 of FUSITT 11012. The event microcode then returns, and the microinstruction at location 0 is executed. If the next microinstruction also has an address larger than 8 bits, the event signal occurs again and the process described above is repeated.

As previously mentioned, FDISP 24218 is faster than FALG 24220. The reason for the difference in speed is that FDISP registers contain only 6 bits for addressing the S-interpreter. The present embodiment assumes that all microcode addressed via FDISP 24218 is contained in FUSITT 11012. It concatenates 2 zero bits with the six bits in the FDISP 24218 register to produce an 8 bit address for FUSITT 11012. FDISP 24218 registers can thus contain the location of every fourth FUSITT 11012 register between FUSITT register 256 and FUSITT register 448. The microcode loaded into these locations in FUSITT 11012 is microcode for operations which are performed at the start of the SIN by many different SINs. For example, all SINs which perform operations on 2 operands and assign the result to a location specified by a third operand must parse and evaluate the first two operands and parse and resolve the third operand. Only after these operations are done are SINs-specific operations performed. In the present embodiment, the microcode which parses, resolves, and evaluates the operands is contained in a part of FUSITT 11012 which is addressable by FDISP 24218.

As previously mentioned, in the present embodiment, FUSITT 11012 and EUSITT 20344 may be loaded only when CS 10110 is initialized. The microcode loaded into FUSITT 11012 and EUSITT 20344 is produced by the microbinder from the microcode for the various SINs. To achieve efficient use of FUSITT 11012 and EUSITT 20344, microcode for operations shared by various S-interpreters appears only once in FUSITT 11012 and EUSITT 20344. While the SINs in different S-Languages which share the microcode have different registers in FDISP 24218, FALG 24220, or EDISP 24222 as the case may be, the registers for each of the S-instructions contain the same location in FUSITT 11012 or EUSITT 20344.

4. The Kernel Operating System

A. Introduction

Many of the unique properties of CS 10110 are produced by the manipulation of tables in MEM 10112 and Secondary Storage 10124 by programs executing on JP 10114. These programs and tables together make up the Kernel Operating System (KOS). Having described CS 10110's components and the means by which they cooperate to execute computer programs, this specification now presents a detailed account of KOS and of the properties of CS 10110 which it produces. The discussion begins with a general introduction to operating systems, then presents an overview of CS 10110's operating systems, an overview of the KOS, and detailed discussions of the implementation of objects, access control, and Processes 610.

a. Operating Systems (Fig. 401)

In CS 10110, as in other computer systems, the operating system has two functions:

- It controls the use of CS 10110 resources such as JP 10114, MEM 10112, and devices in IOS 10116 by programs being executed on CS 10110.
- It defines how CS 10110 resources appear to users of CS 10110.

The second function is a consequence of the first: By controlling the manner in which executing programs use system resources, the operating system in fact determines how the system appears to its users. Figure 401 is a schematic representation of the relationship between User 40101, Operating System 40102, and System Resources 40103. When User 40101 wishes to use a System Resource 40103, User 40101 requests the use of System Resource 40103 from Operating System 40102, and Operating System 40102 in turn commands CS 10110 to provide the requested Resources 40103. For example, when a user program wishes to use a peripheral device, it does not deal with the device directly, but instead calls the Operating System 40102 procedure 602 that controls the device. While Operating System 40102 must take into account the device's complicated physical properties, the user program that requested the device need know nothing about the physical properties, but must only know what information the Operating System 40102 Procedure 602 requires to perform the operation requested by the user program. For example, while the peripheral device may require that a precise pattern of data be presented to it, the Operating System 40102 procedure 602 may only require the data itself from the user program, and may format the data as required by the peripheral device. The Operating System 40102 Procedure 602 that controls the peripheral device thus transforms a complicated physical interface to the device into a much simpler logical interface.

1. Resources Controlled by Operating Systems (Fig. 402)

Operating Systems 40102 control two kinds of resources: physical resources and virtual resources. The physical resources in the present embodiment of CS 10110 are JP 10114, IOS 10116 and the peripheral

devices associated with IOS 10116, MEM 10112, and Secondary Storage 10124. Virtual resources are resources that the operating system itself defines for users of CS 10110. As was explained above, in controlling how CS 10110's resources are used, Operating System 40102 defines how CS 10110 appears to the users. Instead of the physical resources controlled by Operating System 40102, the user sees a far simpler set of virtual resources. The logical I/O device interface that Operating System 40102 gives the user of a physical I/O device is such a virtual resource. Often, an Operating System 40102 will define sets of virtual resources and multiplex the physical resources among these virtual resources. For instance, Operating System 40102 may define a set of Virtual Processors 612 that correspond to a smaller group of physical processors, and a set of virtual memories that correspond to a smaller group of physical resources. When a user executes a program, it runs on a Virtual Processor 612 and uses virtual memory. It seems to the user of the virtual processor and the virtual memory that he has sole access to a physical processor and physical memory, but in fact, Operating System 40102 is multiplexing the physical processors and memories among the Virtual Processors 612 and virtual memories.

Operating System 40102, too, uses virtual resources. For instance, the memory management portion of an Operating System 40102 may use I/O devices; when it does so, it uses the virtual I/O devices defined by the portion of the Operating System 40102 that manages the I/O devices. One part of Operating System 40102 may also redefine virtual resources defined by other parts of Operating System 40102. For instance, one part of Operating System 40102 may define a set of primitive virtual I/O devices and another part may use these primitive virtual I/O devices to define a set of high-level user-oriented I/O devices. Operating System 40102 thus turns the physical CS 10110 into a hierarchy of virtual resources. How a user of CS 10110 perceives CS 10110 depends entirely on the level at which he is dealing with the virtual resources.

The entity that uses the resources defined by Operating System 40102 is the process. A Process 610 may be defined as the activity resulting from the execution of a program with its data by a sequential processor. Whenever a user requests the execution of a program on CS 10110, Operating System 40102 creates a Process 610 which then executes the Procedures 602 making up the user's program. In physical terms, a process 610 is a set of data bases in memory that contain the current state of the program execution that the process represents. Operating System 40102 causes Process 610 to execute the program by giving Process 610 access to the virtual resources which it requires to execute the program, by giving the virtual resources access to those parts of Process 610's state which they require to perform their operations, and by giving these virtual resources access to the physical resources. The temporary relationship of one resource to another or of a Process 610 to a resource is called a binding. When a Process 610 has access to a given Virtual Processor 612 and Virtual processor 612 has access to process 610's state, process 610 is bound to Virtual Processor 612, and when Virtual Processor 612 has access to JP 10114 and Virtual Processor 612's state is loaded into JP 10114 registers, Virtual processor 612 is bound to JP 10114, and JP 10114 can execute SINs contained in Procedures 602 in the program being executed by Process 610 bound to Virtual Processor 612. Binding and unbinding may occur many times in the course of the execution of a program by a Process 610. For instance, if a Process 610 executes a reference to data and the data is not present in MEM 10112, then Operating System 40102 unbinds Process 610's Virtual Processor 612 from JP 10114 until the data is available in MEM 10112. If the data is not available for an extended period of time, or if the user for whom Process 610 is executing the program wishes to stop the execution of the program for a while, Operating System 40102 may unbind process 610 from its Virtual Processor 612. Virtual Processor 612 is then available for use by other Processes 610.

As mentioned above, the binding process involves giving a first resource access to a second resource, and using the first resource's state in the second resource. To permit binding and unbinding, Operating System 40102 maintains data bases that contain the current state of each resource and each Process 610. State may be defined as the information that the operating system must have to use the resource or execute the Process 610. The state of a line printer, for instance, may be variables that indicate whether the line printer is busy, free, off line, or out of order. A Process 610's state is more involved, since it must contain enough information to allow Operating System 40102 to bind Process 610 to a Virtual Processor 612, execute Process 610 for a while, unbind Process 610, and then rebind it and continue execution where it was halted. A process 610's state thus includes all of the data used by Process 610 up to the time that it was unbound from a Virtual Processor 612, along with information indicating whether Process 610 is ready to begin executing again.

Figure 402 shows the relationship between Processes 610, virtual, and physical resources in an operating system. The figure shows a multi-process Operating System 40102, that is, one that can multiplex CS 10110 resources among several Processes 610. The Processes 610 thus appear to be executing concurrently. The solid arrows in Figure 402 indicate bindings between virtual resources or between virtual and physical resources. Each Process 610 is created by Operating System 40102 to execute a user program. The program consists of Procedures 602, and Process 610 executes Procedures 602 in the order prescribed by the program. Processes 610 are created and managed by a component of Operating System 40102 called the Process Manager. Process Manager 40203 executes a Process 610 by binding it to a Virtual Processor 612. There may be more Processes 610 than there are Virtual Processors 612. In this case, Operating System 40102 multiplexes Virtual Processors 612 among Processes 610.

Virtual Processors 612 are created and made available by another component of Operating System 40102, Virtual Processor Manager 40205. Virtual Processor Manager 40205 also multiplexes JP 10114

among Virtual Processors 612. If a Virtual Processor 612 is ready to run, Virtual Processor Manager 40205 binds it to JP 10114. When Virtual Processor 612 can run no longer, or when another Virtual Processor 612 requires JP 10114, Virtual Processor Manager 40205 unbinds running Virtual Processor 612 from JP 10114 and binds another Virtual Processor 612 to it.

Virtual Processors 612 use virtual memory and I/O resources to perform memory access and input-output. Virtual Memory 40206 is created and managed by Virtual Memory Manager 40207, and Virtual I/O Devices 40208 are created and managed by Virtual I/O Manager 40209. Like Virtual Processor Manager 40205, Components 40207 and 40209 of Operating System 40102 multiplex physical resources among the virtual resources. As described above, one set of virtual resources may use another set. One way in which this can happen is indicated by the broken arrows in Figure 402. These arrows show a binding between Virtual Memory 40206 and Virtual I/O Device 40208. This binding occurs when Virtual Memory 40206 must handle a reference to data contained on a peripheral device such as a disk drive. To the user of Virtual Memory 40206, all data appears to be available in MEM 10110. In fact, however, the data is stored on peripheral devices such as disk drives, and copied into MEM 10112 when required. When a Process 610 references data that has not been copied into MEM 10112, Virtual Memory 40206 must use IOS 10116 to copy the data into MEM 10112. In order to do this, it uses a Virtual I/O Device 40208 provided by Virtual I/O Manager 40209.

b. The Operating System in CS 10110

For the sake of clarity, Operating System 40102 has been described as though it existed outside of CS 10110. In fact, however, Operating System 40102 itself uses the resources it controls. In the present embodiment, parts of Operating System 40102 are embodied in JP 10114 hardware devices, parts are embodied in microcode which executes on JP 10114, and parts are embodied in Procedures 602. These Procedures 602 are sometimes called by Processes 610 executing user programs, and sometimes by special Operating System Processes 610 which do nothing but execute operations for Operating System 40102.

The manner in which the components of Operating System 40102 interact may be illustrated by the way in which CS 10110 handles a page fault, i.e., a reference to data which is not available in MEM 10110. The first indication that there may be a page fault is an ATU Miss Event Signal. This Event Signal is generated by ATU 10228 in FU 10120 when there is no entry in ATU 10228 for a Logical Descriptor 27116 used in a read or write operation. The Event Signal invokes Operating System 40102 microcode, which examines a table in MEM 10112 in order to find whether the data described by Logical Descriptor 27116 has a copy in MEM 10112. If the table indicates that there is no copy, Operating System 40102 microcode communicates the fact of the page fault to an Operating System 40102 Virtual Memory Manager process 610 and removes Virtual Processor 612 bound to the Process 610 which was executing when the page fault occurred from JP 10114. Some time later, Virtual Memory Manager Process 610 is bound to JP 10114. Procedures 602 executed by Virtual Memory Manager Process 610 then initiate the I/O operations required to locate the desired data in Secondary Storage 10124 and copy it into MEM 10112. When the data is available in MEM 10112, Operating System 40102 allows Virtual Processor 612 bound to Process 610 which was executing when the page fault occurred to return to JP 10114. Virtual Processor 612 repeats the memory reference which caused the page fault, and since the data is now in MEM 10112, the reference succeeds and execution of Process 610 continues.

c. Extended Operating System and the Kernel Operating System (Fig. 403)

In CS 10110, Operating System 40102 is made up of two component operating systems, the Extended Operating System (EOS) and the Kernel Operating System (KOS). The KOS has direct access to the physical resources. It defines a set of primitive virtual resources and multiplexes the physical resources among the primitive virtual resources. The EOS has access to the primitive virtual resources defined by KOS, but not to the physical resources. The EOS defines a set of user-level virtual resources and multiplexes the primitive virtual resources defined by KOS among the user level virtual resources. For example, KOS provides EOS with Processes 610 and Virtual processors 612 and binds Virtual Processors 612 to JP 10114, but EOS decides when a Process 610 is to be created and when a process 610 is to be bound to a Virtual processor 612.

Figure 403 shows the relationship between a user Process 610, EOS, KOS, and the physical resources in CS 10110. Figure 403 shows three levels of interface between executing user Process 610 and JP 10114. The highest level of interface is Procedure Level 40302. At this level, Process 610 interacts with CS 10110 by calling Procedures 602 as specified by the program Process 610 is executing. The calls may be either calls to User Procedures 40306 or calls to EOS Procedures 40307. When Process 610 is executing a procedure 602, Process 610 produces a stream of SINs. The stream contains two kinds of SINs, S-language SINs 40310 and KOS SINs 40311. Both kinds of SINs interact with CS 10110 at the next level of interface, SIN-level Interface 40309. SINs 40310 and 40311 are interpreted by Microcode 40312 and 40313, and Microinstructions 40315 interact with CS 10110 at the lowest level of interface, JP 10114 Interface 40316. As already explained in the discussion of the FU 10120 micromachine, certain conditions in JP 10114 result in

Event Signals 40314 which invoke microroutines in S-interpreter Microcode 40312 or KOS Microcode 40313. Only Procedure-Level Interface 40302 and SIN-level Interface 40309 are visible to users. Procedure-level Interface 40309 appears as calls in user Procedures 602 or as statements in user Procedures 602 which compilers translate into calls to EOS procedures 602. SIN-level Interface 40309 appears as the Name Tables 10335 and SINs in Procedure Objects 608 generated by compilers.

As Figure 403 indicates, EOS exists only at Procedural Level 40302, while KOS exists at Procedural Level 40302, and SIN Level 40304, and within the microcode beneath SIN Level 40309. The only portion of the operating system that is directly available to user Processes 610 is EOS Procedures 40307. EOS Procedures 40307 may in turn call KOS procedures 40308. In many cases, an EOS Procedure 40307 will contain nothing more than the call to a KOS Procedure 40308.

User Procedures 40306, EOS Procedures 40307, and KOS Procedures 40308 all contain S-language SINs 40310. In addition, KOS Procedures 40308 only may contain special KOS SINs 40311. Special KOS SINs 40311 control functions that are not available to EOS Procedures 40307 or User Procedures 40306, and KOS SINs 40311 may therefore not appear in Procedures 40306 or 40307. S-language SINs 40310 are interpreted by S-interpreter Microcode 40312, while KOS SINs 40311 are interpreted by KOS Microcode 40313. KOS Microcode 40313 may also be called by S-interpreter Microcode 40313. Depending on the hardware conditions that cause Event Signals 40314, Signals 40314 may cause the execution of either S-interpreter Microcode 40312 or KOS Microcode 40313.

Figure 403 shows the system as it is executing a user Process 610. There are in addition special Processes 610 reserved for KOS and EOS use. These Processes 610 work like user Processes 610, but carry out operating system functions such as process management and virtual memory management. With one exception, EOS Processes 610 call EOS Procedures 40307 and KOS Procedures 40308, while KOS Processes 610 call only KOS Procedures 40308. The exception is the beginning of Process 610 execution: KOS performs the KOS-level functions required to begin executing a Process 610 and then calls EOS. EOS performs the required EOS level functions and then calls the first User Procedure 40306 in the program Process 610 is executing.

A description of how KOS handles page faults can serve to show how the parts of the system at the JP 10114—, SIN—, and procedure Levels work together. A page fault occurs when a Process 610 references a data item that has no copy in MEM 10112. The page fault begins as an Event Signal from ATU 10228. The Event Signal invokes a microroutine in KOS Microcode 40313. If the microroutine confirms that the referenced data item is not in MEM 10112, it records the fact of the page fault in some KOS tables in MEM 10112 and calls another KOS microroutine that unbinds Virtual Processor 612 bound to Process 610 that caused the page fault from JP 10114 and allows another Process 610's Virtual Processor 612 to run. Some time after the page fault, a special operating system Process 610, the Virtual Memory Manager Process 610, runs and executes KOS Procedures 40309. Virtual Memory Manager Process 610 initiates the I/O operation that reads the data from Secondary Storage 10124 into MEM 10112. When IOS 10116 has finished the operation, Process 610 that caused the page fault can run again and Virtual Memory Manager Process 610 performs an operation which causes Process 610's Virtual Processor 612 to again be bound to JP 10114. When Process 610 resumes execution, it again attempts to reference the data. The data is now in MEM 10112 and consequently, the page fault does not recur.

The division of Operating System 40102 into two hierarchically-related operating systems is characteristic for CS 10110. Several advantages are gained by such a division:

- Each of the two operating systems is simpler than a single operating system would be. EOS can concern itself mainly with resource allocation policy and high-level virtual resources, while KOS can concern itself with low-level virtual resources and hardware control.
- Because each operating system is simpler, it is easier to verify that each system's components are performing correctly, and the two systems are therefore more dependable than a single system.
- Dividing Operating System 40102 makes it easier to implement different embodiments of CS 10110. Only the interface provided by EOS is visible to the user, and consequently, the user interface to the system can be changed without altering KOS. In fact, a single CS 10110 may have a number of EOSs, and thereby present different interfaces to different users. Similarly, changes in the hardware affect the implementation of the KOS, but not the interface that KOS provides EOS. A given EOS can therefore run on more than one embodiment of CS 10110.
- A divided operating system is more secure than a single operating system. Physical access to JP 10114 is provided solely by KOS, and consequently, KOS can ensure that users manipulate only those resources to which they have access rights.

All CSs 10110 will have the virtual resources defined by KOS, while the resources defined by EOS will vary from one CS 10110 to another and even within a single CS 10110. Consequently, the remainder of the discussion will concern itself with KOS.

The relationship between the KOS and the rest of CS 10110 is governed by four principles:

- Only the KOS has access to the resources it controls. User calls to EOS may result in EOS calls to KOS, and S-language SINs may result in invocations of KOS microcode routines, but neither EOS nor user programs may directly manipulate resources controlled by KOS.
- The KOS is passive. It responds to calls from the EOS, to microcode invocations, and to Event Signals, but it initiates no action on its own.

— The KOS is invisible to all system users but the EOS. KOS does not affect the logical behavior of a Process 610 and is noticeable to users only with regard to the speed with which a Process 610 executes on CS 10110.

As discussed above, KOS manages both physical and virtual resources. The physical resources and some of the virtual resources are visible only within KOS; others of the virtual resources are provided to EOS. Each virtual resource has two main parts: a set of data bases that contain the virtual resource's state, and a set of routines that manipulate the virtual resource. The set of routines for a virtual resource are termed the resource's manager. The routines may be KOS Procedures 40308, or they may be KOS Microcode 40313. As mentioned, in some cases, KOS uses separate Processes 610 to manage the resources.

For the purposes of this specification, the resources managed by KOS fall into two main groups: those associated with objects, and those associated with Processes 610. In the following, first those resources associated with objects, and then those associated with Processes 610 are discussed.

B. Objects and Object Management (Fig. 404)

The virtual resources termed objects are defined by KOS and manipulated by EOS and KOS. Objects as seen by EOS have five properties:

— A single UID that identifies the object throughout the object's life and specifies what Logical Allocation Unit (LAU) the object belongs to.
— A set of attributes that describe the object and limit access to it.
— Bit-addressable contents. I the present embodiment, the contents may range from 0 to (2**32) — 1 bits in length. Any bit in the contents may be addressed by an offset.
— Objects may be created.
— Objects may be destroyed.

All of the data and Procedures 602 in a CS 10110 are contained in objects. Any process 610 executing on a CS 10110 may use a UID-off set address to attempt to access data or Procedures 602 in certain objects on any CS 10110 accessible to the CS 10110 on which Process 610 is executing. The objects which may be thus accessed by any Process 610 are those having UIDs which are guaranteed unique for all present and future CS 10110. Objects with such unique UIDs thus form a single address space which is at least potentially accessible to any process 610 executing on any CS 10110. As will be explained in detail later, whether a Process 610 can in fact access an object in this single address space depends on whether Process 610 has access rights to the object. Other objects, whose UIDs are not unique, may be accessed only by Processes 610 executing on CSs 10110 or groups of CSs 10110 for which the non-unique UID is in fact unique. No two objects accessible to a CS 10110 at a given time may have identical UIDs.

The following discussion of objects will first deal with objects as they are seen directly by EOS and indirectly by user programs, and then deal with objects as they appear to KOS.

Figure 404 illustrates how objects appear to EOS. The object has three parts: the UID 40401, the Attributes 40404, and the Contents, 40406. The object's contents reside in a Logical Allocation Unit (LAU), 40405. UID 40401 has two parts: a LAU Identifier (LAUID) 40402 that indicates what LAU 40405 the object is on, and the Object Serial Number (OSN) 40403, which specifies the object in LAU 40405.

The EOS can create an object on a LAU 40405, and given the object's UID 40401, can destroy the object. In addition, EOS can read and change an object's Attributes 40404. Any Process 610 executing on a CS 10110 may reference information in an object by specifying the object's UID 40401 and the bit in the object at which the information begins. At the highest level, addresses in CS 10110 thus consist of a UID 40401 specifying an object and an offset specifying the number of bits into the object at which the information begins. As will be explained in detail below, KOS translates such UID-offset addresses into intermediate forms called AON-offset addresses for use in JP 10114 and into page number-displacement addresses for use in referencing information which has been copied into MEM 10112.

The physical implementation and manipulation of objects is restricted solely to KOS. For instance, objects and their attributes are in fact stored in Secondary Storage 10124. When a program references a portion of an object, KOS copies that portion of the object from Secondary Storage 10124 into MEM 10112, and if the portion in MEM 10112 is changed, updates the copy of the object in Secondary Storage 10124. EOS and user programs cannot control the location of an object in Secondary Storage 10124 or the location of the copy of a portion of an object in MEM 10112, and therefore can access the object only by means of KOS.

While EOS cannot control the physical implementation of an object, it can provide KOS with information that allows KOS to manage objects more effectively. Such information is termed hints. For instance, KOS generally copies a portion of an object into MEM 10112 only if a Process 610 references information in the object. However, EOS schedules Process 610 execution, and therefore can predict that certain objects will be required in the near future. EOS can pass this information on to KOS, and KOS can use the information to decide what portions of objects to copy into MEM 10112.

a. Objects and User Programs (fig. 405)

As stated above, user programs manipulate objects, but the objects are generally not directly visible to user programs. Instead, user programs use symbols such as variable names or other references to refer to

data stored in objects or file names to refer to the objects themselves. The discussion of Namespace has already illustrated how CS 10110 compilers translate variable names appearing in statements in Procedures 602 into Names, i.e., indexes of NTEs 30401, how Name Resolve microcode resolves NTE 30401 into Logical Descriptors 27116, and how ATU 10228 translates Logical Descriptors 27116 into locations in MEM 10112 containing copies of the portions of the objects in which the data represented by the variables resides.

The translation of filenames to UIDs 40401 is accomplished by EOS. EOS maintains a filename translation table which establishes a relationship between a system filename called a pathname and the UID 40401 of the object containing the file's data, and thereby associates the pathname with the object. A Pathname is a sequence of ASCII characters which identifies a file to a user of CS 10110. Each pathname in a given CS 10110 must be unique. Figure 405 shows the filename translation table. Referring to that figure, when a user gives pathname 40501 to the EOS, EOS uses Filename Translation Table 40503 to translate pathname 40501 into UID 40401 for object 40504 containing the file. An object in CS 10110 may thus be identified in two ways: by means of its UID 40401 or by means of a Pathname 40501. While an object has only a single UID 40401 throughout its life, the object may have many Pathnames 40501. All that is required to change an object's pathname 40501 is the substitution of one Pathname 40501 for another in the object's Entry 40502 in Filename Translation Table 40503. One consequence of the fact that an object may have different Pathnames 40501 during its life is that when a program uses a Pathname 40501 to identify an object, a user of CS 10110 may make the program process a different object simply by giving the object which formerly had Pathname 40501 which appears in the program a new Pathname 40501 and giving the next object to be processed the Pathname 40501 which appears in the program.

In the present embodiment, an object may contain only a single file, and consequently, a Pathname 40501 always refers to an entire object. In other embodiments, a Pathname 40501 may refer to a portion of an object, and in such embodiments, Filename Translation Table 40503 will associate a Pathname 40501 with a UID-offset address specifying the beginning of the file.

b. UIDs 40401 (Fig. 406)

UIDs 40401 may identify objects and other entities in CS 10110. Any entity identified by a UID 40401 has only a single UID throughout its life. Figure 406 is a detailed representation of a CS 10110 UID 40401. UID 40401 is 80 bits long, and has two fields. Field 40402, 32 bits long, is the Logical Allocation Unit Identifier (LAUID). It specifies LAU 40405 containing the object. LAUID 40402 is further subdivided into two subfields: LAU Group Number (LAUGN) 40607 and LAU Serial Number (LAUSN) 40605. LAUGN 40607 specifies a group of LAUs 40405, and LAUSN 40605 specifies a LAU 40405 in that group. Purchasers of CS 10110 may obtain LAUGNs 40607 from the manufacturer. The manufacturer guarantees that he will assign LAUGN 40607 given the purchaser to no other CS 10110, and thus these LAUGNs 40607 may be used to form UIDs 40401 which will be unique for all CSs 10110. Field 40604, 48 bits long, is the Object Serial Number (OSN). It specifies the object in LAU 40405.

UIDs 40401 are generated by KOS Procedures 602.

There are two such procedures 602, one which generates UIDs 40401 which identify objects, and another which generates UIDs 40401 which identify other entities in CS 10110. The former Procedure 602 is called Generate Object UID, and the latter Generate Non-object UID. The Generate Object UID Procedure 602 is called only by the KOS Create Object Procedure 602. Create Object Procedure 602 provides Generate Object UID Procedure 602 with a LAUID 40402, and Generate Object UID Procedure 602 returns a UID 40401 for the object. In the present embodiment, UID 40401 is formed by taking the current value of the architectural clock, contained in a location in MEM 10112, forming an OSN 40403 from the architectural clock's current value, and concatenating OSN 40403 to LAUID 40402.

Generate Non-object UID Procedure 602 may be invoked by EOS to provide a UID 40401 which does not specify an object. Non-object UIDs 40401 may be used in CS 10110 wherever a unique label is required. For example, as will be explained in detail later, all Virtual processors 612 which are available to CS 10110 have non-object UIDs 40401. All such non-object UIDs 40401 have a single LAUSN 40607, and thus, EOS need only provide a LAUGN 40605 as an argument. Generate Non-object UID Procedure 602 concatenates LAUGN 40605 with the special LAUSN 40607, and LAUID 40402 thus produced with an OSN 40403 obtained from the architectural clock. In other embodiments, OSNs 40403 for both object and non-object UIDs 40401 may be generated by other means, such as counters.

CS 10110 also has a special UID 40401 called the Null UID 40401. The Null UID 40401 contains nothing but 0 bits, and is used in situations which require a UID value which cannot represent an entity in CS 10110.

c. Object Attributes

What a program can do with an object is determined by the object's Attributes 40404. There are two kinds of Attributes 40404: Object Attributes and Control Attributes. Object Attributes describe the object's contents; Control Attributes control access to the object. Objects may have Attributes 40404 even though they have no Contents 40406, and in some cases, objects may even exist solely for their Attributes 40404.

For the purposes of this discussion, there are two kinds of Object Attributes: the Size Attribute and the Type Attributes.

An object's Size Attribute indicates the number of bits that the object currently contains. On each

reference to an object's Contents 40406, KOS checks to make sure that the data accessed does not extend beyond the end of the object. If it does, the reference is aborted.

The Type Attributes indicate what kind of information the object contains and how that information may be used. There are three categories of Type Attributes: the Primitive Type Attributes, the Extended Type Attribute, and the Domain of Execution attribute. An object's Primitive Type Attribute indicates whether the object is a data object, a Procedure Object 608, an Extended Type Manager, or an S-interpreter. As their names imply, data objects contain data and Procedure Objects 608 contain Procedures 602. Extended Type Managers (ETMs) are a special type of Procedure Object 608 whose Procedures 608 may perform operations solely on objects called Extended Type Objects. Extended Type Objects (ETOs) are objects which have an Extended Type Attribute in addition to their Primitive Type Attribute; for details, see the discussion of the Extended Type Attribute below. S-interpreters are objects that contain interpreters for S-languages. In the present embodiment, the interpreters consist of dispatch tables and microcode, but in other embodiments, the interpreters may themselves be written in high-level languages. Like the Length Attribute, the Primitive Type Attributes allow KOS to ensure that a program is using an object correctly. For instance, when the KOS executes a call for a Procedure 602 it checks whether the object specified by the call is a Procedure Object 608. If it is not, the call fails.

d. Attributes and Access Control

The remaining Object Attributes and the Control Attributes are all part of CS 10110's Access Control System. The Access Control System is discussed in detail later; here, it is dealt with only to the extent required for the discussion of objects. In CS 10110, an access of an object occurs when a Process 610 fetches SINs contained in a Procedure Object 608, reads data from an object, writes data to an object, or in some cases, when Process 610 transfers control to a Procedure 602. The Access Control System checks whether a Process 610 has the right to perform the access it is attempting. There are two kinds of access in CS 10110, Primitive Access and Extended Access. Primitive Access is access which the Access Control System checks on every reference to an object by a Process 610; Extended Access is access that is checked only on user request. Primitive access checks are performed on every object; extended access checks may be performed only on ETOs, and may be performed only by Procedures 602 contained in ETMs.

The means by which the Access Control System checks a Process 610's access to an object are Process 610's subject and the object's Access Control Lists (ACLs). Each Process 610 has a subject made up of four UIDs 40401. These UIDs 40401 specify the following:

— The user for whom Process 610 was created. This UID 40401 is termed the principal component of the subject.
— Process 610 itself. This UID 40401 is termed the process component.
— The domain in which Process 610 is currently executing. This UID 40401 is termed the domain component.
— A user-defined subgroup of subjects. This UID 40401 is termed the tag component.

A domain is a group of objects which may potentially be accessed by any Process 610 which is executing a Procedure 602 in one of a group of Procedure Objects 608 or ETMs. Each Procedure Object 608 or ETM has a Domain of Execution (DOE) Attribute. This attribute is a UID 40401, and while a Process 610 is executing a Procedure 602 in that Procedure Object 608 or ETM, the DOE attribute UID 40401 is the domain component in Process 610's subject. The DOE attribute thus defines a group of objects which may be accessed by a Process 610 executing Procedures 602 from Procedure Object 608. The group of objects is called Procedure Object 608's domain. As may be seen from the above definition, a subject's domain component may change on any call to or return from a Procedure 602. The tag component may change whenever the user desires. The principal component and the process component, on the other hand, do not change for the life of Process 610.

The ACLs which make up the other half of the Access Control System are attributes of objects. Each ACL consists of a series of Entries (ACLE), and each ACLE has two parts: a Subject Template and a set of Access Privileges. The Subject Template defines a group of subjects, and the set of Access Privileges define the kinds of access that subjects belonging to the group have to the object. To check whether an access to an object is legal, the KOS examines the ACLs. It allows access only if it finds an ACLE whose Subject Template matches the current subject of Process 610 which wishes to make the access and whose set of Access Privileges includes the kind of access desired by Process 610. For example, a Procedure Object 608 may have an ACL with two entries: one whose Subject Template allows any subject access, and whose set of Access Privileges allows only Execute Access, and another whose Subject Template allows only a single subject access and whose set of Access Privileges allows Read, Write, and Execute Access. Such an ACL allows any user of CS 10110 to execute the Procedures 602 in Procedure Object 608, but only a specified Process 610 belonging to a specified user and executing a specified group of Procedures 602 may examine or modify the Procedures 602 in the Procedure Object 608.

There are two kinds of ACLs. All objects have Primitive Access Control Lists (PACLs); ETOs may in addition have Extended Access Control Lists (EACLs). The subject portion of the ACLE is the same in all ACLs; the two kinds of list differ in the kinds of access they control. The access controlled by the PACL is defined by KOS and is checked by KOS on every attempt to gain such access; the access controlled by the EACL is defined by the user and is checked only when the user requests KOS to do so.

e. Implementation of Objects

1. Introduction (Fig. 407, 408)

The user of a CS 10110 need only concern himself with objects as they have just been described. In order for a Process 610 to reference an object, the object's LAU 40405 must be accessible from CS 10110 upon which Process 610 is running, Process 610 must know the object's UID 40401, and Process 610's current subject must have the right to access the object in the desired manner. Process 610 need know neither how the object's Contents 40406 and Attributes 40404 are stored on CS 10110's physical devices nor the methods CS 10110 uses to make the object's Contents 40406 and Attributes 40404 available to Process 610.

The KOS, on the other hand, must implement objects on the physical devices that make up CS 10110. In so doing, it must take into account two sets of physical limitations:

— In logical terms, all CSs 10110 have a single logical memory, but the physical implementation of memory in the system is hierarchical: a given CS 10110 has rapid access to a relatively small MEM 10112, much slower access to a relatively large amount of slow Secondary Storage 10124, and very slow access to LAUs 40405 on other accessible CSs 10110.

— UIDs 40401, and even more, subjects, are too large to be handled efficiently on JP 10114's internal data paths and in JP 10114's registers.

The means by which the KOS overcomes these physical limitations will vary from embodiment to embodiment. Here, there are presented first an overview and then a detailed discussion of the means used in the present embodiment.

The physical limitations of the memory are overcome by means of a Virtual Memory system. The Virtual Memory System creates a one-level logical memory by automatically bringing copies of those portions of objects required by executing Processes 610 into MEM 10112 and automatically copying altered portions of objects from MEM 10112 back to Secondary Storage 10124. Objects thus reside primarily in Secondary Storage 10124, but copies of portions of them are made available in MEM 10112 when a Process 610 makes a reference to them. Besides bringing portions of objects into MEM 10112, when required, the Virtual Memory System keeps track of where in MEM 10112 the portions are located, and when a Process 610 references a portion of an object that is in MEM 10112, the Virtual Memory System translates the reference into a physical location in MEM 10112.

JP 10114's need for smaller object identifiers and subject identifiers is satisfied by the use of internal identifiers called Active Object Numbers (AONs) and Active Subject Numbers (ASNs) inside JP 10114. Each time a UID 40401 is moved from MEM 10112 into JP 10114's registers, it is translated into an AON, and the reverse translation takes place each time an AON is moved from a JP 10114's registers to MEM 10112. Similarly, the current subjects of Processes 610 which are bound to Virtual Processors 612 are translated from four UIDs 40401 into small integer ASNs, and when Virtual Processor 612 is bound to JP 10114, the ASN for the subject belonging to Virtual Processor 612's process 610 is placed in a JP 10114 register. The translations from UID 40401 to AON and vice-versa, and from subject to ASN are performed by KOS.

When KOS translates UIDs 40401 to AONs and vice-versa, it uses AOT 10712. An AOT 10712 Entry (AOTE) for an object contains the object's UID 40401, and the AOTE's index in AOT 10712 is that object's AON. Thus, given an object's AON, KOS can use AOT 10712 to determine the object's UID 40401, and given an obect's UID 40401, KOS can use AOT 10712 to determine the object's AON. If the object has not been referenced recently, there may be no AOTE for the object, and thus no AON for the object's UID 40401. Objects that have no AONs are called inactive objects. If an attempt to convert a UID 40401 to an AON reveals that the object is inactive, an Inactive Object Fault results and KOS must activate the object, that is, it must assign the object an AON and make an AOTE for it.

KOS uses AST 10914 to translate subjects into ASN's. When a Process 610's subject changes, AST 10914 provides Process 610 with the new subject's ASN. A subject may presently have no ASN associated with it. Such subjects are termed inactive subjects. If a subject is inactive, an attempt to translate the subject to an ASN causes KOS to activate the subject, that is, to assign the subject an ASN and make an entry for the subject in AST 10914.

In order to achieve efficient execution of programs by Processes 610, KOS accelerates information that is frequently used by executing processes 610. There are two stages of acceleration:

— Tables that contain the information are wired into MEM 10112, that is, the Virtual Memory System never uses MEM 10112 space reserved for the tables for other purposes.

— Special hardware devices in JP 10114 contain portions of the information in the tables.

MHT 10716, AOT 10712, and AST 10914 are examples of the first stage of acceleration. As previously mentioned, these tables are always present in MEM 10112. Address Translation Unit (ATU) 10228 is an example of the second stage. As previously explained, ATU 10228 is a hardware cache that contains copies of the most recently used MHT 10716 entries. Like MHT 10716, it translates AON offset addresses into the MEM 10112 locations that contain copies of the data that the UID-offset address corresponding to the AON-offset address refers to ATU 10228 is maintained by KOS Logical Address Translation (LAT) microcode.

Figure 407 shows the relationship between ATU 10228, MEM 10112, MHT 10716, and KOS LAT microcode 40704. When JP 10114 makes a memory reference, it passes AON-offset Address 40705 to ATU 10228. If ATU 10228 contains a copy of MHT 10716's entry for Address 40705, it immediately produces the corresponding MEM 10112 Address 40706 and transmits the address to MEM 10112. If there is no copy,

ATU 10228 produces an ATU Miss Event Signal which invokes LAT microcode 40704 in JP 10114. LAT microcode 40704 obtains the MHT entry that corresponds to the AON-offset address from MHT 10716, places the entry in ATU 10228, and returns. JP 10114 then repeats the reference. This time, there is an entry for the reference, and ATU 10228 translates the AON address into the address of the copy of the data contained in MEM 10112.

The relationship between KOS table, hardware cache, and microcode just described is typical for the present embodiment of CS 10110. The table (in this case, MHT 10716), is the primary source of information and is maintained by the Virtual Memory Manager Process, while the cache accelerates portions of the table and is maintained by KOS microcode that is invoked by event signals from the cache.

AOT 10712, AST 10914, and MHT 10716 share another characteristic that is typical of the present embodiment of CS 10110: the tables are constructed in such a fashion that the table entry that performs the desired translation is located by means of a hash function and a hash table. The hash function translates the large UID 40401, subject, or AON into a small integer. This integer is the index of an entry in the hash table. The contents of the hash table entry is an index into AOT 10712, AST 10914, or MHT 10716, as the case may be, and these tables are maintained in such a fashion that the entry corresponding to the index provided by the hash table is either the entry that can perform the desired translation or contains information that allows KOS to find the desired entry. The entries in the tables furthermore contain the values they translate. Consequently, KOS can hash the value, find the entry, and then check whether the entry is the one for the hashed value. If it is not, KOS can quickly go from the entry located by the hash table to the correct entry.

Figure 408 shows how hashing works in AST 10914 in the present embodiment. In the present embodiment, Subject 40801, i.e., the principal, process, and domain components of the current subject, are input into Hash Function 40802. Hash Function 40802 produces the index of an entry in ASTHT 10710. ASTHT Entry 40504 in turn contains the index of an Entry (ASTE) 40806 in AST 10914 These ASTE 40806 indexes are ASNs. ASTE 40806 contains the principal, process, and domain components of some subject and a link field pointing to ASTE 40806'. ASTE 40806' has 0 in its link field, which indicates that it is the last link in the chain of ASTES begining with ASTE 40806. If the hashing of a subject yields ASTE 40806, KOS compares the subject in ASTE 40806 with the hashed subject; if they are identical, ASTE 40806's index in AST 10914 is the subject's ASN. If they are not identical, KOS uses the link in ASTE 40806 to find ASTE 40806'. It compares the subject in ASTE 40806' with the hashed subject; if they are identical, ASTE 40806''s AST index is the subject's ASN; otherwise, ASTE 40806' is the last entry in the chain, and consequently, there is no ASTE 40806 and no ASN for the hashed subject.

In the following, we will discuss the implementation of objects in the present embodiment in detail, beginning with the implementation of objects in Secondary Storage 10124 and proceeding then to CS 10110's Active Object Management System, the Access Control System, and the Virtual Memory System.

2. Objects in Secondary Storage 10124 (Figs. 409, 410)

As described above, objects are collected into LAUs 40405. The objects belonging to a LAU 40405 are stored in Secondary Storage 10124. Each LAU 40405 contains an object whose contents are a table called the Logical Allocation Unit Directory (LAUD). As its name implies, the LAUD is a directory of the objects in LAU 40405. Each object in LAU 40405, including the object containing the LAUD, has an entry in the LAUD. Figure 409 shows the relationship between Secondary Storage 10124, LAU 40405, the LAUD, and objects. LAU 40405 resides on a number of Storage Devices 40904. LAUD Object 40902' in LAU 40405 contains LAUD 40903. Two LAUDEs 40906 are shown. One contains the attributes of LAUD Object 40902 and the location of its contents, and the other contains the attributes of LAUD Object 40902' containing LAUD 40903 and the location of its contents.

KOS uses a table called the Active LAU Table (ALAUT) to locate the LAUD belonging to LAU 40405. Figure 410 illustrates the relationship between ALAUT 41001, ALAUT Entries 41002, LAUs 40405, and LAUD Objects 40902'. Each LAU 40405 accessible to CS 10110 has an Entry (ALAUTE) 41002 in ALAUT 41001. ALAUTE 41002 for LAU 40405 includes LAU 40405's LAUID 40402 and UID 40401 of LAU 40705's LAUD Object 40902'. Hence, given an object's UID 40401, KOS can use UID 40401's LAUID 40402 to locate ALAUTE 41002 for the object's LAU 40405, and can use ALAUTE 41002 to locate LAU 40405's LAUD 40903. Once LAUD 40903 has been found, OSN portion 40402 of the object's UID 40401 provides the proper LAUDE 40906, and LAUDE 40906 contains object's attributes and the location of its contents.

LAUD 40903 and the Procedures 602 that manipulate it belong to a part of KOS termed the Inactive Object Manager. The following discussion of the Inactive Object Manager will begin with the manner in which an object's contents are represented on Secondary Storage 10124, will then discuss LAUD 40903 in detail, and conclude by discussing the operations performed by Inactive Object Manager Procedures 602.

a.a. Representation of an Object's Contents on Secondary Storage 10124

In general, the manner in which an object's contents are represented on Secondary Storage 10124 depends completely on the Secondary Storage 10124. If a LAU 40405 is made up of disks, then the object's contents will be stored in disk blocks. As long as KOS can locate the object's contents, it makes no difference whether the storage is contiguous or non-contiguous.

In the present embodiment, the objects' contents are stored in files created by the Data General

Advance Operating System (AOS) procedures executing on IOS 10116 These procedures manage files that contain objects' contents for KOS. In future CSs 10110, the representation of an object's contents on Secondary Storage 10124 will be managed by a portion of KOS.

b.b. LAUD 40903 (fig. 411, 412)

Figure 411 is a conceptual illustration of LAUD 40903. LAUD 40903 has three parts: LAUD Header 41102, Master Directory 41105, and LAUD Entries (LAUDEs) 40906. LAUD Header 41102 and Master Directory 41105 occupy fixed locations in LAUD 40903, and can therefore always be located from the UID 40401 of LAUD 40903 given in ALAUT 41001. The locations of LAUDEs 40906 are not fixed, but the entry for an individual object can be located from Master Directory 41105.

Turning first to LAUD Header 41102, LAUD Header 41102 contains LAUID 40402 belonging to LAU 40405 to which LAUD 40903 belongs and OSN 40403 of LAUD 40903. As will be explained in greater detail below, KOS can use OSN 40403 to find LAUDE 40906 for LAUD 40903.

Turning now to Master Directory 41105, Master Directory 41105 translates an object's OSN 40403 into the location of the object's LAUDE 40906. Master Directory 41105 contains one Entry 41108 for each object in LAU 40505. Each Entry has two fields: OSN Field 41106 and Offset Field 41107. OSN Field 41106 contains OSN 40403 for the object to which Entry 41108 belongs; Offset Field 41107 contains the offset of the object's LAUDE 40906 in LAUD 40903. KOS orders Entries 41108 by increasing OSN 40403, and can therefore use binary search means to find Entry 41108 containing a given OSN 40403. Once Entry 41108 has been located, Entry 41108's Offset Field 41107, combined with LAUD 40903's OSN 40403, yields the UID offset address of the object's LAUDE 40906.

Once KOS knows the location of LAUDE 40906 it can determine an object's Attributes 40404 and the location of its Contents 40406. Figure 411 gives only an overview of LAUDE 40906's general structure. LAUDE 40906 has three components: a group of fields of fixed size 41109 that are present in every LAUDE 40906, and two variable sized components, one, 41139, containing entries belonging to the object's PACL, and another, 41141, containing the object's EACL.

As the preceding descriptions of the LAUD's components imply, the number of LAUDEs 40906 and Master Directory Entries 41108 varies with the number of objects in LAU 40405. Furthermore, the amount of space required for an object's EACL and PACL varies from object to object. KOS deals with this problem by including Free Space 41123 in each LAUD 40903. When an object is created, or when an object's ACLs are expanded, the Inactive Object Manager expands LAUD 40903 only if there is no available Free Space 41123; if there is Free Space 41123, the Inactive Object Manager takes the necessary space from Free Space 41123; when an object is deleted or an object's ACLs shortened, the Inactive Object Manager returns the unneeded space to Free Space 41123.

Figure 412 is a detailed representation of a single LAUDE 40906. Figure 412 presents those fields of LAUDE 40906 which are common to all embodiments of CS 10110; fields which may vary from embodiment to embodiment are ignored. Starting at the top of Figure 412, Structure Version Field 41209 contains information by which KOS can determine which version of LAUDE 40906 it is dealing with. Size Field 41211 contains the Size Attribute of the object to which LAUDE 40906 belongs. The Size Attribute specifies the number of bits currently contained in the object. Lock Field 41213 is a KOS lock. As will be explained in detail in the discussion of Processes 610, Lock Field 41213 allows only one Process 610 to read or write LAUDE 40906 at a time, and therefore keeps one Process 610 from altering LAUDE 40906 while another Process 610 is reading LAUDE 40906. File Identifier 41215 contains a system identifier for the file which contains the Contents 40406 of the object to which LAUDE 40906 belongs. The form of File Identifier 41215 may vary from embodiment to embodiment; in the present embodiment, it is an AOS system file identifier. UID Field 41217 contains UID 40401 belonging to LAUDE 40906's object. Primitive Type Field 41219 contains a value which specifies the object's Primitive Type. The object may be a data object, a Procedure Object 608, an ETM, or an S-interpreter object. AON Field 41221 contains a valid value only when LAUDE 40906's object is active, i.e., has an entry in AOT 10712. AON Field 41221 then contains the object's AON. If the object is an ETO, Extended Type Attribute Field 41223 contains the UID 40401 of the ETO's ETM. Otherwise, it contains a Null UID 40401. Similarly, if the object is a Procedure Object 608 or an ETM, Domain of Execution Attribute Field 41225 contains the object's Domain of Execution Attribute.

The remaining parts of LAUDE 40906 belong to the Access Control System and will be explained in detail in that discussion. Attribute Version Number Field 41227 contains a value indicating which version of ACLEs this LAUDE 40906 contains, PACL Size Field 41229 and EACL Size Field 41231 contain the sizes of the respective ACLs, PACL Offset Field 41233 and EACL Offset Field 41235 contain the offsets in LAUD 40903 of additional PACLEs 41139 and EACLEs 41141, and fixed PACLEs 41237 contains the portion of the PACL which is always included in LAUDE 40906.

3. Active Objects (fig. 413)

An active object is an object whose UID 40401 has an AON associated with it. In the present embodiment, each CS 10110 has a set of AONs' KOS associates these AONs with UIDs 40401 in such fashion that at any given moment, an AON in a CS 10110 represents a single UID 40401. Inside FU 10120, AONs are used to represent UIDs CS 10110. In the present embodiment, the AON is represented by 14 bits. A 112-bit UID-offset address (80 bits for UID 40401 and 32 for the offset) is thus represented inside FU 10120

by a 46-bit AON-offset address (14 bits for the AON and 32 bits for the offset).

A CS 10110 has far fewer AONs than there are UIDs 40401. KOS multiplexes a CS 10110's AONs among those objects that are being referenced by CS 10110 and therefore require AONs as well as UIDs 40401. While a given AON represents only a single UID 40401 at any given time, at different times, a UID 40401 may have different AONs associated with it.

Figure 413 provides a conceptual representation of the relationship between AONs and UIDs 40401. Each CS 10110 has potential access to 2**80 UIDs 40401. Some of these UIDs, however, represent entities other than objects, and others are never associated with any entity. Each CS 10110 also has a set of AONs 41303 available to it. In the present embodiment, this set may have up to 2**14 values. Since the AONS are only used internally, each CS 10110 may have the same set of AONs 41303. Any AON 41304 in set of AONs 41303 may be associated with a single UID 40401 in set of object UIDs 41301. At different times, an AON 41304 may be associated with different UIDs 40401.

As mentioned above, KOS associates AONs 41304 with UIDs 40401. It does so by means of AOT 10712. Each AOT entry (AOTE) 41306 in AOT 10712 associates a UID 40401 with an AON 41304. AON 41304 is the index of AOTE 41306 which contains UID 40401. Until AOTE 41306 is changed, the AON 41304 which is the index of AOTE 41306 containing UID 40401 represents UID 40401. AOT 10712 also allows UIDs 40401 to be translated into AONs 41303 and vice-versa. Figure 413 illustrates the process for UID-offset Address 41308 and AON-offset Address 41309. AOTE 41306 associates AON 41304 in AON-offset Address 41309 with UID 40401 in UID-offset Address 41308, and Addresses 41308 and 41309 have the same Offset 41307. Consequently, AON-offset Address 41309 represents UID-offset Address 41308 inside JP 10114. Since both addresses use the same Offset, Address 41309 can be translated into address 41308 by translating Address 41309's AON 41304 into Address 41308's UID 40401, and Address 41308 can be translated into Address 41309 by the reverse process. In both cases, the translation is performed by finding the proper AOTE 41306.

The process by which an object becomes active is called object activation. A UID-offset Address 41308 cannot be translated into an AON-offset Address 41309 unless the object to which UID 40401 of UID-offset Address 41308 belongs is active. If a Process 610 attempts to perform such a translation using a UID 40401 belonging to an inactive object, an Inactive Object Fault occurs. KOS handles the fault by removing Process 610 that attempted the translation from JP 10114 until a special KOS Process called the Object Manager Process has activated the object. After the object has been activated, Process 610 may return to JP 10114 and complete the UID 40401 to AON 41304 translation.

The portion of KOS that manages active objects is called the Active Object Manager (AOM). Parts of the AOM are Procedures 602, and parts of it are microcode routines. The high-level language components of the AOM may be invoked only by KOS processes 610. KOS Active Object Manager Process 610 performs most of the functions involved in active object management.

a.a. UID 40401 to AON 41304 Translation

Generally speaking, in CS 10110, addresses stored in MEM 10112 and Secondary Memory 10124 are stored as UID offset addresses. The only form of address that FU 10120 can translate into a location in MEM 10112 is the AON-offset form. Consequently, each time an address is loaded from MEM 10112 into a FU 10120 register, the address must be translated from a UID-offset address to an AON-offset address. The reverse translation must be performed each time an address is moved from a FU 10120 register back into memory.

Such translations may occur at any time. For example, a running Virtual Processor 612 performs such a translation when the Process 610 being executed by Virtual Processor 612 carries out an indirect memory reference. An indirect memory reference is a reference which first fetches a pointer, that is, a data item whose value is the address of another data item, and then uses the address contained in the pointer to fetch the data itself. In CS 10110, pointers represent UID-offset addresses. Virtual Processor 612 performs the indirect memory reference by fetching the pointer from MEM 10112, placing it in FU 10120 registers, translating UID 40401 represented by the pointer into AON 41304 associated with it, and using the resulting AON-offset address to access the data at the location specified by the address.

Most such translations, however, occur when Virtual Processor 612 state is saved or restored. For instance, when one Process 610's Virtual Processor 612 is removed from JP 10114 and another Process 610's Virtual Processor 612 is bound to JP 10114, the state of Virtual Processor 612 being removed from JP 10114 is stored in memory, and the state of Virtual Processor 612 being bound to JP 10114 is moved into JP 10114's registers. Because only UID-offset addresses may be stored in memory, all of the AON-offset addresses in the state of Virtual Processor 612 which is being removed from JP 10114 must be translated into UID-offset addresses. Similarly, all of the UID-offset addresses in the state of Virtual Processor 612 being bound to JP 10114 must be translated into AON-offset addresses before they can be loaded into FU 10120 registers.

C. The Access Control System

As mentioned in the introduction to objects, each time a process 610 accesses data or SINs in an object, the KOS Access Control System checks whether Process 610's current subject has the right to perform the kind of access that Process 610 is attempting. If Process 610's current subject does not have the proper access, the Access Control System aborts the memory operation which Process 610 was attempting to

carry out. The following discussion presents details of the implementation of the Access Control System, beginning with subjects, then proceeding to subject templates, and finally to the means used by KOS to accelerate access checking.

a. Subjects

A Process 610's subject is part of process 610's state and is contained along with other state belonging to Process 610 in an object called a Process Object. Process Objects are dealt with at length in the detailed discussion of Processes 610 which follows the discussion of objects. While a subject has, as mentioned above, four components, the principal component, the process component, the domain component, and the tag component, the Access Control System in the present embodiment of CS 10110 assigns values to only the first three components and ignores the tag component when checking access.

In the present embodiment, the UIDs 40401 which make up the components of a Process 610's subject are the UIDs 40401 of objects containing information about the entities represented by the UIDs 40401. The principal component's UID 40401 represents an object called the Principal Object. The Principal Object contains information about the user for whom Process 610 was created. For example, the information might concern what access rights the user had to the resources of CS 10110, or it might contain records of his use of CS 10110. The process component's UID 40401 represents the Process Object, while the domain component's UID 40401 represents an object called the Domain Object. The Domain Object contains information which must be accessible to any Process 610 whose subject has the Domain Object's UID 40401 as its domain component. Other embodiments of CS 10110 will use the tag component of the subject. In these embodiments, the tag component's UID 40401 is the UID 40401 of a Tag Object containing at least such information as a list of the subjects which make up the group of subjects represented by the tag component's UID.

b. Domains

As stated above, the subject's domain component is the domain of execution attribute belonging to the Procedure Object 608 or ETM whose code is being executed when the access request is made. The domain component of the subject thus gives Process 610 to which the subject belongs potential access to the group of objects whose ACLs have ACLEs with subject templates containing domain components that match the DOE attribute. This group of objects is the domain defined by the Procedure Object 608 or ETM's DOE attribute. When a Process 610 executes a Procedure 602 from a Procedure Object 608 or ETM with a given DOE attribute, Process 610 is said to be executing in the domain defined by that DOE attribute. As may be inferred from the above, different Procedure Objects 608 or ETMs may have the same DOE attribute, and objects may have ACLEs which make them members of many different domains.

In establishing a relationship between a group of Procedure Objects 608 and another group of objects, a domain allows a programmer using CS 10110 to ensure that a given object is read, executed, or modified only by a certain set of Procedures 602. Domains may thus be used to construct protected subsystems in CS 10110. One example of such a protected subsystem is KOS itself: the objects in CS 10110 which contain KOS tables all have ACLs whose domain template components match only the DOE which represents the KOS domain. The only Procedure Objects 608 and ETMs which have this DOE are those which contain KOS Procedures 602, and consequently, only KOS Procedures 602 may manipulate KOS tables.

Since an object may belong to more than one domain, a programmer may use domains to establish hierarchies of access. For example, if some of the objects in a first domain belong both to the first domain and a second domain, and the second domain's objects all also belong to the first domain, then Procedures 602 contained in Procedure Objects 608 whose DOEs define the first domain may access any object in the first domain, including those which also belong to the second domain, while those from Procedure Objects 608 whose DOEs define the second domain may access only those objects in the second domain.

c. Access Control Lists

As previously mentioned, the Access Control System compares the subject belonging to Process 610 making an access to an object and the kind of access Process 610 desires to make with the object's ACLs to determine whether the access is legal. The following discussion of the ACLs will first deal with Subject Templates, since they are common to all ACLs, and then with PACLs and EACLs.

1. Subject Templates (Fig. 416)

Figure 416 shows Subject Templates, PACL Entries (PACLEs), and EACL Entries (EACLEs). Turning first to the Subject Templates, Subject Template 41601 consists of four components, Principal Template 41606, Process Template 41607, Domain Template 41609, and Tag Template 41611. Each template has two fields, Flavor Field 41603, and UID Field 41605. Flavor Field 41603 indicates the way in which the template to which it belongs is to match the corresponding component of the subject for Process 610 attempting the access. Flavor Field 41603 may have one of three values: match any, match one, match group. If Flavor Field 41603 has the value match any, any subject component UID 40401 matches the template, and the Access Control System does not examine UID Field 41605. If Flavor Field 41603 has the value match one, then the corresponding subject component must have the same UID 40401 as the one contained in UID Field 41605. If Flavor Field 41603 has the value match group, finally, then UID Field 41605 contains a UID 40401 of an

object containing information about the group of subject components which the given subject component may match.

2. Primitive Access Control Lists (PACLs)

PACLs are made up of PACLEs 41613 as illustrated in Figure 416. Each PACLE 41613 has two parts: a subject template 41601 and an Access Mode Bits Field 41615. The values in Access Mode Bits Field 41615 define 11 kinds of access. The eleven kinds fall into two groups: Primitive Data Access and Primitive Non-data Access. Primitive Data Access controls what the subject may do with the object's Contents 40406; Primitive Non-data Access controls what the subject may do with the object's Attributes 40404.

There are three kinds of Primitive Data Access: Read Access, Write Access, and Execute Access. If a subject has Read Access, it can examine the data contained in the object; if the subject has Write Access, it can alter the data contained in the object; if it has Execute Access, it can treat the data in the object as a Procedure 602 and attempt to execute it. A subject may have none of these kinds of access, or any combination of the kinds. On every reference to an object, the KOS checks whether the subject performing the reference has the required Primitive Data Access.

Primitive Non-data Access to an object is required only to set or read an object's Attributes 40404, and is checked only when these operations are performed. The kinds of Non-data Access correspond to the kinds of Attributes 40404:

Attributes	Kind of Access
Object Attributes	get object attributes set object attributes
Primitive Control Attributes	get primitive control attributes set primitive control attributes
Extended Control Attributes	get extended control attributes set extended control attributes
ETM Access	use as ETM create ETO

The access rights for object attributes allow a subject to get and set the object attributes described previously. The access rights for primitive and extended control attributes allow a subject to get and set an object's PACL and EACL respectively.

An object may have any number of PACLEs 41613 in its PACL. The first five PACLEs 41613 in an object's PACL are contained in fixed PACLE Field 41237 of LAUDE 40906 for the object; the remainder are stored in LAUD 40903 at the location specified in PACL Offset Field 41233 of LAUDE 40906.

3. APAM 10918 and Protection Cache 10234 (Fig. 421)

Primitive non-data access rights are checked only when users invoke KOS routines that require such access rights, and extended access rights are checked only when users request such checks. Primitive data access rights, on the other hand, are checked every time a Virtual Processor 612 makes a memory reference while executing a Process 610. The KOS implementation of primitive data access right checking therefore emphasizes speed and efficiency. There are two parts to the implementation: APAM 10918 in MEM 10112, and Protection Cache 10234 in JP 10114. APAM 10918 is in a location in MEM 10112 known to KOS microcode. APAM 10918 contains primitive data access information copied from PACLEs 41613 which belong to active objects and whose Subject Template 41601 matches an active subject. Protection Cache 10234, in turn, contain copies of the information in APAM 10918 for the active subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114 and active objects referenced by Process 610. A primitive data access check in CS 10110 begins with Protection Cache 10234, and if the information is not contained in Protection Cache 10234, proceeds to APAM 10918, and if it is not there, finally, to the object's PACL. The discussion which follows begins with APAM 10918.

Figure 421 shows APAM 10918. APAM 10918 is organized as a two-dimensional array. The array's row indexes are AONs 41304, and its column indexes are ASNs. There is a row for each AON 41304 in CS 10110, and a column for each ASN. In Figure 421, only a single row and column are shown. Any primitive data access information in APAM 10918 for the object represented by AON 41304 j is contained in Row 42104, while Column 42105 contains any primitive data access information in APAM 10918 for the subject

represented by ASN k. APAM Entry (APAME) 42106 is at the intersection of Row 42104 and Column 42105, and thus contains the primitive data access information from that PACLE 41613 belonging to the object represented by AON 41304 j whose Subject Template 41601 matches the subject represented by ASN k.

An expanded view of APAME 42106 is presented beneath the epresentation of APAM 10918. APAME 42106 contains four 1-bit fields. The bits represent the kinds of primitive data access that the subject represented by APAME 42106's column index has to the object represented by APAME 42106's row index.

- Field 42107 is the Valid Bit. If the Valid Bit is set, APAME 42106 contains whatever primitive data access information is available for the subject represented by the column and the object represented by the row. The remaining fields in APAME 42106 are meaningful only if Valid Bit 42107 is set.
- Field 42109 is the Execute Bit. If it is set, APAME 42106's subject has Execute Access to APAME 42106's object.
- Field 42111 is the Read Bit. If it is set, APAME 42106's subject has Read Access to APAME 42106's object.
- Field 42113 is the Write Bit. If it is set, APAME 42106's subject has Write Access to APAME 42106's object.

Any combination of bits in Fields 42109 through 42113 may be set. If all of these fields are set to O, APAME 42106 indicates that the subject it represents has no access to the object it represents.

KOS sets APAME 42106 for an ASN and an AON 41304 the first time the subject represented by the ASN references the object represented by AON 41304. Until APAME 42106 is set, Valid Bit 42107 is set to O. When APAME 42106 is set, Valid Bit 42107 is set to 1 and Fields 42109 through 42113 are set according to the primitive data access information in the object's PACLE 41613 whose Subject Template 41601 matches the subject. When an object is deactivated, Valid Bits 42107 in all APAMEs 42106 in the row belonging to the object's AON 41304 are set to 0; similarly, when a subject is deactivated, Valid Bits 42107 in all APAMEs 42106 in the column belonging to the subject's ASN are set to 0.

4. Protection Cache 10234 and Protection Checking (Fig. 422)

The final stage in the acceleration of protection information is Protection Cache 10234 in JP 10114. The details of the way in which Protection Cache 10234 functions are presented in the discussion of the hardware; here, there are discussed the manner in which Protection Cache 10234 performs access checks, the relationship between protection Cache 10234, APAM 10918, and AOT 10712, and the manner in which KOS protection cache microcode maintains Protection Cache 10234.

Figure 422 is a block diagram of Protection Cache 10234, AOTE 10712, APAM 10918, and KOS Microcode 42207 which maintains Protection Cache 10234. Each time JP 10114 makes a memory reference using a Logical Descriptor 27116, it simultaneously presents Logical Descriptor 27116 and a Signal 42208 indicating the kind of memory operation to Protection Cache 10234 and ATU 10228. Entries 42215 in Protection Cache 10234 contain primitive data access and length information for objects previously referenced by the current subject of Process 610 whose Virtual Processor 612 is currently bound to JP 10114. On every memory reference, Protection Cache 10234 emits a Valid/invalid Signal 42205 to MEM 10112. If Protection Cache 10234 contains no Entry 42215 for AON 41304 contained in Logical Descriptor 27116's AON field 27111, if Entry 42215 indicates that the subject does not have the type of access required by process 610, or if the sum of Logical Descriptor 27116's OFF field 27113 and LEN field 27115 exceed the object's current size, Protection Cache 10234 emits an Invalid Signal 42205. This signal causes MEM 10112 to abort the memory reference. Otherwise, Protection Cache 10234 emits a Valid Signal 42205 and MEM 10112 executes the memory reference.

When Protection Cache 10234 emits an Invalid Signal 42205, it latches Logical Descriptor 27116 used to make the reference into Descriptor Trap 20256, the memory command into Command Trap 27018, and if it was a write operation, the data into Data Trap 20258, and at the same time emits one of two Event Signals to KOS microcode. Illegal Access Event Signal 42208 occurs when Process 610 making the reference does not have the proper access rights or the data referenced extends beyond the end of the object. Illegal Access Event Signal 42208 invokes KOS microcode 42215 which performs a Microcode to Software Call 42217 (described in the discussion of Calls) to KOS Access Control System Procedures 602 and passes the contents of Descriptor Trap 20256, Command Trap 27018, the ASN of Process 610 (contained in a register MGR's 10360), and if necessary, Data Trap 20258 to these Procedures 602. These procedures 602 inform EOS of the protection violation, and EOS can then remedy it.

Cache Miss Event Signal 42206 occurs when there is no Entry 42215 for AON 41304 in protection Cache 10234. Cache Miss Event Signal 42206 invokes KOS Protection Cache Miss Microcode 42207, which constructs missing Protection Cache Entry 42215 from information obtained from AOT 10712 and APAM 10918. If APAM 10918 contains no entry for the current subject's ASN and the AON of the object being referenced, protection Cache Miss Microcode 42207 performs a Microcode-to-software Call to KOS Access Control System Procedures 602 which go to LAUDE 40906 for the object and copy the required primitive data access information from the PACLE 41613 belonging to the object whose Subject Template 41601 matches the subject attempting the reference into APAM 10918. The KOS Access Control System Procedures 602 then return to Cache Miss Microcode 42207, which itself returns. Since Cache Miss Microcode 41107 was invoked by an Event Signal, the return causes JP 10114 to reexecute the memory reference which caused the protection cache miss. If protection Cache 10234 was loaded as a result of the

last protection cache miss, the miss does not recur; if Protection Cache 10234 was not loaded because the required information was not in APAM 10918, the miss recurs, but since the information was placed in APAM 10918 as a result of the previous miss, Cache Miss Microcode 42207 can now construct an Entry 42215 in Protection Cache 10234. When Cache Miss Microcode 42207 returns, the memory reference is again attempted, but this time Protection Cache 10234 contains the information and the miss does not recur.

Cache Miss Microcode 42207 creates a new Protection Cache Entry 42215 and loads it into Protection Cache 10234 as follows: Using AON 41304 from Logical Descriptor 27116 latched into Descriptor Trap 20256 when the memory reference which caused the miss was executed and the current subject's ASN, contained in GR's 10360, Cache Miss Microcode locates APAME 42106 for the subject represented by the ASN and the object represented by AON 41304 and copies the contents of APAME 42106 into a JP 10114 register which may serve as a source for JPD Bus 10142. It also uses AON 41304 to locate AOTE 41306 for the object and copies the contents of Size Field 41519 into another JP 10114 register which is a source for JPD Bus 10142. It then uses three special microcommands, executed in successive microinstructions, to load Protection Cache Entry 42215. The first microcommand loads Protection Cache Entry 42215's TS 24010 with AON 41304 of Logical Descriptor 27116 latched into Descriptor Trap 20256; the second loads the object's size into Protection Cache 10234's EXTENT field, and the third loads the contents of APAME 42106 in the same fashion.

Another microcommand invalidates all Entries 42215 in Protection Cache 10234. This operation, called flushing, is performed when an object is deactivated or when the current subject changes. The current subject changes whenever a Virtual Processor 612 is unbound from JP 10114, and whenever a Process 610 performs a call to or a return from a Procedure 602 executing in a domain different from that in which the calling Procedure 602 or the Procedure 602 being returned to executes in. In the cases of the Call and the unbinding of Virtual Processor 612, the cache flush is performed by KOS Call and dispatching microcode; in the case of object deactivation, it is performed by a KOS procedure using a special KOS SIN which invokes Cache Flush Microcode.

D. Processes

1. Synchronization of Processes 610 and Virtual Processors 612

Since Processes 610 and the Virtual Processors 612 to which they are bound may execute concurrently on CS 10110, KOS must provide means for synchronizing Processes 610 which depend on each other. For example, if process 610 A cannot proceed until Process 610 B has performed some operation, there must be a mechanism for suspending A's execution until B is finished. Generally speaking, four kinds of synchronization are necessary:

- One Process 610 must be able to halt and wait for another Process 610 to finish a task before it proceeds.
- One Process 610 must be able to send another Process 610 a message and wait for a reply before it proceeds.
- When processes 610 share a data base, one Process 610 must be able to exclude other Processes 610 from the data base until the first Process 610 is finised using the data base.
- One Process 610 must be able to interrupt another Process 610, i.e., asynchronously cause the second Process 610 to perform some action.

KOS has internal mechanisms for each kind of synchronization, and in addition supplies synchronization mechanisms to EOS. KOS uses the internal mechanisms to synchronize Virtual Processors 612 and KOS Processes 610, while EOS uses the mechanisms supplied by KOS to synchronize all other Processes 610. The internal mechanisms are the following:

- Event counters, Await Entries, and Await Tables. As will be explained in detail below, Event Counters and Await Entries allow one Process 610 to halt and wait for another Process 610 to complete an operation. Event counters and Await Entries are also used to implement process interrupts. Await Entries are organized into Await Tables.
- Message Queues. Message Queues allow one Process 610 to send a message to another and wait for a reply. Message Queues are implemented with Event Counters and queue data structures.
- Locks. Locks allow one Process 610 to exclude other Processes 610 from a data base or a segment of code. Locks are implemented with Event Counters and devices called Sequencers.

KOS makes Event Counters, Await Entries, and Message Queues available to EOS. It does not provide Locks, but it does provide Sequencers, so that EOS can construct its own Locks. The following discussion will define and explain the logical properties of Event Counters, Await Entries, Message Queues, Sequencers, and Locks. Their implementation in the present embodiment will be described along with the implementation of Processes 610 and Virtual Processors 612.

a. Event Counters 44801, Await Entries 44804, and Await Tables (Fig. 448, 449)

Event Counters, Await Entries, and Await Tables are the fundamental components of the KOS Synchronization System. Figure 448 illustrates Event Counters and Await Entries in the present embodiment. Figure 449 gives a simplified representation of Process Event Table 44705, the present embodiment's Await Tables. Turning first to Figure 448, Event Counter 44801 is an area of memory which

contains a value that may only be increased. In one of the present embodiment, Event Counters 44801 for KOS systems which may not page fault are always present in MEM 10112; other Event Counters 44801 are stored in Secondary Storage 10124 unless a Process 610 has referenced them and thereby caused the VMM System to load them into MEM 10112. The value contained in an Event Counter 44801 is termed an Event Counter Value 44802. In the present embodiment, EventCounter 44801 contains 64 bits of data, of which 60 make up Event Counter Value 44802. Event Counter 44801 may be referred to either as a variable or by means of a 128-bit UID pointer which contains Event Counter 44801's location. The UID pointer is termed an Event Counter Name 44803.

Await Entry 44804 is a component of entries in Await Tables. In the present embodiment, there are two Await Tables: Process Event Table 44705 and Virtual Processor Await Table (VPAT) 45401. VPAT 45401 is always present in MEM 10112. As already mentioned, Figure 449 illustrates PET 44705. Both PET 44705 and UPAT 45401 will be described in detail later. Each Await Entry 44804 contains an Event Counter Name 44803, an Event Counter Value 44802, and a Back Link 44805 which identifies a Process 610 or a Virtual Processor 612. Await Entry 44804 thus establishes a relationship between an Event Counter 44801, an Event Counter Value 44802, and a Process 610 or Virtual processor 612.

Turning now to Figure 449, in the present embodiment, all Await Entries 44804 for user Processes 610 are contained in PET 44705. PET 44705 also contains other information. Figure 449 presents only those parts of PET 44705 which illustrate Await Entries 44804. PET 44705 is structured to allow rapid location of Await Entries 44804 belonging to a specific Event Counter 44801. PET entries (PETEs) 44909 contain links which allow them to be combined into lists in PETE 44705. There are four kinds of lists in PET 44705:

- Event counter lists: these lists link all PETEs 44909 for Event Counters 44801 whose Event Counter Names 44803 hash to a single value.
- Await lists: These lists link all PETEs 44909 for Event Counters 44801 which a given Process 610 is awaiting.
- Interrupt lists: These lists link all PETEs 44909 for Event Counters 44801 which will cause an interrupt to occur for a given Process 610.
- The Free list: PETEs 44909 which are not being used in one of the above lists are on a free list.

Each PETE 44909 which is on an await list or an interrupt List is also on an event counter list.

Turning first to the event counter lists, all PETEs 44909 on a given event counter list contain Event Counter Names 44803 which hash to a single value. The value is produced by Hash Function 44901, and then used as an index in PET Hash Table (PETHT) 44903. That entry in PETHT 44903 contains the index in PET 44705 of that PETE 44909 which is the head of the event counter list. PETE List 44904 represents one such event counter list. Thus, given an Event Counter Name 44803, KOS can quickly find all Await Entries 44804 belonging to Event Counter 44801.

In the present embodiment, the implementation of Event Counters 44801 and tables with Await Entries 44804 involves both Processes 610 and Virtual Processors 612 to which Processes 610 are bound. As will be explained later, a large number of Event Counters 44801 and Await Entries 44804 belonging to Processes 610 are multiplexed onto a small number of Event Counters 44801 and Await Entries 44804 belonging to the Processes' Virtual Processors 612. Await entries 44804 for Event Counters 44801 belonging to Virtual Processors 612 are contained in VPAT 45401.

b. Synchronization with Event Counters 44801 and Await Entries 44804

The simplest form of Process 610 synchronization provided by KOS uses only Event Counters 44801 and Await Entries 44804. Coordination takes place like this: A Process 610 A requests KOS to perform an Await Operation, i.e., to establish one or more Await Entries 44804 and to suspend Process 610 A until one of the Await Entries is satisfied. In requesting the Await Operation, Process 610 A defines what Event Counters 44801 it is awaiting and what Event Counter Values 44802 these Event Counters 44801 must have for their Await Entries 44804 to be satisfied. After KOS establishes Await Entries 44804, it suspends Process 610 A. While process 610 A is suspended, other Processes 610 request KOS to perform Advance Operations on the Event Counters 44801 specified in Process 610 A's Await Entries 44804. Each time a Process 610 requests an Advance Operation on an Event Counter 44801, KOS increments Event Counter 44801 and checks Event Counter 44801's Await Entries 44804. Eventually, one Event Counter 44801 satisfies one of Process 610 A's Await Entries 44804, i.e., reaches a value equal to or greater than the Event Counter Value 44802 specified in its Await Entry 44804 for process 610 A. At this point, KOS allows process 610 A to resume execution. As process 610 A resumes execution, it deletes all of its Await Entries 44804.

E. Virtual Processors 612 (fig. 453)

As previously stated, a Virtual processor 612 may be logically defined as the means by which a Process 610 gains access to JP 10114. In physical terms, a Virtual Processor is an area of MEM 10112 which contains the information that the KOS microcode which binds Virtual Processors 612 to JP 10114 and unbinds them from JP 10114 requires to perform the binding and unbinding operations. Figure 453 shows a Virtual Processor 612. The area of MEM 10112 belonging to a Virtual Processor 612 is Virtual processor 612's Virtual Processor State Block (VPSB) 614. Each Virtual Processor 612 in a CS 10110 has a VPSB 614. Together, the VPSBs 614 make up VPSB Array 45301. Within the Virtual Processor management system, each Virtual Processor 612 is known by its VP Number 45304, which is the index of the Virtual Processor

612's VPSB 614 in VPSB Array 45301. Virtual Processors 612 are managed by means of lists contained in Micro VP Lists (MVPL) 45309. Each Virtual processor 612 has an Entry (MVPLE) 45321 in MVPL 45309, and as Virtual Processor 612 changes state, virtual processor management microcode moves it from one list to another in MVPL 45309.

VPSB 614 contains two kinds of information:

information from Process Object 901 belonging to Process 610 which is bound to VPSB 614's Virtual Processor 612, and information used by the Virtual Processor Management System to manage Virtual Processor 612. The most important information from Process Object 901 is the following:

— Process 610's principal and process UIDs 40401.
— AONs 41304 for Process 610's Stack Objects 44703. (VPSB 614 uses AONs 41304 because KOS guarantees that AONs 41304 belonging to Stack Objects 44703 will not change as long as a Process 610 is bound to a Virtual Processor 612.)

Given AON 41304 of Process 610's SS object 10336, the Virtual Processor Management System can locate that portion of Process 610's state which is moved into registers belonging to JP 10114 when process 610's Virtual Processor 612 is bound to JP 10114. Similarly, when Virtual Processor 612 is unbound from JP 10114, the virtual processor management system can move the contents of JP 10114 registers into the proper location in SS Object 10336.

a. Virtual Processor Managment (Fig. 453)

EOS can perform six operations on Virtual Processors 612:

— Request VP allows EOS to request a Virtual Processor 612 from KOS.
— Release VP allows EOS to return a Virtual Processor 612 to KOS.
— Bind binds a Process 610 to a Virtual Processor 612.
— Unbind unbinds a process 610 from a Virtual Processor 612.
— Run allows KOS to bind Process 610's Virtual Processor 612 to JP 10114.
— Stop prevents KOS from binding process 610's Virtual Processor 612 to JP 10114.

As can be seen from the above list of operations, EOS has no direct influence over the actual binding of a Virtual Processor 612 to JP 10114. This operation is performed by a component of KOS microcode called the Dispatcher. Dispatcher microcode is executed whenever one of four things happens:

— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Await Operation.
— Process 610 whose Virtual Processor 612 is currently bound to JP 10114 executes an Advance Operation which satisfies an Await Entry 44801 for some other Process 610.
— Either Interval Timer 25410 or Egg Timer 25412 overflows, causing an Event Signal which invokes Dispatcher microcode.
— IOJP Bus 10132 is activated, causing an Event Signal which invokes Dispatcher microcode. IOS 10116 activates IOJP bus 10132 when it loads data into MEM 10112 for JP 10114.

When Dispatcher microcode is invoked by one of these events, it examines lists in MVPL 45309 to determine which Virtual Processor 612 is to run next. For the purposes of the present discussion, only two lists are important: the running list and the eligible list. In the present embodiment, the running list, headed by Running List Head 45321, contains only a single MVPLE 45321, that representing Virtual Processor 612 currently bound to JP 10114. In embodiments with multiple JPs 10114, the running list may have more than one MVPLE 45321. The eligible list, headed by Eligible List Head 45313, contains MVPLEs 45321 representing those Virtual Processors 612 which may be bound to JP 10114. MVPLEs 45321 on the eligible list are ordered by priorities assigned Processes 610 by EOS. Whenever KOS Dispatcher microcode is invoked, it compares the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is on the running list with the priority of Process 610 whose Virtual Processor 612's MVPLE 45321 is at the head of the eligible list. If the latter Process 610 has a higher priority, KOS Dispatcher microcode places MVPLE 45321 belonging to the former Process 610's Virtual Processor 612 on the eligible list and MVPLE 45321 belonging to the latter Process 610's Virtual Processor 612 onto the running list. Dispatcher microcode then swaps Processes 610 by moving state in JP 10114 belonging to the former Process 610 onto the former Process 610's SS object 10336 and moving JP 10114 state belonging to the latter Process 610 from the latter Process 610's SS object 10336 into JP 10114.

b. Virtual Processors 612 and Synchronization (Fig. 454)

When a synchronization operation is performed on a Process 610, one of the consequences of the operation is a synchronization operation on a Virtual Processor 612. For example, an Advance Operation which satisfies an Await Entry 44804 for a Process 610 causes an Advance Operation which satisfies a second Await Entry 44804 for Process 610's Virtual Processor 612. Similarly, a synchronization operation performed on a Virtual Processor 612 may have a synchronization operation on Virtual Processor 612's Process 610 as a consequence. For example, if a Virtual Processor 612 performs an operation involving file I/O, Virtual Processor 612's Process 610 must await the completion of the I/O operation.

Figure 454 illustrates the means by which process level synchronization operations result in virtual processor-level synchronization operations and vice-versa. The discussion first describes the components which transmit process-level synchronization operations to Virtual Processors 612 and the manner in which these components operate. Then it describes the components which transmit virtual processor-level

synchronization operations to Processes 610 and the operation of these components.

The first set of components is made up of VPSBA 45301 and VPAT 45401. VPSBA 45301 is shown here with two VPSBs 614: one belonging to a Virtual Processor 612 bound to a user Process 610 and one belonging to a Virtual Processor 612 bound to the KOS Process Manager process 610. VPAT 45401 is a virtual processor-level table of Await Entries 44804. Each Await Entry 44804 is contained in a VPAT Entry (VPATE) 45403. Each Virtual Processor 612 bound to a Process 610 has a VPAT Chunk 45402 of four VPATEs 45403 in VPAT 45401, and can thus await up to four Event Counters 44801 at any given time. The location of a Virtual processor 612's VPAT Chunk 45402 is kept in Virtual Processor 612's VPSB 614. When an Advance Operation satisfies any of the Await Entries 44804 belonging to a Virtual Processor 612, all in Virtual Processor 612's VAT Chunk 45402's Await Entries 44804 are deleted. As in PET 44705, VPATES 45403 containing Await Entries 44804 which are awaiting a given Event Counter 44801 are linked together in a list.

VPATEs 45403 for Virtual Processors 612 bound to user Processes 610 may contain Await Entries 44804 for user Process 610's Private Event Counter 45405. Private Event Counter 45405 is contained in Process 610's Process Object 901. It is advanced each time an Await Entry 44804 in a PETE 44909 on a PET List belonging to Process 610 is satisfied.

The components operate as follows: When KOS performs an Await Operation on Process 610, it makes Await Entries 44804 in both PET 44705 and VPAT 45401 and puts Process 610's VP 612 on the suspended list in MVPL 45309. As previously described, an Await Entry 44804 in PET 44705 awaits an Event Counter 44801 specified in the Await Operation which created Await Entry 44804. Await Entry 44804 in VPAT 45401 awaits Process 610's Private Event Counter 45405. Each time an Await Entry 44804 belonging to Process 610 in PET 44705 is satisfied, Process 610's Private Event Counter 45405 is advanced. The advance of Private Event Counter 45405 satisfies Await Entry 44801 for Process 610's Virtual processor 612 in VPAT 45401, and consequently, KOS deletes Virtual Processor 612's VPATEs 45403 and moves Virtual Processor 612's MVPLE 45321 in MVPL 45309 from the suspended list to the eligible list.

The components which allow a Virtual Processor 612 to transmit a synchronization operation to a process 610 are the following: Outward Signals Object (OSO) 45409, Multiplexed Outward Signals Event Counter 45407, and PET 44705. OSO 45409 contains Event Counters 44801 which KOS FU 10120 microcode advances when it perfoms operations which user Processes 610 are awaiting. Event Counters 44801 in OSO 45409 are awaited by Await Entries 44804 in PET 44705. Each time KOS FU 10120 microcode advances an Event Counter 44801 in OSO 45409, it also advances Multiplexed Outward Signals Event Counter 45407. It is awaited by an Await Entry 44804 in VPAT 45401 belonging to Virtual Processor 612 bound to KOS Process Manager Process 610. When Virtual Processor 62 bound to KOS Process Manager Process 610 is again bound to JP 10114, KOS Process Manager Process 610 examines all PETEs 44909 belonging to the Event Counters 44801 in OSO 45423. If an advance of an Event Counter 44801 in OSO 44801 satisfied a PETE 44909 Process 610, that Process 610's Private Event Counter 45405 is advanced as previously described, and Process 610 may again execute.

A user I/O operation illustrates how the components work together. Each user I/O channel has an Event Counter 44801 in OSO 45409. When a Process 610 performs a user I/O operation on a channel, the EOS I/O routine establish an Await Entry 44804 in the PET 44705 list belonging to Process 610 for the channel's Event Counter 44801 in OSO 45409. When the I/O operation is complete, IOS 10116 places a message to JP 10114 in an area of MEM 10112 and activates IOJP Bus 10132. The activation of IOJP Bus 10132 causes an Event Signal which invokes KOS microcode. The microcode examines the message from IOS 10116 to determine which channel is involved, and then advances Event Counter 44801 for that channel in OSO 45409 and Multiplexed Outward Signals Event Counter 45407. The latter advance satisfies an Await Entry 44804 for Process Manager Process 610's Virtual Processor 612 in VPAT 45401, and Process Manager Process 610 begins executing. Process Manager Process 610 examines OSO 45409 to determine which Event Counters 44801 in OSO 45409 have been advanced since the last time process manager Process 610 executed, and when it finds such an Event Counter 44801, it examines the Event Counter Chain in PET 44705 for that Event Counter 44801. If it finds that the advance satisfied any Await Entries 44804 in the Event Counter Chain, it advances Private Event Counter 45405 belonging to Process 610 specified in Await Entry 44804, thereby causing that Process 610 to resume execution as previously described.

F. Process 610 Stack Manipulation

This section of the specification for CS 10110 describes the manner in which Process 610's MAS 502 and SS 504 are manipulated. As previously mentioned, in CS 10110, a Process 610's MAS 502 and SS 504 are contained in several objects. In the present embodiment, there are five objects, one for each domain's portion of the Macro Stack (MAS) (MAS Objects 10328 through 10324) and one for the Secure Stack (SS) (SS Object 10336). In other embodiments, a Process 610's MAS 502 may contain objects for user-defined domains as well. Though a Process 610's MAS 502 and SS 504 are contained in many objects, they function as a single logical stack. The division into several objects is a consequence of two things: the domain component of the protection system, which requires that an object referenced by a Procedure 602 have Procedure 602's domain of execution, and the need for a location inaccessible to user programs for micromachine state and state which may be manipulated only by KOS.

Stack manipulation takes place under the following circumstances:

— When a procedure 602 is invoked or a Return SIN is executed. Procedure 602 invocations are

performed by means of a Call SIN. Call causes a transfer of control to the first SIN in the invoked Procedure 602 and the Return SIN causes a transfer of control back to the SIN in the invoking Procedure 602 which follows the Call SIN.

— When a non-local Go To SIN is executed. The non-local Go To causes a transfer of control to an arbitrary position in some Procedure 602 which was previously invoked by Process 610 and whose invocation has not yet ended.
— When a condition arises, i.e., an execution of a statement in a program puts the executive Process 610 into a state which requires the execution of a previously established Handler Procedure 602.
— When a Process 610 is interrupted, i.e., when an Interrupt Entry 45718 for Process 610 is satisfied.

Most of the mechanisms involved in stack manipulation are used in Call and Return; these operations are therefore dealt with in detail and the other operations only as they differ from Call and Return. The discussion first introduces Call and Return, then explains the stacks in detail, and finally analyzes Call and Return and the other operations in detail.

1. Introduction to Call and Return

As a Process 610 executes a program, it executes Call and Return SINs. A Call SIN begins an invocation of a procedure 602, and a Return SIN ends the invocation. Generally speaking, a Call SIN does the following:

— It saves the state of Process 610's execution of Procedure 602 which contains the Call SIN. Included in this state is the information required to continue Procedure 602's execution after the Call SIN is finished. This portion of the state is termed calling Procedure 602's Macrostate.
— It creates the state which Process 610 requires to begin execution called Procedure 602.
— It transfers control to the first SIN in the called Procedure 602's code.

The Return SIN does the opposite: it releases the state of called Procedure 602, restores the saved state of calling Procedure 602, and transfers control to the SIN in the calling Procedure 602 following the Call SIN. An invocation of a Procedure 602 lasts from the execution of the Call SIN which transfers control to the Procedure 602 to the execution of the Return SIN which transfers control back to Procedure 602 which contained the Call SIN. The state belonging to a given invocation of a Procedure 602 by a Process 610 is called Procedure 602's invocation state.

While Calls and Returns may be implemented in many different fashions, it is advantageous to implement them using stacks. When a Call creates invocation state for a Procedure 602, that invocation state is added to the top of Process 610's stack. The area of a stack which contains the invocation state of a Procedure 602 is called a frame. Since a called Procedure 602 may call another procedure 602, and that another, a stack may have any number of frames, each frame containing the invocation state resulting from the invocation of a Procedure 602 by Process 610, and each frame lasting as long as the invocation it represents. When called Procedure 602 returns to its caller, the frame upon which it executes is released and the caller resumes execution on its frame. Procedure 602 being currently executed by a Process 610 thus always runs on the top frame of Process 610's MAS 502.

Calls and Returns in CS 10110 behave logically like those in other computer systems using stacks to preserve process 610 state. When a Process 610 executes a Call SIN, the SIN saves as Macrostate the current values of the ABPs, the location of the SIN at which the execution of calling Procedure 602 is to continue, and information such as a pointer to calling Procedure 602's Name Table 10350 and UID 40401 belonging to the S-interpreter object which contains the S-interpreter for Procedure 602's S-language. The Call SIN then creates a stack frame for called Procedure 602, obtains the proper ABP values, the location of called Procedure 602's Name Table 10350 and UID 40401 belonging to its S-interpreter object, and begins executing newly-invoked Procedure 602 on the newly-created stack frame. The Return SIN deletes the stack frame obtains the ABP values and name interpreter information from the Macrostate saved during the Call SIN and then transfers control to the SIN at which execution of calling Procedure 602 is to continue.

However the manner in which Call and Return are implemented is deeply affected by CS 10110's Access Control System Broadly speaking there are two classes of Calls and Returns in CS 0110: those which are mediated by KOS and those which are not. In the following discussion, the former class of Calls and Returns are termed Mediated Calls and Returns, and the latter are called Neighborhood Calls and Returns. Most Calls and Returns executed by CS 10110 are Neighborhood Calls and Returns; Mediated Calls and Returns are typically executed when a user procedure 602 calls EOS Procedures 602 and these in turn call KOS Procedures 602. The Mediated Call makes CS 10110 facilities available to user Processes 610 while protecting these CS 10110 facilities from misuse and therefore generally serves the same purpose as system calls in the present art. As will be seen in the ensuing discussion, Mediated Call requires more CS 10110 overhead than Neighborhood Call but the extra overhead is less than that generally required by system calls in the present art.

Mediated Calls and Returns involve S-interpreter, Namespace, and KOS microcode. S-interpreter and Namespace microcode interpret the Names involved in the call and only modifies those portions of Macrostate accessible to the S-interpreter. The remaining Macrostate is modified by KOS microroutines invoked in the course of the Call SIN. A Mediated Call may be made to any Procedure 602 contained in an object to which Process 610's subject has Execute Access at the time the invocation occurs. Mediated Calls and Returns must be made in the following situations:

— When called Procedure 602 has a different Procedure Environment Descriptor (PED) 30303 from that used by calling Procedure 602. Such Calls are termed Cross-PED Calls.
— When called Procedure 602 is in a different Procedure Object 608 from calling Procedure 602. Such Calls are termed Cross-Procedure Object Calls.
— When called Procedure 602's Procedure Object 608 has a different Domain of Execution (DOE) Attribute from that of calling Procedure 602's Procedure Object 608, and therefore must place its Invocation State on a different MAS object from that used by calling Procedure 602. Such Calls are termed Cross-Domain Calls.

In all of the above Calls, the information required to complete the Call is not available to the S-interpreter and consequently, KOS mediation is required to complete the Call. Neighborhood Calls and Returns only modify two components of Macrostate: the pointer to the current SIN and the FP ABP. Both of these components are available to the S-interpreter as long as called Procedure 602 has the same PED 30303 i.e., uses the same Name Tabe 10350 and S-interpreter or the calling Procedure 602 and has Names with the same syllable size as calling Procedure 602. The Call and Return SINs are specific to each S-language, but they resemble each other in their general behavior. The following discussion will deal exclusively with this general behavior and will concentrate on Mediated Calls and Returns. The discussion first describes MAS 502 and SS 504 belonging to a Process 610 and those parts of Procedure Object 608 involved in Calls and Returns, and then describes the implementation of Calls and Returns.

2. Macro Stacks (MAS) 502 (Fig. 467)

Figure 467 gives an overview of an object belonging to a Process 610's MAS 502. The description of this Figure will be followed by descriptions of other Figures containing detailed representations of portions of MAS objects.

At a minimum MAS Object 46703 comprises KOS MAS Header 10410 together with Unused Storage 46727 reserved for the other elements comprising MAS Object 46703. If Process 610 has not yet returned from an invocation of a Procedure 602 contained in a Procedure Object 608 whose DOE is that required for access to MAS Object 46703. MAS object 46703 further comprises a Stack Base 46703 and at least one MAS Frame 46709.

Each MAS Frame 46709 represents one mediated invocation of a procedure 602 contained in a Procedure Objece 608 with the DOE attribute required by MAS 46703, and may in addition represent neighborhood invocations of Procedures 602 which share that Procedure 602's Procedure Object 608. The topmost MAS Frame 46709 represents the most recent group of invocations of Procedures 602 with the DOE attribute required by MAS Object 46703 and the bottom MAS Frame 46709 the earliest group of invocations from which Process 610 has not yet returned. Frames for invocations of Procedures 602 with other domains of execution are contained in other MAS Objects 46703. As will be explained in detail below MAS Frames 46709 in different MAS objects 46703 are linked by pointers.

MAS Domain Stack Base 46703 has two main parts: KOS MAS Header 10410 which contains information used by KOS microcode which manipulates MAS Object 46703, and Perdomain Information 46707, which contains information about 46703's domain and static information, i.e., information which lasts longer than an invocation used by Procedures 602 with MAS Frames 46709 on MAS Object 46703. MAS Frame 46709 also has two main parts, a KOS Frame Header 10414 which contains information used by KOS to manipulate Frame 46709 and S-interpreter Portion 46713 which contains information available to the S-interpreter when it executes the group of Procedures 602 whose invocations are represented by Frame 46709.

When making Calls and Returns, the S-interpreter and KOS microcode use a group of pointers to locations in MAS Object 46703. These pointers comprise the following:
— MAS Object UID 46715 the UID 40401 of AS Object 46703.
— First Frame Offset (FFO) 46719 which locates the beginning of KOS Frame Header 10414 belonging to the first MAS Frame 46709 in MAS Object 46703.
— Frame Header Pointer (FHP) 46702 which locates the beginning of the topmost KOS Frame Header 10414 in MAS Object 46703.
— Stack Top Offset (STO) 46704 a 32-bit offset from Stack UID 46715 which marks the first bit in Unused Storage 46727.

As will be seen presently all of these pointers are contained in fields in KOS MAS Header 46705.

a.a. MAS Base 10410 (Fig. 468)

Figure 468 is a detailed representation of MAS Domain Stack Base 10410 Turning first to the detailed representation of KOS MAS Header 46705 contained therein, there are the following fields:
— Format Information Field 46801 containing information about the format of KOS MAS Header 46705.
— Flags Field 46803. Of these flags, only one is of interest to the present discussion: Domain Active Flag 46804. This flag is set to TRUE when Process 610 to which MAS Object 46703 belongs is executing the invocation of Procedure 602 whose invocation record makes up the topmost MAS Frame 46709 contained in MAS Object 46703 to which KOS MAS Header 46705 belongs.
— PFO Field 46805: All MAS Headers 46705 and Frame Headers 46709 have fields containing offsets locating the previous and following headers in MAS Object 46703. In a Stack Header 46705 there is no

previous header and this field is set to 0.

- FFO Field 46805: The field locating the following header In a Stack Header 46705 this field contains FFO 46719 since the next header is the first Frame Header in MAS Object 46703.
- STO Field 46807: the field containing STO offset 46704.
- Process ID Field 46809: UID 40401 belonging to Process Object 901 for Process 610 to which MAS Object 46703 belongs.
- Domain Environment Information pointer Field 46811: The pointer contained in the field locates an area which contains domain-specific information. In the present embodiment. the area is part of MAS Stack Base 10410; however, in other embodiments, it may be contained in a separate object.
- Signaller Pointer Field 46813: The pointer contained in the field locates a Procedure 602 which KOS invokes when a Process 610's execution causes a condition to arise while it is executing in the domain to which MAS object 46703 belongs.
- AAT Pointer Field 30211: The pointer in Field 30211 locates AAT 30201 for MAS Object 46703. AAT 30201 is described in detail in Chapter 3.
- Frame Label Sequencer Field 46819: This field contains a Sequencer 45102. Sequencer 45102 is used to generate labels used to locate MAS Frames 46709 when a non-local GOTO is executed.

Turning now to the detailed representation of Domain Environment Information 46821 located by Domain Environment Information Pointer Field 46811 there are the following fields:

- KOS Format Information Field 46823.
- Flags Field 46825 containing the following flags:
  - Pending Interrupt Flag 46827, set to TRUE when Process 610 has an interrupt pending for the domain to which MAS Object 46703 belongs.
  - Domain Dead Flag 46829, set to TRUE when Process 610 can no longer execute Procedures 602 with domains of execution equal to that to which MAS Object 46703 belongs.
  - Invoke Verify on Entry Flag 46833 and Invoke Verify on Exit Flag 46835. The former flag is set to TRUE when KOS is to invoke a Procedure 602 which checks the domain's data bases before a Procedure 602 is allowed to execute on the domain's MAS Object 46703; the latter is set to TRUE when KOS is to invoke such a Procedure 602 on exit from a Procedure 602 with the domain as its DOE.
  - Default Handler Non-null Flag 46835 is set to TRUE when there is a default clean-up handler for the domain. Clean-up handlers are described later.
  - Interrupt Mask Field 46839 determines what interrupts set for Process 610 in MAS object 46703's domain will be honored.
  - Domain UID Field 46841 contains UID 40401 for the domain to which MAS Object 46703 belongs.
  - Fields 46843 through 46849 are pointers to Procedures 602 or tables of pointers to Procedures 602. The Procedures 602 so located handle situations which arise as MASs 502 are manipulated. The use of these fields will become clear as the operations which require their use are explained.

b.b. Per-domain-Data Area 46853 (Fig. 468)

Per-domain Data Area 46853 contains data which cannot be kept in MAS Frames 46709 belonging to invocations of Procedures 602 executing in MAS Object 46703's domain, but which must be available to these invocations Per-Domain Data Area 46853 has two components: Storage Area 46854 and AAT 30201. Storage Area 46854 contains static data used by Procedures 602 with invocations on MAS Object 46703 and data used by S-interpreters which are used by such procedures 602. Associated Address Table (AAT) 30201 is used to locate data in Storage Area 46854. A detailed discussion of AAT 30201 is contained in Chapter 3.

Two kinds of data is stored in Storage Area 46854: static data and S-interpreter data.

Static data is stored in Static Data Block 46863. Static Data Block 46863 comprises two parts: Linkage Pointers 46865 and Static Data Storage 46867 Linkage Pointers 46865 are pointers to static data not contained in Static Data Storage 46867 for example, data which lasts longer than Process 610 and pointers to External Procedures 602 which the Procedure 602 to which Static Data Storage 46867 belongs invokes. Static Data Storage 46867 contains storage for static data used by the Procedure 602 which does not last longer than Process 610 executing the Procedure 602.

S-interpreter data is data required by S-interpreters used by Procedures 602 executing on MAS object 46703.

The S-interpreter data is stored in S-interpreter Environment Block (SEB) 46864 which, like Static Data Block 46864 is located via AAT 30201: The contents of SEB 46864 depend on the S-interpreter.

c.c. MAS Frame 46709 Detail (fig. 469)

Figure 469 represents a typical frame in MAS Object 46703. Each MAS Frame 46709 contains a Mediated Frame 46947 produced by a Mediated Call of a Procedure 602 contained in a Procedure Object 608 whose DOE attribute is the one required for execution on MAS object 46703. Mediated Frame 46947 may be followed by Neighborhood Frames 46945 produced by Neighborhood Calls of Procedures 602. Mediated Frame 46947 has two parts, a KOS Frame Header 10414 which is manipulated by KOS microcode, and an S-interpreter portion which is manipulated by S-interpreter and Namespace microcode. Neighborhood Frames 46945 have no KOS Frame Headers 10414. As will become clear upon closer

examination of Figure 469. Mediated Frames 46947 in the present embodiment contain no Macrostate. In the present embodiment, Macrostate for these frames is kept on SS Object 10336; however in other embodiments, Macrostate may be stored in Mediated Frames 46947. Neighborhood Frames 46945 contain those portions of the macrostate which may be manipulated by Neighborhood Call; the location of this macrostate depends on the Neighborhood Call SIN.

Turning now to KOS Frame Header 10414. there are the following fields:

- KOS Format Information Field 46901 containing information about MAS Frame 46709's format.
- Flags Field 46902. This field contains the following flags:
  - Result of Cross-domain Call Flag 46903. This Flag is TRUE if MAS Frame 46709 which precedes this MAS Frame 46709 is in another MAS Object 46703.
  - Is Signaller Flag 46905. This flag is TRUE if this MAS Frame 46709 was created by the invocation of a Signaller Procedure 602.
  - Do Not Return Flag 46907: This flag is TRUE if Process 610 is not to return to the invocation for which this MAS Frame 46709 was created.
  - Flags 46909 through 46915 indicate whether various lists used in condition handling and non-local GOTOs are present in the MAS Frame 46709.
- Previous Frame Offset Field 46917. Next Frame Offset Field 46919. and Frame Top Offset Field 46921 are offsets which give the location where Header 10414 for the previous MAS Frame 46709 in MAS Object 46703 begins. the location where the header for the next MAS Frame 46709 in MAS Object 46703 begins, and the location of the first bit beyond the top of MAS Frame 46709 respectively.
- Fields 46923 through 46927 are offsets which locate lists in S-interpreter portion 46713 of Frame 46709. KOS establishes such lists to handle conditions and non-local GOTOs. Their use will be explained in detail under those headings.
- Fields 46929 and 46933 contain information about Procedure 602 whose invocation is represented by MAS Frame 46709. Field 46929 contains the number of arguments required by procedure 602 and Field 46933 contains a resolvable pointer to Procedure 602's PED 30303. Both these fields are used primarily for debugging.
- Dynamic Back Pointer Field 46931 contains a resolvable pointer to the preceding MAS Frame 46709 belonging to Process 610's MAS 502 when that MAS Frame 46709 is contained in a different MAS Object 46703. In this case, Flag Field 46903 is set to TRUE. When the preceding MAS Frame 46709 is contained in the same MAS object 46703 field 46931 contains a pointer with a null UID 40401 and Flag Field 46903 is set to FALSE.
- Frame Label Field 46935 is for a Frame Label produced when a non-local GOTO is established which transfers control to the invocation represented by MAS Frame 46709. The label is generated by Frame Label Sequencer 46819 in KOS MAS Header 10410.

S-interpreter Portion 46713 of MAS Frame 46709 comprises those portions of MAS Frame 46709 which are under control of the S-interpreter. S-interpreter Portion 46713 in turn comprises two main subdivisions: those parts belonging to Mediated Frame 46947 and those belonging to Neighborhood Frames 46945.

The exact form of S-interpreter portion 46949 of KOS Frame 46947 and of S-Interpreter Frames 46945 depends on the Call SIN which created the frame in question. However all Neighborhood Frames 46945 and S-interpreter portions 46949 of Mediated Frames 46947 have the same arrangements for storing Linkage Pointers 10416 and local data in the frame. Linkage Pointers 10416 are pointers to the locations of actual arguments used in the invocation and Local Storage 10420 contains data which exists only during the invocation. In all Mediated Frames 46947 and Neighborhood Frames 46945. Linkage pointers 10416 precede Local Storage 10420. Furthermore, when a Mediated Frame 46947 or a Neighborhood Frame 46945 is the topmost frame of Process 610's MAS. i.e, when Process 610 is executing on that frame, the FP always points to the beginning of Local Storage 10420. and the beginning of Linkage Pointers 10416 is always at a known displacement from FP. References to Linkage Pointers 10416 may therefore be expressed as negative offsets from FP, and references to Local Storage 10420 as positive offsets.

In addition, S-interpreter Portion 46713 may contain lists of information used by KOS to execute non-local GOTOs and conditions, as well as S-interpreter frames for non-mediated calls. The lists of information used by KOS are contained in List Area 46943. The exact location of List Area 46943 is determined by the compiler which generates the SINs and Name Table for the Procedure 602 whose invocation is represented by Mediated Frame 46947. When Procedure 602's source text contains statements requiring storage in List Area 46943, the compiler generates SINs which place the required amount of storage in Local Storage 10420. KOS routines then build lists in Area 46943, and place the offsets of the heads of the lists in Fields 46923, 46925 or 46927, depending on the kind of list. The lists and their uses are described in detail later.

3. SS 504 (Fig. 470)

Figure 470 presents an overview of SS 504 belonging to a Process 610. SS 504 is contained in SS Object 10336. SS Object 10336 is manipulated only by KOS microcode routines. Neither Procedures 602 being executed by Process 610 nor S-interpreter or Namespace microcode may access information contained in SS Object 10336.

SS Object 10336 comprises two main components, SS Base 47001 and SS Frames 47003. Turning first to the general structure of SS Frames 47003, each time a Process 610 executes a Mediated Call KOS

microcode creates a new SS Frame 47003 on SS Object 10336 belonging to Process 610 and each time a Process 610 executes a Mediated Return, KOS microcode removes the current top SS Frame 47003 from SS Object 10336. There is thus one SS Frame 47003 on SS Object 10336 belonging to a process 610 for each Mediated Frame 46947 on Process 610's MAS 502.

SS Frames 47003 comprise two kinds of frames:

Ordinary Frames 10510 and Cross-domain Frames 47039. Cross-domain Frames 47039 are created whenever Process 610 executes a Cross-domain Call; for all other Mediated Calls. Ordinary Frames 10510 are created. Cross-domain Frames 47039 divide SS Frames 47003 into Groups 47037 of SS Frames 47003 belonging to sequences of invocations in a single domain. The first SS Frame 47003 in a Group 47037 is a Cross-domain Frame 47039 for the invocation which entered the domain, and the remainder of the SS Frames 47003 are Ordinary Frames 10510 for a sequence of invocations in that domain. These groups of SS Frames 47003 correspond to groups of Mediated Frames 46947 in a single MAS Object 46703.

a.a. SS Base 47001 (Fig. 471)

SS Base 47001 comprises four main parts: SS Header 10512 Process Microstate 47017, Storage Area 47033 for JP 10114 register contents, and Initialization Frame Header 47035. Secure Stack Header 10512 contains the following information:

- Fields 47001 and 47009 contain flag and format information; the exact contents of these fields are unimportant to the present discussion.
- Previous Frame Offset Value Field 47011 is a standard field in headers in SS Object 10336; here it is set to 0, since there is no previous frame.
- Secure Stack First Frame Offset Field 47013 contains the offset of the first SS Frame 47039 in SS object 10336, i.e., Initialization Frame Header 47035.
- Process UID field 47015 contains UID 40401 of Process 610 to which SS Object 10336 belongs.
- Number of Cross Domain Frames Field 47016 contains the number of Cross-domain Frames 47039 in SS Object 10336.

Process Microstate 47017 contains information used by KOS microcode when it executes Process 610 to which SS Object 10336 belongs Fields 47019, 47021 and 47022 contain the offsets of locations in SS Object 10336. Field 47019 contains the value of SSTO the location of the first free bit in SS Object 10336; Field 47021 contains the value of SSFO, the location of the topmost frame in SS object 10336; Field 47022 finally contains the value of XDFO, the location of the topmost Cross-domain Frame 47039 in SS Object 10336. All of these locations are marked in Figure 470.

Other fields of interest in Process Microstate 47017 comprise the following: Offsets in Storage Area Field 47023 contains offsets of locations in Storage Area 47033 of SS Object 10336; Domain Number Field 47025 contains the domain number for the DOE of Procedure 602 currently being executed by Process 610. The relationship between domain UIDs and domain numbers is explained in the discussion of domains. VPAT Offset Field 47027 contains the offset in VPAT 45401 of VPAT Chunk 45402 belonging to Virtual Processor 612 to which Process 610 is bound. Signal Pointer Field 47029 contains a resolved pointer to the Signaller (a Procedure 602 used in condition handling) belonging to the domain specified by Domain Number Field 47025 and Trace Information Field 47031 contains a resolved pointer to that domain's Trace Table, described later.

Storage Area for JP 10114 register Contents 47033 is used when a Virtual Processor 612 must be removed from JP 10114. When this occurs, either because Virtual Processor 612 is unbound from JP 10114, because CS 10110 is being halted, or because CS 10110 has failed, the contents of JP 10114 registers which contain information specific to Virtual Processor 612 are copied into Storage Area 47033. When Virtual Processor 612 is returned to JP 10114, these register contents are loaded back into the JP 10114 registers from whence they came. Initialization Frame Header 47035, finally, is a dummy frame header which is used in the creation of SS Object 10336.

b.b. SS Frames 47003 (Fig. 471)

Commencing the discussion of SS Frames 47039 and 10510. Figure 471 illustrates these structures in detail. Ordinary SS Frame 10510 comprises three main divisions: Ordinary SS Frame Header 10514, Macrostate 10516 and Microstate 10520. Ordinary SS Frame Header 10514 contains information used by KOS microcode to manipulate Ordinary SS Frame 10510 to which Header 10514 belongs. Macrostate 10516 contains the values of the ABPs for the frame's mediated invocation and other information required to resume execution of the invocation. Microstate 10520 contains micromachine state from FU 10120 and EU 10122 registers. The amount of micromachine state depends on the circumstances; in the present embodiment, some micromachine state is saved on all Mediated Calls; furthermore, if a Process 610 executes a microcode-to-software Call, the micromachine state that existed at the time of the call is saved; finally. Microstate 10520 belonging to the topmost SS Frame 47003 may contain information which was transferred from FU 10120 GRF registers 10354 or EU 10122 register and stack mechanism 10216 when their capacity was exceeded. For details about this portion of Microstate 10520 see the discussion of the FU 10120 micromachine in Chapter 2. The discussion of SS Object 10336 continues with details concerning SS Header 10514 and Macrostate 05163.

a.a.a. Ordinary SS Frame Headers 10514 (Fig. 741)

Fields of interest in Ordinary Secure Stack Frame Header 10514 are the following:

- Format Information 47103 which identifies the format of Header 10514.
- Flags Field 47105 which contains one flag of interest in this discussion: Frame Type Flag 47107: in Ordinary SS Frames 10510 this field is set to FALSE.
- Offset Fields 47109 through 47113: Field 47109 contains the offset of the previous SS Frame 47039 or 10510. Field 47111 contains the offset of the following SS Frame 47039 or 10510. and Field 47113 contains the offset of the last SS Frame 47039 or 10510 preceding the next Crossdomain Frame 47039
- Field 47117 contains the current domain number for the domain in which the mediated invocation represent SS Frame 47039 or 10510 is executing.
- Field 47119 contains the offset of the preceding Cross-domain Frame 47039.
- Field 47121 contains offsets for important locations in Microstate 10520.

b.b.b. Detailed Structure of Macrostate 10516 (Fig. 471)

These fields are of interest in Macrostate 10516:

- Syllable Size Field 47125 contains the value of K, i.e., the size of the Names in the SINs belonging to Procedure 602 which the invocation is executing.
- End of Name Table Field 47127 contains the location of the last Name in Name Table 10350 belonging to Procedure 602 which the invocation is executing.
- Fields 47129 through 47143 are resolved pointers to locations in Procedure Object 901 containing Procedure 602 being executed by the invocation and resolved pointers to locations containing data being used by Procedure 602. Field 47129 contains a pointer to Procedure 602's PED 30303; if Procedure 602 is an External Procedure 602. Field 47131 contains a pointer to Procedure 602's entry in Gates 10340; Field 47135 contains the UID-offset value of FP for the invocation; Field 47135 contains a pointer to SEB 46864 used by Procedure 602's S-interpreter. Field 47137 contains the UID-offset value of SDP and Field 47139 contains that of PBP. SIP Field 47141 contains a pointer to Procedure 602's S-interpreter object, and NTP, finally, is a pointer to Procedure 602's Name Table 10350.
- Field 47145 contains the PC for the SIN which is to be executed on return from the mediated invocation to which SS Frame 47003 belongs.

c.c.c. Cross domain SS Frames 47039 (Fig. 471)

Cross-domain SS Frames 47039 differ from Ordinary SS Frames 10510 in two respects: they have an additional component, Cross-domain State 10513, and fields in Cross domain Frame Header 47157 have different meanings from those in Ordinary Frame Header 10514.

Cross-domain State 10513 contains information which KOS Call microcode uses to verify that a return to a Procedure 602 whose DOE differs from that of Procedure 602 whose invocation has ended is returning to the proper domain. Fields of interest in Cross-domain State 10513 include GOTO Tag 47155 used for non-local GOTOs which cross domains, Stack Top Pointer Value 47153, which gives the location of the first free bit in the new domain's MAS Object 46703 and Frame Header Pointer Value 47151, which contains the location of the topmost Mediated Frame Header 46709 in new MAS Object 46703.

There are three fields in Cross-domain Frame Header 47157 which differ from those in Ordinary SS Frame Header 47101. These fields are Flag Field 47107 which in Cross-domain Frame Header 47157 always has the value TRUE, preceding Cross-domain Frame Offset Field 47161, which contains the offset of preceding Cross-domain Frame 47039 in SS Object 10336 and Next Cross domain Frame Offset Field 47159, which contains the location of the next Cross-domain Frame 47039. These last two fields occupy the same locations as Fields 47111 and 47109 respectively in Ordinary SS Frame Header 10514.

As will be noted from the above description of SS Frames 47003. Secure Stack Object 10336 in the present embodiment contains three kinds of information: macrostate cross-domain state and microstate. In other embodiments, the information in SS object 10336 may be stored in separate stack structures, for example, separate microstate and cross-domain stacks, or information presently stored in MAS Objects 46703 may be stored in SS Object 10336, and vice-versa.

4. Portion of Procedure Object 608 Relevant to Call and Return (Fig. 472)

The information which Process 610 requires to construct new frames on its MAS Objects 46703 and SS Object 10336 and to transfer control to invoked Procedure 602 is contained in invoked Procedure 602's Procedure Object 608. Figure 472 is an overview of Procedure Object 608 showing the information used in a Call. Figure 472 expands information contained in Figures 103 and 303; fields that appear in those Figures have the names and numbers used there.

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602 that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 608. Procedure Object 608's gates are contained in External Entry Descriptor Area 10340. There are two kinds of gates: those for Procedures 602 contained in Procedure Object 608, and those for procedures 602 contained in other Procedure Objects 608, but callable via

Procedure Object 608. Gates for Procedures 602 contained in Procedure Object 608 are termed Local Gates 47205. Local Gates 47205 contain Internal Entry Offset (IEO) Field 47207 which contains the offset in Procedure Object 608 of Entry Descriptor 47227 for Procedure 602. If Procedure 602 is not contained in Procedure Object 472 its gate is a Link Gate 47206. Link Gates 47206 contain Binder Area Pointer (BAP) Fields 47208. A BAP Field 47208 contains the locations of an area in Binder Area 30323 which in turn contains a pointer to a Gate in another Procedure Object 608. The pointer in Binder Area 30323 may be either resolved or unresolved. If Procedure 602 is contained in that Procedure Object 608, the Gate is a Local Gate 47205; otherwise, it Is another Link Gate 47206.

Procedure Environment Descriptors (PEDS) 10348 contains PEDs 30303 for Procedures 602 contained in Procedure Object 608. Most of the macrostate information for a Procedure 602 may be found in its PED 30303. PED 30303 has already been described, but for ease of understanding, its contents are reviewed here.

- K Field 30305 contains the size of Procedure 602's Names.
- Largest Name (LN) Field 30307 contains the i

Beginning with Procedure Object Header 10336, this area contains two items of information used in Calls: an Offset in Field 47201 giving the location of Argument Information Array 10352 in Procedure Object 608 and a value in Field 47203 specifying the number of gates in Procedure Object 608. Gates allow the invocation of External Procedures 602, that is, Procedures 602 which may be invoked by Procedures 602 contained in other Procedure Objects 6nter to Static Data Block 46863. Thus, for that Invocation of Procedure 602 on invocation, the SDP ABP is derived via SDPP field 30313.

- PBP Field 30315 is the pointer from which the current PC is calculated. When Procedure 602 is invoked, this value becomes the PBP ABP.
- S-interpreter Environment Prototype Pointer (SEPP) Field 30316 contains the location of SEB Prototype Field 30317. When Procedure 602 is invoked, Field 30316 locates SEB 46864 via AAT 30201 in the same manner as SDPP field 30313 locates the invocation's static data.

A Procedure 602's PED 30303 may be located from its Internal Entry Descriptor 47227. A PED 30303 may be shared by several Procedures 602. Of course in this case, the values contained in shared PED 30303 are the same for all Procedures 602 sharing it. As will be explained in detail later in the present embodiment. if a calling Procedure 602 does not share a PED 30303 with called Procedure 602 the Call must be mediated. A calling Procedure 602 may make a Neighborhood Call only to Procedures 602 with which it shares a PED 30303.

The next portion of Procedure Object 608 which is of interest is Internal Entry Descriptors 10342. Each Procedure 602 contained in Procedure Object 608 has an Entry Descriptor 47227. Entry Descriptor 47227 contains four fields of interest:

- PBP Offset Field 47229 contains the offset from PBP at which the first SIN in Procedure 602's code is located.
- Flags Field 47230 contains flags which are checked when Procedure 602 is invoked. Four flags are of interest:
  - Argument Information Array Present Flag 47235 which is set to TRUE if Procedure 602 has entries in Argument Information Array 10352.
  - SEB Flag 47237 is set to TRUE if SEPP 47225 is non-null, i.e., if Procedure 602 has a SEB 46864 for its S-interpreter.
  - Do Not Check Access Flag 47239 is set to TRUE if KOS Call microcode is not to perform protection checking on the actual arguments used to invoke Procedure 602.
- PED Offset Field 47231 contains the offset of Procedure 602's PED 30303 from the beginning of Procedure Object 608.
- Frame Size Field 47233 contains the initial size of the Local Storage Portion 10420 of MAS Frame 46709 for an invocation of procedure 602.

Other areas of interest for Calls are SEB Prototype Area 47241, Static Data Area Prototype 30317, Binder Area 30323 and Argument Information Array 10352. SEB Prototype type Area 47241 and Static Data Area Prototype 30315 contain information used to create an SEB 46864 and Static Data Block 46863 respectively for Procedure 602. These areas are created on a per-MAS Object 46703 basis. The first time that a Process 610 executes a Procedure 602 in a domain, SEB 46864 and Static Data Block 46863 required for Procedure 602 are created either in MAS Object 46703 belonging to the domain or in another object accessible from MAS Object 46703. SEB 46864 and Static Data Block 46863 then remain as long as MAS Object 46703 exists.

Static Data Prototype 30317 contains two kinds of information: Static Data Links 30319 and Static Data Initialization Information 30321 Static Data Links 30319 contain locations in Binder Area 30323, which in turn contains pointers which may be resolved to yield the locations of data or External Procedures 602. When a Static Data Block 46863 is created for a Procedure 602, the information in Binder Area 30323 is used to create Linkage Pointers 46865. Static Data Initialization Information 30321 contains information required to create and initialize static data in Static Data Storage 46867.

As mentioned in the discussions of Link Gates 47206 and Static Data Links 30319 Binder Area 30323 contains pointers which may be resolved as described in Chapter 3 to yield locations of data and External Procedures 602.

Argument Information Array (AIA) 10352 contains information used by KOS Call microcode to check whether the subject which is invoking Procedure 602 has access to the actual arguments used in the invocation which allows the uses made of the arguments in Procedure 602. This so-called "Trojan horse check" is necessary because a Call may change the domain component of a subject. Thus, a subject which is lacking access of a specific kind to a data item could gain that access by passing the data item as an argument to a Procedure 602 whose DOE gives it access rights that the calling subject itself lacks.

Each Local Gate 47205 in Procedure Object 608 has an element in AIA 10352. Each of these Argument Information Array Elements (AIAEs) 60845 has fields indicating the following:

- The minimum number of arguments required to invoke Procedure 602 to which Local Gate 47205 belongs, in Field 47247.
- The maximum number of arguments which may be used to invoke Procedure 602 in Field 47249.
- The access rights that the invoking subject must have to the actual arguments in order to invoke Procedure 602 in Field 47251.

Field 47251 is itself an array which specifies the kinds of access that the invoking subject must have to the actual arguments it uses to invoke Procedure 602. Each formal argument for Procedure 602 has an Access Mode Array Entry (AMAE) 47255. The order of the AMAEs 47255 corresponds to the order of Procedure 602's formal arguments. The first formal argument has the first AMAE 47255, the second the second, and so forth. An AMAE 47253 is four bits long. There are two forms of AMAE 47253: Primitive Access Form 47255 and Extended Access Form 47257. In the former form, the leftmost bit is set to 0. The three remaining bits specify read, write, and execute access. If a bit is on, the subject performing the invocation must have that kind of primitive access to the object containing the data item used as an actual for the formal argument corresponding to that AMAE 47253. In the Extended Access Form 47257, the leftmost bit is set to 1 and the remaining bits are defined to represent extended access required for Procedure 602. The definition of these bits varies from Procedure 602 to Procedure 602.

5. Execution of Mediated Calls

Having described the portions of MAS Object 46703, SS Object 10336. and Procedure Object 608 which are involved in Calls, the discussion turns to the description of the Mediated Call Operation. First, there is presented an overview of the Mediated Call SIN and then the implementation of Mediated Calls in the present embodiment is discussed, beginning with a simple Mediated Call and continuing with Cross-Procedure Object Calls and Cross Domain Calls. The discussion closes with a description of software-to-microcode Calls.

a.a. Mediated Call SINs

While the exact form of a Mediated Call SIN is S-language specific, all Mediated Call SINs must contain four items of information:

- The SOP for the operation.
- A Name that evaluates to a pointer to the Procedure 602 to be invoked by the SIN.
- A literal (constant) specifying the number of actual arguments used in the invocation.
- A list of Names which evaluate to pointers to the actual arguments used in the invocation.

If Procedure 602 requires no arguments, the literal will be 0 and the list of Names representing the actual arguments will be empty.

In the present embodiment, Mediated Call and Return SINs are used whenever called Procedure 602 has a different PED 30303 from calling Procedure 602. In this case, the Call must save and recalculate macrostate other than FP and PC, and mediation by KOS Call microcode is required. The manner in which KOS Call microcode mediates the Call depends on whether the Call is a simple Mediated Call a Cross-procedure Object Call, or a Cross-Domain Call.

b.b. Simple Mediated Calls (Fig. 270, 468, 469, 470, 471, 472)

When the Mediated Call SIN is executed, S-interpreter microcode first evaluates the Name which represents the location of the called Procedure 602. The Name may evaluate to a pointer to a Gate 47205 or 4707 in another Procedure Object 608 or to a pointer to an Entry Descriptor 47227 in the present Procedure Object 608. When the Name has been evaluated, S-interpreter Call microcode invokes KOS Call microcode, using the evaluated Name as an argument. This microcode first fills in Macrostate Fields 10516, left empty until now, in the current invocation's SS Frame 47003. The microcode obtains the values for these fields from registers in FU 10120 where they are maintained while Virtual Processor 612 of Process 610 which is executing the Mediated Call is bound to JP 10114.

The next step to determine whether the pointer which KOS Call microcode received from S-interpreter Call microcode is a pointer to an External Procedure. To make this determination, KOS Call microcode compares the pointer's AON 41304 with that of Procedure Object 608 for Procedure 602 making the Call. If they are different, the Call is a Cross-Procedure Object Call, described below. In the case of the Simple Mediated Call, the format field indicates that the location is an Entry Descriptor 47227. KOS Call microcode continues by saving the location of Entry Descriptor 47227 and creating a new Mediated Frame 46947 on current MAS Object 46703 and a new Ordinary SS Frame 10510 on SS Object 10336 for called Procedure 602. As KOS Call microcode does so, it sets Fields 46917 and 46919 in Mediated Frame Header 10414 and

Fields 47109 and 47111 in Ordinary SS Frame Header 10514 to the values required by the addition of frames to MAS Object 46703 and SS Object 10336.

New Mediated Frame 46947 is now ready for Linkage Pointers 10416 to the actual arguments used in the Call, so KOS Call microcode returns to S-interpreter Call microcode, which parses the SIN to obtain the literal specifying the number of arguments and saves the literal value. S-interpreter Call microcode then parses each argument Name, evaluates it, and places the resulting value in Linkage Pointers Section 10416. When Linkage Pointers Section 10416 is complete, S-Interpreter Call Microcode calculates the new location of FP from the location of the top of Linkage Pointers Section 10416 and places a pointer for the location in the FU 10120 register reserved for FP. At this time, S-interpreter Call microcode also places the new location of the top of the stack in Stack Top Offset Field 46807.

S-interpreter Call microcode then invokes KOS Call microcode to place the value of the literal specifying the number of arguments in MAS Frame Field 46929, to calculate the new value of FHP 46702 and place it in the FU 10120 register reserved for that value, and finally to obtain the state necessary to execute called Procedure 602 from called Procedure 602's Entry Descriptor 47227 and PED 30303. As previously stated, S-interpreter Call microcode saved the location of Entry Descriptor 47227. Using this location, KOS Call Microcode obtains the size of the storage required for local data from Field 47233 and adds that amount of storage to the new MAS Frame 46709. Then KOS Call Microcode uses Field 47231 to locate PED 30303 for Procedure 602. PED 30303 contains the remainder of the necessary information about Procedure 602 and KOS Call microcode copies the location of PED 30303 into PED Pointer Field 46933 and then copies the values of K Field 30305. Last Name Field 30307, NTP Field 30311 and PBP Field 30315 into the relevant registers in FU 10120. KOS Call microcode next translates the pointer in SIP Field 30309 into a dialect number as explained in Chapter 3, and places it in register RDIAL 24212 of FU 10220 and thereupon derives SDP by resolving the pointer in SDPP Field 30313 and a pointer to SEB 46864 by resolving the pointer in SEPP Field 30316. Having performed these operations, KOS Call microcode returns to S-interpreter Call microcode, which finishes the Call by obtaining a new PC, that is, resetting registers in I-stream Reader 27001 in FU 10120 so that the next SIN to be fetched will be the first SIN of called procedure 602 S-interpreter Call microcode obtains the information required to change PC from Field 47229 in Entry Descriptor 47227 which contains the offset of the first SIN of called Procedure 602 from PBP.

In the present embodiment, some FU 10120 state produced by the Mediated Call SIN is retained on SS 504 throughout the duration of Procedure 602's invocation. The saved state allows Process 610 to reattempt the Mediated Call if the Call fails before the called Procedure 602 begins executing. When a Mediated Return SIN is executed, it resumes execution on the retained state from the CALL SINT. The Mediated Return is much simpler than the Call. Since all of the information required to resume execution of the invocation which perfomed the Call is contained in Macrostate 10516 in the calling invocation's SS Frame 47003, Return need only pop the called invocation's frames from current MAS Object 46703 and SS Object 10336, copy Macrostate 10516 47123 from the calling invocation's SS Frame 47003 into the proper FU 10120 registers, translate SIP Value 47141 into a dialect number, and resume executing the calling invocation. The pop operation involves nothing more than updating those pointers in MAS Object 46703 and SS Object 10336 which pointed to locations in the old topmost frame so that they now point to equivalent locations in the new topmost frame.

c.c. Invocations of Procedures 602 Requiring SEBs 46864 (Fig. 270, 468, 469, 470, 471, 472)

If a Procedure 602 requires a SEB 46864, this fact is indicated by Flag Field 47237 in Procedure 602's Entry Descriptor 47227. PED 30303 for such a Procedure 602 contains SEPP Field 47225, whose value is a non-resolvable pointer. The manner in which a SEB 46864 is created for Procedure 602 and SEPP field 47225 is translated into SEP, a pointer which contains the location of SEB 46864 and is saved as part of the invocation's macrostate on SS 10336, is similar to the manner in which a Static Data Block 46863 is created and the non-resolvable pointer contained in SDPP field 47225 is translated into SDP. The first time that a Procedure 602 requiring a SEB 46864 is invoked on a MAS Object 46703, a SEB 46864 is created for the Procedure 602 and an AATE 46857 is created which associates the nonresolvable pointer in SEPP field 47225 and the location of SEB 46864. That location is the value of SEP when the procedure is executing on MAS object 46703. On subsequent invocations of Procedure 602. AATE 46857 serves to translate the value in SEPP field 47225 into SEP.

d.d. Cross-Procedure Object Calls (Fig. 270, 468, 469, 470, 471, 472)

A Mediated Call which invokes an External Procedure 602 is called a Cross-Procedure Object Call. As previously mentioned. KOS Call microcode assumes that any time the Name representing the called Procedure 602 in a Mediated Call SIN resolves to the location of a Gate that the Call is to an External Procedure 602. As long as newly-called External procedure 602 has the same DOE as calling Procedure 602. Cross-Procedure Object Calls differ from the Simple Mediated Call only in the manner in which called Procedure 602's Entry Descriptor 47227 is located. Once KOS Call microcode has determined as described above that a Mediated Call is a Cross-Procedure Object Call it must next determine whether it is a Cross-Domain Call. To do so, KOS Call microcode compares the DOE Attribute of called Procedure 602's Procedure Object 608 with the domain component of the current subject. KOS Call microcode uses Procedure Object 608's AON 41304 to obtain Procedure Object 608's DOE from Field 41521 of its AOTE

41306 and it uses the ASN for the current subject, stored in an FU 10120 register, to obtain the current subject's domain component from AST 10914. If the DOE and the current subject's domain component differ, the Call is a Cross-domain Call, described below; otherwise, the Call locates the Gate 47205 or 47206 specified by the evaluated Name for called Procedure 602 in its Procedure Object 608. If the Gate is a Local Gate 47205, the Call uses Entry Descriptor Offset Field 47207 to locate Entry Descriptor 47227 belonging to Called Procedure 602 and then proceeds as described in the discussion of a Simple Mediated Call.

If the Gate is a Link Gate 47206, KOS Call microcode obtains the pointer corresponding to Link Gate 47206 from Binder Area 47245 and resolves it to obtain a pointer to another Gate 47205 or 47206, which KOS Call microcode uses to repeat the External Procedure 602 call described above. The repetitions continue until the newly-located gate is a Local Gate 47205, whereupon Call proceeds as described for Simple Mediated Calls.

e.e. Cross-domain Calls (Fig. 270, 408, 418, 468, 469, 470, 471, 472)

If a called Procedure 602's Procedure Object 608 has a DOE attribute differing from that of calling Procedure 602's Procedure Object 608, the Call is a Cross-domain Call. The means by which KOS Call microcode determines that a Mediated Call is a Cross-Domain Call have previously been described; If the Call is a Cross-Domain Call, KOS Call microcode must inactivate MAS Object 46703 for the domain from which the Call is made, perform trojan horse argument checks, switch subjects, place a Cross-domain Frame 47039 on SS object 10336, and locate and activate MAS Object 46703 for the new domain before it can make a Mediated Frame 46947 on new MAS Object 46703 and continue as described in the discussion of a Simple Mediated Call.

Cross-domain Call microcode first inactivates the current MAS Object 46703 by setting Domain Active Flag 46804 to FALSE. The next step is the trojan horse argument checks. In order to perform trojan horse argument checks, Cross-domain Call must have pointers to the actual arguments used in the cross-domain invocation. Consequently, Cross-domain Call first continues like a non-cross-domain Call: it creates a Mediated Frame Header 10414 on old MAS Object 46703 and returns to S-interpreter microcode, which evaluates the Names of the actual arguments, and places the pointers in Linkage Pointers 10416 above Mediated Frame Header 10414. However, the macrostate for the invocation performing the call was placed on SS Object 10336 before Mediated Frame Header 10414 and Linkage Pointers 10416 were placed on old MAS Object 46703. Consequently, when calling Procedure 602 resumes execution after a Return, it will resume on MAS Frame 46709 preceding the one built by Cross-domain Call microcode.

Once the pointers to the actual arguments are available, Cross-domain Call Microcode performs the trojan horse check. As described in the discussion of Procedure Object 608 and illustrated in Figure 472, the information required to perform the check is contained in AIA 10352. Each Local Gate 47205 in Procedure Object 608 has an AIAE 47245, each formal argument in Local Gate 47205's procedure has an entry in AIAE 47245's AMA 47251, and the formal argument's AMAE 47253 indicates what kind of access to the formal argument's actual argument is required in called Procedure 602.

Field AIA OFF 47201 contains the location of AIA 10352 in Procedure Object 608, and using this information and Local Gate 47205's offset in Procedure Object 608, Cross-domain Call microcode locates AIAE 47245 for Local Gate 47205. The first two fields in AIAE 47245 contain the minimum number of arguments in the invocation and the maximum number of arguments. Cross-domain Call microcode checks whether the number of actual arguments falls between these values. If it does, Cross-domain Call microcode begins checking the access allowed individual arguments. For each argument pointer. Cross-domain Call microcode calls LAR microcode to obtain the current AON 41304 for the pointer's UID and uses AON 41304 and the ASN for Process 610's current subject (i.e., the caller's subject) to locate an entry in either APAM 10918 or ANPAT 10920, depending on whether the argument's AIAE specifies primitive access (47255) or extended access (47257) respectively. If the information from APAM 10918 or ANPAT 10920 confirms that Process 610's current subject has the right to access the argument in the manner required in called Procedure 602, the Trojan Horse microcode goes on to the next argument. If the current subject has the required access to all arguments, the trojan horse check succeeds and the Cross-domain Call continues, Otherwise, it fails and Cross-domain Call performs a microcode-to-software Call as explained below.

Next, Cross-domain Call microcode places Cross domain State 10513 on SS Object 10336. As explained in the discussion of SS object 10336, Cross-domain State 10513 contains the information required to return to the caller's frame on former MAS Object 46703. Having done this, Cross-domain Call microcode changes subjects. Using the current subject's ASN, Cross-Domain Call microcode obtains the current subject from AST 10914 replaces the subject's domain component with DOE Attribute 41225 for called Procedure 602's Procedure Object 608 and uses AST 10914 to translate the new subject thus obtained into a new ASN. That ASN then is placed in the appropriate FU 10120 register.

After the subject has been changed, Cross-domain Call microcode uses Domain Table 41801 to translate the DOE of called Procedure 602 into a domain number. Cross-domain Call microcode then uses the domain number as an index into Array of MAS AONs 46211 in VPSB 614 for Virtual Processor 612 belonging to Process 610 making the cross-domain call. The entry corresponding to the domain number contains AON 41304 of MAS Object 46703 for that domain.

Having located the proper MAS Object 46703, Cross-domain Call microcode uses STO field 46807 in MAS Header 10410 belonging to the new domains MAS Object 46703 to locate the top of the last MAS

Frame 46709. It then saves the value of FHP 46702 used in the preceding invocation in a FU 10120 register, adds a Mediated Frame Header 10414 to the top of MAS Object 46703, and calculates a new FHP 46702 which points to new Mediated Frame Header 10414. KOS Cross-Domain Call microcode then places the old value of FHP 46702 in FHP Value Field 47151 of SS Object 10336 and the old value of STO 46704 (pointing to the top of the last complete MAS Frame 46709 on previous MAS Object 46703) in Field 47153 of Cross-Domain State 10513 and fills in Mediated Frame Header 10414 fields as follows: Result of Cross-domain Call Field 46903 is set to TRUE. Previous Frame Offset Field 46917 is set to 0, and Dynamic Back Pointer Field 46931 is set to the saved value of FHP 46702. Dynamic Back Pointer Field 46931 thus points to the header of the topmost Mediated Frame 46947 on the previous MAS Object 46703. The values of the remaining fields are copied from Mediated Frame Header 10414 which Cross-Domain Call created on previous MAS Object 46703.

Cross-domain Call microcode next copies the argument pointers for the formal arguments from the top of previous MAS Object 46703 to new Mediated Frame 46947 and calculates FP. Cross-domain Call Microcode finishes by returning to S-interpreter Call microcode, which completes the Call as described for Simple Mediated Calls.

Except for the work involved in transferring to a new MAS Object 46703, Cross-domain Return is like other Returns from Mediated Calls. Old FHP 46701 from Field 47151 of Cross-Domain State 10513 and old STO 46704, from Field 47153 of Cross-domain State are placed in FU 10120 registers. Then the frames belonging to the invocation that is ending are popped off of SS Object 10336 and off of MAS Object 46703 belonging to the domain of called Procedure 602 and MAS Object 46703 is inactivated by setting Domain-Active Flag 46804 to FALSE. Then KOS Cross-domain Return microcode uses old FHP 46701 and old STO 46704 to locate MAS Object 46703 being returned to and the topmost Mediated Frame 46947 on that MAS Object 46703. MAS Object 46703 being returned to is activated, and finally, the contents of Macrostate 10516 belonging to the invocation being returned to are placed in the appropriate registers of FU 10120 and execution of the invocation resumes.

f.f. Failed Cross-Doamin Calls (Fig. 270, 468, 469, 470, 471, 472)

A Cross-Domain Call as described above may fail at several points between the time that the calling invocation begins the call and called Procedure 602 begins executing. On failure, Cross-Domain Call microcode performs a microcode-to-software Call. KOS Procedures 602 invoked by this Call may remedy the reason for the Cross Domain Call's failure and reattempt the Cross-domain Call. This is possible because the implementation of Cross Domain Call in CS 10110 saves sufficient FU 10120 state to allow Process 610 executing the Cross-Domain Call to return to the invocation and the Mediated Call SIN from which the Cross-Domain Call began. On failure, the invocation's MAS Frame 46709 may be located from the values of STO Field 47153 and FHP Field 47151 in Cross-Domain State 10513, and the Mediated Call SIN may be located by using information saved in FU 10120 state.

6. Neighborhood Calls (Fig. 468, 479, 472)

As previously mentioned, Procedures 602 called via Neighborhood Calls must have the same PED 30303 as calling Procedure 602. The only macrostate values which are not part of PED 30303 are PC and FP; consequently Neighborhood Call need only save PC and FP of the invocation performing the call and calculate these values for the new invocation. In addition, Neighborhood Call saves STO 46704 in order to make it easier to locate the top of the previous invocation's Neighborhood Frame 46947. Neighborhood Return simply restores the saved values. Since the macrostate values copied from or obtained via PED 30303 do not change during the sequence of invocations, and therefore need not be saved on SS Object 10336. Neighborhood Calls do not have SS Frames 47003.

The invention may be embodied in yet other specific forms without departing from the spirit or essential characteristics thereof. Thus, the present embodiments are to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than by the foregoing description.

Claims

1. A digital computer system (CS 101) including processor means (JP 114) for performing operations upon operands, memory means (MEM 112) for storing said operands and procedures, said procedures including instructions for controlling said operations and names referring to certain of said operands to be operated upon, ALU means (2034, 2074) for performing said operations, bus means (MOD 140, JPB 142) for conducting said instructions, names and operands between said memory means and said processor means, and I/O means (IOS 116) for conducting at least said operands between said memory means and devices external to said digital computer system, characterised in that said processor means (JP 114) comprises means for addressing said operands, including name table means (10350) for storing name table entries, each name table entry corresponding to one of said names included in each one of said procedures and each name table entry comprising first data from which may be determined an address of a location in said memory means of the operand referred to by one of said names and second data identifying a format of that operand, and translation means (NAME TRANS UNIT 27015) connected to said

bus means and responsive to said name table entries for providing outputs to said memory means representing said addresses, and further characterised in that said instructions are intermediate level S-language instructions from a plurality of sets of such instructions, each set corresponding to a particular higher level user programming language, and further characterised by receiving means (INSTB 20262) connected to said bus means for receiving said instructions from said memory means, and microcode control means (10240, 27003, 27013) connected between said receiving means and said ALU means for providing sequences of microinstructions for controlling said ALU means, said sequences being selected from a plurality of sequences of microinstructions corresponding to said S-language instructions respectively.

2. A digital computer system according to claim 1, characterised in that the S-language instructions have a uniform, fixed format.

3. A digital computer system according to claim 1 or 2, characterised in that the names are of uniform length and format.

4. A digital computer system according to any of claims 1 to 3, characterised in that each procedure further includes a name table pointer (NTP 30311) representing a base location in said memory means (MEM 112), and said first data of each name table entry contains information from which may be determined an address offset of a memory location relative to the base location, and in that said translation means (NAME TRANS UNIT 27015) further comprises base register means (NCR, MCR 10366) connected to said bus means for receiving and storing said name table pointer of the procedure currently controlling the operations performed by said ALU means.

5. A digital computer system according to any of claims 1 to 4, characterised by name cache means (10226) connected to outputs of said translation means (NAME TRANS UNIT 27015) and having outputs to said memory means (MEM 112) for storing said addresses, and further connected to said receiving means (INSTB 20262) and responsive to said names to provide name cache outputs to said memory means representing said addresses of certain operands for which said name cache means has stored said addresses.

6. A digital computer system according to any of claims 1 to 5, characterised in that each of said S-Language instructions is a member of an S-Language dialect of a plurality of S-Language dialects, and in that said receiving means (INSTB 20262) further comprises dialect code means (RDIAL 24212) for storing a dialect code specifying the dialect of which the received S-Language instructions are members, and in that said sequences of microinstructions include a set of sequences of microinstructions, corresponding to each said S-Language dialect, each set of sequences of microinstructions including at least one sequence of microinstructions corresponding to each S-Language instruction in a corresponding S-Language dialect, and in that said microcode control means (10240, 27003, 27013) is responsive to the dialect code and to each received S-Language instruction to provide to said ALU means (2034, 2074) a sequence of microinstructions corresponding to that S-Language instruction.

7. A digital computer system according to claim 1 or 2, characterised in that each procedure includes a dialect code denoting an S-Language dialect of which the S-Language instructions of the procedure are members, and in that said microcode control means (10240, 27003, 27013) further comprises control store means (SITT 11012) for storing said sequences of microinstructions for controlling said ALU means (2034, 2074), and dispatch table means (SIDT 11010) for storing addresses corresponding to locations in said control store means of each sequence of microinstructions, and in that said dispatch table means is responsive to said dialect code and to each instruction to provide to said control store means each address corresponding to said at least one microinstruction sequence corresponding to each said instruction, and said control store means is responsive to each address to provide to said ALU means said sequence of microinstructions corresponding to each instruction.

8. A digital computer system according to claim 1, 6 or 7, characterised in that said microcode control means (10240, 27003, 27013) comprises writable control store means (11012) connected to said bus means for storing said sequences of microinstructions, and control store addressing means (SITTNAS 20286) responsive to each S-Language instruction and to operation of said processor means for generating control store read addresses and write addresses (CSADR 20204), and in that said writable control store means is responsive to said read addresses to provide said sequences of microinstructions to said ALU means (2034, 2074) and is responsive to said write addresses to store said sequences of microinstructions.

9. A digital computer system according to claim 7, characterised in that said control store means (SITT 11012) comprises writable control store means connected to said bus means for storing said sequences of microinstructions, and in that said dispatch table means comprises write address means responsive to operation of said processor means for generating write addresses, and in that said writable control store means is responsive to said write addresses for storing said sequences of microinstructions.

Patentansprüche

1. Digitales Datenverarbeitungssystem (CS 101), enthaltend: Prozessormittel (MEM 114) zur Durchführung von Operationen an Operanden, Speichermittel (MEM 112) zum Speichern der Operanden und von Prozeduren, die Befehle zur Steuerung der Operationen und Namen enthalten, die auf gewisse der Operanden Bezug nehmen, an denen Operationen durchgeführt werden sollen, eein Rechenwerk (2034,

2074) zur Durchführung der Operationen, Bus-Mittel (MOD 140, JPE 118) für den Verkehr der Befehle, Namen und Operanden zwischen den Speichermitteln und den Prozessormitteln, und Eingabe/Ausgabe-Mittel (IOS 116) für den Verkehr wenigstens der Operanden zwischen den Speichermitteln und Geräten außerhalb des digitalen Datenverarbeitungssystems, gekennzeichnet durch Prozessormittel (JP 114), die Mittel zur Adressierung der Operanden einschließlich Namenstabellenmittel (10350) zur Speicherung von Namenstabellen-Einsprungpunkten enthalten, wobei jeder Namenstabellen-Einsprungpunkt einem der Namen entspricht, die in jeder der Prozeduren enthalten sind, und erste Daten, aus denen eine Adresse eines Platzes derjenigen Operanden in den Speichermitteln bestimmt werden kann, auf die durch einen der Namen Bezug genommen wird, und zweite Daten enthalten die ein Format dieses Operanden identifizieren, und durch Übersetzungsmittel (NAME TRANS UNIT 27015), die mit den Bus-Mitteln verbunden sind und auf die Namenstabellen-Einsprungpunkte unter Bereitstellung von diese Adressen repräsentierenden Ausgaben für die Speichermittel ansprechen, ferner dadurch gekennzeichnet, daß die Befehle mittlere S-Sprache-Befehle von einer Vielzahl von Sätzen solcher Befehle sind, von denen jeder Satz einer besonderen höheren Benutzerprogrammiersprache entspricht, und ferner gekennzeichnet durch ein mit den Bus-Mitteln verbundenes Empfangsmittel (INSTB 20262) zum Empfang der Befehle von den Speichermitteln, und durch mit dem Empfangsmittel und dem Rechenwerk verbundene Mikrocode-Steuermittel (10240, 27003, 27013) zur Bereitstellung von Mikrobefehlssequenzen zur Steuerung des Rechenwerks, wobei diese Sequenzen aus einer Vielzahl von Mikrobefehlssequenzen ausgewählt sind, die den jeweiligen S-Sprache-Befehlen entsprechen.

2. Digitales Datenverarbeitungssystem nach Anspruch 1, dadurch gekennzeichnet, daß die S-Sprache-Befehle ein gleichförmiges, festes Format haben.

3. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß die Namen eine gleichförmige Länge und ein gleichförmiges Format haben.

4. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, daß jede Prozedur weiter einen Namenstabellenzeiger (NTP 30311) enthält, der einen Basisplatz in den Speichermitteln (MEM 112) repräsentiert, daß die ersten Daten jedes Namenstabellen-Einsprungpunktes Informationen enthalten, aus denen die Adresse eines vom Basisspeicherplatz versetzten Speicherplatzes bestimmt werden können, und daß die Übersetzungsmittel (NAME TRANS UNIT 27015) weiter Basisregistermittel (NCR, MCR 10366) enthalten, die mit den Bus-Mitteln verbunden sind, um den Namenstabellenzeiger derjenigen Prozedur zu empfangen und zu speichern, die gerade die vom Rechenwerk durchgeführten Operationen steuert.

5. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 4, gekennzeichnet durch Namens-Cache-Speichermittel (10226), die mit den Ausgängen der Übersetzungsmittel (NAME TRANS UNIT 27015) verbunden sind und zu den Speichermitteln (MEM 112) führend Ausgänge zum Speichern der Adressen haben, und die weiter mit dem Empfangsmittel (INSTB 20262) verbunden sind und auf die Namen unter Bereitstellung von Namens-Cache-Ausgaben für die Speichermittel ansprechen, die die Adressen von gewissen Operanden repräsentieren, für die die Namens-Cache-Speichermittel die Adressen gespeichert haben.

6. Digitales Datenverarbeitungssystem nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, daß jeder der S-Sprache-Befehle ein Mitglied eines S-Sprache-Dialekts einer Vielzahl von S-Sprache-Dialekten ist, daß das Empfangsmittel (INSTB 20262) weiter ein Dialekt-Code-Mittel (RDIAL 24212) zur Speicherung eines Dialekt-Codes enthält, der den Dialekt bestimmt, von dem die empfangenen S-Sprache-Befehle Mitglieder sind, daß die Mikrobefehlssequenzen einen Satz von Mikrobefehlssequenzen entsprechend jedem S-Sprache-Dialekt enthalten, wobei jede Mikrobefehlssequenz wenigstens eine jedem S-Sprache-Befehl in einem entsprechenden S-Sprache-Dialekt entsprechenden Mikrobefehlssequenz enthält, und daß die Mikrocode-Steuermittel (10240, 27003, 27013) auf den Dialekt-Code und jeden empfangenen S-Sprache-Befehl unter Bereitstellung einer diesem S-Sprache-Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

7. Digitales Datenverarbeitungssystem nach Anspruch 1 oder 2, dadurch gekennzeichnet, daß jede Prozedur einen Dialektcode enthält, der einen S-Sprache-Dialekt bezeichnet, von dem die S-Sprache Befehle der Prozedur Mitglieder sind, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ferner Steuerspeichermittel (SITT 11012) zur Speicherung der Mikrobefehlssequenzen für die Steuerung des Rechenwerks (2034, 2074) und Verteilertabellenmittel (SIDT 11010) zur Speicherung von Adressen enthalten, die Plätzen jeder Mikrobefehlssequenz in den Steuerspeichermitteln entsprechen, und daß die Verteilertabellenmittel auf den Dialektcode und jeden Befehl unter Bereitstellung jeder Adresse, die der wenigstens einen, zu jedem Befehl gehörenden Mikrobefehlssequenz entspricht, für die Steuerspeichermittel ansprechen, während die Steuerspeichermittel auf jede Adresse unter Bereitstellung der jedem Befehl entsprechenden Mikrobefehlssequenz für das Rechenwerk ansprechen.

8. Digitales Datenverarbeitungssystem nach Anspruch 1, 6 oder 7, dadurch gekennzeichnet, daß die Mikrocode-Steuermittel (10240, 27003, 27013) ein mit den Bus-Mitteln verbundenes Schreibsteuerspeichermittel (11012) zur Speicherung der Mikrobefehlssequenzen und Steuerspeicheradressiermittel (SITTNAS 20286) enthalten, die auf jeden S-Sprache-Befehl und auf Operationen des Prozessormittels unter Erzeugung von Steuerspeicherlese- und -schreibadressen (CSADR 20204) ansprechen, und daß die Schreibsteuerspeichermittel auf die Leseadressen unter Bereitstellung der Mikrobefehlssequenzen für das Rechenwerk und auf die Schreibadressen unter Speicherung dieser Mikrobefehlssequenzen ansprechen.

9. Digitales Datenverarbeitungssystem nach Anspruch 7, dadurch gekennzeichnet, daß die Steuerspeichermittel (SITT 11012) mit den Bus-Mitteln verbundene Schreibsteuerspeichermittel zur Speicherung der Mikrobefehlssequenzen enthalten, daß die Verteilertabellenmittel Schreibadressenmittel enthalten, die auf Operationen des Prozessormittels unter Erzeugung von Schreibadressen ansprechen, und daß die Schreibsteuerspeichermittel auf die Schreibadressen unter Speicherung der Mikrobefehlssequenzen ansprechen.

Revendications

1. Un système d'ordinateur numérique (CS 101), comprenant un processeur (JP 114) pour effectuer des opérations sur des opérandes, une mémoire (MEM 112) pour mémoriser lesdits opérandes et des procédures, lesdites procédures contenant des instructions pour commander lesdites opérations et des désignations se rapportant à certains desdits opérandes pour les traiter, une unité arithmétique et logique ALU (2034, 2074) pour effectuer lesdites opérations, des bus (MOD 140, JPB 142) pour transmettre lesdites instructions, lesdites désignations et lesdits opérandes entre ladite mémoire et ledit processeur, et des moyens d'entrée/sortie I/O (IOS 116) pour transmettre au moins lesdits opérandes entre ladite mémoire et des dispositifs extérieurs audit système d'ordinateur numérique, caractérisé en ce que ledit processeur (JP 114) comprend des moyens pour l'adressage desdits opérandes, comportant une table de désignations (10350) pour mémoriser des entrées de table de désignations, chaque entrée de table de désignations correspondant à une desdites désignations incluses dans chacune desdites procédures et chaque entrée de table de désignations comprenant une première donnée à partir de laquelle peut être déterminée une adresse d'un emplacement de ladite mémoire contenant l'opérande auquel se reflète l'une desdites désignations et une seconde donnée identifiant un format de cet opérande, et des moyens de transcodage (NAME TRANS UNIT 27015) reliés auxdits bus et réagissant auxdites entrées de tables de désignations de façon à transmettre à ladite mémoire des signaux de sortie représentant lesdites adresses, et en outre caractérisé en ce que lesdites instructions sont des instructions en langage-S de niveau intermédiaire provenant d'une pluralité d'ensembles de telles instructions, chaque ensemble correspondant à un langage de programmation par utilisateur de niveau supérieur particulier, et en outre caractérisé en ce que des moyens de réception (INSTB 20262) sont reliés auxdits bus pour recevoir lesdites instructions à partir de ladite mémoire, et des moyens de commande de microcode (10240, 27003, 27013) connectés entre lesdits moyens de réception et ladite ALU pour fournir des séquences de microinstructions servant à commander ladite ALU, les dites séquences étant sélectionnées parmi une pluralité de séquences de micro-instructions correspondant respectivement auxdites instructions en langage-S.

2. Un système d'ordinateur numérique selon la revendication 1, caractérisé en ce que les instructions en langage-S ont un format fixe et uniforme.

3. Un système d'ordinateur numérique selon une des revendications 1 ou 2, caractérisé en ce que les désignations ont une longueur et un format uniformes.

4. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 3, caractérisé en ce que chaque procédure comprend en outre un pointeur de table de désignations (NTP 30311) représentant un emplacement de base dans ladite mémoire (MEM 112) et ladite première donnée de chaque entrée de la table de désignations contient une information à partir de laquelle peut être déterminé un décalage d'adresse d'un emplacement de mémoire par rapport à l'emplacement de base, et en ce que lesdits moyens de transcodage (NAME TRANS UNIT 27015) comprennent en outre un moyen formant registre de base (NCR, MCR 10366), qui est relié auxdits bus de façon à recevoir et mémoriser ledit pointeur de table de désignations dans la procédure qui est en train de commander les opérations effectuées par ladite ALU.

5. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 4, caractérisé par un moyen formant antémémoire de désignations (10226), relié aux sorties desdits moyens de transcodage (NAME TRANS UNIT 27015) et comportant des sorties reliées à ladite mémoire (MEM 112) pour mémoriser lesdites adresses, et en outre relié auxdits moyens de réception (INSTB 20262) et réagissant auxdites désignations pour fournir à ladite mémoire des sorties de l'antémémoire de désignations représentant lesdites adresses de certains opérandes pour lesquels ladite antémémoire de désignations a mémorisé lesdites adresses.

6. Un système d'ordinateur numérique selon une quelconque des revendications 1 à 5, caractérisé en ce que chacune desdites instructions en langage-S est un élément d'un dialecte en langage-S faisant partie d'une pluralité de dialectes en langage-S et en ce que lesdits moyens de réception (INSTB 20262) comprennent en outre un moyen de codage de dialecte (RDIAL 24212) pour mémoriser un code de dialecte spécifiant le dialecte dont les instructions en langage-s reçues sont des éléments, et en ce que lesdites séquences de micro-instructions contiennent un ensemble de séquences de micro-instructions correspondant à chacun desdits dialectes en langage-S, chaque ensemble de séquences de micro-instructions comprenant au moins une séquence de micro-instructions correspondant à chaque instruction en langage-S dans un dialecte en langage-S correspondant, et en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) réagissent audit code de dialecte et à chaque instruction en langage-S reçue pour fournir à ladite ALU (2034, 2074) une séquence de micro-instructions correspondant à cette instruction en langage-S.

7. Un système d'ordinateur numérique selon une des revendications 1 et 2, caractérisé en ce que chaque procédure comprend un code de dialecte définissant un dialecte en langage-S dont les instructions en langage-S de la procédure sont des éléments et en ce que lesdits moyens de commande de microcode (1020, 27003, 27013 comprennent en outre une mémoire de commande (SITT 11012) pour mémoriser lesdites séquences de micro-instructions pour commander ladite ALU (2034, 2074), et un moyen à table de distribution (SIDT 11010) pour mémoriser des adresses correspondant aux emplacements de chaque séquence de micro-instructions dans ladite mémoire de commande, et en ce que ledit moyen à table de distribution réagit audit code de dialecte et à chaque instruction pour fournir à ladite mémoire de commande chaque adresse correspondant à ladite séquence de micro-instructions au moins prévue correspondant à chacune desdites instructions, et ladite mémoire de commande réagit à chaque adresse pour fournir à ladite ALU ladite séquence de micro-instructions correspondant à chaque instruction.

8. Un système à ordinateur numérique selon une des revendications 1, 6 et 7, caractérisé en ce que lesdits moyens de commande de microcode (10240, 27003, 27013) comprennent une mémoire de commande inscriptible (11012) reliée auxdits bus pour mémoriser lesdites séquences de micro-instructions et un moyen d'adressage de mémoire de commande (SITTNAS 20286) réagissant à chaque instruction en langage-S et au fonctionnement dudit processeur pour produire des adresses de lecture et des adresses d'écriture dans la mémoire de commande (CSADR 20204) et en ce que ladite mémoire de commande inscriptible réagit auxdites adresses de lecture pour fournir lesdites séquences de micro-instructions à ladite ALU (2034, 2074) et réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

9. Un système d'ordinateur numérique selon la revendication 7, caractérisé en ce que ladite mémoire de commande (SITT 11012) comprend une mémoire de commande inscriptible qui est reliée auxdits bus de mémoriser lesdites séquences de micro-instructions et en ce que ledit moyen à table de distribution comprend un moyen d'adressage d'écriture réagissant au fonctionnement dudit processeur pour produire des adresses d'écriture, et en ce que la mémoire de commande inscriptible réagit auxdites adresses d'écriture pour mémoriser lesdites séquences de micro-instructions.

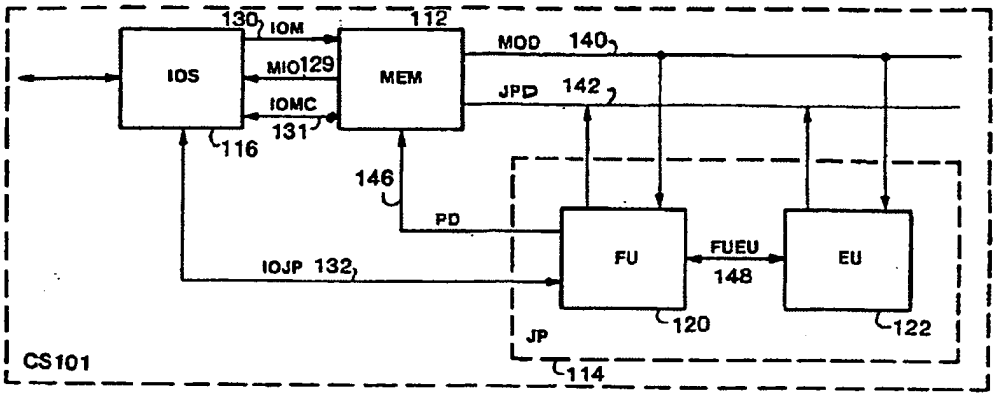


FIG 1

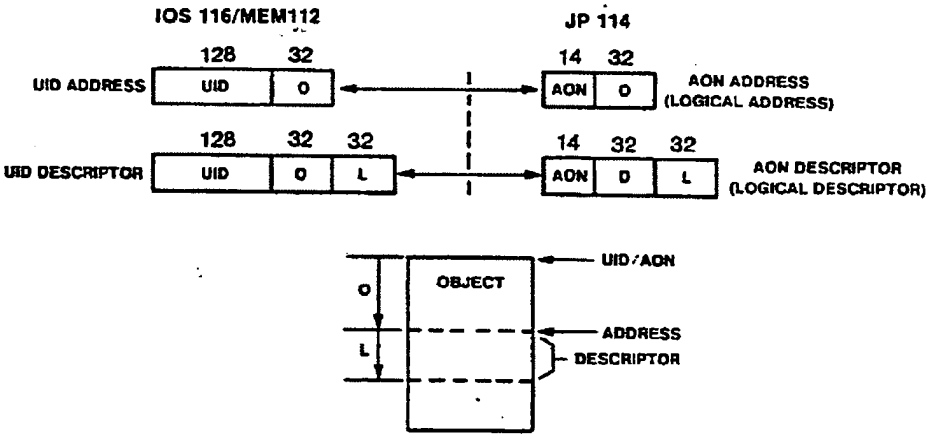


FIG 2

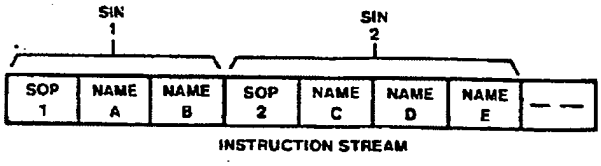


FIG 3

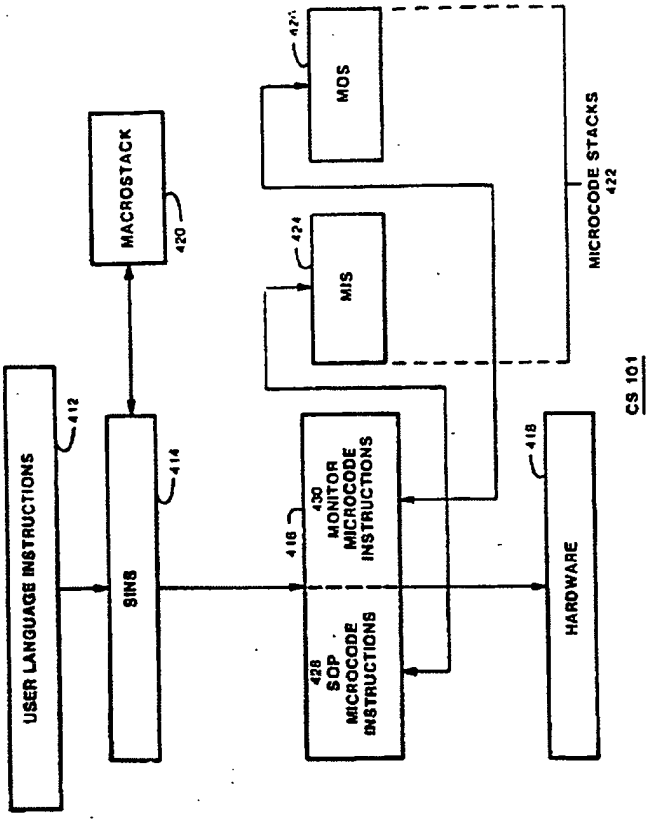


FIG 4A

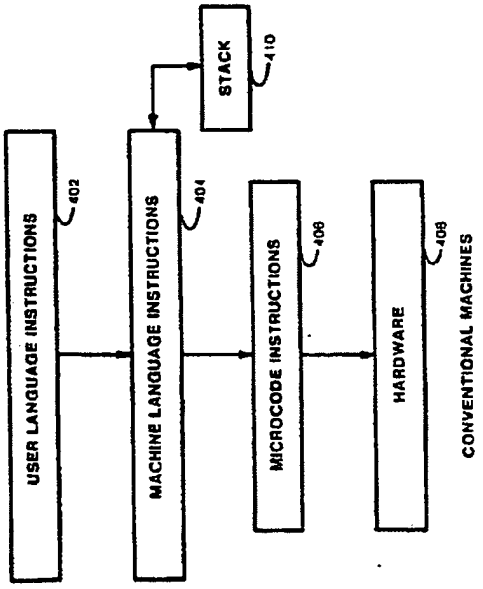


FIG 4

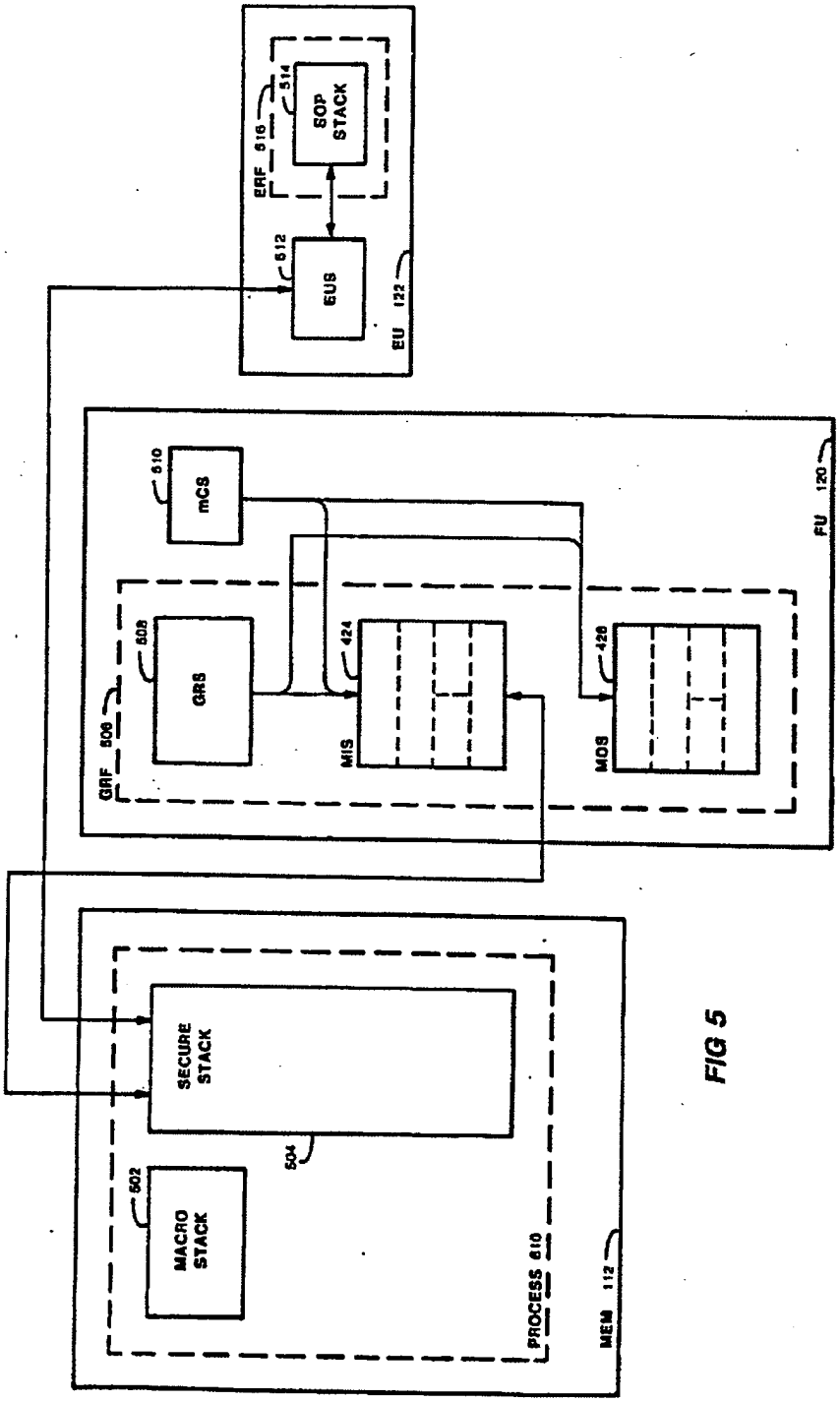


FIG 5

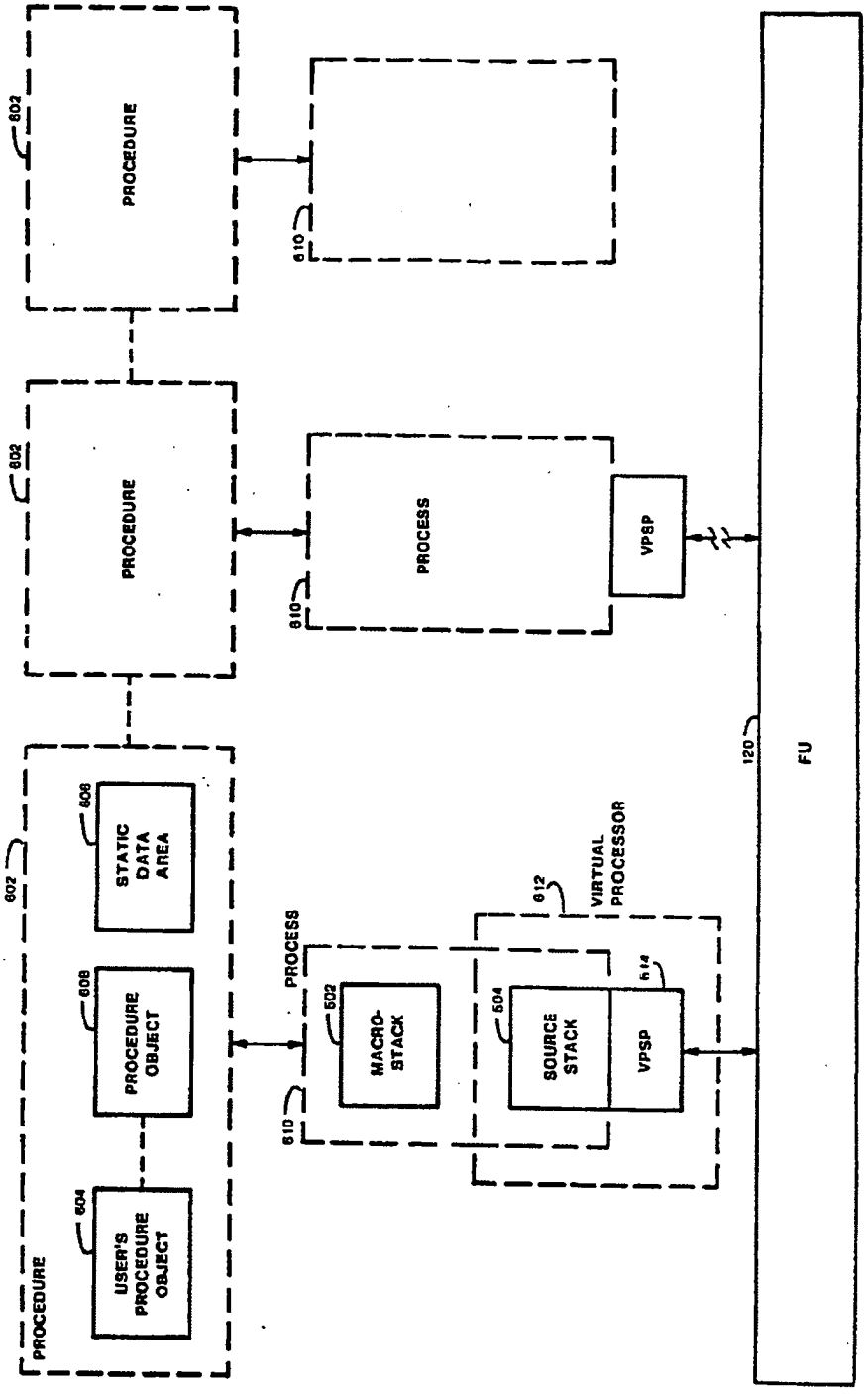


FIG 6

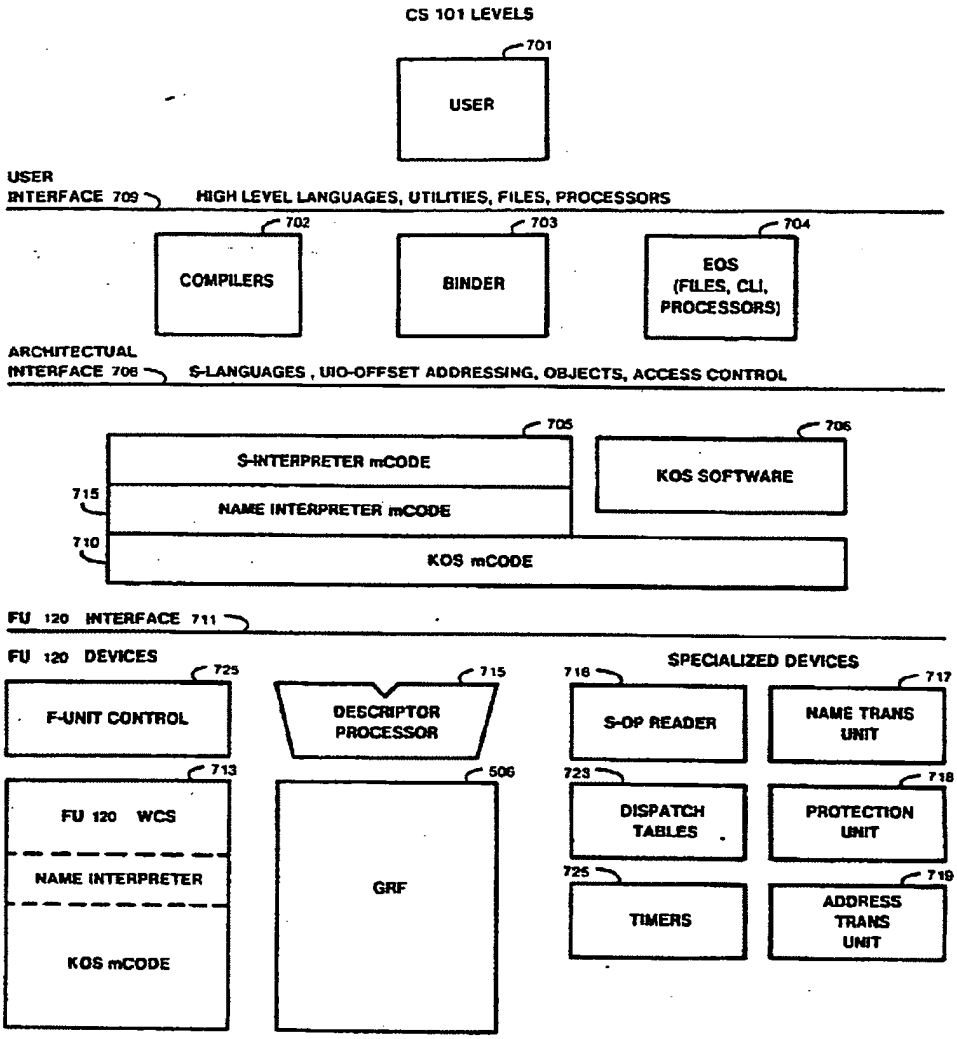


FIG 7

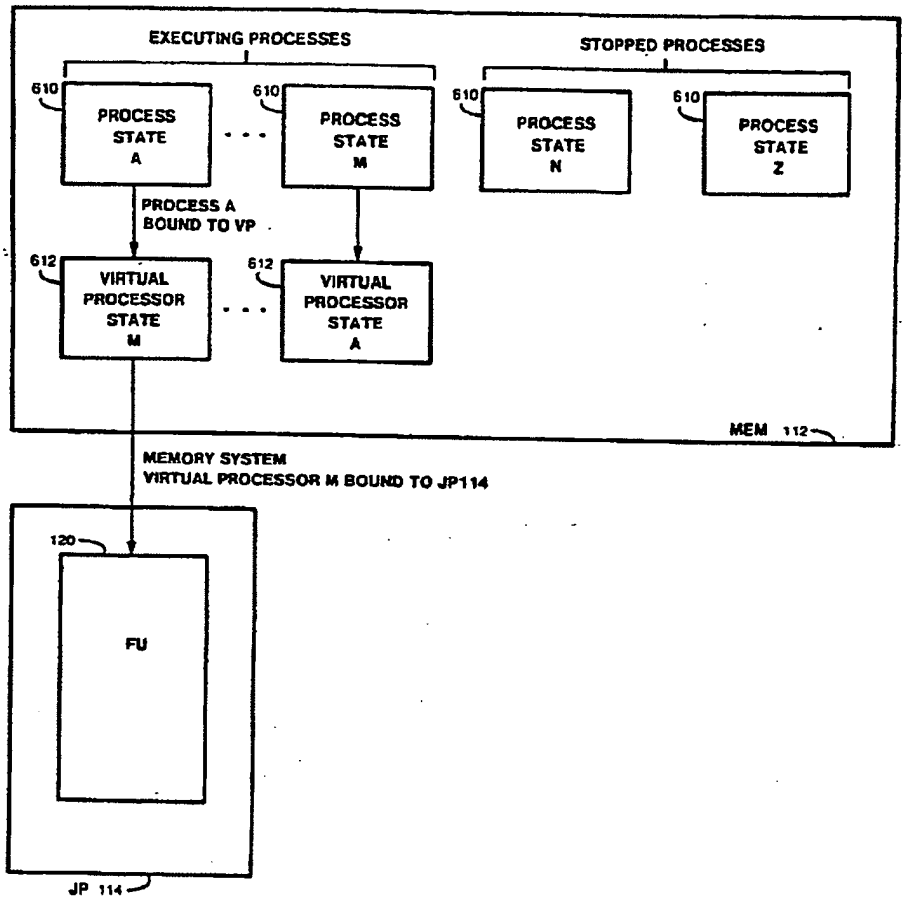


FIG 8

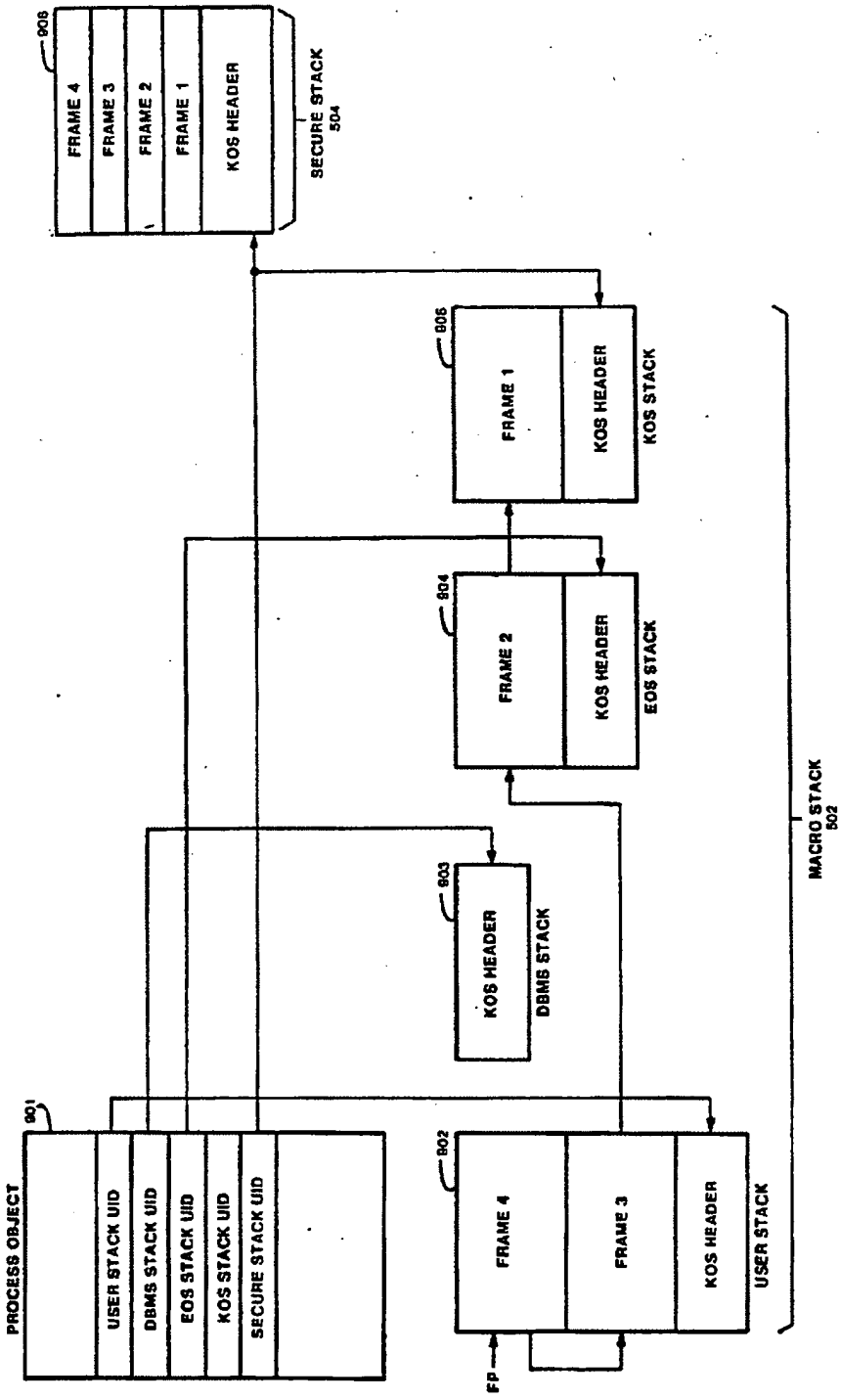


FIG 9

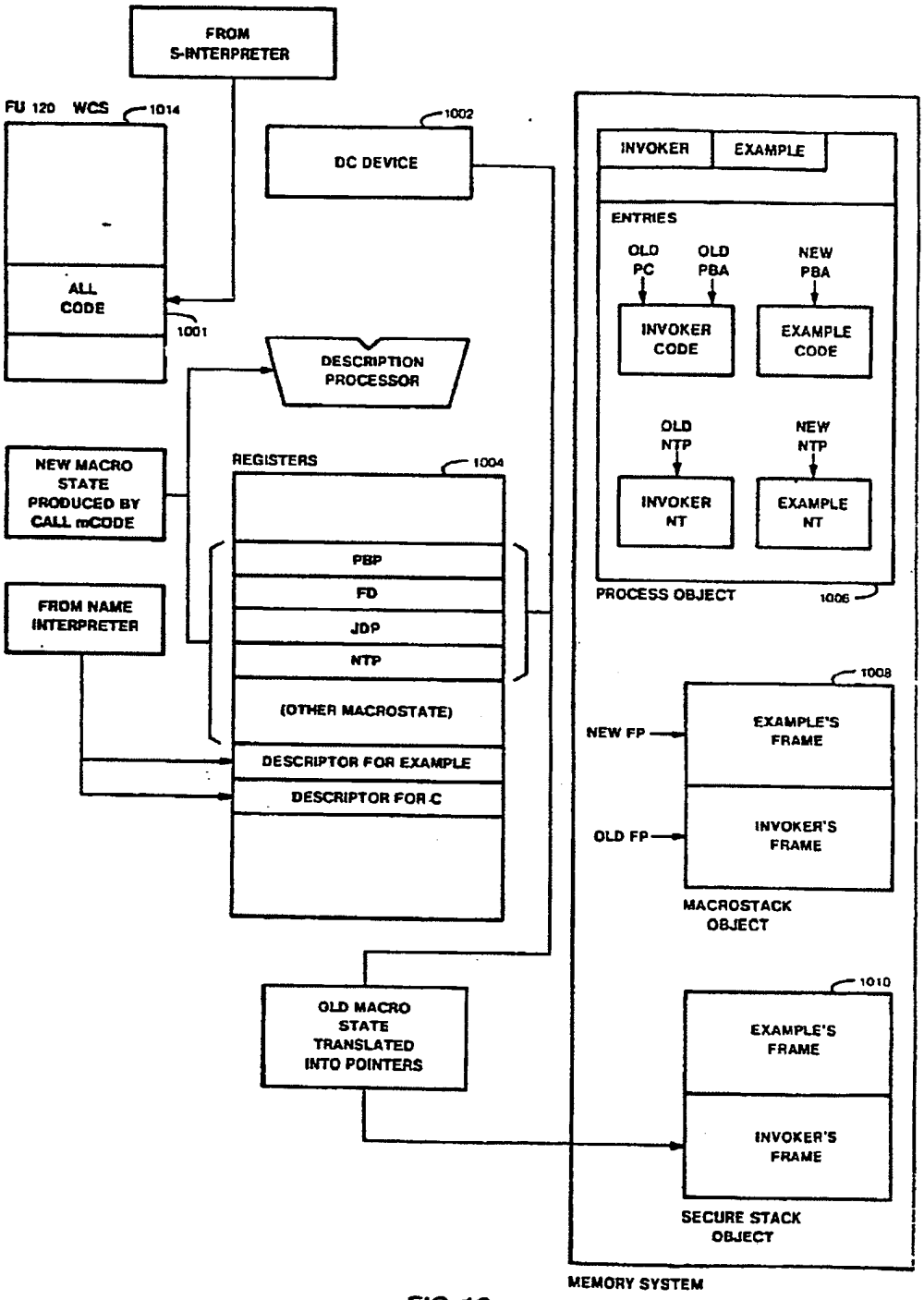


FIG 10

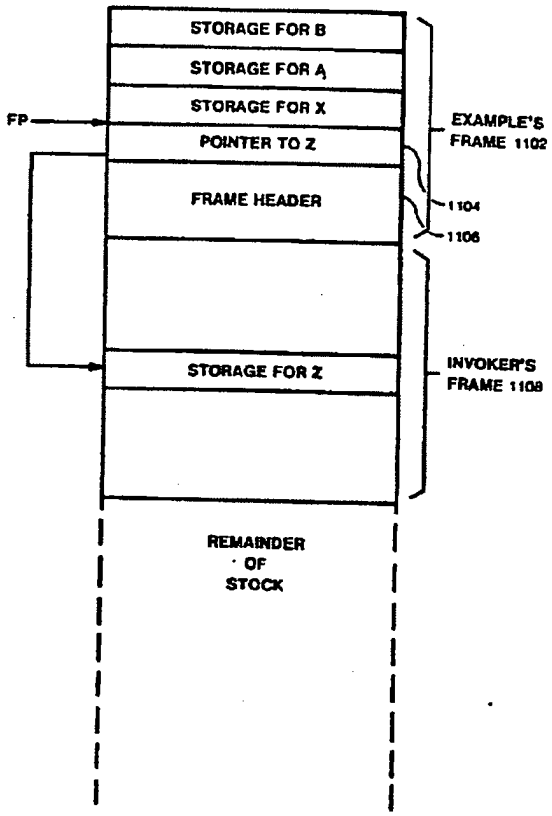


FIG 11

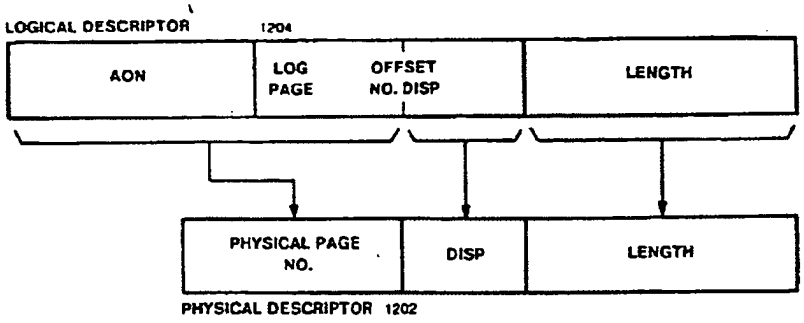


FIG 12

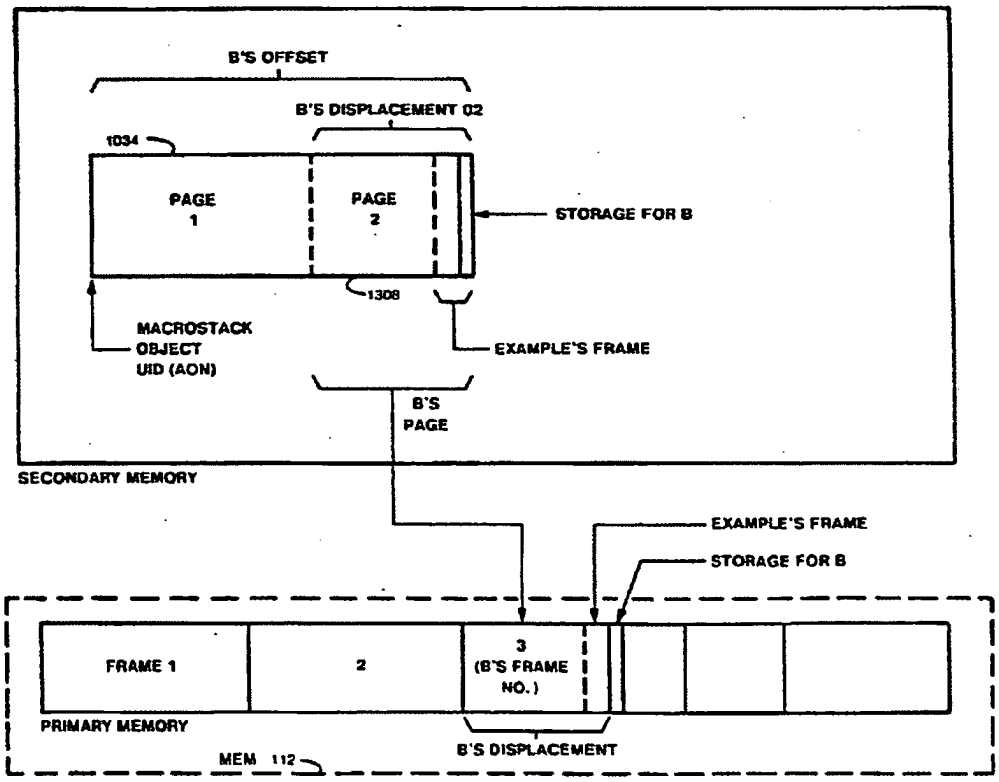


FIG 13

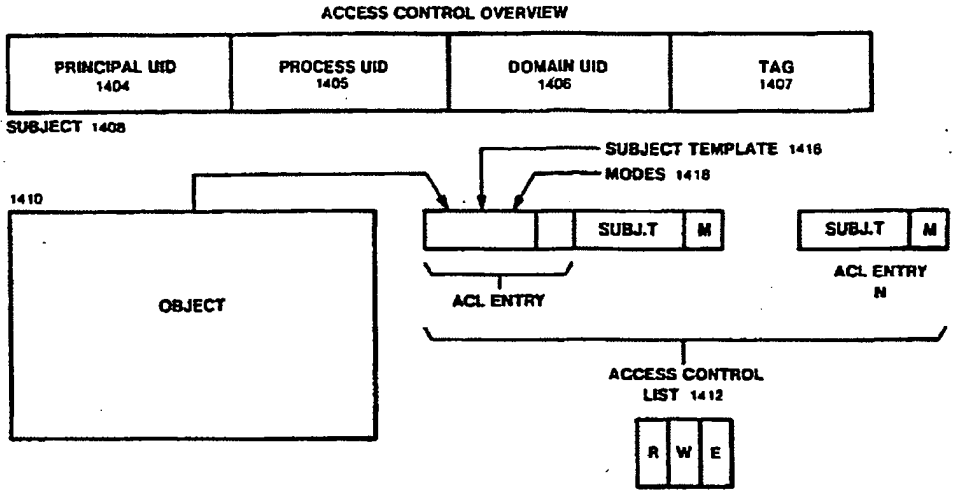


FIG 14

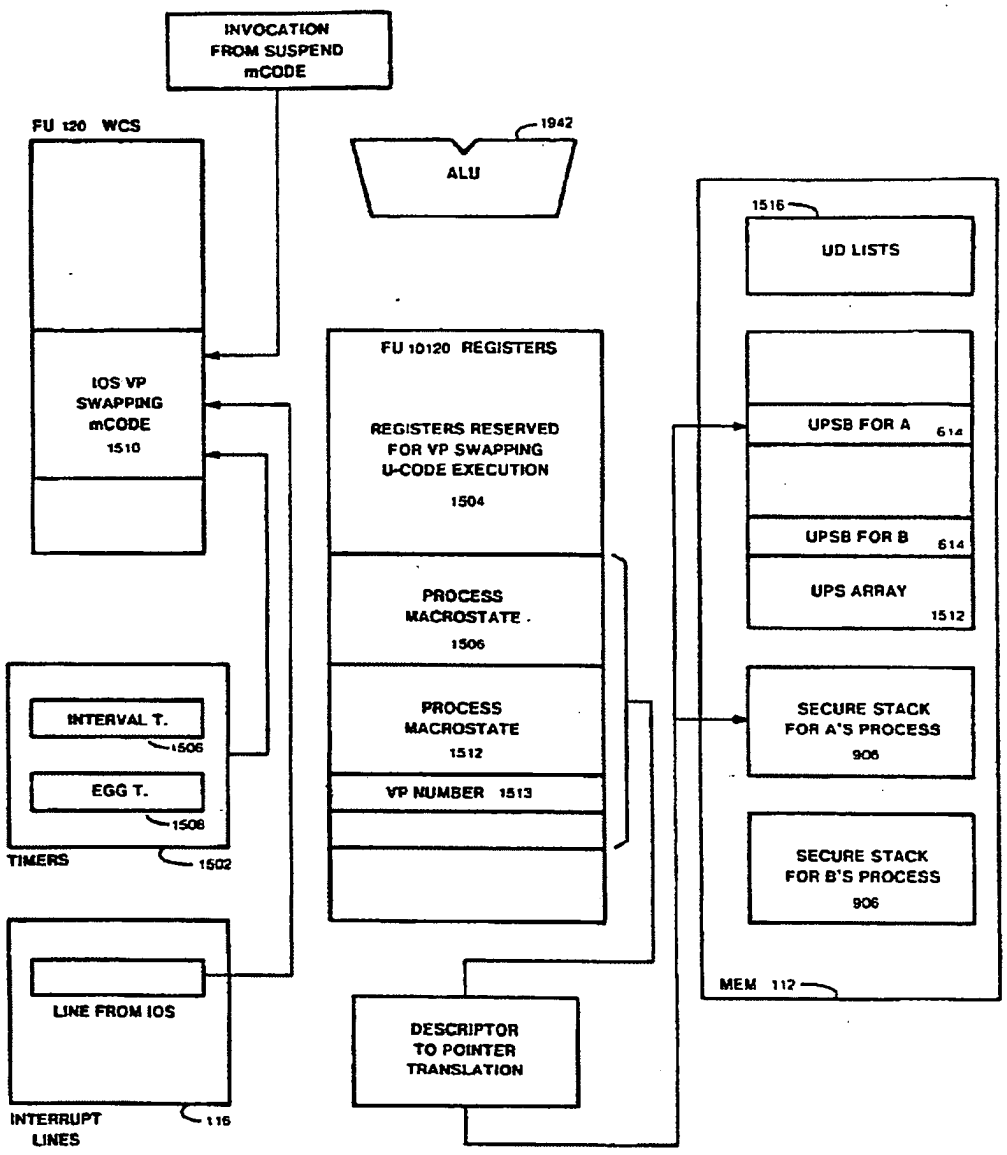


FIG 15

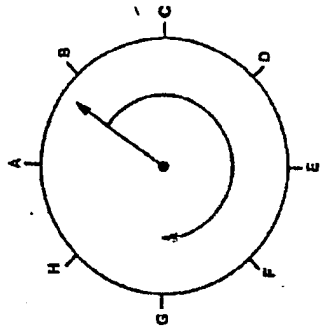


FIG 17

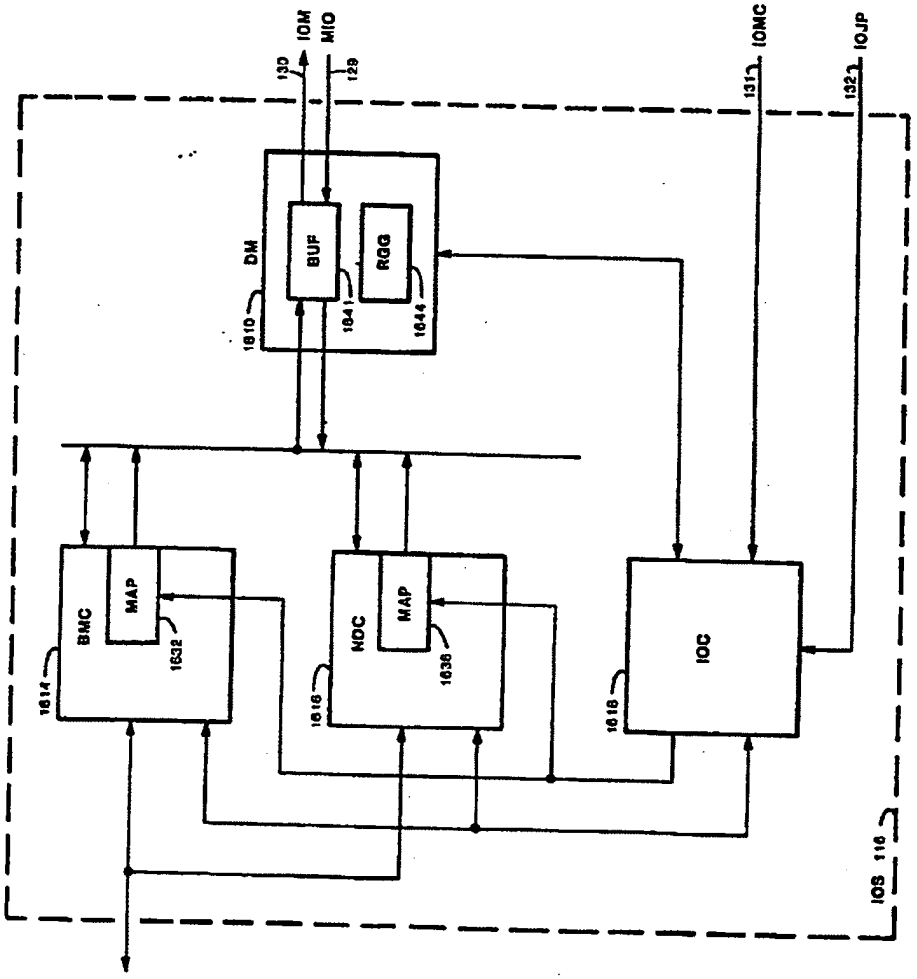


FIG 16

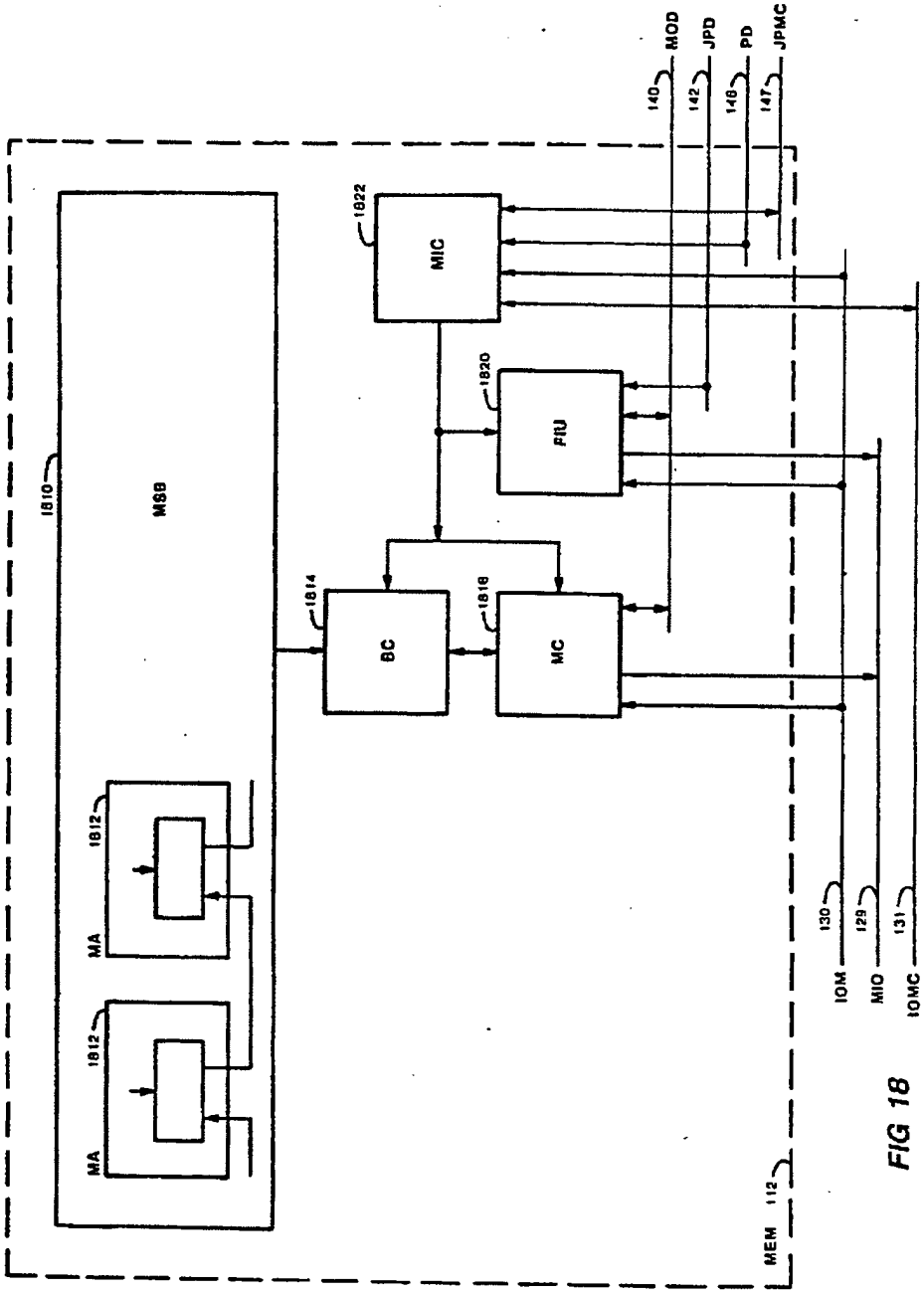


FIG 18

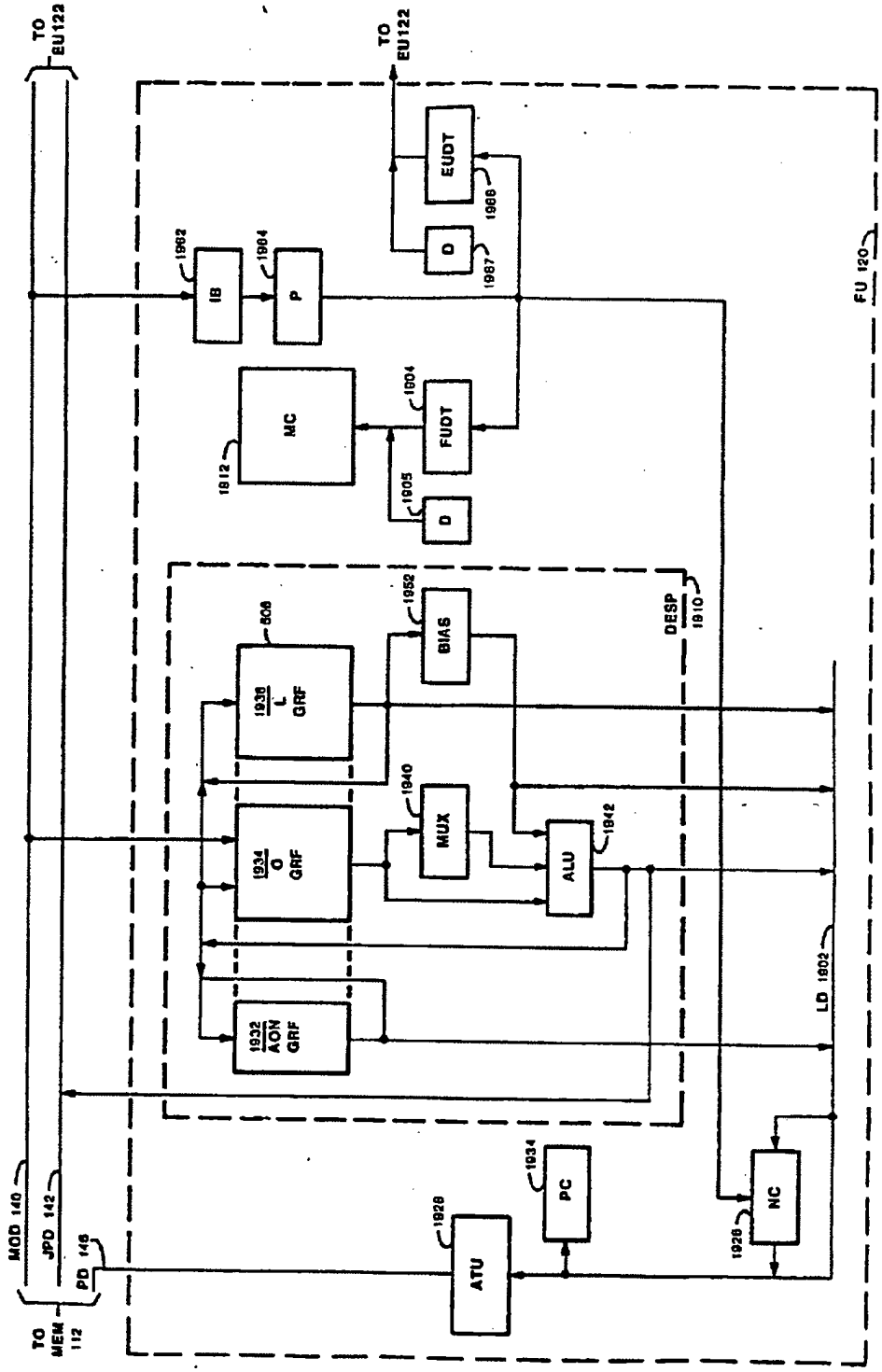


FIG 19

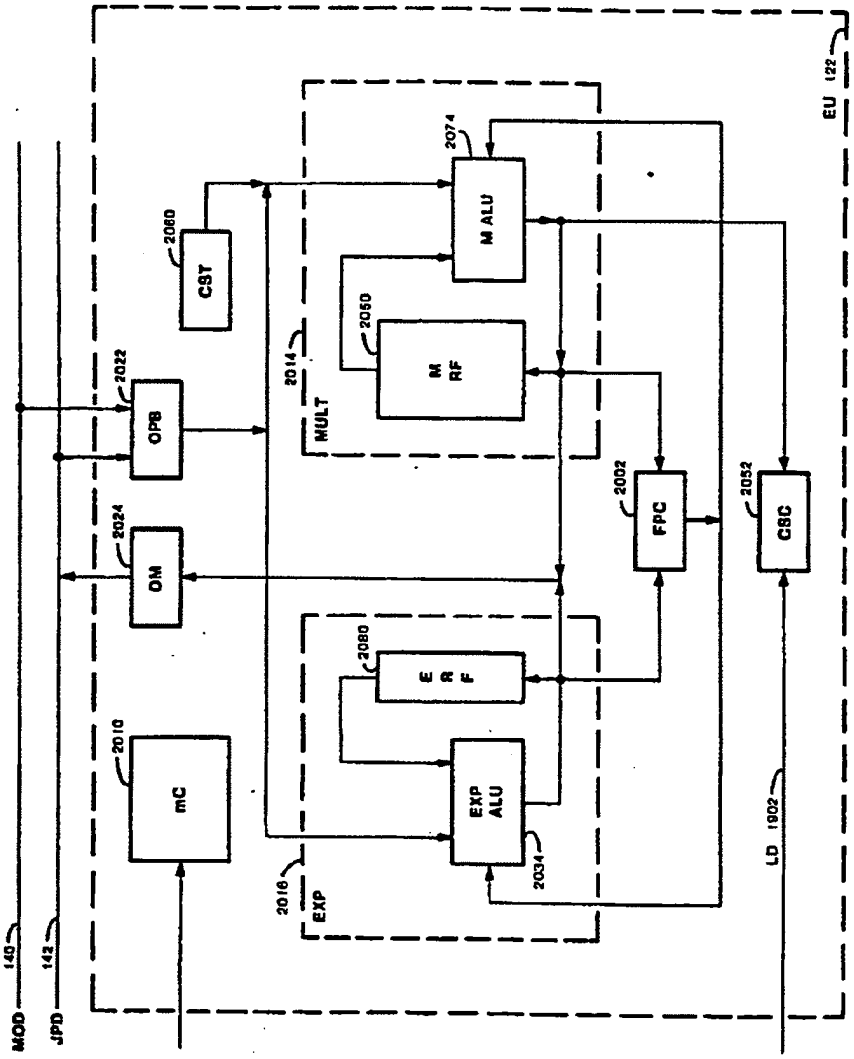


FIG 20

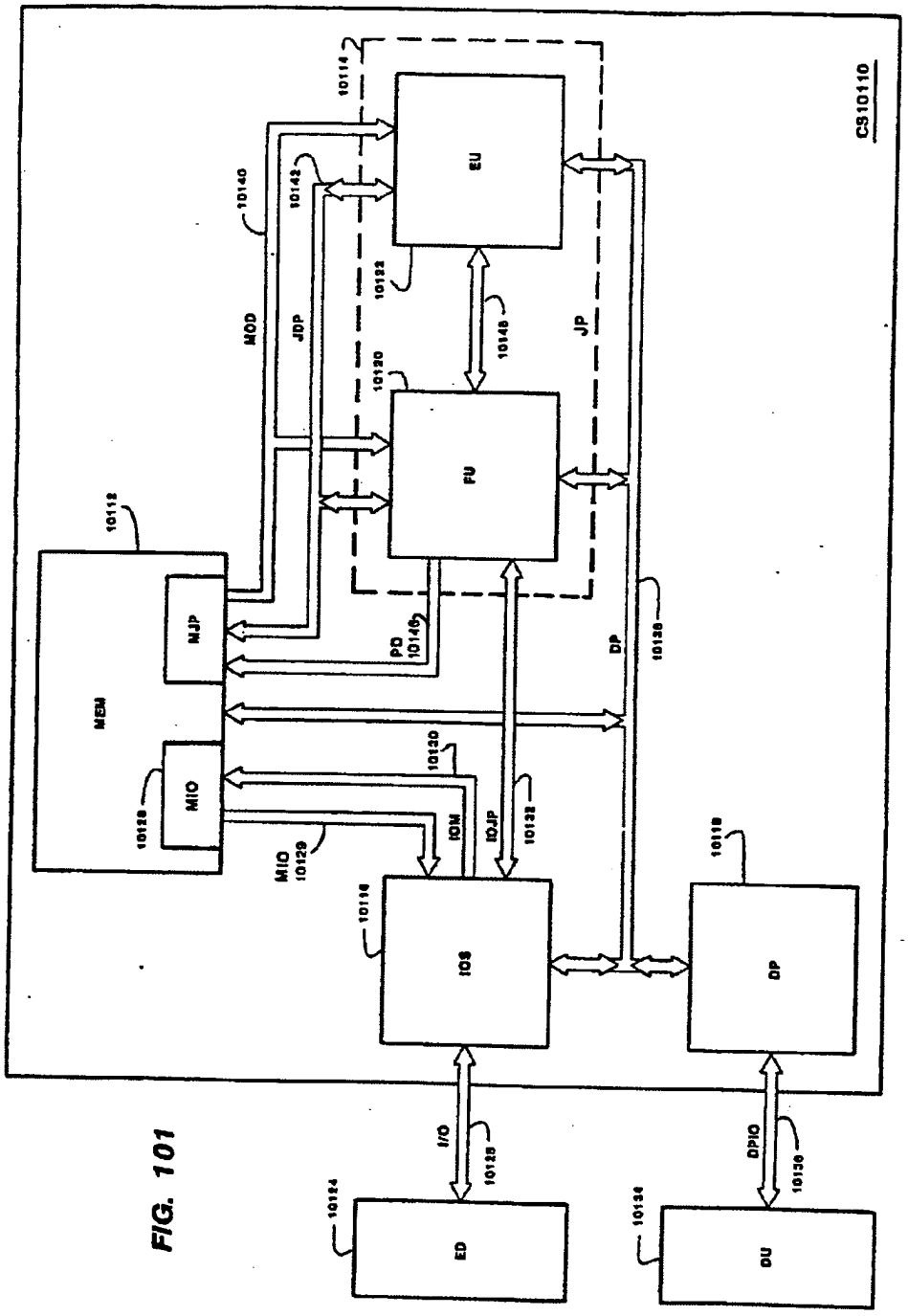


FIG. 101

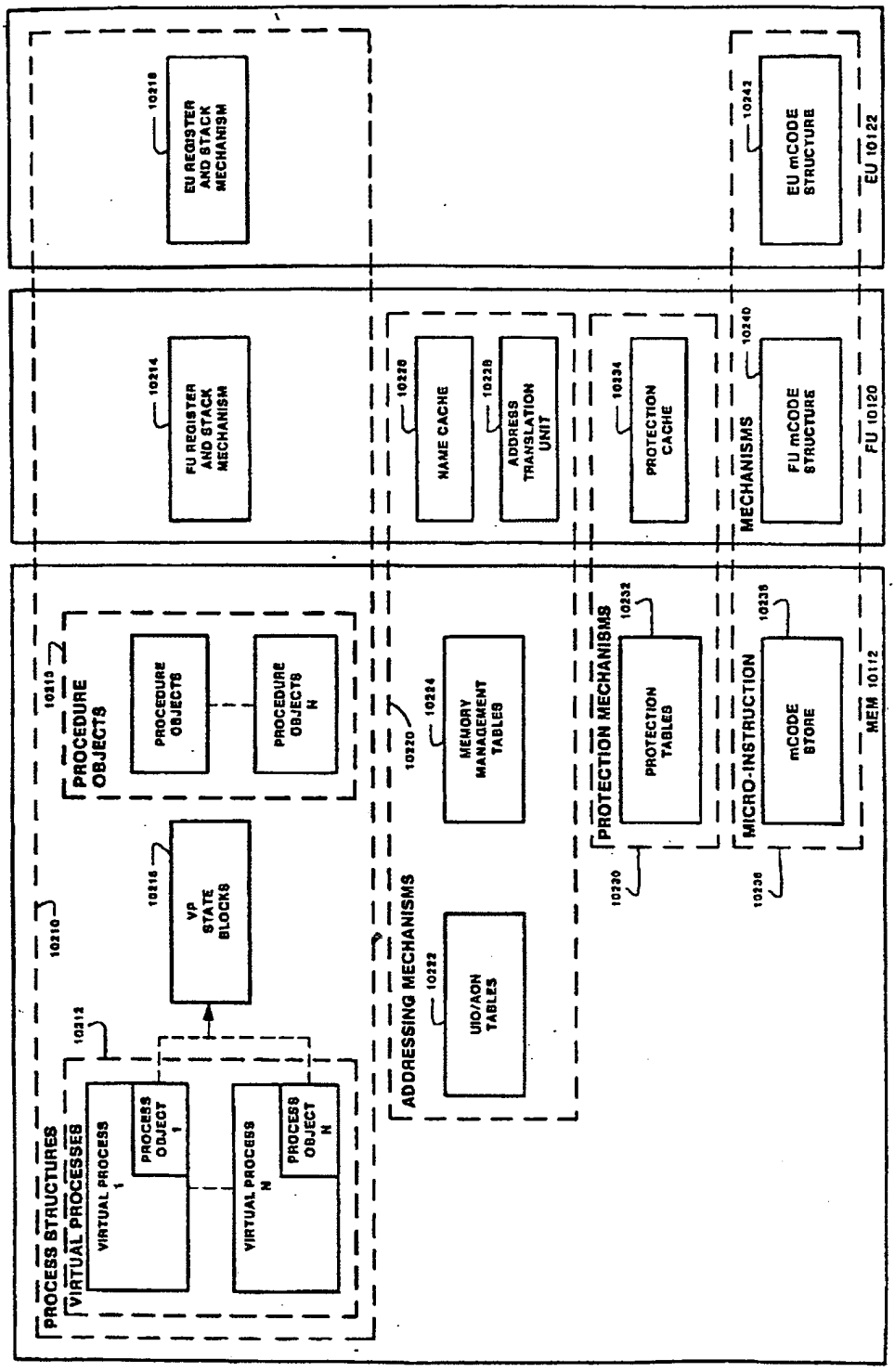


FIG. 102

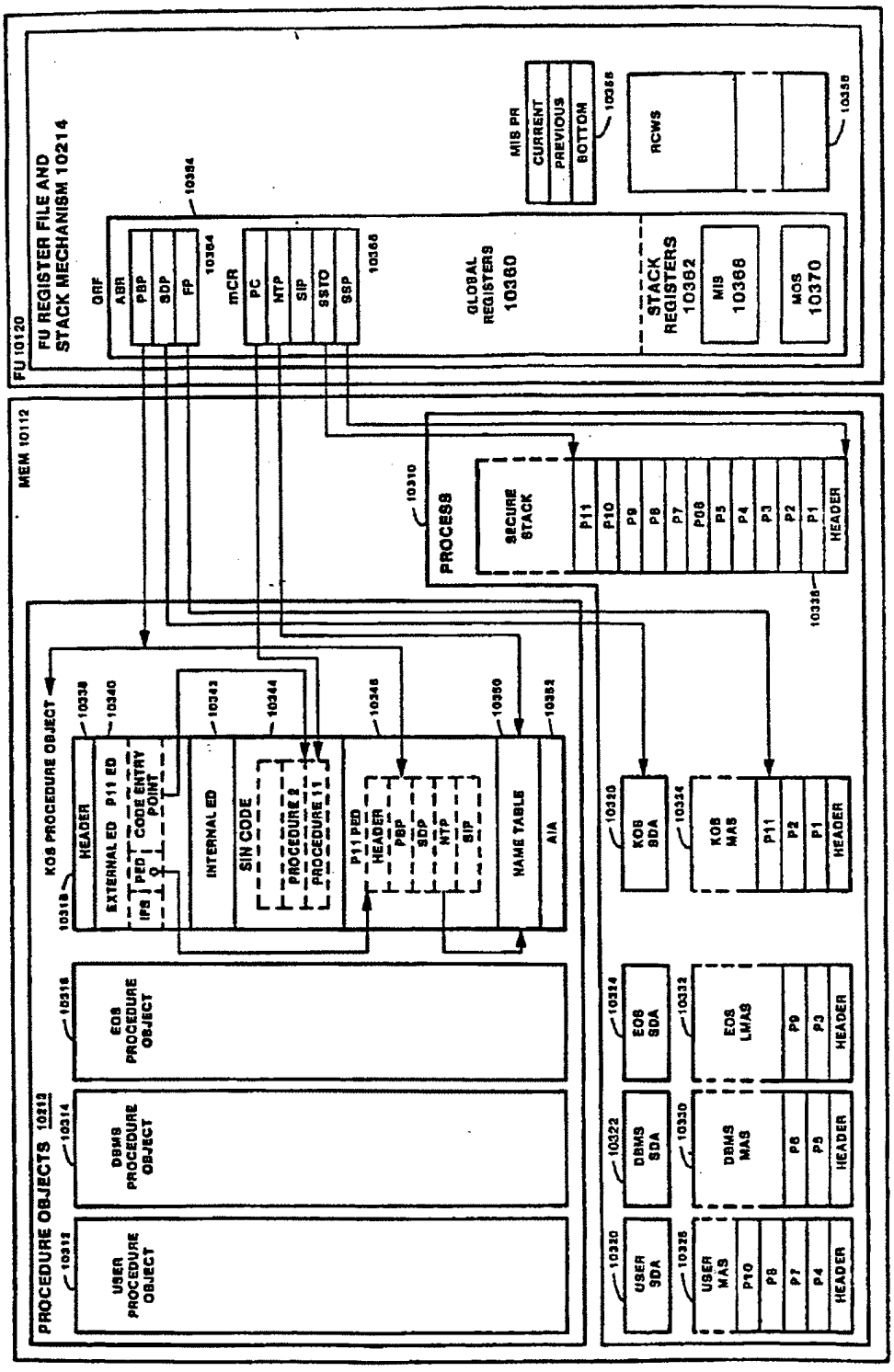


FIG. 103

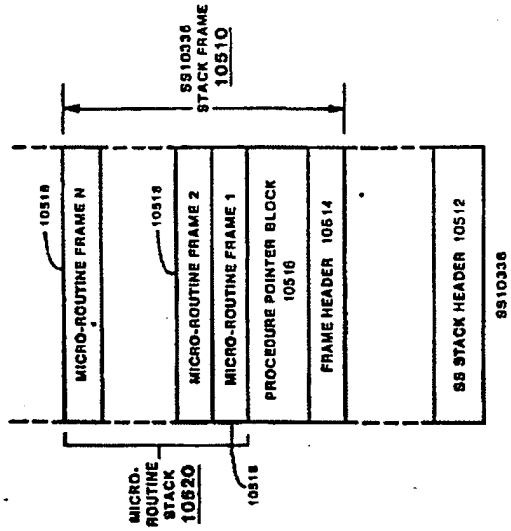


FIG. 105

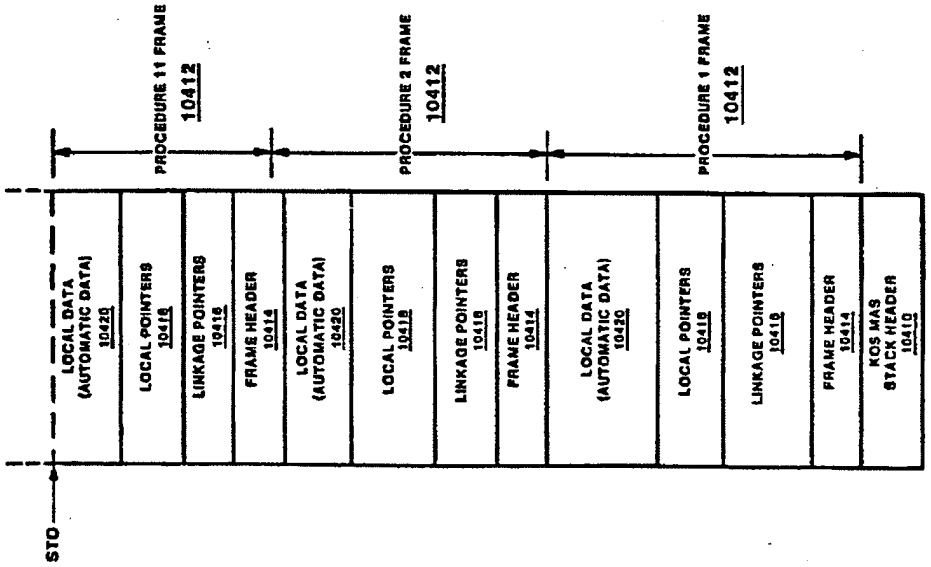


FIG. 104

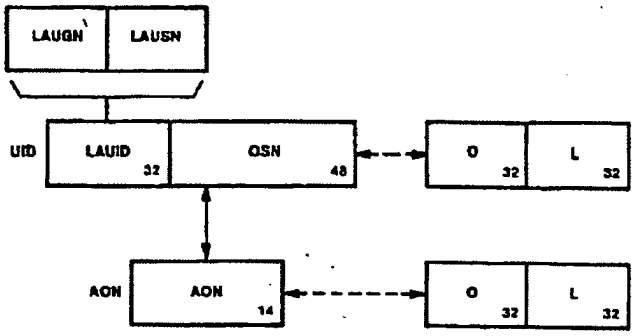


FIG. 106A

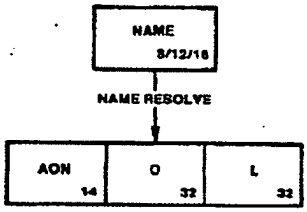


FIG. 106B

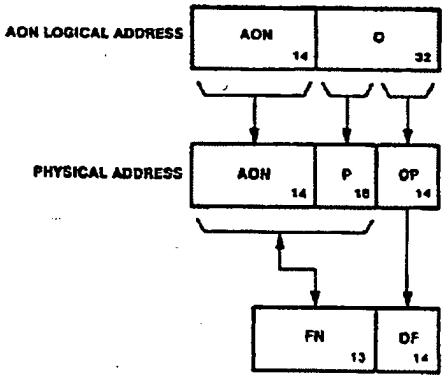


FIG. 106C

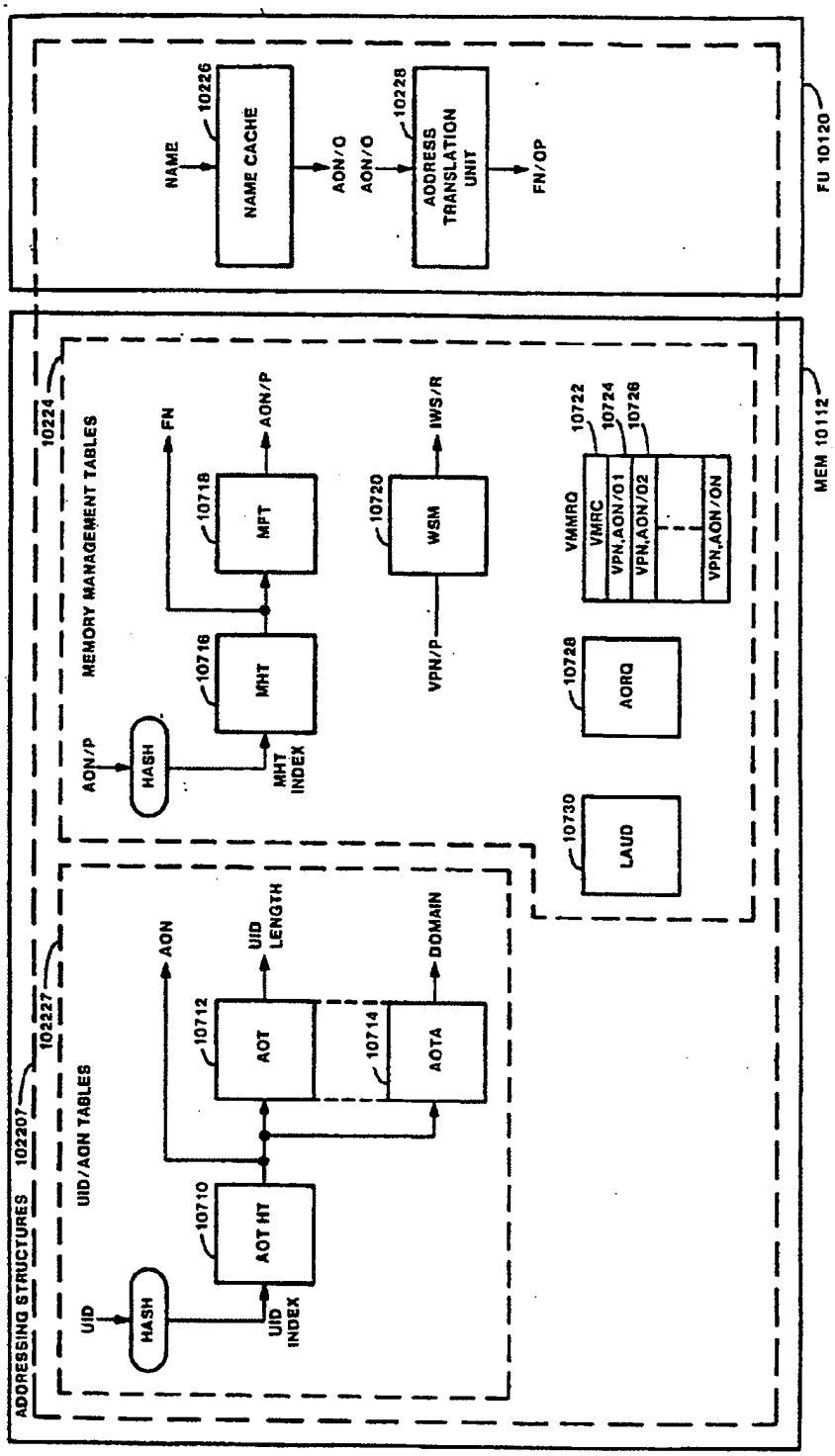


FIG 107

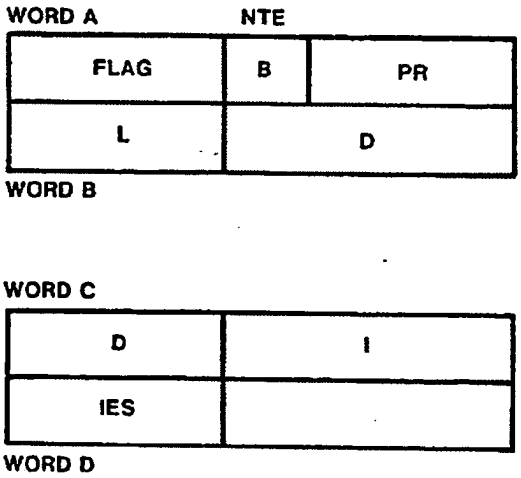


FIG. 108

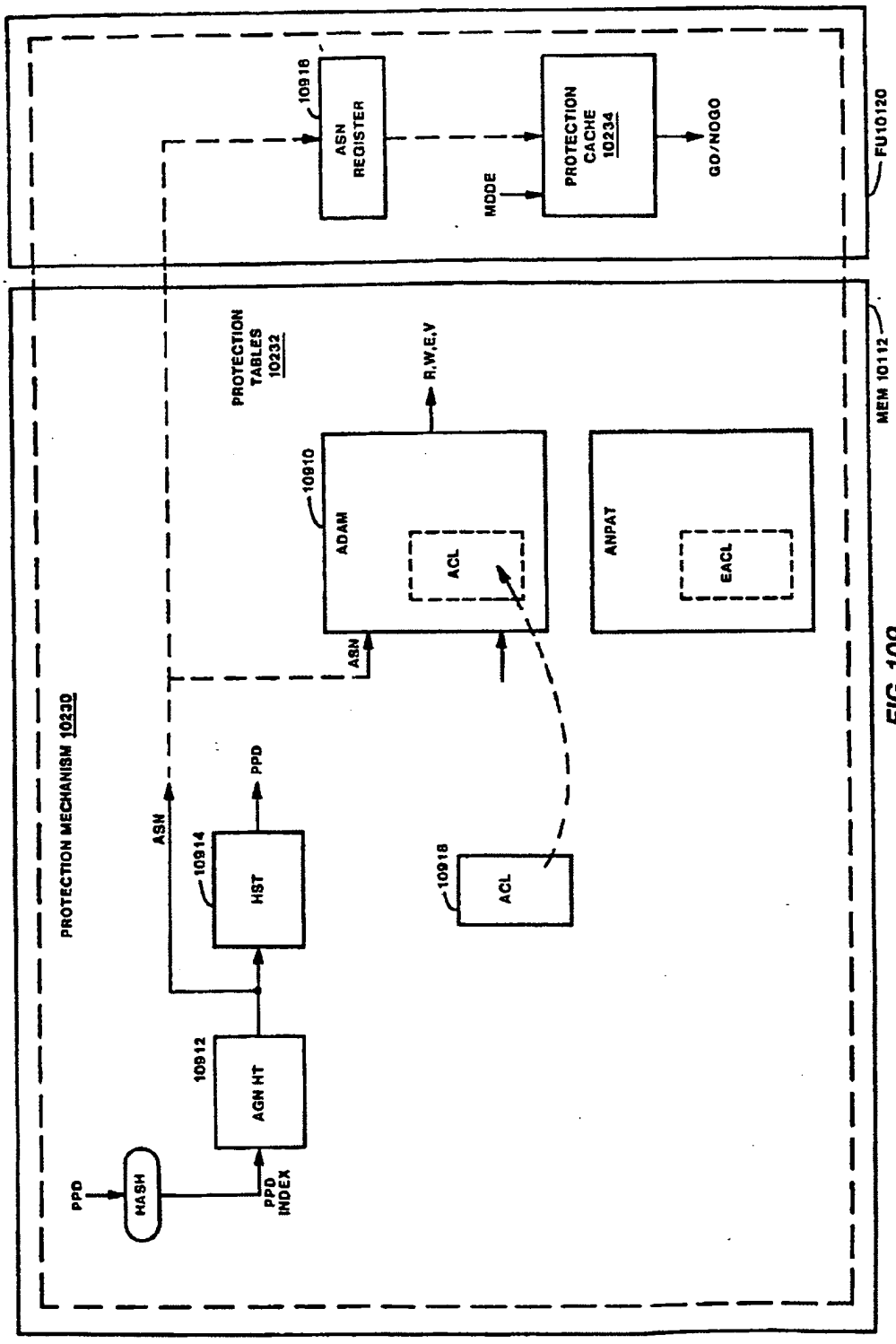


FIG 109

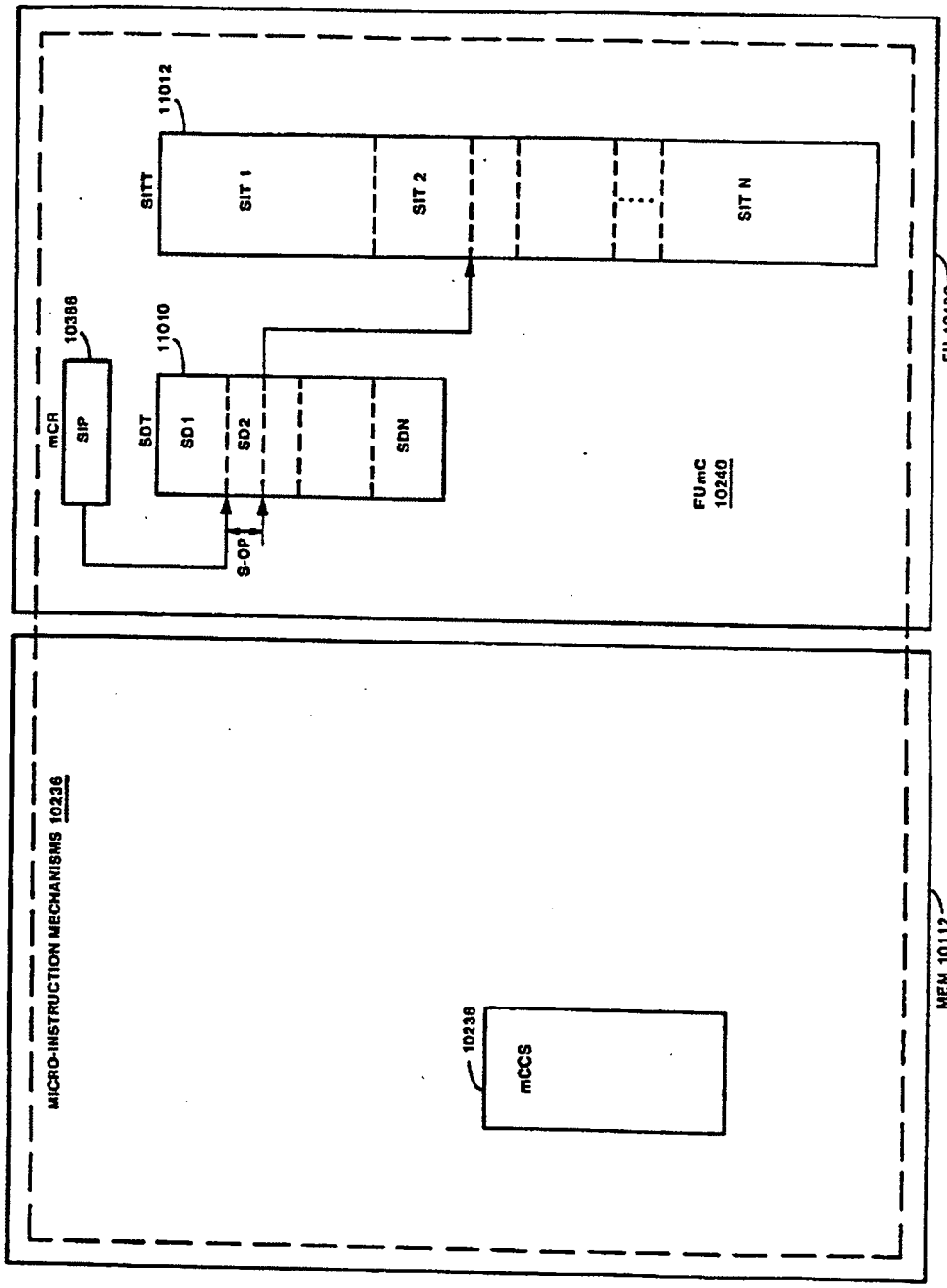


FIG 110

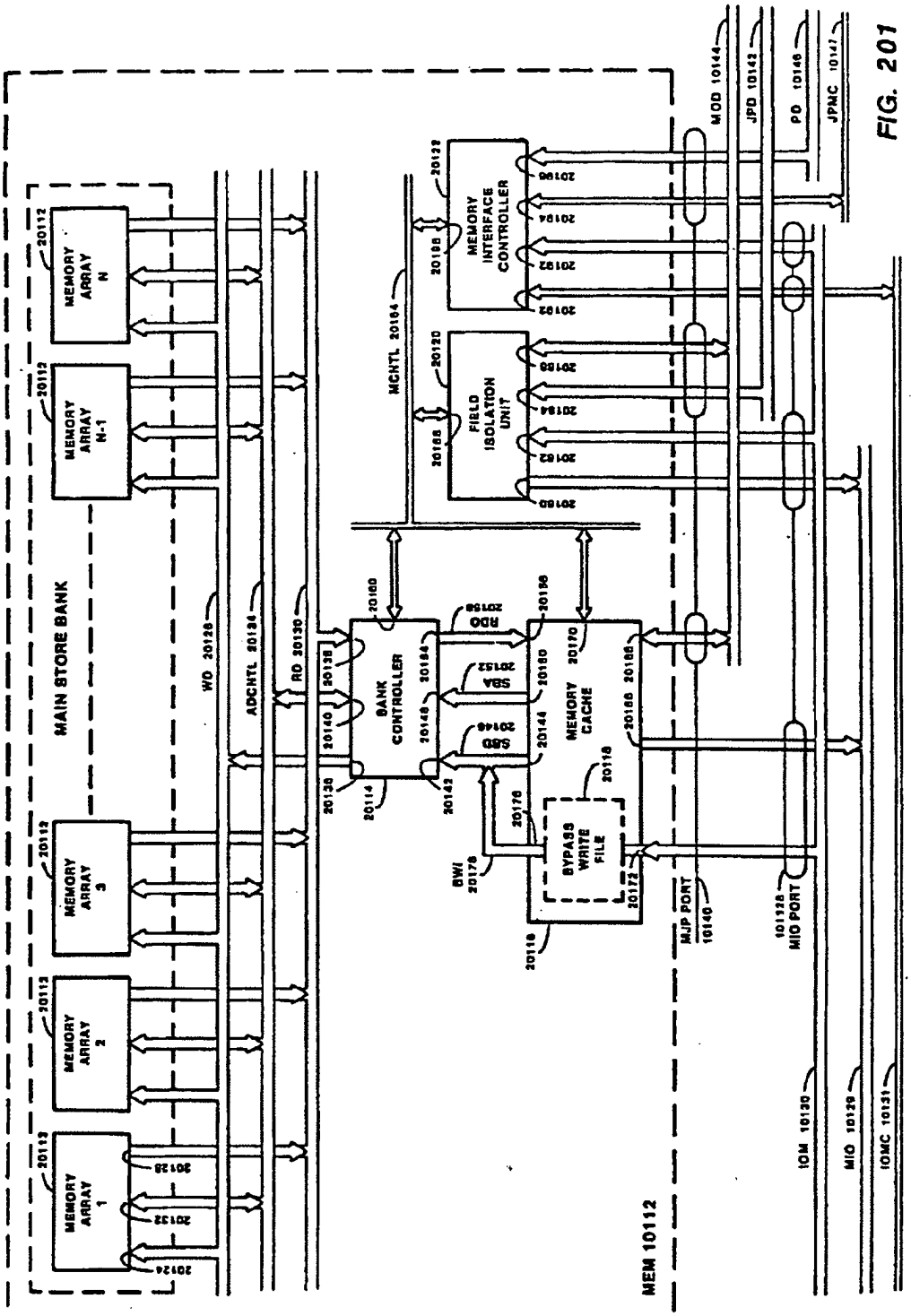


FIG. 201

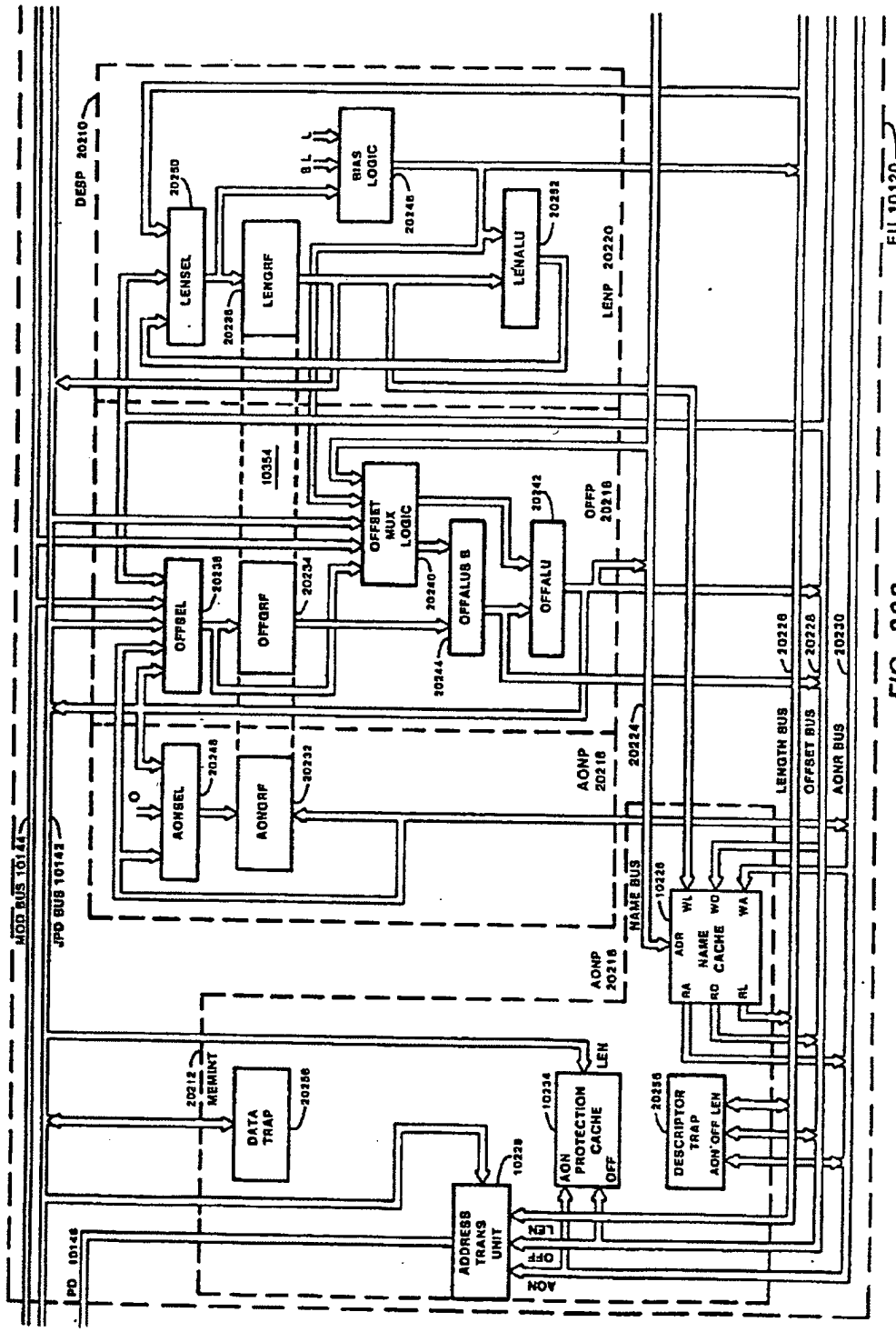


FIG. 202

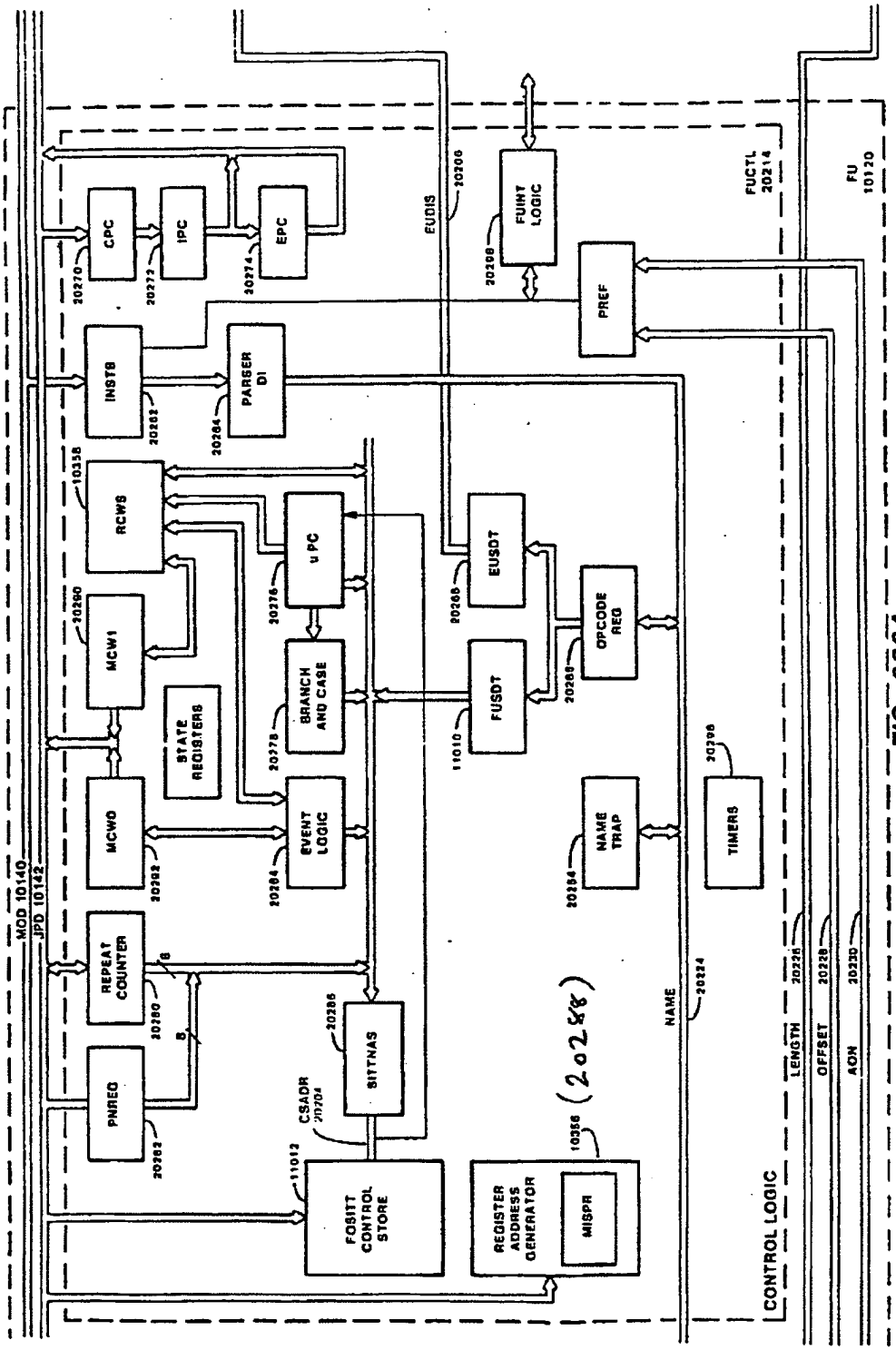


FIG. 202A

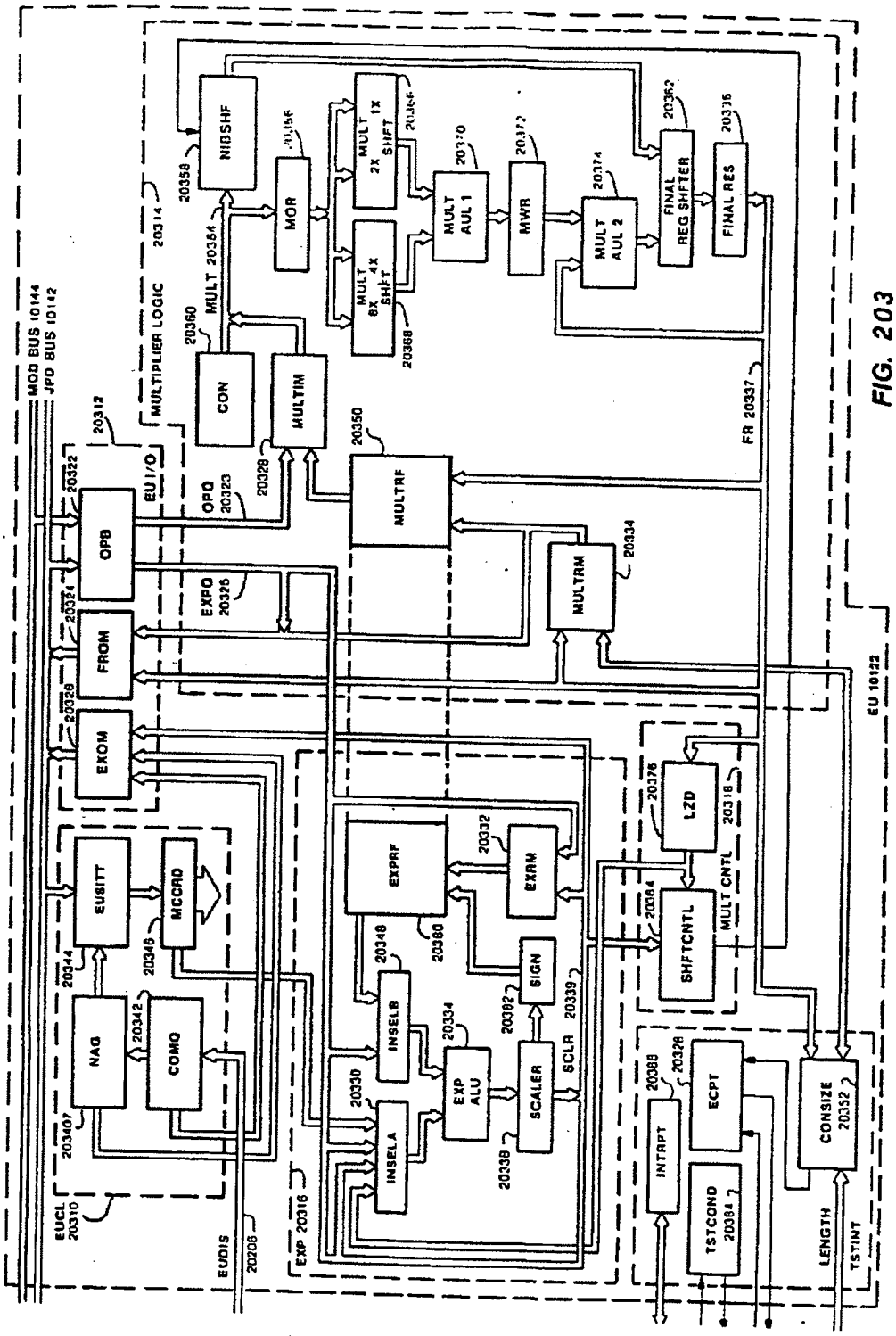


FIG. 203

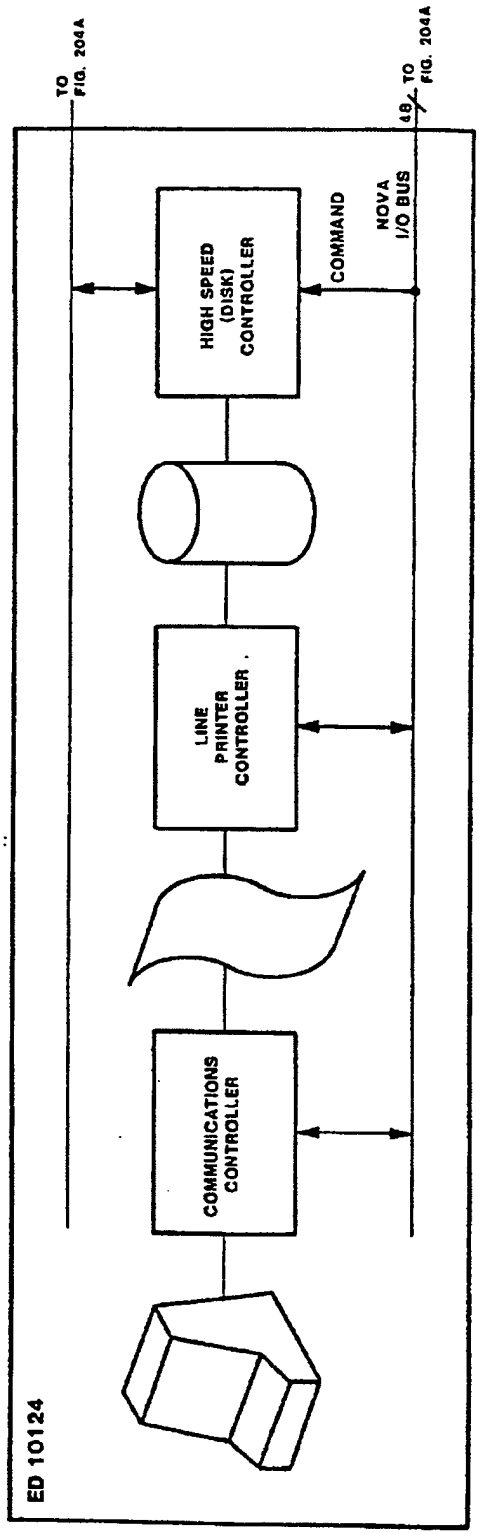


FIG. 204

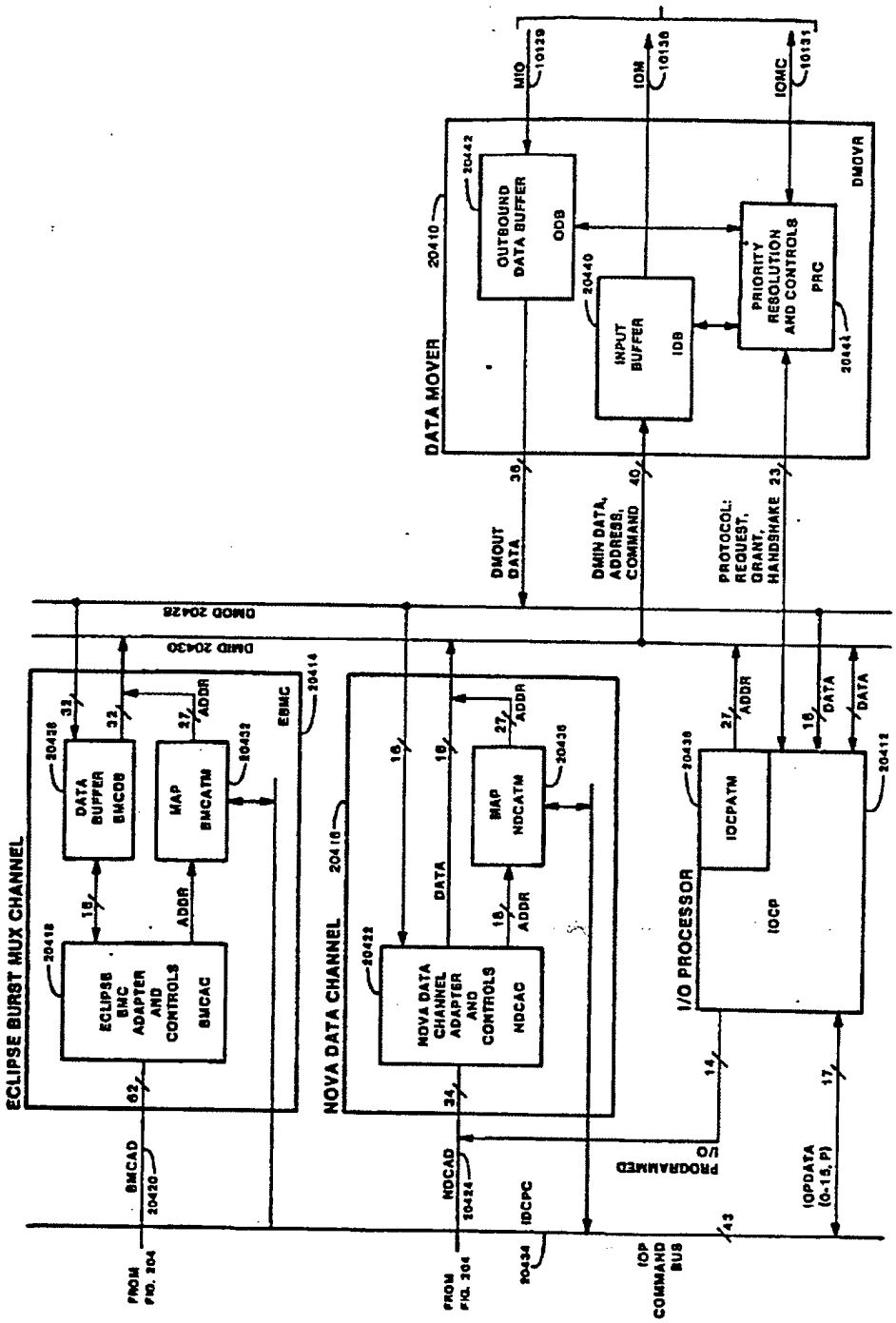


FIG. 204A

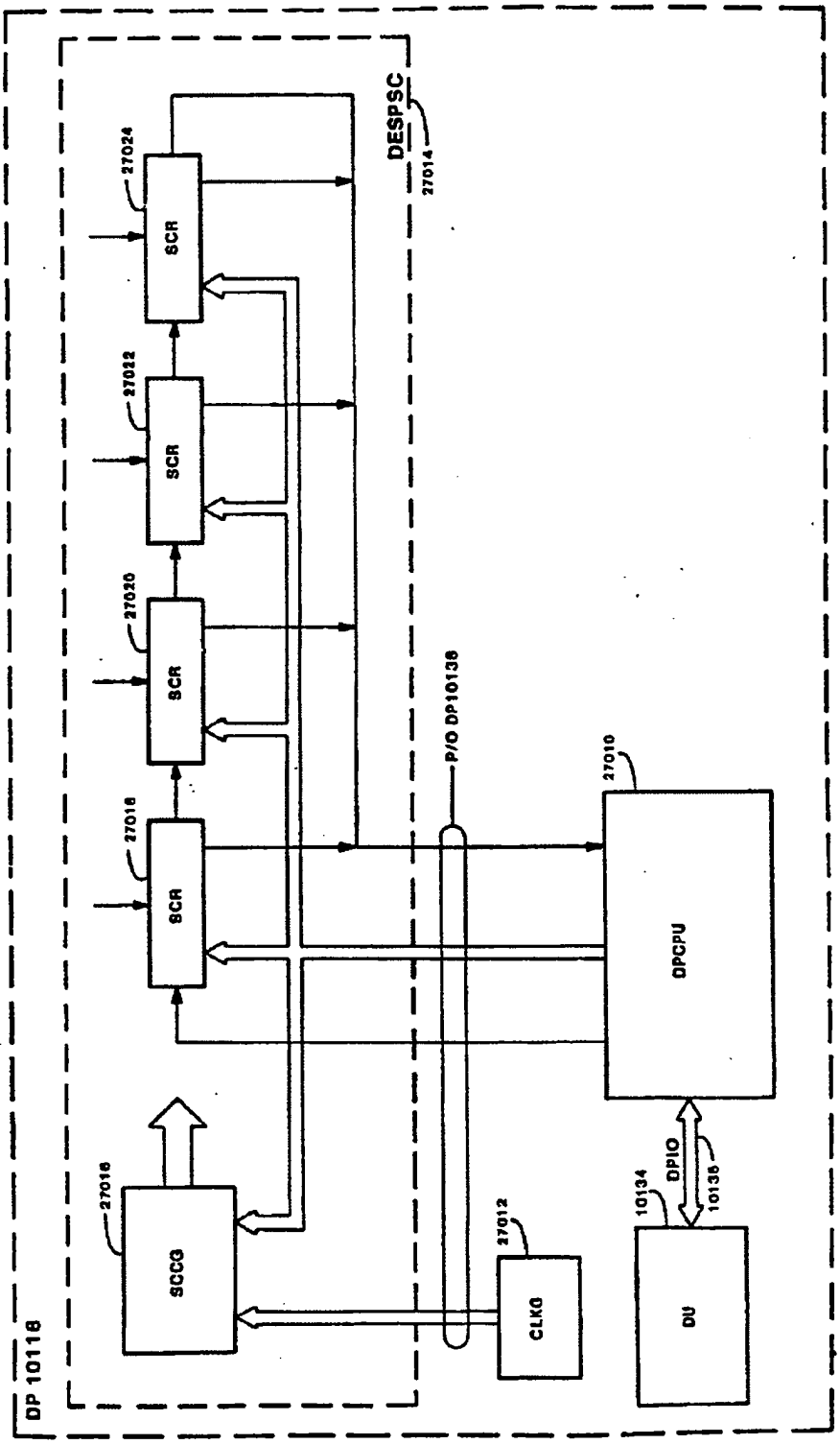


FIG. 205

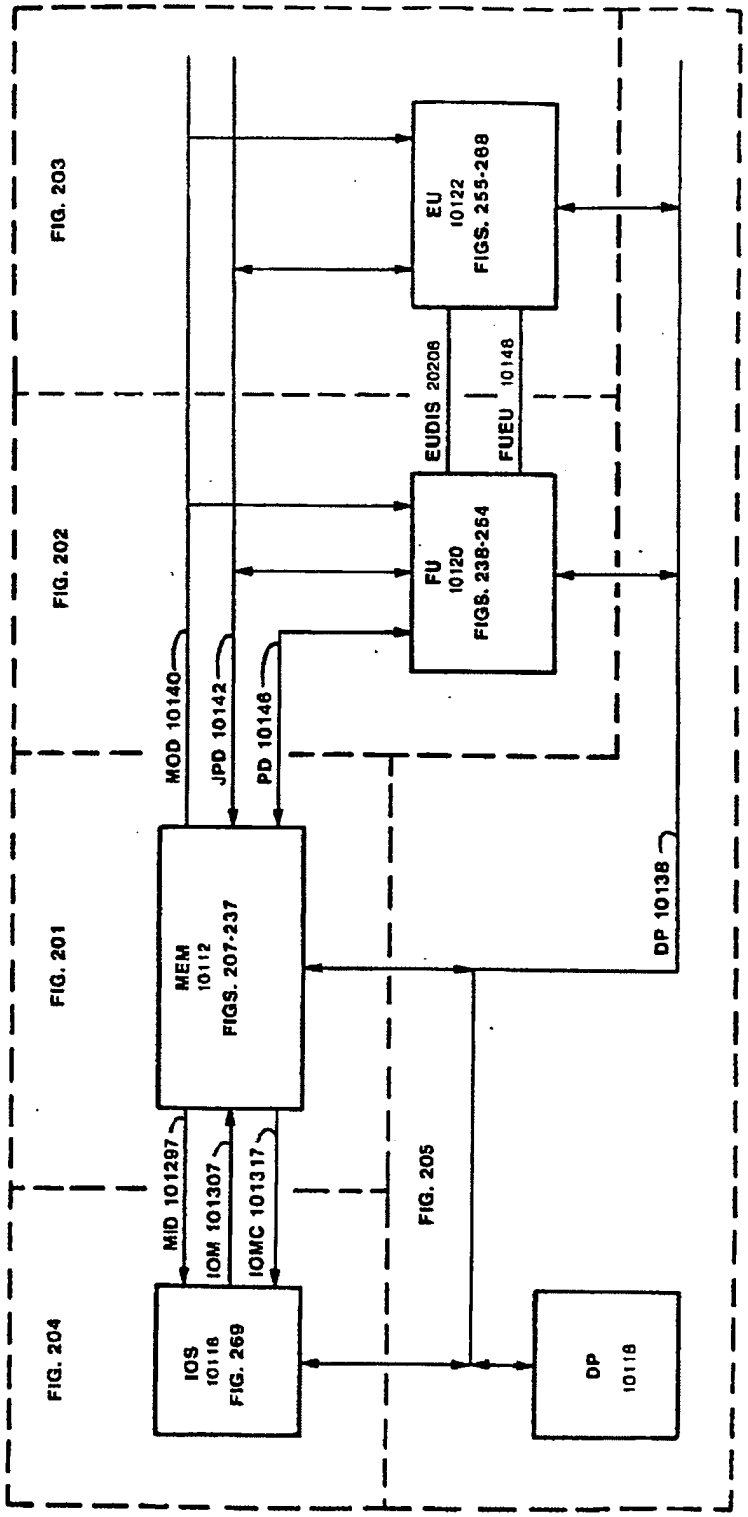


FIG. 206

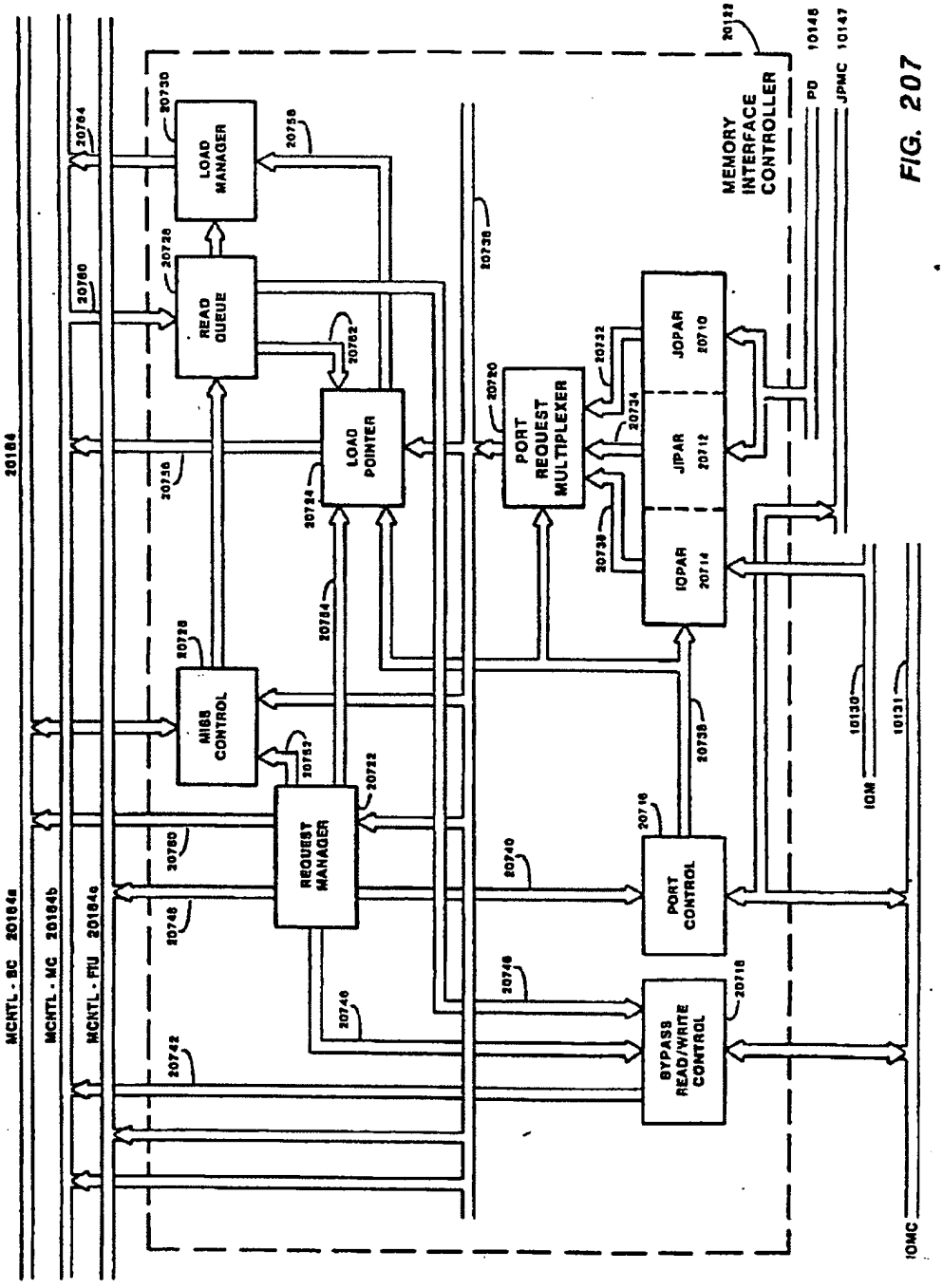


FIG. 207

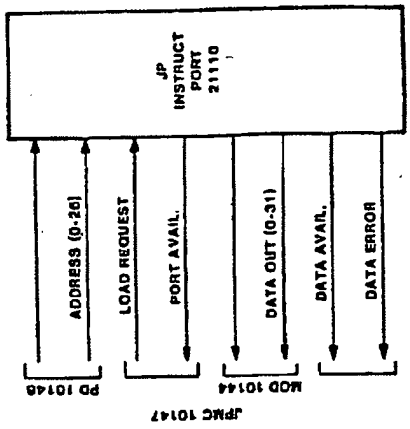


FIG. 211

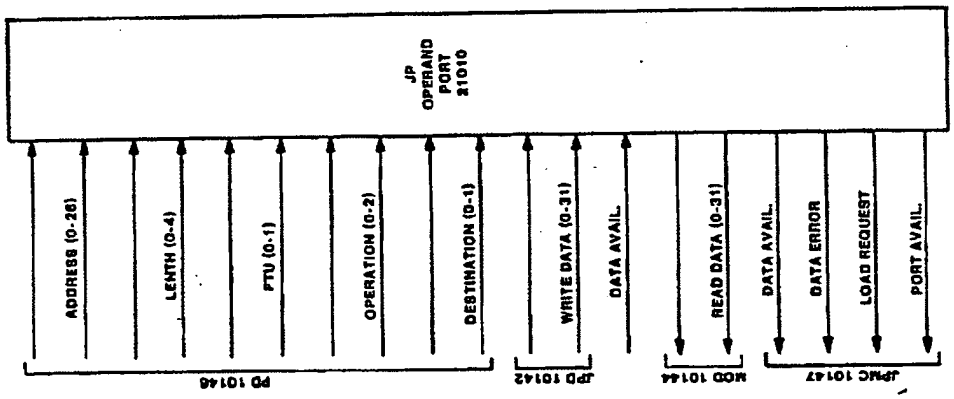


FIG. 210

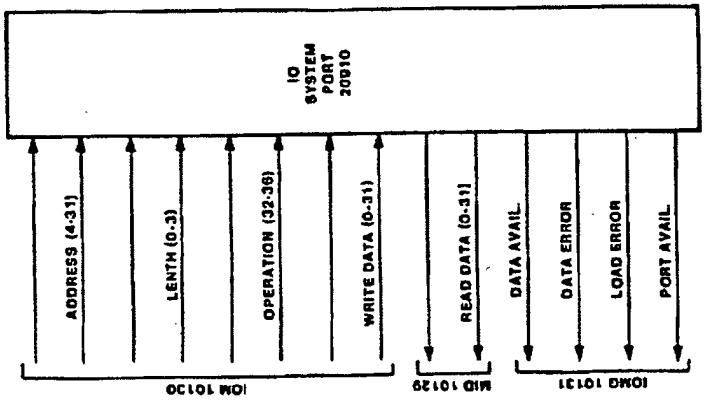


FIG. 209

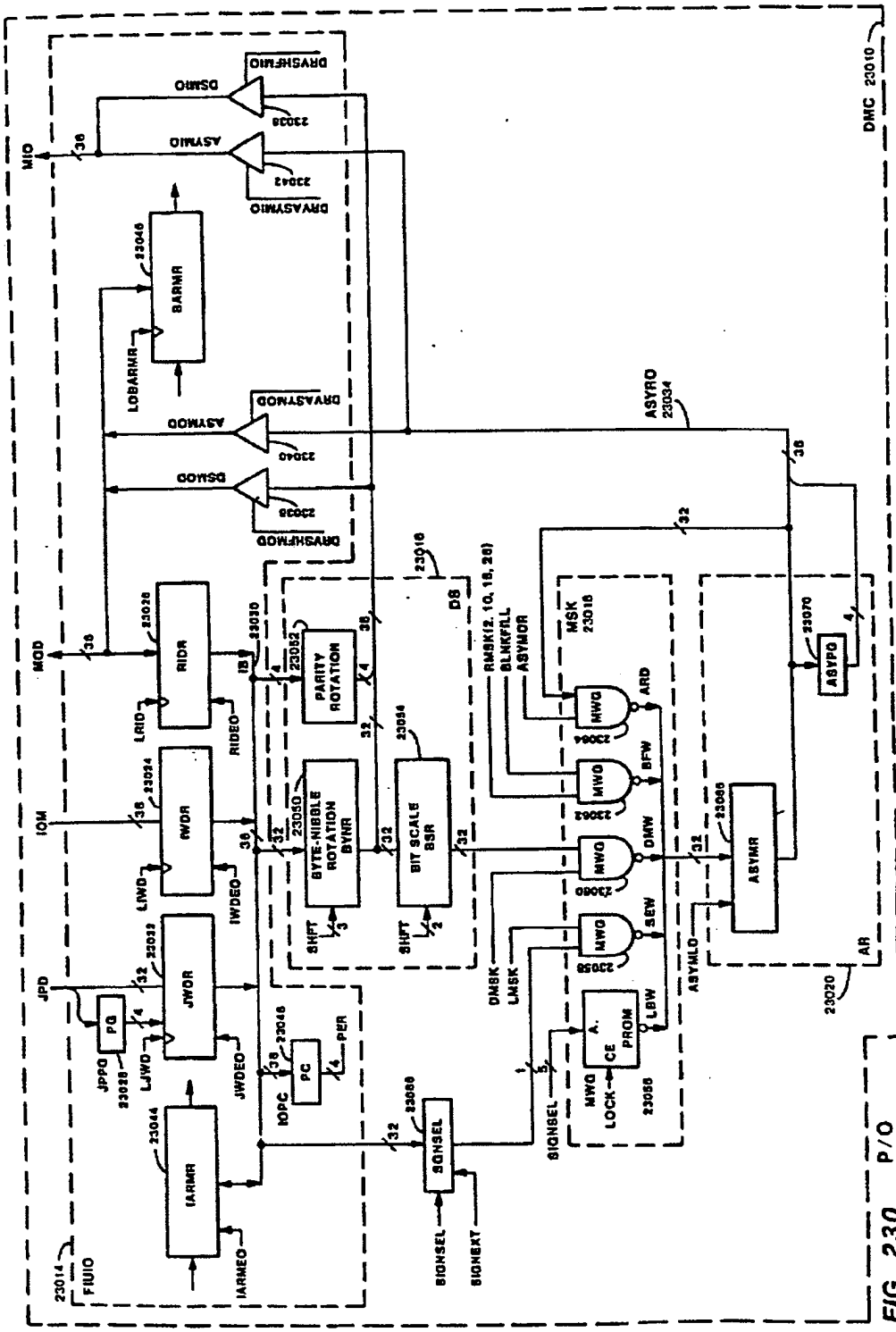
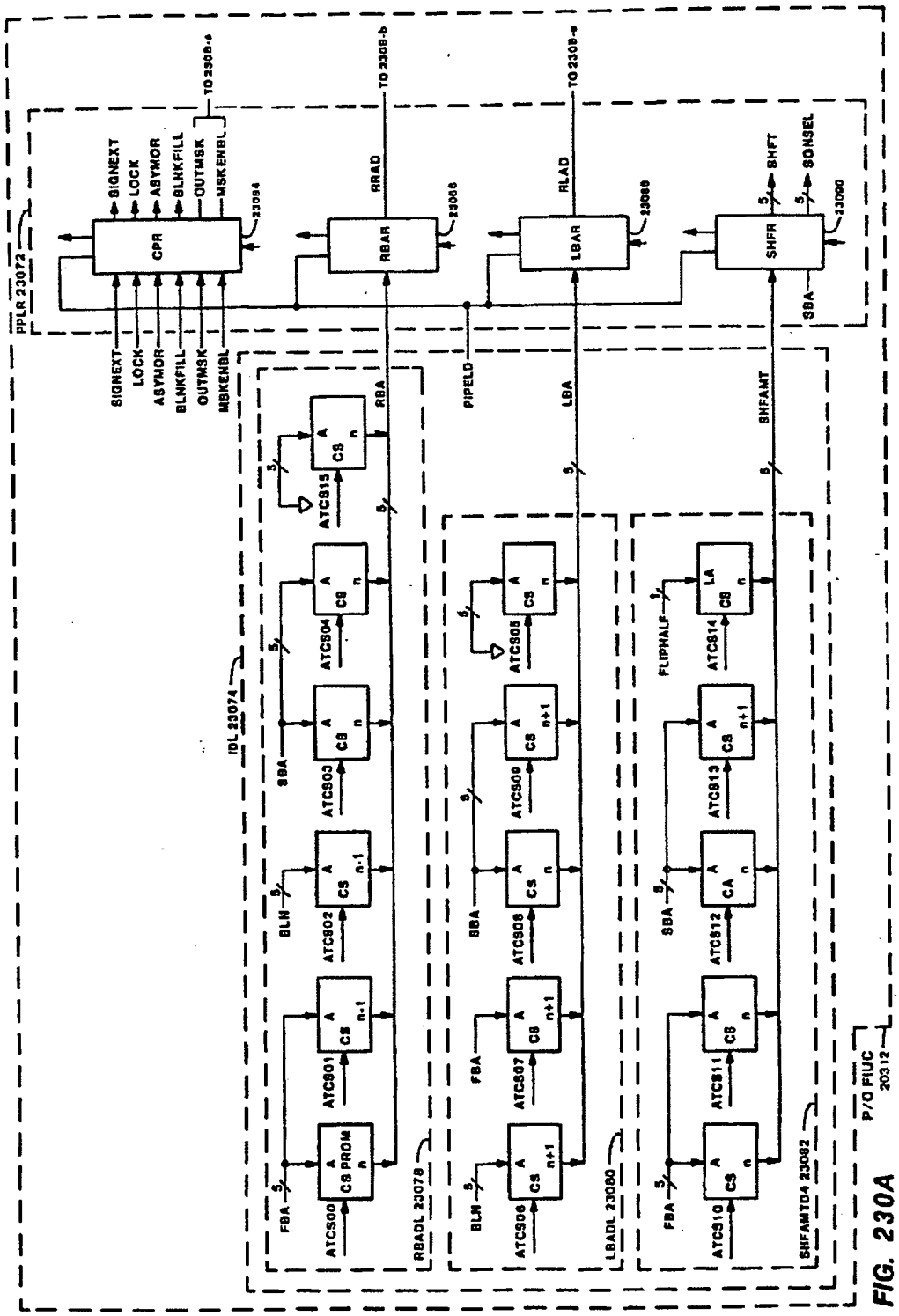


FIG. 230



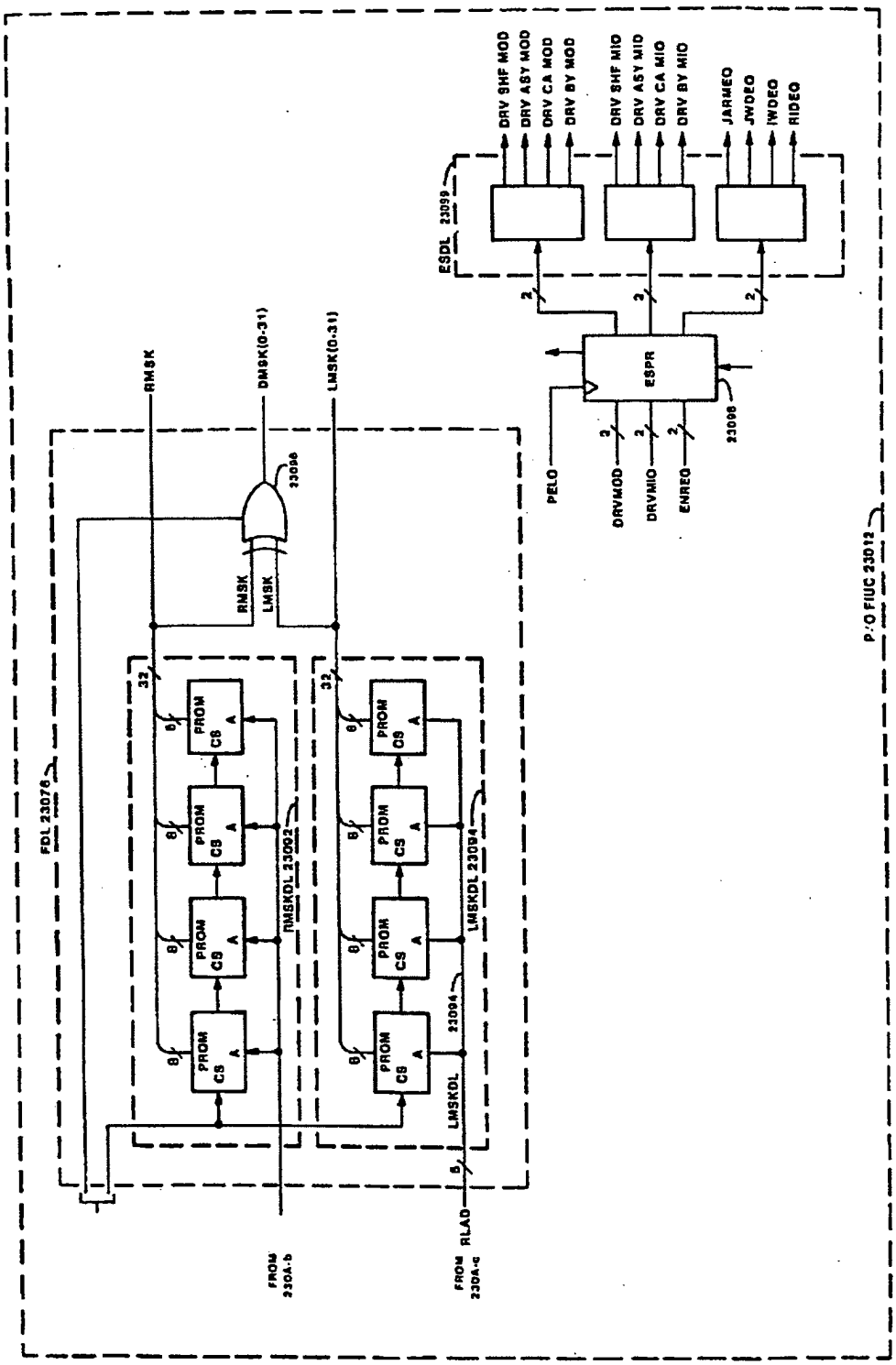


FIG 230B

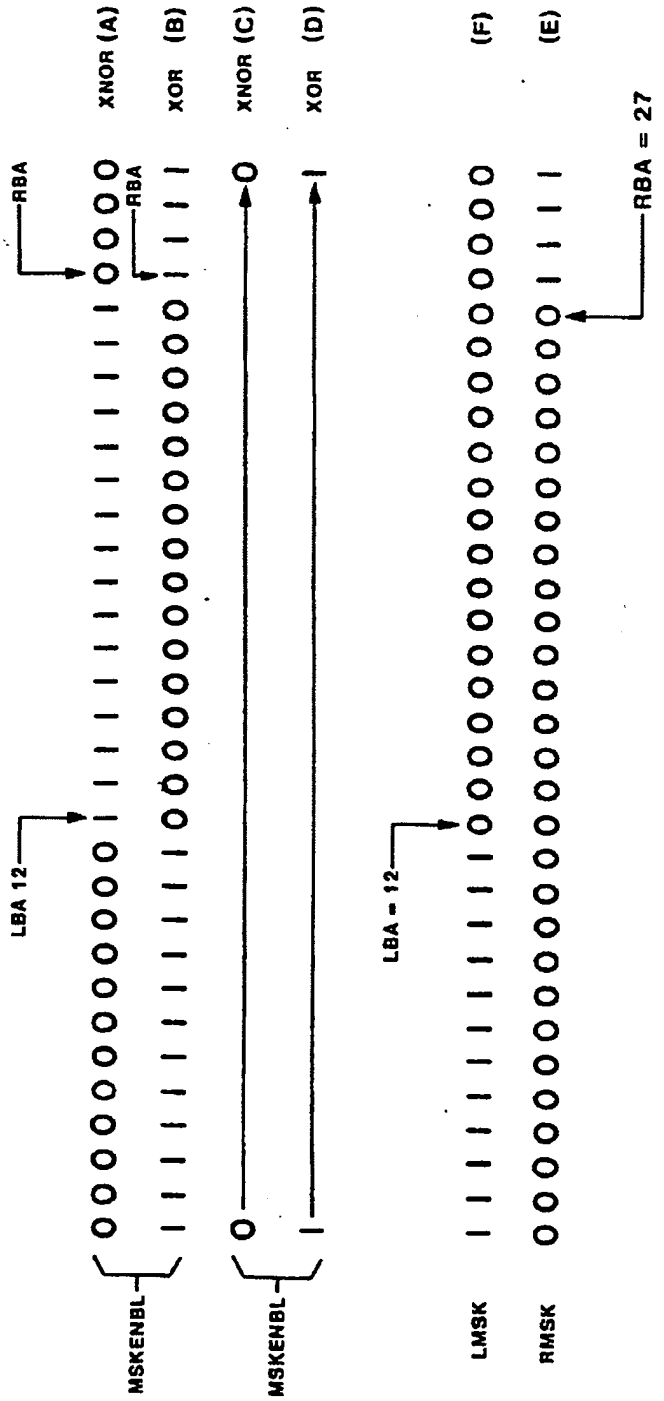


FIG. 231

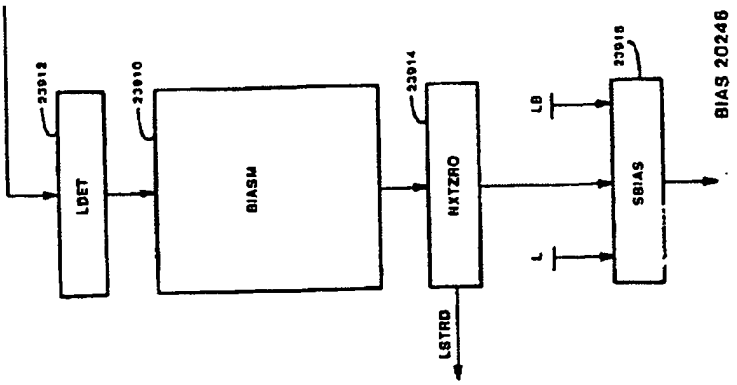


FIG. 239

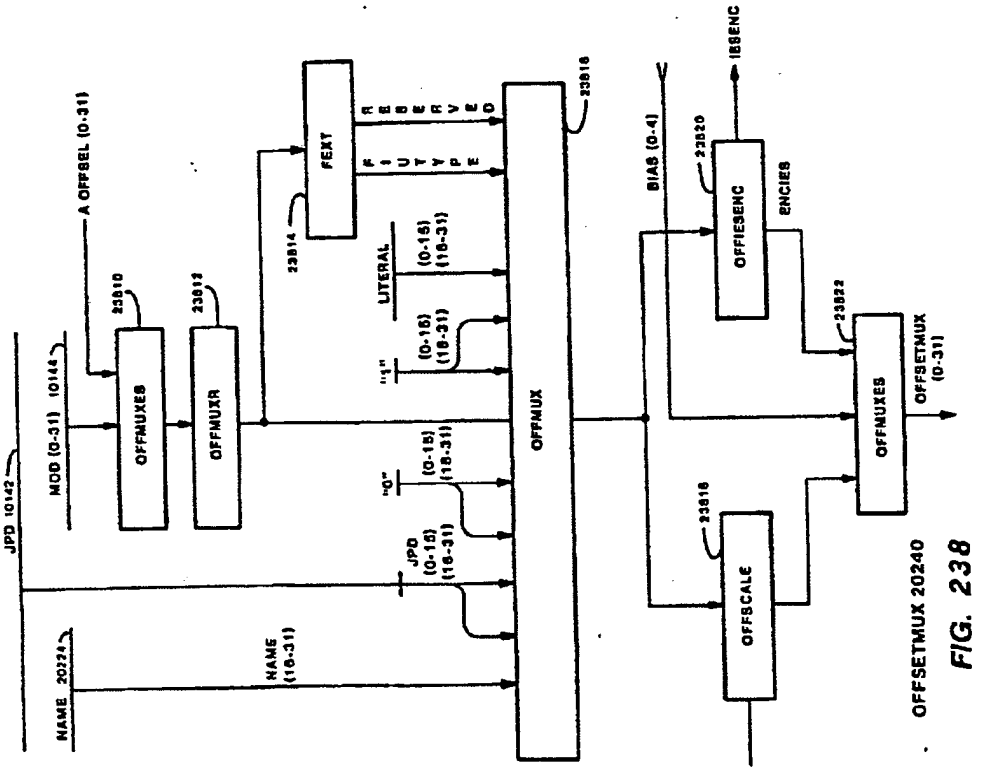
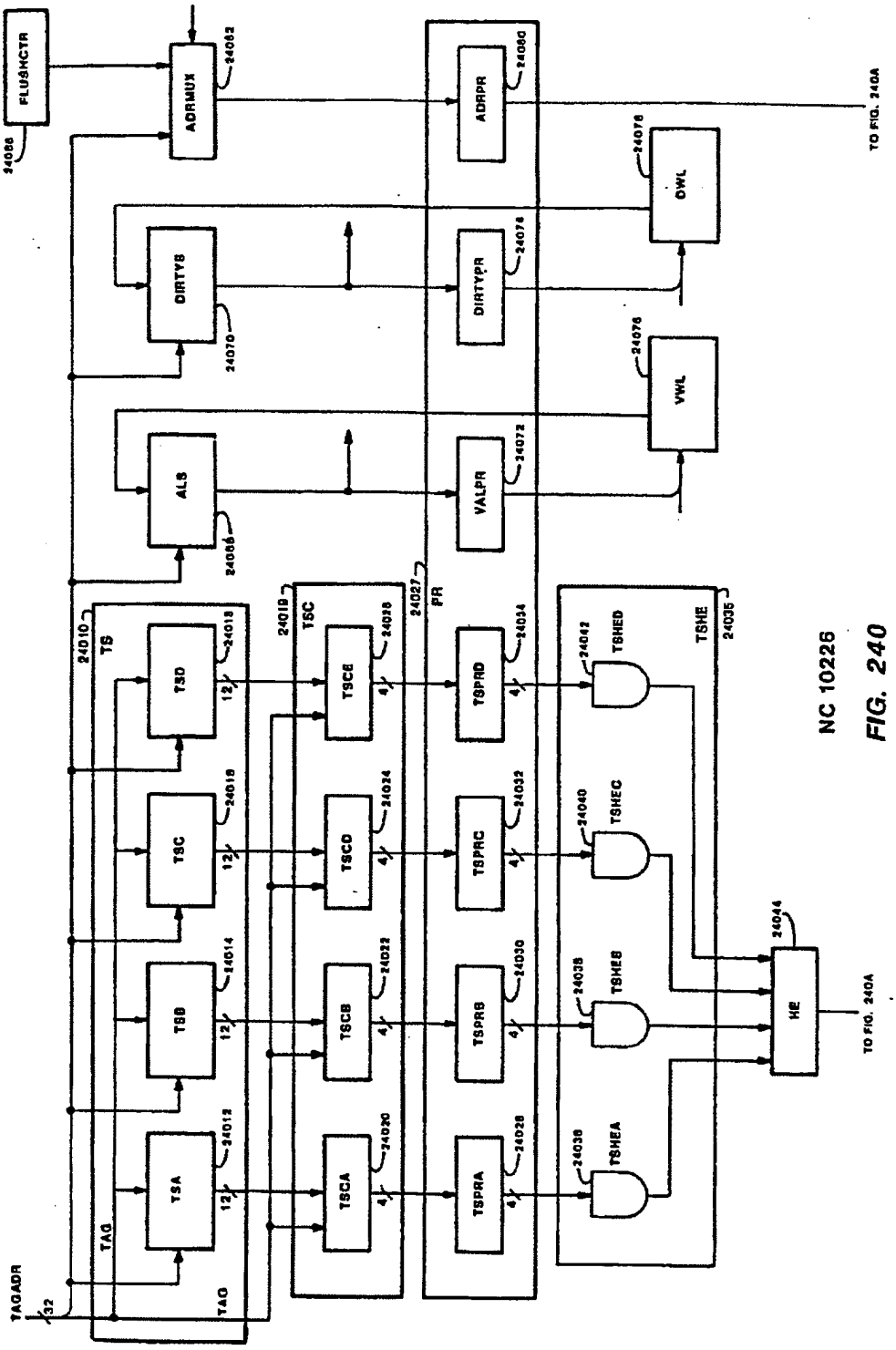
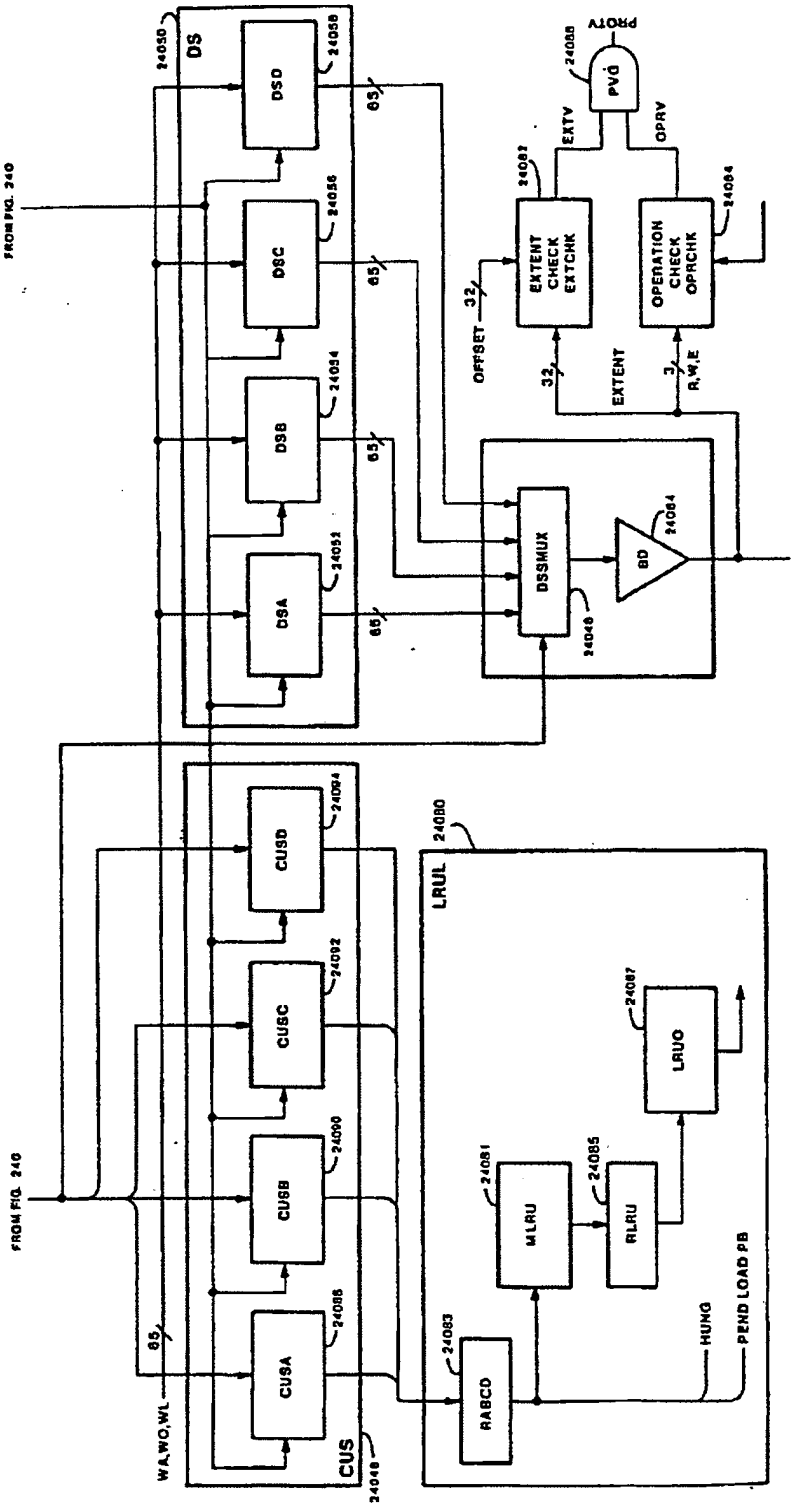


FIG. 238



NC 10226

FIG. 240



NC 10226

FIG. 240A

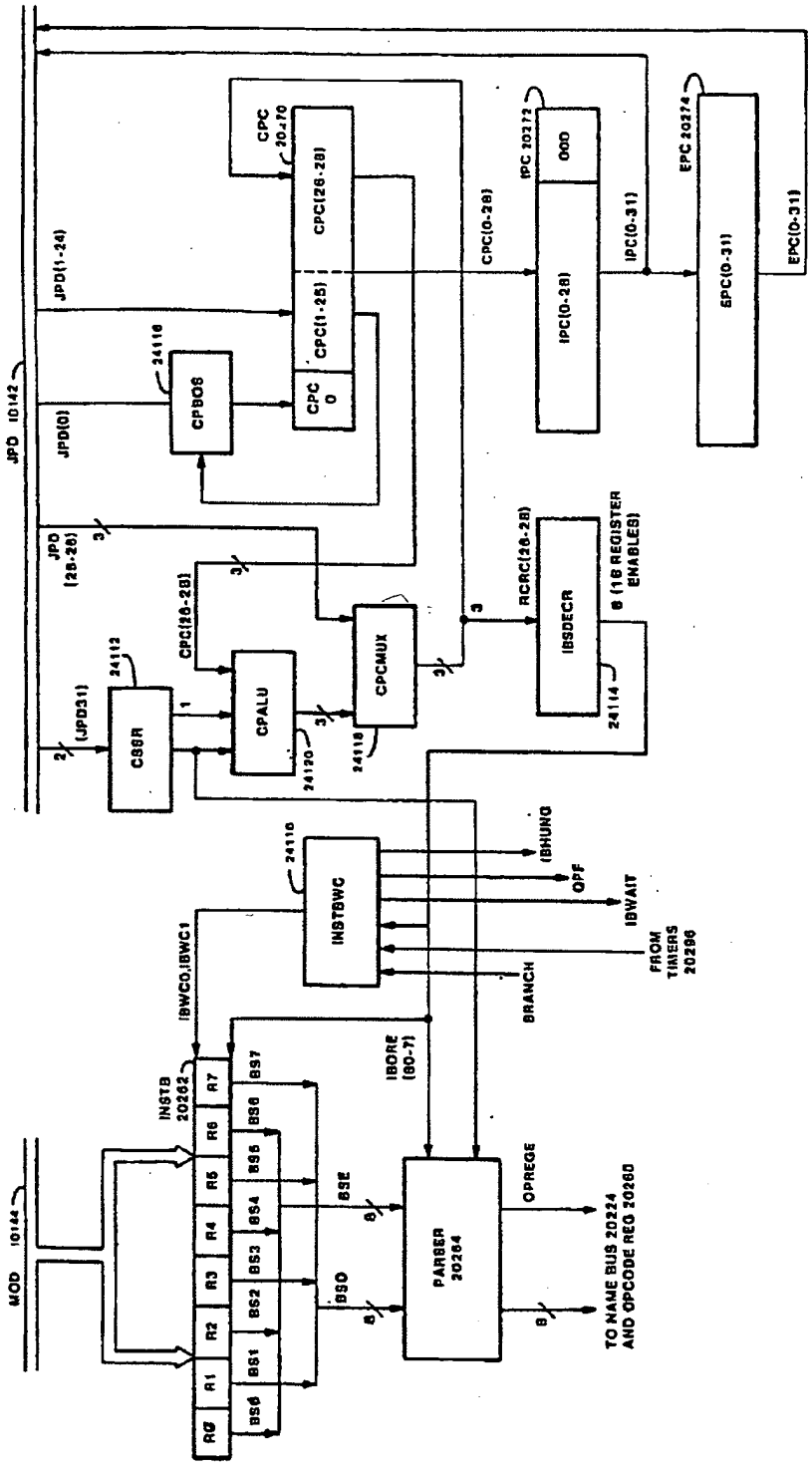


FIG. 241

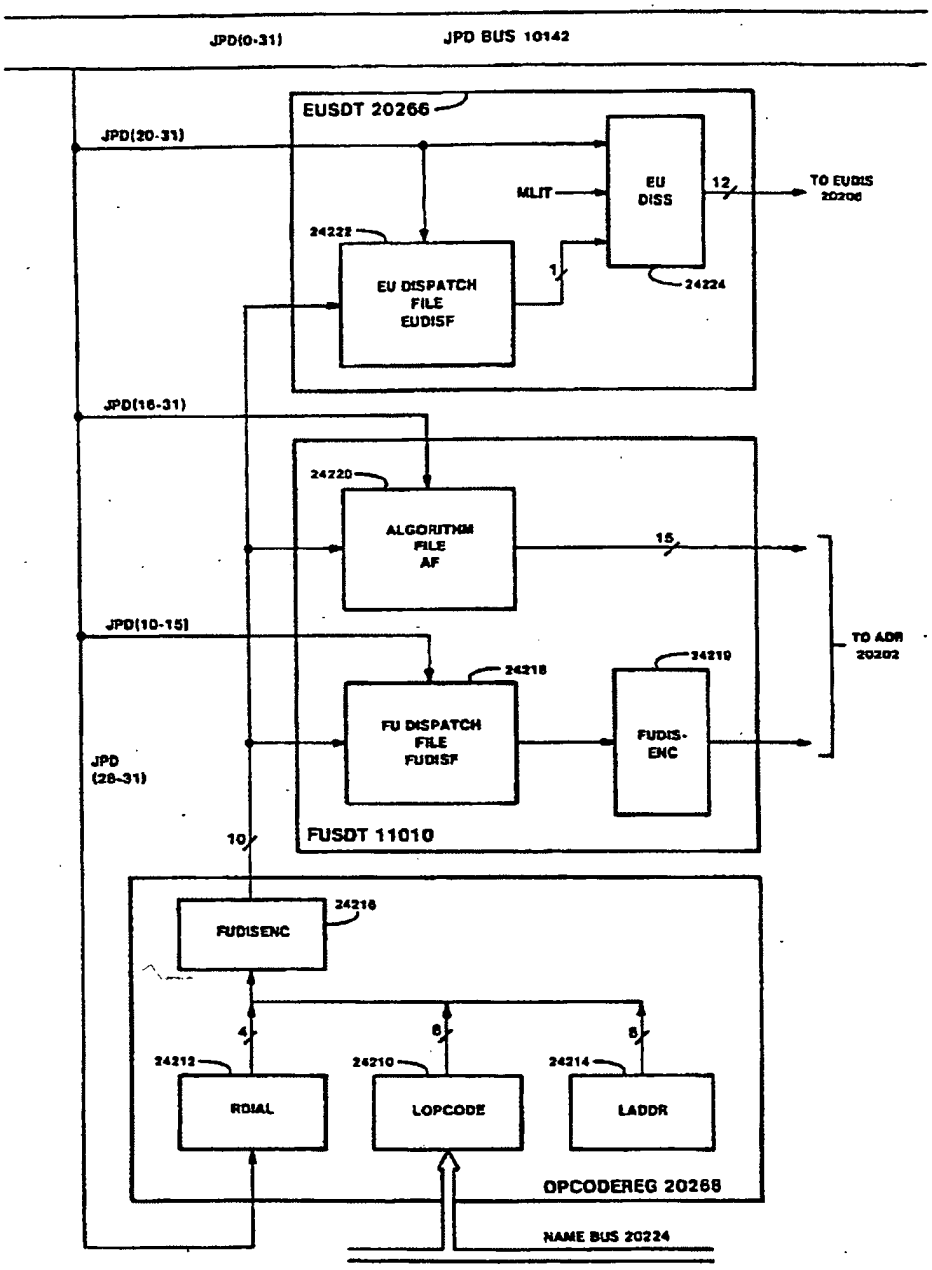


FIG. 242

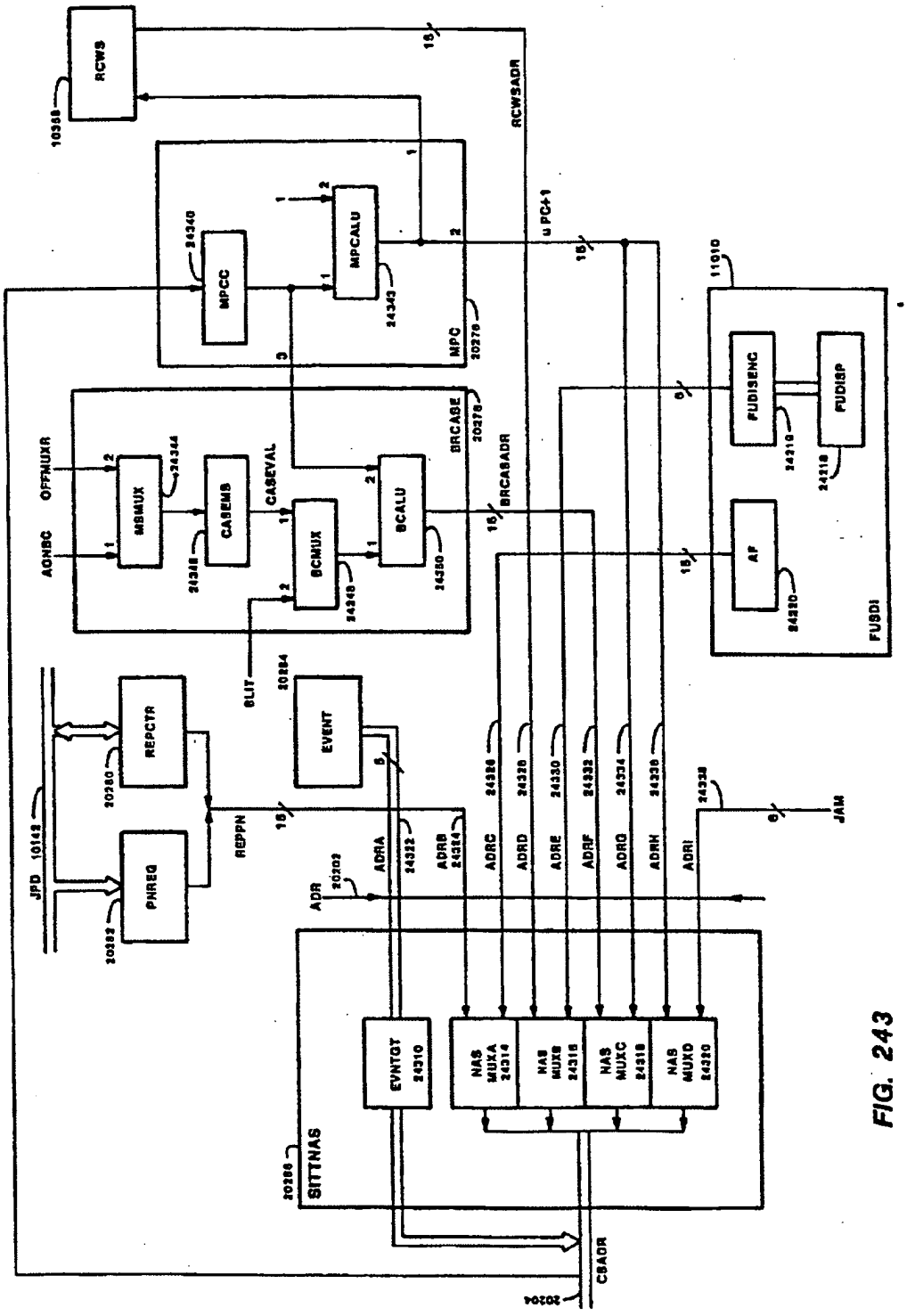


FIG. 243

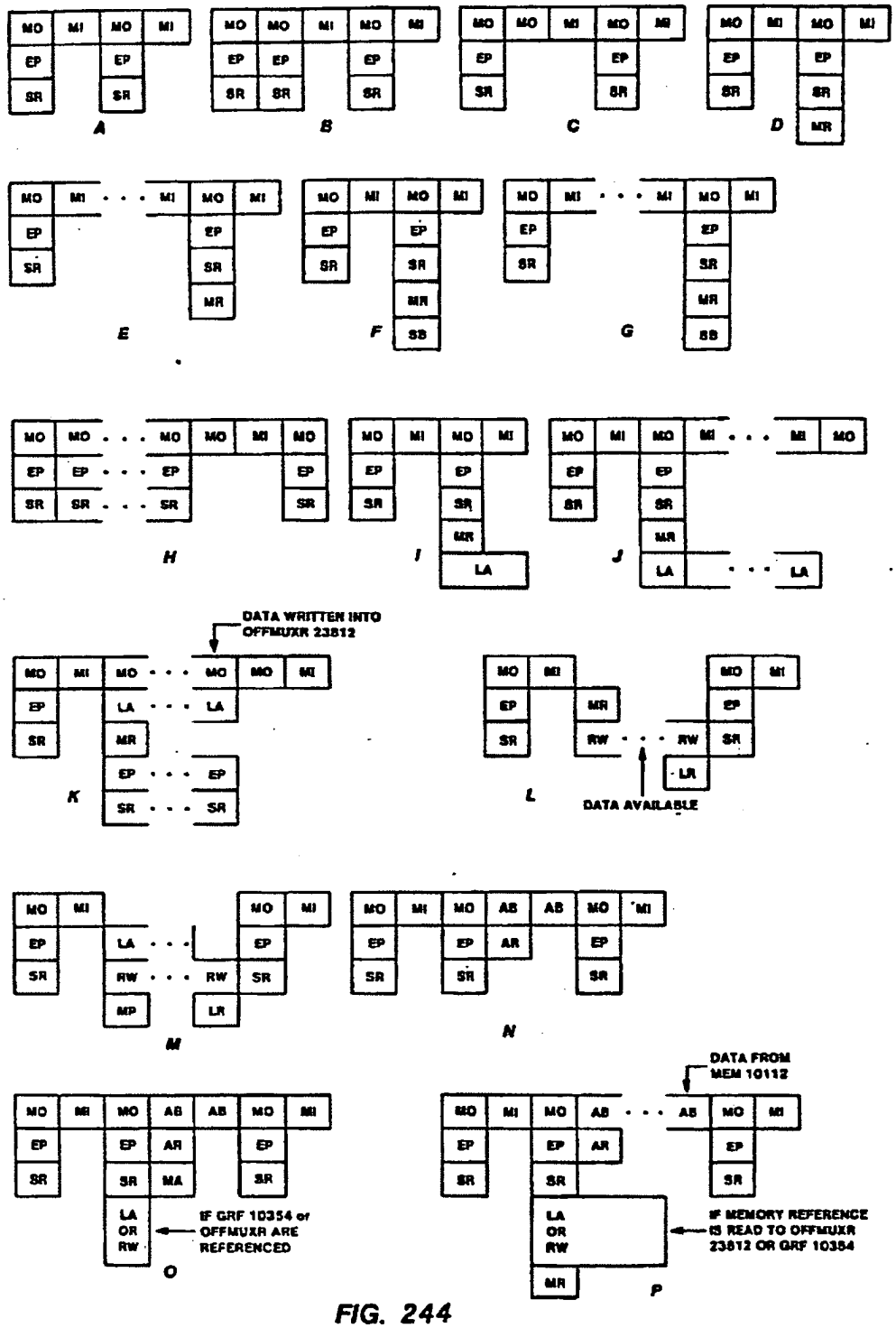


FIG. 244

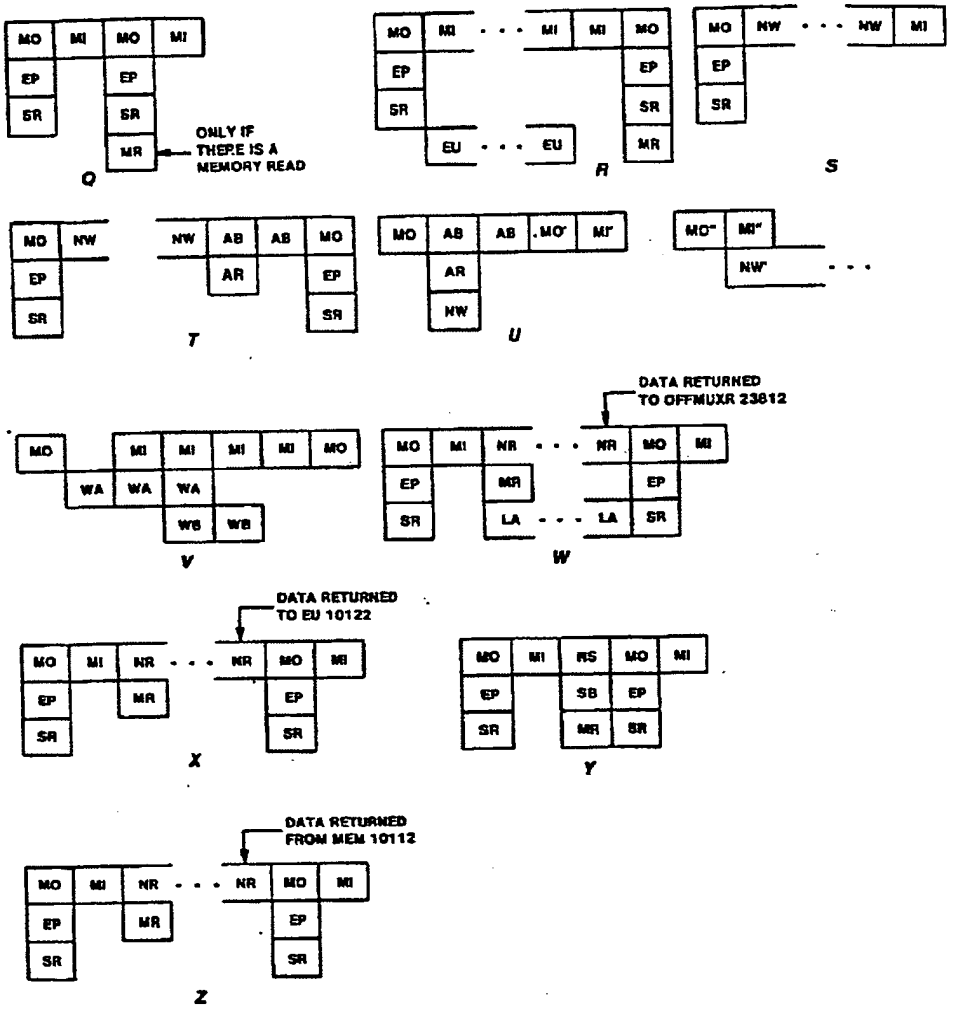


FIG. 244A

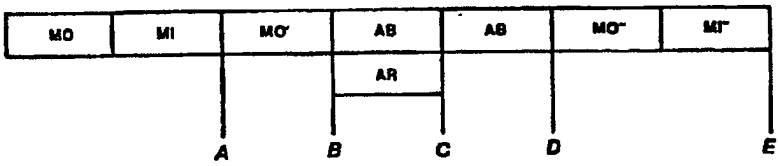


FIG. 245

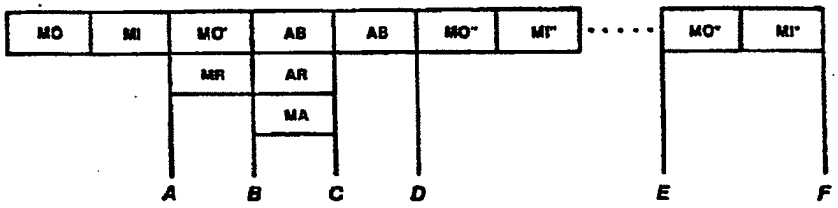


FIG. 246

PRIORITY LEVEL	EVENT	MASKED BY
0	E-UNIT STACK OVERFLOW	NONE
1	FATAL MEMORY ERROR	NONE
2	POWER FAIL	I
3	F-BOX STACK OVERFLOW	M,T,I
4	ILLEGAL E-UNIT DISPATCH (GATE FAULT)	NONE
5	STOREBACK EXCEPTION	MCWD
6	NAME TRACE TRAP	T,I
7	LOGICAL READ TRACE TRAP	T,I AND DES
8	LOGICAL WRITE TRACE TRAP	T,I AND DES
9	UID READ DEREFERENCE TRAP	DES
10	UID WRITE DEREFERENCE TRAP	DES
11	PROTECTION CACHE MISS	NONE
12	PROTECTION VIOLATION	MCWD
13	PAGE CROSSING INTERRUPT	NONE
14	LAT	NONE
15	WRITE LAT	NONE
16	MEMORY REFERENCE REPEAT	NONE
17	EGG TIMER OVERFLOW	A,M,T,I
18	E-BOX STACK UNDERFLOW	A,M,T,I
19	NON-FATAL MEMORY ERROR	NONE
20	INTERVAL TIMER OVEREFLOW	NONE
21	IPM INTERRUPT	NONE
22	S-OP TRACE TRAP	NONE
23	ILLEGAL S-OP	NONE
24	MICROINSTRUCTION TRACE TRAP	NONE
25	NON-PRESENT MICROINSTRUCTION	NONE
26	INSTRUCTION PREFETCH IS HUNG	NONE
27	F-BOX STACK UNDERFLOW	NONE
28	MICROINSTRUCTION BREAK POINT TRACE TRAP	T,I AND MCWD
29	MISS ON NAME CACHE LOAD OR READ REGISTER	NONE

FIG. 247

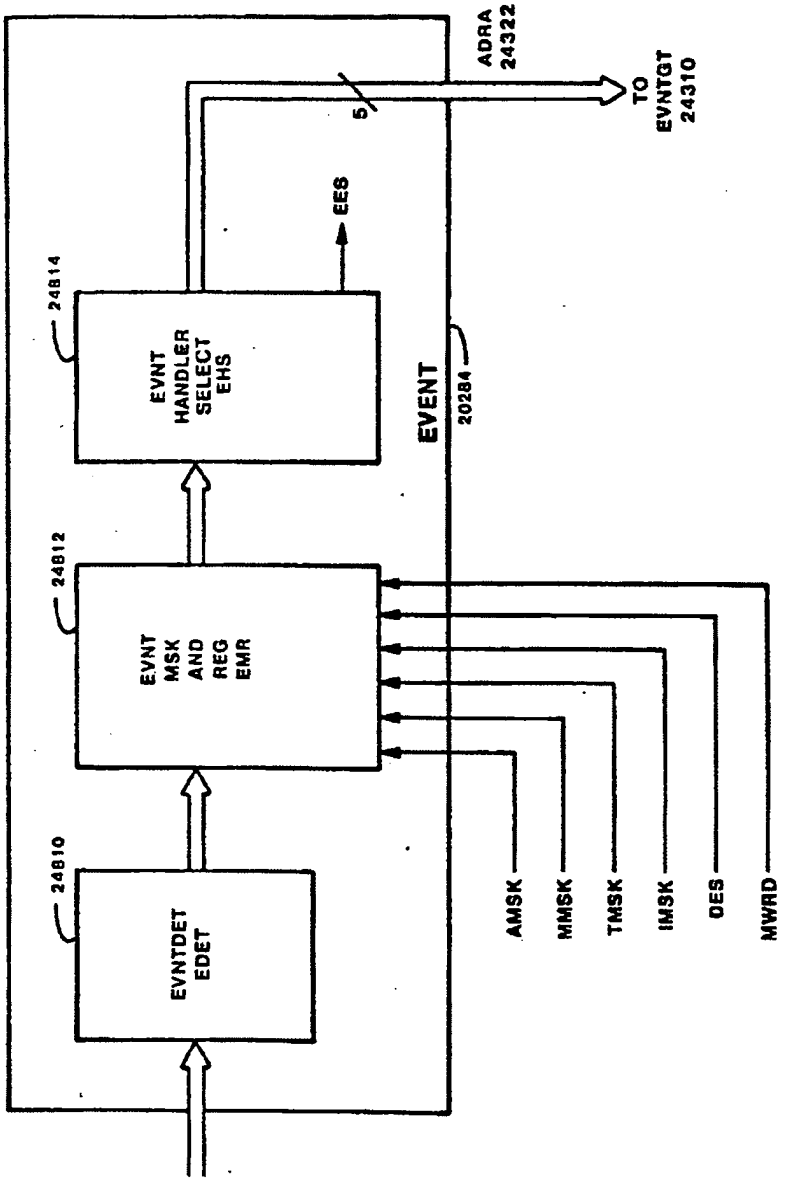


FIG. 248

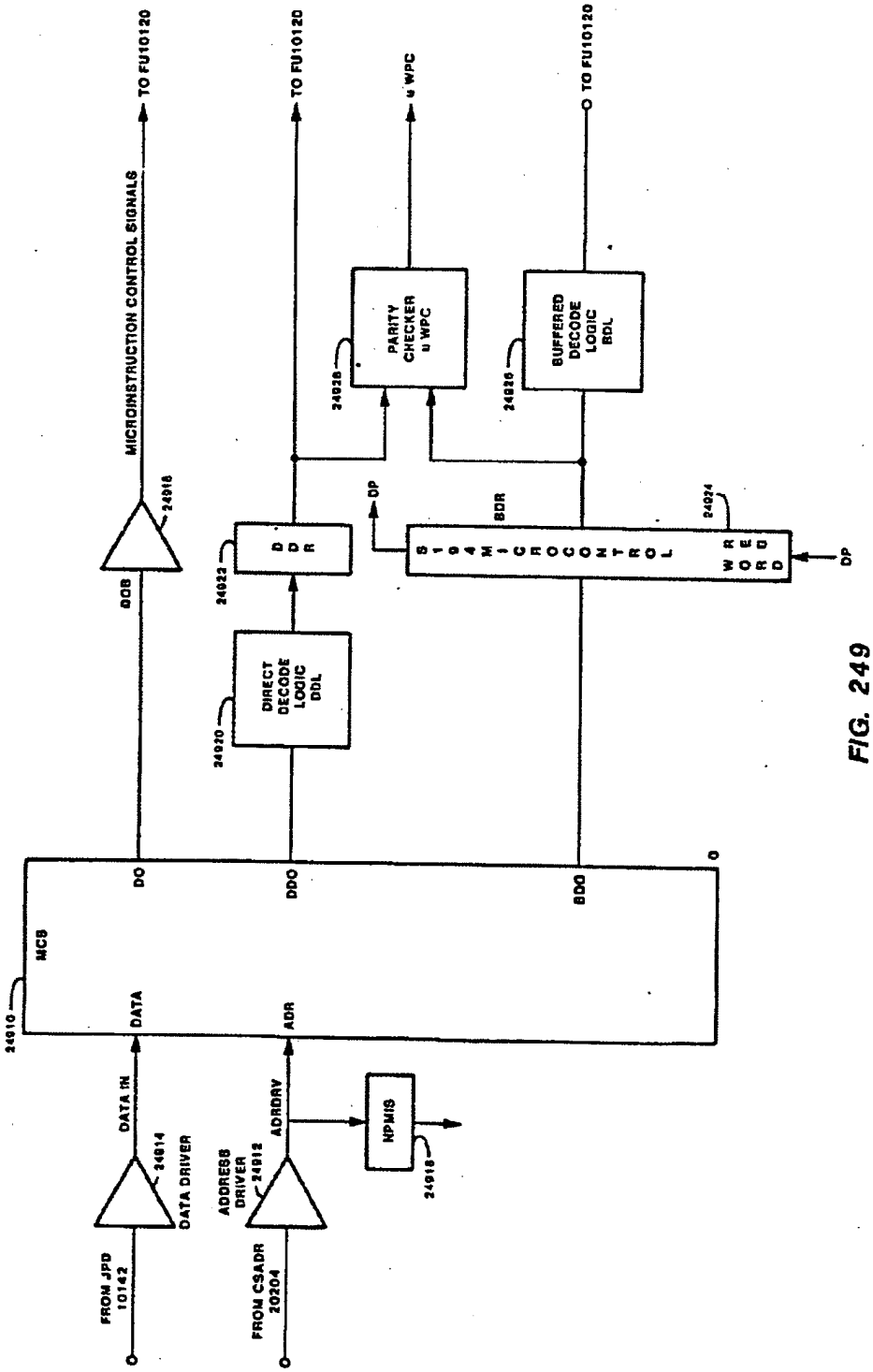


FIG. 249

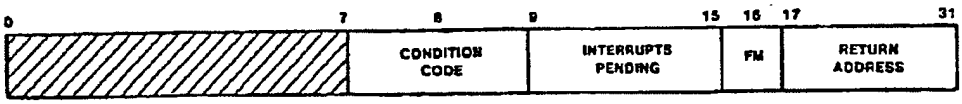


FIG. 251

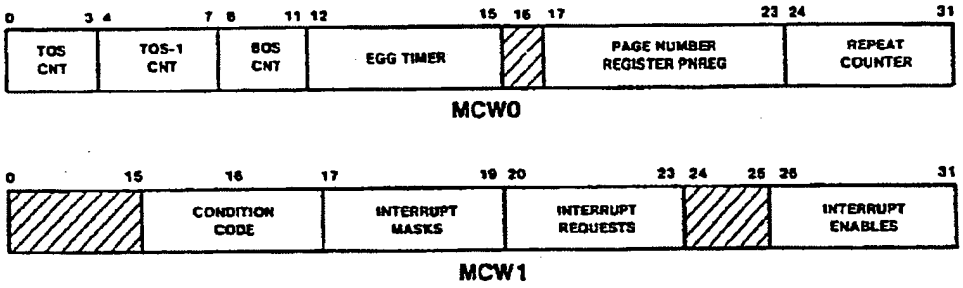


FIG. 252

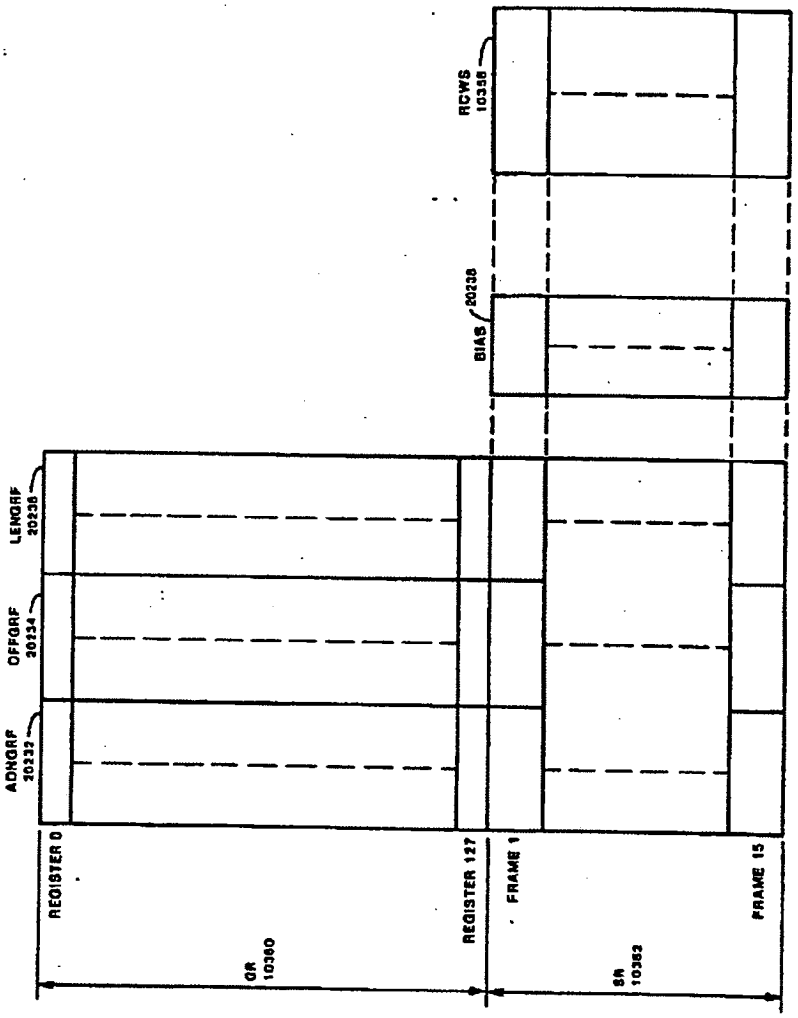


FIG. 253

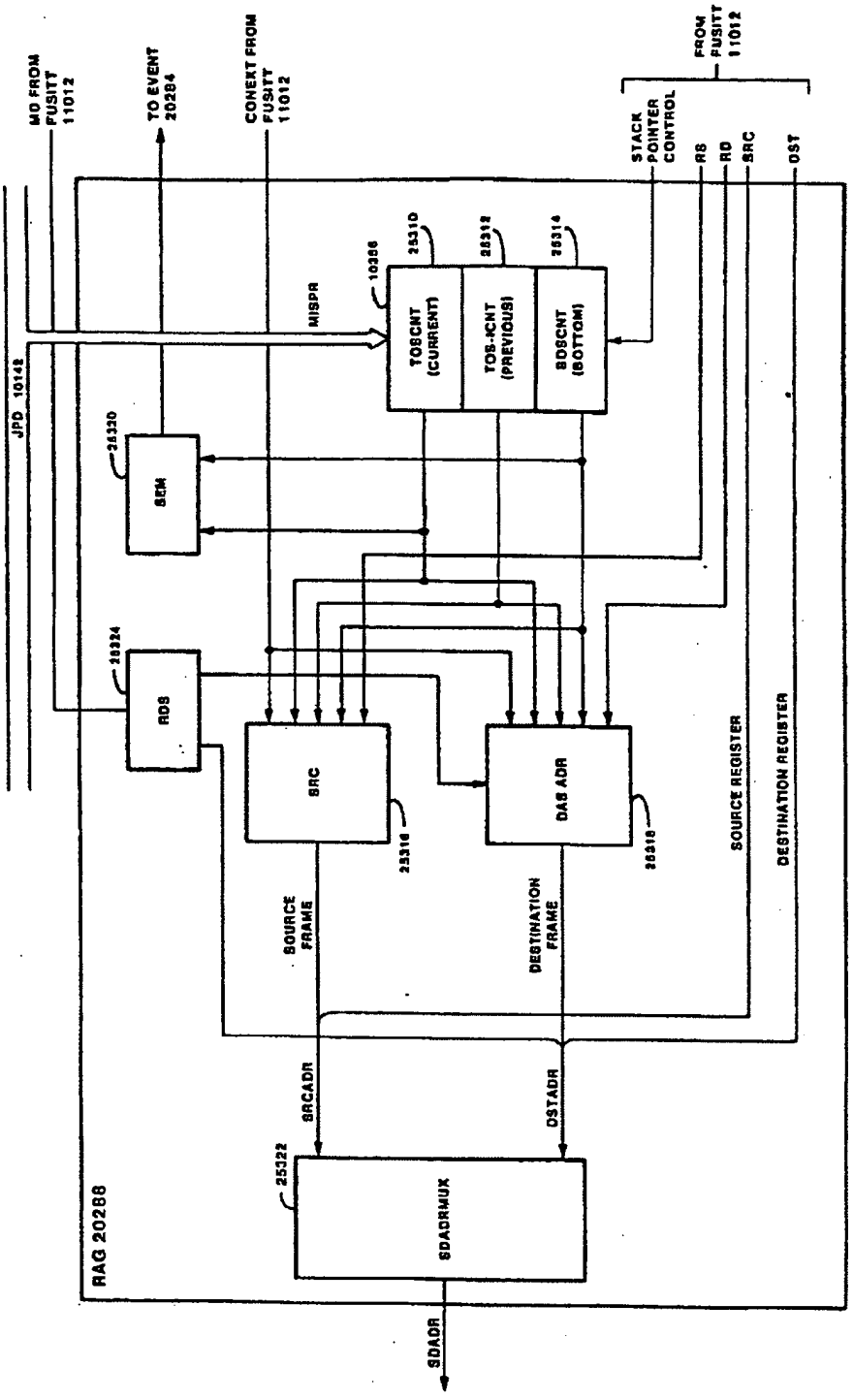


FIG. 253A

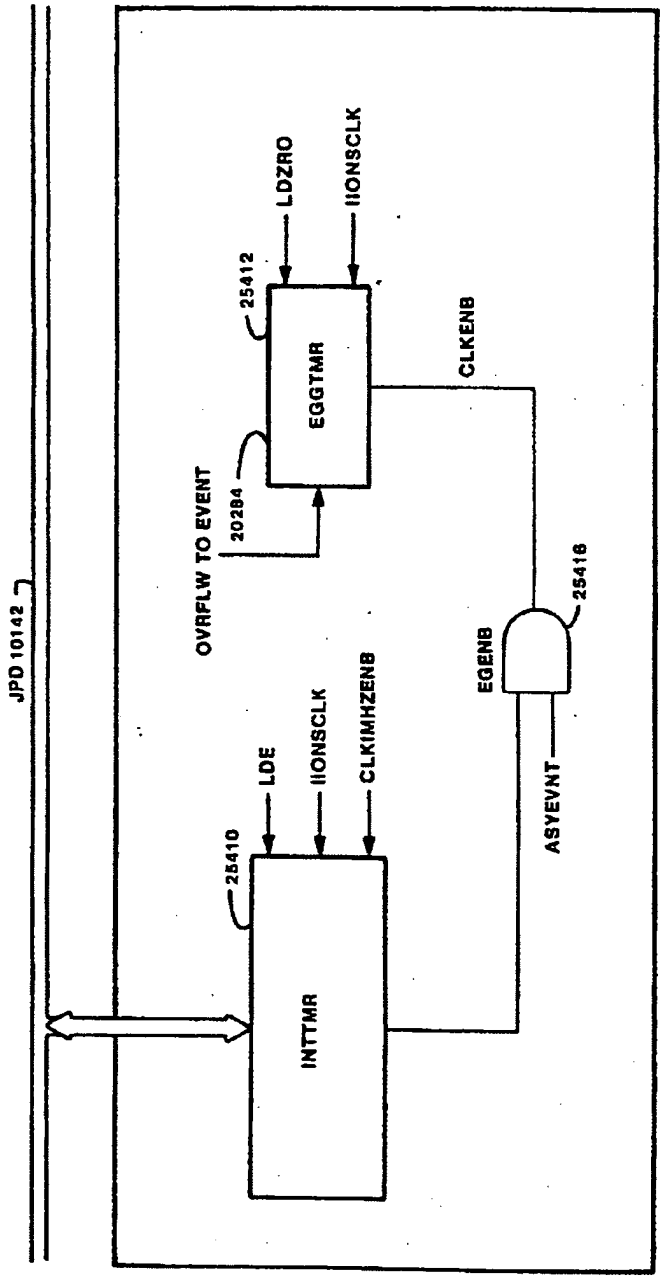


FIG. 254

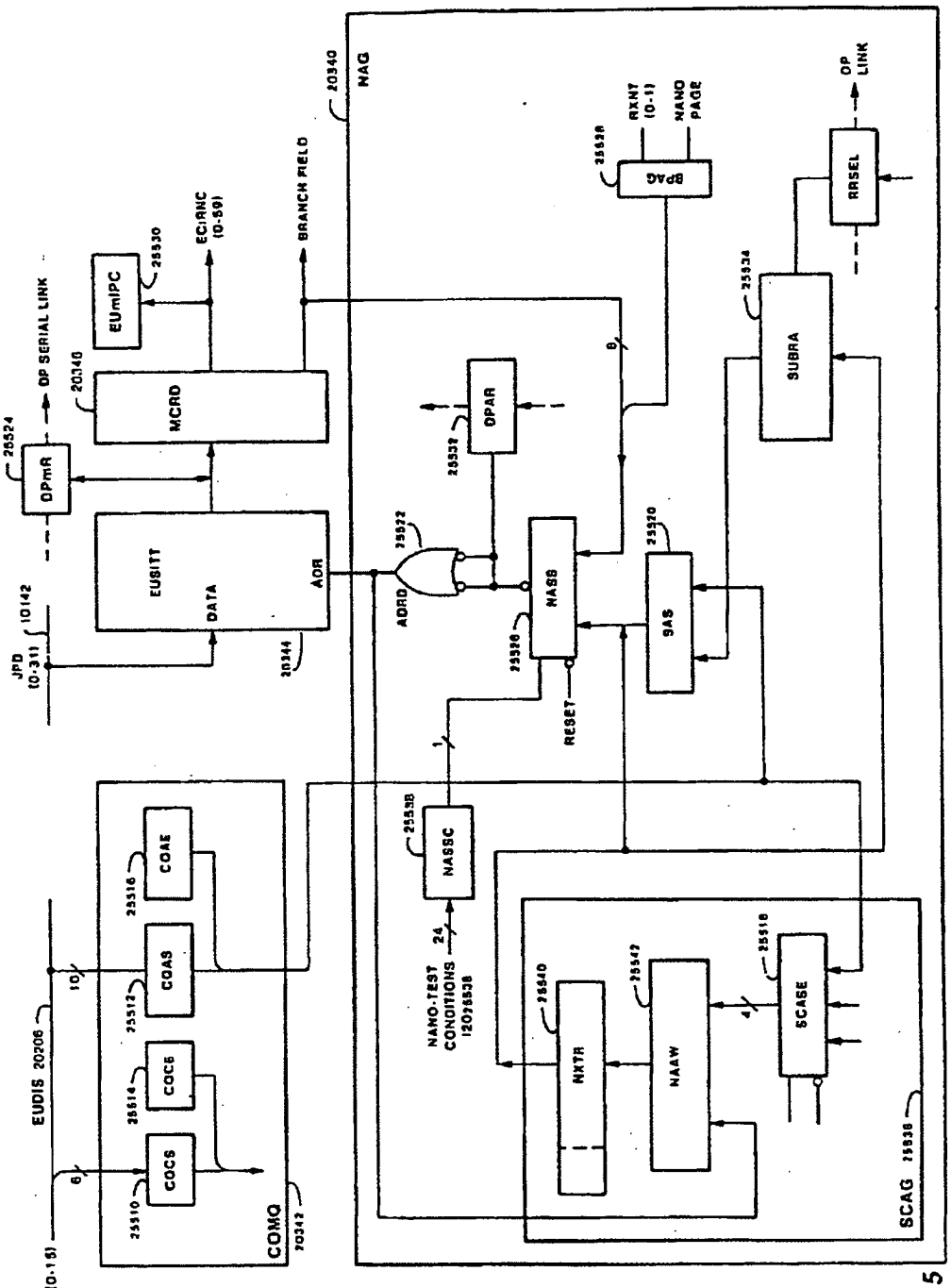


FIG. 255

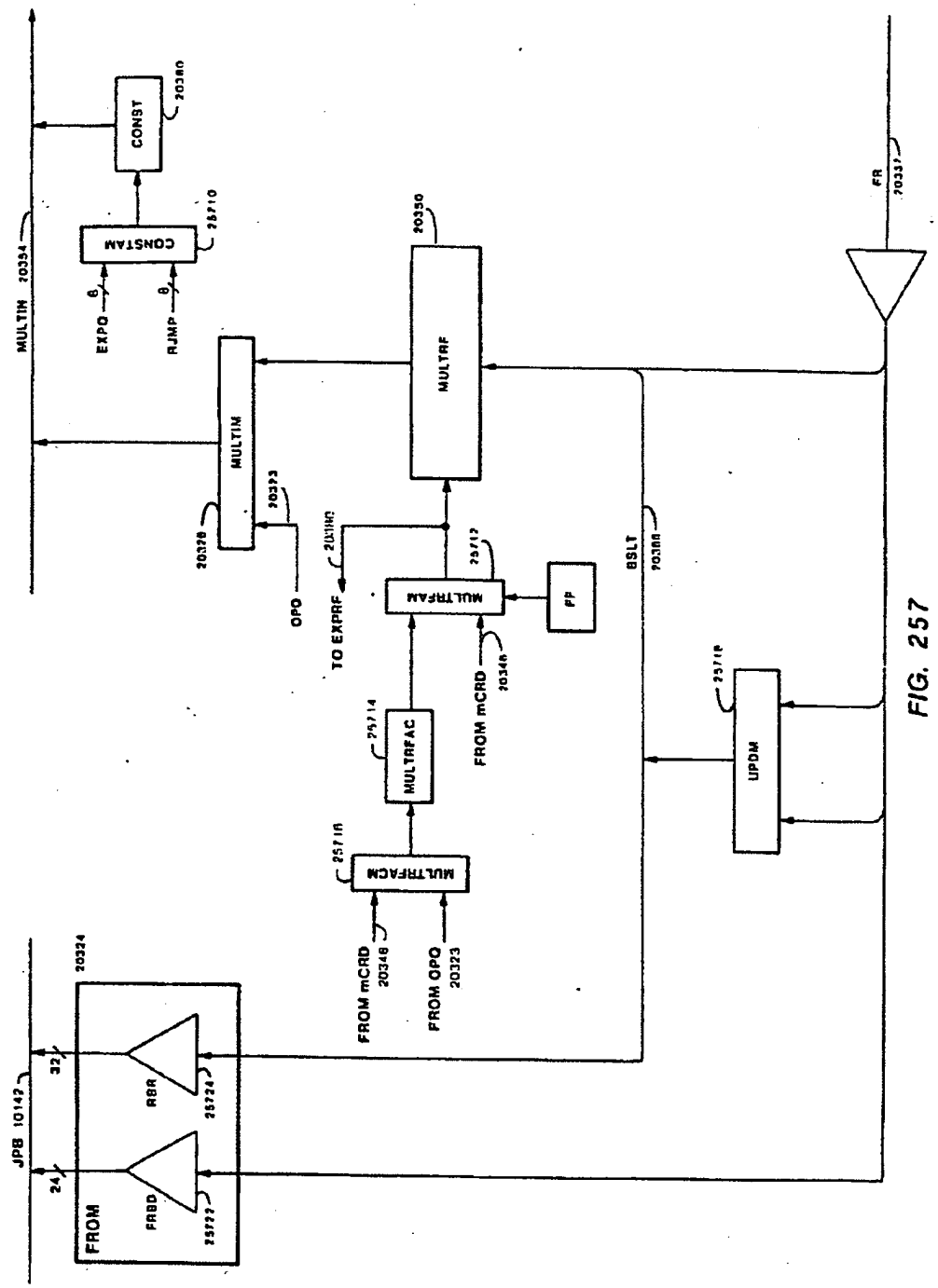


FIG. 257

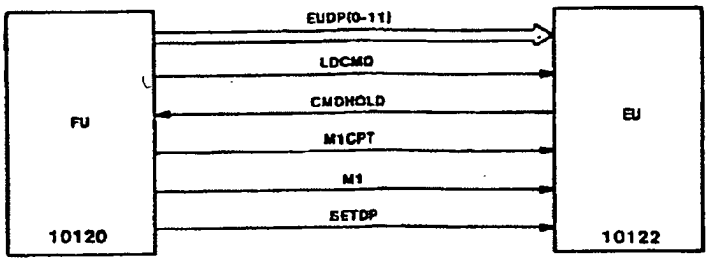


FIG. 260

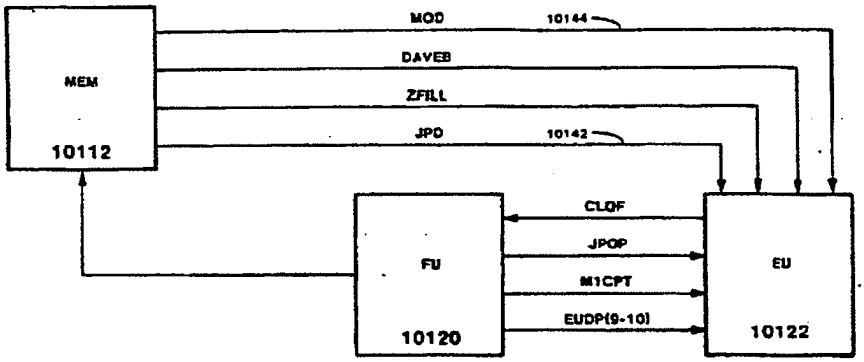


FIG. 261

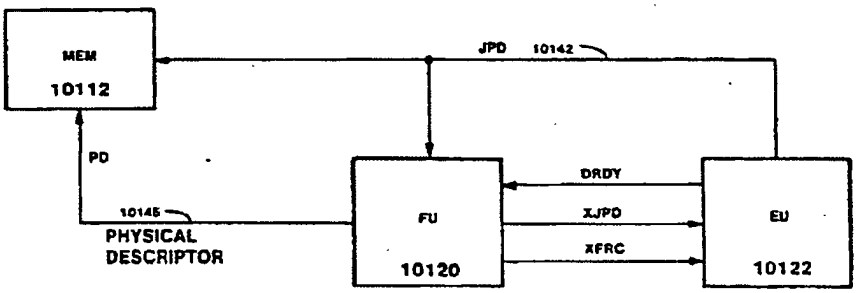


FIG. 262

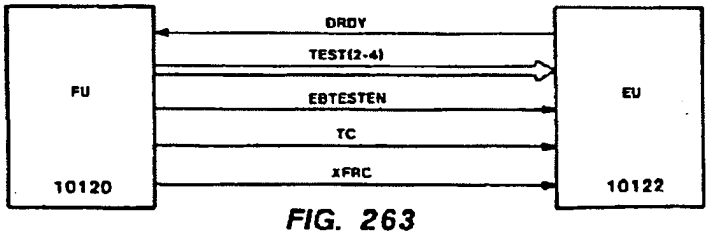


FIG. 263

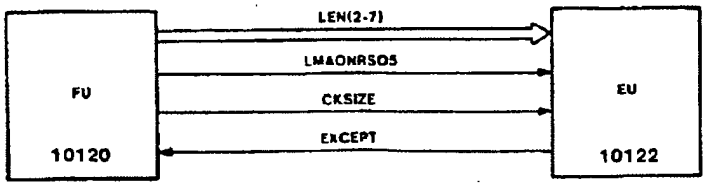


FIG. 264

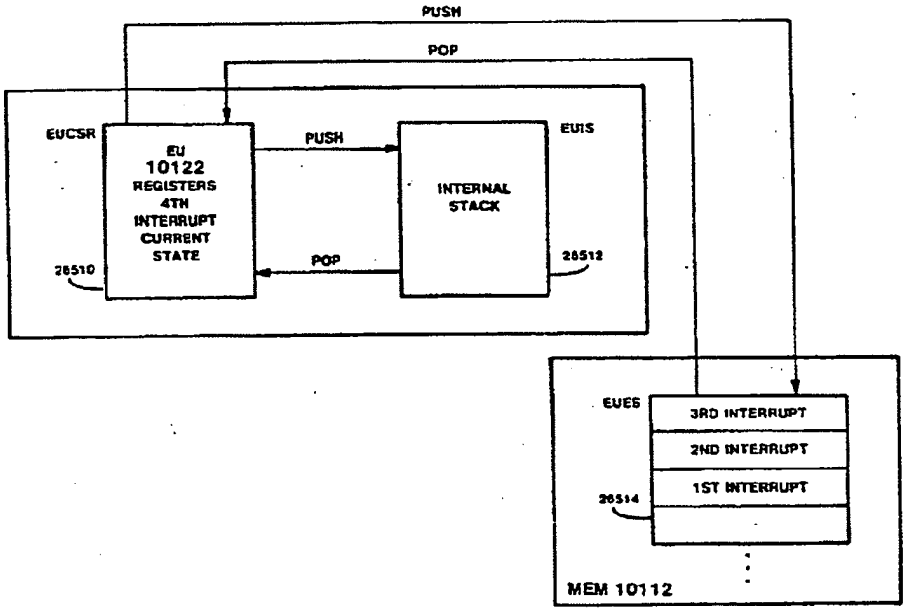


FIG. 265

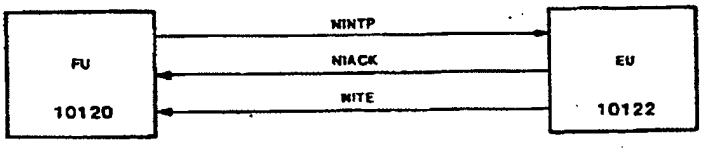


FIG. 266

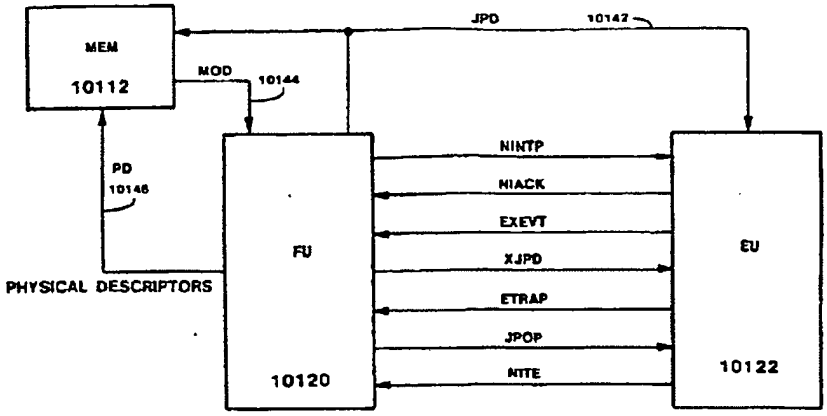


FIG. 267

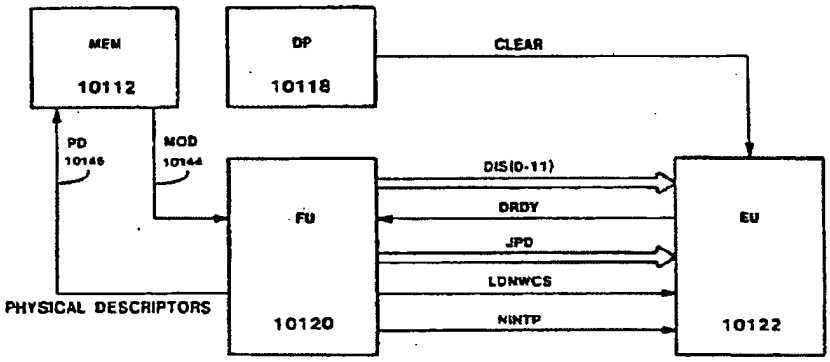


FIG. 268

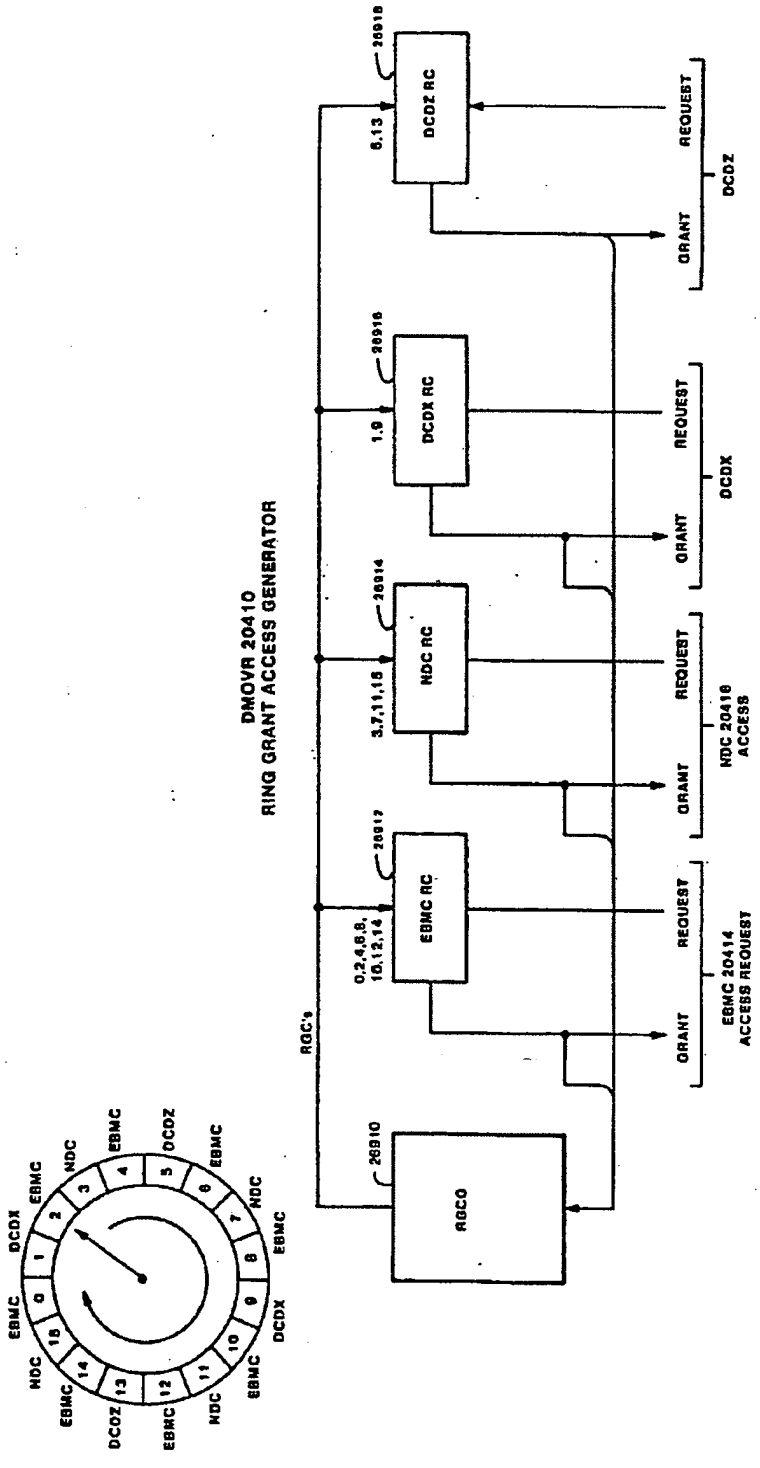


FIG 269

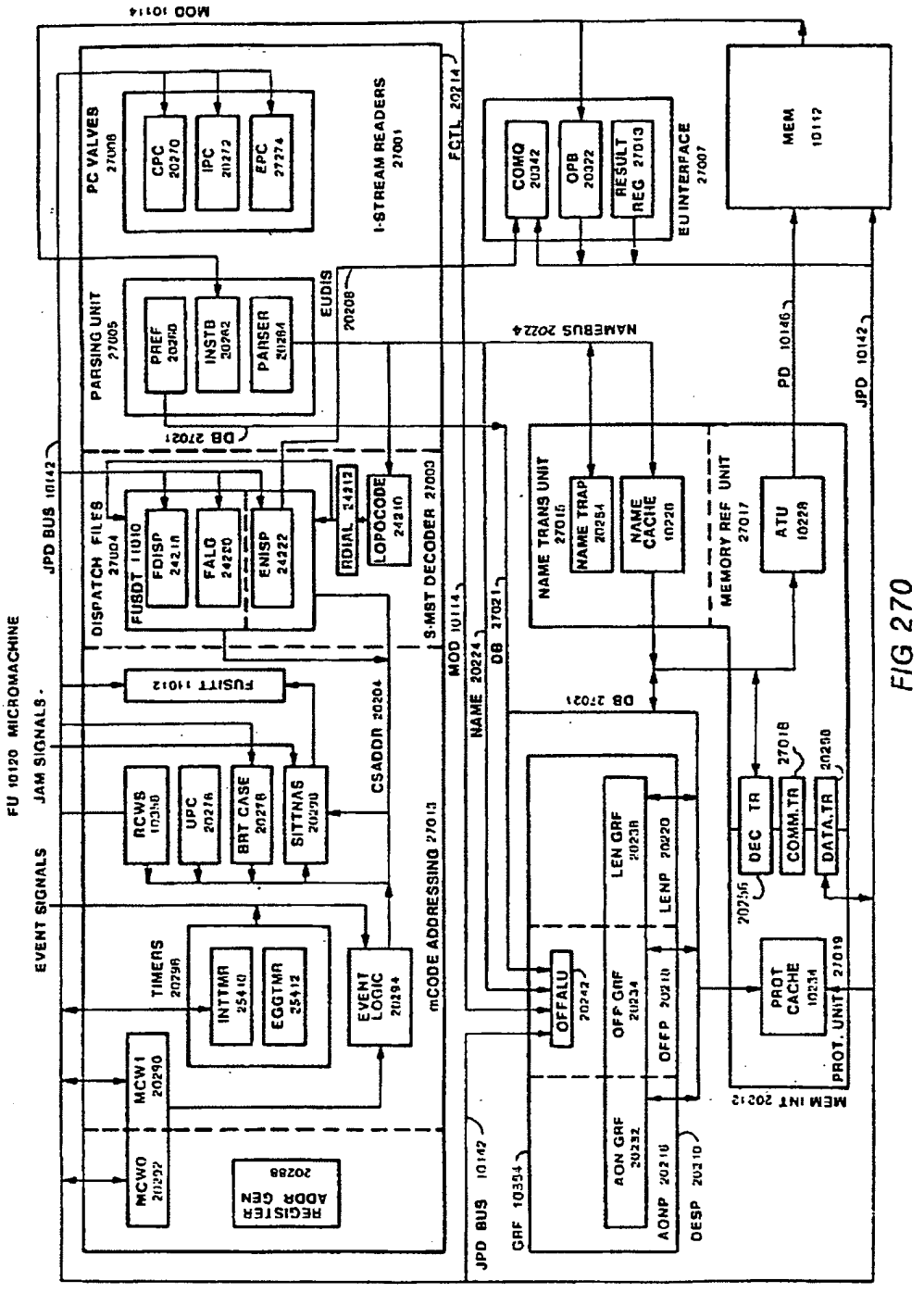


FIG 270

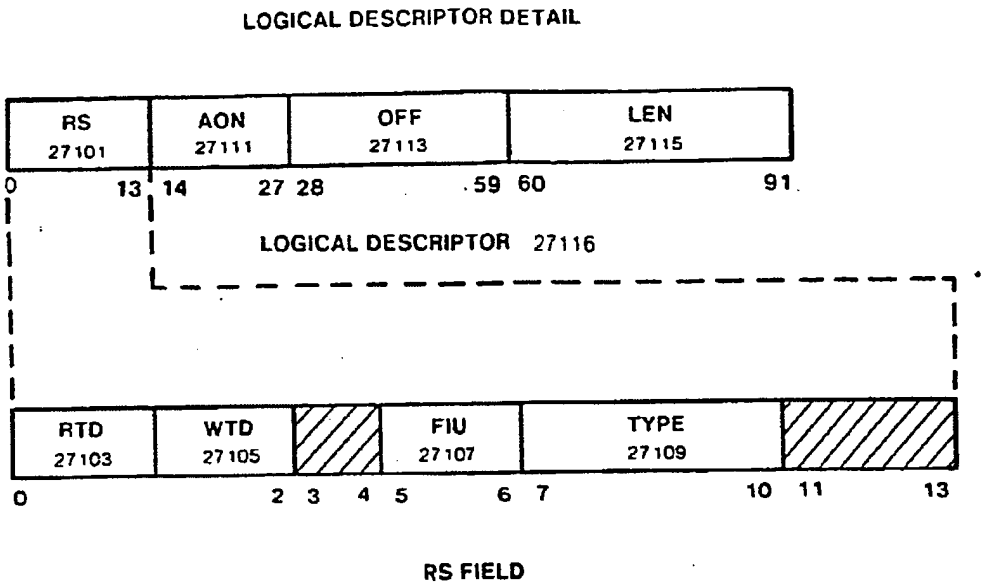


FIG. 271

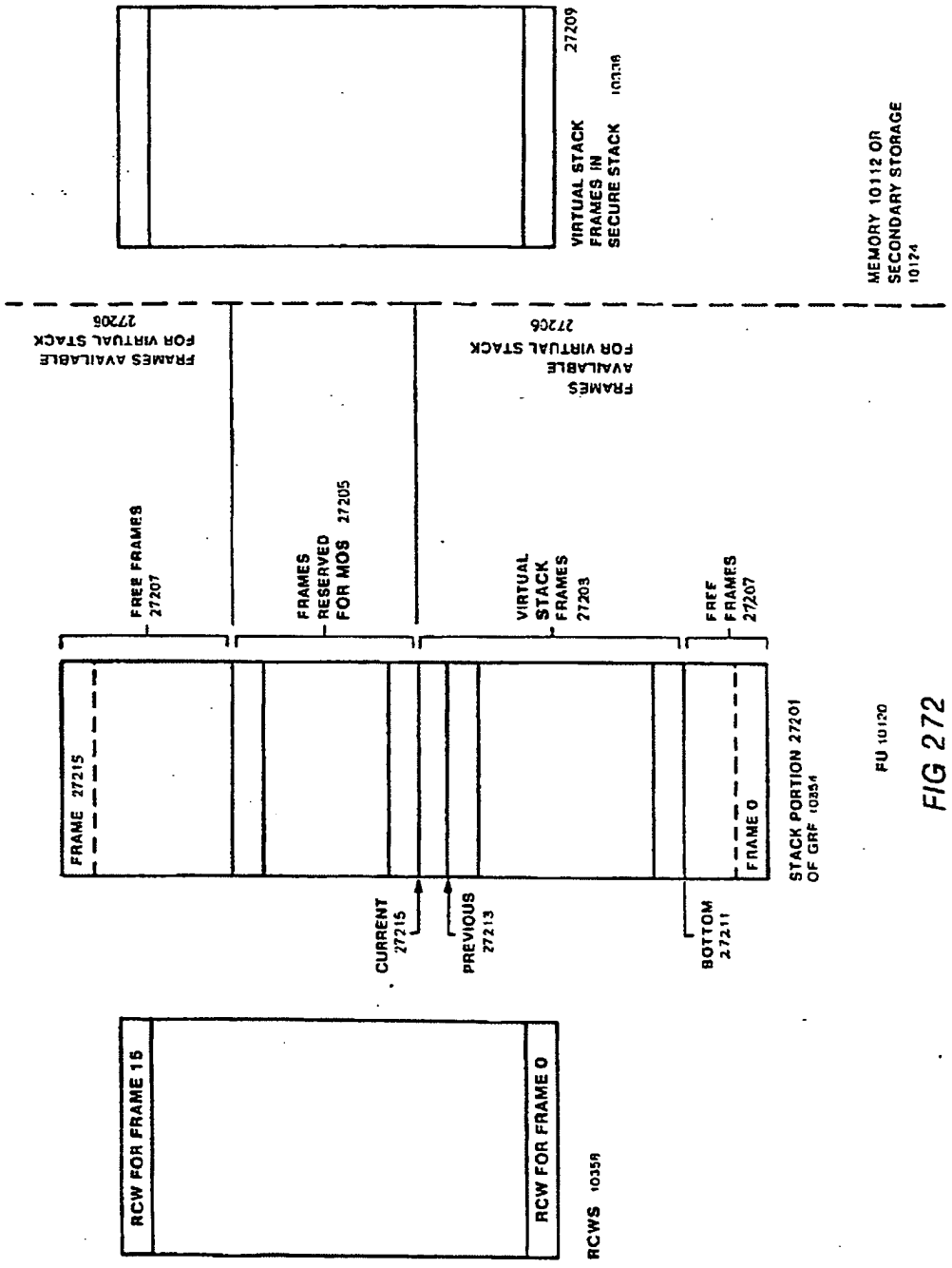


FIG 272

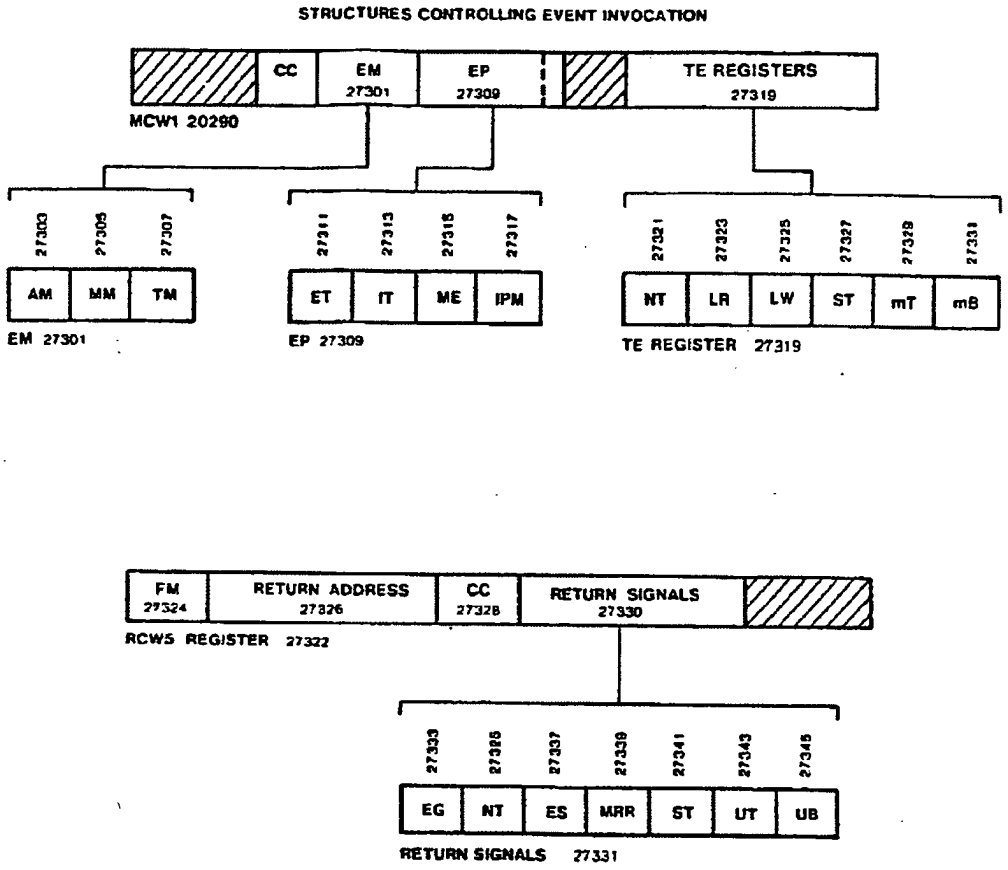


FIG. 273

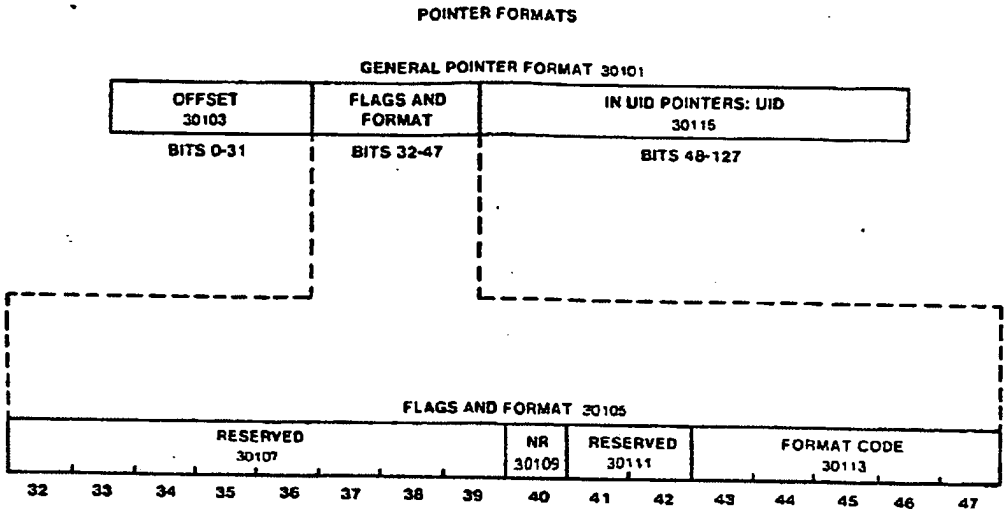


FIG. 301

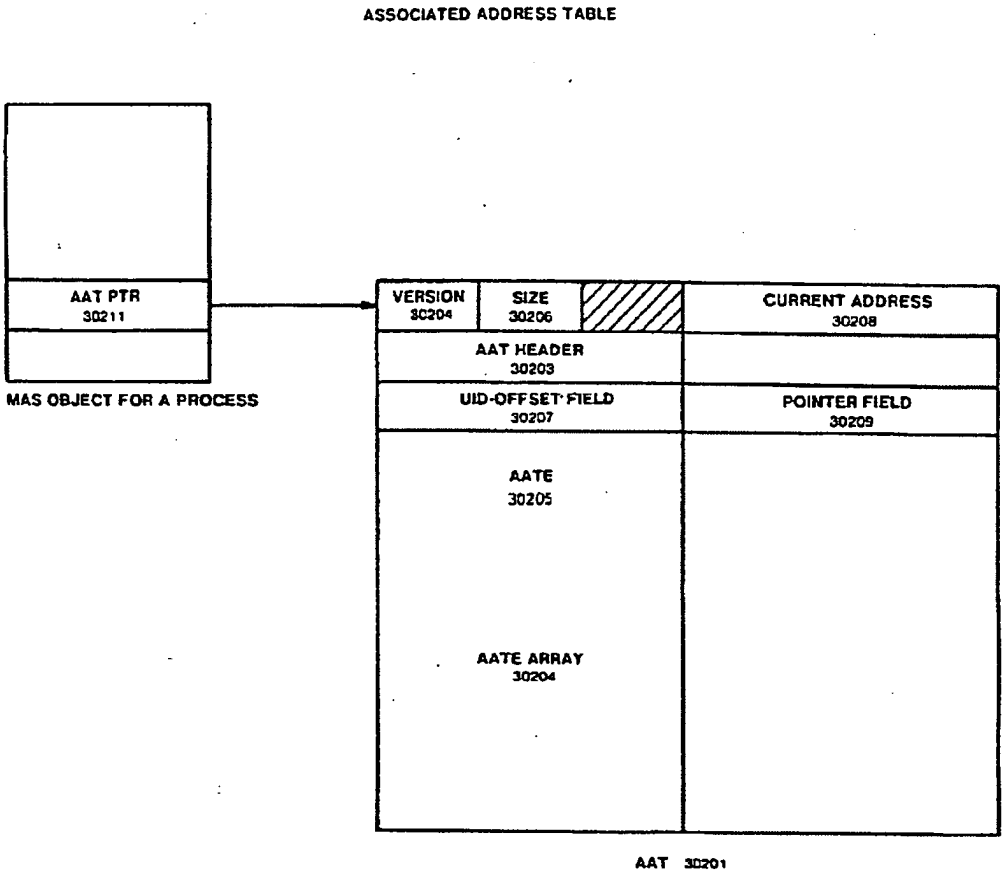


FIG. 302

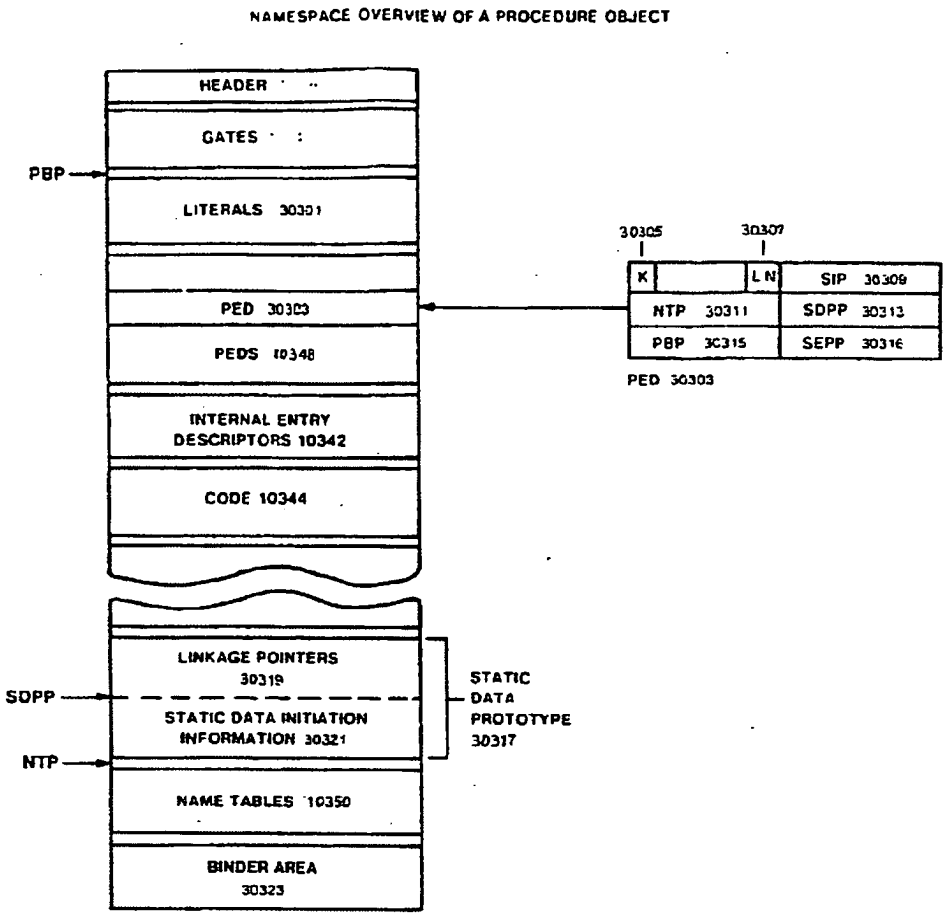


FIG. 303

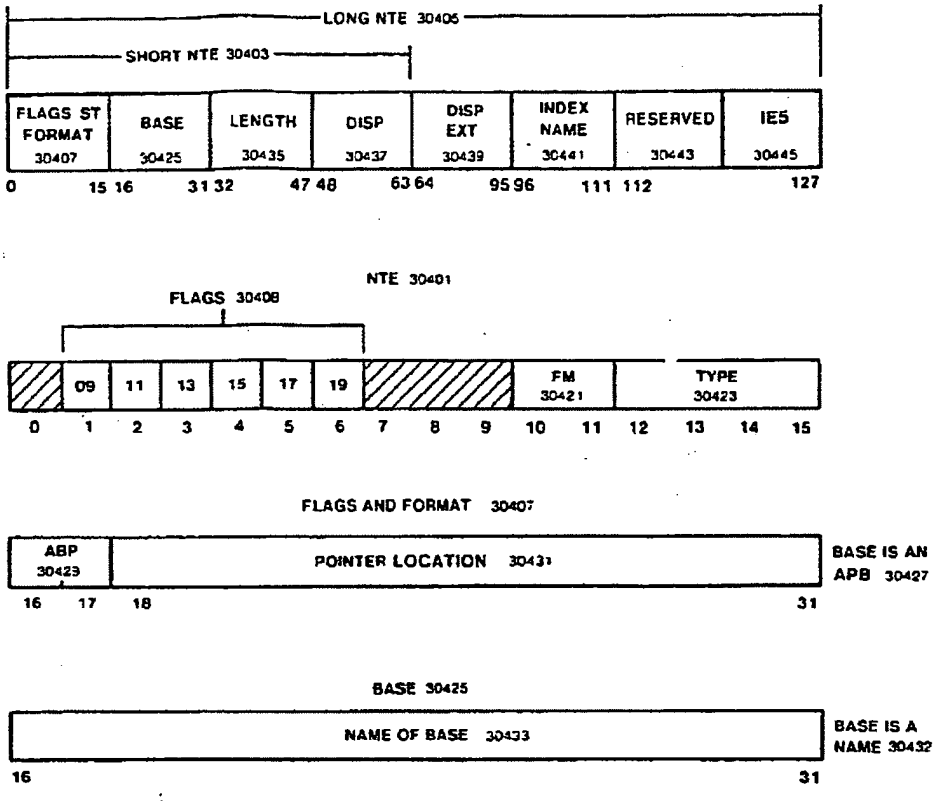


FIG. 304

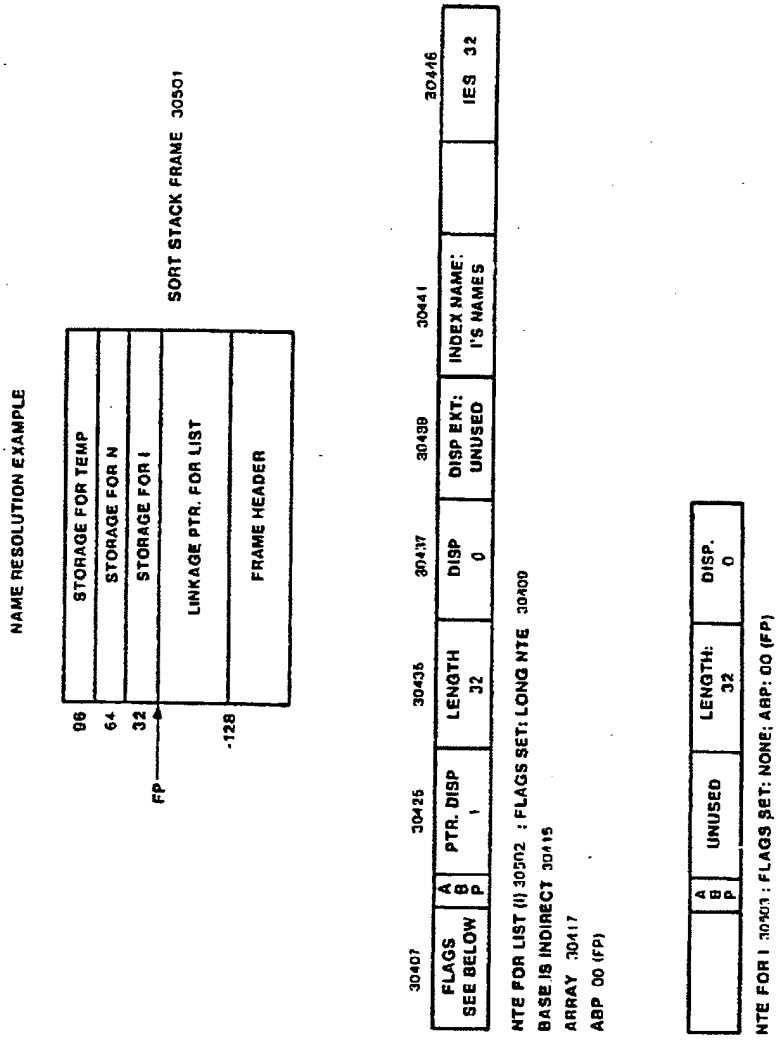


FIG. 305

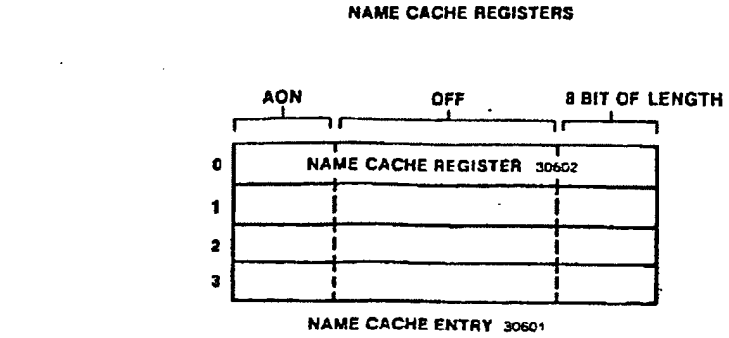


FIG. 306

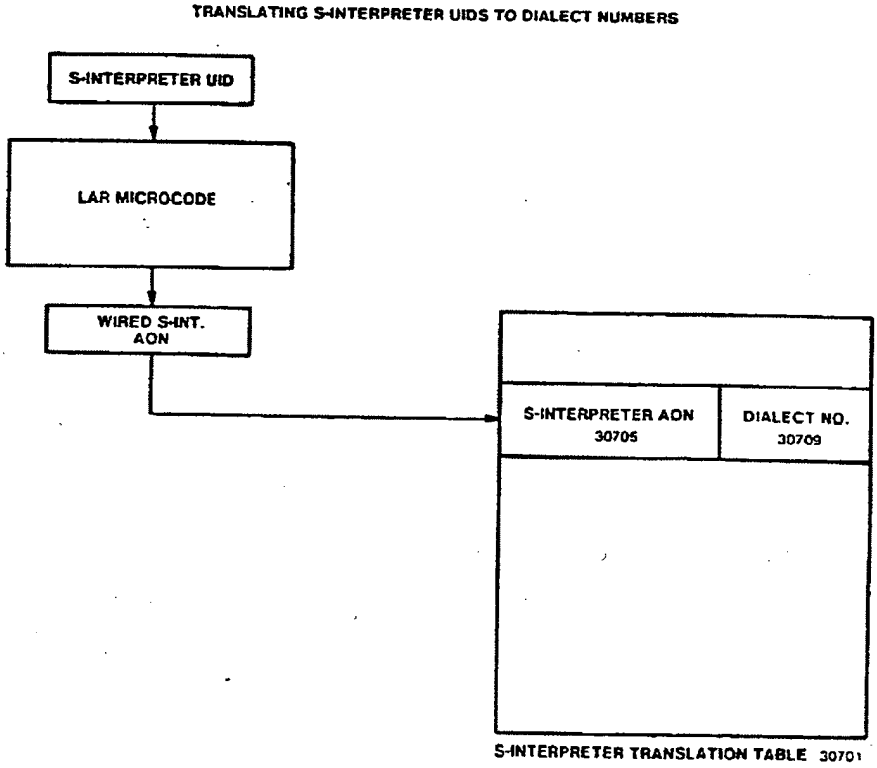


FIG. 307

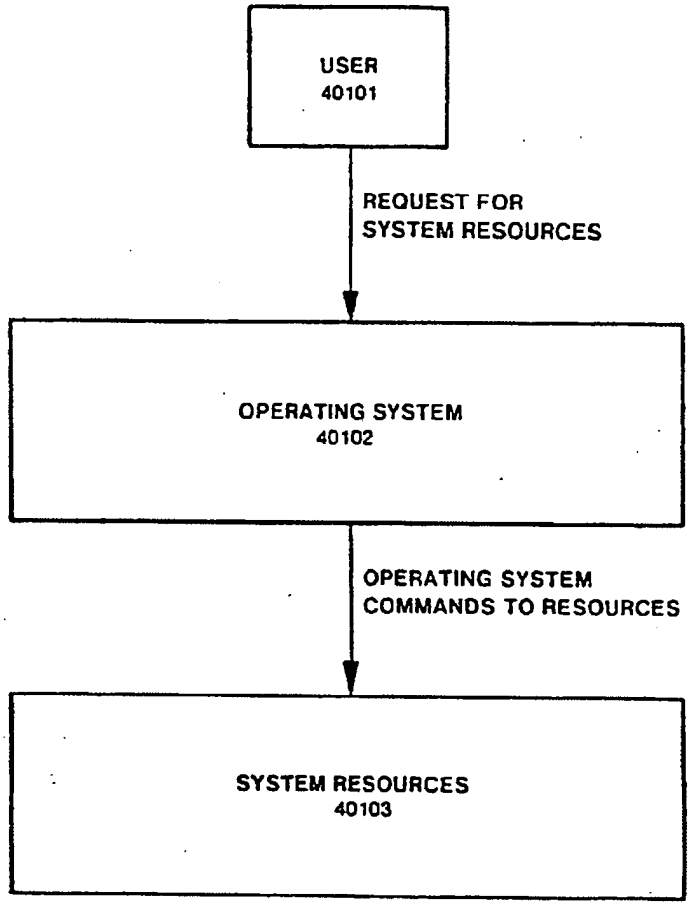


FIG 401

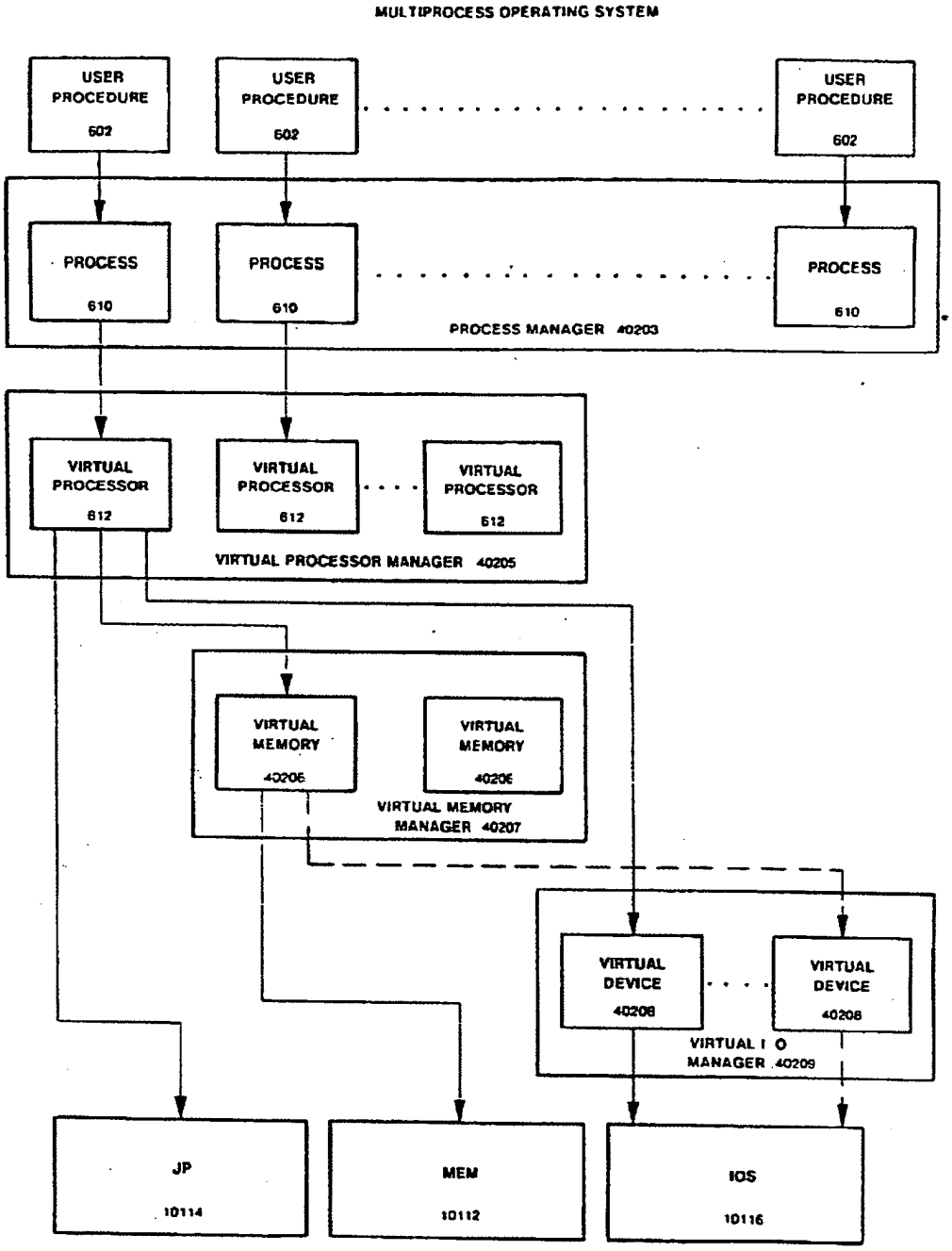


FIG 402

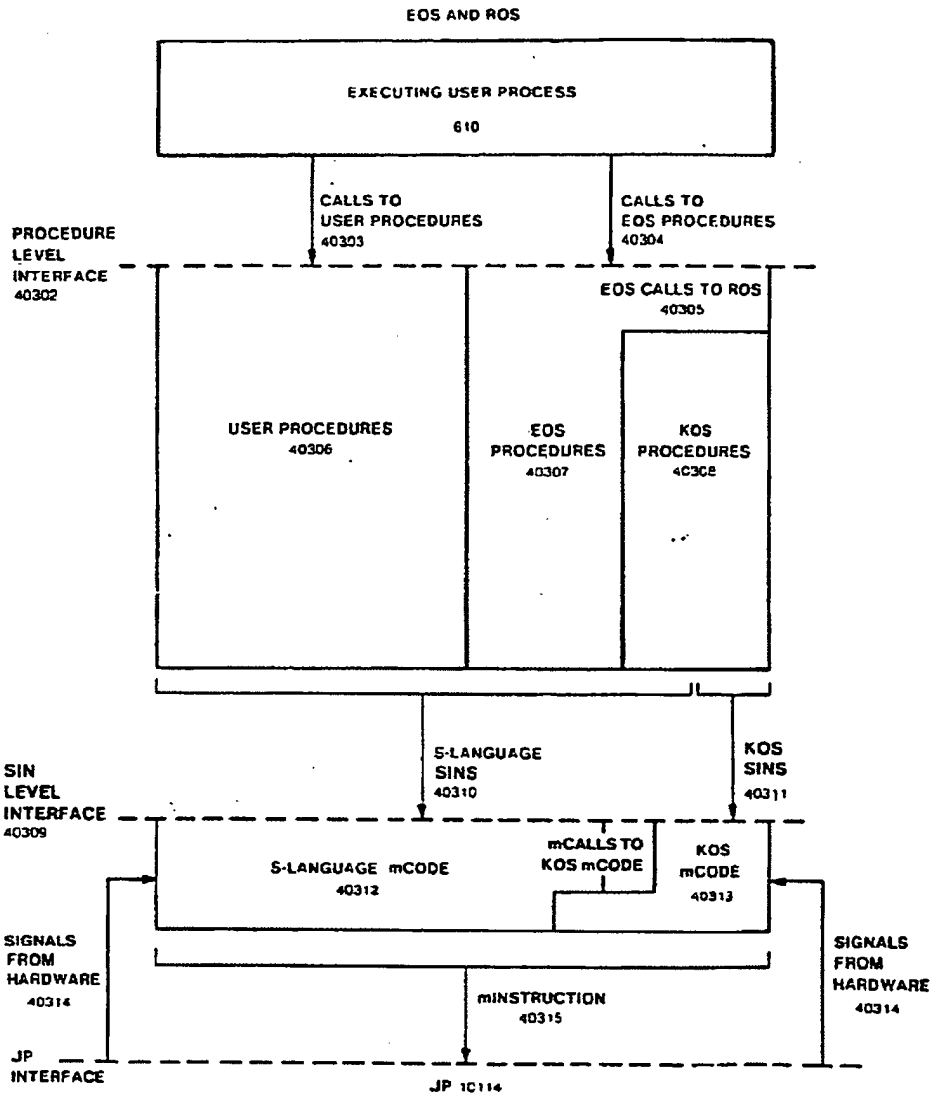


FIG. 403

EOS VIEW OF OBJECTS

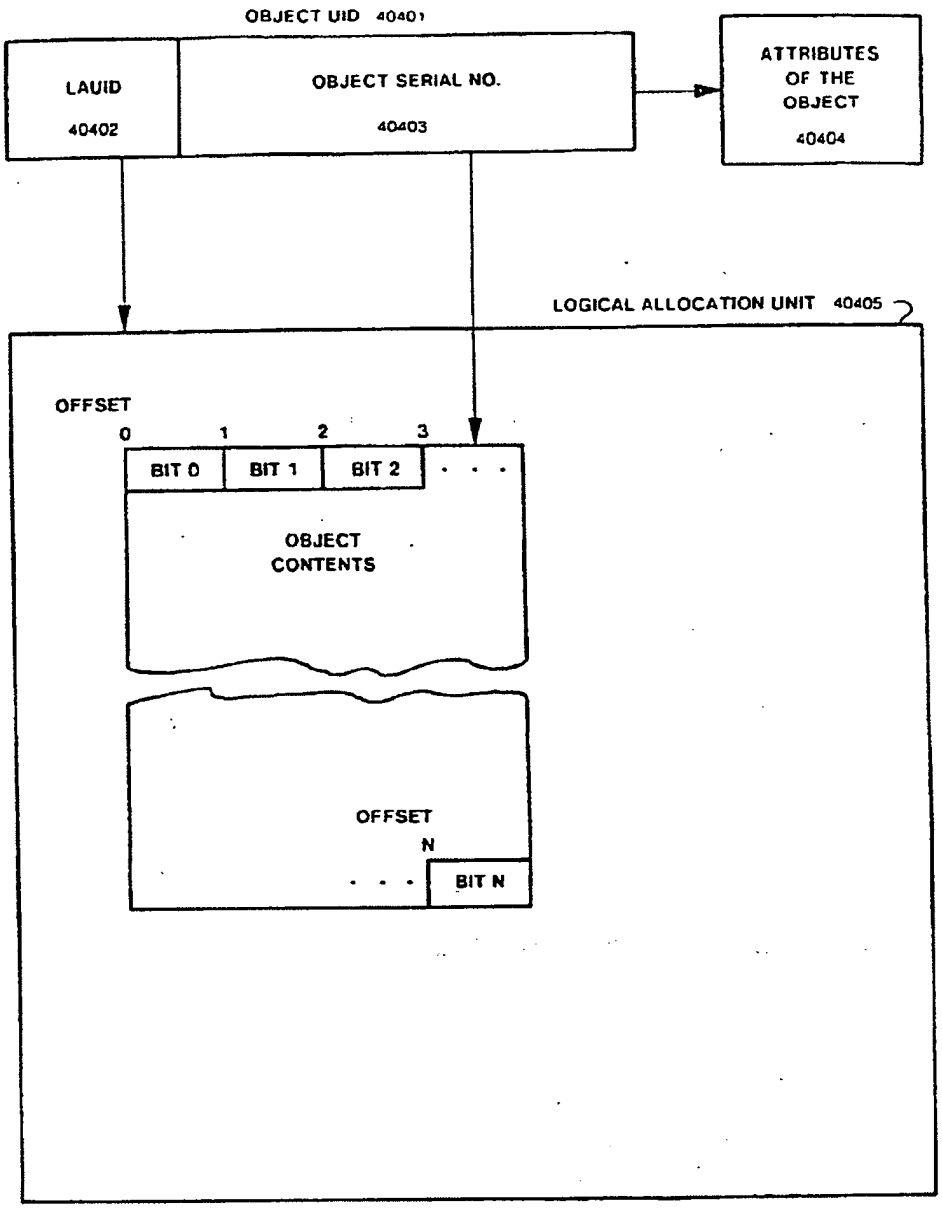


FIG 404

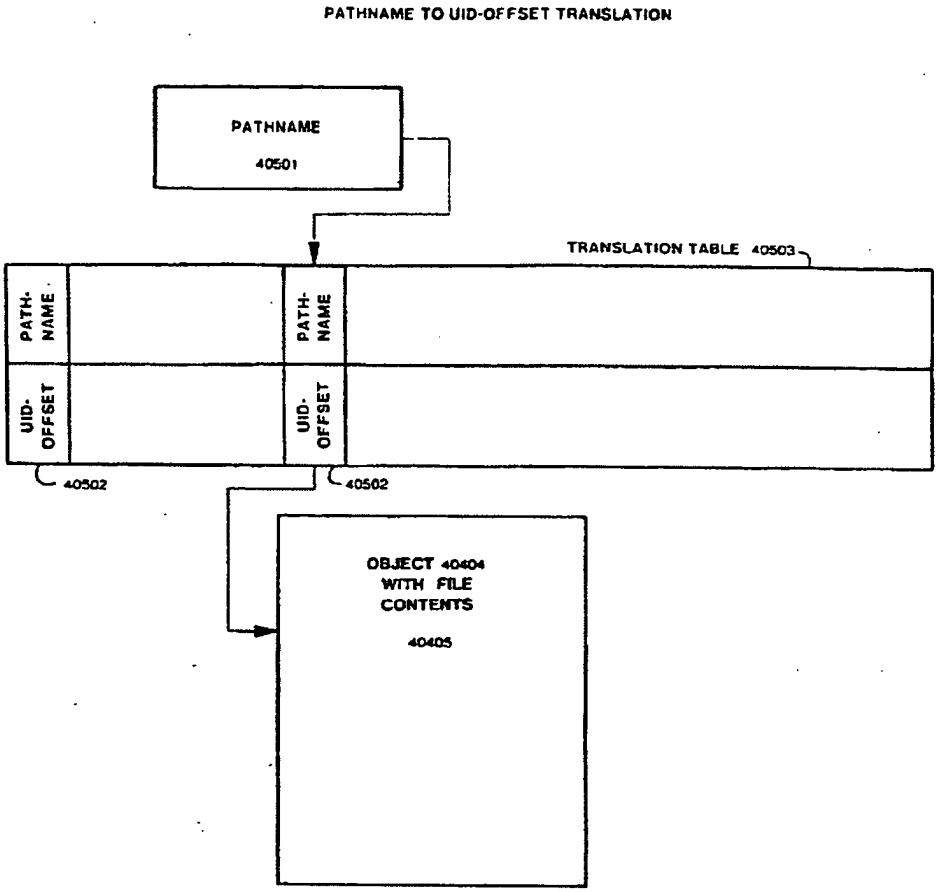


FIG 405

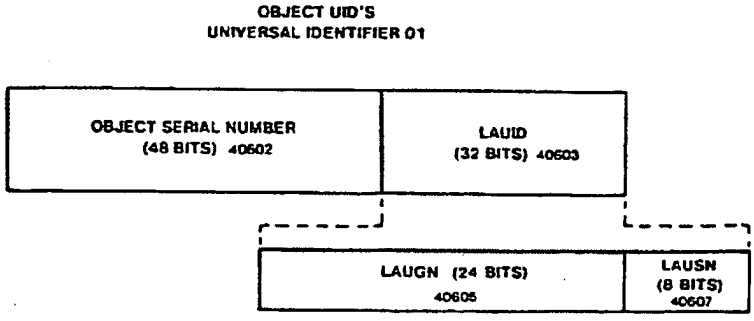


FIG 406

ATU, MHT, AND MEMORY

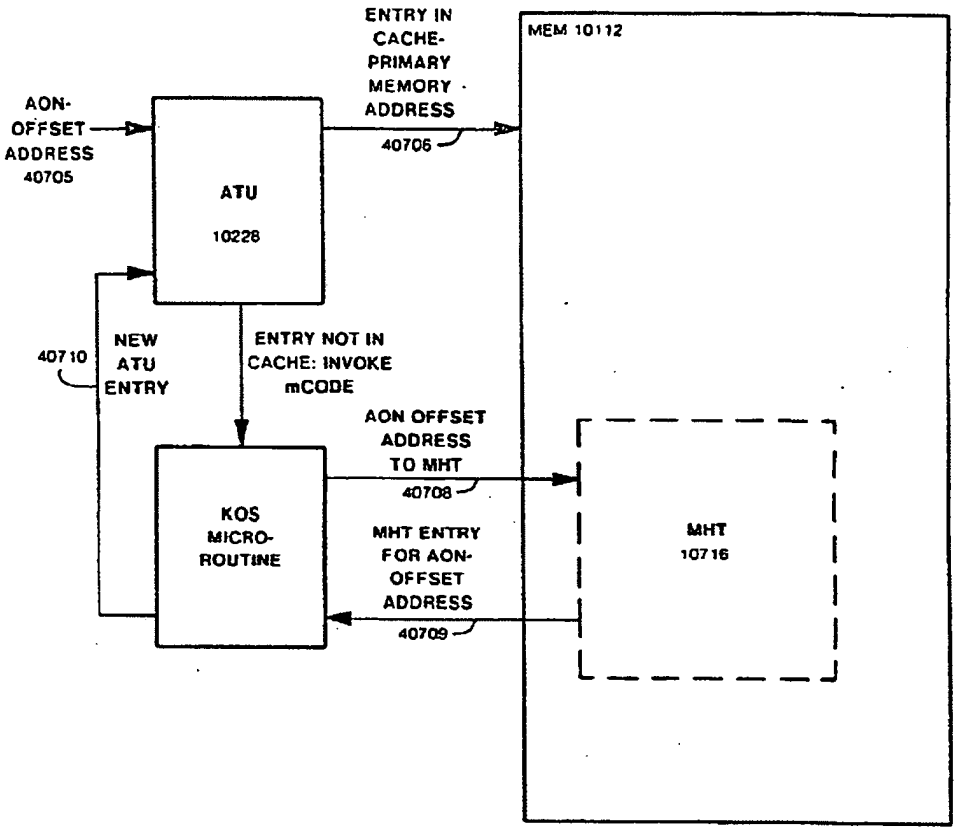


FIG 407

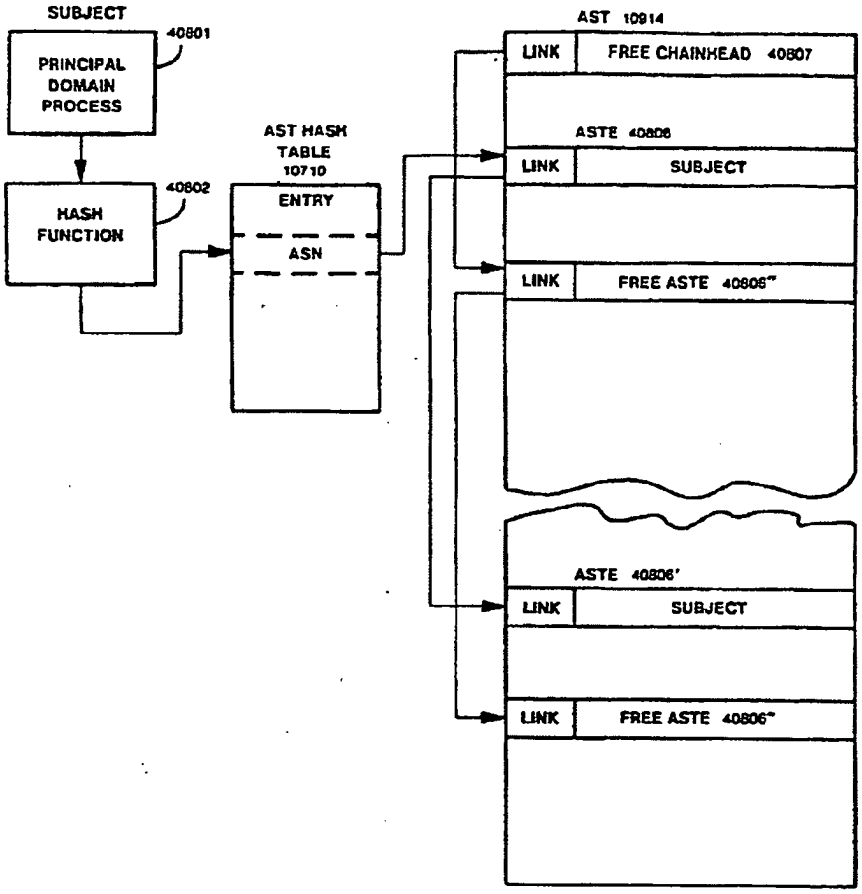


FIG 408

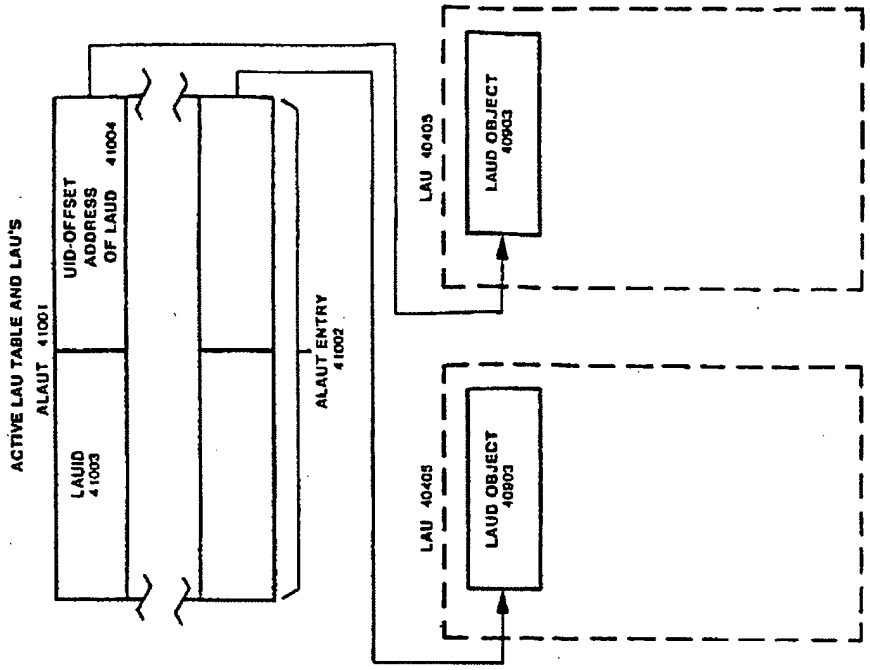


FIG 410

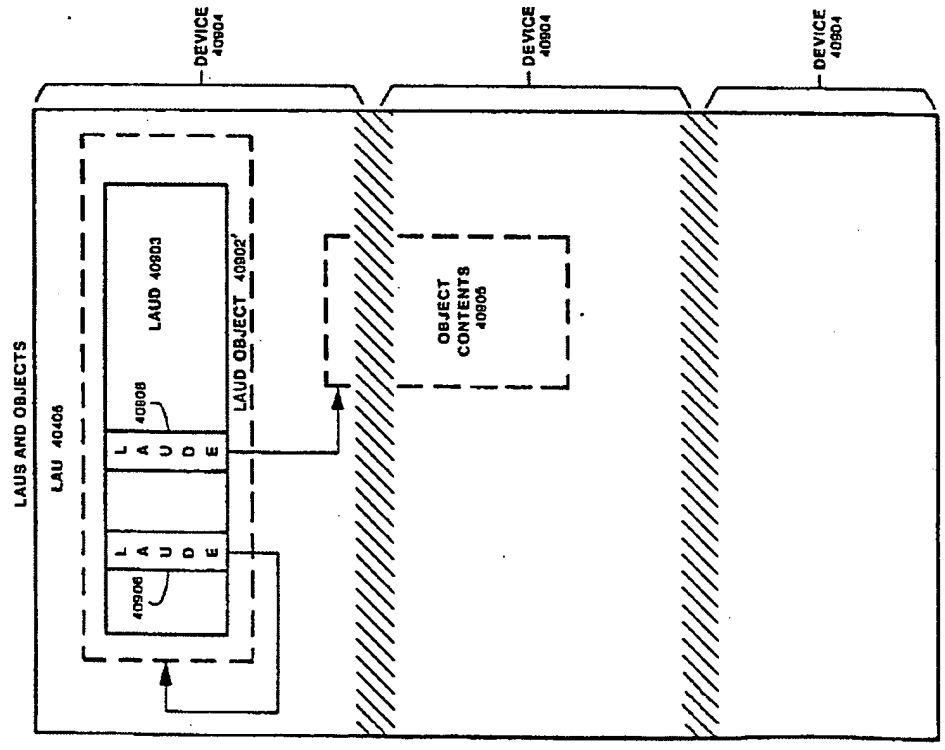


FIG 409

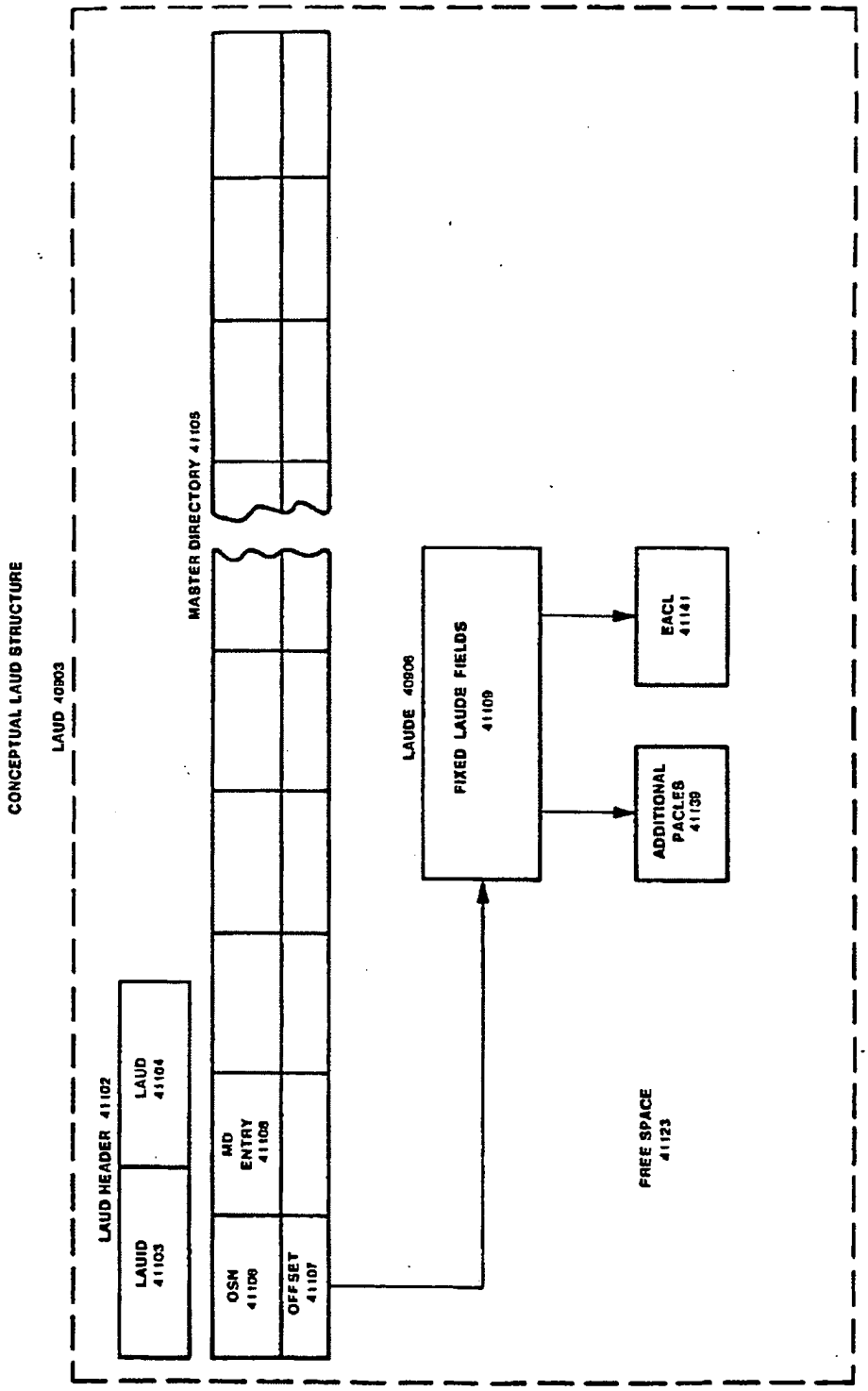


FIG 411

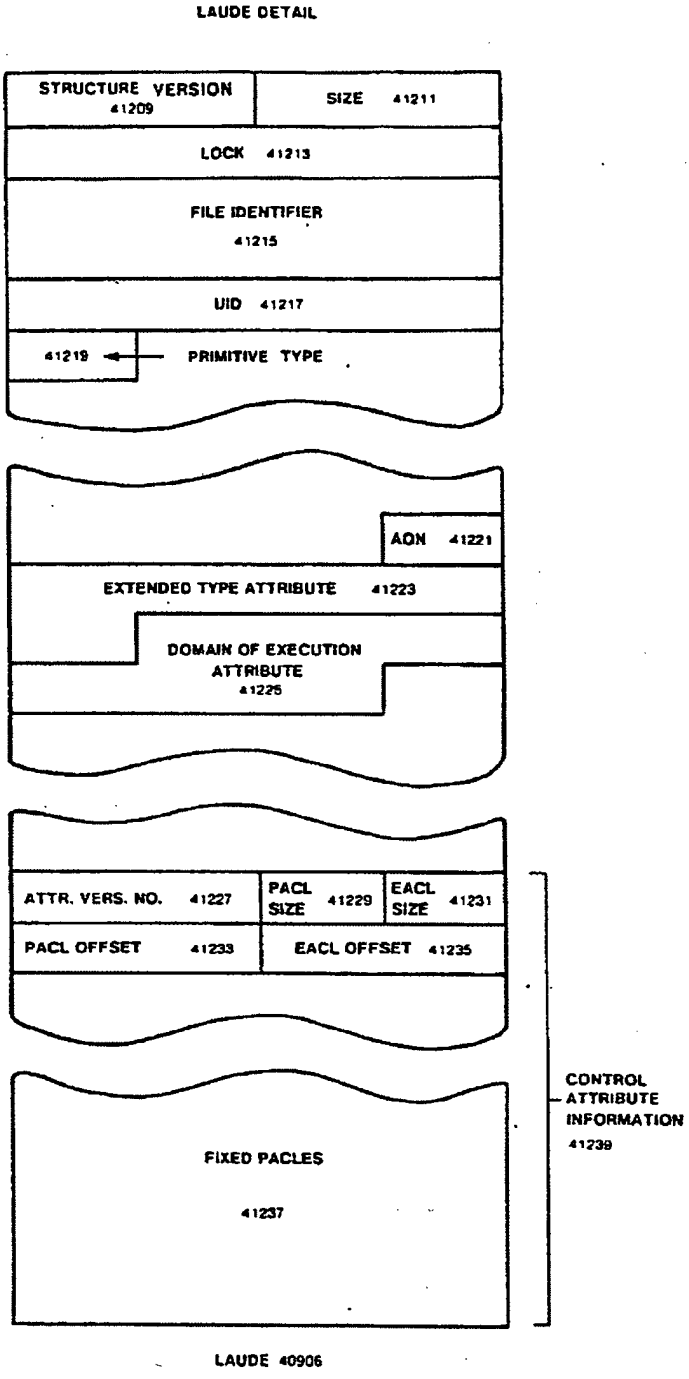


FIG. 412

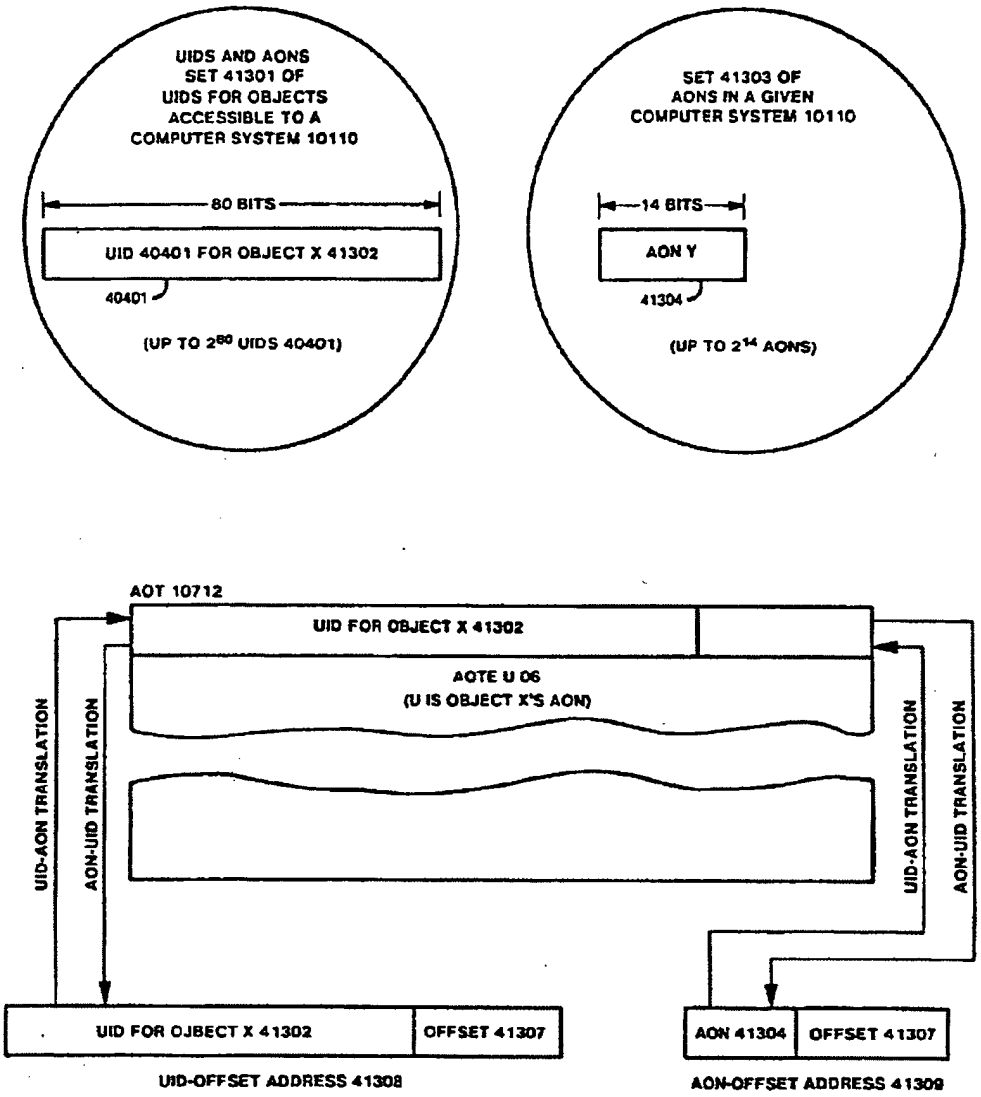


FIG 413

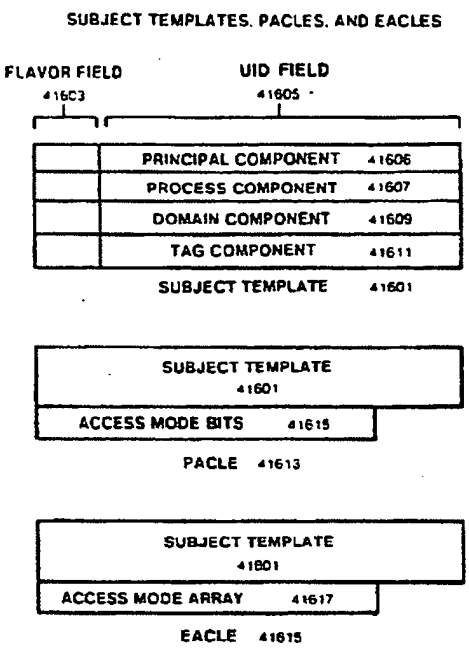


FIG. 416

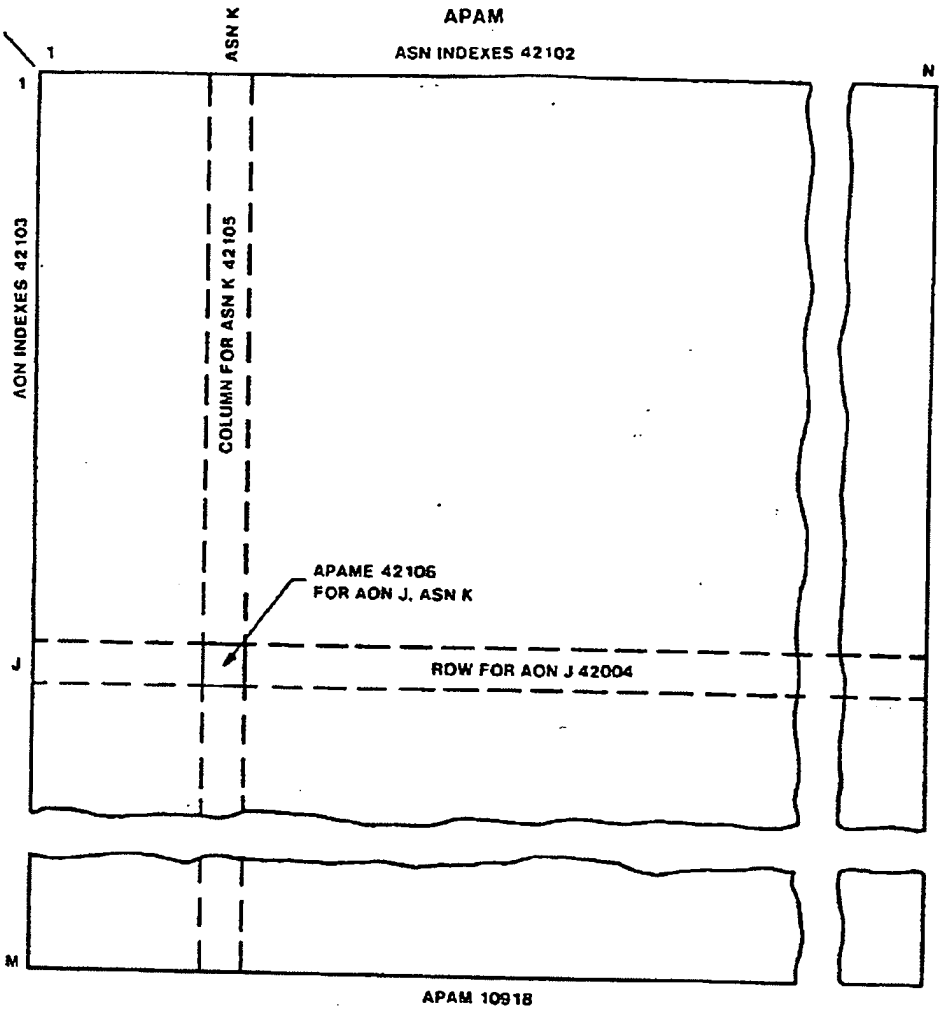


FIG. 421

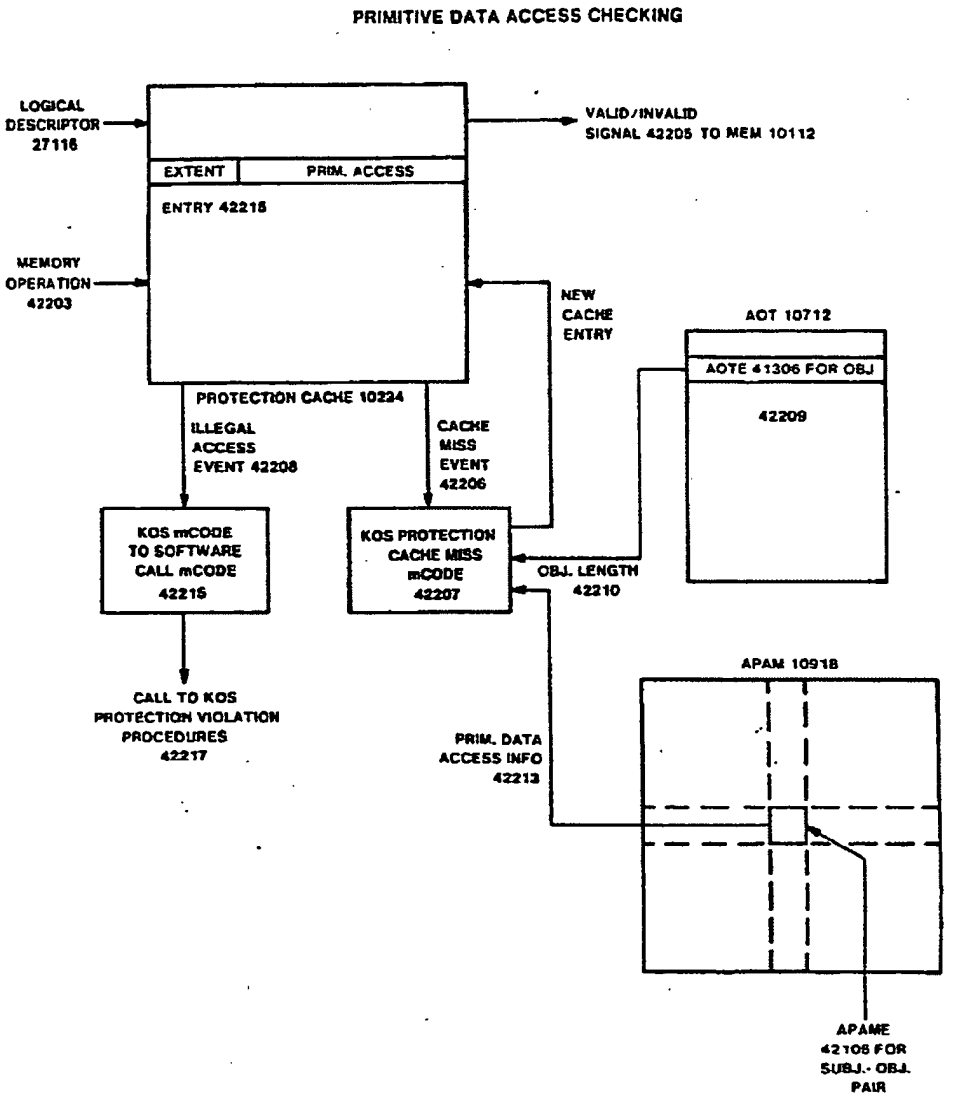


FIG. 422

EVENT COUNTERS AND AWAIT ENTRIES

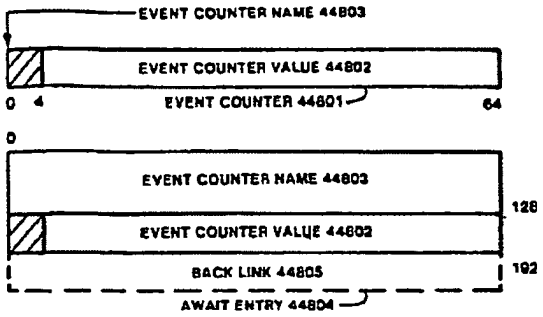


FIG. 448

AWAIT TABLE OVERVIEW

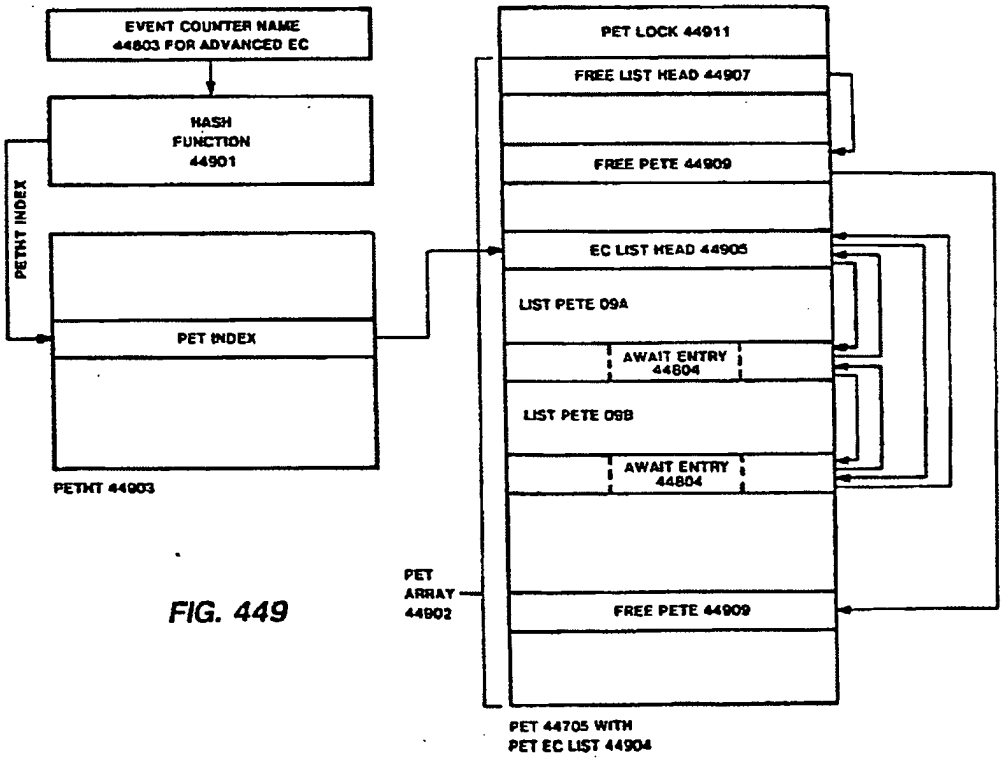


FIG. 449

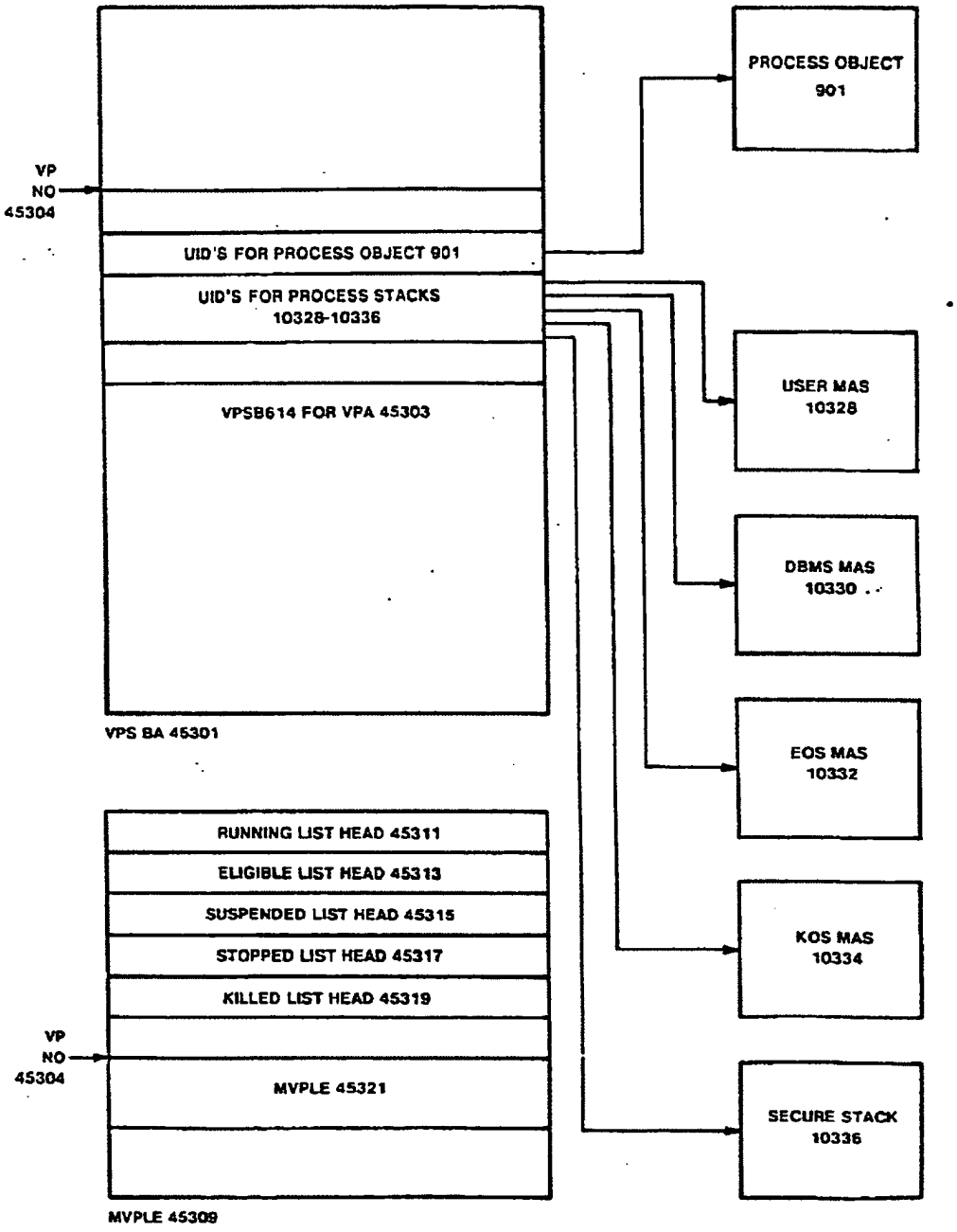


FIG. 453

VIRTUAL PROCESSOR SYNCHRONIZATION

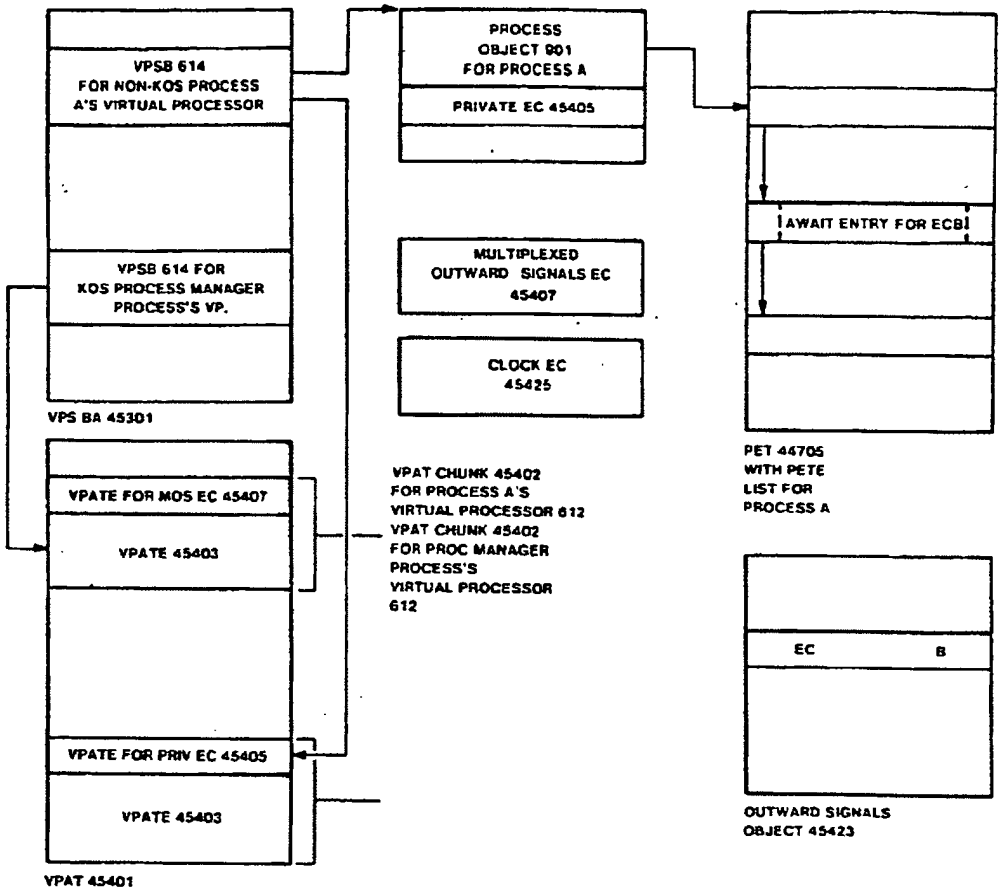


FIG. 454

MAS OBJECT OVERVIEW

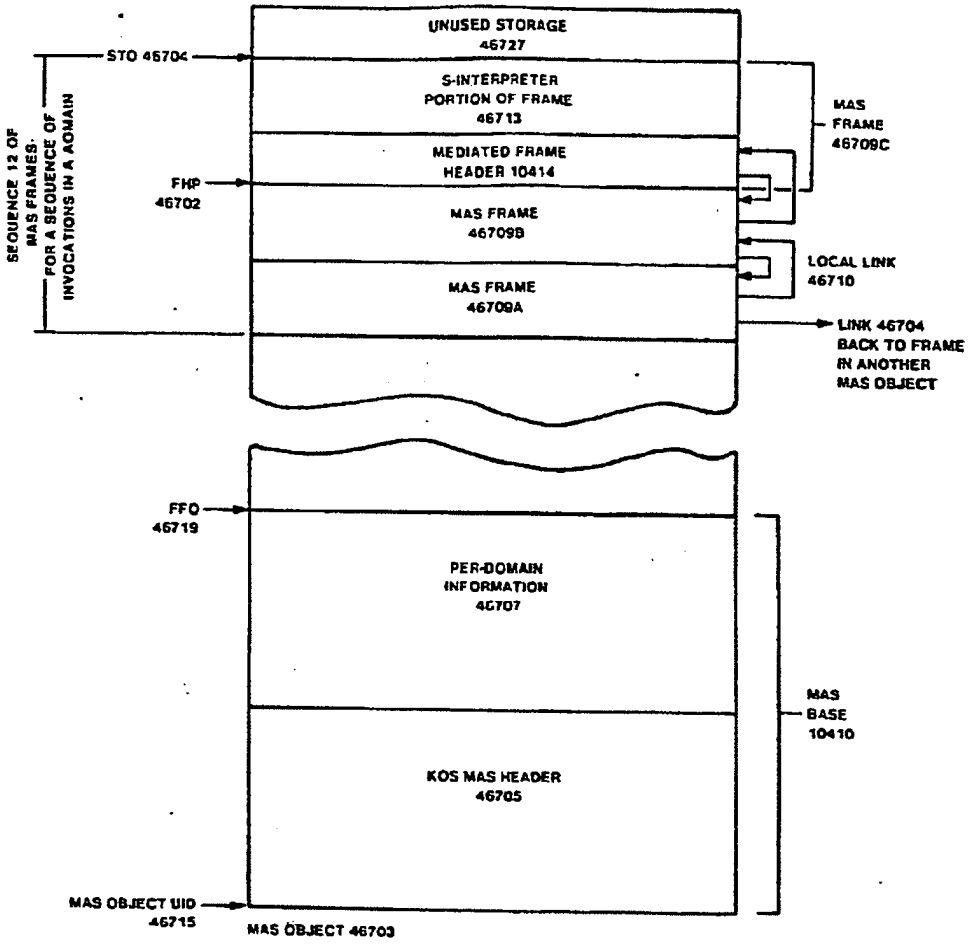


FIG. 467

MAS BOSE DETAIL

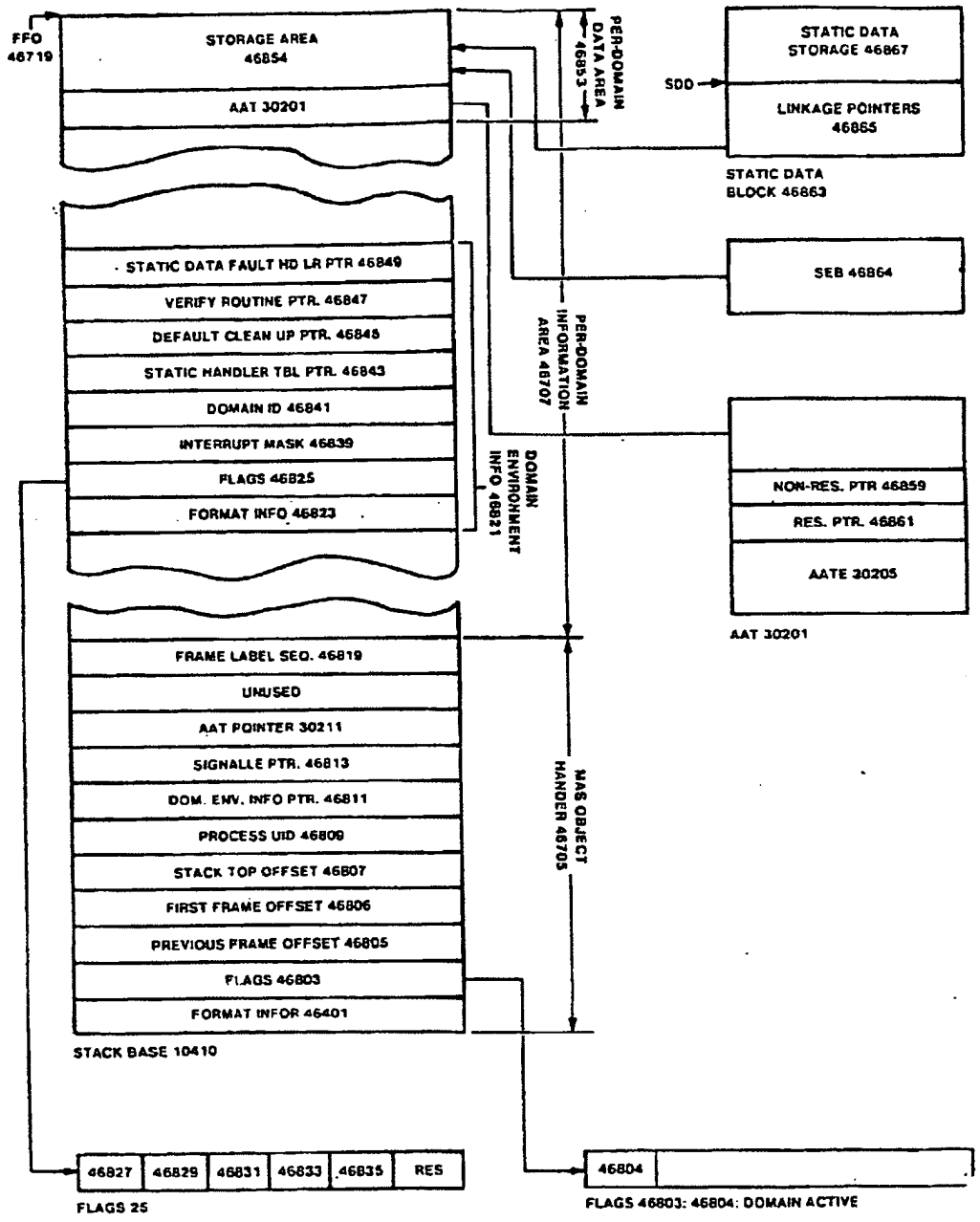


FIG. 468

MAS FRAME DETAIL

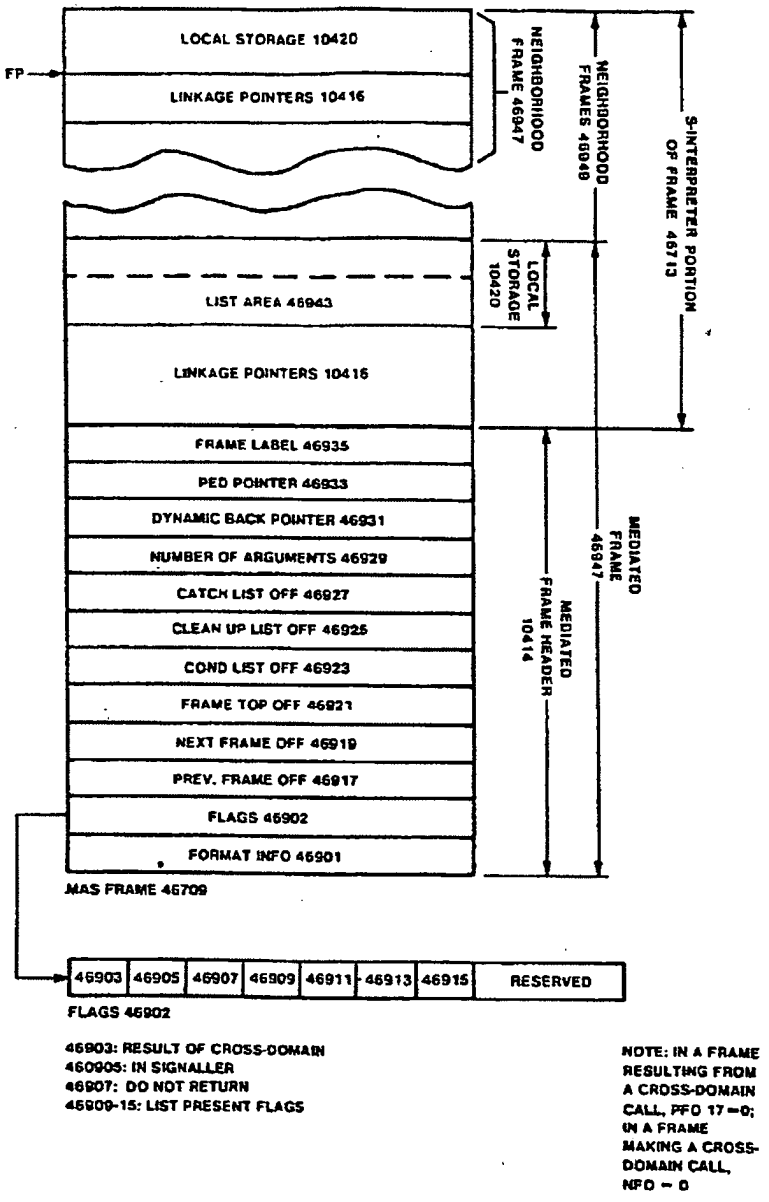


FIG. 469

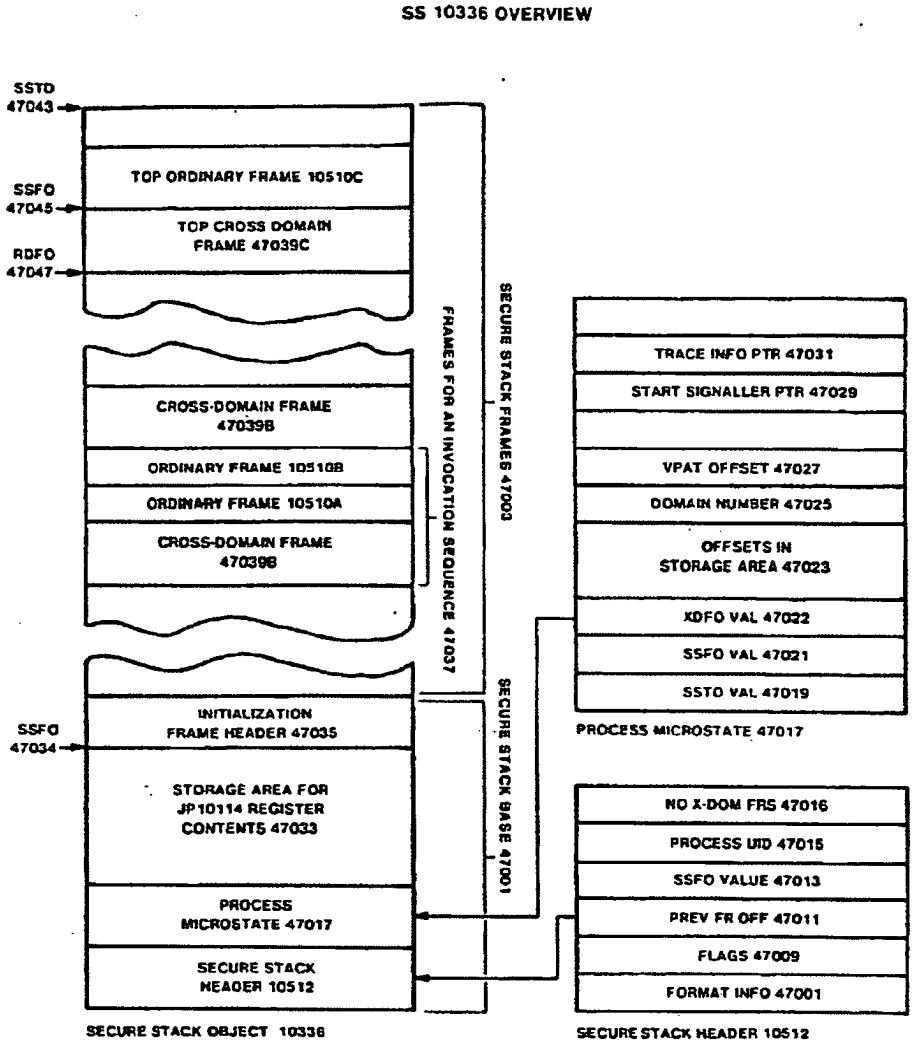


FIG. 470

SECURE STACK 10336 FRAME DETAIL

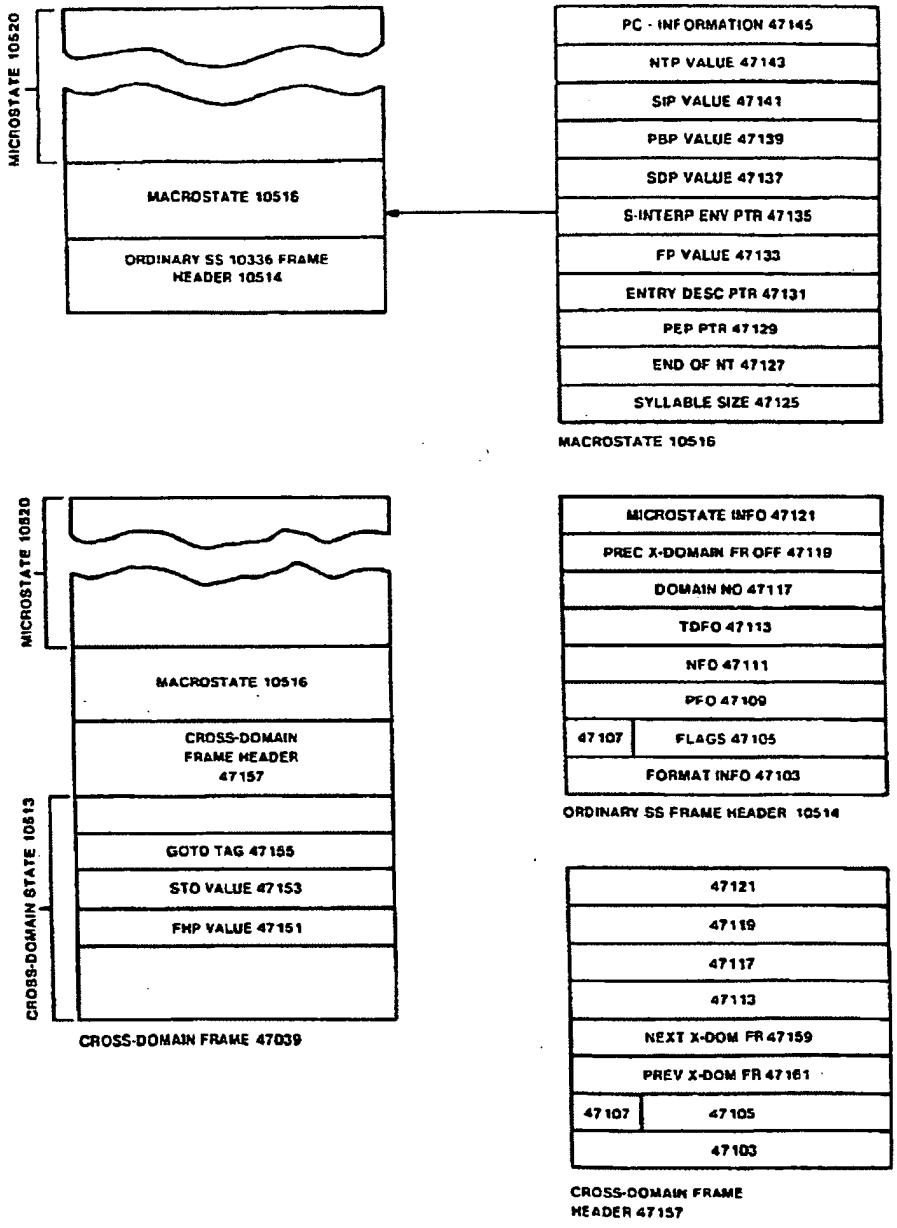


FIG. 471

PROCEDURE OBJECT OVERVIEW

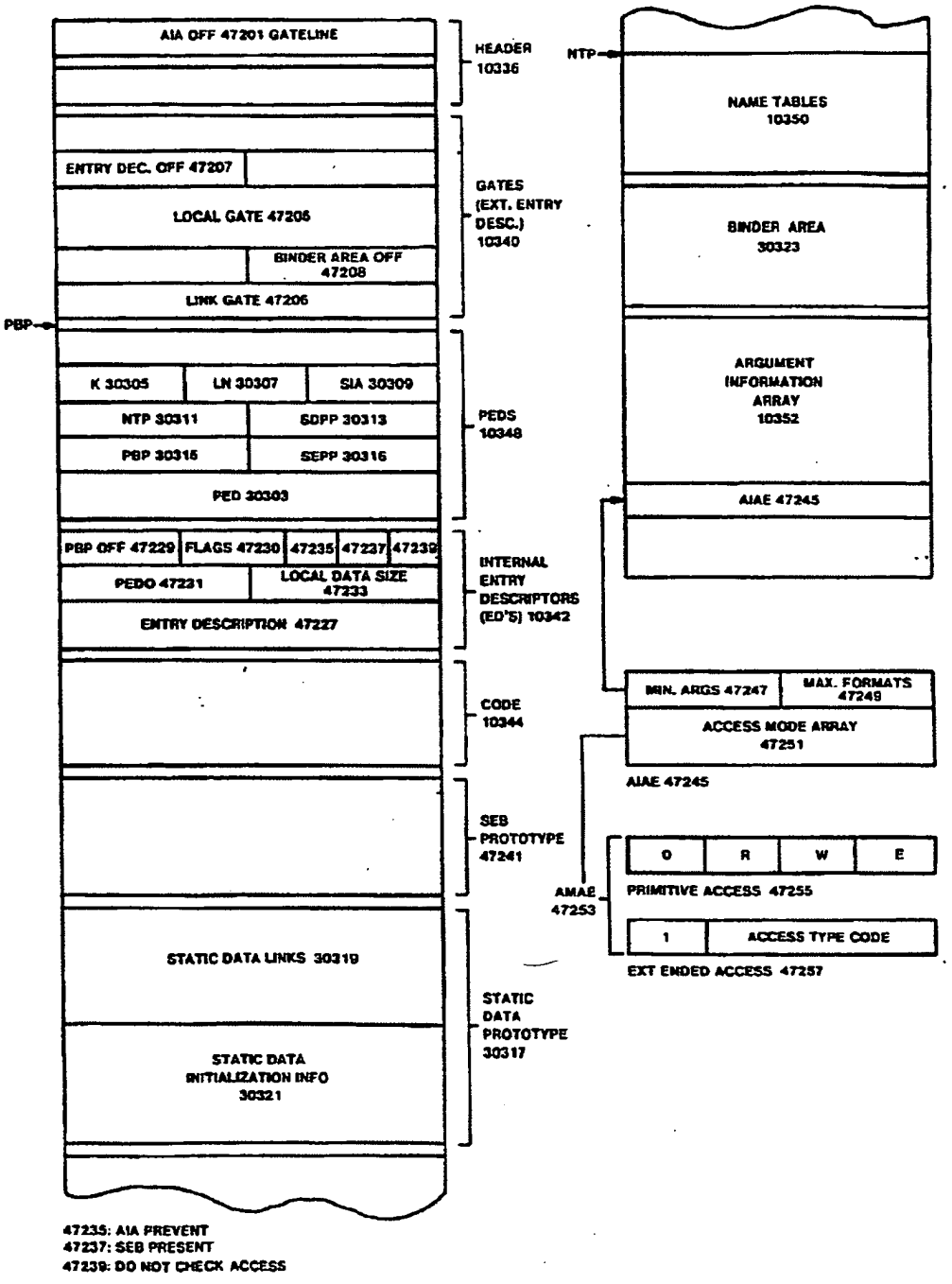


FIG. 472

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 257 585 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **87112158.8**

(51) Int. Cl.⁴: **H04L 9/00**

(22) Date of filing: **21.08.87**

(30) Priority: **22.08.86 JP 197610/86**
22.08.86 JP 197611/86

(43) Date of publication of application:
02.03.88 Bulletin 88/09

(84) Designated Contracting States:
BE DE FR GB

(71) Applicant: **NEC CORPORATION**
33-1, Shiba 5-chome, Minato-ku
Tokyo 108(JP)

(72) Inventor: **Okamoto, Eiji c/o NEC Corporation**
33-1, Shiba 5-chome
Minato-ku Tokyo(JP)

(74) Representative: **Vossius & Partner**
Siebertstrasse 4 P.O. Box 86 07 67
D-8000 München 86(DE)

(54) Key distribution method.

(57) The invention relates to a method of distributing a key for enciphering un unenciphered or plaintext message and for deciphering the enciphered message.

The method comprises the following steps: generating a first random number in a first system (101); generating first key distribution information in the first system (101) by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system (101); transmitting the first key distribution information to a second system (102) via a communication channel (103); receiving the first key distribution information in the second system (102); generating a second random number in the second system (102); generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system (102); transmitting the second key distribution information to the first system (101) via the channel (103); receiving the second key distribution information in the first system (101); and generating an enciphering key in the first system (101) by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system (102) which is not secret.

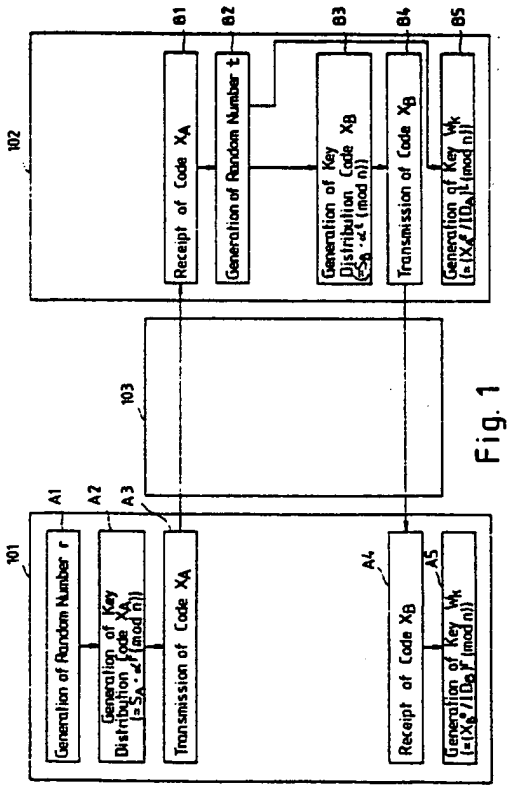


Fig. 1

EP 0 257 585 A2

KEY DISTRIBUTION METHOD

BACKGROUND OF THE INVENTION

The invention relates to a method of distributing a key for enciphering an unenciphered or plaintext message and for deciphering the enciphered message.

A public key distribution method used in a public key cryptosystem as a well-known key distribution method is disclosed in a paper entitled "New Directions in Cryptography" by W. Diffie and M.E. Hellman, published in the IEEE Transactions on Information Theory, Vol. IT-22, No. 6, pp. 644 to 654, November issue, 1976. The key distribution method disclosed in the paper memorizes public information for each of conversers. In the system, before a converser A sends an enciphered message to a converser B, the converser A prepares an enciphering key (which represents a number obtained by calculating $Y_B{}^{X_A} \pmod{\underline{p}}$) generated from public information Y_B of the converser B and secret information X_A which is kept secret by the converser A. The number $\underline{p}$ is a large prime number of about 256 bits in binary representation, which is publicly known. $\underline{a} \pmod{\underline{b}}$ means a remainder of division of the number $\underline{a}$ by the number $\underline{b}$. The converser B also prepares the key wk in accordance to $Y_A{}^{X_B} \pmod{\underline{p}}$ in a similar manner. Y_A and Y_B are selected so as to be equal to $\alpha^{X_A} \pmod{\underline{p}}$ and $\alpha^{X_B} \pmod{\underline{p}}$, respectively. As a result, $Y_B{}^{X_A} \pmod{\underline{p}}$ becomes equal to $Y_A{}^{X_B} \pmod{\underline{p}}$. It is known that even if Y_A, $\underline{a}$ and $\underline{p}$ are known, it is infeasible for anybody except the converser A to obtain X_A which satisfies $Y_A = \alpha^X A \pmod{\underline{p}}$.

The prior art key distribution system of the type described, however, has disadvantages in that since the system needs a large amount of public information corresponding to respective conversers, the amount of the public information increases as the number of conversers increases. Further, strict control of such information becomes necessary to prevent the information from being tampered.

SUMMARY OF THE INVENTION

An object of the invention is, therefore, to provide a key distribution method free from the above-mentioned disadvantages of the prior art system.

According to an aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in a first system; generating first key distribution information in the first system by applying a predetermined first transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information to a second system via a communication channel; receiving the first key distribution information in the second system; generating a second random number in the second system; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information to the first system via the channel; receiving the second key distribution information in the first system; and generating an enciphering key in the first system by applying a predetermined second transformation to the second key distribution information on the basis of the first random number and identification information of the second system which is not secret.

According to another aspect of the invention, there is provided a method which comprises the following steps: generating a first random number in the first system; generating first key distribution information by applying a predetermined first transformation to the first random number on the basis of public information in the first system and generating first identification information by applying a predetermined second transformation to the first random number on the basis of first secret information known only by the first system; transmitting the first key distribution information and the first identification information to a second system via a communication channel; receiving the first key distribution information and the first identification information in the second system; examining whether or not the result obtained by applying a predetermined third transformation to the first key distribution information on the basis of the first identification information satisfies a first predetermined condition, and, if it does not satisfy, suspending key distribution processing; generating a second random number if said condition is satisfied in the preceding step; generating second key distribution information by applying the predetermined first transformation to the second random number on the basis of the public information, and generating second identification information by applying the predetermined second transformation to the second random number on the basis of second secret information known only by the second system; transmitting the second key distribution information and the second identification information to the first system via the communication channel; and exam-

ining whether or not the result obtained by applying a third predetermined transformation to the second key distribution information on the basis of the second identification information in the first system satisfies a predetermined second condition, and if the result does not satisfy the second condition, suspending the key distribution processing, or if it satisfies the second condition, generating an enciphering key by applying a fourth predetermined transformation to the first random number on the basis of the second key distribution information.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features and advantages of the invention will become more apparent from the following detailed description when taken in conjunction with the accompanying drawings in which:

FIG. 1 is a block diagram of a first embodiment of the invention;

FIG. 2 is a block diagram of a second embodiment of the invention; and

FIG. 3 is a block diagram of an example of systems 101, 102, 201 and 202.

In the drawings, the same reference numerals represent the same structural elements.

PREFERRED EMBODIMENTS

Referring now to FIG. 1, a first embodiment of the invention comprises a first system 101, a second system 102 and an insecure communication channel 103 such as a telephone line which transmits communication signals between the systems 101 and 102. It is assumed herein that the systems 101 and 102 are used by users or conversers A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α and n which are not necessarily secret while the user B has or knows a secret integer number $S_B$and the public integer numbers. These integer numbers are designated and distributed in advance by a reliable person or organization. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case in which the user A starts communication. The system 101 of the user A generates a random number γ (Step A1 in FIG. 1) and sends a first key distribution code X_A representative of a number obtained by computing $S_A \cdot \alpha^{\gamma} \pmod{n}$ (Step A2) to the system 102 of the user B (step A3). Next, when the system 102 receives the code X_A(Step B1), it generates a random number t (Step B2), calculates $(X_A^e/ID_A)^t \pmod{n}$ (Step B5), and keeps the resulting number as a enciphering key wk for enciphering a message into storage means (not shown). The identification code ID_A represents herein a number obtained by considering as a numeric value a code obtained by encoding the address, the name and so on of the user A. The encoding is, for instance, performed on the basis of the American National Standard Code for Information Interchange. Then, the system 102 transmits to the system 101 of the user A a second key distribution code X_B representative of a number obtained by calculating $S_B \cdot \alpha^t \pmod{n}$ (Steps B3 and B4).

The system 101, on the other hand, receives the code X_B (Step A4), calculates $(X_B^e/ID_B)^{\gamma} \pmod{n}$ (Step A5), and keeps the resulting number as the key wk for enciphering a message. The identification code ID_B represents the numbers obtained by considering as a numeric value a code obtained by encoding the name, address, and so on of the user B.

Subsequently, communication between the users A and B will be conducted by transmitting messages enciphered with the enciphering key wk via the channel 103.

The integer numbers S_A, S_B, e, c, α and n are determined as follows. n is assumed to be a product of two sufficiently large prime numbers p and q. For instance, p and q may be 2^{256} or so. e and c are prime numbers which are equal to or less than n, while α is a positive integer number which is equal to or less than n. Further, d is defined as an integer number which satisfies $e \cdot d \pmod{(p-1)\cdot(q-1)} = 1$. S_A and S_B are defined as numbers obtainable from $ID_A^d \pmod{n}$ and $ID_B^d \pmod{n}$, respectively.

If S_A, S_B, e, c, α, and n are defined as above, ID_A and ID_B become equal to $S_A^e \pmod{n}$ and $S_B^e \pmod{n}$, respectively. This can be proved from a paper entitled "A Method for Obtaining Digital Signatures and Publick-Key Cryptosystems" by R.L. Rivest et al., published in the Communication of the ACM, Vol. 21, No. 2, pp. 120 to 126. Since the key obtained by $(X_B^e/ID_B)^r \pmod{n}$ on the side of the user A becomes equal to $\alpha^{ert} \pmod{n}$ and the key obtained by $(X_A^e/ID_A)^t \pmod{n}$ on the side of the user B becomes equal to $\alpha^{ert} \pmod{n}$, they can prepare the same enciphering key. Even if a third party tries to assume the identity of the user A, he cannot prepare the key wk since he cannot find out z which meets $ID_A = Z^e \pmod{n}$.

Referring now to FIG. 2, a second embodiment of the invention comprises a first system 201, a second system 202 and an insecure communication channel 203. It is assumed herein that the systems 201 and 202 are used by users A and B, respectively. The user A has or knows a secret integer number S_A and public integer numbers e, c, α, and n, which are not necessarily secret while

the user B has or knows a secret integer number $S_B$and the public integer numbers. These integer numbers are designated and distributed by a reliable person or organization in advance. The method to designate the integer numbers will be described later.

An operation of the embodiment will next be described on a case where the user A starts communication. The system 201 of the user A generates a random number r (Step AA1 in FIG. 2) and determines a first key distribution code X_A representative of a number obtained by computing $\alpha^{e \cdot r} \pmod{n}$ as well as a first identification code $Y_A$indicative of a number obtained by computing $S_A \circ \alpha^{c \cdot r} \pmod{n}$ (AA2). The system 201 then transmits a first pair of X_A and Y_A to the system 202 of the user B (Step AA3). Thereafter, the system 202 receives the first pair (X_A , Y_A) (Step BB1), calculates $Y_A^e / X_A^c \pmod{n}$, and examines whether or not the number obtained by the calculation is identical to the number indicated by an identification code ID_A obtained by the address, the name and so on of the user A in a similar manner to in the first embodiment (Step BB2). If they are not identical to each other, the system suspends processing of the key distribution (Step BB7). On the other hand, if they are identical to each other, the system 202 generates a random number t (Step BB3) and determines a second key distribution code X_B representative of a number obtained by calculating $\alpha^{e \cdot t} \pmod{n}$ and a second identification code Y_B obtained by calculating $S_B \circ \alpha^{c \cdot t} \pmod{n}$ (Step BB4). The system 202 then transmits a second pair of X_B and Y_B to the system 201 of the user A (Step BB5). The system 202 calculates $X_A^t \pmod{n}$ and keeps the number thus obtained as a enciphering key wk (Step BB6).

The system 201, on the other hand, receives the second pair (X_B, Y_B) (Step AA4), calculates $Y_B^e / X_B^c \pmod{n}$, and examines whether or not the number thus obtained is identical to the number indicated by an identification code ID_B obtained by the address, the name and so on of the user B in a similar manner to in the first embodiment (Step AA5). If they are not identical to each other, the system suspends the key distribution processing (Step AA7). If they are identical to each other, the system 201 calculates $X_B^r \pmod{n}$, and stores the number thus obtained as a enciphering key wk (Step AA6). Although the codes ID_A and ID_B are widely known, they may be informed by the user A to the user B.

The integer numbers S_A, S_B, e, c, α and n are determined in the same manner as in the first embodiment. As a result, ID_A and ID_B becomes equal to $Y_A^e / X_A^c \pmod{n}$ ($= S_A^e \circ \alpha^{erc} / \alpha^{erc} \pmod{n}$) and $Y_B^e / X_B^c \pmod{n}$ ($= S_B^e \circ \alpha^{etc} / \alpha^{etc} \pmod{n}$), respectively. If we presuppose that the above-mentioned reliable person or organization who prepared S_A and S_B do not act illegally, since S_A is possessed only by the user A while S_B is possessed only by the user B, the first pair (x_A, y_A) which satisfies $y_A^e / x_A^c \pmod{n} = ID_A$ can be prepared only by the user A while the second pair (x_B , y_B) which satisfies $y_B^e / x_B^c \pmod{n} = ID_B$ can be prepared only by the user B. It is impossible to find out a number x which satisfies $x^f \pmod{n} = b$ on the basis of f, $b$and n since finding out X is equivalent to breaking the RSA public key cryptogram system disclosed in the above-mentioned the Communication of the ACM. It is described in the above-referenced IEEE Transactions on Information Theory that the key wk cannot be calculated from the codes x_A or x_B and n. The key distribution may be implemented similarly by making the integer number C variable and sending it from a user to another.

An example of the systems 101, 102, 201 and 202 to be used in the first and second embodiments will next be described referring to FIG. 3.

Referring now to FIG. 3, a system comprises a terminal unit (TMU) 301 such as a personal computer equipped with communication processing functions, a read only memory unit (ROM) 302, a random access memory unit (RAM) 303, a random number generator (RNG) 304, a signal processor (SP) 306, and a common bus 305 which interconnects the TMU 301, the ROM 302, the RAM 303, the RNG 304 and the SP 306.

The RNG 304 may be a key source 25 disclosed in U.S. Patent No. 4,200,700. The SP 306 may be a processor available from CYLINK Corporation under the trade name CY 1024 KEY MANAGEMENT PROCESSOR.

The RNG 304 generates random numbers r or t by a command given from the SP 306. The ROM 407 stores the public integer numbers e , c, α, n and the secret integer number S_A (if the ROM 407 is used in the system 101 or 201) or the secret integer number S_B (if the ROM 407 is used in the system 102 or 202). The numbers S_A and S_B may be stored in the RAM 303 from the TMU 301 everytime users communicates. According to a program stored in the ROM 407, the SP 306 executes the above-mentioned steps A2, A5, AA2, AA5, AA6 and AA7 (if the SP 306 is used in the system 101 or 201), or the steps B3, B5, BB2, BB4, BB6 and BB7 (if the SP 306 is used in the system 102 or 202). The RAM 303 is used to temporarily store calculation results in these steps.

Each of the systems 101, 102, 201 and 202 may be a data processing unit such as a general purpose computer and an IC (integrated circuit) card.

As described in detail hereinabove, this invention enables users to effectively implement key distribution simply with a secret piece of information and several public pieces of information.

While this invention has thus been described in conjunction with the preferred embodiments thereof, it will now readily be possible for those skilled in the art to put this invention into practice in various other manners.

Claims

1. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information to a second system via a communication channel;

d) receiving said first key distribution information in said second system;

e) generating a second random number in said second system;

f) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of second secret information known only by said second system;

g) transmitting said second key distribution information to said first system via said channel;

h) receiving said second key distribution information in said first system; and

i) generating an enciphering key in said first system by applying a predetermined second transformation to said second key distribution information on the basis of said first random number and identification information of said second system which is not secret.

2. A key distribution method as claimed in Claim 1, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said steps c) and h).

3. A key distribution method as claimed in Claim 1 or 2, in which said second system includes second data processing means for executing said steps e) and f), and second communication processing means for executing said steps d) and g).

4. A key distribution method comprising the following steps:

a) generating a first random number in a first system;

b) generating first key distribution information in said first system by applying a predetermined first transformation to said first random number on the basis of public information and generating first identification information by applying a predetermined second transformation to said first random number on the basis of first secret information known only by said first system;

c) transmitting said first key distribution information and said first identification information to a second system via a communication channel;

d) receiving said first key distribution information and said first identification information in said second system;

e) examining whether or not the result obtained by applying a predetermined third transformation to said first key distribution information on the basis of said first identification information satisfies a predetermined first condition and, if it does not satisfy, suspending key distribution processing;

f) generating a second random number if said first condition is satisfied at said step e);

g) generating second key distribution information by applying said predetermined first transformation to said second random number on the basis of said public information, and generating second identification information by applying said predetermined second transformation to said second random number on the basis of second secret information known only by said second system;

h) transmitting said second key distribution information and said second identification information to said first system via said communication channel; and

i) examining in said first system whether or not the result obtained by applying a predetermined third transformation to said second key distribution information on the basis of said second identification information satisfies a predetermined second condition and, if the result does not satisfy said second condition, suspending said key distribution processing or, if it satisfies said second condition, generating said enciphering key by applying a predetermined fourth transformation to said first random number on the basis of said second key distribution information.

5. A key distribution method as claimed in Claim 4, in which said first system includes first data processing means for executing said steps a), b) and i), and first communication processing means for executing said step c).

6. A key distribution method as claimed in Claim 4 or 5, in which said second system includes second data processing means for executing said steps e), f) and g), and second communication processing means for executing said steps d) and h).

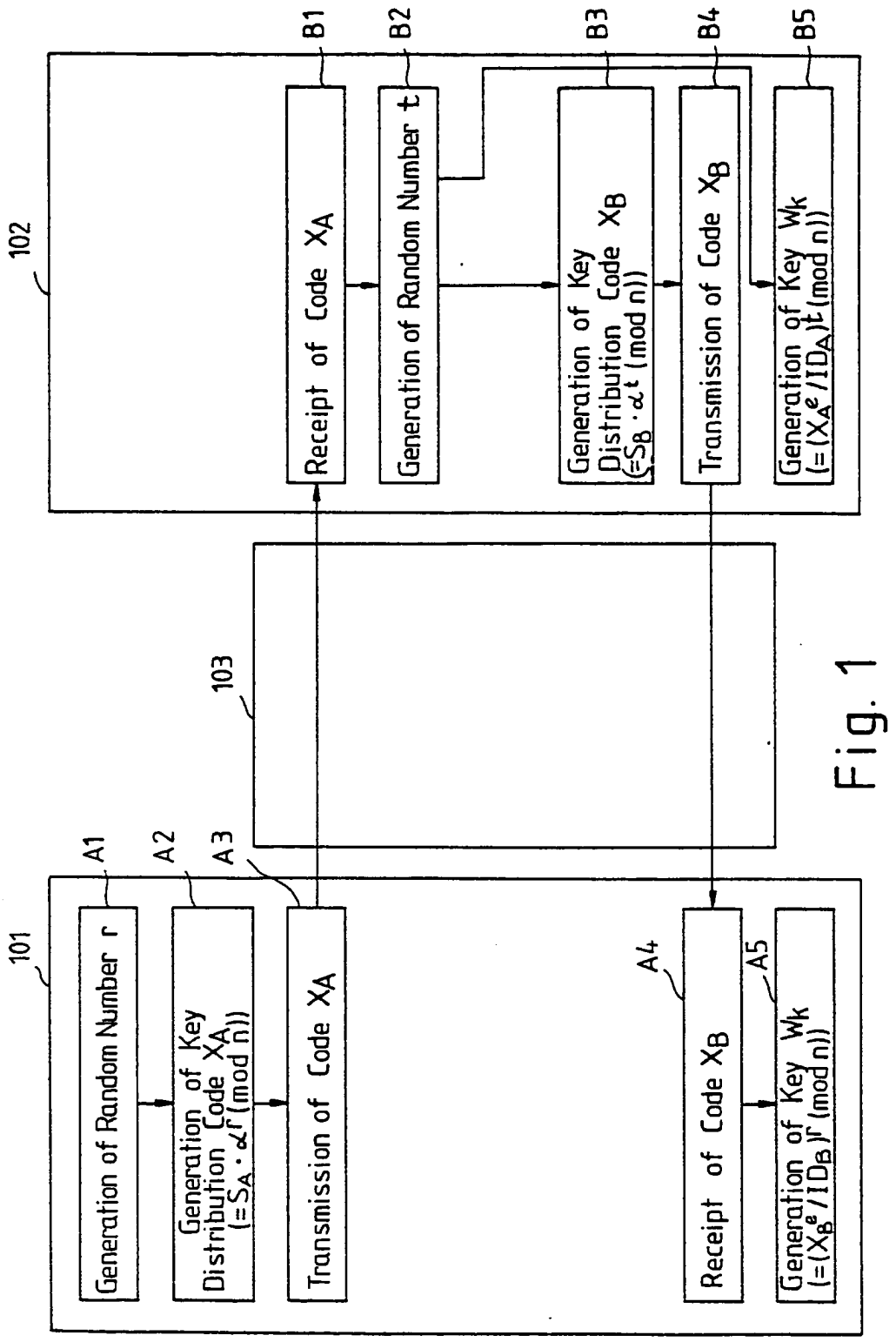

102
B1
B2
B3
B4
B5
Receipt of Code X_A
Generation of Random Number t
Generation of Key Distribution Code X_B ($=S_B \cdot \alpha^t \pmod n$)
Transmission of Code X_B
Generation of Key W_k ($=(X_A^e/ID_A)^t \pmod n$)
103
101
A1
A2
A3
A4
A5
Generation of Random Number r
Generation of Key Distribution Code X_A ($=S_A \cdot \alpha^r \pmod n$)
Transmission of Code X_A
Receipt of Code X_B
Generation of Key W_k ($=(X_B^e/ID_B)^r \pmod n$)

Fig. 1

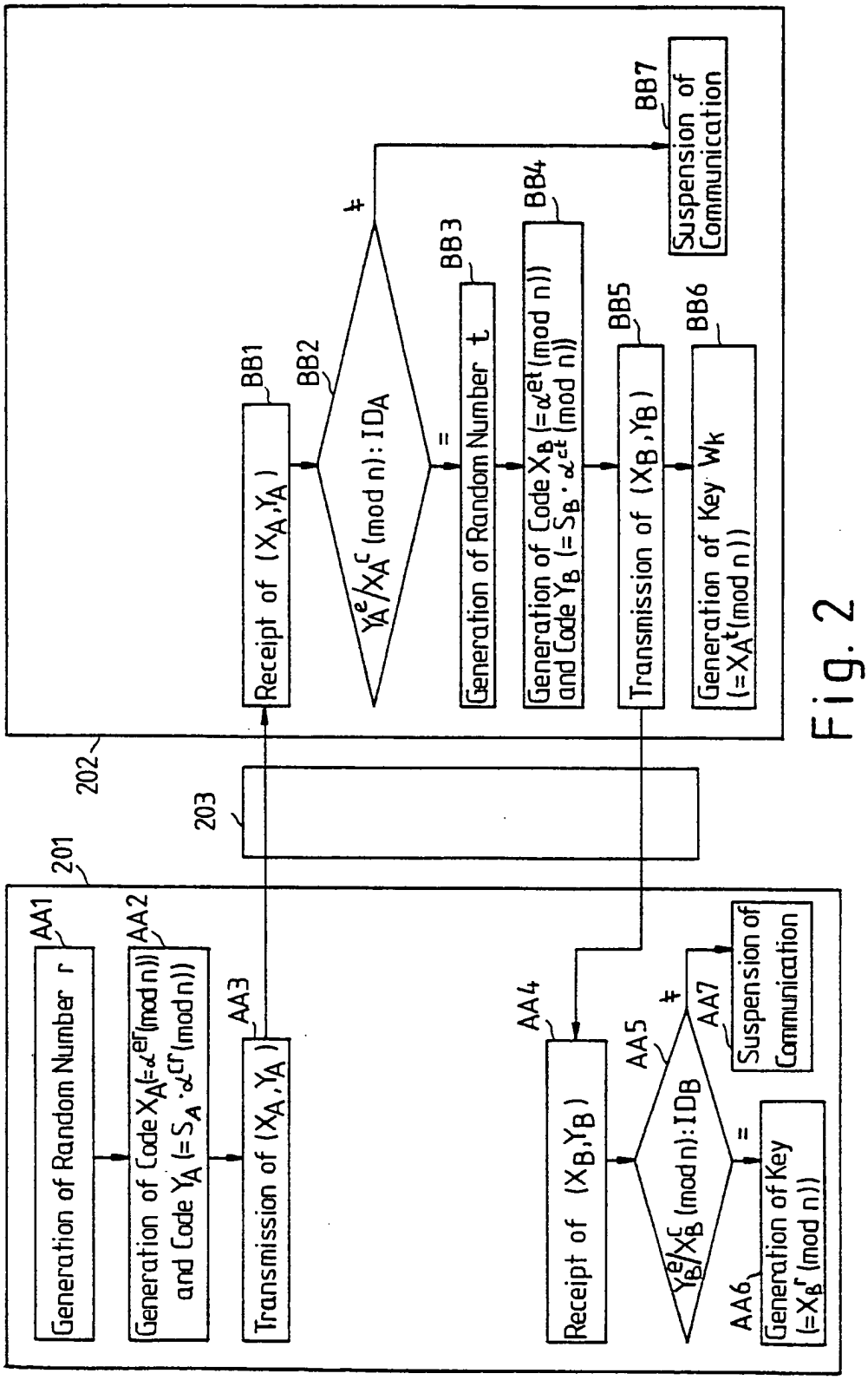

201
202
203
AA1
Generation of Random Number r
AA2
Generation of Code $X_A(=\alpha^{er} \pmod n)$ and Code $Y_A(=S_A \cdot \alpha^{cr} \pmod n)$
AA3
Transmission of (X_A, Y_A)
AA4
Receipt of (X_B, Y_B)
AA5
$Y_B^e/X_B^c \pmod n$: ID_B
AA6
Generation of Key $(=X_B^r \pmod n)$
AA7
Suspension of Communication
BB1
Receipt of (X_A, Y_A)
BB2
$Y_A^e/X_A^c \pmod n$: ID_A
BB3
Generation of Random Number t
BB4
Generation of Code $X_B(=\alpha^{et} \pmod n)$ and Code $Y_B(=S_B \cdot \alpha^{ct} \pmod n)$
BB5
Transmission of (X_B, Y_B)
BB6
Generation of Key W_k $(=X_A^t \pmod n)$
BB7
Suspension of Communication

Fig. 2

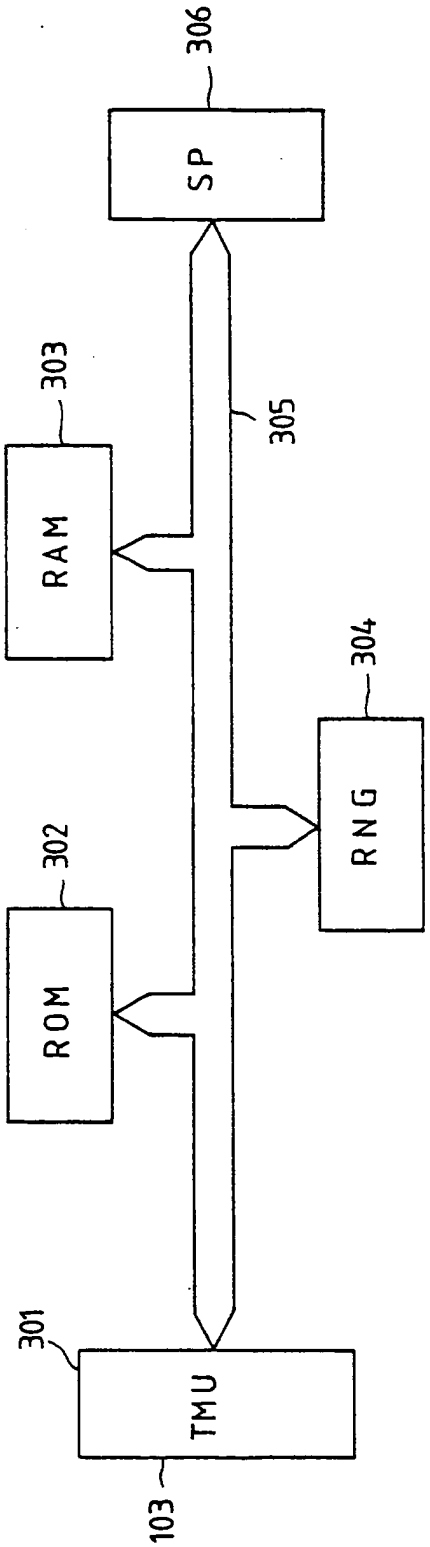


Fig. 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 332 304 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89301510.7

(51) Int. Cl.4: G06F 1/00

(22) Date of filing: 16.02.89

(30) Priority: 07.03.88 US 164944

(43) Date of publication of application:
13.09.89 Bulletin 89/37

(84) Designated Contracting States:
DE FR GB

(71) Applicant: DIGITAL EQUIPMENT CORPORATION
111 Powdermill Road
Maynard Massachusetts 01754-1418(US)

(72) Inventor: Robert, Gregory
12 Carson Circle
Nashua New Hampshire 03062(US)
Inventor: Chase, David
28 Bay View Road
Wellesley Massachusetts 02181(US)
Inventor: Schaefer, Ronald
7 Gioconda Avenue
Acton Massachusetts 01720(US)

(74) Representative: Goodman, Christopher et al
Eric Potter & Clarkson 14 Oxford Street
Nottingham NG1 5BP(GB)

(54) Software licensing management system.

(57) A license management system which includes a license management facility that determines whether usage of a licensed program is within the scope of the license. The license management system maintains a license unit value for each licensed program and a pointer to a table identifying an allocation unit value associated with each use of the licensed program. In response to a request to use a licensed program, the license management system responds with an indication as to whether the license unit value exceeds the allocation unit value associated with the use. Upon receiving the response, the operation of the licensed program depends upon policies established by the licensor.

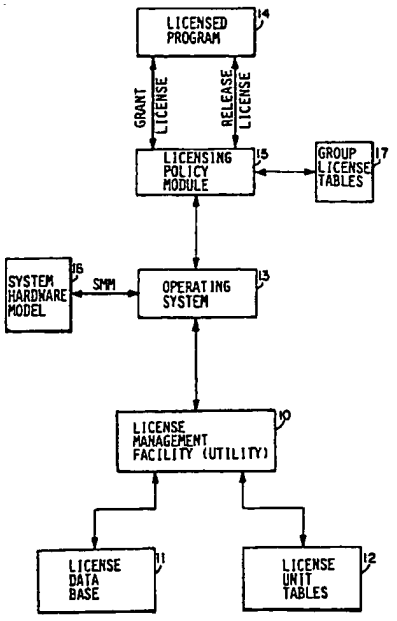


FIG. 1

EP 0 332 304 A2

SOFTWARE LICENSING MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of digital data processing systems, and more specifically to a system for managing licensing for, and usage of, the various software programs which may be processed by the systems to ensure that the software programs are used within the terms of the software licenses.

2. Description of the Prior Art

A digital data processing system includes three basic elements, namely, a processor element, a memory element and an input/output element. The memory element stores information in addressable storage locations. This information includes data and instructions for processing the data. The processor element fetches information from the memory element, interprets the information as either an instruction or data, processes the data in accordance with the instructions, and returns the processed data to the memory element for storage therein. The input/output element, under control of the processor element, also communicates with the memory element to transfer information, including instructions and data to be processed, to the memory, and to obtain processed data from the memory.

Typically, an input/output element includes a number of diverse types of units, including video display terminals, printers, interfaces to the public telecommunications network, and secondary storage subsystems, including disk and tape storage devices. A video display terminal permits a user to run programs and input data and view processed data. A printer permits a user to obtain a processed data on paper. An interface to the public telecommunications network permits transfer of information over the public telecommunications network.

The instructions processed by the processor element are typically organized into software programs. Recently, generation and sales of software programs have become significant businesses both for companies which are primarily vendors of hardware, as well as for companies which vend software alone. Software is typically sold under license, that is, vendors transfer copies of software to users under a license which governs how the users may use the software. Typically, software costs are predicated on some belief as to the amount of usage which the software program may provide and the economic benefits, such as cost saving which may otherwise be incurred, which the software may provide to the users. Thus, license fees may be based on the power of the processor or the number of processors in the system, or the number of individual nodes in a network, since these factors provide measures of the number of users which may use the software at any given time.

In many cases, however, it may also be desirable, for example, to have licenses and license fees more closely relate to the actual numbers of users which can use the program at any given time or on the actual use to which a program may be put. Furthermore, it may be desirable to limit the use of the program to specified time periods. A problem arises particularly in digital data processing systems which have multiple users and/or multiple processors, namely, managing use of licensed software to ensure that the use is within the terms of the license, that is, to ensure that the software is only used on identified processors or by the numbers of users permitted by the license.

SUMMARY OF THE INVENTION

The invention provides a new and improved licensing management system for managing the use of licensed software in a digital data processing system.

In brief summary, the license management system includes a license management facility and a licensing policy module that jointly determine whether a licensed program may be operated. The license management facility maintains a license unit value for each licensed program and a pointer to a table identifying a license usage allocation unit value associated with usage of the licensed program. In response to a request to use a licensed program, the license management facility determines whether the remaining license unit value exceeds the license usage allocation unit value associated with the use. If the license unit value does not exceed the license usage allocation unit value, the license management facility permits usage of the licensed program and adjusts the license unit value by a function of the license usage allocation unit value to reflect the usage. On the other hand, if the license unit value associated with use of the license program does exceed the li-

cense usage allocation unit value, the licensing policy module determines whether to allow the licensed program to be used in response to other licensing policy factors.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1 is a general block diagram of a new system in accordance with the invention;

Figs. 2 and 3 are diagrams of data structures useful in understanding the detailed operation of the system depicted in Fig. 1; and

Figs. 4A-1 through 4B-2 are flow diagrams which are useful in understanding the detailed operations of the system depicted in Fig. 1.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

Fig. 1 depicts a general block diagram of a system in accordance with the invention for use in connection with a digital data processing system which assists in managing software use in accordance with software licenses. With reference to Fig. 1, the new system includes a license management facility 10 which operates in conjunction with a license data base 11 and license unit tables 12, and under control of an operating system 13 and licensing policy module 15 to control use of licensed programs, such as licensed program 14, so that the use is in accordance with the terms of the software license which controls the use of the software program on a system 16 identified by a system marketing model (SMM) code in a digital data processing system.

As is conventional, the digital data processing system including the licensing management system may include one or more systems 16, each including one or more processors, memories and input/output units, interconnected in a number of ways. For example, the digital data processing system may comprise one processor, which may include a central processor unit which controls the system and one or more auxiliary processors which assist the central processor unit. Alternatively, the digital data processing system may comprise multiple processing systems, in which multiple central processor units are tightly coupled, or clustered or networked systems in which multiple central processor units are loosely coupled, generally operating relatively autonomously, interacting by means of messages transmitted over a cluster or network connection. In a tightly coupled multiple processing system, for example, it may be desirable to control the number of users which may use a particular software program at one time. A similar restriction may be obtained in a cluster or network environment by controlling the number of particular nodes, that is, connections to the communications link in the cluster or network over which messages are transferred. In addition, since the diverse processors which may be included in a digital data processing system may have diverse processing speeds and powers, represented by differing system marketing model (SMM) codes, it may be desirable to include a factor for speeds and power in determining the number of processors on which a program may be used concurrently.

As will be explained in greater detail below, the license data base 11 contains a plurality of entries 20 (described below in connection with Fig. 2) each containing information relating to the terms of the license for a particular licensed program 14. In one embodiment such information may include a termination date, if the license is for a particular time period or expires on a particular date, and a number of licensing units if the license is limited by usage of the license program. In that embodiment, the entry also includes identification of a license unit table 40 (described below in connection with Fig. 3) in the license unit tables 12 that identifies the number of allocation units for usage of the licensed program on the types of systems 16 which may be used in the digital data processing system as represented by the system marketing model (SMM) codes.

When a user wishes to use a licensed program 14, a GRANT LICENSE request message is generated which requests information as to the licensing status of the licensed program 14. The GRANT LICENSE request message is transmitted to the licensing policy module 15, which notifies the operating system of the request. The operating system 13, in turn, passes the request, along with the system marketing model of the specific system 16 being used by the user, to the license management facility 10 which determines whether use of the program is permitted under the license.

In response to the receipt of the GRANT LICENSE request from the user and the system marketing model (SMM) code of the system 16 being used by the user on which the licensed program will be processed, the license management facility 10 obtains from the license data base the entry 20 associated with the licensed program

14 and determines whether the use of the licensed program 14 is within the terms of the license as indicated by the information in the license data base 11 and the license unit tables 12.

In particular, the license management facility 10 retrieves the contents of the entry 20 associated with the licensed program. If the entry 20 indicates a termination data, the license management facility 10 compares the system data, which is maintained by the digital data processing system in a conventional manner, with the termination date identified in the entry. If the system date is after the termination date identified in the entry 20, the license has expired and the license management facility 10 generates a usage disapproved message, which it transmits to the operating system 13. On the other hand, if the termination date indicated in the entry 20 is after the system date, the license has not expired and the license management facility 10 proceeds to determine whether the usage of the licensed program 14 is permitted under other terms of the license which may be embodied in the entry 20.

In particular, the license management facility 10 then determines whether the usage of the licensed program is permitted under usage limitations. In that operation, the license management facility obtains the number of license units remaining, which indicates usage of the licensed program 14 not including the usage requested by the user, as well the identification of the table 40 in license unit tables 12 associated with the licensed program 14. The license management facility 10 then compares the number of license units which would be allocated for use of the licensed program 14, which it obtains from the table 40 identified by entry 20 in the license data base 11, and the number of remaining units to determine whether sufficient license units remain to permit usage of the licensed program 14.

If the number of remaining license units indicated by entry 20 in the license data base 11 exceeds the number, from license unit tables 12, of license units which would be allocated for use of the licensed progam 14, the usage of the licensed program is permitted under the license. Accordingly, the license management facility transmits a usage approved response to the operating system 13. In addition, the license management facility 10 adjusts the number of remaining license units in entry 20 by a function of the license units allocated to use of the licensed program to reflect the usage.

On the other hand, if the number of remaining license units indicated by entry 20 in the license data base is less than the number of license units which would be allocated for use of the licensed program 14, the usage of the licensed program 14 is not permitted by the license. In that case, the license management facility 10 transmits a usage disapproved response to the operating system 13. In addition, the license management facility 10 may also log the usage disapproved response; this information may be used by a system operator to determine whether usage of the licensed program 14 is such as to warrant obtaining an enlarged license.

Upon receipt of either a usage approved response or a usage disapproved response to the GRANT LICENSE request, the operating system 13 passes the response to the licensing policy module 15. If a usage approved response is received, the licensing policy module normally allows usage of the licensed program 14. If a usage disapproved response is received, the licensing policy module determines whether the usage of the licensed program may be permitted for other reasons. For example, usage of the licensed program 14 may be permitted under a group license, whose terms are embodied in entries in group license tables 17. Under a group license, usage may be permitted of any of a group of licensed programs. The operations to determine to whether usage is permitted may be performed in the same manner as described above in connection with license management facility 10. In addition, if the usage of the licensed program 14 is not permitted under a group license, usage may nonetheless be permitted under the licensor's licensing practices, which may be embodied in the licensing policy module 15. If the licensing policy module determines that usage of the program should be permitted, notwithstanding a usage disapproved response from the license management facility 10, because the usage is permitted under a group license or the licensor's licensing practices, the licensing policy module 15 permits usage of the licensed program. Otherwise, the licensing policy module does not permit usage of the licensed program in response to the GRANT LICENSE request.

When a user no longer requires use of a licensed program 14, it transmits a RELEASE LICENSE request to the licensing policy module 15. The operations performed by the licensing policy module depend on the basis for permitting usage of the licensed program. If usage was permitted as a result of a group license, if the group license is limited by usage, the licensing policy module 15, if necessary, adjusts the records in the group license tables 17 related to the group license to reflect the fact that the licensed program 14 related to the group license is not being used. If the usage was permitted as a result of a group license which is not limited by usage, but instead is limited in duration, or if the usage was permitted in response to the licensor's licensing policies, the licensing policy module 15 need do nothing. If the licensing

policy module 15 maintains a log of usage outside the scope of a group or program license, it may make an entry in the log of the RELEASE request.

Finally, if usage was permitted as a result of the license management facility 10 providing an approve usage response to the GRANT LICENSE request, the licensing policy module 15 transmits the RELEASE LICENSE request to the operating system 13. In response, the operating system 13 transfers the RELEASE LICENSE request to the license management facility 10, along with an identification of the system 16 using the licensed program 14. The license management facility 10 then obtains from the license data base the identification of the appropriate license usage allocation unit value table in license unit tables 12, and determines the number of allocation units associated with this use of the licensed program 14 based on the identified allocation table and the processor. The license management facility 10 then adjusts the number of license units for the licensed program 14 in the license data base 11 to reflect the release.

It will be appreciated by those skilled in the art that, the license management facility 10 may, in response to a GRANT LICENSE request, instead of deducting allocation units from the entries in the license data base 11 associated with the licensed programs 14, determine the number of allocation units which would be in use if usage of the licensed program 14 is permitted, and respond based on that determination. If the license mangement facility 10 operates in that manner, it may be advantageous for the entries in license data base 11 relating to each licensed program 14 to maintain a running record of the number of allocation units associated with its usage. The licensing policy module 15 may operate similarly in connection with group licenses that are limited by usage.

It will also be appreciated that the new license management system thus permits the digital data processing system to control use of a licensed program 14 based on licensing criteria in the license data base 11, the license unit tables 12, the group licensing tables 17 and the licensor's general licensing policies rather than requiring an operator to limit or restrict use of a licensed program or charging for the license based on some function of the capacity of all of the processors in the digital data processing system. The new license management system allows for very flexible pricing of licenses and licensing policies, since the digital data processing system itself enforces the licensing terms controlling use of the licensed programs 14 in the system.

Fig. 2 depicts the detailed structure of the license data base 12 (Fig. 1) used in the license management system depicted in Fig. 1. With reference to Fig. 2, the license data base includes a plurality of entries generally identified by reference numberal 20, with each entry being associated with one licensed program 14. Each entry 20 includes a number of fields, including an issuer name field 21 identifying the issuer of the license, an authorization number field 22 which contains an authorization number, a producer name field 23 which identifies the name of the vendor of the licensed program, and a product name field 24 which contains the name of the licensed program. The contents of these fields may be used, for example, in connection with other license management operations, such as determining the source of licensed programs in the event of detection of errors in programs, and in locating duplicate entries in the license data base or entries which may be combined as a result of licenses being obtained and entered by, perhaps different operators or at different times.

Each entry 20 in the licensing data base 11 also includes a license number field 25 whose contents identify the number of licensing units remaining. A license of a licensed program 14 identifies a number of licensing units, which may be a function of the price paid for the license. An availability table field 26 and an activity table field 27 identify license usage allocation unit value tables in the license unit tables 12 (described in connection with Fig. 3) to be used in connection with the GRANT LICENSE and RELEASE LICENSE requests.

By way of background, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program 14 on several processors to be a function of the processor power and capacity, and the availability table field 26 identifies a license usage allocation unit table to be used in connection with that. In an alternative, a license may be in accordance with a licensing paradigm which requires concurrent use of the licensed program to be a function of the number of users using the program, and the activity table field 27 identifies a license usage allocation unit valve table in the license unit tables 12 to be used in connection with that. If either licensing paradigm is used to the exclusion of the other, one field contains a non-zero value and the other field contains a zero value. In addition, a license may be in accordance with both licensing paradigms, that is, concurrent use of a program may be limited by both processor power and capacity and by the number of concurrent users, and in that case both fields 26 and 27 have non-zero values.

In one embodiment of the licensing management system, fields 21 through 27 of an entry 20 in the licensing data base 11 are required. In that embodiment, an entry 20 in the licensing data may

also have several optional fields. In particular, an entry 20 may include a date/version number field 30 whose contents comprise either a date or version number to identify the licensed program. If a license is to terminate on a specific date, the entry 20 may include a licensor termination date field 31 or a licensee termination date field 32 whose contents specify the termination date assigned by the licensor or licensee. This may be particularly useful, for example, as a mechanism for permitting licensees to demonstrate or try a program before committing to a long or open term license.

Finally, an entry 20 in the license data base includes a checksum field 33, which includes a checksum of the contents of the other fields 21 through 27 and 30 through 32 in the entry 20, which may be established by means of a mathematical algorithm applied to the contents of the various fields. The general mechanism for establishing checksums is well known in the art, and will not be described further herein. The contents of all fields 21 through 27 and 30 through 33 of a new entry 20 are entered by an operator. Prior to establishment of an entry in the license data base 11, the license management facility 10 may verify correct entry of the information in the various fields by calculating a checksum and comparing it to the checksum provided by the operator. If the checksum provided by the operator and the checksum determined by the license management facility are the same, the entry 20 is established in the license data base 11. On the other hand, if the checksum provided by the operator and the checksum determined by the license management facility differ, the license management facility 10 determines that the information is erroneous or the license is invalid and does not establish the entry 20 in the license data base 11. It will be appreciated that, if the checksum-generation algorithm is hidden from an operator, the checksum provides a mechanism for verifying, not only that the information has been properly loaded into the entry, but also that the license upon which the entry is based is authorized by the licensor.

The structure of group license tables 17 may be similar to the structure of the license data base 11, with the addition that the entries for each license reflected in the group license tables 17 will need to identify all of the licensed programs covered thereby.

As described above, the licensing unit tables 12 (Fig. 1) contain information as to the allocation units for use in determining the number of licensing units associated with use of a licensed program. The structure of a licensing unit table 40 is depicted in Fig. 3. With reference to Fig. 3, the licensing unit table includes a plurality of entries 41(1) through 41(N) (generally identified by reference numeral 41) each identified by a particular type of processor. One entry 41 in the table 40 is provided for each type of processor which can be included in the digital data processing system which can use the licensed programs 14 which reference the license unit table 40. The processor associated with each entry is identified by a processor identification field 42. The successive fields in the entries 41 (which form the various columns in the table 40 depicted in Fig. 3) form license usage allocation unit value tables 43(1) through 43-(M) (generally identified by reference numeral 43). The contents of the availability table field 26 and the activity table field 27 identify a license usage allocation unit value table 43. If there are non-zero contents in both availability field 26 and activity field 27, the contents which identify be the same license usage allocation unit value table 43 or different license usage allocation unit value tables 43. As described above, the contents of the license usage allocation unit value table identify the number of licensing units associated with use of the licensed programs which identify the particular license usage allocation unit value table, for each of the identified processors.

The operation of the licensing management system is depicted in detail in Figs. 4A-1 through 4-B. Figs. 4A-1 through 4A-4 depict, in a number of steps the details of operation of the licensing management system in connection with the GRANT LICENSE request from a licensed program 14. Figs. 4B-1 and 4B-2 depict, in a number of steps, the details of operation in connection with the RELEASE LICENSE request from a licensed program 14. In the Figs., the particular steps performed by the licensing policy module 15, the license management facility 10 and the operating system 13 are indicated in the respective steps. Since the operations depicted in Figs. 4A-1 through 4B-2 are substantially as described above in connection with Fig. 1, they will not be described further herein.

The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

Claims

1. A license management system for managing usage of a licensed software program comprising:
licensing storage means for storing a licensing unit value identifying a number of licensing units asso-

ciated with the licensed software program;
usage allocation value storage means for storing a usage allocation value identifying a number of licensing units associated with a use of the licensed software program; and
licensing verification means responsive to a usage request to use said licensed software program for determining, based on the contents of said licensing storage means and said usage allocation value storage means, whether usage of said licensed software program is permitted and, if usage is permitted, for adjusting the contents of said licensing storage means by a value to the contents of said usage allocation value storage means.

2. A license management system as defined in claim 1 for use in a digital data processing system which generates a system date value, said licensing storage means includes a plurality of fields including a licensing unit storage field for storing said licensing unit number identifying value and a field identifying a termination date, said licensing verification means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

3. A license management system as defined in claim 1 for managing usage of plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value, said licensing verification means including:
request receiving means for receiving a usage request identifying a licensed software program;
licensing unit retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said licensing unit storage field from the entry of said licensing storage means whose program identification field identifies the licensed software program identified in said usage request; and
licensing unit processing means for determining, based on the contents of retrieved licensing unit storage field and said usage allocation value storage means, whether usage of said licensed software program is permitted and, is usage is permitted, for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

4. A license management system as defined in claim 3 for use in a digital data processing system which generates a system date value, each entry in said licensing storage means further including a termination date field identifying a termination date, said licensing unit processing means further determining whether usage of said licensed software program is permitted in response to a comparison of said system date and said termination date.

5. A license management system as defined in claim 3 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of the usage allocation table identified by the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using said retrieved usage allocation table in its determination.

6. A license management system as defined in claim 5 wherein a request message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its determination.

7. A license management system as defined in claim 3 wherein a request message further includes licensing usage allocation value selection criteria and said usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation selection criterion, said licensing verification means including means for retrieving the usage allocation value associated with the licensing usage allocation selection criterion in said request message and using said retrieved usage allocation value in its determination.

8. A license management system as defined in claim 1 wherein said licensing verification means further operates in response to a release request message for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

9. A license management system as defined in claim 8 for managing usage of a plurality of licensed software programs, wherein said licensing storage means includes a plurality of entries each containing a program identification field identifying a licensed software program and a licensing unit storage field for storing said licensing unit value,

said licensing verification means including:
request receiving means for receiving a release request identifying a licensed software program;
licensing unit processing means for adjusting the contents of said licensing storage means by a value related to the contents of said usage allocation value storage means.

10. A license management system as defined in claim 9 wherein said usage allocation value storage means includes a plurality of usage allocation tables each storing a value identifying a number of licensing units, each entry in said licensing storage means further including a usage allocation table identification field identifying a usage allocation table, said licensing verification means further including usage allocation table retrieval means responsive to said request receiving means receipt of a usage request for retrieving the contents of said usage allocation table identification field of said retrieved entry, said licensing unit processing means using retrieved usage allocation table in its adjusting.

11. A license management system as defined in claim 10 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

12. A license management system as defined in claim 8 wherein a release message further includes licensing usage allocation value selection criteria and each usage allocation table includes a plurality of entries each identifying a usage allocation value associated with a licensing usage allocation value selection criterion, said licensing verification means including means for retrieving, from the usage allocation value table identified by said entry in said licensing storage means, the usage allocation value associated with the licensing usage allocation value selection criterion in said request message and using said retrieved usage allocation value in its adjusting.

13. A license management system for use in a digital data processing system including a system date generating means for generating a system date value, said license management system comprising:
licensing storage means including a plurality of entries each associated with a licensed software program, each entry containing a licensing units field for storing a licensing unit value identifying a number of licensing units associated with the license software program, a usage allocation table, and a termination date;
usage allocation table storage means for storing a plurality of usage allocation tables, each usage allocation table having a plurality of usage allocation entries each usage allocation entry being associated with a licensing usage allocation value selection criterion and storing a usage allocation value identifying a number of licensing units; and
licensing verification means including:
usage grant means including:
usage request message receiving means for receiving a usage request message from a licensed software program, said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the received usage request message;
licensing request processing means including:
usage determination means including licensing unit comparing means for comparing the contents of said licensing units field and said usage allocation units field and date comparison means for comparing the system date value with the contents of said termination date field to determine whether usage of said licensed software program is permitted.
response generation means for generating a message in response to the determination by said usage determination means; and
licensing unit adjusting means for adjusting the contents of said licensing units field in response to a positive determination by said usage determination means;
usage release means including:
usage release message receiving means for receiving a usage request message from a licensed software program; said usage request message identifying said licensed software program and usage grant criteria;
entry retrieval means responsive to the receipt of a usage request message for retrieving from said licensing storage means the licensing table entry associated with said licensed software program;
usage allocation table retrieval means for retrieving from said usage allocation table storage means a usage allocation entry identified by said retrieved licensing table entry and the licensing usage allocation value selection criterion identified by the

received usage request message;
licensing release processing means for adjusting the contents of said licensing units field in relation to the value of said usage allocation entry.

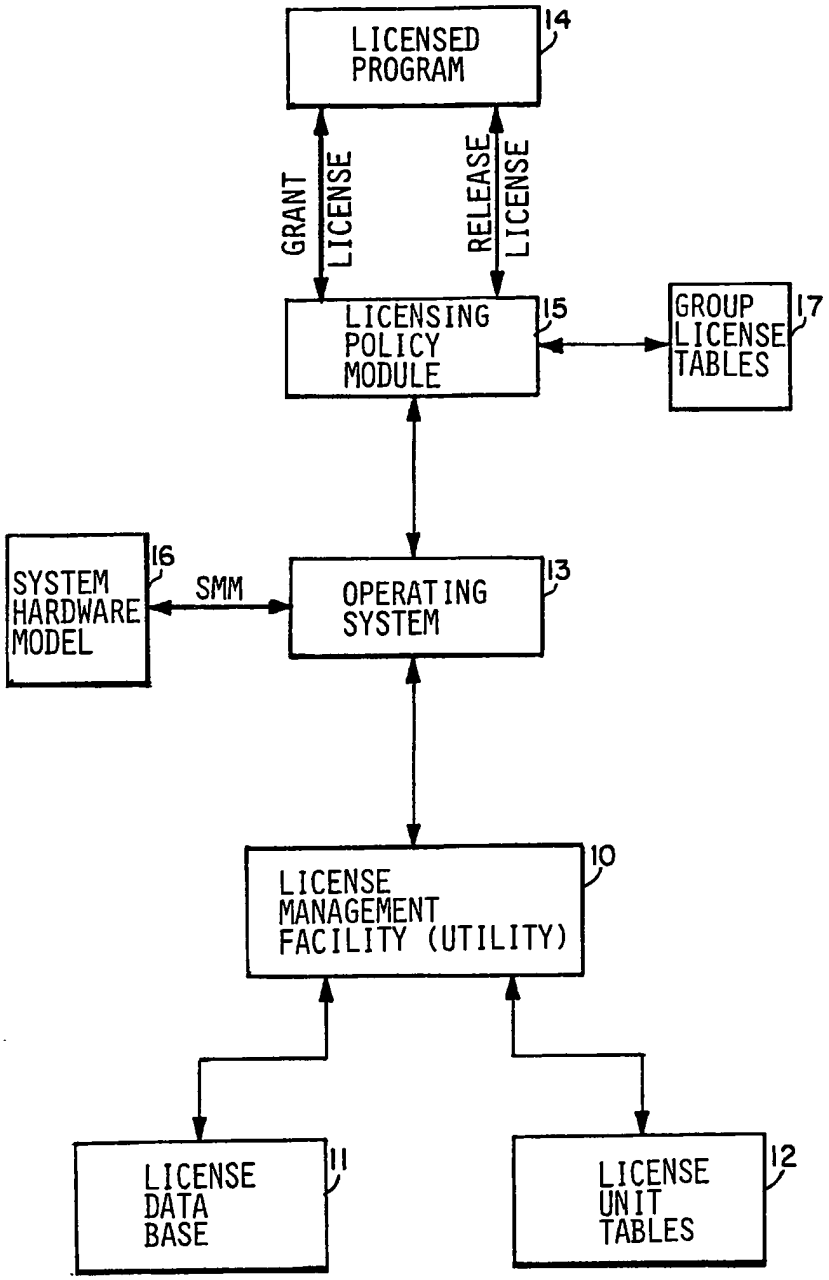


FIG. 1

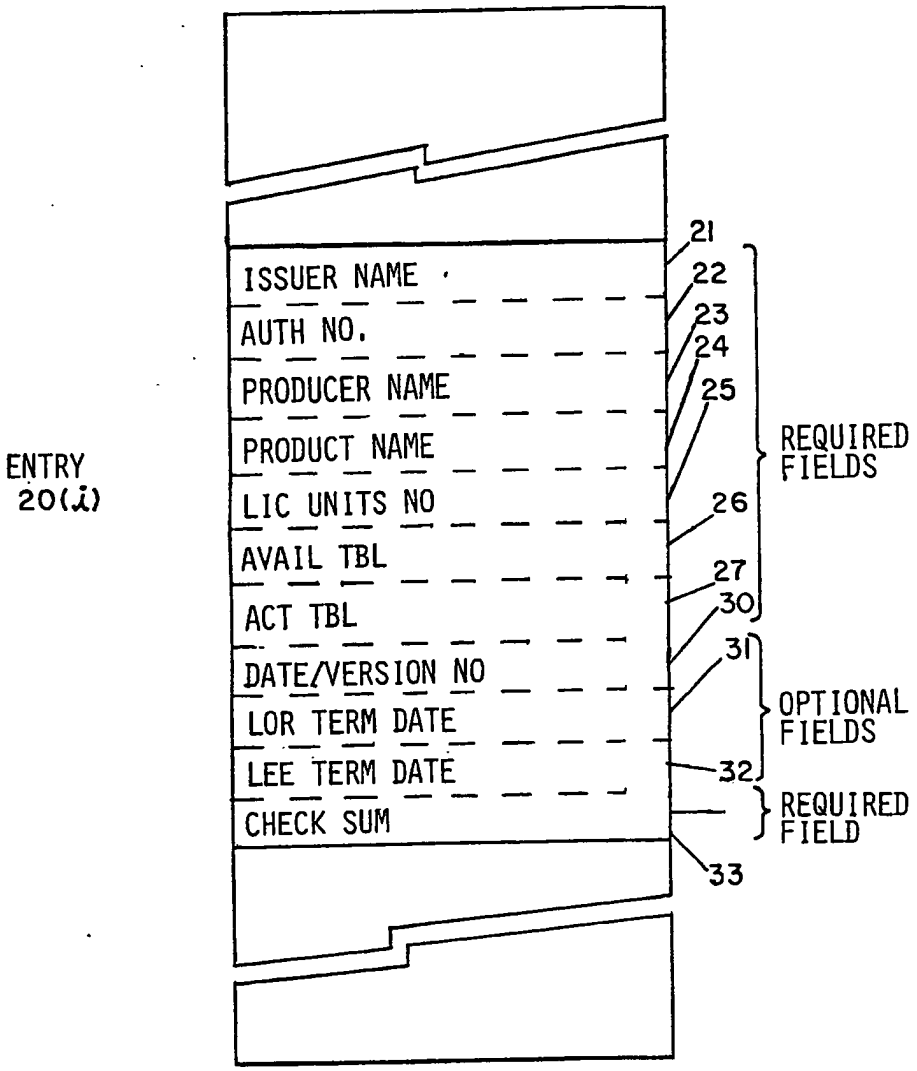
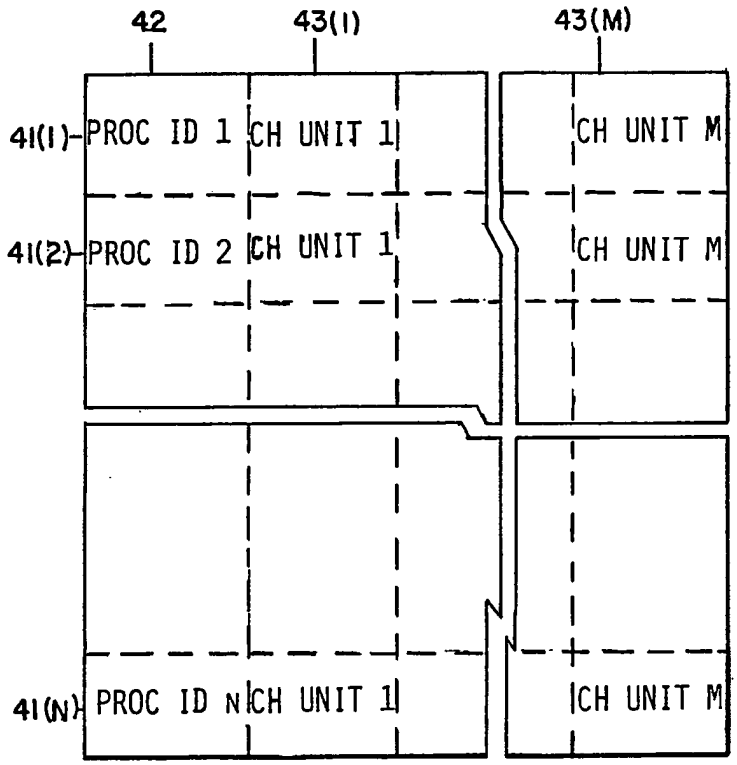


FIG. 2



LICENSE UNIT TABLE 40

FIG. 3

FIG. 4A-1

GRANT LICENSE

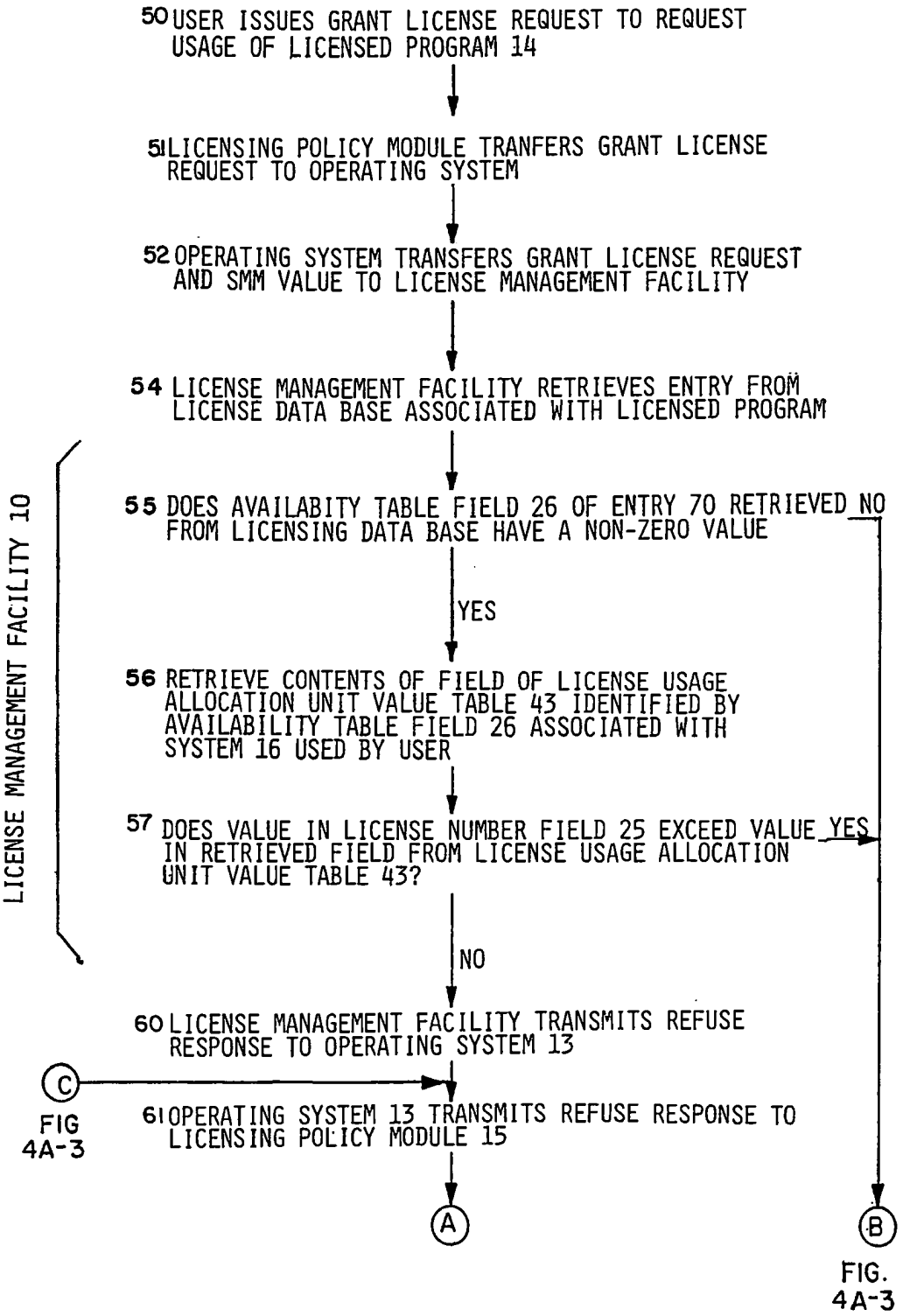


FIG. 4A-2

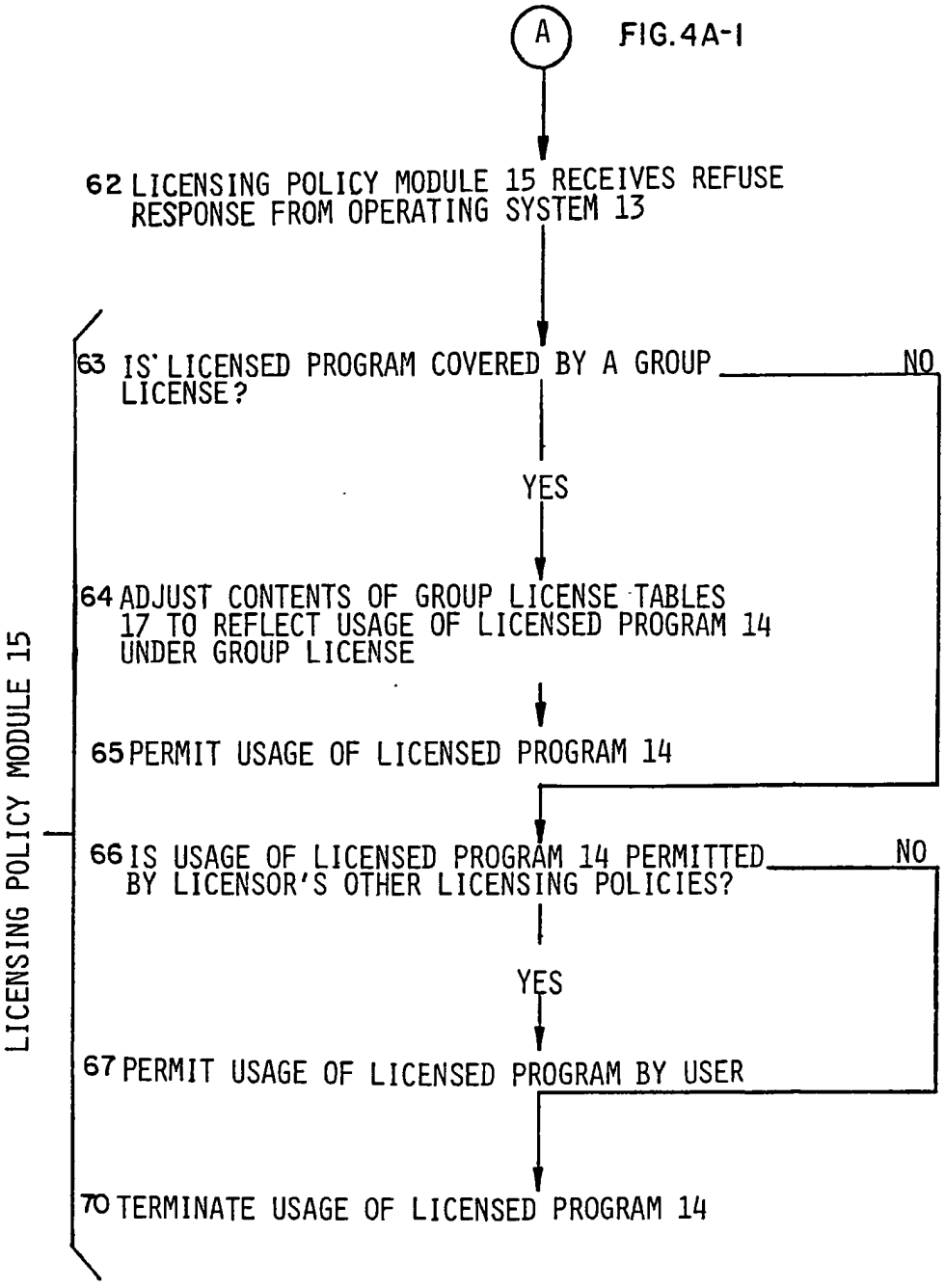


FIG. 4A-3

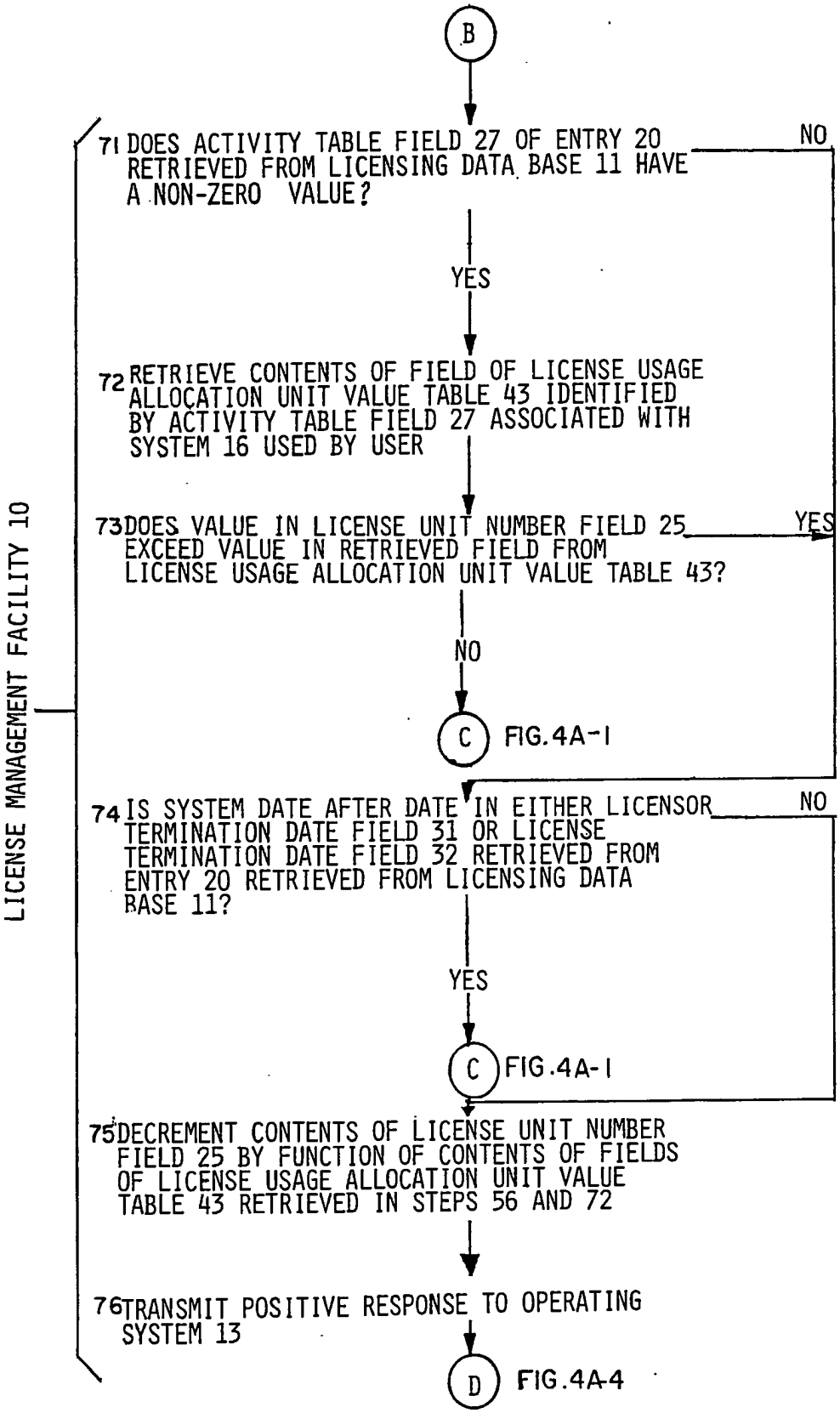

B
71 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
72 RETRIEVE CONTENTS OF FIELD OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 IDENTIFIED BY ACTIVITY TABLE FIELD 27 ASSOCIATED WITH SYSTEM 16 USED BY USER
73 DOES VALUE IN LICENSE UNIT NUMBER FIELD 25 EXCEED VALUE IN RETRIEVED FIELD FROM LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43?
YES
NO
C
FIG.4A-1
74 IS SYSTEM DATE AFTER DATE IN EITHER LICENSOR TERMINATION DATE FIELD 31 OR LICENSE TERMINATION DATE FIELD 32 RETRIEVED FROM ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11?
NO
YES
C
FIG.4A-1
75 DECREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF CONTENTS OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 RETRIEVED IN STEPS 56 AND 72
76 TRANSMIT POSITIVE RESPONSE TO OPERATING SYSTEM 13
D
FIG.4A-4
LICENSE MANAGEMENT FACILITY 10

FIG. 4A-4

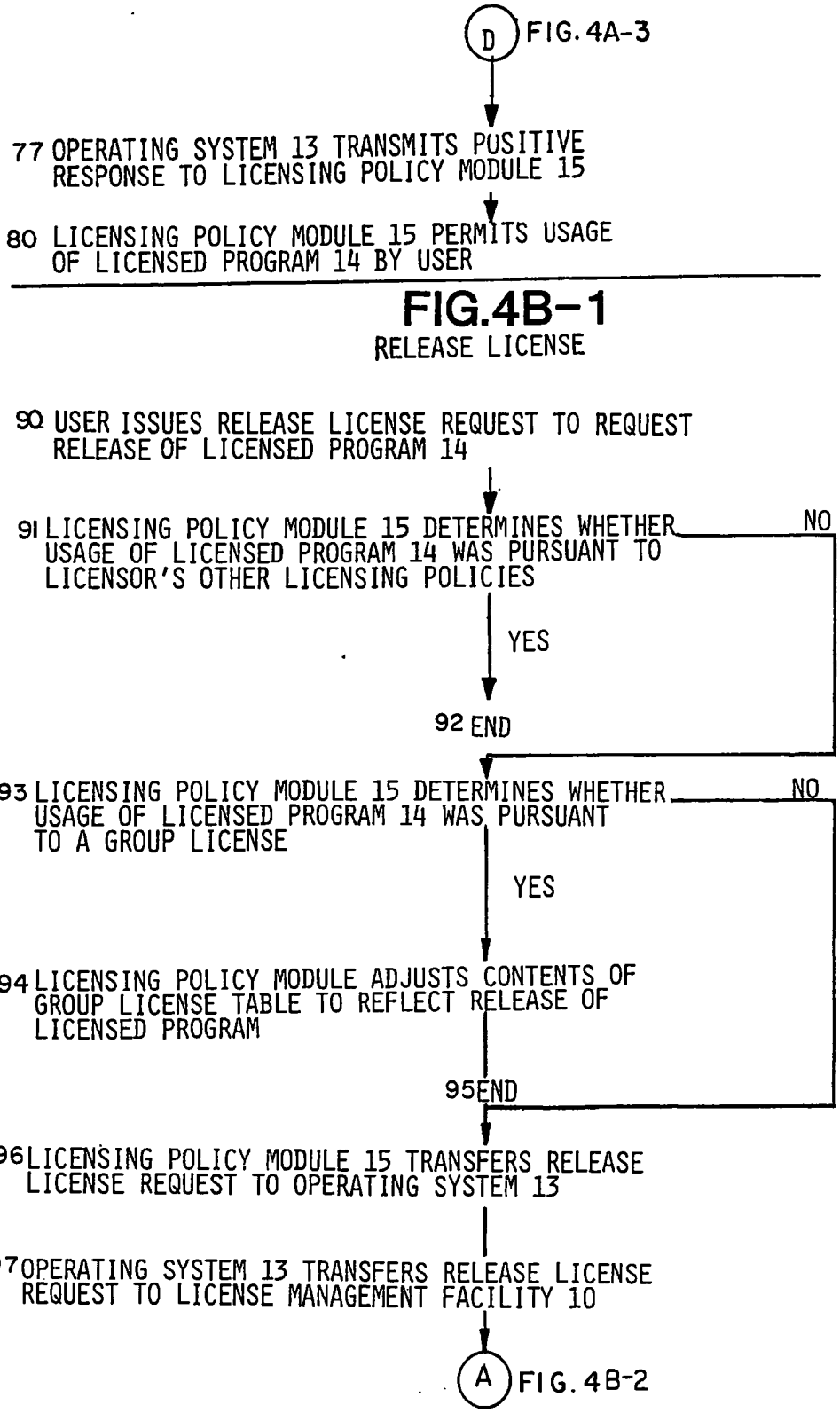
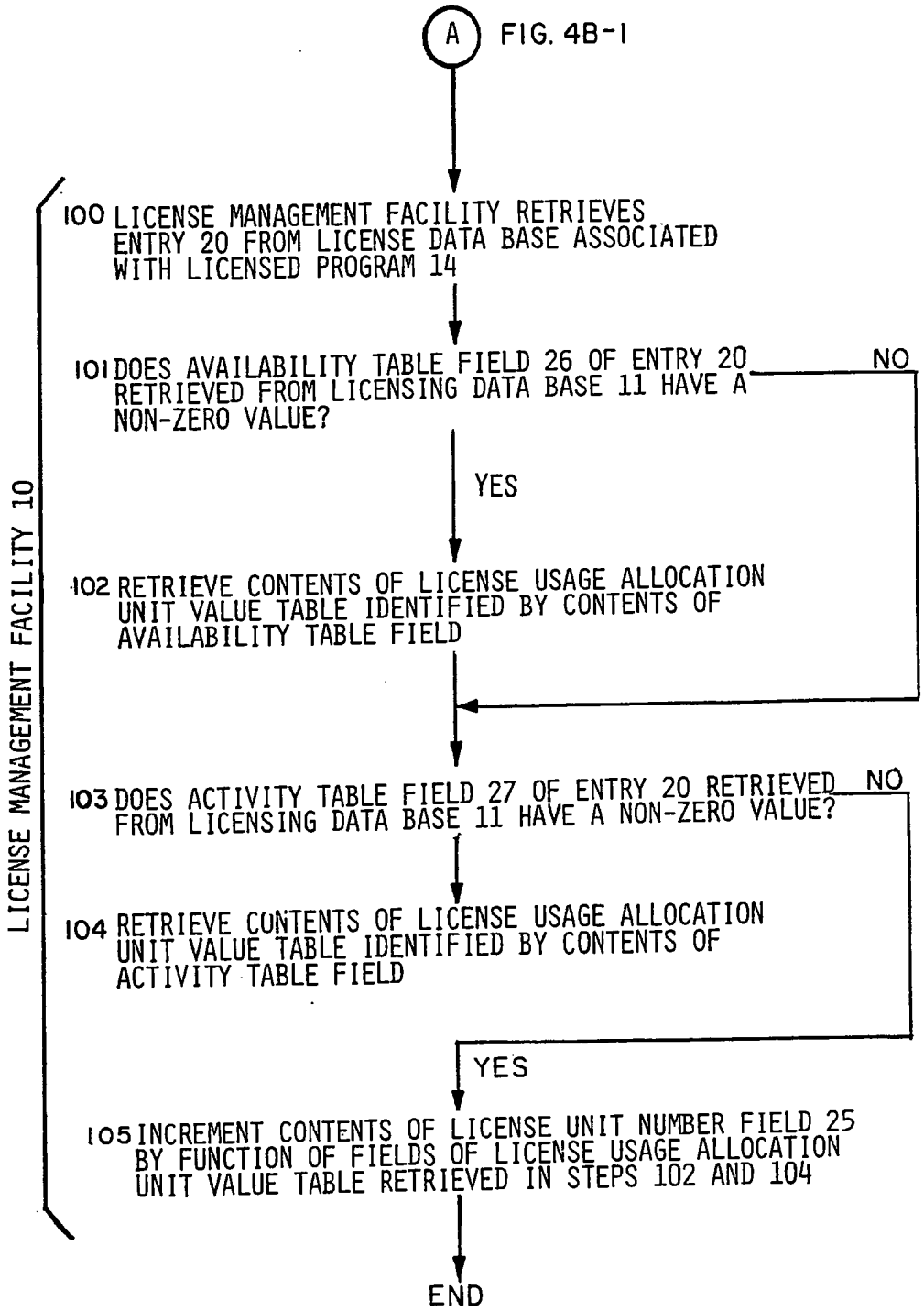


FIG.4B-1

RELEASE LICENSE

FIG. 4B-2


A
FIG. 4B-1
LICENSE MANAGEMENT FACILITY 10
100 LICENSE MANAGEMENT FACILITY RETRIEVES ENTRY 20 FROM LICENSE DATA BASE ASSOCIATED WITH LICENSED PROGRAM 14
101 DOES AVAILABILITY TABLE FIELD 26 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
102 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF AVAILABILITY TABLE FIELD
103 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
104 RETRIEVE CONTENTS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE IDENTIFIED BY CONTENTS OF ACTIVITY TABLE FIELD
YES
105 INCREMENT CONTENTS OF LICENSE UNIT NUMBER FIELD 25 BY FUNCTION OF FIELDS OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE RETRIEVED IN STEPS 102 AND 104
END

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 393 806 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90300115.4**

(51) Int. Cl.5: **H04L 9/32, H04L 9/08**

(22) Date of filing: **05.01.90**

(30) Priority: **17.04.89 US 339555**

(43) Date of publication of application:
24.10.90 Bulletin 90/43

(84) Designated Contracting States:
DE FR GB IT

(71) Applicant: **TRW INC.**
1900 Richmond Road
Cleveland Ohio 44124(US)

(72) Inventor: **Goss, Kenneth C.**
1470 Island Court
Oceano California 93445-9464(US)

(74) Representative: **Allden, Thomas Stanley et al**
A.A. THORNTON & CO. Northumberland
House 303-306 High Holborn
London WC1V 7LE(GB)

(54) Cryptographic method and apparatus for public key exchange with authentication.

(57) A technique for use in a public key exchange cryptographic system, in which two user devices establish a common session key by exchanging information over an insecure communication channel, and in which each user can authenticate the identity of the other, without the need for a key distribution center. Each device has a previously stored unique random number Xi, and a previously stored composite quantity that is formed by transforming Xi to Yi using a transformation of which the inverse in computationally infeasible; then concatenating Yi with a publicly known device identifier, and digitally signing the quantity. Before a communication session is established, two user devices exchange their signed composite quantities, transform them to unsigned form, and authenticate the identity of the other user. Then each device generates the same session key by transforming the received Y value with its own X value. For further security, each device also generates another random number X'i, which is transformed to a corresponding number Y'i. These Y'i values are also exchanged, and the session key is generated in each device, using a transformation that involves the device's own Xi and X'i numbers and the Yi and Y'i numbers received from the other device.

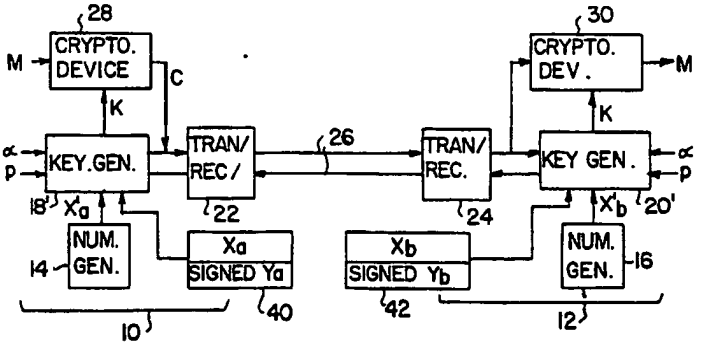


FIG. 3

EP 0 393 806 A2

BACKGROUND OF THE INVENTION

This invention relates generally to cryptographic systems and, more particularly, to cryptographic systems in which an exchange of information on an unsecured communications channel is used to establish a common cipher key for encryption and decryption of subsequently transmitted messages. Cryptographic systems are used in a variety of applications requiring the secure transmission of information from one point to another in a communications network. Secure transmission may be needed between computers, telephones, facsimile machines, or other devices. The principal goal of encryption is the same in each case: to render the communicated data secure from unauthorized eavesdropping.

By way of definition, "plaintext" is used to refer to a message before processing by a cryptographic system. "Ciphertext" is the form that the message takes during transmission over a communications channel. "Encryption" or "encipherment" is the process of transformation from plaintext to ciphertext. "Decryption" or "decipherment" is the process of transformation from ciphertext to plaintext. Both encryption and decryption are controlled by a "cipher key" or keys. Without knowledge of the encryption key, a message cannot be encrypted, even with knowledge of the encrypting process. Similarly, without knowledge of the decryption key, the message cannot be decrypted, even with knowledge of the decrypting process.

More specifically, a cryptographic system can be thought of as having an enciphering transformation E_k, which is defined by an enciphering algorithm E that is used in all enciphering operations, and a key K that distinguishes E_k from other operations using the algorithm E. The transformation E_k encrypts a plaintext message M into an encrypted message, or ciphertext C. Similarly, the decryption is performed by a transformation D_k defined by a decryption algorithm D and a key K.

Dorothy E.R. Denning, in "Cryptography and Data Security," Addison-Wesley Publishing Co. 1983, suggests that, for complete secrecy of the transmitted message, two requirements have to be met. The first is that it should be computationally infeasible for anyone to systematically determine the deciphering transformation D_k from intercepted ciphertext C, even if the corresponding plaintext M is known. The second is that it should be computationally infeasible to systematically determine plaintext M from intercepted ciphertext C. Another goal of cryptography systems is that of data authenticity. This requires that someone should not be able to substitute false ciphertext C' for ciphertext C without detection.

By way of further background, cryptographic systems may be classified as either "symmetric" or "asymmetric." In symmetric systems, the enciphering and deciphering keys are either the same easily determined from each other. When two parties wish to communicate through a symmetric cryptographic system, they must first agree on a key, and the key must be transferred from one party to the other by some secure means. This usually requires that keys be agreed upon in advance, perhaps to be changed on an agreed timetable, and transmitted by courier or some other secured method. Once the keys are known to the parties, the exchange of messages can proceed through the cryptographic system.

An asymmetric cryptosystem is one in which the enciphering and deciphering keys differ in such a way that at least one key is computationally infeasible to determine from the other. Thus, one of the transformations E_k or D_k can be revealed without endangering the other.

In 1976, the concept of a "public key" encryption system was introduced by W. Diffie and M. Hellman, "New Directions in Cryptography," IEEE Trans. on Info. Theory, Vol. IT-22(6), pp. 644-54 (Nov. 1976). In a public key system, each user has a public key and private key, and two users can communicate knowing only each other's public keys. This permits the establishment of a secured communication channel between two users without having to exchange "secret" keys before the communication can begin. As pointed out in the previously cited text by Denning, a public key system can be operated to provide secrecy by using a private key for decryption; authenticity by using a private key for encryption; or both, by using two sets of encryptions and decryptions.

In general, asymmetric cryptographic systems require more computational "energy" for encryption and decryption than symmetric systems. Therefore, a common development has been a hybrid system in which an asymmetric system, such as a public key system, is first used to establish a "session key" for use between two parties wishing to communicate. Then this common session key is used in a conventional symmetric cryptographic system to transmit messages from one user to the other. Diffie and Hellman have proposed such a public key system for the exchange of keys on an unsecured communications channel. However, as will be described, the Diffie-Hellman public key system is subject to active eavesdropping. That is to say, it provides no foolproof authentication of its messages. With knowledge of the public keys, an eavesdropper can decrypt received ciphertext, and then re-encrypt the resulting plaintext for transmission to the intended receiver, who has no way of knowing that

the message has been intercepted. The present invention relates to a significant improvement in techniques for public key exchange or public key management.

One possible solution to the authentication problem in public key management, is to establish a key distribution center, which issues secret keys to authorized users. The center provides the basis for identity authentication of transmitted messages. In one typical technique, a user wishing to transmit to another user sends his and the other user's identities to the center; e.g. (A,B). The center sends to A the ciphertext message $E_A(B,K,T,C)$, where E_A is the enciphering transformation derived from A's private key, K is the session key, T is the current date and time, and $C = E_B(A,K,T)$, where E_B is the enciphering transformation derived from B's private key. Then A sends to B the message C. Thus A can send to B the session key K encrypted with B's private key; yet A has no knowledge of B's private key. Moreover, B can verify that the message truly came from A, and both parties have the time code for further message identity authentication. The difficulty, of course, is that a central facility must be established as a repository of private keys, and it must be administered by some entity that is trusted by all users. This difficulty is almost impossible to overcome in some applications, and there is, therefore, a significant need for an alternative approach to public key management. The present invention fulfills this need.

Although the present invention has general application in many areas of communication employing public key management and exchange, the invention was first developed to satisfy a specific need in communication by facsimile (FAX) machines. As is now well known, FAX machines transmit and receive graphic images over ordinary telephone networks, by first reducing the images to digital codes, which are then transmitted, after appropriate modulation, over the telephone lines. FAX machines are being used at a rapidly increasing rate for the transmission of business information, much of which is of a confidential nature, over lines that are unsecured. There is a substantial risk of loss of the confidentiality of this imformation, either by deliberate eavesdropping, or by accidental transmission to an incorrectly dialed telephone number.

Ideally, what is needed is an encrypting/decrypting box connectable between the FAX machine and the telephone line, such that secured communications can take place between two similarly equipped users, with complete secrecy of data, and identity authentication between the users. For most users, a prior exchange of secret keys would be so inconvenient that they could just as well exchange the message itself by the same secret technique. A public key exchange system is by far the most convenient solution but each available variation of these systems has its own problems, as discussed above. The Diffie-Hellman approach lacks the means to properly authenticate a message, and although a key distribution center would solve this problem, as a practical matter no such center exists for FAX machine users, and none is likely to be established in the near future. Accordingly, one aspect of the present invention is a key management technique that is directly applicable to data transmission using FAX machines.

SUMMARY OF THE INVENTION

The present invention resides in a public key cryptographic system that accomplishes both secrecy and identity authentication, without the need for a key distribution center or other public facility, and without the need for double encryption and double decryption of messages. Basically, the invention achieves these goals by using a digitally signed composite quantity that is pre-stored in each user communication device. In contrast with the conventional Diffie-Hellman technique, in which random numbers Xi are selected for each communication session, the present invention requires that a unique number Xi be preselected and pre-stored in each device that is manufactured. Also stored in the device is the signed composite of a Yi value and a publicly known device identifier. The Yi value is obtained by a transformation from the Xi value, using a transformation that is practically irreversible.

Before secure communications are established, two devices exchange these digitally signed quantities, which may then be easily transformed into unsigned form. The resulting identifier information is used to authenticate the other user's identity, and the resulting Yi value from the other device is used in a transformation with Xi to establish a session key. Thus the session key is established without fear of passive or active eavesdropping, and each user is assured of the other's identity before proceeding with the transfer of a message encrypted with the session key that has been established.

One way of defining the invention is in terms of a session key generator, comprising storage means for storing a number of a first type selected prior to placing the key generator in service, and a digitally signed composite quantity containing both a unique and publicly known identifier of the session key generator and a number of a second type obtained by a practically irreversible transformation of the

number of the first type. The session key generator has a first input connected to receive the number of the first type, and a second input connected to receive an input quantity transmitted over an insecure communications channel from another session key generator, the input quantity being digitally signed and containing both a publicly known identifier of the other session key generator and a number of the second type generated by a practically irreversible transformation of a number of the first type stored in the other session key generator. The session key generator also has a first output for transmitting the stored, digitally signed composite quantity over the insecure communications channel to the other session key generator, a second output, means for decoding the signed input quantity received at the second input, to obtain the identifier of the other session key generator and the received number of the second type, and means for generating a session key at the second output, by performing a practically irreversible transformation of the number of the second type received through the second input, using the number of the first type received through the first input.

For further security of the session key, the session key generator further includes a third input, connected to receive another number of the first type, generated randomly, and means for generating at the first output, for transmission with the digitally signed composite quantity, a number of the second type obtained by a practically irreversible transformation of the number of the first type received through the third input. The session key generator also includes means for receiving from the second input another number of the second type generated in and transmitted from the other session key generator. The means for generating a session key performs a practically irreversible transformation involving both numbers of the first type, received at the first and third inputs, and both numbers of the second type received at the second input, whereby a different session key may be generated for each message transmission session.

More specifically, the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters. The number of the second type received in the digitally signed composite quantity from the other session key generator is designated Yb, and the means for generating the session key performs the transformation $K = Yb^{Xa} \bmod p$.

When additional numbers $X'a$ and $X'b$ are also generated prior to transmission, the means for generating the session key performs the transformation $K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,

where $X'a$ is the number of the first type that is randomly generated, $Y'b$ is the additional number of the second type received from the other session key generator, and the $\oplus$ symbol means an exclusive OR operation.

In terms of a novel method, the invention comprises the steps of transmitting from each device a digitally signed composite quantity to the other device, the composite quantity including a publicly known device identifier IDa and a number Ya derived by a practically irreversible transformation of a secret number Xa that it is unique to the device, receiving a similarly structured digitally signed composite quantity from the other device, and transforming the received digitally signed composite quantity into an unsigned composite quantity containing a device identifier IDb of the other device and a number Yb that was derived by transformation from a secret number Xb that is unique to the other device. Then the method performs the steps of verifying the identity of the other device from the device identifier IDb, and generating a session key by performing a practically irreversible transformation involving the numbers Xa and Yb.

Ideally, the method also includes the steps of generating another number $X'a$ randomly prior to generation of a session key, transforming the number $X'a$ to a number $Y'a$ using a practically irreversible transformation, transmitting the number $Y'a$ to the other device, and receiving a number $Y'b$ from the other device. In this case, the step of generating a session key includes a practically irreversible transformation involving the numbers Xa, $X'a$, $Y'b$ and $Y'b$.

In particular, the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations, and the step of generating a session key includes the transformation
$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,
where $\oplus$ denotes an exclusive OR operation.

It will be appreciated from this brief summary that the present invention represents a significant advance in the field of cryptography. In particular, the invention provides for both secrecy and identity authenticity when exchanging transmissions with another user to establish a common session key. Other aspects and advantages of the invention will become apparent from the following more detailed description, taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram showing a public key cryptographic system of the prior art;

FIG. 2 is a block diagram similar to FIG. 1, and showing how active eavesdropping may be used to attack the system;

FIG. 3 is a block diagram of a public key cryptographic system in accordance with the present invention;

FIG. 4 is a block diagram of a secure facsimile system embodying the present invention; and

FIG. 5 is a block diagram showing more detail of the cryptographic processor of FIG. 4.

DESCRIPTION OF THE PREFERRED EMBODIMENT

As shown in the accompanying drawings for purposes of illustration, the present invention is concerned with a public key cryptographic system. As discussed at length in the preceding background section of this specification, public key systems have, prior to this invention, been unable to provide both secrecy and identity authentication of a message without either a costly double transformation at each end of the communications channel, or the use of key distribution center.

U.S. Patent No. 4,200,770 to Hellman et al. discloses a cryptographic apparatus and method in which two parties can converse by first both generating the same session key as a result of an exchange of messages over an insecure channel. Since the technique disclosed in the Hellman et al. '770 patent attempts to provide both secrecy and authentication in a public key cryptographic system, the principles of their technique will be summarized here. This should provide a better basis for an understanding of the present invention.

In accordance with the Hellman et al. technique, two numbers α and p are selected for use by all users of the system, and may be made public. For increased security, p is a large prime number, and α has a predefined mathematical relationship to p, but these restrictions are not important for purposes of this explanation. Before starting communication, two users, A and B, indicated in FIG. 1 at 10 and 12, perform an exchange of messages that results in their both computing the same cipher key, or session key K, to be used in transmitting data back and forth between them. The first step in establishing the session key is that each user generates a secret number in a random number generator 14, 16. The numbers are designated Xa, Xb, respectively, and are selected from a set of positive integers up to p-1. Each user also has a session key generator 18, 20, one function of which is to generate other numbers Y from the numbers X, α and p, using the transformations:

$Ya = \alpha^{Xa} \bmod p$,

$Yb = \alpha^{Xb} \bmod p$.

The values Ya, Yb are then processed through a conventional transmitter/receiver 22, 24, and exchanged over an insecure communications channel 26.

The term "mod p' means modulo p, or using modulo p arithmetic. Transforming an expression to modulo p can be made by dividing the expression by p and retaining only the remainder. For example, 34 mod 17 = 0, 35 mod 17 = 1, and so forth. Similarly, the expression for Ya may be computed by first computing the exponential expression α^{Xb}, then dividing the result by p and retaining only the remainder.

If α and p are appropriately chosen, it is computationally infeasible to compute Xa from Ya. That is to say, the cost of performing such a task, in terms of memory or computing time needed, is large enough to deter eavesdroppers. In any event, new X and Y values can be chosen for each message, which is short enough to preclude the possibility of any X value being computed from a corresponding Y value.

After the exchange of the values Ya, Yb, each user computes a session key K in its session key generator 18, 20, by raising the other user's Y value to the power represented by the user's own X value, all modulo p. For user A, the computation is:

$K = Yb^{Xa} \bmod p$.

Substituting for Yb,

$K = (\alpha^{Xb})^{Xa} \bmod p = \alpha^{XaXb} \bmod p$.

For user B, the computation is:

$K = Ya^{Xb} \bmod p$.

Substituting for Ya,

$K = (\alpha^{Xa})^{Xb} \bmod p = \alpha^{XaXb} \bmod p$.

The two users A, B now have the same session key K, which is input to a conventional cryptographic device 28, 30. A transmitting cryptographic device, e.g. 28, transforms a plaintext message M into ciphertext C for transmission on the communications channel 26, and a receiving cryptographic device 30 makes the inverse transformation back to the plaintext M.

The Hellman et al. 770 patent points out that the generation of a session key is secure from eavesdropping, because the information exchanged on the insecure channel includes only the Y values, from which the corresponding X values cannot be easily computed. However, this form of key exchange system still has two significant problems. One is that the system is vulnerable to attack from active eavesdropping, rather than the passive eavesdropping described in the patent. The other is that identity authentication can be provided only by means of a public key directory.

Active eavesdropping takes place when an unauthorized person places a substitute message on

the communications channel. FIG. 2 depicts an example of active eavesdropping using the same components as FIG. 1. The active eavesdropper E has broken the continuity of the unsecured line 26, and is receiving messages from A and relaying them to B, while sending appropriate responses to A as well. In effect, E is pretending to be B, with device Eb, and is also pretending to be A, with device Ea. E has two cryptographic devices 34a, 34b, two session key generators 36a, 36b, and two number generators 38a, 38b. When device Eb receives Ya from A, it generates Xb′ from number generator 38b, computes Yb′ from Xb′ and transmits Yb′ to A. Device Eb and user A compute the same session key and can begin communication of data. Similarly, device Ea and user B exchange Y numbers and both generate a session key, ifferent from the one used by A and Eb. Eavesdropper E is able to decrypt the ciphertext C into plaintext M, then encipher again for transmission to B. A and B are unaware that they are not communicating directly with each other.

In accordance with the present invention, each user is provided with proof of identity of the party with whom he is conversing, and both active and passive eavesdropping are rendered practically impossible. FIG. 3 shows the key management approach of the present invention, using the same reference numerals as FIGS. 1 and 2, except that the session key generators are referred to in FIG. 3 as 18′ and 20′, to indicate that the key generation function is different in the present invention. The user devices also include a number storage area 40, 42. Storage area 40 contains a preselected number Xa, stored at the time of manufacture of the A device, and another number referred to as "signed Ya," also stored at the time of manufacture. Xa was chosen at random, and is unique to the device. Ya was computed from Xa using the transformation

$Ya = \alpha^{Xa} \bmod p.$

Then the Ya value was concatenated with a number IDa uniquely identifying the user A device, such as a manufacturer's serial number, and then encoded in such a way that it was digitally "signed" by the manufacturer for purposes of authenticity. The techniques for digitally signing data are known in the cryptography art, and some will be discussed below. For the present, one need only consider that the number designated "signed (Ya,IDa)" contains the value Ya and another value IDa uniquely identifying the A device, all coded as a "signature" confirming that the number originated from the manufacturer and from no-one else. User B's device 12 has stored in its storage area 42 the values Xb and signed (Yb,IDb).

Users A and B exchange the signed (Ya,IDa) and signed (Yb,IDb) values, and each session key generator 18, 20 then "unsigns" the received values and verifies that it is conversing with the correct user device. The user identifiers IDa and IDb are known publicly, so user device A verifies that the number IDb is contained in the signed (Yb,IDb) number that was received. Likewise, user device B verifies that the value signed (Ya,IDa) contains the known value IDa. By performing the process of "unsigning" the received messages, the user devices also confirm that the signed data originated from the manufacturer and not from some other entity.

Since the Xa, Xb values are secret values, and it is infeasible to obtain them from the transmitted signed (Ya,IDa) and signed (Yb,IDb) values, the users may both compute identical session keys in a manner similar to that disclosed in the Hellman et al. '770 patent. If an eavesdropper E were to attempt to substitute fake messages for the exchanged ones, he would be unable to satisfy the authentication requirements. E could intercept a signed (Ya,IDa) transmission, could unsign the message and obtain the values Ya and IDa. E could similarly obtain the values Yb and IDb. However, in order for E and A to use the same session key, E would have to generate a value Xe, compute Ye and concatenate it with IDb, which is known, and then digitally "sign" the composite number in the same manner as the manufacturer. As will be explained, digital signing involves a transformation that is very easy to effect in one direction, the unsigning direction, but is computationally infeasible in the other, the signing direction. Therefore, eavesdropper E would be unable to establish a common session key with either A or B because he would be unable to generate messages that would satisfy the authentication requirements.

As described thus far, the technique of the invention establishes a session key that is derived from X and Y values stored in the devices at the time of manufacture. Ideally, a new session key should be established for each exchange of message traffic. An additional unsecured exchange is needed to accomplish this.

The number generator 14 in the A device 10 generates a random number X′a and the number generator 16 in the B device 12 generates a random number X′b. These are supplied to the session key generators 18, 20, respectively, which generate values Y′a and Y′b in accordance with the transformations:

$Y'a = \alpha^{X'a} \bmod p,$

$Y'b = \alpha^{X'b} \bmod p.$

These values are also exchanged between the A and B devices, at the same time that the values of signed (Ya,IDa) and signed (Yb,IDb) are exchanged. After the authenticity of the message has been confirmed, as described above, the session

key generators perform the following transformations to derive a session key. At the A device, the session key is computed as

$Ka = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p,$

and at the B device, the session key is computed as

$Kb = (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p,$

where "⊕" means an exclusive OR operation.

Thus the session key is computed at each device using one fixed number, i.e. fixed at manufacturing time, and one variable number, i.e. chosen at session time. The numbers are exclusive ORed together on a bit-by-bit basis. It can be shown that Ka = Kb by substituting for the Y values. Thus:

$Ka = (\alpha^{X'b})^{Xa} \bmod p \oplus (\alpha^{Xb})^{X'a} \bmod p$
$= (\alpha^{Xa})^{X'b} \bmod p \oplus (\alpha^{X'a})^{Xb} \bmod p$
$= (Ya)^{X'b} \bmod p \oplus (Y'a)^{Xb} \bmod p$
$= (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p$
$= Kb.$

This common session key satisfies secrecy and authentication requirements, and does not require double encryption-decryption or the use of a public key directory or key distribution center. The only requirement is that of a manufacturer who will undertake to supply devices that have unique device ID's and selected X values encoded into them. For a large corporation or other organization, this obligation could be assumed by the organization itself rather than the manufacturer. For example, a corporation might purchase a large number of communications devices and complete the manufacturing process by installing unique ID's, X values, and signed Y values in the units before distributing them to the users. This would relieve the manufacturer from the obligation.

The process described above uses parameters that must meet certain numerical restrictions. The length restrictions are to ensure sufficient security, and the other requirements are to ensure that each transformation using modulo arithmetic produces a unique transformed counterpart. First, the modulus p must be a strong prime number 512 bits long. A strong prime number is a prime number p that meets the additional requirement that (p-1)/2 has at least one large prime factor or is preferably itself a prime number. The base number must be a 512-bit random number that satisfies the relationships:

$\alpha^{(p-1)/2} \bmod p = p-1,$ and

$1 < \alpha < p-1.$

Finally, the values X and X' are chosen as 512-bit random numbers such that

$1 < X, X' < p-1.$

As indicated above, the process of authentication in the invention depends on the ability of the manufacturer, or the owner of multiple devices, to supply a signed Y value with each device that is distributed. A digital signature is a property of a message that is private to its originator. Basically, the signing process is effected by a transformation that is extremely difficult to perform, but the inverse transformation, the "unsigning," can be performed easily by every user. The present invention is not limited to the use of a particular digital signature technique.

One approach is to use an RSA public key signature technique. The RSA technique takes its name from the initial letters of its originators, Rivest, Shamir and Adleman, and is one of a class of encryption schemes known as exponentiation ciphers. An exponentiation cipher makes the transformation $C = P^e \bmod n$, where e and n constitute the enciphering key. The inverse transformation is accomplished by $P = C^d \bmod n$. With appropriate selection of n, d and e, the values of n and d can be made public without giving away the exponent e used in the encryption transformation. Therefore, a digital signature can be applied to data by performing the exponentiation transformation with a secret exponent e, and providing a public decryption exponent d, which, of course, will be effective to decrpyt only properly "signed" messages.

In the preferred embodiment of the present invention, another approach is used for digital signature, namely a modular square-root transformation. In the expression $x = m^2 \bmod n$, the number m is said to be the square root of x mod n, or the modular square root of x. If n is appropriately selected, the transformation is very difficult to perform in one direction. That is to say, it is very difficult to compute m from x, although easy to compute x from m. If the modulus n is selected to be the product of two large prime numbers, the inverse or square-root transformation can only be made if the factors of the modulus are known. Therefore, the modulus n is chosen as the product of two prime numbers, and the product is 1,024 bits long. Further, the factors must be different in length by a few bits. In the devices using the present invention, the value "signed (Ya,IDa)" is computed by first assembling or concatenating the codes to be signed. These are:

1. A numerical code IDa uniquely identifying the A device. In the present embodiment of the invention, this is a ten-digit (decimal) number encoded in ASCII format, but it could be in any desired format.

2. A number of ASCII numerical codes indicating a version number of the device. This may be used for device testing or analyzing problems relating to device incompatability.

3. The value Ya computed from the chosen value of Xa, encoded in binary form.

4. A random value added to the least-significant end of the composite message, and used to ensure that the composite message is a perfect

modular square.

The last element of the message is needed because of inherent properties of the modular squaring process. If one were to list all possible values of a modular square x, from 1 to n-1, and all corresponding values of the modular square root m, some of the values of x would have multiple possible values of m, but others of the values of x would have no corresponding values of m. The value added to the end of the message ensures that the number for which a modular square root is to be computed, is one that actually has a modular square root. A simple example should help make this clear.

Suppose the modulus n is 7849. It can be verified by calculator that a value x of 98 has four possible values of m in the range 1 to n-1: 7424, 1412, 6437 and 425, such that m^2 mod 7849 = 98. However, the x value 99 has no possible modular square root values m. If the composite message to be signed had a numerical value of 99, it would be necessary to add to it a value such as 1, making a new x value of 100, which has four possible square root values in the range 1 to n-1, namely 1326, 7839, 10 and 6523. In most instances, it does not matter which of these is picked by the modular square root process employed, since the squaring or "unsigning" process will always yield the composite message value 100 again. However, there are a few values of m that should be avoided for maximum security. If the x value is a perfect square in ordinary arithmetic (such as the number 100 in the example), two values of m that should be avoided are the square root of x by ordinary arithmetic (the number 10 in the example), and the number that is the difference between the modulus n and the ordinary-arithmetic square root of x (i.e. 7839 in the example). If a number fitting this definition is used as a signed message, the signature is subject to being "forged" without knowledge of the factors of n. Therefore, such numbers are avoided in assigning signatures, and each device can be easily designed to abort an exchange when the signed message takes the form of one of these avoided numbers.

When the modular square root process is used for digitally signing the composite data stored in each device, the "unsigning" process upon receipt of a signed composite message is simply the squaring of the message, modulo n. The value n is not made public, although it could be determined by close examination of one of the devices. Even with knowledge of the modulus n, however, the computation of the modular square root is computationally infeasible without knowledge of the factorization of n.

With a knowledge of the factorization of the modulus n, the computation of the modular square root becomes a feasible, although laborious task, which may be performed by any known computational method. It will be recalled that this process is performed prior to distribution of the devices embodying the invention, so computation time is not a critical factor.

It will be understood that the cryptographic technique of the invention may be implemented in any form that is convenient for a particular application. Modular arithmetic is now well understood by those working in the field, and may be implemented in hardware form in the manner described in the '770 Hellman et al. patent. More conveniently, off-the-shelf modular arithmetic devices are available for connection to conventional microprocessor hardware. For example, part number CY1024 manufactured by CYLINK, of Sunnyvale, California 94087, performs modular addition, multiplication and exponentiation.

For application to facsimile communications, the technique of the invention may be made completely "transparent" to the user. FIG. 4 shows the architecture of a device for connection between a conventional FAX machine 50 and a telephone line 52. The device includes a first conventional modem 54 (modulator/demodulator) for connection to the FAX machine 50 and a second modem 56 for connection to the telephone line 52. The modems 54, 56 function to demodulate all messages entering the device from either the FAX machine or the telephone line, and to modulate messages for transmission to the FAX machine or onto the telephone line. The device further includes a communications processor 58 connected between the two modems 54, 56, and a cryptographic processor 60 connected to the communications processor 58. The communications processor 58 manages message traffic flow to and from the modems 54, 56 and to and from the cryptographic processor 60, and ensures that the necessary communications protocols are complied with. In one preferred embodiment of the invention, the communications processor is a microprocessor specified by part number MC68000, manufactured by Motorola Corporation.

As shown in FIG. 5, the cryptographic processor 60 includes a conventional microprocessor 62 having a data bus 64 and a data bus 66, to which various other modules are connected. The microprocessor 62 may be, for example, a National Semiconductor Company device specified by part number NSC800. The connected modules include a random access memory (RAM) 68, a read-only memory (ROM) 70, which serves as a storage area for the X value and the signed Y value, an integrated-circuit chip 72 for implementation of the Data Encryption Standard (DES), a modular arithmetic device 74 such as the CYLINK CY1024,

and an interface module 76 in the form of a dual-port RAM, for connection to the communications processor 58.

For transparent operation of the device shown in FIGS. 4 and 5, a user supplies not only the telephone number of a destination FAX machine, but also the ID of the intended destination FAX encoding/decoding device. When the digitally signed Y values are exchanged, the sending user device automatically "unsigns" the transmission by performing a modular squaring function; then compares the intended destination ID with the user ID returned with the Y value, and aborts the session if there is not a match. The key management steps previously described proceed automatically under control of the cryptographic processor 60, and when a session key has been derived, this is automatically applied in a conventional cryptographic process, such as the DES, to encrypt and decrypt a facsimile transmission.

It will be appreciated from the foregoing that the present invention represents a significant advance in cryptographic systems. In particular, the invention provides a technique for establishing a common session key for two users by means of an exchange of messages over an insecure communications channel. What distinguishes the invention from prior approaches to public key exchange systems is that the technique of the invention provides for identity authentication of the users without the need for a key distribution center or a public key register. Further, the technique is resistant to both passive and active eavesdropping. It will also be appreciated that, although an embodiment of the invention has been described in detail for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention. Accordingly, the invention is not to be limited except as by the appended claims.

Claims

1. A secure key generator, comprising:
storage means for storing a number of a first type selected prior to placing the key generator in service, and a digitally signed composite quantity containing both a unique and publicly known identifier of the key generator and a number of a second type obtained by a practically irreversible transformation of the number of the first type;
a first input connected to receive the number of the first type;
a second input connected to receive an input quantity transmitted over an insecure communications channel from another key generator, the input quantity being digitally signed and containing both a publicly known identifier of the other key generator and a number of the second type generated by a practically irreversible transformation of a number of the first type stored in the other key generator;
a first output for transmitting the stored, digitally signed composite quantity over the insecure communications channel to the other key generator;
a second output;
means for decoding the signed input quantity received at the second input, to obtain the identifier of the other key generator and the received number of the second type; and
means for generating a session key at the second output, by performing a practically irreversible transformation of the number of the second type received through the second input, using the number of the first type received through the first input.

2. A secure key generator as defined in claim 1, wherein the key generator further comprises:
a third input, connected to receive another number of the first type, generated randomly;
means for generating at the first output, for transmission with the digitally signed composite quantity, a number of the second type obtained by a practically irreversible transformation of the number of the first type received through the third input; and
means for receiving from the second input another number of the second type generated in and transmitted from the other key generator;
and wherein the means for generating a session key performs a practically irreversible transformation involving both numbers of the first type, received at the first and third inputs, and both numbers of the second type received at the second input, whereby a different session key may be generated for each message transmission session.

3. A secure key generator as defined in claim 1, wherein:
the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters;
the number of the second type received in the digitally signed composite quantity from the other key generator is designated Yb; and
the means for generating the session key performs the transformation $K = Yb^{Xa} \bmod p$.

4. A secure key generator as defined in claim 2, wherein:
the number of the second type stored in digitally signed form in the storage means is obtained by the transformation $Ya = \alpha^{Xa} \bmod p$, where Xa is the number of the first type stored in the storage means, and α and p are publicly known transformation parameters;

the number of the second type received in the digitally signed composite quantity from the other key generator is designated Yb; and
the means for generating the session key performs the transformation
$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,
where $X'a$ is the number of the first type that is randomly generated, $Y'b$ is the additional number of the second type received from the other key generator, and the $\oplus$ symbol denotes an exclusive OR operation.

5. A method of generating a secure session key between two user devices connected by an insecure communications channel, comprising the following steps performed at both devices:
transmitting a digitally signed composite quantity to the other device, the composite quantity including a publicly known device identifier IDa and a number Ya derived by a practically irreversible transformation of a secret number Xa that it is unique to the device;
receiving a similarly structured digitally signed composite quantity from the other device;
transforming the received digitally signed composite quantity into an unsigned composite quantity containing a device identifier IDb of the other device and a number Yb that was derived by transformation from a secret number Xb that is unique to the other device;
verifying the identity of the other device from the device identifier IDb; and
generating a session key by performing a practically irreversible transformation involving the numbers Xa and Yb.

6. A method as defined in claim 5, and further including the steps of:
generating another number $X'a$ randomly prior to generation of a session key;
transforming the number $X'a$ to a number $Y'a$ using a practically irreversible transformation;
transmitting the number $Y'a$ to the other device; and
receiving a number $Y'b$ from the other device;
wherein the step of generating a session key includes a practically irreversible transformation involving the numbers Xa, $X'a$, Yb and $Y'b$.

7. A method as defined in claim 6, wherein:
the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations; and
the step of generating a session key includes the transformation
$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p$,
where $\oplus$ denotes an exclusive OR operation.

8. A method of authentication in a public key cryptographic system, the method comprising the steps of:
selecting a unique random number Xi for each cryptographic device to be distributed;
transforming the number Xi to a new number Yi using a practically irreversible transformation;
forming a composite quantity by combining the number Yi with a publicly known device identifier IDi;
digitally signing the composite quantity containing Yi and IDi;
storing the signed composite quantity and the number Xi permanently in each device;
exchanging, between two devices, a and b, desiring to establish secured communication, the signed composite quantities stored in each;
authenticating, in each of the two devices, the identity of the other device; and
generating, in each of the two devices, a session key to be used for secured communication.

9. A method as defined in claim 8, wherein the step of authenticating includes:
transforming the digitally signed composite quantity received from the other device into unsigned form; and
comparing the value of IDb in the unsigned quantity with the known IDb of the other device.

10. A method as defined in claim 9, wherein:
the step of generating the session key includes performing a transformation that involves a value Yb received from the other device and the value Xa of this device.

11. A method as defined in claim 10, wherein:
the step of digitally signing includes computing a modular square root of the composite quantity; and
the step of transforming the digitally signed composite quantity to unsigned form includes computing a modular square of the signed quantity.

12. A method as defined in claim 11, wherein:
the steps of computing a modular square root and computing a modular square both employ a modulus that is the product of two prime numbers.

13. A method as defined in claim 8, and further comprising the steps of:
transforming, in each of the two devices, the digitally signed composite quantity received from the other device into unsigned form; and
generating, in each of the two devices, a, b, a random number $X'a$, $X'b$;
transforming the numbers $X'a$, $X'b$ into numbers $Y'a$, $Y'b$ by a transformation that is practically irreversible; and
exchanging the numbers $Y'a$, $Y'b$ between the two devices;
and wherein the step of generating the session key includes performing a practically irreversible transformation involving the numbers Xa, $X'a$, Yb, and $Y'b$ in device a, and the numbers Xb, $X'b$, Ya, and $Y'a$ in device b.

14. A method as defined in claim 13, wherein:

the transformations from X numbers to Y numbers is of the type $Y = \alpha^X \bmod p$, where α and p are chosen to maximize irreversibility of the transformations; and

the step of generating a session key includes the transformations

$K = (Y'b)^{Xa} \bmod p \oplus (Yb)^{X'a} \bmod p,$

for device a, and

$K = (Y'a)^{Xb} \bmod p \oplus (Ya)^{X'b} \bmod p,$

for device b, where $\oplus$ denotes an exclusive OR operation.

15. A method as defined in claim 13, wherein:

the step of digitally signing includes computing a modular square root of the composite quantity; and

the step of transforming the digitally signed composite quantity to unsigned form includes computing a modular square of the signed quantity.

16. A method as defined in claim 15, wherein:

the steps of computing a modular square root and computing a modular square both employ a modulus that is the product of two prime numbers.

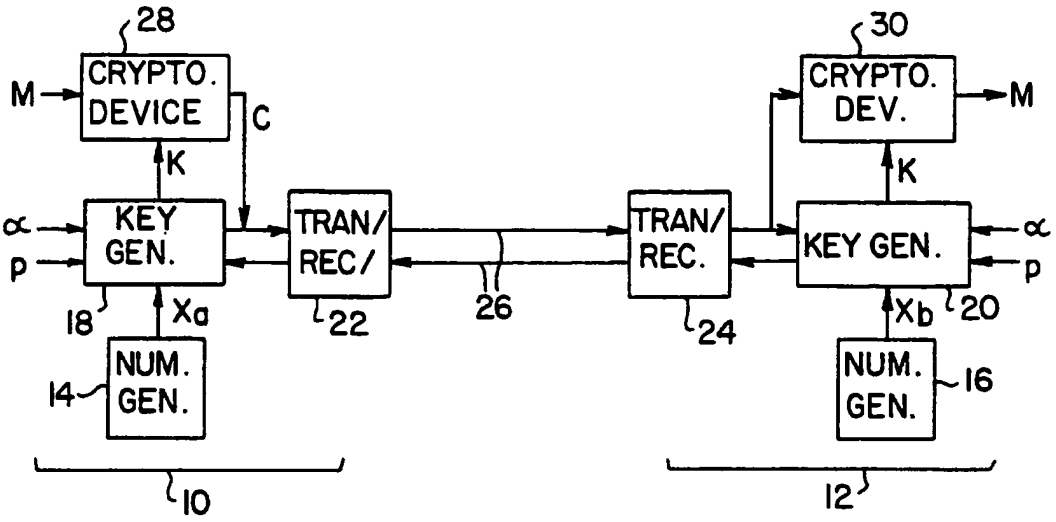


FIG. 1 (PRIOR ART)

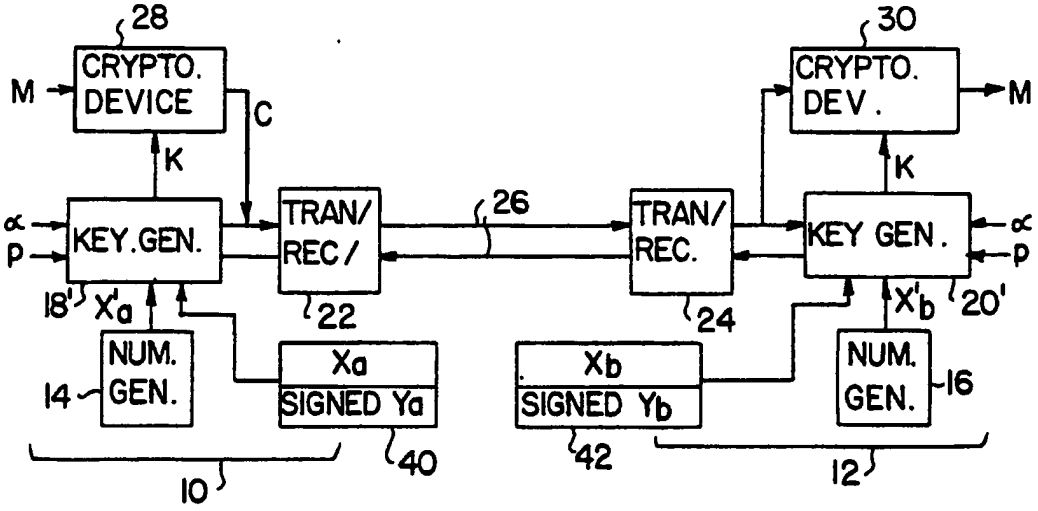


FIG. 3

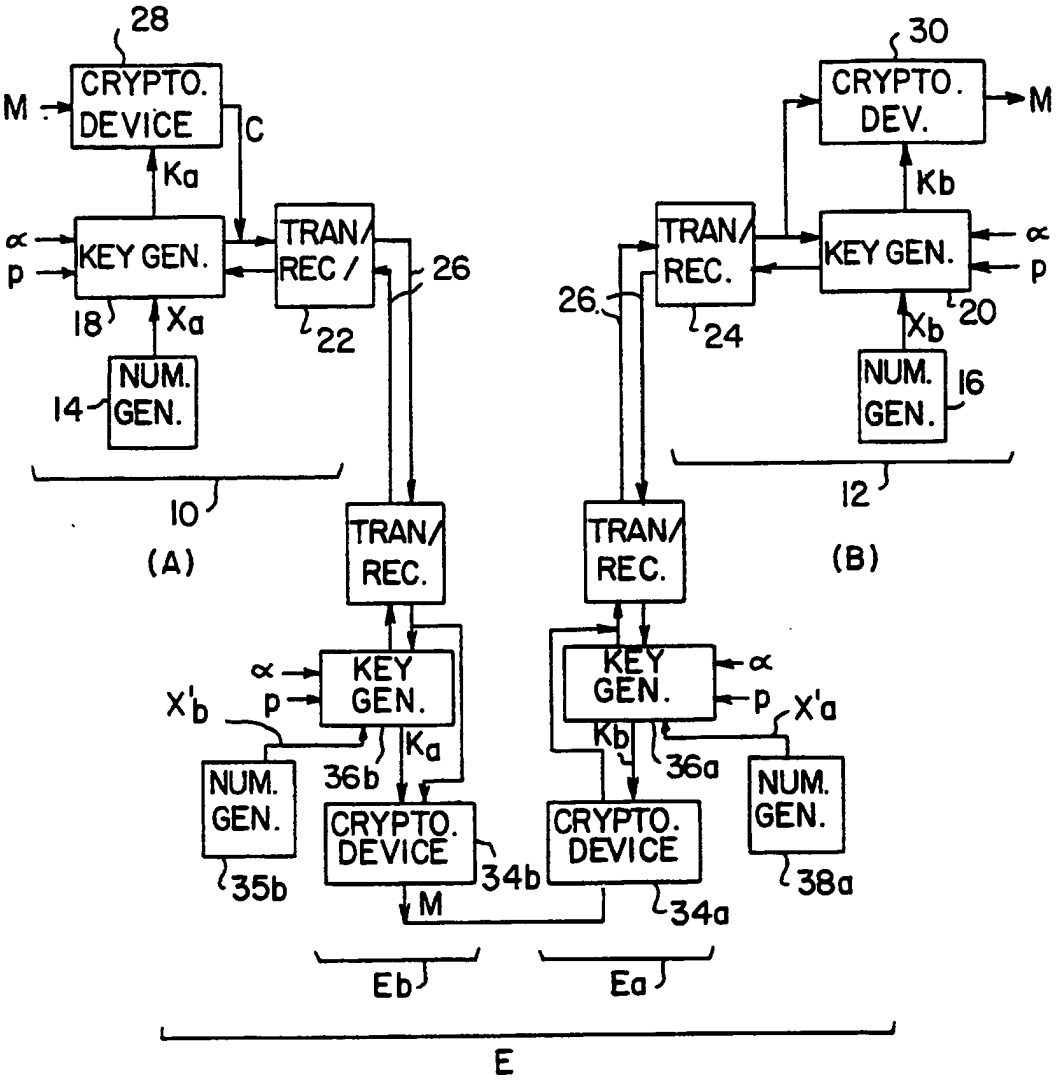
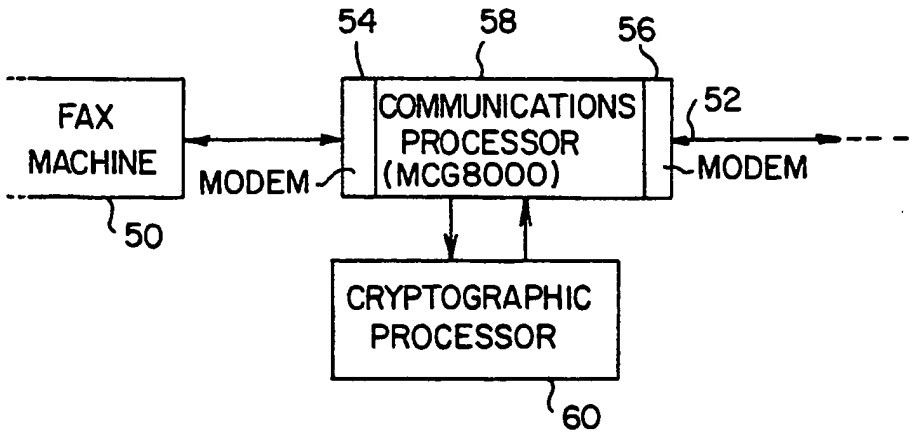


FIG. 2 (PRIOR ART)

FIG. 4


54
58
56
FAX MACHINE
MODEM
COMMUNICATIONS PROCESSOR (MCG8000)
52
MODEM
50
CRYPTOGRAPHIC PROCESSOR
60

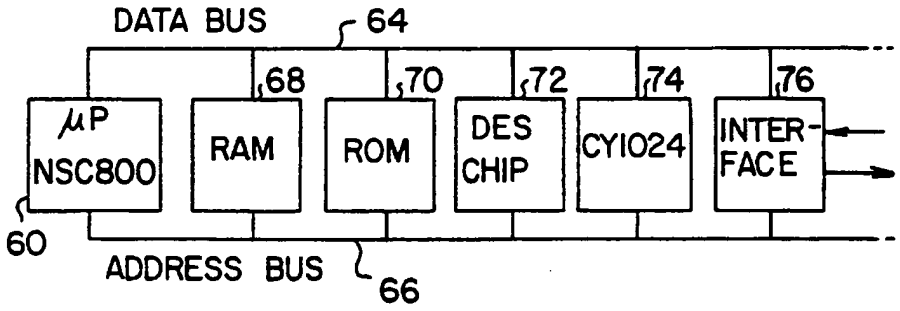

DATA BUS
64
68
70
72
74
76
μP NSC800
RAM
ROM
DES CHIP
CYI024
INTER-FACE
60
ADDRESS BUS
66

FIG. 5

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 450 841 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 91302657.1

(51) Int. Cl.[5]: H04N 7/16

(22) Date of filing: 25.03.91

(30) Priority: 29.03.90 US 501620
29.03.90 US 501682
29.03.90 US 501683
29.03.90 US 561684
29.03.90 US 501685
29.03.90 US 501658

(43) Date of publication of application:
09.10.91 Bulletin 91/41

(84) Designated Contracting States:
BE DE FR GB IT

(71) Applicant: GTE LABORATORIES INCORPORATED
1209 Orange Street
Wilmington Delaware 01901 (US)

(72) Inventor: Walker, Stephen S.
117 Kelleher Road
Marlborough, MA 01752 (US)
Inventor: Sidlo, Clarence M.
5 Lowry Road
Framlingham, MA 01701 (US)
Inventor: Teare, Melvin J.
21 Woodleigh Road
Framlingham, MA 01701 (US)

(74) Representative: Bubb, Antony John Allen et al
GEE & CO. Chancery House Chancery Lane
London WC2A 1QU (GB)

(54) Video control system.

(57) A video control system includes a central facility (11) and a terminal (10). Video program means provided the terminal with a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code. The terminal (10) includes means (22) for sending the program identification data to the central facility (11). The central facility includes a data base (19) for storing and retrieving at least one code encryption key corresponding to the program identification data and means (20) for sending the code encryption key from the central facility (11) to the terminal (10). The terminal (10) further includes means (22) for receiving the code encryption key from the central facility, decrypting means (23) for decrypting the encrypted digital code of the first frame in accordance with the code encryption key and means (24) for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted random digital code. The video program means may transmit the program to said terminal (10) or be located at the terminal (10) for playing a video recording medium storing the program.

EP 0 450 841 A2

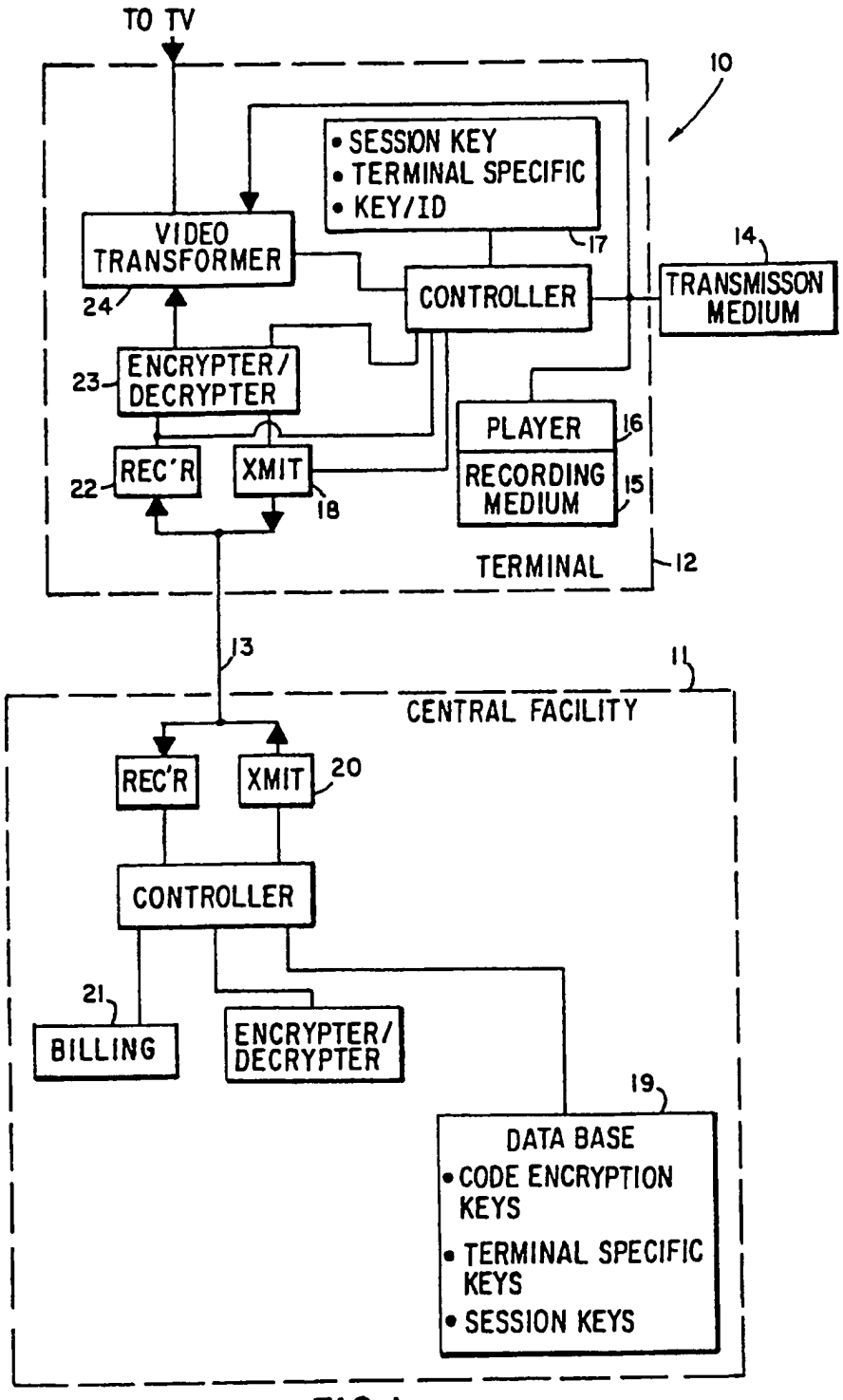

TO TV
10
• SESSION KEY
• TERMINAL SPECIFIC
• KEY/ID
17
VIDEO TRANSFORMER
24
CONTROLLER
14
TRANSMISSON MEDIUM
23
ENCRYPTER/ DECRYPTER
PLAYER
16
RECORDING MEDIUM
15
22
REC'R
XMIT
18
TERMINAL
12
13
11
CENTRAL FACILITY
REC'R
XMIT
20
CONTROLLER
21
BILLING
ENCRYPTER/ DECRYPTER
19
DATA BASE
• CODE ENCRYPTION KEYS
• TERMINAL SPECIFIC KEYS
• SESSION KEYS

FIG. 1

This invention is concerned with video control systems. It is desirable to provide a video control system which decrypts encrypted broadcasts or recorded copies of video material such that the subsequent viewing is controlled. This allows the owner to either forbid viewing, or collect revenue at his or her discretion.

In the prior art, a software distribution system is known wherein a computer program is downloaded once, followed by an access key to allow use of it on each subsequent use. This system uses a dynamic key that constantly changes, and is directly related to a user's decoder box, both by ID and an internal dynamic counter.

Also known is a video system that autonomously controls the viewing of a recording for either 24 hours or once only. It does not have the power of control desired.

Accordingly the present invention provides a video system comprising: a central facility; a terminal; and video program means for providing to said terminal a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code; said terminal including means for sending said program identification data to said central facility; said central facility including a data base for storing and retrieving at least one code encryption key corresponding to the program identification data and means for sending said code encryption key from said central facility to said terminal; said terminal further including means for receiving the code encryption key from said central facility, decrypting means for decrypting the encrypted digital code of said first frame in accordance with said code encryption key and means for transforming said unintelligible video signal of said second frame to said intelligible video signal using the decrypted random digital code.

One embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings in which:

Figure 1 is a block diagram of a video system embodying the invention; and

Figure 2 shows an encryption arrangement according to the invention.

Reference is made to Figure 1 which is a block diagram of a video system 10 embodying the invention. The video system comprises a central facility 11, a terminal 12, and a duplex communication link 13 between central facility 11 and terminal 12. An overview of the system is first given.

Terminal 12 is provided with a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code.

The video program may be transmitted by broadcast, cable, satellite, fiber, or any other transmission medium 14. Alternative the video program may be stored on a video recording medium 15 such as magnetic tape or video disk and played by player 16. The unintelligible video signal may be either analog or digital.

A second field has a vertical blanking interval containing both a random digital code encrypted according to a code encryption key and program identification data, is followed by a third field containing an unintelligible video signal previously transformed from an intelligible video signal according to the random digital code of the second field.

Terminal 12 includes means 17 to store terminal identification data and means to send to the central facility 11 the terminal identification data and the program identification data over link 13.

Central facility 11 includes a data base 19 for storing and retrieving at least one code encryption key corresponding to the program identification data, means 20 for sending the code encryption key from the central facility 11 to the terminal 12, and means 21 for generating billing data based on both terminal identification data and program identification data.

Terminal 12 further including means 22 for receiving the code encryption key from central facility 11, decrypting means 23 for decrypting the encrypted random digital code of the first frame in accordance with the code encryption key, and means 24 for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted random digital code.

Each terminal 12 may have a terminal specific encryption key and means 18 to send to the central facility the program identification data and the terminal 11 identification data encrypted according to the terminal specific encryption key. The central facility 11 has means for storing a duplicate of the terminal specific encryption key, means for encrypting the code encryption key according to the terminal specific encryption key; and means for sending the encrypted code encryption key from central facility 11 to terminal 12.

Terminal 12 further includes means 22 for receiving the encrypted code encryption key from central facility 11, decryption means 23 for decrypting the code encryption key according to the terminal specific encryption key, and decrypting the encrypted random digital code of the first frame in accordance with the code encryption key, and means 24 for transforming the unintelligible video signal of the second frame to the intelligible video signal using the decrypted ran-

dom digital code.

Terminal 12 includes means to encrypt the terminal identification data according to the terminal specific encryption key, means to send unencrypted terminal identification data and encrypted terminal identification data to the central facility, which in turn includes means to compare unencrypted and encrypted terminal identification data to verify terminal identity.

A plurality of code encryption keys may be used for one program wherein a desired code encryption key is selected from the plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code.

Various features of the system are now discussed in more detail.

System 10 controls the viewing of video programs, by which is meant any video material, either transmitted or recorded, in television format consisting of a series of fields of lines. Two interlaced fields make up a television frame.

Video programs are rendered unintelligible, e.g. scrambled, by any analog or digital method, and are made intelligible, e.g. descrambled, using random digital codes located in fields. The random digital keys are themselves encrypted, and decrypted by a one or more key obtained from a database located at the central facility, along with user-specific information at the time of viewing. The system does not stop copying, it controls viewing, while protecting revenues. As such, it can encourage copying, which could ease the distribution issue by controlling the playback such that revenue can be collected each time.

Preferably duplex communication link 13 is a continuous data channel between a terminal and a central facility such as an ISDN D-channel or by modem over a regular phone line.

The video program is encrypted, and needs a decrypter in the terminal for viewing. The decrypter uses data embedded in the video program along with a data access to correctly perform the decryption, so the process is completely controlled. The embedded data and key transfer from the remote database may be protected with public domain encryption techniques, providing high level security before first viewing.

The video program may be recorded as is, but it is still unviewable. To view it, the decrypter is used, along with the encrypted embedded data, and an access to a secure database, to perform the decryption. Recordings may be freely copied, but remain unviewable unless used with the decrypter.

To view the programs requires access to the database using encrypted data transfer. This process yields the control of the video program, whether recording or trans mission. The decrypter requires one or more keys that arrives from the database. To get the key, information from the video program as well as terminal identification is sent to the database.

A direct Electronic funds Transfer (EFT) debit can be performed using the information. It the program is a video store copy, the EFT could include the store fee and the copyright fee. Note that the video distribution to video stores becomes trivial, as they are encouraged to take a direct recording with a video store key, along with their authorized converter box, and make as many copies as they like. The revenue control takes place at viewing time. This encourages a shareware type of distribution.

A passkey can be sent to the database, to allow viewing of questionable taste films by adults, controlling access by minors.

On the first access, the database will capture a signature derived from the user's equipment and the recording, and store it for subsequent tracking. As there is a compelled database access in this process, data on usage may be collected. This same process may be used for revenue collection.

The system preferably uses at least one downloadable key, an encrypted video program that uses the key for decryption, and data stored in a field of the video program. It may be implemented in an all digital, analog, or mixed analog/digital environment.

The video programs are encrypted, with data relating to the programs, e.g. where and when, who transmitted it. The data may also contain part of the decryption key. This information would be extracted from the signal, and used to access a database, maintained by the program's owners, to obtain an encrypted key for the decrypter. After a subscriber and/or a credit check is successfully completed, the one or more keys would be transmitted. At this time the owner has obtained usage data, with a specific user's ID, and has the option of billing him. If it is a free program, at least the viewer data is available.

If a user records a transmission or another recording, he captures the encrypted signal, along with embedded data, as described above. This accomplishes the signature part of the process. A recording created by this method may be on a regular VCR, but is encrypted and individually marked. Copying a recording does not affect the system, as the rerecording is only usable with the correct keys. Potentially, the first few minutes of a program might be viewable without the need of a key, to allow the user to see what the contents of the program are, as well as to allow time for the database access and key synchronization process.

To play a recording back, it is necessary to re-obtain the one or more keys. The combination of data stored in a field is used to access the database. Before the keys are made available, there is a check that the terminal identification and the embedded data match.

In the case wherein a recording is rented from a video store, a code may identify the store. The database recognizes the recording as a rental copy, and

charge either the user or the video store a fee. If the recording is viewed a second time, the charge is repeated. In the event a copy is made, when it is played, the database will identify the originating video store, but not the actual copier. However, if validation is performed at rental time, there would be some measure of control. If the entire charging process were to be reversed, such that the viewer carries all the liability for charges, then copying is encouraged, as per shareware, and the distribution problem is minimized, while revenues are maintained on a usage basis.

The program's owner has the responsibility to get a secured copy to whoever deals with the distribution of the programs. The programs are encrypted, and require a database update to enable viewers to make use of the program. The viewer has a terminal including a decrypter, linked to the central facility's database via an automatic dial-up, that, when enabled, decrypts the video program. As appropriate, there can be credit checks and billing from the database, as well as statistics collection.

The encryption has two levels, one for protection of video decryption codes on the program, and one for protection of messages between the terminal and the central facility. Both may use the NBS Data Encryption Standard (DES).

DES encryption and decryption may be implemented with a commercial Motorola 6859 Data Security Device or similar product at the terminal and at the central facility.

The decryption code itself is protected by being DES-encrypted. The decryption key is not on the video program but is retained in the database at the central facility. A program identification number and a decryption key number allow the central facility to recover the decryption key itself and send it to the terminal for decrypting the decryption codes.

A different DES decryption key is not required for every field. One key can span several fields. DES key requests and acknowledgements from the terminal may also act as keep-alive messages to the central facility.

DES decryption keys are transmitted from the central facility to the terminal protected by a higher-level DES "session" key. terminal requests for new keys as the tape progresses are also protected by the DES session key. This key is generated by the central facility at the beginning of the session and remains valid for the duration of the session. The terminal begins the session using a terminal-unique DES key stored in a ROM.

Frame contents are transferred from the Analog Subsystem to the DCSS and the decrypted decryption code from the DCSS to the Analog Subsystem over the analog interface shown in the Figure. Transfer of data between the subsystems may be coordinated by means of the vertical and horizontal blanking signals and their derivative interrupts.

All messages between terminal and central facility use Cyclic Redundancy Code (CRC) checking to verify message integrity. The CRC-CCITT generating polynomial generates two block check characters (BCC) for each message. If the terminal receives a message that is not verified by the BCC, it sends a request (ARQ) to the central facility to retransmit the last message. The central facility does not attempt to ARQ garbled messages. It discards them and waits for a terminal to send again.

Message exchange in the VCS is by a positive acknowledgent scheme in which a response of some kind is expected for every message sent. For example, a terminal expects a DES decryption key message after it sends a request for the same; the central facility expects a key receipt acknowledge after it sends the key message.

When a user begins to play a protected program, the terminal initiates a session by sending a "session start" message (STS) to the central facility containing user and program identifications. The message contains message type, user number and CRC code in the clear, but the balance of the message is DES-encrypted with the initial DES session key stored in the terminal ROM. (The user identification is also stored in ROM.) The central facility uses the unencrypted data to access its database and find the user DES value for decrypting the remainder of the message.

The central facility authenticates the message by comparing clear and decrypted user numbers. If the user numbers are identical, the central facility then confirms that the program serial number is valid. The central facility may also check user credit. If all is well, the central facility accepts the session and generates a new (and random) DES key that is unique for that session. It encrypts this using the initial user value in the database and sends it to the terminal, which decrypts the message and stores the new value in its database (MCU RAM) as the session key for the remainder of the session.

The central facility then uses the tape and decryption key number in the STS message to recover a set of DES decryption keys for the program from the database. These are encrypted with the session key and sent to the terminal at the start of a session or during the course of a session.

The terminal generates session start, key acknowledgement, and ARQ messages. The central facility responds in kind. Both the central facility and the terminal generate and verify block check characters.

The preferred embodiment and best mode of practicing the invention have been described. Alternatives now will be apparent to those skilled in the art in light of these teachings. Accordingly the invention is to be defined by the following claims and not by the particular examples given.

Claims

1. A video system comprising:
a central facility;
a terminal; and
video program means for providing to said terminal a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video-signal according to said random digital code;
said terminal including means for sending said program identification data to said central facility;
said central facility including a data base for storing and retrieving at least one code encryption key corresponding to the program identification data and means for sending said code encryption key from said central facility to said terminal;
said terminal further including means for receiving the code encryption key from said central facility, decrypting means for decrypting the encrypted digital code of said first frame in accordance with said code encryption key and means for transforming said unintelligible video signal of said second frame to said intelligible video signal using the decrypted random digital code.

2. The system of claim 1 wherein a plurality of code encryption keys are used for one program, and wherein a desired code encryption key is selected from said plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code encrypted with said desired code encryption key.

3. The system of claim 1 or 2 wherein said video program means is means for transmitting said program to said terminal.

4. The system of claim 3 wherein said means for transmitting is a CATV system.

5. The system of any one of claims 1-4 wherein:
said terminal further includes means to store terminal identification data and a terminal specific encryption key; and means to send to said central facility said terminal identification data with said program identification data;
said central facility further includes means for storing a duplicate of said terminal specific encryption key; means for encrypting said code encryption key according to said terminal specific encryption key; and means for sending the encrypted code encryption key from said central facility to said terminal; and
said terminal further further includes means for receiving the encrypted code encryption key from said central facility; and decryption means for decrypting said code encryption key according to said terminal specific encryption key.

6. The video system of any one of claims 1-4 wherein:
said terminal further includes means to store terminal identification data and a terminal specific encryption key; and means to send to said central facility said program identification data and said terminal identification data,
said central facility further includes means for providing a session encryption key; means for encrypting said session encryption key according to said terminal specific encryption key; means for sending the encrypted session encryption key from said central facility to said terminal;
means for encrypting said code encryption key according to said encrypted session encryption key; and means for sending the encrypted code encryption key from said central facility to said terminal; and
said terminal further includes means for receiving the encrypted session encryption key from said central facility; decryption means for decrypting said session encryption key according to said terminal specific encryption key, means for receiving the encrypted code encryption key from said central facility; and decryption means for decrypting said code encryption key according to said session encryption encryption key.

7. The system of claim 5 or 6 wherein said terminal includes means to encrypt said terminal identification data according to said terminal specific encryption key, and means to send unencrypted terminal identification data and encrypted terminal identification data to said central facility, and said central facility includes means to compare unencrypted and encrypted terminal identification data to authenticate terminal identity.

8. The system of any one of claims 5-7 wherein said central facility further includes means for generating billing data based on said terminal identification data and said program identification data.

9. The video system of any one of claims 1-8 wherein said video program means is a means located at said terminal for playing a video recording medium storing said program.

10. A video recording medium storing a video program including a series of television fields including a first field containing both a random digital code encrypted according to a code encryption key and program identification data, and a second field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code.

11. The medium of claim 10 wherein a plurality of code encryption keys are used for one program, and wherein a desired code encryption key is selected from said plurality of code encryption keys in accordance with code encryption key identification data corresponding to the random digital code encrypted with said desired code encryption key.

12. The medium of claim 10 or 11 wherein said second field has a vertical blanking interval containing both a random digital code encrypted according to a code encryption key and program identification data, and is followed by a third field containing an unintelligible video signal previously transformed from an intelligible video signal according to said random digital code of the second field.

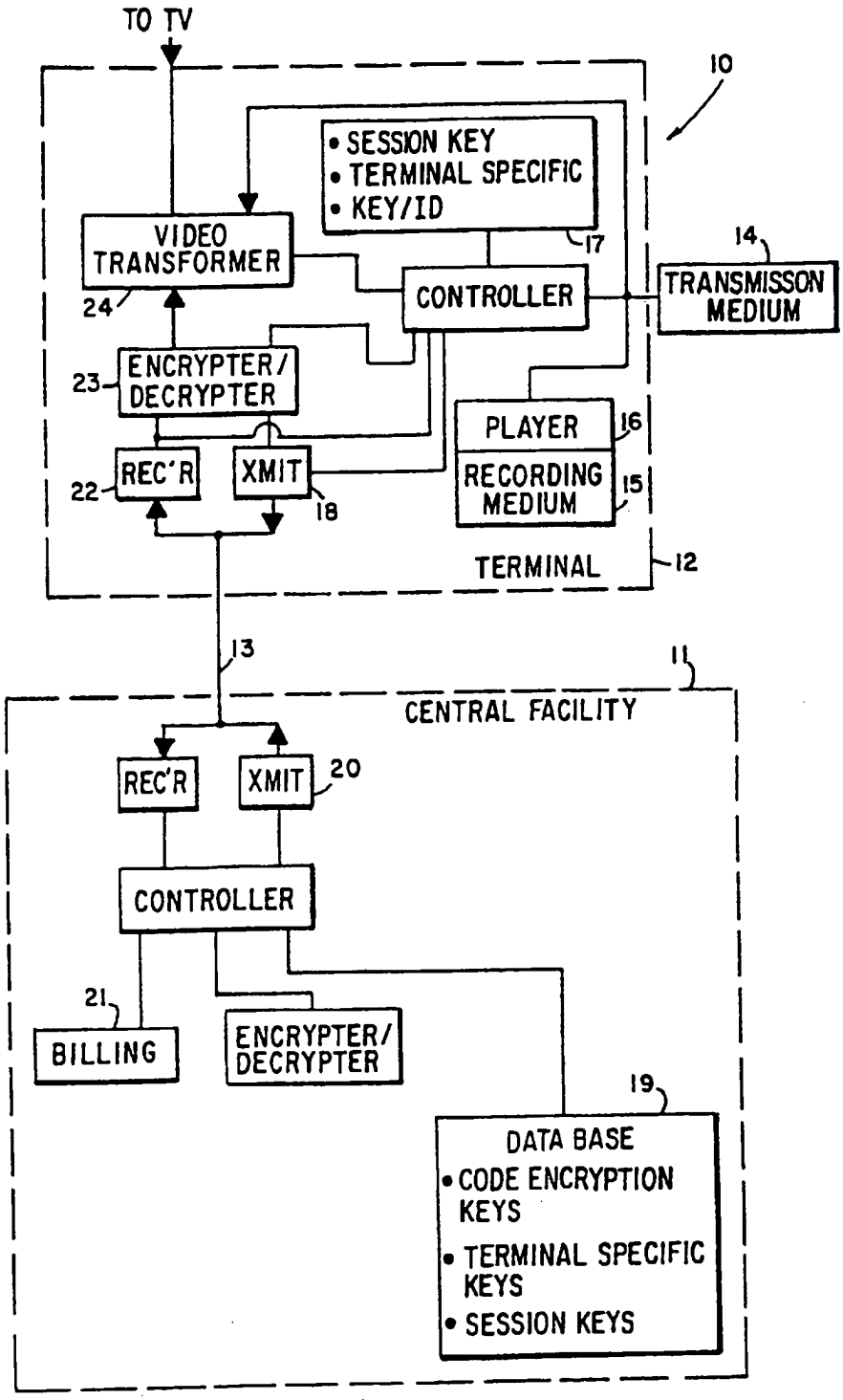


FIG.1

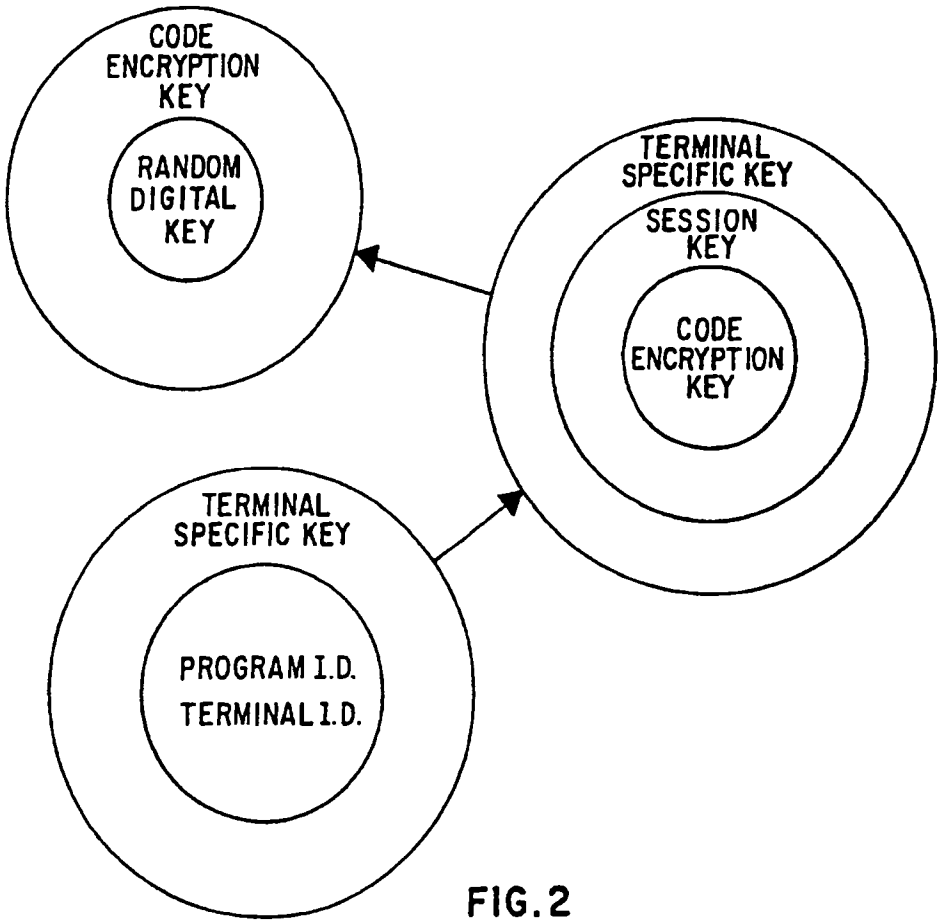


FIG. 2

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 529 261 A2**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: **92111758.6**

(51) Int. Cl.5: **H04L 9/08**

(22) Date of filing: **10.07.92**

(30) Priority: **22.08.91 US 748407**

(43) Date of publication of application:
03.03.93 Bulletin 93/09

(84) Designated Contracting States:
CH DE FR GB IT LI NL SE

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Matyas, Stephen M.**
10298 Cedar Ridge Drive
Manassas, VA 22110(US)
Inventor: **Johnson, Donald B.**
11635 Crystal Creek Lane
Manassa, VA 22111(US)
Inventor: **Le, An V.**
10227 Battlefield Drive
Manassas, Va 22110(US)
Inventor: **Martin, William C.**
1835 Hilliard Lane
Concord, NC 28025(US)
Inventor: **Prymak, Rostislaw**
15900 Fairway Drive
Dumfries, VA 22026(US)
Inventor: **Rohland, William S.**
4234 Rotunda Road
Charlotte, NC 28226(US)
Inventor: **Wilkins, John D.**
P.O. Box 8
Somerville, VA 22739(US)

(74) Representative: **Herzog, Friedrich Joachim, Dipl.-Ing.**
IBM Deutschland GmbH, Patentwesen und Urheberrecht, Schönaicher Strasse 220
W-7030 Böblingen (DE)

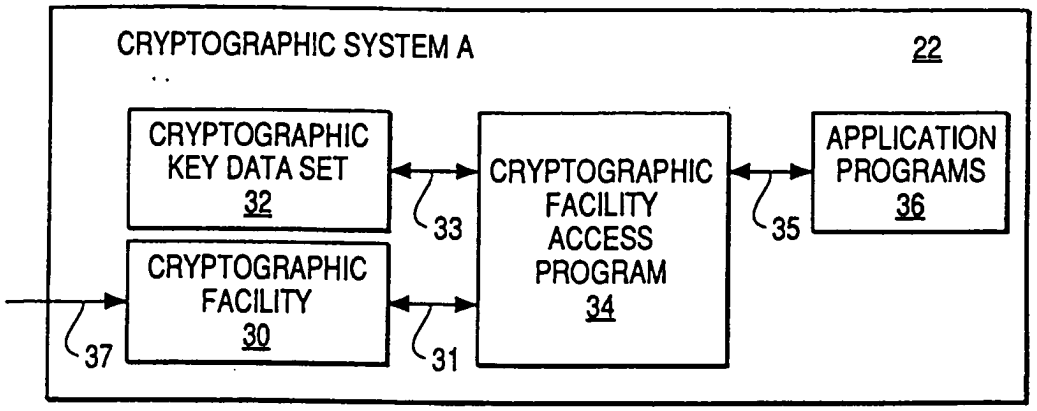
(54) **A hybrid public key algorithm/data encryption algorithm key distribution method based on control vectors.**

(57) The patent describes a method and apparatus for securely distributing an initial Data Encryption Algorithm (DEA) key-encrypting key by encrypting a key record (consisting of the key-encrypting key and control information associated with that key-encrypting key) using a public key algorithm and a public key belonging to the intended recipient of the key record. The patent further describes a method and apparatus for securely recovering the distributed key-encrypting key by the recipient by decrypting the received key record using the same public key algorithm and private key associated with the public key and re-encrypting the key-encrypting key under a key formed by arithmetically combining the recipient's master key with a control vector contained in the control information of the received key record. Thus the type and usage attributes assigned by the originator of the key-encrypting key in the form of a control vector are cryptographically coupled to the key-encrypting key such that the recipient may only use the received key-encrypting key in a manner defined by the key originator.

The patent further describes a method and apparatus to improve the integrity of the key distribution process by applying a digital signature to the key record and by including identifying information (i.e., an originator identifier) in the control information of the key record. The integrity of the distribution process is enhanced by verifying the digital signature and originator identifier at the recipient node.

EP 0 529 261 A2

FIG. 2



The invention disclosed broadly relates to data processing systems and methods and more particularly relates to cryptographic systems and methods for use in data processing systems to enhance security.

The following patents are related to this invention and are incorporated herein by reference:

B. Brachtl, et al., "Controlled Use of Cryptographic Keys Via Generating Stations Established Control Values," USP 4,850,017, issued July 18, 1989, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors," USP 4,941,176, issued July 10, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Data Cryptography Operaions Using Control Vectors," USP 4,918,728, issued April 17, 1990, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Personal Identification Number Processing Using Control Vectors," USP 4,924,514, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Management of Keys Using Extended Control Vectors," USP 4,924,515, issued May 8, 1990, assigned to IBM Corporation and incorporated herein by reference.

S. M. Matyas, et al., "Secure Key Management Using Programmable Control Vector Checking," USP 5,007,089, issued April 9, 1991, assigned to IBM Corporation and incorporated herein by reference.

B. Brachtl, et al., "Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function," USP 4,908,861, issued March 13, 1990, assigned to IBM Corporation and incorporated herein by reference.

D. Abraham, et al., "Smart Card Having External Programming Capability and Method of Making Same," serial number 004,501, filed January 19, 1987, assigned to IBM Corporation, and incorporated herein by reference.

S. M. Matyas, et al., "Method and Apparatus for Controlling the Use of a Public Key, Based on the Level of Import Integrity for the Key," serial number 07/602,989, filed October 24, 1990, assigned to the IBM Corporation.

S. M. Matyas, et al., "Secure Key Management Using Programmable Control Vector Checking," USP 5,007,089, issued April 9, 1991, assigned to IBM Corporation and incorporated herein by reference.

The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on associating with a cryptographic key, a control vector which provides the authorization for the uses of the key intended by the originator of the key. The cryptographic architecture described in the cited patents by S. M. Matyas, et al. is based on the Data Encryption Algorithm (DEA), whereas the present invention is based on both a secret key algorithm, such as the DEA, and a public key algorithm. Various key management functions, data cryptography functions, and other data processing functions are possible using control vectors, in accordance with the invention. A system administrator can exercise flexibility in the implementation of his security policy by selecting appropriate control vectors in accordance with the invention. A cryptographic facility (CF) in the cryptographic architecture is described in the above cited patents by S. M. Matyas, et al. The CF is an instruction processor for a set of cryptographic instructions, implementing encryption methods and key generation methods. A memory in the crypto facility stores a set of internal cryptographic variables. Each cryptographic instruction is described in terms of a sequence of processing steps required to transform a set of input-parameters to a set of output parameters. A cryptographic facility application program is also described in the referenced patents and patent applications, which defines an invocation method, as a calling sequence, for each cryptographic instruction consisting of an instruction mnemonic and an address with corresponding input and output parameters.

Public key encryption algorithms are described in a paper by W. Diffie and M. E. Hellman entitled "Privacy and Authentication: An Introduction to Cryptography," Proceedings of the IEEE, Vol. 67, No. 3, March 1979, pp. 397-427. Public key systems are based on dispensing with the secret key distribution channel, as long as the channel has a sufficient level of integrity. In a public key crypto system, two keys are used, one for enciphering and one for deciphering. Public key algorithm systems are designed so that it is easy to generate a random pair of inverse keys PU for enciphering and PR for deciphering and it is easy to operate with PU and PR, but is computationally infeasible to compute PR from PU. Each user generates a pair of inverse transforms, PU and PR. He keeps the deciphering transformation PR secret, and makes the enciphering transformation PU public by placing it in a public directory. Anyone can now encrypt messages and send them to the user, but no one else can decipher messages intended for him. It is possible, and often desirable, to encipher with PU and decipher with PR. For this reason, PU is usually referred to as a public key and PR is usually referred to as a private key. A corollary feature of public key crypto systems is the provision of a digital signature which uniquely identifies the sender of a message. If user A wishes to send a signed message M to user B, he operates on it

with his private key PR to produce the signed message S. PR was used as A's deciphering key when privacy was desired, but it is now used as his "enciphering" key. When user B receives the message S, he can recover the message M by operating on the ciphertext S with A's public PU. By successfully decrypting A's message, the receiver B has conclusive proof it came from the sender A. Examples of public key cryptography are provided in the following U. S. patents:

USP 4,218,582 to Hellman, et al., "Public Key Cryptographic Apparatus and Method;" USP 4,200,770 to Hellman, et al., "Cryptographic Apparatus and Method;" and USP 4,405,829 to Rivest, et al., "Cryptographic Communications System and Method," which discloses the RSA public-key algorithm.

In general, it is preferable for performance reasons to use symmetric algorithms such as the Data Encryption Algorithm (DEA) bulk data encryption rather to use a public key algorithm for such purposes. However to use DEA both the data originator and intended recipient must first share a common, secret key. This requires the secure distribution of at least one DEA key for each secure "channel" between originator and recipient. The problem can be reduced to distributing one secret DEA key-encrypting key (KEK) between the originating node and receiving node, and thereafter transmitting all other DEA keys encrypted under this common KEK. The usual method of distributing the initial KEK is via trusted couriers.

It is well-known that a hybrid system employing a public key algorithm and the DEA may be effective in solving the initial KEK distribution problem, while still retaining the faster bulk data encryption capabilities of the DEA. In such a hybrid cryptographic system A, a public key PU is transmitted with integrity (see S. M. Matyas, et al., "Method and Apparatus for Controlling the Use of a Public Key, Based on the Level of Import Integrity for the Key", serial number 07/602,989, filed October 24, 1990) to a second hybrid cryptographic system B. A secret DEA KEK, say KK, is generated and encrypted under PU at system B and transmitted to system A. System A uses the corresponding private key PR to decrypt KK. KK may then be used with the DEA algorithm to distribute additional DEA keys for use by systems A and B.

Prior art, however, has not provided a cryptographically secure means to define the type and to control the usage of the generated KEK to insure that the type and uses defined by the originator of the key (system B) are enforced at both the originating node and the recipient node (system A). Without such controls (as described in S. M. Matyas, et al., "Secure Management of Keys Using Control Vectors", USP 4,941,176, issued July 10, 1990), the distributed KEK may be subject to misuse by either party to weaken the security of the system (e.g., by allowing the KEK to be used in a data decrypt operation and thus allowing DEA keys encrypted under the KEK to be decrypted and exposed in the clear).

While the prior art addresses the concept of unidirectional key-encrypting keys, i.e., key-encrypting keys that establish a key distribution channel in one direction only, the method for establishing, with integrity, such a unidirectional channel using a public key algorithm has not been addressed. To accomplish this, a unique Environment Identifier (EID) is stored at each cryptographic device such that a distributed key-encrypting key can be imported only at the designated receiving device, but it does not allow the key-encrypting key to be imported or re-imported at the sending device, as described below.

The originating node B generates the KEK in two forms: one form to be exported to the recipient node A (encrypted under the PU received from A) and a second form to be used at B ultimately to either export or import additional DEA keys (encrypted under some form of the local master key). As was described in the above reference U. S. patent 4,941,176, "Secure Management of Keys Using Control Vectors," it is critical to the security of each cryptographic system that the type and usage attributes of a given KEK on one system be limited to either EXPORTER usage or IMPORTER usage, but never both. Correspondingly, it must not be possible to generate or introduce two copies of the same KEK into the system, one with EXPORTER usage and one with IMPORTER usage. Such a pair of key forms is known as a bi-functional key pair.

Prior art has provided no cryptographically secure means to insure that the generated KEK cannot be re-imported into the originating node to form a bi-functional key pair. Since key PU is public, system A cannot be certain that system B is the originator of the generated KEK.

It is therefore a main object of the invention to provide an improved method for distributing DEA keys using a public key crypto system.

It is another object of the invention to provide an improved method of distributing a DEA key-encrypting key using a public key crypto system.

It is another object of the invention to provide an improved method of distributing a DEA key-encrypting key that does not require the use of couriers.

It is another object of the invention to provide control information associated with a distributed key, which defines the type and usage of the distributed key.

It is another object of the invention to provide a means to cryptographically couple the control information and key using a public-key algorithm.

It is another object of the invention to provide control information that prevents a distributed key from being imported at the originating device.

It is another object of the invention to provide a method of key distribution which is compatible with a key management based on control vectors (in the above referenced patents).

It is another object of the invention to provide a method of key distribution that does not also provide a covert privacy channel.

It is another object of the invention to provide a means for a receiving device to validate that a received distributed key has originated with an expected originating device.

It is another object of the invention to provide a means for a distributed key to be authenticated on the basis of a signature generated on the distributed key by the cryptographic system software.

It is still a further object of the invention to provide a higher integrity means for a distributed key to be authenticated on the basis of a signature generated on the distributed key as an integral part of the cryptographic system hardware export function.

These and other objects, features, and advantages are accomplished by the invention disclosed herein. A method and apparatus are disclosed for generating and distributing a DEA key-encrypting key from a sending device implementing a public-key cryptographic system to a receiving device implementing a public-key cryptographic system. The method and apparatus find application in a cryptographic system implementing both a symmetric encryption algorithm, such as the Data Encryption Standard, and an asymmetric encryption algorithm, such as the RSA public-key algorithm. The method begins by generating a key-encrypting key at a sending device and producing two encrypted copies of the generated key. The generated key is encrypted first under the public key of a designated receiving device and the encrypted key is then electronically transmitted to the receiving device. The generated key is also encrypted under the master key of the sending device and stored in a key storage for later use in a DEA key management scheme for distributing further DEA keys to the designated receiving device. At the receiving device, the encrypted key is decrypted using the private key of the receiving device and the clear key is then re-encrypted under the master key of the receiving device and the encrypted key is stored in a key storage for later use in a DEA key management scheme for receiving further DEA keys from the same sending device. In accordance with the invention, the method of key distribution makes use of a key block containing the distributed key-encrypting key and control information associated with the distributed key, which includes a control vector to limit uses of the key and an environment ID to identify the sender of the key. The method of key distribution also makes use of an optional digital signature generated on the encrypted key block at the originating device and validated at the receiving device.

These and other objects, features, and advantages of the invention will be more fully appreciated with reference to the accompanying figures.

Fig. 1 illustrates a communications network 10 including a plurality of data processors, each of which includes a cryptographic system;

Fig. 2 is a block diagram of a cryptographic system 22;

Fig. 3 is a block diagram of a cryptographic facility 30;

Fig. 4 is a block diagram showing the public and private keys that must first be initialized at two cryptographic systems A and B in order that they may electronically distribute DEA keys using a public key algorithm;

Fig. 5 is a block diagram illustrating DEA key distribution using the GKSP and IDK instructions without digital signatures;

Fig. 6 is a block diagram of a key block;

Fig. 7 is a block diagram of an external key token;

Fig. 8 is a block diagram illustrating DEA key distribution using the GKSP and IDK instructions with digital signatures;

Fig. 9 is a block diagram of the Generate Key Set PKA (GKSP) instruction;

Fig. 10 is a block diagram of the Import DEA Key (IDK) instruction;

Fig. 11 is a block diagram of control vectors for public and private keys used for key distribution (i.e., key management purposes);

Fig. 12 is a block diagram depicting an encrypted channel and a clear channel between two cryptographic systems A and B;

Fig. 13 is a block diagram illustrating the processing of control information at a receiving cryptographic device;

Fig. 14 is a block diagram of a cryptographic facility at a sending location, in accordance with the invention;

Fig. 15 is a block diagram of a cryptographic facility at a receiving location, in accordance with the invention;

Fig. 16 is a block diagram of the crypto-variable retrieval means 40 which is a component of the cryptographic facility shown in Fig. 14.

Environment Description: Fig. 1 illustrates a network block diagram showing a communications network 10 to which is connected a plurality of data

processors including data processor 20, data processor 20', and data processor 20". Also included in each data processor is a cryptographic system, as shown in Fig. 1. Data processor 20 includes cryptographic system 22, data processor 20' includes cryptographic system 22' and data processor 20" includes cryptographic system 22". Each data processor supports the processing of one or more applications which require access to cryptographic services such as for the encryption, decryption and authenticating of application data and the generation and installation of cryptographic keys. The cryptographic services are provided by a secure cryptographic facility in each cryptographic system. The network provides the means for the data processors to send and receive encrypted data and keys. Various protocols, that is, formats and procedural rules, govern the exchange of cryptographic quantities between communicating data processors in order to ensure the interoperability between them.

Fig. 2 illustrates the cryptographic system 22. In the cryptographic system 22, the cryptographic facility (CF) 30 has an input 37 from a physical interface. The cryptographic facility access program (CFAP) 34 is coupled to the cryptographic facility 30 by means of the interface 31. The cryptographic key data set (CKDS) 32 is connected to the cryptographic facility access program 34 by means of the interface 33. The application programs (APPL) 36 are connected to the cryptographic facility access program 34 by means of the interface 35.

A typical request for cryptographic service is initiated by APPL 36 via a function call to the CFAP 34 at the interface 35. The service request includes key and data parameters, as well as key identifiers which the CFAP 34 uses to access encrypted keys from the CKDS 32 at the interface 33. The CFAP 34 processes the service request by issuing one or more cryptographic access instructions to the CF 30 at the interface 31. The CF 30 may also have an optional physical interface 37 for direct entry of cryptographic variables into the CF 30. Each cryptographic access instruction invoked at the interface 31 has a set of input parameters processed by the CF 30 to produce a set of output parameters returned by the CF 30 to the CFAP 34. In turn, the CFAP 34 may return output parameters to the APPL 36. The CFAP 34 may also use the output parameters and input parameters to subsequently invoke instructions. If the output parameters contain encrypted keys, then the CFAP 34, in many cases, may store these encrypted keys in the CKDS 32.

Fig. 3 illustrates the cryptographic facility 30. The cryptographic facility 30 is maintained within a secure boundary 140. The cryptographic facility 30 includes the instruction processor 142 which is coupled to the cryptographic algorithms 144 which are embodied as executable code. The cryptographic facility environment memory 146 is coupled to the instruction processor 142. The physical interface can be coupled over line 37 to the CF environment memory 146, as shown in the figure. The instruction processor 142 is coupled to the cryptographic facility access program (CFAP) 34 by means of the interface at 31.

The instruction processor 142 is a functional element which executes cryptographic microinstructions invoked by the CFAP access instruction at the interface 31. For each access instruction, the interface 31 first defines an instruction mnemonic or operation code used to select particular microinstructions for execution. Secondly a set of input parameters is passed from the CFAP 34 to the CF 30. Thirdly, a set of output parameters is returned by the CF 30 to the CFAP 34. The instruction processor 142 executes the selected instruction by performing an instruction specific sequence of cryptographic processing steps embodied as microinstructions stored in cryptographic microinstruction memory 144. The control flow and subsequent output of the cryptographic processing steps depend on the values of the input parameters and the contents of the CF environment memory 146. The CF environment memory 146 consists of a set of cryptographic variables, for example keys, flags, counters, CF configuration data, etc., which are collectively stored within the CF 30. The CF environment variables in memory 146 are initialized via the interface 31, that is by execution of certain CF microinstructions which read input parameters and load them into the CF environment memory 146. Alternately, initialization can be done via an optional physical interface which permits cryptographic variables to be loaded directly into the CF environment memory 146, for example via an attached key entry device.

The physical embodiment of the cryptographic facility secure boundary 140, incorporates the following physical security features. The physical embodiment resists probing by an insider adversary who has limited access to the cryptographic facility 30. The term "limited" is measured in minutes or hours as opposed to days or weeks. The adversary is constrained to a probing attack at the customer's site using limited electronic devices as opposed to a laboratory attack launched at a site under the control of the adversary using sophisticated electronic and mechanical equipment. The physical embodiment also detects attempts at physical probing or intruding, through the use of a variety of electro-mechanical sensing devices. Also, the physical embodiment of the cryptographic facility 30 provides for the zeroization of all internally

stored secret cryptographic variables. Such zeroization is done automatically whenever an attempted probing or intrusion has been detected. The physical embodiment also provides a manual facility for a zeroization of internally stored secret cryptographic variables. Reference to the Abraham, et al. patent application cited above, will give an example of how such physical security features can be implemented.

Initialization of Public-Key Cryptographic System: Fig. 4 illustrates two cryptographic systems, A and B, that wish to communicate cryptographically using public key cryptography. Cryptographic system A generates a public and private key pair (PUa, PRa), where PUa is the public key of A and PRa is the private key of A. In like manner, cryptographic system B generates a public and private key pair (PUb, PRb), where PUb is the public key of B and PRb is the private key of B.

Referring to Fig. 4, the cryptographic facility 30 of cryptographic system A contains a master key KMa and the cryptographic facility 30' of cryptographic system B contains a master key KMb. KMa and KMb are ordinarily different, being equal only by mere chance. At cryptographic system A, the public key PUa is encrypted with the Data Encryption algorithm (DEA) using variant key KMa.C1 to form the encrypted value eKMa.C1-(PUa), where KMa.C1 is formed as the Exclusive OR product of master key KMa and control vector C1. Likewise, at cryptographic system A, the private key PRa is encrypted with the DEA using variant key KMa.C2 to form the encrypted value eKMa.C2(PRa), where KMa.C2 is formed as the Exclusive OR product of master key KMa and control vector C2. The symbol "." denotes the Exclusive OR operation. The encrypted values eKMa.C1(PUa) and eKMa.C2(PRa) are stored in cryptographic key data set 32.

The control vector specifies whether the key is a public or private key and contains other key usage control information specifying how the key may be used. For example, when the encrypted key eKMa.C2(PRa) is decrypted for use within the cryptographic facility 30, control vector C2 indicates to the cryptographic facility how and in what way the key PRa may be used. Control vector C1 similarly controls the use of public key PUa. The use of the control vector to control key usage is described in U.S. Patents 4,850,017, 4,941,176, 4,918,176, 4,924,514, 4924,515, and 5,007,089 cited in the background art and in co-pending patent application serial number 07/602,989 also cited in the background art. Fig. 11 illustrates control vectors that define public and private keys, where the public and private keys are key management keys used by the cryptographic system to distribute DEA keys. The fields in each control vector consist of a CV TYPE, which specifies whether the control vector is a public or a private key and additionally whether the key pair is a key management key pair for use in distributing DEA keys or whether the key pair is some other kind of key pair. Other types of key pairs are possible, such as user keys which can be used for generation and verification of digital signatures but not for key distribution. Each control vector has a PR USAGE and PU USAGE field. For the public key control vector, the PU USAGE field controls the usage of the public key in cryptographic instructions whereas the PR USAGE field is only informational. For the private key control vector, the PR USAGE field controls the usage of the private key in cryptographic instructions whereas the PU USAGE field is only informational. The ALGORITHM field indicates the public key algorithm to which this key pair pertains. The HIST field records history information, e.g., the options used to import a public key (see co-pending patent application serial number 07/602,989 as cited in the background art, which describes the use of history information fields in the public key control vector). The reader will appreciate that the control vector may contain a variety of different control vector fields for the purpose of controlling the operation and use of the key within the cryptographic network and cryptographic systems within the network.

In an alternate embodiment, the public key PUa may be stored in an unencrypted form, since there is no intent to keep the value of this key secret. Encrypting PUa is done for sake of uniformity, so that all keys in the cryptographic key data set 32 are stored and recovered using one common method. Those skilled in the art will also recognize that the length of PUa and PRa will likely be different than the block size of the DEA, which is 64 bits, and hence PUa and PRa may need to be encrypted in separate 64-bit pieces. The particular method for encrypting PUa and PRa is unimportant to the invention. However, one way that this encryption can be carried out is to use the Cipher Block Chaining (CBC) mode of DEA encryption described in DES modes of operation, Federal Information Processing Standards Publication 81, National Bureau of Standards, US Department of Commerce, December 1980. In cases where KMa is a 128-bit key, the CBC mode of DEA encryption can be adapted to encrypt PUa under KMa. PUa is first encrypted with the leftmost 64 bits of KMa, then decrypted with the rightmost 64 bits of KMa, and then encrypted again with the leftmost 64 bits of KMa.

In like manner, at cryptographic system B, the public and private keys, PUb and PRb, are encrypted with master key KMb and control vectors C3 and C4, per the same method described for cryp-

tographic system A. The encrypted values eKMb.C3(PUb) and eKMb.C4(PRb) are stored in cryptographic key data set 32'. Control vectors C3 and C4 control the usage of PUb and PRb, respectively.

Although encryption of the public and private keys has been described in terms of a DEA-based master key, those skilled in the art will appreciate that the DEA could be replaced by a public key algorithm and the master keys could be replaced by a PKA-based key pair used for this purpose. Moreover, the encryption of the public and private keys has been described in terms of encryption of the keys only. In some implementations it may be more practical to imbed these keys within key records that contain other key-related information besides the keys themselves.

In order for cryptographic systems A and B to carry out cryptographic operations using their respective implemented public key algorithms, they must share their public keys with each other. Thus, at cryptographic system A a function exists that permits the encrypted value eKMa.C1(PUa) to be accessed from cryptographic key data set 32 and decrypted so that the clear value of PUa may be exported to cryptographic system B at 300. At cryptographic system B a function exists that permits the clear value of PUa to be imported and encrypted under the variant key KMb.C1. The so-imported encrypted value eKMb.C1(PUa) is then stored in cryptographic key data set 32'. In like manner, functions exist at B and A that permit public key PUb to be decrypted at B, sent to A at 301, and re-encrypted at A for storage in A's cryptographic key data set.

Co-pending patent application by S. M. Matyas et al., serial number 07/602,989, "Method and Apparatus for Controlling the use of a Public Key, Based on the Level of Import Integrity for the Key," describes a method for generating public and private keys and for distributing public keys in order to initialize a public-key cryptographic system, and is incorporated by reference herein.

Key Distribution: Fig. 5 illustrates the process by which cryptographic system A may distribute a key to cryptographic system B using a public key algorithm (PKA). That is, it illustrates the process of key distribution using a PKA. In a hybrid key distribution scheme, the distributed key is a DEA key, e.g., an initial key-encrypting key to be used later with a DEA-based key distribution scheme to distribute all subsequent DEA keys. However, any key can be distributed using the so-described PKA-based key distribution scheme, including both DEA keys and PKA keys. The distributed DEA and PKA keys can be of any type or designated use. However, for purposes of illustration, Fig. 5 shall assume that the distributed key is a DEA key.

Referring to Fig. 5, the steps involved in distribution of a key from cryptographic system A to cryptographic system B are these. At cryptographic system A, a Generate Key Set PKA (GKSP) instruction is executed within the CF 30. Control information at 303 is provided to the GKSP instruction as input. In response, the GKSP instruction generates a key K and produces two encrypted copies of K, which are returned by the GKSP instruction at 305 and 306. The first encrypted copy of K is produced by encrypting K with the DEA using variant key KMa.C5 formed as the Exclusive OR product of master key KMa and control vector C5. C5 may be input to the GKSP instruction as part of the control information, at 303, or it may be produced within the CF 30 as part of the GKSP instruction, or it may be produced as a combination of both methods. The second encrypted copy of K is produced as follows. A key block (designated keyblk) is first formed. The key block includes the clear value of K, control information, and possibly other information unimportant to the present discussion, as illustrated in Fig. 6. The format of the keyblk is unimportant to the present discussion, and those skilled in the art will recognize that many possible arrangements of the keyblk information are possible. In all cases, the keyblk contains the necessary information to accomplish the task of key distribution. The length of the keyblk is assumed to be equal to the block size of the public key algorithm. For example, if the public key algorithm is the RSA algorithm, then the block size is just the modulus length. Also, it is assumed that the numeric value of the keyblk, say its binary value, is adjusted as necessary to permit it to be encrypted as a single block by the public key algorithm. For example, if the public key algorithm is the RSA algorithm, then the keyblk is adjusted so that its binary value is less than the binary value of the modulus. This can be done by forcing the high order (most significant) bit in the keyblk to zero. Once the keyblk has been formatted, it is encrypted with the public key PUb of cryptographic system B to form the encrypted value ePUb(keyblk), which is returned at 306. To permit this to be accomplished, the encrypted value eKMa.C3(PUb) and control vector C3 are supplied to the GKSP instruction at 304 as inputs and eKMa.C3(PUb) is decrypted under variant key KMa.C3. KMa.C3 is formed as the Exclusive OR product of master key KMa stored within the CF 30 and control vector C3.

The first encrypted output eKMa.C5(K) at 305 is stored in the cryptographic key data set 22 of cryptographic system A. Control vector C5 is also stored in the cryptographic key data set 22 together with the encrypted key eKMa.C5(K). In some implementations it may be convenient to

store eKMa.C5(K) and C5 in an internal key token together with other key-related information. The internal key token is not relevant to the present discussion, and is therefore not shown in Fig. 5. If C5 is generated within the CF 30, it may also be provided as an output at 305 so that it may be store in CKDS 22.

The second encrypted output ePUb(keyblk) at 306 is formatted within an external key token 308. The external key token contains the encrypted key or encrypted key block ePUb(keyblk), control information, and other information unimportant to the present discussion, as shown in Fig. 7. The control information supplied as input to the GKSP instruction at 303 is also stored in external key token 308 at 307. However, the control information at 307 may include additional information available to the cryptographic facility access program (CFAP) which is not specified as an input to the GKSP instruction at 303. In other words, the source of the control information in the external key token 308 may be much broader than the control information supplied as input the the GKSP instruction at 303. One example, is the Environment Identifier (EID) value stored both in the CF 30 and in the CFAP. The EID value is an identifier that uniquely identifies each cryptographic facility or cryptographic system within a network. The EID value is loaded into the CF 30 during an initialization sequence prior to performing routine cryptographic operations within the cryptographic system. Another example of initialization is the loading of the master key KMa. The EID value need not be supplied to the CF since it is already stored in the CF. But the EID value may be stored within the external key token, in which case it is supplied as an input at 307. In like manner, the control information in the keyblk may include a control vector C6 specifying the usage of K at cryptographic system B. In that case, C6 may be supplied as part of the control information at 303, in which case it is also supplied as part of the control information at 307. If however C6 is generated within CF 30, then C6 is not supplied as part of the control information at 303, but is supplied as part of the control information at 307. Those skilled in the art will recognize that various alternatives exist for the specification or derivation of the necessary control information and that different combinations of inputs to the GKSP instruction and to the external key token are therefore possible.

The formatted external key token 308 is transmitted to cryptographic system B where it is processed. The CFAP at B first checks the control information in the external key token for consistency. For example, if the control information contains a control vector C6, then C6 is checked to ensure that it represents a key type and key usage approved by cryptographic system B. Likewise, if the control information contains an EID value, then the EID value is checked to ensure that the external key token and the key to be imported originated from cryptographic system A, i.e., it originated from the expected or anticipated cryptographic system that B 'thinks' it is in communication with and which it desires to establish a keying relationship. Once this has been accomplished, the received key is imported as follows. The encrypted keyblk, ePUb(keyblk) and part or all of the control information in the external key token are supplied as inputs to an Import DEA Key (IDK) instruction at 309, which is executed within CF 30' at cryptographic system B. In response, the IDK instruction decrypts ePUb(keyblk) under the private key PRb belonging to cryptographic system B. To permit this to be accomplished, the encrypted value eKMb.C4(PRb) and control vector C4 are supplied to the IDK instruction at 310 as inputs and eKMb.C4(PRb) is decrypted under variant key KMb.C4. KMb.C4 is formed as the Exclusive OR product of master key KMb stored within the CF 30' and control vector C4. Once ePUb(keyblk) has been decrypted and the clear value of keyblk has been recovered, the keyblk is processed as follows. The control information contained in the keyblk is checked for consistency against the control information, or reference control information, supplied as input at 309. If the consistency checking is satisfactory (okay), then the clear value of K is extracted from keyblk and it is encrypted with the variant key KMb.C6 to produce the encrypted key value eKMb.C6(K). KMb.C6 is formed as the Exclusive OR product of master key KMb stored within CF 30' and control vector C6. Control vector C6 may be obtained in different ways. C6 may be contained in the control information in keyblk, in which case it is extracted from keyblk. In other cases, C6 may be produced within CF 30'. For example, if there is only one key type and key usage permitted, then C6 can be a constant stored within the IDK instruction. The so-produced encrypted key value eKMb.C6(K) is provided as an output of the IDK instruction at 311, and is stored together with its control vector C6 within CKDS 22'. The value of C6 stored in CKDS 22' is obtained either from the control information input to the IDK instruction at 309 or, if C6 is not in the control information input to the IDK instruction at 309, then it is produced by the CFAP in the same way that it is produced by the IDK instruction and stored in CKDS 22'. Alternatively, C6 could be returned as an output of the IDK instruction. Those skilled in the art will realize that several alternatives exist for obtaining C6 depending on how and where it is produced within the cryptographic system and whether it is or is not included as part of the

control information in the external key token.

In the preferred embodiment, the control information at 303 supplied to the GKSP instruction includes a specification of control vectors C5 and C6. This allows the GKSP instruction the freedom and flexibility to generate two encrypted copies of key K that have different key types and usages, as specified by C5 and C6. In that case, the GKSP instruction must incorporate some control vector checking to determine that C5 and C6 constitutes a valid pair. The various options for control vector design and checking pursued here are based on the control vector designs included in prior art, cited in the background art, and already discussed. Likewise, in the preferred embodiment, control vector C6 is included in the control information in the key block (keyblk) and also in the control information in the external key token. This permits the receiving cryptographic system to import keys of different types while still permitting the receiving system to verify that the imported key is one that it wants or expects. This is accomplished by the CFAP first checking the control vector in the external key token to make sure that it prescribes a key type and key usage that it expects or will allow to be imported. C6 is then supplied as an input in the control information at 309 to the CF 30'. At the time the IDK instruction recovers the clear value of keyblk, the value of C6 in the control information in keyblk is checked against the value of C6, or the reference value of C6, supplied as input. This permits the CF to verify that the value of C6 used to import the key K is the same control vector C6 in the external key token. Otherwise, if this check was ignored it would be possible for an adversary to substitute C6' for C6 in the external key token, causing a key to be imported that the CFAP may not permit.

In the preferred embodiment, each cryptographic facility stores a unique EID value, e.g., a 128-bit value set within the CF during an initialization sequence before routine operations are permitted. At the time a keyblk is prepared within the CF by a GKSP instruction, the EID value is obtained from the CF and included within the control information in the keyblk. In like manner, a duplicate copy of the EID value is stored outside the CF with integrity such that it is available to the CFAP. This EID value is obtained by the CFAP and is included within the control information in the external key token. Thus, the CFAP at the receiving cryptographic system can check the EID value in the control information of the received external key token to ensure that the external key token originated from the cryptographic system that is expected or anticipated. That is, B knows that the external key token came from A, which is what is expected. The EID value is also supplied as part of the control information at 309. Thus, when the IDK instruction obtains the clear keyblk, the EID value in the control information in the clear keyblk can be checked against the EID value, or reference EID value, supplied as an input. In this way, the CFAP is sure that the IDK instruction will import K only if the two EID values are equal. This prevents an adversary from changing the EID value in the external key token to a different value that might also be accepted by the receiving device. This might lead to a situation where B imports a key from A, thinking that it came from C.

The EID also serves another purpose, as now described. At the time the clear value of keyblk is obtained by the IDK instruction, a check is performed to ensure that the value of EID in the control information in keyblk is not equal to the value of EID stored in the CF at the receiving device. Thus, the encrypted value of ePUx(keyblk) produced at cryptographic system A, where PUx may be the public key of any cryptographic system in the network, including A itself, cannot be imported by A. This prevents an adversary at A, who specifies his own public key PUa to the GKSP instruction, from importing ePUa(keyblk) at A and thereby obtaining two encrypted copies eKMa.C5-(K) and eKMa.C6(K) of the same key with potentially different key types and key usage attributes. In some cases, bi-functional key pairs are undesirable and the key management design will specifically disallow such key pairs to be created using the key generation facilities provided by the key management services.

Key Distribution with Digital Signatures: The key distribution scheme described in Fig. 5 is not by itself the preferred embodiment of the invention. This is so because, as it stands, the scheme can be attacked by an adversary who knows the public value of B's key, PUb. In public key cryptographic systems, one naturally makes the assumption that PUb is known by anyone, even an adversary. The adversary can forge values of keyblk containing DEA keys of his choosing and freely encrypt these key blocks under PUb. Thus, at B, there is no way to know that an imported key originated with A or with an adversary posing as A. The importing function will import the forged values of keyblk, which results in known values of K being encrypted under the master key, of the form eKMb.C6(K), and stored in CKDS 22'. In that case, data or keys encrypted under K are easily deciphered by the adversary who knows K.

The preferred embodiment of the invention therefore includes a means by which the receiver, say cryptographic system B, can ensure that a received encrypted keyblk of the form ePUb-(keyblk) did in fact originate with the intended sender, say cryptographic system A. To accom-

plish this, the GKSP instruction at cryptographic system A produces a digital signature (designated DSIGa) on ePUb(keyblk) using its private key PRa. The so-produced digital signature is transmitted together with the external key token to cryptographic system B where the key is imported using an IDK instruction. In this case, the IDK instruction first verifies the digital signature DSIGa using the previously imported copy of PUa received from cryptographic system A. Only after DSIGa has been successfully verified will the IDK instruction continues as already described in Fig. 5 and import the key K.

Fig. 8 illustrates the scheme for DEA key distribution with digital signatures, which is the same as the scheme shown in Fig. 5 except as follows. Once the encrypted key value ePUb(keyblk) has been produced, the GKSP instruction additionally produces the digital signature DSIGa from ePUb(keyblk) and the private key PRa belonging to cryptographic system A. A common method for producing such a signature is to first calculate a hash value on ePUb(keyblk) using a one way cryptographic function, such as described in U. S. patent 4,908,861 by Brachtl et al., cited in the background art, which uses either two DEA encryptions or four DEA encryptions per each 64 bits of input text to be hashed, and then decrypt (or transform) the hash value using the private key PRa to produce a DSIGa of the form dPRa(hash value). The clear value of PRa is obtained by decrypting the encrypted value of eKMa.C2(PRa) supplied as an input to the GKSP instruction at 313 using the DEA and the variant key KMa.C2. KMa.C2 is formed as the Exclusive OR product of master key KMa stored in CF 30 and control vector C2 supplied as input to the GKSP instruction at 313. For example, if the public key algorithm is the RSA algorithm, a the digital signature may be calculated using the method as described in ISO Draft International Standard 9796 entitled "Information Technology -- Security Techniques -- Digital Signature Scheme Giving Message Recovery." The so-produced DSIGa 315 is returned as an output at 314. Both the external key token 308 and the DSIGa 315 are transmitted to cryptographic system B. At cryptographic system B, the IDK instruction is used to import the key K in similar fashion as described in Fig. 5 except that the IDK instruction first validates DSIGa using the public key PUa previously imported, encrypted, and stored in CKDS 22'. A DSIGa of the form dPRa(hash value) is validated by encrypting dPRa(hash value) with PUa, calculating a hash value on ePUb(keyblk) using the same one way cryptographic function, called the hash value of reference, and comparing the hash value of reference and the recovered clear hash value for equality. Only if this comparison check is successful does the IDK instruction continue and import the key K. The clear value of PUa is obtained by decrypting the encrypted value of eKMb.C1(PUa) supplied as an input to the IDK instruction at 316 using the DEA and the variant key KMb.C1. KMb.C1 is formed as the Exclusive OR product of master key KMb stored in CF 30' and control vector C1 supplied as input to the IDK instruction at 316. Thus, the GKSP instruction at cryptographic system A produces DSIGa and the IDK instruction at cryptographic system B verifies DSIGa. In an alternate embodiment, DSIGa can be calculated by the CF 30 using a separate instruction for generating digital signatures. In that case, after the GKSP instruction has been executed, the CFAP invokes the generate digital signature instruction causing DSIGa to be generated. In like manner, DSIGa can be verified by the CF 30' using a separate instruction for verifying digital signatures. In that case, before the IDK instruction is invoked, the CFAP invokes the verify digital signature instruction to ensure that DSIGa is valid.

Generate Key Set PKA (GKSP) Instruction: Fig. 9 illustrates the Generate Key Set PKA (GKSP) instruction. The GKSP instruction of Fig. 9 is identical to the GKSP instruction contained within the CF 30 of Fig. 8. The GKSP instruction generates a two encrypted copies of a generated DEA key K. The first copy is of the form eKM.C5(K) and is stored in the cryptographic key data set of the generating cryptographic device, say A. The second copy is of the form ePU(keyblk) and is transmitted to a designated receiving cryptographic device, say B, where the public key PU belonging to the receiving cryptographic device B. Also, the GKSP instruction produces a digital signature DSIG on ePU(keyblk) using the private key PR of the generating cryptographic device A. DSIG is also transmitted to cryptographic device B to serve as proof that ePU(keyblk) was produced at cryptographic device A, i.e., produce at a valid network cryptographic device.

Referring to Fig. 9, GKSP instruction 500 consists of control information retrieval means 504, PU recovery means 506, PR recovery means 507, key generation means 508, eKM.C5(K) production means 509, ePU(keyblk) production means 510, DSIG production means 511, and hash algorithms 512. GKSP instruction 500 is located in instruction processor 142 within cryptographic facility 30, as shown in Fig. 3. The inputs to the GKSP instruction are supplied to the GKSP instruction by CFAP 34, i.e., by the CFAP 34 to the CF 30 across the CFAP-to-CF interface. In similar manner, the outputs from the GKSP instruction are supplied to the CFAP 34, i.e., by the CF 30 to the CFAP 34 across the CFAP-to-CF interface.

The inputs to GKSP instruction 500 are (1) at 501, control information such as control vectors C5 and C6 that specify the key usage attributes of the two encrypted copies of the generated DEA key K, (2) at 502, control vector C3 and encrypted public key eKM.C3(PU), where C3 specifies the key usage attributes of public key PU belonging to the receiving cryptographic device, and (3) at 503, control vector C2 and encrypted private key eKM.C2(PR), where C2 specifies the key usage attributes of private key PR belonging to the sending or generating cryptographic device. The outputs from GKSP instruction 500 are (1) at 521, the encrypted key, eKM.C5(K), where K is encrypted under variant key KM.C5 formed as the Exclusive OR product of master key KM and control vector C5, (2) at 522, the encrypted key block, ePU-(keyblk), where keyblk is encrypted under public key PU belonging to the intended receiving cryptographic device and where keyblk is a key block containing the generated DEA key K, control information, and possibly other information as depicted in Fig. 6, and (3), at 523, a digital signature DSIG generated on ePU(keyblk) using the private key PR belonging to the sending or generating cryptographic device.

Control information retrieval means 504 accepts and parses control information supplied as input to the GKSP instruction at 501. Also, control information retrieval means 504 accesses control information stored within the secure boundary of the cryptographic facility, e.g., the Environment Identifier (EID) at 505. Control information retrieval means 504 may also perform consistency checking on the assembled control information. For example, control vectors C5 and C6 may be checked and cross checked for consistency, i.e., to ensure they are a valid control vector pair. GKSP instruction 500 is aborted if C5 and C6 are incorrect or do not specify the correct key usage required by the GKSP instruction. In an alternate embodiment, it may be possible for control information retrieval means 504 to generate or produce control vector C6 from control vector C5, or vice versa, in which case only one control vector is specified in the control information supplied at 501. In that case, cross checking of C5 and C6 is unnecessary. Control vector checking of C5 or C6 can be performed in control information retrieval means 504 or in eKM.C5(K) production means 509 if the control vector is C5 or in ePU(keyblk) production means 510 if the control vector is C6. The reader will appreciate that control vector checking may be accomplished in variety of ways within the different components parts of the GKSP instruction, and that these variations do not significantly depart of the general framework of the invention. In any event, control information retrieval means 504 makes the control information available to other component parts of the GKSP instruction. C5 is passed to eKM.C5(K) production means 509 and EID and C6 are passed to ePU(keyblk) production means 510. Optionally, control information may also be passed to DSIG production means 511 such as the identifier or name of a hashing algorithm to be used in the preparation of the digital signature. The GKSP instruction may support only one hashing algorithm in which case the identifier or name of a hashing algorithm need not be passed to DSIG production means. Those skilled in the art will recognize that many possible variations exist for inputting and accessing control information, for parsing, checking and making the control information available to different component parts of the GKSP instruction.

PU recovery means 506 decrypts input eKM.C3(PU) under variant key KM.C3 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to GKSP instruction 500 and control vector C3 specified as an input to GKSP instruction 500. Prior to decrypting eKM.C3(PU), PU recovery means 506 performs control vector checking on C3. GKSP instruction 500 is aborted if C3 is incorrect or does not specify the correct key usage required by the GKSP instruction. Public key PU is stored in encrypted form so that PU Recovery means 506 will be, for all practical purposes, identical to PR Recovery means 507. Encryption of PU is also preferred since it permits control vector C3 to be cryptographically coupled with public key PU. Even though PU is public, and there is no need to protect the secrecy of PU, encryption of PU thus ensures that PU can be used only if C3 is correctly specified as an input to the GKSP instruction. This ensures that PU is used by the GKSP instruction only if it has been so designated for use. In an alternate embodiment, PU could be stored outside the cryptographic facility in clear form and PU Recovery means 506 could be omitted from GKSP instruction 500. In this case, the embodiment may choose to fix the usage attributes of PU so that there is no chance for an adversary to specify a control vector C3 that is incorrect, i.e., C3 is a fixed constant value. In any event, the recovered clear value of PU is supplied as an input to ePU(keyblk) production means 510.

PR recovery means 507 decrypts input eKM.C2(PR) under variant key KM.C2 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to GKSP instruction 500. Prior to decrypting eKM.C2(PR), PR recovery means 507 performs control vector checking on C2. GKSP instruction 500 is aborted if C2 is incorrect or does not specify the correct key usage required by the GKSP instruction. The recovered clear value of PR

is supplied as an input to DSIG production means 511.

Key generator means 508 is a pseudo random number generator for generating DEA keys. Alternatively, key generator means 508 could be a true random number generator. For sake of simplicity, key generator means 508 generates 64-bit random numbers which are adjusted for odd parity. That is, the eight bit of each byte in the generated random number is adjusted so that the value in each byte is odd. DEA keys may contain either 64 or 128 bits depending on their intended usage. Data-encrypting-keys used for encrypting data are 64-bit keys. Key-encrypting-keys used for encrypting keys are generally 128-bit keys, but may in some cases be 64-bit keys. To produce a 128-bit key, key generator means 508 is invoked twice. The so-generated DEA key is supplied as an input to both eKM.C5(K) production means 509 and ePU(keyblk) production means 510.

eKM.C5(K) production means 509 Exclusive ORs input KM and C5 to produce variant key KM.C5 and then encrypts input K with KM.C5 to form the encrypted output eKM.C5(K), which is returned to the CFAP at 521. If control vector C5 is not consistency checked in control information retrieval means, it may alternatively be checked here.

ePU(keyblk) production means 510 first prepares a key block, designated keyblk, from the inputs K, EID, and C6, and then encrypts keyblk with public key PU to form the encrypted output ePU(keyblk), which is returned to the CFAP at 522. If control vector C6 is not consistency checked in control information retrieval means, it may alternatively be checked here. The value ePU(keyblk) is also supplied as an input to DSIG production means 511 to allow the digital signature DSIG to be produced. The format of keyblk is shown in Fig. 6 and has been discussed previously. The procedure of preparing keyblk accomplishes two main goals. It ensures that all necessary information such as the key, control information, key-related information, keyblk parsing information, etc. is included within keyblk. Also, it ensure that keyblk is constructed in a way that keyblk can be encrypted with PU using the public key algorithm. For example, it may be necessary to pad keyblk so that its length and binary value are such that keyblk is encrypted properly and in conformance with restrictions imposed or that may be imposed by the public key algorithm.

DSIG production means 511 produces a digital signature on ePU(keyblk) using private key PR. To accomplish this, a hash value is first calculated on ePU(keyblk) using hash algorithm 512. Hash algorithm 512 may in fact be a set of hash algorithms. In that case, the hash algorithm is selected on the basis of a hash algorithm identifier or other appropriate encoded value passed by the control information retrieval means 504 to the DSIG production means 511. The so-produced hash value is then formatted in a suitable signature block and decrypted with private key PR to produce DSIG, which is returned to the CFAP at 523. The signature block can in the simplest case consist of the hash value and padding data, so as to construct a signature block whose length and value are in conformance with restrictions imposed or that may be imposed by the public key algorithm, as already discussed above. The DSIG production means 511 may also implement a digital signature method based on a national or international standard, such as International Standards Organization draft international standard (ISO DIS) 9796.

Import DEA Key (IDK) Instruction: Fig. 10 illustrates the Import DEA Key (IDK) instruction. The IDK instruction of Fig. 10 is identical to the IDK instruction contained within the CF 30 of Fig. 9 The IDK instruction permits a cryptographic device, say B, to import an encrypted DEA key of the form ePU(keyblk) that has been received from a sending cryptographic device, say A. The received digital signature DSIG is used by the IDK instruction to verify that ePU(keyblk) originated with cryptographic device A, i.e., at a valid network cryptographic device.

Referring to Fig. 10, IDK instruction 600 consists of PU recovery means 606, PR recovery means 607, control information retrieval means 608, hash algorithms 610, DSIG verification means 611, keyblk recovery means 612, eKM.C6(K) production means 613, and control information consistency checking means 614. IDK instruction 600 is located in instruction processor 142 within cryptographic facility 30, as shown in Fig. 3. The inputs to the IDK instruction are supplied to the IDK instruction by CFAP 34, i.e., by the CFAP 34 to the CF 30 across the CFAP-to-CF interface. In similar manner, the outputs from the IDK instruction are supplied to the CFAP 34, i.e., by the CF 30 to the CFAP 34 across the CFAP-to-CF interface.

The inputs to the IDK instruction 600 are (1) at 601, control vector C1 and encrypted public key eKM.C1(PU), where C1 specifies the key usage attributes of public key PU belonging to the sending cryptographic device, (2) at 602, digital signature DSIG, (3) at 603, encrypted key block ePU(keyblk), where keyblk is encrypted under public key PU belonging to the the receiving cryptographic device and where keyblk is a key block containing the to-be-imported DEA key K, control information, and possibly other information as depicted in Fig. 6, (4) at 604, control vector C4 and encrypted private key eKM.C4(PR), where C4 specifies the key usage attributes of private key PR belonging to the receiving cryptographic device, and (5) at 605,

control information, such as a reference control vector C6 and a reference EID value of the sending cryptographic device. The output of the IDK instruction 600 is the encrypted key eKM.C6(K), where K is the to-be-imported DEA key encrypted under variant key KM.C6 formed as the Exclusive OR product of master key KM and control vector C6.

PU recovery means 606 decrypts input eKM.C1(PU) under variant key KM.C1 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to IDK instruction 600 and control vector C1 specified as an input to IDK instruction 600. Prior to decrypting eKM.C1(PU), PU recovery means 606 performs control vector checking on C1. IDK instruction 600 is aborted if C1 is incorrect or does not specify the correct key usage required by the IKK instruction. Public key PU is stored in encrypted form so that PU recovery means 606 well be, for all practical purposes, identical to PR recovery means 607 Encryption of PU is also preferred since it permits control vector C1 to be cryptographically coupled with public key PU, as argued previously under the description of the GKSP instruction. In an alternate embodiment, PU could be stored outside the cryptographic facility in clear form and PU recovery means 606 could be omitted from IDK instruction 600. In this case, the embodiment may choose to fix the usage attributes of PU so that there is no chance for an adversary to specify a control vector C1 that is incorrect, i.e., C1 is a fixed constant value. In any event, the recovered clear value of PU is supplied as an input to DISG verification means 611.

PR recovery means 607 decrypts input eKM.C4(PR) under variant key KM.C4 formed as the Exclusive OR product of master key KM stored in clear form within the cryptographic facility and directly accessible to IDK instruction 600. Prior to decrypting eKM.C4(PR), PR recovery means 607 performs control vector checking on C4. IDK instruction 600 is aborted if C4 is incorrect or does not specify the correct key usage required by the IDK instruction. The recovered clear value of PR is supplied as an input to keyblk recovery means 612.

Control information retrieval means 608 accepts and parses control information supplied as input to the IDK instruction at 605. Also, control information retrieval means 608 accesses control information stored within the secure boundary of the cryptographic facility, e.g., the Environment Identifier (EID) at 609. Control information retrieval means 608 supplies control information to control information consistency checking means 614 and possibly to other component parts of the IDK instruction, such as a hash algorithm identifier supplied to DSIG verification means 611 (not shown in Fig. 10).

DSIG verification means 611 uses public key PU belonging to the sending cryptographic device to verify the digital signature DSIG generated on ePU(keyblk) at the sending cryptographic device. To accomplish this, a hash value is first calculated on ePU(keyblk) using hash algorithm 512. Hash algorithm 512 may in fact be a set of hash algorithms. In that case, the hash algorithm is selected on the basis of a hash algorithm identifier or other appropriate encoded value passed by the control information retrieval means 608 to the DSIG verification means (not shown in Fig. 10). The clear public key PU obtained from PU recovery means 606 is then used to encrypt the value of DSIG specified as an input at 602. This recovers the original signature block in clear form, which is then parsed to recover the original hash value. The recovered hash value and the calculated hash value are then compared for equality. If this comparison is favorable, then DSIG is considered valid; otherwise, DSIG is not considered valid and IDK instruction 600 is aborted. The signature block recovery and processing of course will depend on the method of digital signature implemented. In the description of the GKSP instruction it was indicated that the signature block may consist of the hash value and padding data or it may be constructed on the basis of a national or international standard, such as International Standards Organization draft international standard (ISO DIS) 9796. Those skilled in the art will appreciate that many possible implementations of the digital signature are possible and that the precise method of digital signatures is unimportant to the invention. What is important is that a method of digital signature is used in the preferred embodiment to ensure that the receiving cryptographic device can authenticate that the to-be-imported DEA key did in fact originate from a valid network cryptographic device. As the reader will also see, the digital signature is made an integral part of the GKSP and IDK instructions themselves, which ensures that the process of signature production and signature verification occurs as part of the key export and key import processes and therefore the highest possible integrity over these processes is achieved. Although it is possible to perform signature production and signature verification as separate instructions, which achieves complete compatibility with the present descriptions of the GKSP and IDK instructions, one also sees that less integrity is achieved. This is so because the signature generate instruction has no way to ensure that a key of the form ePU(keyblk) was in fact produced by the GKSP instruction.

Keyblk recovery means 612 decrypts input ePU(keyblk), provided as an input to the IDK in-

struction at 603, under private key PR, provided as an output of PR recovery means 607. The recovered clear key block, keyblk, is provided as an output to both eKM.C6(K) production means 613 and control information consistency checking means 614.

Control information consistency checking means 614 checks the control information in the recovered keyblk output from keyblk recovery means 612 and the reference control information output from control information retrieval means 608 for consistency. A first check consists in checking control vector C6 in keyblk for consistency with the reference control vector C6 supplied as an input to the IDK instruction at 605. This ensures that the receiving cryptographic application imports a key from with the expected or intended key usage attributes. In this case, reference control vector C6 represents the expected control vector, whereas the recovered control vector C6 represents the actual control vector. The simplest form of consistency checking consists of checking these two control vectors for equality. However, a more refined procedure is possible wherein attributes in the reference control vector are allowed to override corresponding attributes in the recovered control vector. For example, the reference control vector could disable the ability to re-export the imported DEA key K, whereas the recovered control vector may or may not permit the imported DEA key K to be re-exported. More generally, the receiving device may disable any attribute granted within the received control vector. One will appreciate that taking away a right is not the same as granting a right, which only the sending cryptographic device is permitted to do. The IDK instruction can be designed to permit this kind of control vector override or it may not, depending on the desires of the designer of the IDK instruction. A second check consists of checking the EID value in keyblk for equality with the reference EID value supplied as an input to the IDK instruction at 605. This ensures that the receiving cryptographic application imports a key from the expected or intended sending cryptographic device. In this case, the reference EID value is the EID of the intended sending cryptographic device, which is checked against the EID value in keyblk which represents the EID value of the actual sending cryptographic device. A third check consists of checking the EID value in keyblk for inequality with the EID value stored in the cryptographic facility of the receiving device. This ensures that the imported DEA K originated at another cryptographic device, i.e., that A can't import a K produced at A, that B can't import a K produced at B, etc. The usefulness of this check has been discussed previously. In all cases, if the consistency checking fails, then the IDK instruction is aborted.

eKM.C6(K) production means 613 extracts the clear value of DEA key K and the control vector C6 from keyblk, obtained as an output from the keyblk recovery means 612. and K is then encrypted under variant key KM.C6 formed as the Exclusive OR product of master key KM and control vector C6 recovered from C6 to produce the encrypted key value eKM.C6(K). In an alternative embodiment where the reference control vector C6 can override the recovered control vector C6, the value of C6 used to form the variant key KM.C6 can be the reference control vector C6. In yet another alternative embodiment the IDK instruction itself can modify information in the control vector C6, so that K is encrypted with variant key KM.C6', where C6' is the IDK modified value of C6. In any event, the encrypted key eKM.C6(K) is returned to the CFAP as an instruction output at 615.

The reader will appreciate that the IDK instruction has been designed to perform consistency checking within the cryptographic facility in lieu of returning the recovered clear values of C6 and EID to the CFAP and performing this consistency checking outside of the cryptographic facility. In the preferred embodiment, this consistency checking is performed in the cryptographic facility hardware and the recovered clear values of C6 and EID are not exposed outside the CF. The reason for doing this is to ensure that the DEA key distribution channel does not also provide a covert privacy channel whereby secret data may be incorporated in the control information portion of the key block and transmitted from the sending cryptographic device to the receiving cryptographic device. In a good cryptographic design, the cryptographic instructions will perform only those cryptographic functions for which they were designed, and no more. Doing so, limits the ways in which an attacker can manipulate the cryptographic instructions for the purpose of subverting their intended security. For example, a system administrator in charge of security policy for the sending and recieving locations, may have a security policy which prohibits the transmission of private messages over the communications link, for example when the link is dedicated merely to the transmission of new keys. In an alternate security policy where the system administrator is to selectively allow privacy channels, there should be no "back door" method for subverting the system administrator's authority in enabling or prohibiting such privacy channels. The use of the control information transmitted over the separate channel to the receiver, is to enable the recipient to inspect the type of uses imposed on the receive key and allow the recipient the option of rejecting the keyblock. However, an alternate embodiment is possible wherein the recovered

clear values of C6 and EID are returned to the CFAP and consistency checking is then performed by the CFAP.

Control Information: Fig. 12 further illustrates the unique role played by the encrypted key block, ePU(keyblk), and the external key token. Although Figs. 5, 7, and 8 depict external key token as containing an encrypted key block of the form ePU(keyblk) and reference control information in clear form, in the logical sense there are two information channels over which information flows: (1) and encrypted channel and (2) a clear channel. Referring now to Fig. 12, therein is shown two cryptographic systems, A and B, that communicate via a key distribution protocol through an encrypted channel 701 and a clear channel 702. Encrypted channel 701 is facilitated via the encrypted key block, ePU(keyblk). Control information in keyblk, which is subsequently encrypted with public key PU, is thus sent from A to B via an encrypted channel. Clear channel 702 is facilitated via the external key token. Control information in clear form stored in the external key token is, for all intents and purposes, passed from A to B via a clear channel.

Another distinguishing feature of the two channels is this. Encrypted channel 701 is a logical channel between the cryptographic facility 30 of cryptographic system A and cryptographic facility 30' of cryptographic system B. Clear channel 702 is a logical channel between application 36 in cryptographic system A and application 36' in cryptographic system B. and possibly a logical channel between CFAP 34 in cryptographic system A and CFAP 34' in cryptographic system B, depending on how the external key tokens are to be managed. In any event, the key distribution process is designed such that (1) in the case of encrypted channel 701, only the cryptographic facilities have access to the control information in keyblk, whereas (2) in the case of clear channel 702, the applications and possibly the CFAPs have access to the control information in the external key token as a routine part of the key distribution protocol. Since the CF is typically implemented within secure hardware, the CF is said to have a higher level of integrity than other parts of the the cryptographic system, such as the CFAP and applications operating within the cryptographic system. Thus, a higher degree of protection is achieved within the key distribution process by controlling that process, to the degree possible, from within the CF itself. To this end, control information is passed via encrypted channel 701, in keyblk, from A to B, thus enabling B to process the imported keyblk with a high degree of integrity. Of course, the assumption is made here that digital signatures are also a part of the key distribution process, as shown in Fig. 8, which forms another underpinning or layer of integrity that augments and enhances the overall integrity of information passed via encrypted channel 701.

Fig. 13 illustrates the process of reconciliation between control information transmitted via encrypted channel 701 and control information, called reference control information, transmitted via clear channel 702. The importance in having these two channels for passing control information can now be seen. Control information transmitted via encrypted channel 701 can be 'seen' by the cryptographic facility, but by no one outside the cryptographic facility. This ensures that the key distribution channel is not used as a covert privacy channel. Thus, the only way that the application program or the CFAP has of validating the control information transmitted via encrypted channel 701 is the specify reference control information to the CF in clear form. Since it is the application program or the CFAP that specifies the reference control information, the reference control information is consistency checked to determine its accuracy before being passed to the CF. Inside the protected boundary of the CF, the control information recovered from the decrypted keyblk is checked for consistency with the reference control information supplied in clear form by the application to the CFAP and thence by the CFAP to the CF. This permits all parties (CF, CFAP, and application) to be sure that the the control information associated with the to-be-imported key is correct and in accordance with expectations. In summary, all parties look at the reference control information and have a chance to agree or disagree with it, but only the CF sees the control information passed in keyblk and only the CF with highest integrity determines whether the control information received via encrypted channel 701 (i.e., in keyblk) is consistent with the reference control information received in the external key token by the application, or by the CFAP depending on whether key distribution is implemented at the application layer or at the cryptographic facility access program layer. Referring now to Fig. 13, reference control information (designated RCI) received via Clear Channel 702 is inspected by the receiving application program APPL 36. If APPL 36 finds the reference control information to be okay, i.e., it is accurate acceptable, and in accordance with the protocol, in all respects, then APPL 36 will issue a request to CFAP 34 to import the received DEA key, passing the reference control information in the received external key token to CFAP 34. If the CFAP 34 finds the reference control information to be okay, then CFAP 36 will issue an IDK instruction to CF 30 to import the received DEA key, passing the reference control information in the received external key token to CF 30. The control information

(designated CI) received via Encrypted Channel 701 and the reference control information RCI received from the CFAP 34 are inspected by themselves for consistency and then they are compared or consistency checked, one against the other, to determine that CI is consistent with RCI. Only if this consistency checking succeeds, will the CF 30 import the DEA key.

Fig. 14 is a block diagram of the cryptographic facility 30 in the sending location A, as it is organized for performing the generate key set PKA (GKSP) instruction, illustrated in Fig. 9. Fig. 14 shows the cryptographic facility 30 at the sending location which includes the crypto variable retrieval means 40, shown in greater detail in Fig. 16. To prepare a crypto variable such as the key K for transmission from the cryptographic facility 30 to the cryptographic facility 30' at the receiving location, the key K is accessed from the crypto variable retrieval means 40 over the line 62 and applied to the concatenation means 42. In addition, control information such as a control vector and an environmental identification are accessed over line 60 from the crypto variable retrieval means 40 and are applied to the concatenation means 42. Concatenation means 42 will concatenate the key K with the control vector and the environmental identification and that will form the key block 80 which is applied to the public key algorithm encryption means 44. The public key is accessed over line 70 from the crypto variable retrieval means 40 and is applied to the key input of the encryption means 44. The key block 80 is encrypted forming the encrypted key block 85 which is then applied to the transmitting means 46. The encrypted key block 85 is then transmitted over the transmission link 12 to the cryptographic facility 30' at the receiving location shown in Fig. 15. The control information consisting of the control vector and the environmental identification which has been accessed over line 60 is also output as a separate information unit to the transmitting means 46 for transmission over the link 12 to the cryptographic facility 30' at the receiving location. The control information, which can be referred here as the reference control information, is separate from the encrypted key block 85. The reference control information can be transmitted over the same physical communications link 12 as the encrypted key block 85, in a different time slot or frequency slot in the case of frequency division multiplexing. Alternately, completely separate physical communication links can be employed to transmit the reference control information as distinguished from the transmission of the encrypted key block 85. The transmission of the reference control information can be in either clear text form, or alternately the reference control information can be encrypted and transmitted over a privacy channel if the sender and receiver share suitable keys.

In the receiving cryptographic facility 30' shown in Fig. 15, the reference control information is transferred from the communications link 12 to the overline 74 to the control information comparison means 59. The encrypted key block 85 is transferred from the receiving means 56 to the public key algorithm decryption means 54. A privacy key is accessed over line 72 from the crypto variable retrieval means 40' and is applied to the key input of the decryption means 54. The operation of the decryption means 54 generates the recovered key block 80 which is applied to the extraction means 52. The extraction means 52 extracts the control information 60 from the recovered key block 80 and applies the extracted control information to the control information comparison means 59. The control information comparison means 59 then compares the identity of the extracted control information from the key block 80 with the reference control information received from the communications link 12 over line 74. The control information comparison means 59 has an enabling output signal 90 which is produced if the comparison is satisfied. The enabling signal 90 is applied to an enabling input of the crypto variable storage means 50. The crypto variable, in this example the key K, is output from the extraction means 52 on line 62 and applied to the crypto variable storage means 50. The key K will be successfully stored in a crypto variable storage means 50 if the enabling signal 90 is applied from the comparison means 59. In addition, the control information which can include the control vector and the environmental ID of the sending location, can also be stored in the crypto variable storage means 50, if the enabling signal 90 is present.

Further in accordance with the invention, a comparison can be made between the environmental ID of the receiving station B and the environmental ID of the transmitting station A, in order to ensure that the environmental ID for the receiving station B is not identical with the environmental ID contained in the recovered key block 80. This comparison can also be performed in the comparison means 59 and the successful comparison can be made necessary to the generation of the enabling signal 90 as described above. The environmental ID in the reference control information should successfully compare with the environmental ID extracted by the extraction means 52 from the key block 80. In addition, the environmental ID extracted from the key block 80, which represents the environmental ID of the sending location A, should not be the same as the environmental ID of the receiving station B. When these two conditions exist and also when the control vector in the refer-

ence control information successfully compares with the control vector extracted by the extraction means 52 from the key block 80, then the comparison means 59 will output an enabling signal 90 to the storage means 50.

The crypto variable retrieval means 40 and 40' is shown in greater detail in Fig. 16. Input parameters 311 can be transferred over line 33 from the external storage 400 in the CFAP 34. These input parameters can then be applied to the crypto facility 30, over lines 31. Op codes 310 in the CFAP 34 can also be applied over lines 31 to the crypto facility 30. The crypto facility 30 includes the crypto variable retrieval means 40 which contains a random number generator 95, a data encryption algorithm decryption means 410 and an output selection means 420. A master key storage 99 is contained in the crypto facility 30, having an output connected to the key input of the decryption means 410. The random number generator 95 can generate a first type key K' to be applied to the output selection means 420. Alternately, a second type key K" in clear text form can be applied to the output selection means 420. Alternately, a third type key K"' can be applied to the output selection means 420, which is derived from the decryption by the decryption means 410 of an encrypted form of the key K"' which has been encrypted under the exclusive OR product of the master key KM and the control vector C5. The output of the selection means 420 is the key K which is the crypto variable which is discussed in relation to Figs. 14 and 15.

In the preferred embodiment of the invention, public key encryption is used as the encryption technique for transmitting the key block from the sending location to the receiving location, however, it is within the scope of the invention to use symmetric, private key techniques for enciphering and deciphering the key block. Also, in the preferred embodiment of the invention, where digital signatures are employed, as described above, the public key encryption technique for forming digital signatures is employed. However, in an alternate embodiment, conventional Message Authentication Code (MAC) techniques may be employed using a private key algorithm. In the preferred embodiment of the invention, Data Encryption Standard (DES) key is the crypto variable which is transmitted in the key block from the sending location to the receiving location, however, in alternate embodiments of the invention, the crypto variable can be a public key or a non-key-type expression.

Although a specific embodiment of the invention has been disclosed, it will be understood by those having skill in the art that changes can be made to the specific embodiment without departing from the spirit and the scope of the invention.

Claims

1. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, an apparatus for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

   a storage means at a transmitting node in the system for storing a crypto variable which is to be transmitted to a receiving node in the system;

   said storage means storing control information including a control vector to control said crypto variable after it is transmitted from said transmitting node;

   said storage means storing a first key expression;

   concatenating means at said transmitting node, coupled to said storage means, for concatenating said crypto variable with said control information, forming a key block;

   encryption means at said transmitting node, coupled to said storage means and said concatenating means, for encrypting said key block with said first key expression, forming an encrypted key block; and

   transmitting means at said transmitting node coupled to said encryption means and coupled over a communications link to a receiving means at said receiving node, for transmitting said encrypted key block to said receiving node.

2. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, an apparatus for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

   a first storage means at a transmitting node in the system for storing a crypto variable which is to be transmitted to a receiving node in the system;

   a second storage means at said transmitting node for storing control information to control said crypto variable after it is transmitted from said transmitting node said control information

including a control vector to limit the uses of said crypto variable;

a third storage means at said transmitting node for storing a first key expression;

concatenating means at said transmitting node, coupled to said first and second storage means, for concatenating said crypto variable with said control information, forming a key block;

encryption means at said transmitting node, coupled to said third storage means and said concatenating means, for encrypting said key block with said first key expression, forming an encrypted key block;

transmitting means at said transmitting node coupled to said encryption means and coupled over a communications link to a receiving means at said receiving node, for transmitting said encrypted key block to said receiving node;

said transmitting means coupled to said second storage means, for transmitting a second copy of said control information to said receiving node;

fourth storage means at said receiving node, for storing a second key expression corresponding to said first key expression;

decryption means at said receiving node coupled to said receiving means and to said fourth storage means, for decrypting said encrypted key block using said second key expression, to obtain a recovered key block;

extraction means at said receiving node coupled to said decryption means, to extract said control information and said crypto variable from said recovered key block;

comparison means at said receiving node coupled to said extraction means and coupled to said receiving means for comparing said control information extracted from said recovered key block to said second copy of said control information, said comparison means having an enabling output for signalling when said comparison is satisfied;

control means coupled to said extraction means and having an enabling input coupled to said output of said comparison means, for controlling said crypto variable with said control information.

3. Apparatus for generating and distributing a Data Encryption Algorithm (DEA) key in a communications network, comprising:
   a) sending means for generating and producing at least two copies of a key-encrypting key (k-ek), and control information including a control vector for permitted uses of the k-ek;
   b) means included in the sending means for encrypting one copy of the k-ek under the public key of a receiving means and transmitting the public key encrypted k-ek to the receiving means in association with said control information;
   c) means further included in the sending means for encrypting another copy of the k-ek under a master key of the sending means;
   d) means further included in the sending means for storing the master key encrypted k-ek as a common distributing key for other encrypted keys used in the network, in association with said control information;
   e) control means included in the sending means, to limit uses of the k-ek to said permitted uses in response to said control information.

4. The apparatus of claim 1, 2, or 3, wherein:

   said first key expression and said second key expression being symmetric keys, and/or wherein

   said first key expression being a public key issued by said receiving node and said second key expression being a private key corresponding to said public key.

5. The apparatus of anyone of the preceeding claims, wherein said control means further comprises:

   a reference storage means at said receiving node for storing a reference control vector characterizing required uses of said crypto variable at said receiving station;

   a received control vector storage means at said receiving node, coupled to said extraction means, for storing said received control vector extracted from said recovered key block;

   said comparison means at said receiving node coupled to said reference storage means and to said received control vector storage means,

for comparing said reference control vector with said received control vector, and outputting an acceptance signal if the comparison succeeds;

crypto variable storage means at said receive node coupled to said extraction means and to said comparison means, for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means.

6. The apparatus of claim 5, wherein said crypto variable storage means further comprises:

a master key storage means at said receiving node for storing a master key;

an exclusive OR means at said receiving node coupled to said master key storage means and to said received or reference control vector storage means respectively, for forming an exclusive OR product of said master key and said received or reference control vector, respectively, forming a product key expression;

an encryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said crypto variable storage means, for encrypting said crypto variable under said master key, forming an encrypted crypto variable;

an encrypted crypto variable storage means at said receiving node, coupled to said encryption engine, for storing said encrypted crypto variable.

7. The apparatus of claim 6, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said received control vector storage means, for checking said received control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said received control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said received control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable.

8. The apparatus of claim 6 or 7, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said reference control vector storage means, for checking said reference control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said reference control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said reference control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable;

wherein said reference control vector is received preferably from said transmitting node.

9. The apparatus of claim 8, which further comprises:

said received control vector is a first hashed product of said reference control vector, received from said transmitting node;

hashing means in said receiving node coupled to said reference control vector storage means, for forming a second hash product of said reference control vector;

second comparison means coupled to said received control vector storage means and to said hashing means, for comparing said first hashed product with said second hashed product and outputting a second acceptance signal when the comparison is satisfied.

10. The apparatus of anyone of claims 1 to 4, which further comprises:

said control information includes a hashed control vector which represents limitations on uses of said crypto variable.

11. The apparatus of claim 10, wherein said control means further comprises:

a reference control vector storage means at said receiving node for receiving from said transmitting node and storing a reference control vector characterizing required uses of said crypto variable at said receiving station;

a hashed control vector storage means at said receiving node, coupled to said extraction means, for storing said hashed control vector extracted from said recovered key block;

hashing means in said receiving node coupled to said reference control vector storage means, for forming a hash product of said reference control vector;

compare means at said receiving node coupled to said hashing means and to said hashed control vector storage means, for comparing said hash product with said hashed control vector, and outputting an acceptance signal if the comparison succeeds;

crypto variable storage means at said receive node coupled to said extraction means and to said compare means, for storing said crypto variable extracted by said extraction means if said acceptance signal is hashed from said compare means.

12. The apparatus of claim 11, wherein said crypto variable storage means further comprises:

a master key storage means at said receiving node for storing a master key;

an exclusive OR means at said receiving node coupled to said master key storage means and to said hashed control vector storage means, for forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression;

an encryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said crypto variable storage means, for encrypting said crypto variable under said master key, forming an encrypted crypto variable;

an encrypted crypto variable storage means at said receiving node, coupled to said encryption engine, for storing said encrypted crypto variable.

13. The apparatus of claim 12, which further comprises:

a control vector checking means at said receiving node coupled to a user input, for receiving a request from a user for using said crypto variable;

said control vector checking means being coupled to said reference control vector storage means, for checking said reference control vector to determine if said requested uses are permitted;

said control vector checking means outputting an enabling signal if said requested uses are permitted;

a processing means at said receiving node coupled to said control vector checking means, to said hashed control vector storage means and to said master key storage means, for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression;

a decryption engine at said receiving node having a key input coupled to said exclusive OR means for inputting said product key expression, and having an operand input coupled to said encrypted crypto variable storage

means, for decrypting said encrypted crypto variable under said master key, recovering said crypto variable.

14. The apparatus of anyone of the preceeding claims, which further comprises:

said control information includes a transmitting node environment identification which characterizes the identity of said transmitting node.

15. The apparatus of claim 14, wherein said control means further comprises:

a receiving node environment identification storage means at said receiving node for storing a receiving node environment identification;

a received transmission node environment identification storage means at said receiving node, coupled to said extraction means, for storing said transmitting node environment identification extracted from said recovered key block;

compare means at said receiving node coupled to said receiving node environment identification storage means and to said received transmission node environment identification storage means, for comparing said receiving node environment identification and said transmitting node environment identification and outputting an acceptance signal if the comparison fails;

crypto variable storage means at said receive node coupled to said extraction means and to said compare means, for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means.

16. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a method for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

storing a crypto variable which is to be transmitted to a receiving node in the system, at a transmitting node;

storing control information to control said crypto variable after it is transmitted from said transmitting node, at said transmitting node said control information including a control vector to limit the uses of said crypto variable;

storing a first key expression at said transmitting node;

concatenating said crypto variable with said control information, forming a key block, at said transmitting node;

encrypting said key block with said first key expression, forming an encrypted key block, at said transmitting node;

transmitting said encrypted key block to said receiving node;

transmitting a second copy of said control information to said receiving node;

storing a second key expression corresponding to said first key expression, at said receiving node;

decrypting said encrypted key block using said second key expression, to obtain a recovered key block, at said receiving node;

extracting said control information and said crypto variable from said recovered key block, at said receiving node;

comparing said control information extracted from siad recovered key block with said second copy of said control information and generating an enabling signal when the compare is satisfied;

controlling said crypto variable with said control information when said enabling signal has been generated.

17. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a method for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

concatenating a crypto variable with control information including a control vector to control said crypto variable after it is transmitted from said transmitting node, forming a key block, at said transmitting node;

encrypting said key block with a first key expression, forming an encrypted key block, at said transmitting node;

transmitting said encrypted key block to said receiving node;

decrypting said encrypted key block using a second key expression, to obtain a recovered key block, at said receiving node;

extracting said control information and said crypto variable from said recovered key block, at said receiving node;

validating said control information extracted from said recovered key block and generating an enabling signal;

controlling said crypto variable with said control information when said enabling signal has been generated.

18. In a data processing system having a plurality of communicating nodes, at least a pair of nodes in the system exchanging cryptographic communications, a program for execution on the data processing system for enabling a first node of the pair to control a crypto variable after its transmission from the first node to a second node of the pair, comprising:

said program controlling the data processing system for storing a crypto variable which is to be transmitted to a receiving node in the system, at a transmitting node;

said program controlling the data processing system for storing control information to control said crypto variable after it is transmitted from said transmitting node, at said transmitting node said control information including a control vector to limit the uses of said crypto variable;

said program controlling the data processing system for storing a first key expression at said transmitting node;

said program controlling the data processing system for concatenating said crypto variable with said control information, forming a key block, at said transmitting node;

said program controlling the data processing system for encrypting said key block with said first key expression, forming an encrypted key block, at said transmitting node;

said program controlling the data processing system for transmitting said encrypted key block to said receiving node;

said program controlling the data processing system for transmitting a second copy of said control information to said receiving node;

said program controlling the data processing system for storing a second key expression corresponding to said first key expression, at said receiving node;

said program controlling the data processing system for decrypting said encrypted key block using said second key expression, to obtain a recovered key block, at said receiving node;

said program controlling the data processing system for extracting said control information and said crypto variable from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said control information extracted from said recovered key block with said second copy of said control information and generating an enabling signal when the compare is satisfied;

said program controlling the data processing system for controlling said crypto variable with said control information when said enabling signal has been generated.

19. The method of claim 16 or 17, or the program of claim 18, which further comprises:

said first key expression and said second key expression being symmetric keys, and/or

said first key expression being a public key issued by said receiving node and said second key expression being a private key corresponding to said public key.

20. The method of claim 16, 17, or the program of claim 18 or 19, which further comprises:

said control information includes a received control vector which defines limitations on uses of said crypto variable.

21. The program of claim 20, which further comprises:

said program controlling the data processing system for storing a reference control vector characterizing required uses of said crypto

variable at said receiving node;

said program controlling the data processing system for storing said received control vector extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said reference control vector with said received control vector, and outputting an acceptance signal if the comparison succeeds, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means, at said receiving node.

22. The program of claim 21, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said received control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

23. The program of claim 22, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said received control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said received control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node.

24. The program of claim 21, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said reference control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

25. The program of claim 24, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said reference control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR

product of said master key and said reference control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node;

wherein said reference control vector is received preferably from said transmitting node.

26. The program of claim 25, which further comprises:

said received control vector is a first hashed product of said reference control vector, received from said transmitting node;

said program controlling the data processing system for forming a second hash product of said reference control vector, at said receiving node;

said program controlling the data processing system for comparing said first hashed product with said second hashed product and outputting a second acceptance signal when the comparison is satisfied, at said receiving node.

27. The program of claim 20, which further comprises:

said control information includes a hashed control vector which represents limitations on uses of said crypto variable.

28. The program of claim 27, wherein said control means further comprises:

said program controlling the data processing system for receiving from said transmitting node and storing a reference control vector characterizing required uses of said crypto variable at said receiving station, at said receiving node;

said program controlling the data processing system for storing said hashed control vector extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for forming a hash product of said reference control vector, at said receiving node;

said program controlling the data processing system for comparing said hash product with said hashed control vector, and outputting an acceptance signal if the comparison succeeds, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is hashed from said compare means, at said receiving node.

29. The program of claim 28, which further comprises:

said program controlling the data processing system for storing a master key at said receiving node;

said program controlling the data processing system for forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and encrypting said crypto variable under said master key, forming an encrypted crypto variable, at said receiving node;

said program controlling the data processing system for storing said encrypted crypto variable, at said receiving node.

30. The program of claim 29, which further comprises:

said program controlling the data processing system for receiving a request from a user for using said crypto variable, at said receiving node;

said program controlling the data processing system for checking said reference control vector to determine if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for outputting an enabling signal if said requested uses are permitted, at said receiving node;

said program controlling the data processing system for receiving said enabling signal and in response thereto, forming an exclusive OR product of said master key and said hashed control vector, forming a product key expression, at said receiving node;

said program controlling the data processing system for inputting said product key expression, and decrypting said encrypted crypto variable under said master key, recovering said crypto variable, at said receiving node.

31. The method or the program of anyone of the claims 16 to 30, which further comprises:

said control information includes a transmitting node environment identification which characterizes the identity of said transmitting node.

32. The program of claim 31, which further comprises:

said program controlling the data processing system for storing a receiving node environment identification, at said receiving node;

said program controlling the data processing system for storing said transmitting node environment identification extracted from said recovered key block, at said receiving node;

said program controlling the data processing system for comparing said receiving node environment identification and said transmitting node environment identification and outputting an acceptance signal if the comparison fails, at said receiving node;

said program controlling the data processing system for storing said crypto variable extracted by said extraction means if said acceptance signal is received from said compare means, at said receiving node.

FIG. 1

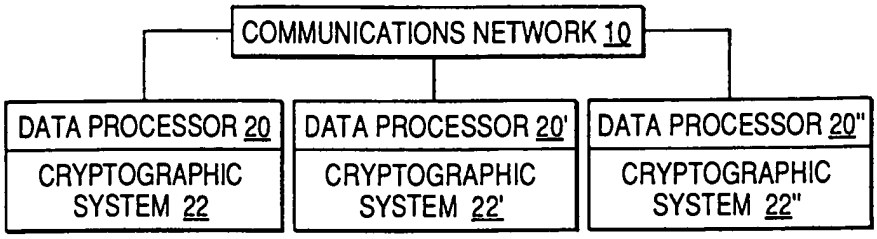


FIG. 2

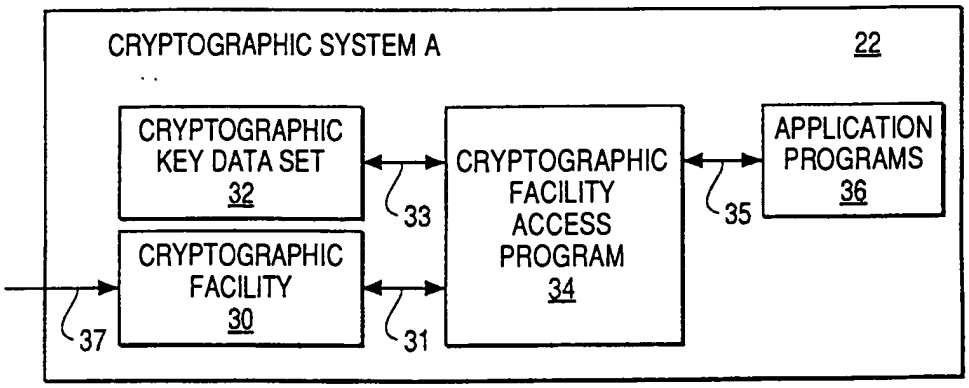


FIG. 3

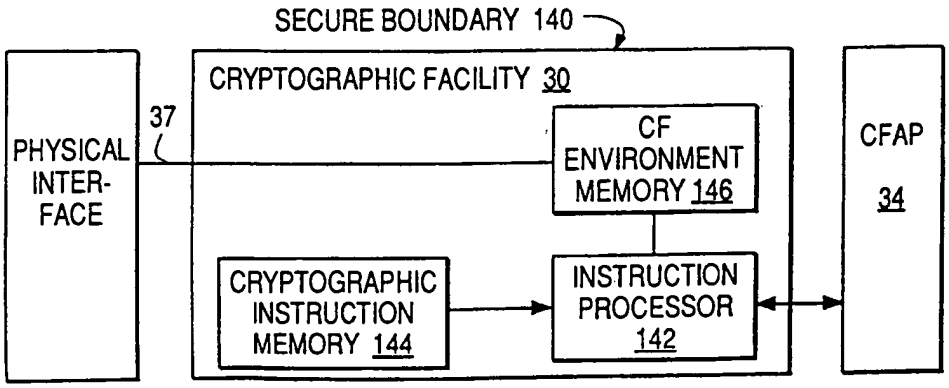


FIG. 4

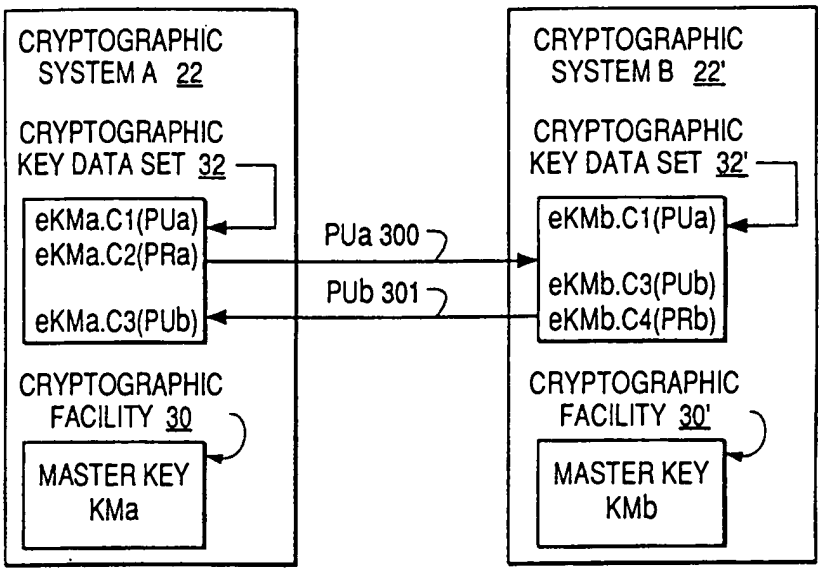


FIG. 5

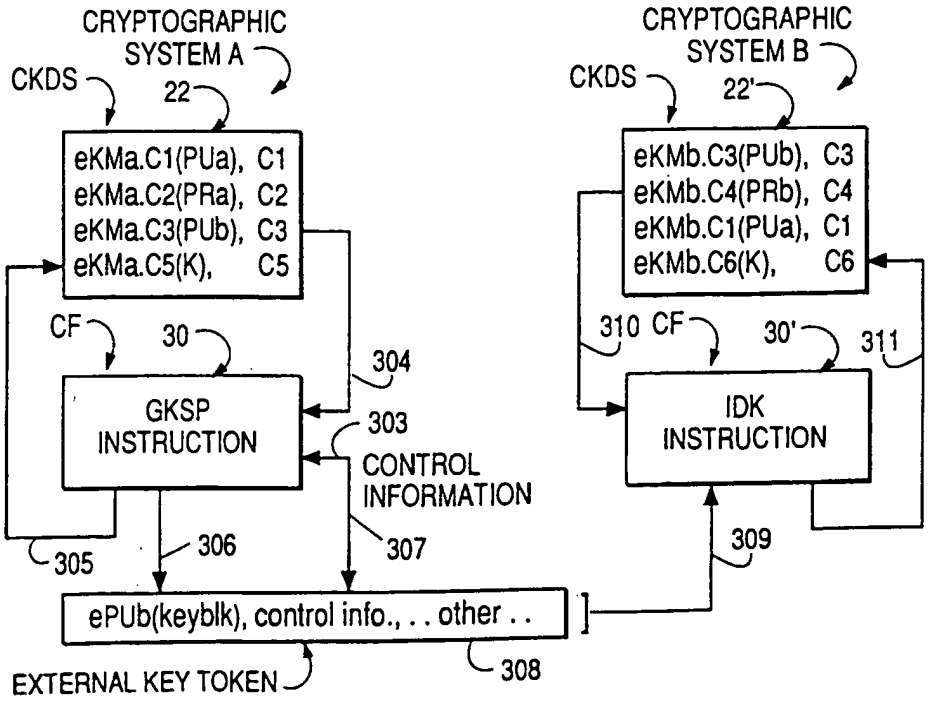


FIG. 6

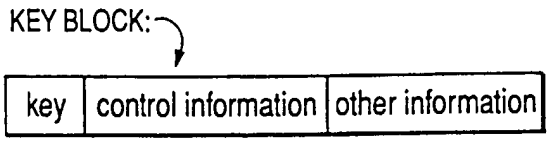


FIG. 7

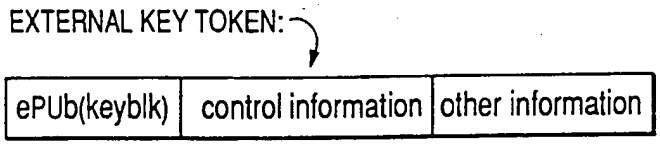


FIG. 8

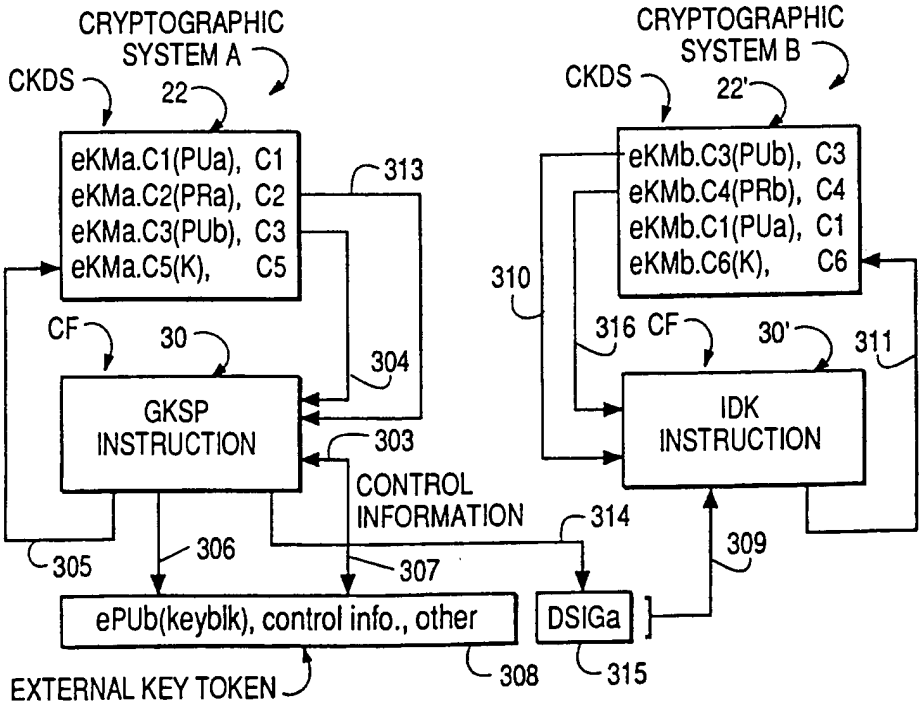


FIG. 9

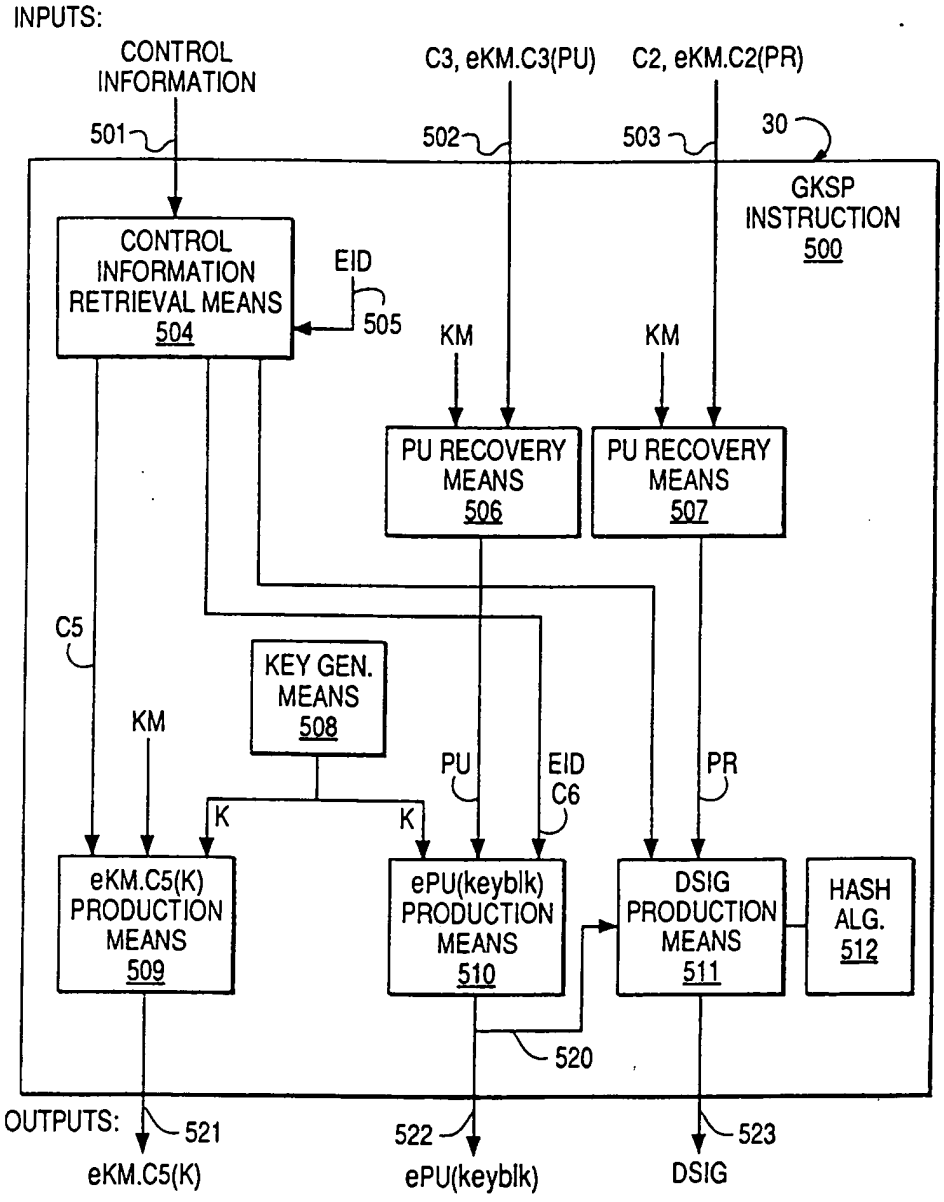


FIG. 10

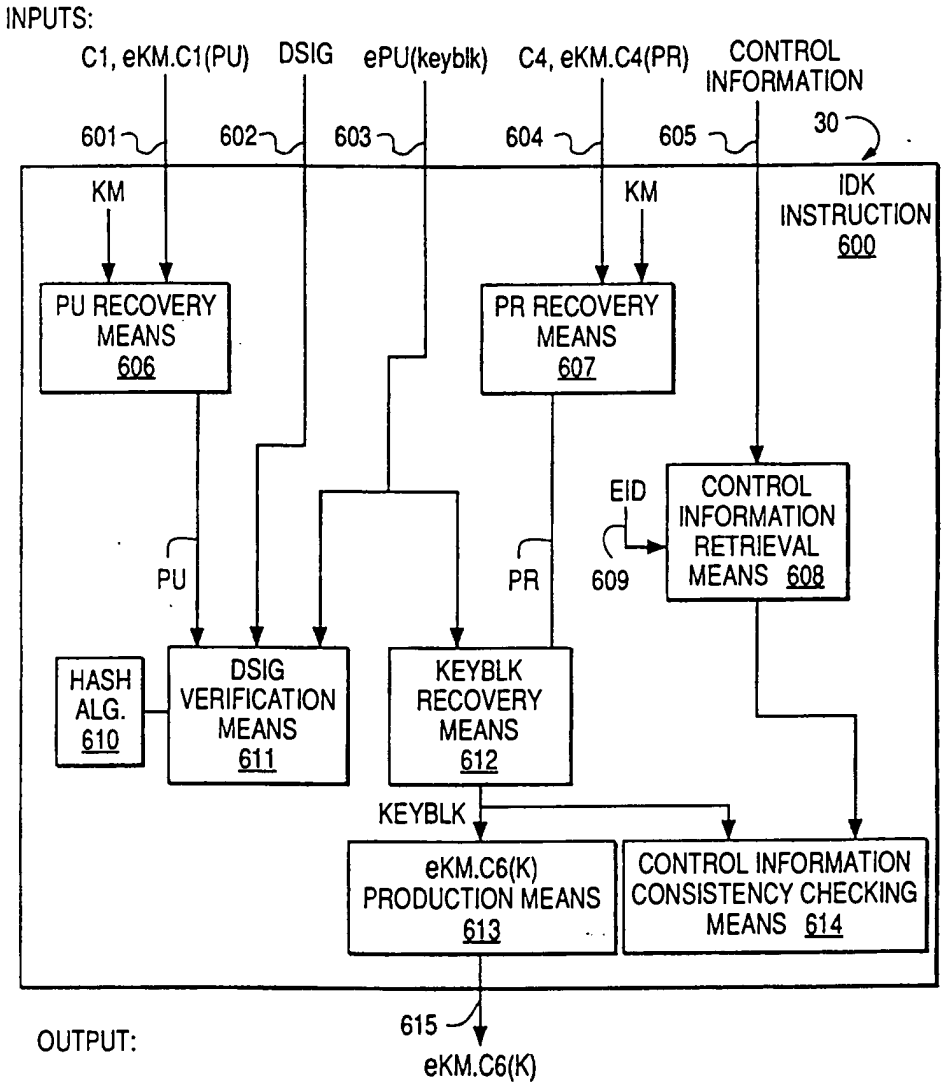


FIG. 11

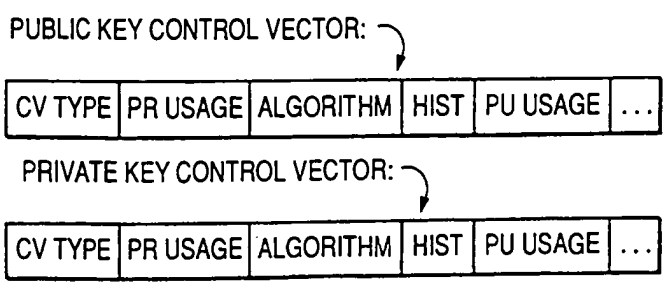


FIG. 12

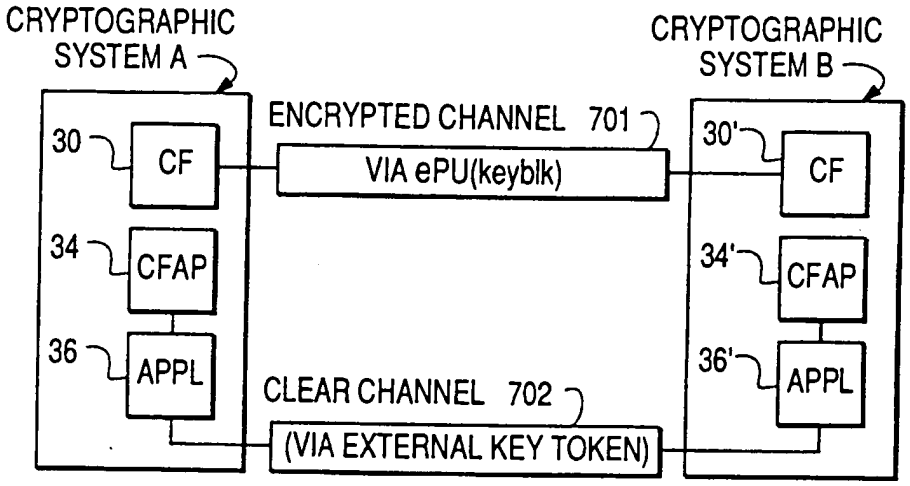


FIG. 13

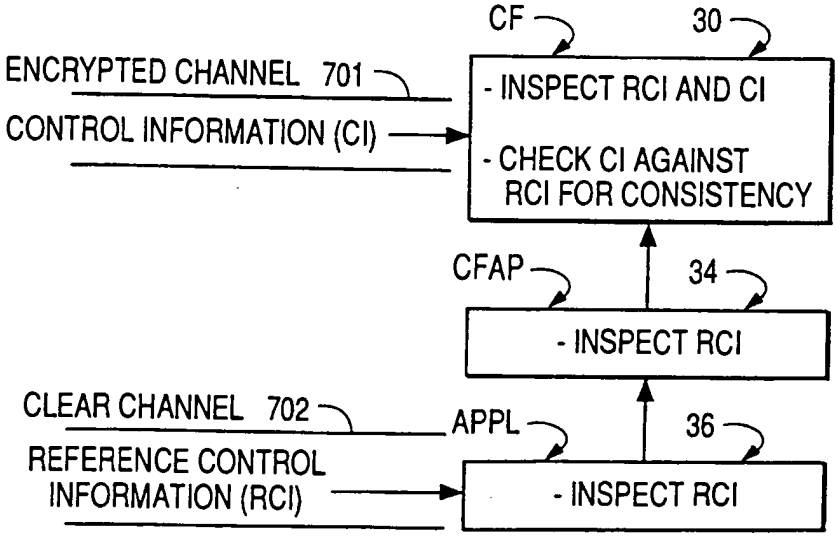


FIG. 14

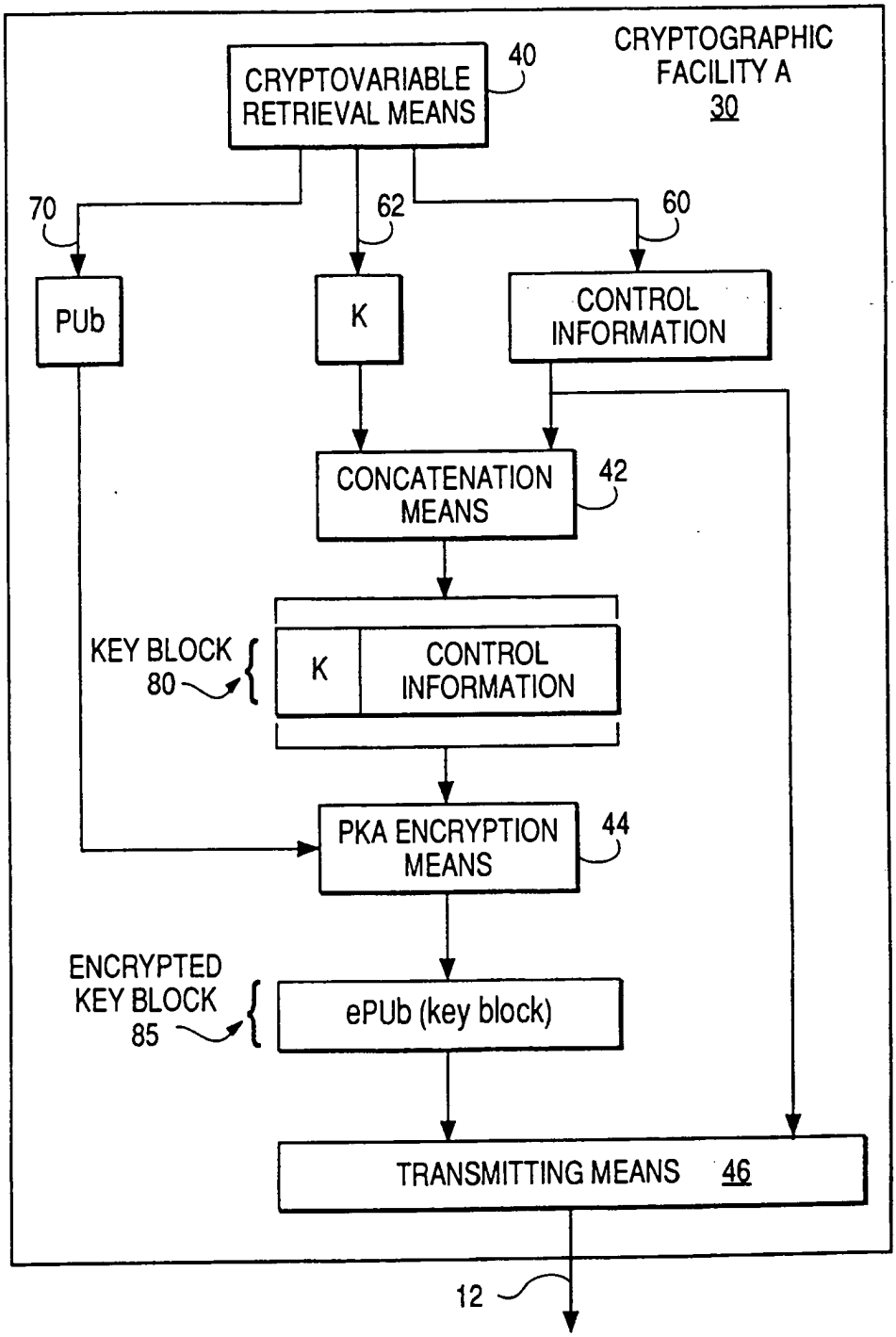


FIG. 15

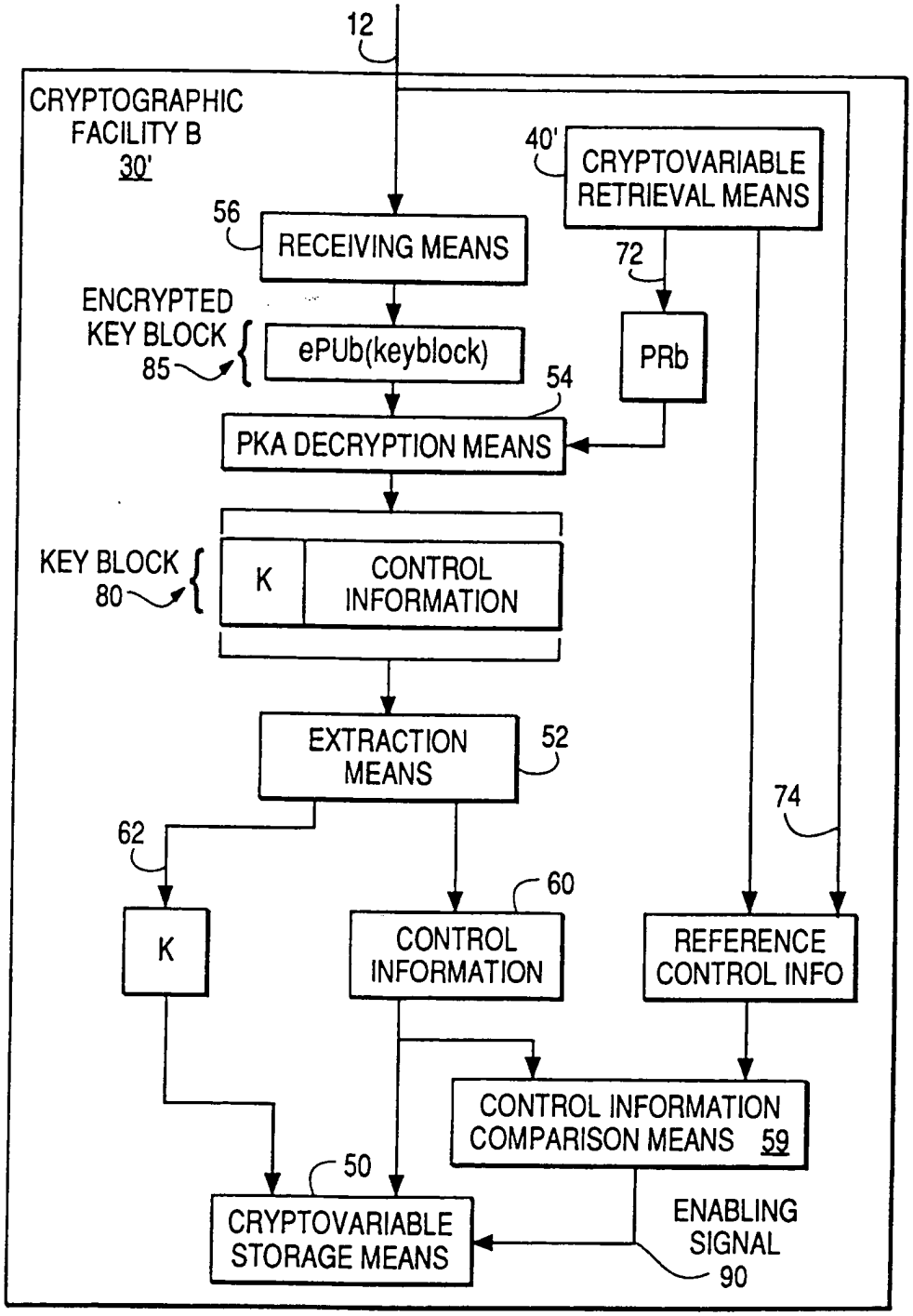
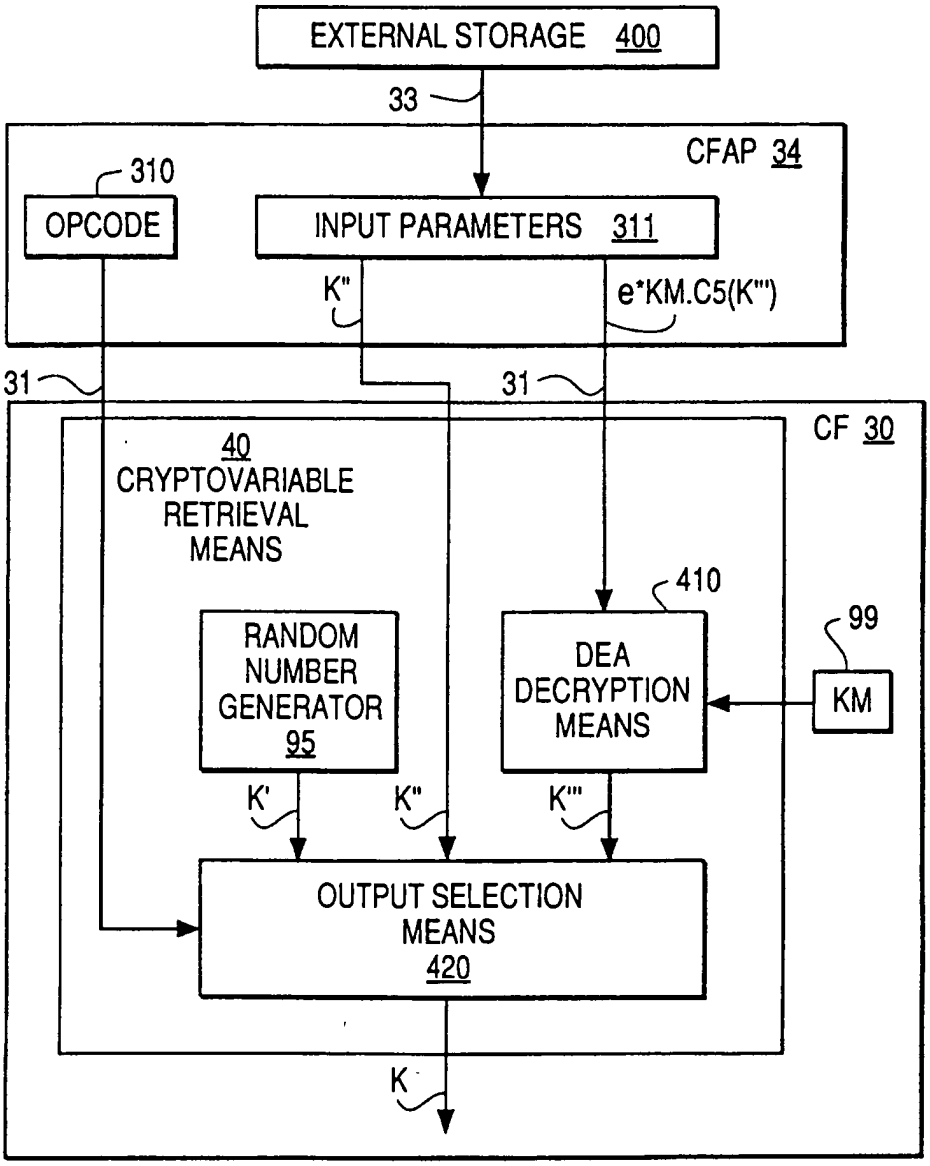


FIG. 16



(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 613 073 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **93306468.5**

(51) Int. Cl.5: **G06F 1/00**

(22) Date of filing: **17.08.93**

(30) Priority: **23.02.93 GB 9303595**

(43) Date of publication of application:
31.08.94 Bulletin 94/35

(84) Designated Contracting States:
DE FR GB SE

(71) Applicant: **INTERNATIONAL COMPUTERS LIMITED**
ICL House
Putney, London, SW15 1SW (GB)

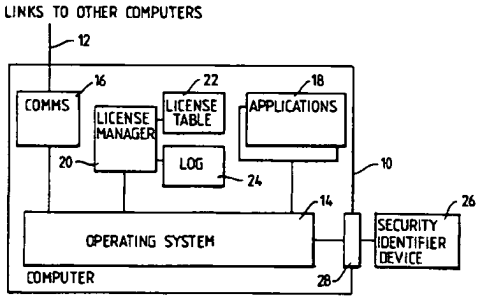
(72) Inventor: **Archer, Barrie**
Lilac Cottage,
Honey Hall
Wokingham, Berkshire RG11 3BA (GB)

(74) Representative: **Guyatt, Derek Charles**
Intellectual Property Department
International Computers Limited et al
Cavendish Road
Stevenage, Herts, SG1 2DY (GB)

(54) Licence management mechanism for a computer system.

(57) A computer system includes a license manager for regulating usage of software items. The license manager checks the host identity of the computer on which it runs and permits usage only if the host identity matches an identity value in a license key. The host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer. Communication of the host identity between the security identifier device and the license manager is protected by encryption.

Fig.1.



EP 0 613 073 A1

Background to the invention

This invention relates to a license management mechanism for a computer system, for controlling use of licensed software.

Software is normally licensed rather than sold in order that restrictions on unauthorised use can be legally enforced. Various schemes have been tried to make the software enforce these restrictions itself, including copy protection, hardware keys, etc., but the current trend is to the use of license keys that are packets of data which permit the software to work only on a particular machine.

One way in which this has been implemented is through the provision of a mechanism referred to as a license manager to which the handling of these license keys is delegated. By centralising the handling of the license keys it is possible to restrict the use of software not just to a single machine but to a network of machines. This provides additional flexibility for the user as well as providing the potential for more sophisticated control over the use of the software within a user organisation.

Central to the use of license managers to control the use of software in this way is the ability to identify which machine the license manager is running. If this were not done it would be possible to obtain license keys for use on one machine and use them on any number of machines. Various schemes have been used to achieve this identification, including serial numbers built into the machine processor, use of Ethernet DTE addresses, etc.

The object of the present invention is to provide a novel way of identifying the machine on which a license manager is running.

Summary of the invention

According to the invention there is provided a computer system including a license manager for regulating usage of software items in accordance with license keys issued to the license manager, the license manager being arranged to check the host identity of the computer on which it runs and to permit usage only if the host identity matches an identity value in the license keys, characterised in that the host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer.

Such identification devices have been used for PC software to permit the software to run only on machines that have the device attached. These devices are usually referred to as dongles. The present invention differs from such known use of dongles in that in the present case the device is used to identify the machine to the license manager, rather than to authorise a particular item of software.

Brief description of the drawings

Figure 1 is a block diagram of a computer system embodying the invention.

Figure 2 is a flow chart showing the operation of a license manager in response to a request to use a feature.

Figure 3 is a flow chart showing a host identity checking function performed by the license manager.

Description of an embodiment of the invention

One embodiment of the invention will now be described by way of example with reference to the accompanying drawing.

Referring to Figure 1, the system comprises a number of computers 10, linked together by means of communications links 12 to form a data processing network.

Each of the computers runs an operating system 14 which controls and coordinates the operation of the computer, and communications software 16 which allows the computer to communicate with the other computers in the system over the links 12. Each computer also runs a number of applications 18 (where an application is any logical software entity).

At least one of the computers runs a program referred to herein as the license manager (LM) 20. The function of the LM is to regulate the applications within a particular domain, so that each application can be used only to the extent permitted by licenses granted to the system owner. The domain comprises those applications that can communicate with the LM. In this example, the domain extends over a multi-computer network, but in other examples it could consist of a single computer.

Each application has a number of features associated with it. A "feature" is defined herein as an aspect of an application that is subject to license control by the LM. A feature may, for example, simply be the invocation of the application by a user. However, more complex features may be defined such as number of users, number of communication links and database size.

Each application also has an application key associated to it, which is unique to the application. As will be described, application keys are used to ensure security of communication between the applications and the LM.

The LM has a private area of memory in which it maintains a license table 22 and a log 24.

The license table holds a number of license keys that have been issued for this system. Each license key contains the following package of information:-

Machine identifier: the identity of the computer on

which the license manager is permitted to run.
Expiry date: the date until which the license key is valid.
Limit: the number of units of a particular feature that are licensed (eg the number of users, number of communication links, or database size).
Application key: the key value of the application to which the license key relates.
Signature: a cryptographic signature which ensures that the license key cannot be changed without detection.

Whenever one of the applications requires to use a feature, it sends a request message to the LM. The request message includes:

- the identity of the feature required
- the number of units of the feature required
- the application key
- a timestamp value.

Referring to Figure 2, when the LM receives this request message, it checks that the timestamp value is current. Assuming the timestamp value is current, the LM then checks whether there is a license key in the license table for the required feature.

If there is a license key in the table, the LM then checks whether the expiry date of the license has passed, and checks the signature of the license key to ensure that it has not been modified. The LM also checks whether the required number of units are available for the feature (ie whether the number of requested units plus the number of units already granted is less than or equal to the limit value in the license key).

If all these checks are satisfactory, the LM returns a "license granted" message to the application, sealed under the application key. The LM keeps a record of the number of units granted for each feature. If, on the other hand, any of the checks fails, the LM returns a "license denied" message to the application. The LM also writes a record in the log 24 to indicate whether a license has been granted or denied.

If the application receives a "license granted" message, it proceeds to use the requested features as required. If, on the other hand, it receives a "license denied" message, it performs one of the following actions, as determined by the designer of the application:

- the application may simply shut itself down.
- in the case where the license was denied because there were not enough units of the requested feature available, the application may display a "call again later" message to the user.
- the application may continue running in a reduced service mode eg a demonstration mode.

When an application terminates, it sends a "license relinquish" message to the LM. The LM will then withdraw any licenses issued to this application, making the units available to other applications.

Each application is required to send a revalidation message periodically to the LM, to re-validate its license. For example, a revalidation message may be required every 5 minutes. If the application does not receive any response to this message, it assumes that it has lost contact with the LM, and shuts down or continues in a reduced service mode.

The LM periodically checks whether it has received revalidation messages from all the application to which it has granted licenses. If a revalidation message has not been received from an application, the LM assumes that the application has failed, and therefore withdraws the license, making the units available to other applications.

In order to ensure that unauthorised copies of the LM cannot be run on other systems, it is necessary to provide a way of identifying the machine on which the LM runs. This is achieved by means of a security identification device (SID) 26, which stores an identifier unique to this device, referred to as the secure host identifier. The SID is attached to the computer 10 by way of an external port 28. In this example, the port is a standard parallel printer port, and the SID is designed so that a printer may be plugged into the back of the SID, so that both the printer and SID share the same port. Messages for the SID are identified by special commands.

In other embodiments of the invention, the SID may be attached to a special dedicated port, or to some other type of standard port. The port may be serial rather than parallel.

Referring to Figure 3, in order to check the host identity, the LM sends a request message to the SID at regular intervals, requesting it to supply the secure host identifier.

The SID responds to this by returning a message encrypted under a key known only to the SID and the LM.

The message contains:

- the secure host identifier
- a sequence number, which is incremented each time the SID returns a message.

When the LM receives this message, it decrypts it, and checks the sequence number to ensure that it is the next expected sequential value. This ensures that it is not possible to replace the SID by a program which intercepts the requests from the LM and returns a copy of the SID's response, or which passes the request to a SID on another system.

The LM then checks whether the returned secure host identifier matches the machine identifiers of the license keys held in the license table 22.

If the LM does not receive any response to a request to the SID, or if the response does not contain the correct sequence number, or if the secure host identifier does not match the machine identifiers in the license keys, the LM closes down. This means that the LM will not issue any more licenses to applications. Also, because the LM will not now respond to the revalidation message from the application, any outstanding licenses are effectively cancelled.

In summary, it can be seen that the LM will issue licenses, permitting applications to operate, only if a security identification device SID is connected to the computer, and if the machine identifiers in the individual license keys issued to the LM match the secure host identifier held in the SID.

It should be noted that the LM can grant licenses to applications running in any of the computers 10 in the network, not just to applications running in the same computer as the LM. The number of licenses that may be granted is restricted by the limit in the license keys. Thus, for example, if a license key sets a limit on the number of users, then the total number of users of a particular application in the network cannot exceed this limit.

The use of the device for the provision of the identifier to the license manager has several very important advantages:

- if the machine to which the device is attached fails, the device can be transferred to another machine (new keys are not required)
- the supplier of the device can retain title to the device, so in the event of the machine being sold the device has to be returned to the supplier. Hence all software on the machine that would only work with a license manager will no longer function as required by the terms of supply of the software which is licensed to a legal entity not to a machine.
- if the user of the software wishes to change the license he has to reduce its capability, the device can be replaced and new keys issued. Current schemes do not provide for the secure revocation of the keys.
- the device can be used to provide secure identification on standard hardware platforms which do not inherently provide such a facility, and hence can enable the use of license management on such hardware.

It should be noted that although the embodiment of the invention described above is a multi-computer system, the invention is equally applicable to single processor systems, or to multi-nodal systems, comprising a plurality of multi-processor nodes.

Claims

1. A computer system including a license manager for regulating usage of software items in accordance with license keys issued to the license manager, the license manager being arranged to check the host identity of the computer on which it runs and to permit usage only if the host identity matches an identity value in the license keys, characterised in that the host identity of the computer is supplied by a security identification device removably coupled to an external port on the computer.

2. A system according to Claim 1 wherein communication of the host identity between the security identifier device and the license manager is protected by encryption.

3. A system according to Claim 2 wherein each host identity returned by the security identifier device is encrypted together with a sequence number which is incremented each time the host identity is returned.

4. A system according to any preceding claim wherein the license manager regulates the usage of software items within a domain comprising software items that can communicate with the license manager.

5. A system according to Claim 4 wherein said domain is distributed over a network of computers.

Fig.1.

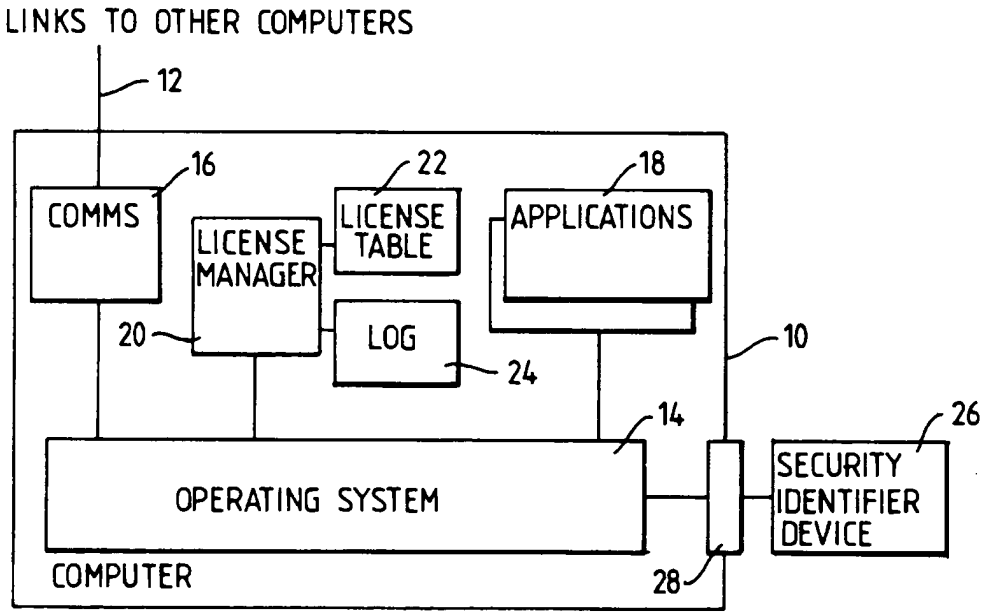


Fig. 2.

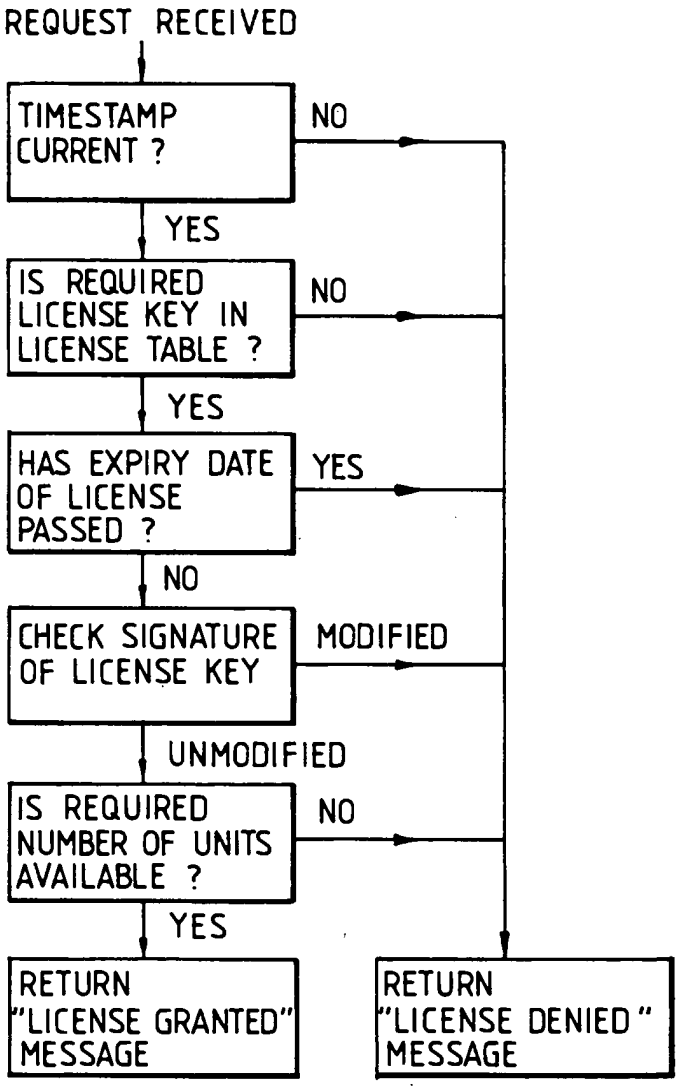
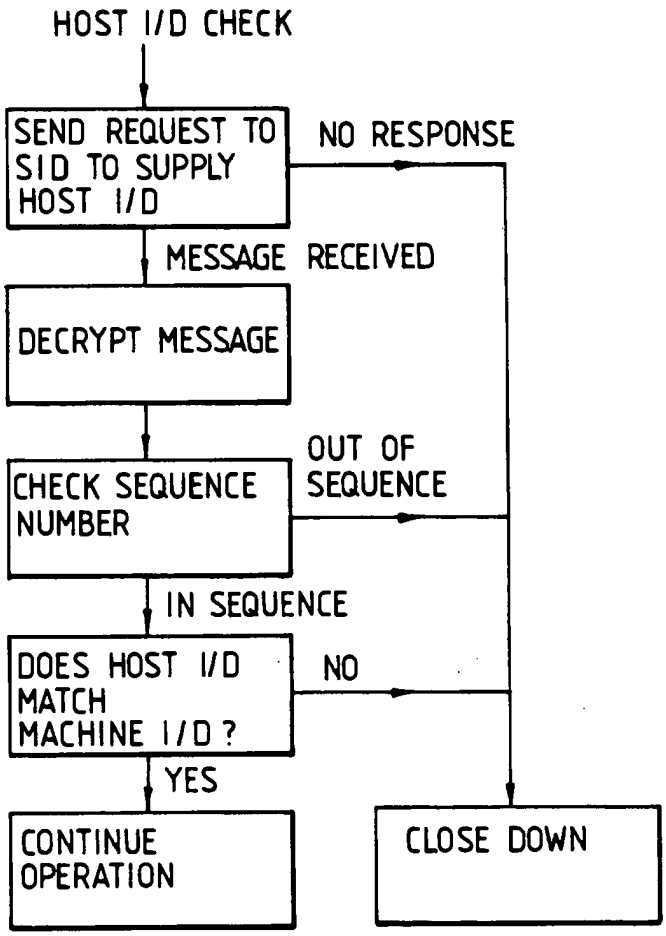


Fig.3.



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 93 30 6468

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
Y	US-A-4 924 378 (HERSHEY ET AL) * abstract; figures 1,3,5,7 * * column 1, paragraph 2 * * column 2, line 1 - column 3, line 36 * * column 7, line 22 - column 8, line 12 * * column 10, line 27 - line 40 * * claims 1-5,11-23 * ---	1-5	G06F1/00
Y	PTR PHILIPS TELECOMMUNICATION AND DATA SYSTEMS REVIEW, vol. 47, no. 3 , September 1989 , HILVERSUM, NL; pages 1 - 19 R.C.FERREIRA 'The Smart Card: A High Security Tool in EDP' * summary; figures 4,5 * * page 5, line 6 - page 7, line 5 * * page 9, line 1 - page 11, line 40 * * page 12, line 36 - page 13, line 4 * ---	1-5	
A	EP-A-0 191 162 (IBM) * abstract; figures 4,9 * * column 6, line 8 - column 7, line 14 * * column 9, line 6 - line 39 * * column 10, line 5 - line 40 * * column 13, line 5 - line 36 * -----	1,3	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	4 May 1994	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 678 836 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **94105573.3**

(51) Int. Cl.[6]: **G07F 7/10**

(22) Date of filing: **11.04.94**

(43) Date of publication of application:
25.10.95 Bulletin 95/43

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **TANDEM COMPUTERS INCORPORATED**
10435 North Tantau Avenue,
Loc. 200-16
Cupertino,
California 95014-0709 (US)

(72) Inventor: **Atalla, Martin M.**
18 Monte Vista
Atherton
CA 94025 (US)
Inventor: **Hopkins, W. Dale**
2425 Rio Drive
Gilroy
CA 95020 (US)

(74) Representative: **KUHNEN, WACKER & PARTNER**
Alois-Steinecker-Strasse 22
D-85354 Freising (DE)

(54) Method and means for combining and managing personal verification and message authentication encryptions for network transmission.

(57) The method and means of transmitting a user's transaction message to a destination node in a computer-secured network operates on the message, and a sequence number that is unique to the transaction message to form a message authentication code in combination with the user's personal identification number. The message authentication code is encrypted with a generated random number and a single session encryption key which also encrypts the user's personal identification number. An intermediate node may receive the encryptions to reproduce the personal identification number that is then used to encrypt the received message and sequence number to produce the random number and a message authentication code for comparison with a decrypted message authentication code. Upon favorable comparison, the random number and the message authentication code are encrypted with a second session encryption key to produce an output code that is transmitted to the destination node along with an encrypted personal identification number. There, the received encryptions are decrypted using the second session key to provide the personal identification number for use in encrypting the message and sequence number to produce a message authentication code for comparison with a decrypted message authentication code. Upon favorable comparison, the transaction is completed and a selected portion of the decrypted random number is returned to the originating node for comparison with the corresponding portion of the random number that was generated there. Upon unfavorable comparison at the destination node or at an intermediate node, a different portion of the decrypted random number is returned to the originating node for comparison with the corresponding portion of the random number that was generated there. The comparisons at the originating node provide an unambiguous indication of the completion or non-completion of the transaction at the destination node.

EP 0 678 836 A1

Related Cases

The subject matter of this application is related to the subject matter disclosed in U.S. Patents 4,268,715; 4,281,215; 4,283,599; 4,288,659; 4,315,101; 4,357,529; 4,536,647 and pending application for U.S. Patent Serial No. 547,207, entitled POCKET TERMING, METHOD AND SYSTEM FOR SECURED BANKING TRANSACTIONS, filed October 31, 1983 by M.M. Atalla.

Background of the Invention

Conventional data encryption networks commonly encrypt a Personal Identification Number with a particular encryption key for transmission along with data messages, sequence numbers, and the like, from one location node in the data network to the next location or node in the network. There, the encrypted PIN is decrypted using the encryption key, and re-encrypted with another encryption key for transmission to the next node in the network, and so on to the final node destination in the network.

In addition, such conventional data encryption networks also develop a Message Authentication Codes in various ways, and then encrypt such MAC for transmission to the next node using a MAC-encryption key that is different from the encryption key used to encrypt the PIN. At such next node, the MAC is decrypted using the MAC encryption key and then re-encrypted using a new MAC-encryption key for transmission to the next node, and so on to the final destination node in the network.

Further, such conventional networks operate upon the PIN, MAC, data message, sequence number, and the like; received and decrypted at the final destination node to consummate a transaction, or not, and then communicate an ACKnowledgment or Non-ACKnowledgment message back to the originating node of the network. Such ACK or NACK codes may be encrypted and decrypted in the course of transmission node by node through the network back to the originating node to provide an indication there of the status of the intended transaction at the final destination node.

Conventional data encryption networks of this type are impeded from handling greater volumes of messages from end to end by the requirement for separately encrypting and decrypting the PIN and MAC codes at each node using different encryption/decryption keys for each, and by the requirement for encrypting/decrypting at least the ACK code at each node along the return path in the network.

In addition, such conventional data encryption networks are susceptible to unauthorized intrusion and compromise of the security and message authenticity from node to node because of the separated PIN and MAC encryption/decryption techniques involved. For example, the encrypted PIN is vulnerable to being "stripped" away from the associated MAC, message, sequence number, and the like, and to being appended to a different MAC, message, sequence number, and the like, for faithful transmission over the network. Further, the return acknowledgment code may be intercepted and readily converted to a non-acknowledgment code or simply be altered in transmission after the transaction was completed at the destination node. Such a return code condition could, for example, cause the user to suffer the debiting of his account and, at the same time, the denial of completion of a credit purchase at point-of-sale terminal or other originating node.

Summary of the Invention

Accordingly, the method and means for integrating the encryption keys associated with the PIN and MAC codes according to the present invention assure that these codes are sufficiently interrelated and that alteration of one such code will adversely affect the other such code and inhibit message authentication in the network. In addition, the return acknowledgment or non-acknowledgment code may be securely returned from node to node in the network without the need for encryption and decryption at each node, and will still be securely available for proper validation as received at the originating node. This is accomplished according to the present invention by using one session key to encrypt the PIN along with the MAC, a random number, the message, and the sequence number which are also encrypted with the PIN such that re-encryption thereof in the transmission from location to location, or node to node over a network is greatly facilitated and validatable at each node, if desired. In addition, portions of the random number are selected for use as the Acknowledgment or Non-Acknowledgment return codes which can be securely returned and which can then only be used once to unambiguously validate the returned code only at the originating node in the network.

Description of the Drawings

Figure 1 is graphic representation of a typical conventional encryption scheme which operates with two independent session keys;

Figure 2 is a schematic representation of a second network according to the present inventions; and

Figure 3 is a graphic representation of the signal processing involved in the operation of the net-

work of Figure 2.

Description of the Preferred Embodiment

Referring now to Figure 1, there is shown a graphic representation of the encoding scheme commonly used to produce the PIN and MAC codes using two session keys for transmission separately to the next network node. As illustrated, one session key 5 may be used to encrypt the PIN entered 7 by a user (plus a block of filler bits such as the account number, as desired) in a conventional encryption module 9 which may operate according to the Data Encryption Standard (DES) established by the American National Standards Institute (ANSI) to produce the encrypted PIN signal 11 (commonly referred to as the PIN block" according to ANSI standard 9.3) for transmission to the next network node. In addition, the message or transaction data which is entered 13 by the user and which is to be transmitted to another node, is combined with a sequence number 15 that may comprise the date, time, station code, and the like, for encryption by a DES encryption module 17 with another session key 19 to produce a Message Authentication Code (MAC) 21 for that message and sequence number. The MAC may comprise only a selected number of significant bits of the encrypted code. The message and MAC are separately transmitted to the next node along with the encrypted PIN, and these codes are separately decrypted with the respective session keys and then re-encrypted with new separate session keys for transmission to the next network node, and so on, to the destination node. Conventional PIN validation at the destination node, and message authentication procedures may be performed on the received, encrypted PIN and MAC, (not illustrated) and the message is then acted upon to complete a transaction if the PIN is valid and the MAC is unaltered. A return ACKnowledgment (or Non-ACKnowledgment) code may be encrypted and returned to the next node in the network over the return path to the originating node. At each node in the return path, the ACK code is commonly decrypted and re-encrypted for transmission to the next node in the return path, and so on (not illustrated), to the originating node where receipt of the ACK is an indication that the transaction was completed at the destination node. Conventional systems with operating characteristics similar to those described above are more fully described, for example, in U.S. Patent 4,283,599.

One disadvantage associated with such conventional systems is the need to encrypt and decrypt at each node using two separate session keys. Another disadvantage is that such conventional systems are vulnerable to unauthorized manipulation at a network node by which the message and MAC may be "stripped away" from the encrypted PIN associated with such message and replaced with a new message and MAC for transmission with the same encrypted PIN to the next network node. Further, the acknowledgement code that is to be returned to the originating node not only must be decrypted and re-encrypted at each node along the return path, but the return of an acknowledgment code that is altered along the return path may connote non-acknowledgment or non-completion of the intended transaction at the destination node. This condition can result In the account of the user being debited (the PIN and MAC were valid and authentic as received at the destination node), but the user being denied completion of a credit transaction (e.g., transfer of goods) at the originating node.

Referring now to Figures 2 and 3, there are shown schematic and graphic representations, respectively, of network operations according to the present invention. Specifically, there is shown a system for transmitting a message over a network 29 from an originating node 31 to a destination node 33 via an intermediate node 35. At the originating node 31, an authorized user enters his PIN 37 of arbitrary bit length with the aid of a key board, or card reader, or the like, and the entered PIN is then filled or blocked 39 with additional data bits (such as the user's account number in accordance with ANSI standard 9.3) to configure a PIN of standard bit length.

In addition, the transaction data or message 41 entered through a keyboard, or the like, by the user is combined with a sequence number 43 which is generated to include date, time of day, and the like. The combined message and sequence number is encrypted 45 with the PIN (or blocked PIN) in a conventional DES module to produce a multi-bit encrypted output having selected fields of bits, one field of which 51 serves as the Message Authentication Code (MAC). Other schemes may also be used to produce a MAC, provided the PIN (or blocked PIN) is used as the encryption key, and the resulting MAC, typically of 64-bit length, may be segregated into several sectors or fields 51. A random number (R/N) is generated 52 by conventional means and is segregated into several sectors or fields 54, 56, 58. The first sector or field 54 of, say 32-bits length, is then encrypted with the selected MAC field 53 in a conventional DES encryption module 55 (or in DES module 45 in time share operation) using the session key K_1 as the encryption key 50. In addition, the PIN (or blocked PIN) 39 is encrypted in DES encryption module 60 (or in DES module 45 in time share operation) using the session key K1 as the encryption Key 50. The session key 50 may be transmitted to successive

nodes 35, 33 in secured manner, for example, as disclosed in U.S. Patent 4,288,659. The resulting encrypted output codes 62, 64 are then transmitted along with sequence number 43 and the message 41 (in clear or cypher text) over the network 29 to the next node 35 in the path toward the destination node 33. Thus, only a single session key K_1 is used to encrypt the requisite data for transmission over the network, and the residual sectors or fields 56, 58 of the random number from generator 52 remain available to verify successful completion of the transaction at the destination node 33, as later described herein.

At the intermediate node 35, the encrypted PIN 64 received from the originating node 31 is decrypted in conventional DES module 70 using the session key K_1 to produce the blocked PIN 63. In addition, the encrypted MAC and R/N 68 received from the originating node is decrypted in conventional DES module 61 (or in DES module 70 operating in timeshare relationship) using session key K_1 to produce the MAC and the R/N in segregated fields. An initial validation may be performed by encrypting the received message 41 and sequence number 43 in conventional DES module 67 using the decrypted PIN 63 as the encryption key. Of course, the original PIN as entered by the user may be extracted from the decrypted, blocked PIN 63 to use as the encryption key in module 67 if the corresponding scheme was used in node 31. (It should be understood that the PIN or blocked PIN does not appear in clear text outside of such decryption or encryption modules 70, 67 (or 69, later described herein), and that these modules may be the same DES module operated in time-shared relationship.)

The encrypted output of module 67 includes several sectors, or fields, similar to those previously described in connection with the encrypted output of module 45. The selected sector 53 of significant bits that constitutes the MAC is selected for comparison with the MAC 65 that is decrypted in DES module 61. This decryption also provides the R/N having several selected sectors or fields 72. If the comparison of the decrypted and encrypted MAC's in comparator 74 is favorable, gate 76 is enabled and the decrypted MAC and R/N are encrypted in conventional DES module 69 using new session key K_2 as the encryption key, and gate 88 is enabled to encrypt the decrypted PIN in DES module 78 (or in DES module 67 or 69 in time share operating). If comparison is unfavorable, the transaction may be aborted and the gate 80 is enabled to transmit back to the originating node 31 the sector or field 58 of the R/N which constitutes the Non ACKnowledge sector of the decrypted R/N output of module 61. The encrypted PIN output 82 of module 78 and the encrypted MAC and R/N output 84 of the module 69 are thus transmitted along with the message 41 and sequence number 43 over the network 29 to the destination node 35 upon favorable comparison 74 of the encrypted and decrypted MACs.

At the destination node 33, the encrypted PIN output 86 received from the intermediate node 35 is decrypted in conventional DES module 71 using the session key K_2 to produce the PIN 73. An initial validation may be performed by encrypting the received message 41 and sequence number 43 in conventional DES module 77, using the decrypted PIN 73 as the encryption key. As was described in connection with the intermediate node 35, the original PIN as entered by the user may be extracted from the decrypted, blocked PIN 73 to use as the encryption Key in module 77 if the corresponding scheme was used in node 31. And, it should be understood that the PIN or blocked PIN does not appear in clear text outside of the decryption or encryption modules 71, 77, which modules may be the same DES module operated in time-shared relationship. In addition, the encrypted MAC and R/N received at the destination node 33 is decrypted in DES module 92 using the session key K_2 to produce the MAC 75 and the R/N 94 in segregated sectors or fields. The selected sector 53 of significant bits that constitutes the MAC in the encrypted output of module 77 is compared 79 for parity with the decrypted MAC 75. If comparison is favorable, the transaction may be completed in response to the message 41, and gate 81 may be enabled to transmit 29 back to the intermediate node 35 a second selected sector or field 56 which constitutes the ACKnowledge output sector of the R/N decrypted output from module 92. If comparison 79 is unfavorable, the transaction is not completed and gate 83 is enabled to transmit 29 back to the intermediate node 35 a third selected sector or field 58 which constitutes the Non-ACKnowledge sector of the R/N decrypted output from module 92.

In accordance with one aspect of the present invention, the returned ACK or NACK codes do not require decryption and re-encryption when transmitted from node to node along the return path in the network back to the originating node 31. Instead, these codes are already in encoded form and may be transmitted directly from node to node without encumbering a node with additional operational overhead. These codes are therefore secured in transmission over the network and are only cypherable in the originating node 31 which contains the ACK and NACK fields or sectors 56 and 58 of the random number from generator 52. At the originating node 31, the second and third sectors or fields 56 and 58 of the random number are compared 98 with the corresponding sectors of

decrypted R/N outputs received from the destination node 33 (or the sector 58 of the decrypted R/N output received from intermediate node 35) to provide an indication at the originating node that the transaction was either completed 89 or aborted 91. Of course, the ACK and NACK may be encrypted as a network option when returned to the originating node 31. And, it should be understood that the encryption and decryption modules at each node may be the same conventional DES module operated in timeshare relationship.

Therefore, the system and method of combining the management of PIN and MAC codes and the session keys associated therewith from node to node along a data communication network obviates the conventional need for separate session keys for the PIN and the MAC, and also obviates the need for conventional encryption/decryption schemes for an acknowledgment code at each node along the return path back to the originating node. If desired, PIN validations may be performed at each node since the PIN is available within the DES module circuitry. In addition, the present system and method also reduces the vulnerability of a secured transmission system to unauthorized separation of a valid PIN code from its associated message and MAC code for unauthorized attachment to a different message and MAC code. Further, the method and means of the present invention reduces the ambiguity associated with the return or not of only an acknowledgment code in conventional systems by returning either one of the ACK and NACK codes without additional operational overhead at each node.

Claims

1. The method of securing transaction data between two locations in response to a user's message ad personal identification number, the method comprising:

forming a sequence number representative of the user's transaction;

encoding in a first logical combination at the first location the user's message and the sequence number in accordance with the personal identification number received from the user to produce a message authentication code having a plural number of digit sectors;

generating a random number;

establishing a first encoding key;

encoding in a second logical combination at the first location the random number and a selected number of sectors of the message authentication code in accordance with the first encryption key to produce a first coded output;

encoding in a third logical combination at the first location the user's personal identification number in accordance with the first encoding key to produce a second coded output;

transmitting to another location the user's message and the sequence number and the first ad second coded outputs;

establishing the first encoding key at such other location;

decoding the first coded output received at such other location with the first encoding key according to said second logical combination thereof to provide the random number and message authentication code;

decoding the second coded output received at such other location with the first encoding key according to said third logical combination to provide the user's personal identification number;

encoding in the first logical combination at such other location the user's message and sequence number received thereat in accordance with the decoded personal identification number to produce a message authentication code having a plural number of digit sectors; and

comparing selected corresponding digit sectors of the decoded message authentication code and the encoded message authentication code to provide an indication upon favorable comparison of the valid transmission of the user's message between the two locations.

2. The method according to claim 1 comprising the steps of:

establishing a second encoding key at the other location;

encoding in a fourth logical combination at such other location the decoded random number and selected sector of the message authentication code in accordance with the second encoding key to produce a third coded output;

encoding in a fifth logical combination at the other location the decoded user's personal identification number in accordance with the second encoding key to produce a fourth coded output;

transmitting to a remote location the user's message and the sequence number and the third and fourth coded outputs;

establishing the second encoding key at the remote location;

decoding the third coded output as received at the remote location according to the fourth logical combination in accordance with the second encoding key to provide the random number and the message authentication code having a plural number of digit sectors;

decoding the fourth coded output received

at the remote location according to the fifth logical combination to provide the user's personal identification number;

encoding the message and the sequence number received at the remote location according to the first logical combination in accordance with the decoded personal identification number to produce a message authentication code having a plural number of digit sectors; and

comparing corresponding digit sectors of the decoded message authentication code and the encoded message authentication code at the remote location to provide an indication upon favorable comparison of the unaltered transmission of the message, or an indication upon unfavorable comparison of an alteration in the transmission of the message.

3. The method according to claim 1 comprising the steps of:

transmitting a selected sector of the decoded random number from the other location to the one location in response to unfavorable comparison; and

comparing the selected sector of the random number received at the one location from the other location with the corresponding selected sector at the one location to provide an indication of the altered transmission of the message to the other location.

4. The method according to claim 2 comprising the steps of:

completing the transaction and returning a second selected sector of the decoded random number from the remote location to the one location in response to said favorable comparison, and inhibiting completion of the transaction and returning a third selected sector of the decoded random number from the remote location to the one location in response to said unfavorable comparison; and

comparing the selected sector of the random number received at the one location from the remote location with the corresponding selected sector of the number generated at the one location to provide an indication of the completion or non-completion of the transaction at the remote location.

5. Apparatus for securing transaction data between two locations in response to a user's message and personal identification number, the apparatus comprising:

means for generating a sequence number associated with a user's transaction;

means for generating a random number;

first encryption means at one location for encrypting according to a first logical combination of the user's message and the sequence number applied thereto with the personal identification number received from the user for producing a message authentication code therefrom having a plural number of digit sectors;

means at said one location for producing a first session key;

second encryption means coupled to receive the random number from the user and a selected sector of the message identification code for encrypting the same with the first session key according to a second logical combination thereof to produce a first encoded output;

third encryption means coupled to receive the personal identification number from the user for encrypting the same with the first session key according to a third logical combination thereof to produce a second encoded output;

means for transmitting the first and second encoded outputs and message and sequence number from the one location to the next location;

means at the next location for producing the first session key;

first decryption means at the next location coupled to receive the transmitted first encoded output and the first session key for decrypting in accordance with said second logical combination to provide the random number and the message authentication code;

second decryption means at the next location coupled to receive the transmitted second encoded output and the first session key for decrypting in accordance with the third logical combination thereof to produce the user's personal identification number;

third encrpytion means at the next location coupled to receive the transmitted message and sequence number for encoding the same according to said first logical combination with the decrypted personal identification number to produce a message authentication code having a plural number of digit sectors;

comparison means at the next location coupled to receive the corresponding selected sectors of the decrypted message authentication code and of the encrypted message authentication code for producing an output indication of the parity thereof; and

means at the next location responsive to said output indication for operating upon the received message in response to favorable comparison.

6. Apparatus as in claim 5 comprising:

means at the next location responsive to the unfavorable comparison for transmitting to the one location a selected sector of the random number.

7. Apparatus as in claim 5 comprising:

means at the next location for producing a second encoding key;

first encryption means at the next location coupled to receive the decrypted message authentication code and random number for encoding the same with the second encoding key in accordance with a fourth logical combination in response to said favorable comparison for producing a third output code for transmission to a destination location;

second encryption means at the next location coupled to receive the decrypted personal identification number for encoding the same with the second encoding key in accordance with a fifth logical combination in response to said favorable comparison for producing a fourth output code for transmission to a destination location;

means at the destination location for producing the second encoding key;

first decryption means at the destination location for receiving the third output code transmitted from said next location and the second encoding key for decoding the same according to said fourth logical combination to provide the random number and the message authentication code;

second decryption means at the destination location for receiving the fourth output code transmitted from said next location and the second encoding key for decoding the same according to said fifth logical combination to provide the personal identificaton number;

encryption means at the destination location for receiving the message and the sequence number for encoding the same with the decrypted personal identification number in accordance with the first logical combination to produce a message authentication code having a plural number of digit sectors;

means at the destination location for comparing corresponding selected sectors of the encrypted message authentication code and the decrypted message authentication code to produce output indications of favorable and unfavorable comparisons;

means at the destination location responsive to favorable output indication for operating upon the transmitted message and for transmitting a selected sector of the random number to said one location, and responsive to unfavorable comparison for transmitting another selected sector of the random number to said one location; and

comparator means at the one location coupled to receive the corresponding selected sectors of the random number for providing an output indication of the status of operation upon the message at the destination location.

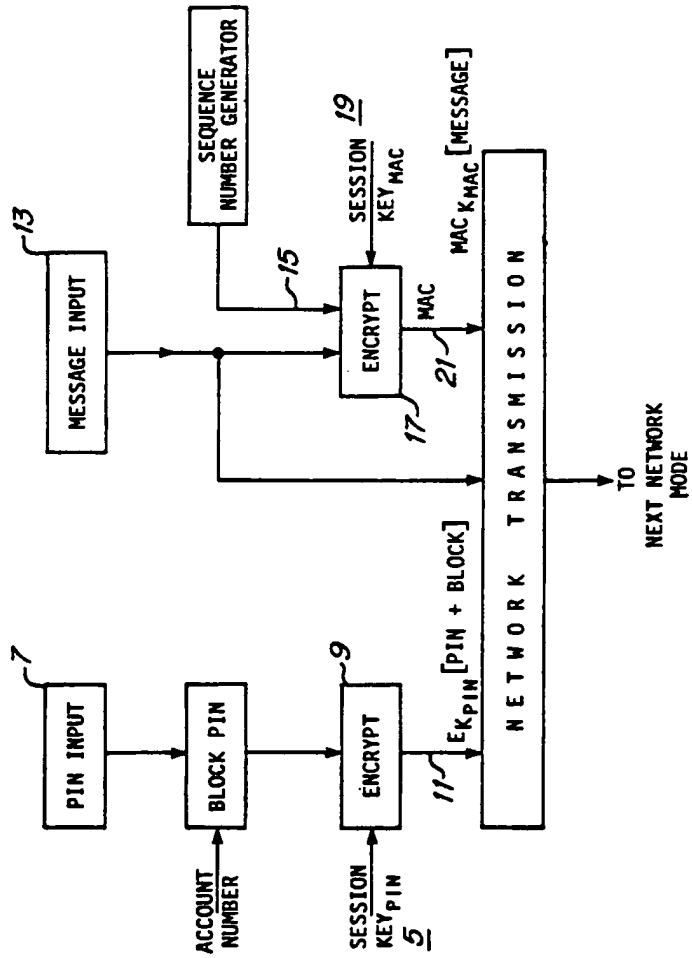


Figure 1
(PRIOR ART)

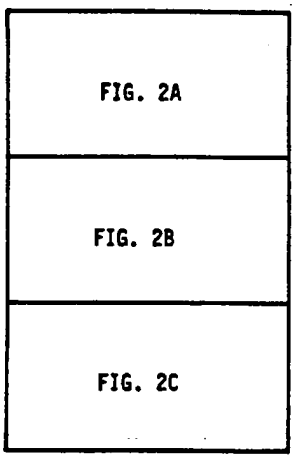


Figure 2

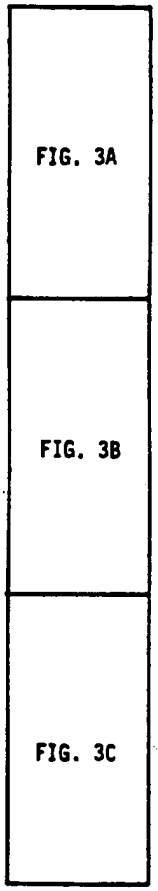


Figure 3

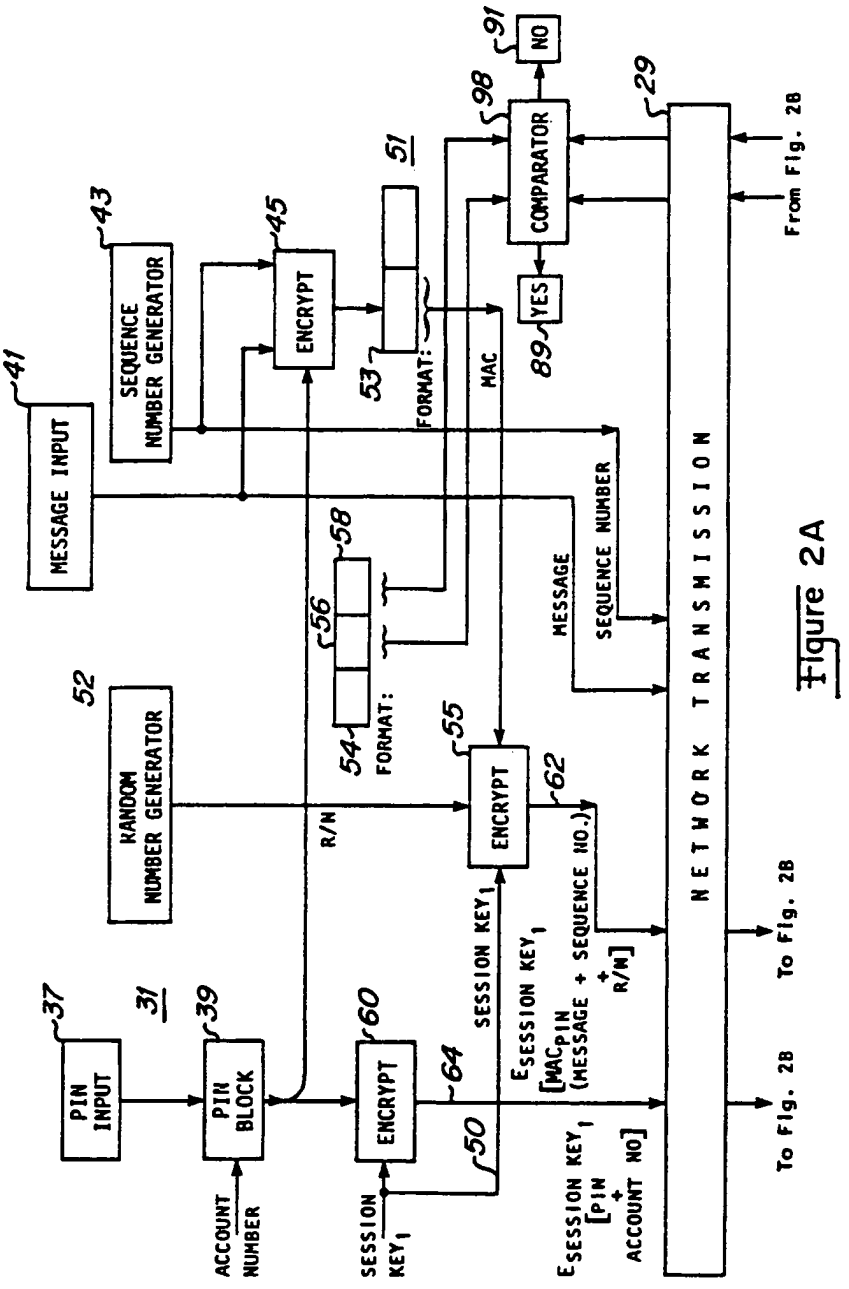


Figure 2A

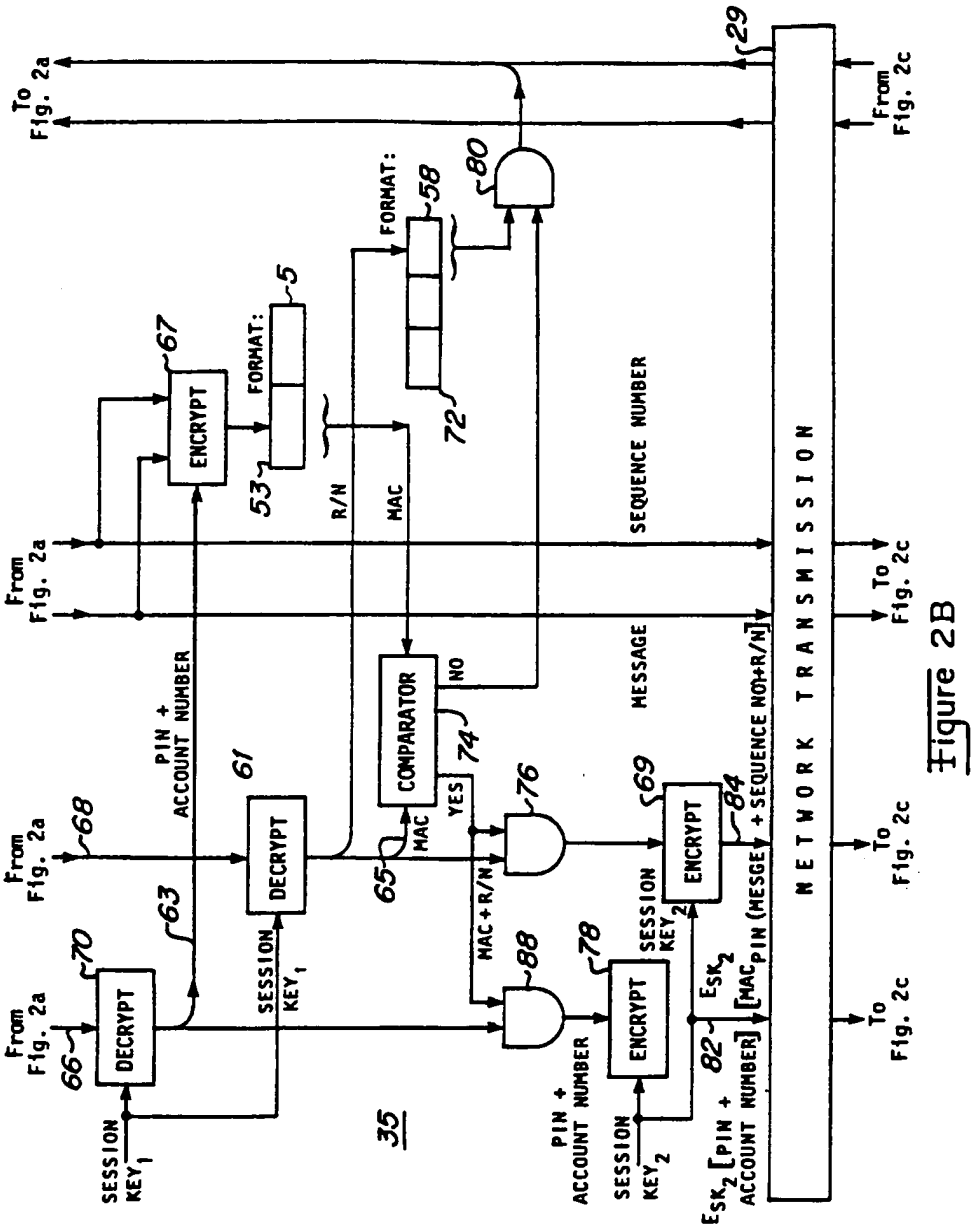


Figure 2B

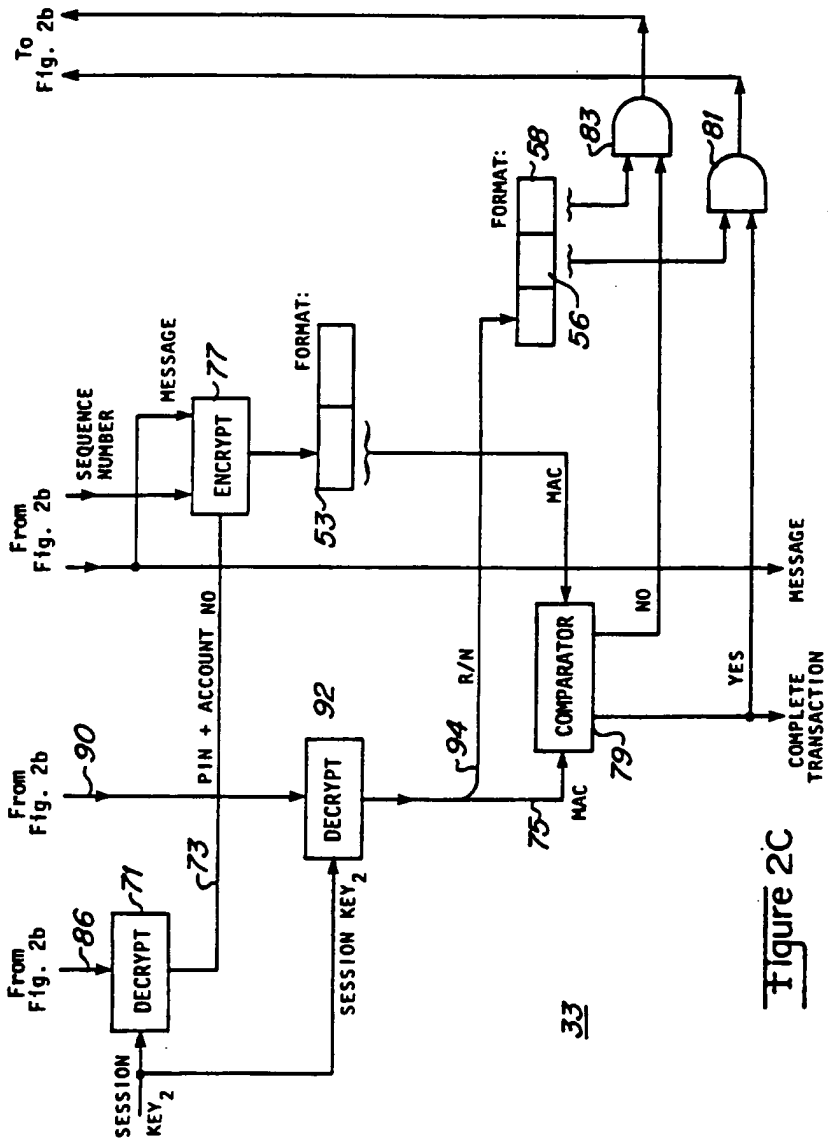


Figure 2C

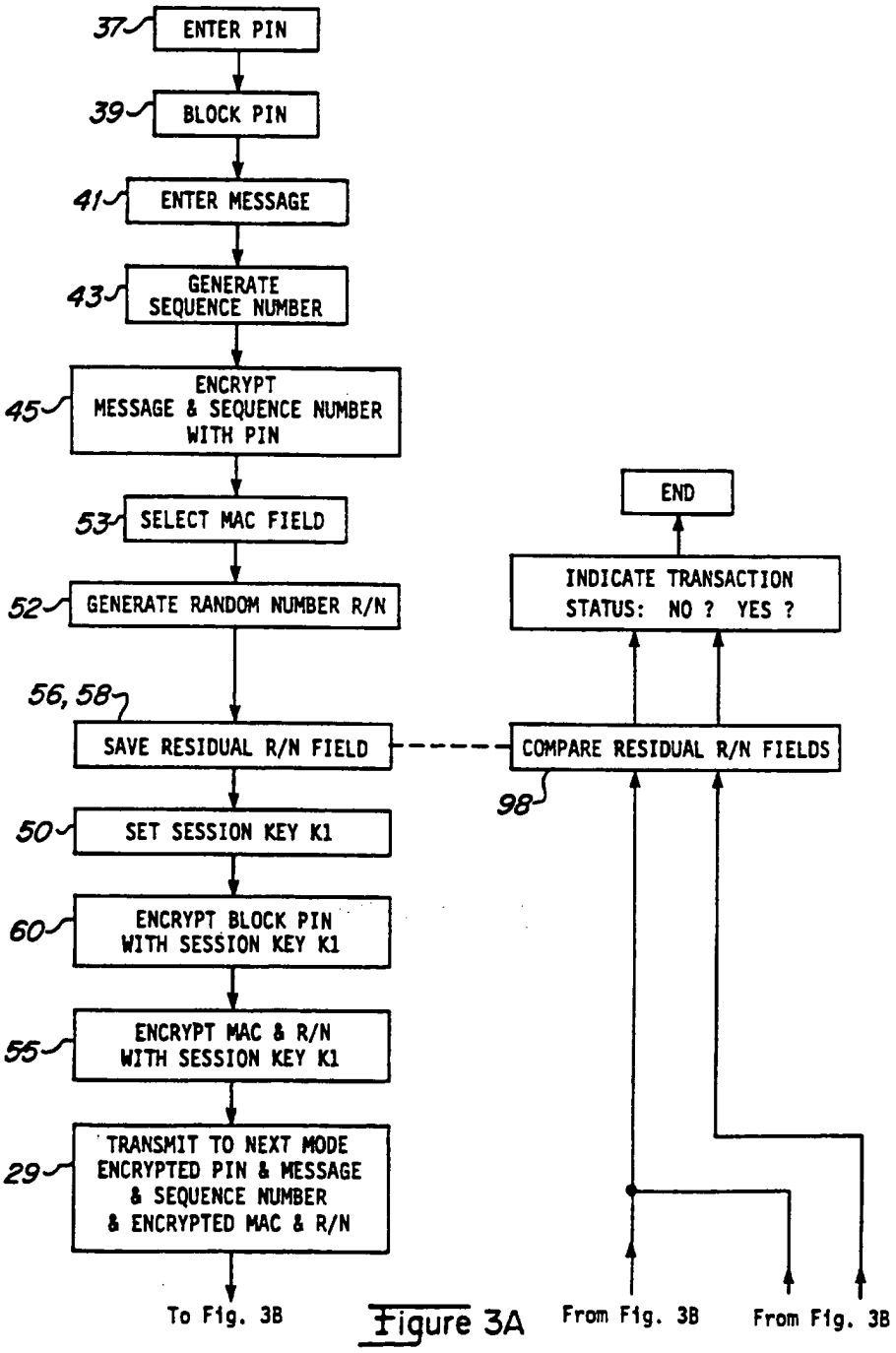


Figure 3A

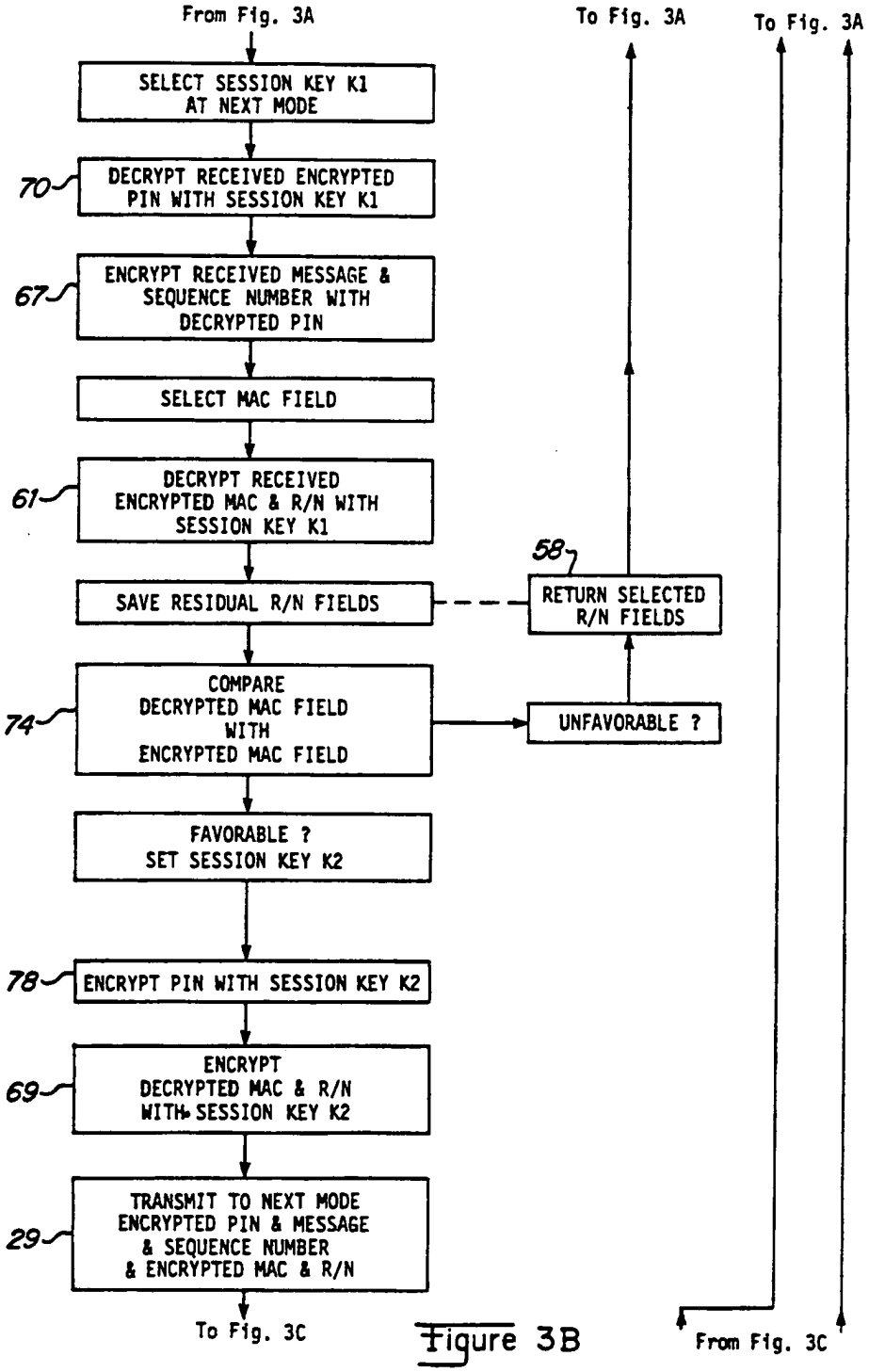


Figure 3B

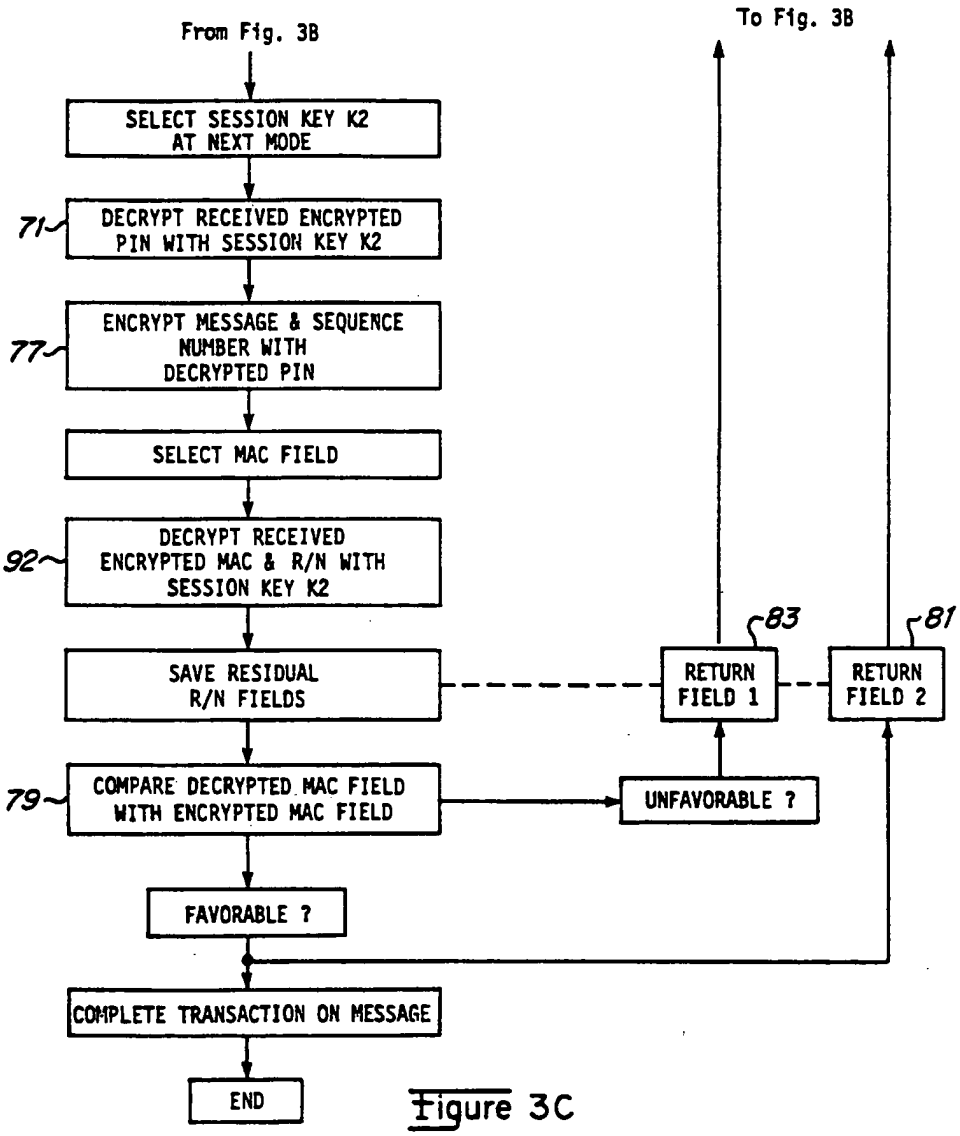


Figure 3C

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 94 10 5573

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X A	EP-A-0 391 261 (NIPPON TELEGRAPH) * abstract * * page 2, line 19 - line 31 * * page 4, line 31 - page 5, line 12 * * page 6, line 21 - line 25 * * page 7, line 2 - line 11 * * page 9, line 33 - line 54 * * page 16, line 41 - page 17, line 32 * * claim 1; figures 2A,2B *	1 2,5	G07F7/10
X A	US-A-5 101 373 (KATSUAKI) * column 5, line 32 - line 59 * * claims 1,4,5 *	1 2,3,5	
A	US-A-5 016 277 (HAMILTON) * column 16, line 60 - column 17, line 7 *	1,5	
A	EP-A-0 547 975 (BULL CP8) * abstract *	1,5	
A	EP-A-0 500 245 (TOSHIBA) * abstract * * claim 1 *	1,5	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G07F H04L
A	EP-A-0 494 796 (NCR CORPORATION) * abstract *	1,5	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 September 1994	Taccoen, J-F

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 679 977 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **95105400.6**

(51) Int. Cl.[6]: **G06F 1/00**, G06F 12/14

(22) Date of filing: **10.04.95**

(30) Priority: **25.04.94 US 238418**

(43) Date of publication of application:
02.11.95 Bulletin 95/44

(84) Designated Contracting States:
DE FR GB

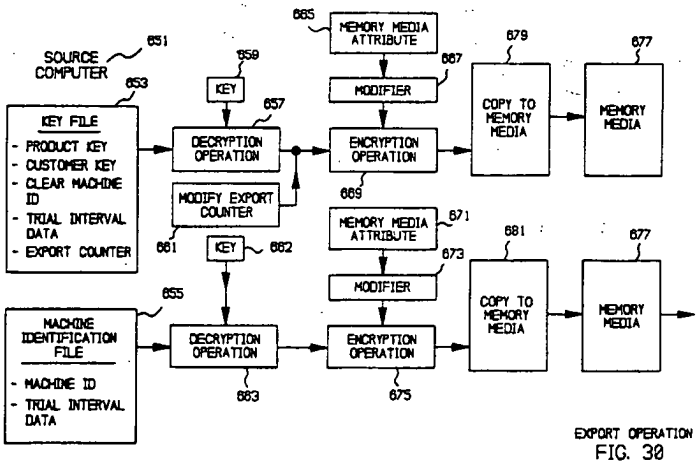
(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: **Cooper, Thomas Edward**
858 West Willow Street
Louisville,
Colorado 80027 (US)
Inventor: **Nagda, Jagdish**
701 Kalmia Avenue
Boulder,
Colorado 80304 (US)
Inventor: **Pryor, Robert Franklin**
7380 Mt. Meeker Road
Lognmont,
Colorado 80503 (US)

(74) Representative: **Schäfer, Wolfgang, Dipl.-Ing.**
IBM Deutschland
Informationssysteme GmbH
Patentwesen und Urheberrecht
D-70548 Stuttgart (DE)

(54) Method and apparatus enabling software trial allowing the distribution of software objects.

(57) A method and apparatus is provided for transferring encrypted files from a source computer to one or more target computers. An export program is provided in the source computer and an import program is provided in the target computer. The export program decrypts the encrypted file and tags the export operation with an export counter value. The clear text file is then encrypted with an encryption operation utilizing a key which is unique to a transfer memory media, such as diskette serial number. The memory media is carried to a target computer which utilizes the import file to decrypt the encrypted file.



EXPORT OPERATION
FIG. 30

EP 0 679 977 A1

CROSS-REFERENCE TO RELATED APPLICATION

The present application is related to U.S. Patent Application Serial No. 08/235,033, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Utilizing a Decryption Stub," further identified by Attorney Docket No. BT9-93-070; U.S. Patent Application Serial No. 08/235,035, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Allowing a Try-and-Buy User Interaction," further identified by Attorney Docket No. DA9-94-008; U.S. Patent Application Serial No. 08/235,032, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Generating a Machine-Dependent Identification," further identified by Attorney Docket No. DA9-94-009; and U.S. Patent Application Serial No. 08/235,418, entitled "Method and Apparatus for Enabling Trial Period Use of Software Products: Method and Apparatus for Utilizing an Encryption Header," further identified by Attorney Docket No. DA9-94-010, all filed of even date herewith by the inventors hereof and assigned to the assignee herein, and incorporated by reference herein.

BACKGROUND OF THE INVENTION

1.Technical Field:

The present invention relates in general to techniques for securing access to software objects, and in particular to techniques for temporarily encrypting and restricting access to software objects.

2. Description of the Related Art:

The creation and sale of software products has created tremendous wealth for companies having innovative products, and this trend will continue particularly since consumers are becoming evermore computer literate as time goes on. Computer software is difficult to market since the potential user has little opportunity to browse the various products that are available. Typically, the products are contained in boxes which are shrink-wrapped closed, and the potential customer has little or no opportunity to actually interact with or experience the software prior to purchasing. This causes considerable consumer dissatisfaction with products, since the consumer is frequently forced to serially purchase a plurality of software products until an acceptable product is discovered. This is perhaps one significant cause of the great amount of software piracy which occurs in our economy. A potential software purchaser will frequently "borrow" a set of diskettes from a friend or business associate, with the stated intention of using the software for a temporary period. Frequently, such temporary use extends for long intervals and the potential customer may never actually purchase a copy of the software product, and may instead rely upon the borrowed copy.

Since no common communication channel exists for the sampling of software products, such as those created in movie theaters by movie trailers, and in television by commercials, software manufacturers are forced to rely upon printed publications and direct mail advertisements in order to advertise new products and solicit new customers. Unfortunately, printed publications frequently fail to provide an accurate description of the product, since the user interaction with the product cannot be simulated in a static printed format. The manufacturers of computer software products and the customers would both be well served if the customers could have access to the products prior to making decisions on whether or not to purchase the product, if this could be accomplished without introducing risk of unlawful utilization of the product.

The distribution of encrypted software products is one mechanism a software vendor can utilize to distribute the product to potential users prior to purchase; however, a key must be distributed which allows the user access to the product. The vendor is then forced to rely entirely upon the honesty and integrity of a potential customer. Unscrupulous or dishonest individuals may pass keys to their friends and business associates to allow unauthorized access. It is also possible that unscrupulous individuals may post keys to publicly-accessible bulletin boards to allow great numbers of individuals to become unauthorized users. Typically, these types of breaches in security cannot be easily prevented, so vendors have been hesitant to distribute software for preview by potential customers.

SUMMARY OF THE INVENTION

It is one object of the present invention to provide a method and apparatus for distributing software objects from a producer to potential users which allows the user a temporary trial period without subjecting the software product to unnecessary risks of piracy or unauthorized utilization beyond the trial interval. Preferably this is accomplished by providing a software object on a computer-accessible memory media along with a file management program. Preferably, the software object is reversibly functionally limited, through one or more particular encryption operations. The computer-accessible memory media is shipped from the producer

to the potential user utilizing conventional mail and delivery services. Upon receipt, the potential user loads the file management program into a user-controlled data processing system and associates it with the operating system for the data processing system. Then, the computer-accessible memory media is read utilizing the user-controlled data processing system. The file management program is executed by the user-controlled data processing system and serves to restrict access to the software object for a predefined and temporary trial period. During the temporary trial mode of operation, the software object is temporarily enabled by reversing the reversible functional limitation of the software object. This is preferably accomplished by decryption of the encrypted software object when the software object is called by the operating system of the user-controlled data processing system. The file management program preferably prevents copying operations, so the encrypted software project is temporarily decrypted when it is called by the operating system. If the potential user elects to purchase the software object, a permanent use mode of operation is entered, wherein the functional limitation of the software object is permanently reversed, allowing unlimited use to the software object by the potential user. This facilitates browsing operations which allow the potential user to review the software and determine whether it suits his or her needs.

The file management program continuously monitors the operating system of the user-controlled data processing system for operating system input calls and output calls. The file management program identifies when the operating system of the user-controlled data processing system calls for a software object which is subject to trial-interval browsing. Then, the file management system fetches a temporary access key associated with the software object, and then examines the temporary access key to determine if it is valid. Next, the file management program reverses the functional limitation of the software object, and passes it to the data processing system for processing.

It is another objective of the present invention to provide a method and apparatus for distributing a software object from a source to a user, wherein a software object is encrypted utilizing a long-lived encryption key, and directed from the source to the user. The encrypted software object is loaded onto a user-controlled data processing system having a particular system configuration. A numerical machine identification based at least in part upon the particular configuration of the user-controlled data processing system is then derived. Next, a temporary key is derived which is based at least in part upon the numerical machine identification and the long-lived encryption key. A long-lived key generator is provided for receiving the temporary key and producing the long-lived encryption key. The temporary key allows the user to generate for a prescribed interval the long-lived encryption key to access the software object. These operations are performed principally by a file management program which is operable in a plurality of modes. These modes include a set up mode of operation, a machine identification mode of operation, and a temporary key derivation mode of operation. During the set up mode of operation, the file management program is loaded onto a user-controlled data processing system and associated with an operating system for the user-controlled data processing system. During the machine identification mode of operation, the file management program is utilized to derive a numerical machine identification based upon at least on attribute of the user-controlled data processing system. During the temporary key derivation mode of operation, a temporary key is derived which is based at least in part upon the numerical machine identification. The file management program also allows a trial mode of operation, wherein the file management program is utilized by executing it with the user-controlled data processing system to restrict access to the software object for an interval defined by the temporary key, during which the long-lived key generator is utilized in the user-controlled data processing system to provide the long-lived key in response to receipt of at least one input including the temporary key.

It is yet another objective of the present invention to provide a method and apparatus in a data processing system for securing access to particular files which are stored in a computer-accessible memory media. A file management program is provided as an operating system component of the data processing system. A plurality of files are stored in the computer-accessible memory media, including at least one encrypted file and at least one unencrypted file. For each encrypted file, a preselected portion is recorded in computer memory, a decryption block is generated which includes information which can be utilized to decrypt the file, and the decryption block is incorporated into the file in lieu of the preselected portion which has been recorded elsewhere in computer memory. The file management program is utilized to monitor data processing operation calls for a called file stored in the computer-accessible memory media. The file management program determines whether the called file has an associated decryption block. The file management program processes the called file in a particular manner dependent upon whether or not the called file has an associated decryption block. The incorporation of the decryption block does not change the size of the encrypted file, thus

preventing certain types of processing errors. During the trial interval, the encrypted file is maintained in an encrypted condition, and cannot be copied. If the potential user opts to purchase the software product, a permanent key is provided which results in replacement of the preselected portion to the file in lieu of the decryption block. Once the decryption block is removed, the encrypted file may be decrypted to allow unrestricted use by the purchaser. Preferably, the file management program is utilized to intercept files as they are called by the operating system, and to utilize the decryption block to derive a name for a key file and read the called file. The decryption block of each encrypted file includes a validation segment which is decrypted by the file management program and compared to a selected segment for the called file to determine whether the key can decrypt the particular file. If the decrypted validation segment matches a known clear text validation segment, the file is then dynamically decrypted as it is passed for further processing.

It is yet another objective of the present invention to provide a method and apparatus in a data processing system for securing access to particular files which are stored in a computer-accessible memory media. A file management program is provided as an operating system component of a data processing system. In a computer-accessible memory media available to the data processing system, at least one encrypted file and one unencrypted file are stored. The encrypted file has associated with it an unencrypted security stub which is at least partially composed of executable code. The file management program is utilized to monitor the data processing system calls for a called file stored in the computer accessible memory media, to determine whether the called file has an associated unencrypted security stub, and to process the called file in a particular manner dependent upon whether or not the called file has an associated unencrypted security stub. More particularly, if it is determined that the called file has no associated unencrypted security stub, the called file is allowed to be processed. However, if it is determined that the called file has an associated unencrypted security stub, it must be examined before a decision can be made about whether or not to allow it to be processed. First, the unencrypted security stub is examined in order to obtain information which allows decryption operations to be performed. Then, the decryption operations are performed. Finally, the called file is allowed to pass for further processing. Preferably, the called file is dynamically decrypted as it is passed to the operating system for processing. Also, the unencrypted security stub is separated from the called file prior to execution of the called file. However, if the unencrypted security stub accidentally remains attached to the called file, processing operations must be stopped, and a message must be posted in order to prevent the processor from becoming locked-up.

It is still another objective of the present invention to provide a method and apparatus for distributing a software object from a source to a user. A computer-accessible memory media is distributed from the source to a potential user. It includes a software object which is encrypted utilizing a predetermined encryption engine and a long-lived and secret key. An interface program is provided which facilitates interaction between the source and the user. The interface program includes machine identification module which generates a machine identification utilizing at least on predetermined attribute of the user-controlled data processing system. It also further includes a long-lived and secret key generator which receives as an input at least a temporary key and produces as an output a long-lived and secret key. A validation module is provided which tests temporary key determined its validity. The source of the software object maintains a temporary key generator which receives as an input at least a machine identification and produces an output of the temporary key. An interface program is loaded onto the user-controlled data processing system. The machine identification module is utilized to examine at least one predetermined attribute of the user-controlled data processing system and to generate the machine identification. During interaction between the source and the user, the machine identification is communicated over an insecure communication channel. At the source of the software object, the temporary key is generated utilizing the machine identification (and other information) as an input to the temporary key generator. During interaction between the source and the user, the temporary key is communicated, typically over an insecure communication channel. Next, the validation module is utilized to determine the validity of the temporary key. The long-lived and secret key generator is then utilized to receive the temporary key and generate the long-lived and secret key in order to decrypt and temporarily gain access to the software object. The user is also provided with an import module and an export module which allow for the utilization of portable memory media to transfer the encrypted software object, a key file, and a machine identification file from one machine in a distributed data processing system to another machine in the distributed data processing system, while allowing the temporary key to allow temporary trial access to the software object.

The above as well as additional objectives, features, and advantages of the present invention

will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a stand-alone data processing system, a telephone, and a variety of computer-accessible memory media all of which may be utilized in the implementation of the preferred technique of enabling trial period use of software products;

Figure 2 is a pictorial representation of a distributed data processing system which may utilize the technique of the present invention of enabling trial period use of software products;

Figure 3 is a block diagram representation of data processing system attributes which may be utilized to generate a machine identification, in accordance with the present invention;

Figure 4 is a block diagram depiction of a routine for encrypting software objects;

Figure 5 is a pictorial representation of the exchange of information between a source (a software vendor) and a user (a customer), in accordance with the teachings of the present invention;

Figure 6 is a flowchart representation of the broad steps employed in building a user interface shell, in accordance with the present invention;

Figure 7 is a flowchart representation of vendor and customer interaction in accordance with the present invention;

Figures 8, 9, 10a, and 10b depict user interface screens which facilitate trial period operations in accordance with the present invention;

Figure 11 depicts a user interface which is used to initiate a temporary access key;

Figure 12 is a block diagram depiction of the preferred technique of generating a machine identification;

Figure 13 is a block diagram depiction of an encryption operation which is utilized to encrypt a machine identification, in accordance with the present invention;

Figure 14 is a block diagram representation of the preferred technique for generating a product key, in accordance with the present invention;

Figure 15 is a block diagram representation of a preferred technique utilizing a temporary product key to generate a real key which can be utilized to decrypt one or more software objects;

Figures 16 and 17 depict a preferred technique of validating the real key which is derived in accordance with the block diagram of Figure 15;

Figure 18 is a block diagram depiction of the preferred routine for encyrpting a key file which contains information including a temporary product key;

Figure 19 is a block diagram depiction of the preferred technique of handling an encryption header in an encrypted file, in accordance with the present invention;

Figure 20 depicts in block diagram form the technique of utilizing a plurality of inputs in the user-controlled data processing system to derive the real key which may be utilized to decrypt an encrypted software object;

Figure 21 depicts a decryption operation utilizing the real key derived in accordance with Figure 20;

Figure 22 is a block diagram depiction of a comparison operation which is utilized to determine the validity of the real key;

Figure 23 depicts a decryption operation utilizing a validated real key;

Figures 24, 25, 26, 27, 28 depict the utilization of an encryption header in accordance with the present invention;

Figure 29 is a flowchart representation of the preferred technique of providing a trial period of use for an encrypted software object;

Figures 30 and 31 depict export and import operations which may be utilized to perform trial period use operations in a distributed data processing system;

Figures 32 and 33 provide an alternative view of the import and export operations which are depicted in Figures 30 and 31;

Figures 34 and 35 provide a block diagram depiction of an alternative technique for performing an export/import operation.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

The method and apparatus of the present invention for enabling trail period use of software products can be utilized in stand-alone PCs such as that depicted in Figure 1, or in distributed data processing systems, such as that depicted in Figure 2. In either event, temporary trial period access to one or more software products depends upon utilization of the trial product on a particular data processing system with particular data processing system attributes. This is accomplished by encrypting the trial software product utilizing a temporary access key which is based upon one or more data

processing system attributes. Figure 3 graphically depicts a plurality of system configuration attributes, which may be utilized in developing a temporary access key, as will be described in greater detail herebelow. To begin with, the environment of the stand-alone data processing system of Figure 1, and the distributed data processing system of Figure 2 will be described in detail, followed by a description of particular system configuration attributes which are depicted in Figure 3.

With reference now to the figures and in particular with reference to Figure 1, there is depicted a pictorial representation of data processing system 10 which may be programmed in accordance with the present invention. As may be seen, data processing system 10 includes processor 12 which preferably includes a graphics processor, memory device and central processor (not shown). Coupled to processor 12 is video display 16 which may be implemented utilizing either a color or monochromatic monitor, in a manner well known in the art. Also coupled to processor 12 is keyboard 14. Keyboard 14 preferably comprises a standard computer keyboard which is coupled to the processor by means of a cable.

Also coupled to processor 12 is a graphical pointing device, such as mouse 20. Mouse 20 is coupled to processor 12, in a manner well known in the art, via a cable. As is shown, mouse 20 may include left button 24, and right button 26, each of which may be depressed, or "clicked", to provide command and control signals to data processing system 10. While the disclosed embodiment of the present invention utilizes a mouse, those skilled in the art will appreciate that any graphical pointing device such as a light pen or touch sensitive screen may be utilized to implement the method of the present invention. Upon reference to the foregoing, those skilled in the art will appreciate that data processing system 10 may be implemented utilizing a so-called personal computer, such as the Model 80 PS/2 computer manufactured by International Business Machines Corporation of Armonk, New York.

While the present invention may be utilized in stand-alone data processing systems, it may also be utilized in a distributed data processing system, provided the import and export routines of the present invention are utilized to transfer one or more encrypted files, their encrypted key files, and associated file management programs through a portable memory media (such as diskettes or tapes) between particular data processing units within the distributed data processing system. While the import and export routines of the present invention will be described in greater detail herebelow, it is important that a basic distributed data processing system be described and understood.

Figure 3 provides a block diagram depiction of a plurality of data processing system attributes which may be utilized to uniquely identify a particular data processing system (whether a stand-alone or a node in a distributed data processing system), and which further can be utilized to generate in the machine identification value which is utilized to derive or generate a temporary access product key which may be utilized to gain access to an encrypted product for a particular predefined trial interval. A data processing system may include a particular system bus 60 architecture, a particular memory controller 74, bus controller 76, interrupt controller 78, keyboard mouse controller 80, DMA controller 66, VGA video controller 82, parallel controller 84, serial controller 86, diskette controller 88, and disk controller 82. Additionally, a plurality of empty or occupied slots 106 may be used to identify the particular data processing system. Each particular data processing system may have attributes which may be derived from RAM 70, ROM 68, or CMOS RAM 72. End devices such as printer 96, monitor 94, mouse 92, keyboard 90, diskette 100, or disk drive 104 may be utilized to derive one or more attributes of the data processing system which may be processed in a predetermined manner to derive a machine identification value. The derivation of the machine identification value will be described in greater detail below. The present invention is directed to an efficient method of distributing software programs to users which would provide to them a means to try the program before obtaining (by purchasing) a license for it. In accordance with this concept, complete programs are distributed to potential users on computer-accessible memory media such as diskettes or CD-ROMs. The concept is to generate keys that allow the user to access the programs from the distributed media. In this environment, a file management program provides a plurality of interfaces which allows the user to browse the different products. The interfaces allow ordering and unlocking of the software products contained on the distributed media. Unlocking of the software product is accomplished by the reception, validation, and recording of a temporary access (decryption) key.

The file management program is resident in the user-controlled data processing system and becomes a part of the operating system in the user's computer. An example of such a resident program (in the PC DOS environment) would be a resident program TSR, for "terminate and stay resident" operations, that intercepts and handles DOS file input and output operations. When a temporary access key is provided to a user, system files are checked to see if this file has been used in a trial mode of operation before. If the product has

never been used in a trial mode of operation, the temporary key is saved. Once the trial mode of operation key exists, an encrypted application can only be run if it is initiated by the file management program. The file management program will recognize that the application is encrypted and that a valid trial mode of operation key exists for the particular operation. A valid trial mode of application key is one that has not expired. The trial mode of operation may be defined by either a timer, or a counter. A timer can be used to count down a particular predefined period (such as thirty days); alternatively, the counter can be used to decrement through a predefined number of trial "sessions" which are allowed during the trial mode of operation. If the key is valid, the file management program communicates directly with the TSR and enables the trial mode of operation for a particular encrypted application. The file management program then kicks off the encrypted application. The code which is resident in the operating system of the user-controlled data processing system maintains control over the operating system. It monitors the use of the trial mode of operation keys to allow files to be decrypted and loaded into memory, but prevents the encrypted files from being decrypted and copied to media. This is done by using the operating system to determine which applications are trying to access the data and only allowing the applications that have permission to access the data to do so.

Figure 4 is a block diagram depiction of a routine for encrypting software objects. The binary characters which make up software object 201 are supplied as an input to encryption engine 205. Real key 203 is utilized as an encryption key in encryption engine 205. The output of encryption engine 205 is an encrypted software object 207. Encryption engine 205 may be any conventional encryption operation such as the published and well known DES algorithm; alternatively, the encryption engine 205 may be an exclusive-OR operation which randomizes software object 201.

Figure 5 is a pictorial representation of the exchange of information between a source 209 (a software vendor) and a user 211 (a potential customer, in accordance with the teachings of the present invention. The arrows between source 209 and user 211 represent exchanges of objects or information between vendor 209 and 211. In the exchange of flow 203, computer-accessible memory media is directed from source 209 to user 211. This transfer may occur by US mail delivery, courier delivery, express service delivery, or by delivery through printed publications such as books and magazines. Alternatively, an electronic document may be transferred from source 209 to user 211 utilizing electronic mail or other transmission techniques. In flow 215, user-specific information, preferably including a unique machine identification number which identifies the data processing system of user 211, is transferred from user 211 to source 209 via an insecure communication channel; typically, this information is exchanged over the telephone, but may be passed utilizing electronic mail or other communication techniques. In flow 217, source 209 provides a product key to user 211. The product key allows the product contained in the memory media to be temporarily accessed for a prescribed and predefined interval. This interval is considered to be a "trial" interval during which user 211 may become familiar with the software and make a determination on whether or not he or she wishes to purchase the software product. User 211 must communicate additionally with source 209 in order to obtain permanent access to the software product. The product key allows user 211 to obtain access to the software product for a particular predefined time interval, or for a particular number of predefined "sessions." As time passes, the user's clock or counter runs down. At the termination of the trial period, further access is denied. Therefore, the user 211 must take affirmative steps to contact source 209 and purchase a permanent key which is communicated to user 211 and which permanently unlocks a product to allow unrestricted access to the software product.

The communication between source 209 and user 211 is facilitated by a user interface. The creation of the interface is depicted in flowchart form in Figure 6. The process begins at software block 219, and continues at software block 221, wherein source 209 makes language and locale selections which will determine the language and currencies utilized in the interface which facilitates implementation of the trial period use of the software products. A plurality of software products may be bundled together and delivered to user 211 on a single computer-accessible memory media. Therefore, in accordance with software block 223, source 209 must make a determination as to the programs which will be made available on a trial basis on the computer-accessible memory media, and the appropriate fields are completed, in accordance with software block 223. Next, in accordance with software block 225, the programs are functionally limited or encrypted. Then, in accordance with software block 227, the shell is loaded along with the computer program products onto a computer-accessible memory media such as a diskette or CD ROM. The process ends at software block 229.

Figure 7 is a flowchart representation of vendor and customer interaction in accordance with the present invention. The flow begins at software block 231, and continues at step 233, wherein

computer-accessible memory media are distributed to users for a try-and-buy trial interval. Then, in accordance with step 235, the file management program is loaded from the computer-accessible memory media onto a user-controlled data processing system for execution. The file management program includes a plurality of interface screens which facilitate interaction between the vendor and the customer, which and which set forth the options available to the customer. Thus, in accordance with step 237, the file management program allows browsing and displays appropriate user interfaces. Next, in accordance with step 239, the customer and the vendor interact, typically over the telephone or electronic mail, to allow the vendor to gather information about the customer and to distribute a temporary key which allows access to one or more software products which are contained on the computer-accessible memory media for a predefined trial interval. Typically, the interval will be defined by an internal clock, or by a counter which keeps track of the number of sessions the potential purchaser has with a particular software product or products. Step 241 represents the allowance of the trial interval use. Then, in accordance with software block 243, the file management program monitors and oversees all input and output calls in the data processing system to prevent unauthorized use of the encrypted software products contained on the computer-accessible memory media. In the preferred embodiment of the present invention, the file management program monitors for calls to encrypted files, and then determines whether access should be allowed or denied before the file is passed for further processing. The customer can assess the software product and determine whether he or she desires to purchase it. If a decision is made to purchase the product, the customer must interact once again with the vendor, and the vendor must deliver to the customer a permanent key, as is set forth in step 245. The process ends when the customer receives the permanent key, decrypts the one or more software products that he or she has purchased, and is then allowed ordinary and unrestricted access to the software products.

Figures 8, 9, 10a, and 10b depict user interface screens which facilitate trial period operations in accordance with the present invention. Figure 8 depicts an order form user interface 249 which is displayed when the customer selects a "view order" option from another window. The order form user interface 249 includes a title bar 251 which identifies the software vendor and provides a telephone number to facilitate interaction between the potential customer and the vendor. An order form field 255 is provided which identifies one or more software products which may be examined during a trial interval period of operation. A plurality of subfields are provided including quantity subfield 259, item subfield 257, description subfield 260, and price subfield 253. Delete button 261 allows the potential customer to delete items from the order form field. Subtotal field 263 provides a subtotal of the prices for the ordered software. Payment method icons 265 identify the acceptable forms of payment. Of course, a potential user may utilize the telephone number to directly contact the vendor and purchase one or more software products; alternatively, the user may select one or more software products for a trial period mode of operation, during which a software product is examined to determine its adequacy. A plurality of function icons 267 are provided at the lowermost portion of order form interface 249. These include a close icon, fax icon, mail icon, print icon, unlock icon, and help icon. The user may utilize a graphical pointing device in a conventional point-and-click operation to select one or more of these operations. The fax icon facilitates interaction with the vendor utilizing a facsimile machine or facsimile board. The print icon allows the user to generate a paper archival copy of the interaction with the software vendor.

The customer, the computer-accessible memory media, and the computer system utilized by the customer are identified by media identification 269, customer identification 273, and machine identification 271. The media identification is assigned to the computer-accessible memory media prior to shipping to the potential customer. It is fixed, and cannot be altered. The customer identification 273 is derived from interaction between the potential customer and the vendor. Preferably, the customer provides answers to selected questions in a telephone dialogue, and the vendor supplies a customer identification 273, which is unique to the particular customer. The machine identification 271 is automatically derived utilizing the file management program which is resident on the computer-accessible memory media, and which is unique to the particular data processing system being utilized by the potential customer. The potential customer will provide the machine identification to the vendor, typically through telephone interaction, although fax interaction and regular mail interaction is also possible.

Figure 9 is a representation of an order form dialog interface 275. This interface facilitates the acquisition of information which uniquely identifies the potential customer, and includes name field 277, address field 279, phone number field 281, facsimile number field 283, payment method field 285, shipping method field 287, account number field 289, expiration date field 291, value added tax ID field 293. Order information dialog interface 275

further includes print button 295 and cancel button 297 which allow the potential user to delete information from these fields, or to print a paper copy of the interface screen.

Figures 10a and 10b depict unlock dialog interface screens 301, 303. The user utilizes a graphical pointing device to select one or more items which are identified by the content item number field 307 and description field 309 which are components of unlock list 305. The interface further includes customer ID field 313 and machine ID field 315. Preferably, the vendor provides the customer identification to the customer in an interaction via phone, fax, or mail. Preferably, the customer provides to the vendor the machine identification within machine identification field 315 during interaction via phone, fax, or mail. Once the information is exchanged, along with an identification of the products which are requested for a trial interval period of operation, a temporary access key is provided which is located within key field 311. The key will serve to temporarily unlock the products identified and selected by the customer. Close button 319, save button 317, and help button 321 are also provided in this interface screen to facilitate user interaction.

Figure 10b depicts a single-product unlock interface screen 303. This interface screen includes only machine identification field 315, customer identification field 315, and key field 311. The product which is being unlocked need not be identified in this interface, since the dialog pertains only to a single product, and it is assumed that the user knows the product for which a temporary trial period of operation is being requested. Save button 317, cancel button 319, and help button 321 are also provided in this interface to facilitate operator interaction.

Figure 11 depicts a user interface screen which is utilized in unlocking the one or more encrypted products for the commencement of a trial interval mode of operation. The starting date dialog of Figure 11 is displayed after the "SAVE" push button is selected in the unlock dialog of either Figure 10a or Figure 10b. The user will be prompted to verify the correct starting date which is provided in date field 310. The user responds to the query by pointing and clicking to either the "continue" button 312, the "cancel" button 314, or the "help" button 316. The date displayed in field 310 is derived from the system clock of the user-controlled data processing system. The user may have to modify the system clock to make the date correspond to the official or stated date of commencement of the trial period of operation.

A trial interval operation can take two forms: one form is a functionally disabled product that allows a user to try all the features, but may not allow a critical function like printing or saving of data files. Another type of trial interval is a fully functional product that may be used for a limited time. This requires access protection, and allows a customer to try all the functions of a product for free or for a nominal fee. Typically, in accordance with the present invention, access to the product is controlled through a "timed" key. The trial period for using the product is a fixed duration determined by the vendor. The trial period begins when the key is issued. In accordance with the present invention, the products being previewed during the trial interval of operation can only be run from within a customer shell. A decryption driver will not allow the encrypted products to be copied in the clear, nor will it allow the product to be run outside the customer's shell. In an alternative embodiment, the trial interval is defined by a counter which is incremented or decremented with each "session" the customer has with the product. This may allow the customer a predefined number of uses of the product before decryption is no longer allowed with the temporary key.

The limits of the temporary access key are built into a "control vector" of the key. Typically, a control vector will include a short description of the key, a machine identification number, and a formatted text string that includes the trial interval data (such as a clock value or a counter value). The control vector cannot be altered without breaking the key. When a protected software product is run, the usage data must be updated to enforce the limits of the trial interval period of operation. In order to protect the clock or counter from tampering, its value is recorded in a multiple number of locations, typically in encrypted files. In the preferred embodiment of the present invention, the trial interval information (clock value and/or counter value) is copied to a "key file" which will be described in further detail herebelow, to a machine identification file, which will also be discussed herebelow, and to a system file. When access to an encrypted program is requested, all of these locations are checked to determine if the value for the clock and/or counter is the same. It is unlikely that an average user has the sophistication to tamper successfully with all three files. In the preferred embodiment, a combination of a clock and a counter is utilized to prevent extended use of backup and restore operations to reset the system clock. Although it is possible to reset a PC's clock each time a trial use is requested, this can also be detected by tracking the date/time stamps of certain files on the system and using the most recent date between file date/time stamps and the system clock. As stated above, one of the three locations the timer and/or counter information is stored is a system file. When operating in an OS/2 operating

system, the time and usage data can be stored in the system data files, such as the OS2.INI in the OS/2 operating system. The user will have to continuously backup and restore these files to reset the trial and usage data. These files contain other data that is significant to the operation of the user system. The casual user can accidentally lose important data for other applications by restoring these files to an older version. In the present invention, these protection techniques greatly hinder a dishonest user's attempts to extend the trial interval use beyond the authorized interval.

In broad overview, in the present invention, the vendor loads a plurality of encrypted software products onto a computer-accessible memory media, such as a CD ROM or magnetic media diskette. Also loaded onto the computer-accessible memory media is a file management program which performs a plurality of functions, including the function of providing a plurality of user interface screens which facilitate interaction between the software vendor and the software customer. The computer-accessible memory media is loaded onto a user-controlled data processing system, and the file management program is loaded for execution. The file management program provides a plurality of user-interface screens to the software customer which gathers information about the customer (name, address, telephone number, and billing information) and receives the customer selections of the software products for which a trial interval is desired. Information is exchanged between the software vendor card customer, including: a customer identification number, a product identification number, a media identification number, and a machine identification number. The vendor generates the customer identification number in accordance with its own internal record keeping. Preferably, the representative of the software vendor gathers information from the software customer and types this information into a established blank form in order to identify the potential software customer. Alternatively, the software vendor may receive a facsimile or mail transmission of the completed order information dialog interface screen 275 (of Figure 9). The distributed memory media (such as CDs and diskettes) also include a file management program which is used to generate a unique machine identification based at least in part upon one attribute of the user-controlled data processing system. This machine identification is preferably a random eight-bit number which is created during a one-time setup process. Preferably, eight random bits are generated from a basic random number generator using the system time as the "seed" for the random number generator. Preferably, check bits are added in the final result. Those check bits are critical to the order system because persons taking orders must key in the machine ID that the customer reads over the phone. The check bits allow for instant verification of the machine ID without requiring the customer to repeat the number. Preferably, a master file is maintained on the user-controlled data processing system which contains the clear text of the machine identification and an encrypted version of the machine identification.

When the software customer places an order for a temporary trial use of the software products, he or she verbally gives to the telephone representative of the software vendor the machine identification. In return, the telephone representative gives the software customer a product key which serves as a temporary access key to the encrypted software products on the computer-accessible memory media, as well as a customer identification number. Preferably, the product key is a function of the machine identification, the customer number, the real encryption key for the programs or programs ordered, and a block of control data. The software customer may verify the product key by combining it with the customer number, and an identical block of control data to produce the real encryption key. This key is then used to decrypt an encrypted validation segment, to allow a compare operation. If the encrypted validation segment is identical to known clear text for the validation segment, then the user's file management program has determined that the product key is a good product key and can be utilized for temporary access to the software products. Therefore, if the compare matches, the key is stored on the user-controlled data processing system in a key file. Preferably, the key file contains the product key, a customer key (which is generated from the customer number and an internal key generating key) and a clear ASCII string containing the machine identification. All three items must remain unchanged in order for the decryption tool to derive the real encryption key. To further tie the key file to this particular user-controlled data processing system, the same key file is encrypted with a key that is derived from system parameters. These system parameters may be derived from the configuration of the data processing system.

Stated broadly, in the present invention the temporary key (which is given verbally over the phone, typically) is created from an algorithm that utilizes encryption to combine the real key with a customer number, the machine identification number, and other predefined clear text. Thus, the key is only effective for a single machine: even if the key were to be given to another person, it would not unlock the program on that other person's machine. This allows the software vendor to market software programs by distributing complete programs on computer-accessible memory media

such as diskettes or CD ROMs, without significant risk of the loss of licensing revenue.

Some of the preferred unique attributes of the system which may be utilized for encryption operations include the hard disk serial number, the size and format of the hard disk, the system model number, the hardware interface cards, the hardware serial number, and other configuration parameters. The result of this technique is that a machine identification file can only be decrypted on a system which is an identical clone of the user-controlled data processing system. This is very difficult to obtain, since most data processing systems have different configurations, and the configurations can only be matched through considerable effort. These features will be described in detail in the following written description.

Turning now to Figure 12, the file management program receives the distributed computer-accessible memory media with encrypted software products and a file management program contained therein. The file management program assesses the configuration of the user-controlled data processing system, as represented in step 351 of Figure 12. The user-specific attributes of the data processing system are derived in step 353, and provided as an input to machine identification generator 355, which is preferably a random number generator which receives a plurality of binary characters as an input, and generates a pseudo-random output which is representative of machine identification 357. The process employed by machine identification generator 355 is any conventional pseudo-random number generator which receives as an input of binary characters, and produces as an output a plurality of pseudo-random binary characters, in accordance with a predefined algorithm.

With reference now to Figure 13, machine identification 357 is also maintained within the file management program in an encrypted form. Machine identification 357 is supplied as an input to encryption engine 359 to produce as an output the encrypted machine identification 361. Encryption engine 359 may comprise any convention encryption routine, such as the DES algorithm. A key 363 is provided also as an input to encryption engine 359, and impacts the encryption operation in a conventional manner. Key 363 is derived from system attribute selector 365. The types of system attributes which are candidates for selection include system attribute listing 367 which includes: the hard disk serial number, the size of the hard disk, the format of the hard disk, the system model number, the hardware interface card, the hardware serial number, or other configuration parameters.

In accordance with the present invention, the clear text machine identification 357 and the encrypted machine identification 361 are maintained in memory. Also, in accordance with the present invention, the file management program automatically posts the clear text machine identification 357 to the appropriate user interface screens. The user then communicates the machine identification to the software vendor where it is utilized in accordance with the block diagram of Figure 14. As is shown, product key encryption engine 375 is maintained within the control of the software vendor. This product key encryption engine 375 receives as an input: the machine identification 357, a customer number 369 (which is assigned to the customer in accordance with the internal record keeping of this software vendor), the real encryption key 371 (which is utilized to decrypt the software products maintained on the computer-accessible memory media within the custody of the software customer), a control block text 373 (which can be any predefined textural portion), and trial interval data 374 (such as clock and/or counter value which defines the trial interval of use). Product key encryption engine produces as an output a product key 377. Product key 377 may be communicated to the software customer via an insecure communication channel, without risk of revealing real key 371. Real key 371 is masked by the encryption operation, and since the product key 377 can only be utilized on a data processing system having a configuration identical to that from which machine identification 357 has been derived, access to the encrypted software product is maintained in a secure condition.

Upon delivery of product key 377, the file management program resident in the user-controlled data processing system utilizes real key generator 379 to receive a plurality of inputs, including product key 377, customer number 369, control block text 373, machine identification 357 and trial interval data 374. Real key generator 379 produces as an output the derived real key 381.

Encryption and decryption algorithm utilized to perform the operations of the product key encryption engine 375 and the real key generator 379 (of Figures 14 and 15) is described and claimed in co-pending U.S. Patent Application Serial No. 07/964,324, filed October 21, 1992, entitled "Method and System for Multimedia Access Control Enablement", which is incorporated herein as if fully set forth.

Next, as is depicted in Figures 16 and 17, the derived real key 381 is tested to determine the validity and authenticity of the product key 377 which has been provided by the software vendor. As is shown, the derived real key 381 is supplied as an input to encryption engine 385. A predetermined encrypted validation data segment 383 is supplied as the other input to encryption engine 385. Encryption engine supplies as an output de-

rived clear validation text 387. Then, in accordance with Figure 17, the derived clear validation text 387 is compared to the known clear validation text 391 in comparator 389. Comparator 389 simply performs a bit-by-bit comparison of the derived clear validation text 387 with the known clear validation text 391. If the derived clear validation text 387 matches the known clear validation text 391, a key file is created in accordance with step 393; however, if the derived clear validation text 387 does not match the known clear validation text 391, a warning is posted to the user-controlled data processing system in accordance with step 395.

Turning now to Figure 18, key file 397 is depicted as including the temporary product key, the customer key (which is an encrypted version of the customer number), the machine identification number in clear text and the trial interval data (such as a clock and/or counter value). This key file is supplied as an input to encryption engine 399. Key 401 is also provided as an input to encryption engine 399. Key 401 is derived from unique system attributes 403, such as those system attributes utilized in deriving the machine identification number. Encryption engine 399 provides as an output the encrypted key file 405.

Figures 19, 20, 21, 22, and 23 depict operations of the file management program after a temporary access key has been received, and validated, and recorded in key file 397 (of Figure 18).

Figure 19 is a block diagram representation of the steps which are performed when an encrypted software product is called for processing by the user-control data processing system. The encrypted file 405 is fetched, and a "header" portion 407 is read by the user-controlled data processing system. The header has a number of components including the location of the key file. The location of the key file is utilized to fetch the key file in accordance with step 409. The header further includes an encrypted validation text 411. The encrypted validation text 411 is also read by the user-controlled data processing system. As is stated above (and depicted in Figure 18) the key file includes the product key 419, a customer key 417, and the machine identification 415. These are applied as inputs to decryption engine 413. Decryption engine 413 provides as an output real key 421. Before real key 421 is utilized to decrypt encrypted software products on the distributed memory media, it is tested to determine its validity. Figure 21 is a block diagram of the validation testing. Encrypted validation text 423, which is contained in the "header", is provided as an input to decryption engine 425. Real key 421 (which was derived in the operation of Figure 20) is also supplied as an input to decryption engine 425. Decryption engine 425 provides as an output clear validation text 427.

As is set forth in block diagram form in Figure 22, clear validation text 427 is supplied as an input to comparator 429. The known clear validation text 431 is also supplied as an input to comparator 429. Comparator 429 determines whether the derived clear validation text 427 matches the known clear validation text 431. If the texts match, the software object is decrypted in accordance with step 433; however, if the validation text portions do not match, a warning is post in accordance with step 435. Figure 23 is a block diagram depiction of the decryption operation of step 433 of Figure 22. The encrypted software object 437 is applied as an input to decryption engine 439. The validated real key 441 is also supplied as an input to decryption engine 439. Decryption engine 439 supplies as an output the decrypted software object 443.

The encryption header is provided to allow for the determination of whether or not a file is encrypted when that file is stored with clear-text files. In providing the encryption header for the encrypted file, it is important that the file size not be altered because the size may be checked as part of a validation step (unrelated in any way to the concept of the present invention) during installation. Therefore, making the file larger than it is suppose to be can create operational difficulties during installation of the software. The encryption header is further necessary since the file names associated with the encrypted software products cannot be modified to reflect the fact that the file is encrypted, because the other software applications that may be accessing the encrypted product will be accessing those files utilizing the original file names. Thus, altering the file name to indicate that the file is encrypted would prevent beneficial and desired communication between the encrypted software product and other, perhaps related, software products. For example, spreadsheet applications can usually port portions of the spreadsheet to a related word processing program to allow the integration of financial information into printed documents. Changing the hard-coded original file name for the word processing program would prevent the beneficial communication between these software products. The encryption header of the present invention resolves these problems by maintaining the encrypted file at its nominal file length, and by maintaining the file name for the software product in an unmodified form.

Figure 24 graphically depicts an encrypted file with encryption header 451. The encryption header 451 includes a plurality of code segments, including: unique identifier portion 453, the name of the key file portion 455, encrypted validation segment 457, encryption type 459, offset to side file 461, and encrypted file data 463. Of course, in this view, the encrypted file data 463 is representative of the

encrypted software product, such as a word processing program or spreadsheet. The encryption header 451 is provided in place of encrypted data which ordinarily would comprise part of the encrypted software product. The encryption header is substituted in the place of the first portion of the encrypted software product. In order to place the encryption header 451 at the front of the encrypted software product of encrypted file data 463, a portion of the encrypted file data must be copied to another location. Offset to side file 461 identifies that side file location where the displaced file data is contained.

Figure 25 graphically depicts the relationship between the directory of encrypted files and the side files. As is shown, the directory of encrypted files 465 includes file aaa, file bbb, file ccc, file ddd, through file nnn. Each of these files is representative of a directory name for a particular encrypted software product. Each encrypted software product has associated with it a side file which contains the front portion of the file which has been displaced to accommodate encryption header 451 without altering the size of the file, and without altering the file name. File aaa has associated with it a side file AAA. Software product file bbb has associated with it a side file BBB. Encrypted software product ccc has associated with it a side file CCC. Encrypted software product ddd has associated with it a side file DDD. Encrypted software product nnn has associated with it a side file NNN. In Figure 25, directory names 467, 469, 471, 473, 475 are depicted as being associated with side files 477, 479, 481, 483, and 485. The purpose of the side files is to allow each of the encrypted software products to be tagged with an encryption header without changing the file size.

Encryption type segment 459 of the encryption header 451 identifies the type of encryption utilized to encrypt the encrypted software product. Any one of a number of conventional encryption techniques can be utilized to encrypt the product, and different encryption types can be utilized to encrypt different software products contained on the same memory media. Encryption type segment 459 ensures that the appropriate encryption/decryption routine is called so that the encrypted software product may be decrypted, provided the temporary access keys are valid and not expired. The name of key file segment 455 of encryption header 451 provides an address (typically a disk drive location) of the key file. As is stated above (in connection with Figure 18) the key file includes the product key, a customer key, and the clear machine ID. All three of these pieces of information are required in order to generate the real key (in accordance with Figure 20). Encrypted validation segment 457 includes the encrypted validation text which is utilized in the routine depicted in Figure 21 which generates a derived clear validation text which may be compared utilizing the routine of Figure 22 to the known clear validation text. Only if the derived clear validation text exactly matches the known clear validation text can the process continue by utilizing the derived and validated real key to decrypt the encrypted software product in accordance with the routine of Figure 23. However, prior to performing the decryption operations of Figure 23, the contents of the corresponding side file must be substituted back into the encrypted software product in lieu of encryption header 451. This ensures that the encrypted software product is complete prior to the commencement of decryption operations.

Each time a file is called for processing by the operating system of the user-controlled data processing system, the file management program which is resident in the operating system intercepts the input/output requests and examines the front portion of the file to determine if a decryption block identifier, such as unique identifier 453, exists at a particular known location. For best performance, as is depicted in Figure 24, this location will generally be at the beginning of the file. If the file management program determines that the file has the decryption block, the TSR will read the block into memory. The block is then parsed in order to build a fully qualified key file name by copying an environment variable that specifies the drive and directory containing the key files and concatenating the key file name from the encryption block. The TSR then attempts to open the key file. If the key file does not exist, the TSR returns an "access denied" response to the application which is attempting to open the encrypted file. If the key file is determined to exist, the TSR opens the key file and reads in the keys (the product key, the customer key, and the machine identification) and generates the real key. This real key is in use to decrypt the decryption block validation data. As is stated above, a comparison operation determines whether this decryption operation was successful. If the compare fails, the key file is determined to be "invalid", and the TSR returns an "access denied message" to the application which is attempting to open the encrypted software product. However, if the compare is successful, the file management program prepares to decrypt the file according to the encryption type found in the encryption header. The TSR then returns a valid file handle to the calling application to indicate that the file has been opened. When the application reads data from the encrypted file, the TSR reads and decrypts this data before passing it back to the application. If the data requested is part of the displaced data that is stored in the side file, the TSR will read the side

file and return the appropriate decrypted block to the calling application without the calling application being aware that the data came from a separate file.

While the broad concepts of the encryption header are depicted in Figures 24 and 25, the more particular aspects of creating the encrypted files are depicted in Figures 26, 27, and 28. Figures 27 and 28 depict two types of data files. Figure 27 depicts a non-executing data file, while Figure 28 depicts an executing data file. Figure 26 depicts a header 499 which includes signature segment 501, header LEN 503, side file index 505, side file LEN 507, decryption type identifier 509, verification data 511, and key file name 518. As is shown in Figure 27, a software product begins as a clear file 521, and is encrypted in accordance with a particular encryption routine into encrypted file 523. Encryption type segment 509 of header 499 identifies the type of encryption utilized to change clear file 521 to encrypted file 523. Next, the front portion of encrypted file 523 is copied to side file 527 which is identified by side file index 505 and side file LEN 507 of header 499. Additionally, a copy of the clear text of the verification data is also included in side file 527. Then, header 499 is copied to the front portion of encrypted file 523 to form modified encrypted files 525. A similar process is employed for executing files, as depicted in Figure 28. The clear text copy of the software product (represented as clear file 531) is encrypted in accordance with a conventional routine, to form encrypted file 533. The front portion of encrypted file 533 is copied to side file 539 so that the overlaid data of encrypted file 533 is preserved. Furthermore, side file 539 includes a copy of the clear text of the verification data. Then, the encrypted file 533 is modified by overlaying and executable stub 537 and header 599 onto the first portion of encrypted file 553.

The purpose of executable stub 537 of Figure 28 will now be described. The DOS operating system for a personal computer will try to execute an encrypted application. This can result in a system "hang" or unfavorable action. The executable stub 357 of the executing file of Figure 28 is utilized to protect the user from attempting to execute applications that are encrypted: there would be considerable risk that a user would hang his system or format a drive if he or she try to run an encrypted file. The executable stub is attached to the front portion of the encrypted software product so that this stub is executed whenever the application is run without the installed TSR or run from a drive the TSR is not "watching". This stub will post a message to the user that explains why the application cannot run. In addition to providing a message, this executable stub can be used to perform sophisticated actions, such as:

(1) it can duplicate the functionality of the TSR and install dynamic encryption before kicking off the application a second time;

(2) it can turn on a temporary access key and kick off the application a second time;

(3) it can communicate with the TSR and inform it to look at the drive the application is being run from.

The executable stub is saved or copied into the encrypted program as follows:

(1) the application is encrypted;

(2) a decryption block is created for this program;

(3) a pre-built executable stub is attached to the front end of the decryption block;

(4) the length of the combined decryption header and executable stub is determined;

(5) the bytes at the front of the executable file equal to this length are then read into memory, preferably into a predefined side file location; and

(6) the encryption header and executable stub are then written over the leading bytes in the executable code.

The TSR can determine if an executable is encrypted by searching beyond the "known size" of the executable stub for the decryption block portion. When the TSR decrypts the executable stub it accesses the side file to read in the bytes that were displaced by the stub and header block.

Figure 29 provides a flowchart representation of operation during a trial period interval, which begins at software block 601. In accordance with software block 603, the file management program located in the operating system of the user-controlled data processing system continually monitors for input/output calls to the memory media. Then, in accordance with software block 605, for each input/output call, the called file is intercepted, and in accordance with software block 607 the operating system is denied access to the called file, until the file management program can determine whether access should be allowed or not. A portion of the called file is read where the decryption block should be located. This portion of the called file is then read, in accordance with software block 609, to derive a key file address in accordance with software block 611. The address which is derived is utilized to fetch the key file, in accordance with software block 613. In accordance with decision block 615, if the key file cannot be located, the process ends at software block 617; however, if it is determined in decision block 615 that the key file can be located, the key is derived in accordance with software block 619. The derived key is then utilized to decrypt the validation segment which is located within the encryption header, in

accordance with software block 621. In decision block 623, the decryption validation segment is compared to the clear text for the decryption validation segment; if it is determined that the decrypted segment does not match the known clear text segment, the process continues at software block 625 by ending; however, if it is determined in decision block 623 that the decrypted validation segment does match the known clear text validation segment, the process continues as software block 627, wherein access to the called file is allowed. Then, the decryption type is read from the decryption header in accordance with software block 629, and the called file is dynamically decrypted in accordance with software block 631 as it is passed for processing by the operating system of the user-controlled data processing system, in accordance with software block 633. The process terminates at software block 635.

If unauthorized execution of an encrypted file is attempted, the executable stub will at least temporarily deny access and post a message to the system, but may handle the problem in a number of sophisticated ways which were enumerated above.

In accordance with the preferred embodiment of the present invention, during the trial interval, or at the conclusion of the trial interval, the prospective purchaser may contact the vendor to make arrangements for the purchase of a copy of the one or more software products on the computer-accessible memory media. Preferably, CD ROMs or floppy disks have been utilized to ship the product to the potential user. Preferably, the computer-accessible memory media includes the two encrypted copies of each of the products which are offered for a trial interval of use. One encrypted copy may be decrypted utilizing the file management program and the temporary key which is communicated from the vendor to the purchaser. The other encrypted copy is not provided for use in the trial interval mode of operation, but instead is provided as the permanent copy which may be decrypted and utilized once the software product has been purchased. In broad overview, the user selects a software product for a trial interval mode of operation, and obtains from the vendor temporary access keys, which allow the user access to the product (through the file management program) for a predefined trial interval. Before or after the conclusion of the trial interval, the user may purchase a permanent copy of the software product from the vendor by contacting the vendor by facsimile, electronic mail, or telephone. Once payment is received, the vendor communicates to the user a permanent access key which is utilized to decrypt the second encrypted copy of the software product. This encrypted product may be encrypted utilizing any conventional encryption routine, such as the DES algorithm. The permanent key allows the software product to be decrypted for unrestricted use. Since multiple copies of the product may be purchased in one transaction, the present invention is equipped with a technique for providing movable access keys, which will be discussed below in connection with Figures 30 through 35. In the preferred embodiment of the present invention, the encryption algorithm employed to encrypt and decrypt the second copy of the software product is similar to that employed in the trial interval mode of operation.

The present invention includes an export/import function which allows for the distribution of permanent access keys, after the conclusion of a trial interval period. Typically, an office administrator or data processing system manager will purchase a selected number of "copies" of the encrypted product after termination of a trial interval period. Certain individuals within the organization will then be issued permanent keys which allow for the unrestricted and permanent access to the encrypted product. In an office or work environment where the computing devices are not connected in a distributed data processing network, the permanent access keys must be communicated from the office administrator or data processing manager to the selected individuals within an organization who are going to receive copies of the encrypted software product. The permanent keys allow for permanent access to the product. Since not all employees within an organization may be issued copies of the particular encrypted product, the vendor would like to have the distribution occur in a manner which minimizes or prevents the distribution beyond the sales agreement or license agreement. Since the products are encrypted, they may be liberally distributed in their encrypted form. It is the keys which allow unrestricted access to the product which are to be protected in the current invention. To prevent the distribution of keys on electronic mail or printed communications, the present invention includes an export program which is resident in a source computer and an import program which is resident in a target computer which allow for the distribution of the access keys via a removable memory media, such as a floppy diskette. This ensures that the access keys are not subject to inadvertent or accidental distribution or disclosure. There are two principal embodiments which accomplish this goal.

In the first embodiment, one or more encrypted files which are maintained in the source computer are first decrypted, and then encrypted utilizing an encryption algorithm and an encryption key which is unique to the transportable memory media (such as a diskette serial number). The key file may then

be physically carried via the diskette to a target computer, where it is decrypted utilizing a key which is derived by the target computer from interaction with the transferable memory media. Immediately, the key file or files are then encrypted utilizing an encryption operation which is keyed with a key which is derived from a unique system attribute of the target computer.

In the alternative embodiment, the transferrable memory media is loaded onto the target computer to obtain from the target computer import file a transfer key which is uniquely associated with the target computer, and which may be derived from one or more unique system attributes of the target computer. The memory media is then transferred to the source computer, where the one or more key files are decrypted, and then encrypted utilizing the transfer key. The memory media is then carried to the target computer where the transfer key is generated and utilized in a decryption operation to decrypt the one or more key files. Preferably, immediately the key files are encrypted utilizing an encryption operation which is keyed with a key which is uniquely associated with the target computer, and which may be derived from one or more unique computer configuration attributes. The first embodiment is discussed herein in connection with Figures 30, 31, 32, and 33. The second embodiment is discussed in connection with Figures 34 and 35.

Figures 30 and 31 depict in block diagram form export and import operations which allow an authorized user to move his permanent key to another data processing system using an "export" facility that produces a unique diskette image of the access key that has been enabled for import into another system. In accordance with the present invention, the access keys which are delivered over the telephone by the software vendor to the customer are less than 40 bytes in length. The key file that is produced is over 2,000 bytes in length. An export facility is provided for copying the key file and the machine identification file to a diskette. Both files are then encrypted with a modified diskette serial number to inhibit these files from being copied to a public forum where anyone could use them. An import facility provided in another system decrypts these files and adds the product key and machine identification from the diskette to a list of import product keys and machine identifications in the import systems master file, and copies the key file to the import system hard disk. The key file is encrypted on the import system as is disclosed above.

Figure 30 is a block diagram depiction of an export operation in accordance with the preferred embodiment of the present invention. As is shown, source computer 651 includes a key file 653 and a machine identification file 655. Key file 653 includes the product key, the customer key, the clear text of the machine identification for source computer 653, trial interval data (such as a clock and/or counter which define the trial interval period), and an export counter which performs the dual functions of defining the maximum number of export operations allowed for the particular protected software products and keeping track of the total number of export operations which have been accomplished. The machine identification file includes the machine identification number and trial interval data (such as a clock and/or counter which defines the trial interval period). Both key file 653 and machine identification file 655 are encrypted with any conventional encryption operation (such as the DES algorithm), which is keyed with a key which is derived from a unique system attribute of source computer 651. At the commencement of an export operation, key file 653 and machine identification file 655 are decrypted. Key file 653 is supplied as an input to decryption operation 657 which is keyed with key 659. Likewise, machine identification file 655 is supplied as an input to decryption operation 663 which is keyed with key 661. Decryption operations 657, 663 generate a clear text version of key file 653 and machine identification file 655. Once the clear text is obtained, the export counter which is contained within key file 653 is modified in accordance with block 661. For example, if this is the seventh permitted export operation out of ten permissible operations, the counter might read "7:10". The clear text version of key file 653 is supplied as an input to encryption operation 669. Encryption operation 669 may be any conventional encryption operation (such as the DES algorithm), which is keyed with a memory media attribute 665 which is unique to a memory media which is coupled to source computer 651, which has been subjected to modification of modifier 667. For example, a unique diskette serial number may be supplied as the "memory media attribute" which is unique to memory media 677. The diskette serial number is modified in accordance with modifier 667 to alter it slightly, and supply it as an input to encryption operations 669. The same operation is performed for the clear text of machine identification file 655. A unique memory media attribute 671 is modified by modifier 673 and utilized as a key for encryption operation 675, which may comprise any conventional encryption operation, such as the DES operation. Finally, the output of encryption operations 669 and 675 are supplied as inputs to copy operations 679, 681 which copy the encrypted key file 653 and machine identification file 655 to memory media 677.

Figure 31 is a block diagram depiction of an import operation. Memory media 677 (of Figure 30)

is physically removed from source computer 651 (of Figure 30) and physically carried over to computer 707 (of Figure 31); alternatively, in a distributed data processing system, this transfer may occur without the physical removal of memory media 677. With reference now to Figure 31, in accordance with block 683, the machine identification of the target machine is copied to memory media 677 to maintain a record of which particular target computer received the key file and machine identification file. Then, in accordance with blocks 685, 693 the encrypted key file 653 and machine identification file 655 are copied from the memory media to target computer 707. The encrypted key file 653 is supplied as an input to decryption operation 689 which is keyed with key 687. Decryption operation 689 reverses the encryption operation of block 669, and provides as an output a clear text version of key file 653. Likewise, machine identification file 655 is supplied as an input to decryption operation 697, which is keyed with key 695. Decryption operation 697 reverses the encryption of encryption operation 675 and provides as an output the clear text of machine identification file 655. In accordance with block 691, the machine identification of the source computer 651 is retrieved and recorded in memory in the clear text of key file 653. Next, the clear text of key file 653 is supplied as an input to encryption operation 699. Encryption operation 699 is a conventional encryption operation, such as the DES operation, which is keyed with a target computer unique attribute, such as the machine identification or modified machine identification for the target computer 707. The clear text of machine identification file 655 is supplied as an input to encryption operation 703. Encryption operation 703 is any conventional encryption operation, such as the DES encryption operation, which is keyed with a unique target computer attribute 705, such as machine identification or modified machine identification of target computer 707. The output of encryption operation 699 produces an encrypted key file 709 which includes a product key (which is the same temporary product key of key file 653 of source computer 651), a customer number (which is the same customer number of key file 653 of source computer 651), and clear machine identification (which is the machine identification for target computer 707, and not that of source computer 651), trial interval data (which is identical to the trail interval data of key file 653 of source 651), and an identification of the machine identification of the source computer 651. The output of encryption operation 703 defines machine identification file 711, which includes the machine identification of the target computer 707 (and not that of the source computer 651), and the trial interval data (which is identical to that of machine identification file 655 of source computer 651).

Figures 32 and 33 provide alternative views of the import and export operations which are depicted in Figures 30 and 31, and emphasize several of the important features of the present invention. As is shown, source computer 801 includes machine identification file 803 which is encrypted with a system attribute key which is unique to the source computer 801. The machine identification file includes machine identification file number as well as count of the number of exports allowed for each protected software product, and a count of the total number of exports which have been utilized. For example, the first export operation carries a count of "1:10", which signifies that one export operation of ten permitted export operations has occurred. In the next export operation, the counter is incremented to "2:20" which signifies that two of the total number of ten permitted export operations has occurred. Each target computer which receives the results of the export operation is tagged with this particular counter value, to identify that it is the recipient of a particular export operation. For example, one source computer system may carry a counter value of "1:10", which signifies that it is the recipient of the first export operation of ten permitted export operations. Yet another target computer may carry the counter value of "7:10", which signifies that this particular target computer received the seventh export operation of a total of ten permitted export operations. In this fashion, the target computer maintains a count of a total number of used export operations, while the source computers each carry a different counter value which identifies it a the recipient of the machine identification file and key file from the source computer from particular ones of the plurality of permitted export operations.

Note that in source computer 801 machine identification file 803 and key file 805 are encrypted with an encryption algorithm which utilizes as a key a system attribute which is unique to source computer 801; however, once machine identification file 803 and key file 805 are transferred to a memory media, such as export key diskette 807, machine identification file 809 and key file 811 are encrypted in any conventional encryption operation which utilizes as an encryption key a unique diskette attribute, such as the diskette's serial number. This minimizes the possibility that the content of the machine ID file 809 and/or key file 811 can be copied to another diskette or other memory media and then utilized to obtain unauthorized access to the software products. This is so because for an effective transfer of the content of machine ID file 809 and key file 811 to a target computer to occur, the target computer must be able to read and utilize the unique diskette attribute from the export

key diskette 807. Only when the machine ID file 809 and key file 811 are presented to a target computer on the diskette onto which these items were copied can an effective transfer occur. The presentation of the machine ID file 809 and key file 811 on a diskette other than export key diskette 807 to a potential target computer will result in the transfer of meaningless information, since the unique attribute of export key diskette 807 (such as the diskette serial number) is required by the target computer in order to successfully accomplish the decryption operation.

As is shown in Figure 33, export key diskette 807 is presented to target computer 813. Of course, the machine identification file 809 and key file 811 are in encrypted form. In the transfer from export key diskette 807 to target computer 813, the content of machine ID file 809 is updated with the machine identification of the target computer 813, and the count of imports utilized. In accomplishing the transfer to target computer 813, a machine identification file 815 is constructed which includes a number of items such as machine identification for the target computer 813, customer information, as well as a list of the machine identification number of the source computer 801. Both machine identification file 815 and the key file 817 are encrypted utilizing a conventional encryption operation which uses as a key a unique attribute of target computer 813. This ties machine identification file 815 and key file 817 to the particular target computer 813.

By using an export/import counter to keep track of the total number of authorized export/import operations, and the total number of used export/import operations, the present invention creates an audit trail which can be utilized to keep track of the distribution of software products during the trial interval. Each source computer will carry a record of the total number of export operations which have been performed. Each source computer will carry a record of which particular export/import operation was utilized to transfer one or more protected software products to the target computer. The memory media utilized to accomplish the transfer (such as a diskette, or group of diskettes) will carry a permanent record of the machine identification numbers of both the source computer and the target computer's utilized in all export/import operations.

The procedure for implementing export and import operations ensures that the protected software products are never exposed to unnecessary risks. When the machine identification file and key file are passed from the source computer to the export diskette, they are encrypted with the unique attribute of the export diskette which prevents or inhibits copying of the export diskette or posting of its contents to a bulletin board as a means for illegally distributing the keys. During the import operations, the machine identification and key files are encrypted with system attributes which are unique to the target computer to ensure that the software products are maintained in a manner which is consistent with the security of the source computer, except that those software products are encrypted with attributes which are unique to the target computer, thus preventing illegal copying and posting of the keys.

The second embodiment of the export/import function is depicted in block diagram form in Figures 34 and 35. In broad overview, memory media 1677 is first utilized to interact with target computer 1707 to obtain from target computer 1707 a transfer key which is unique to target computer 1707, and which is preferably derived from one or more unique system attributes of target computer 1707. The transfer key may he a modification of the machine identification for target computer 1707. Next, the memory media 1677 is utilized to interact with source computer 1651 in an export mode of operation, wherein key file 1653 and machine identification file 1655 are first decrypted, and then encrypted utilizing the transfer key.

Figure 34 is a block diagram depiction of an export operation in accordance with the preferred embodiment of the present invention. As is shown, source computer 1651 includes a key file 1653 and a machine identification file 1655. Key file 1653 includes the product key, the customer key, the clear text of the machine identification for source computer 1653, trial interval data (such as a clock and/or counter which define the trial interval period), and an export counter which performs the dual functions of defining the maximum number of export operations allowed for the particular protected software products and keeping track of the total number of export operations which have been accomplished. The machine identification file includes the machine identification number and trial interval data (such as a clock and/or counter which defines the trial interval period). Both key file 1653 and machine identification file 1655 are encrypted with any conventional encryption operation (such as the DES algorithm), which is keyed with a key which is derived from a unique system attribute of source computer 1651. At the commencement of an export operation, key file 1653 and machine identification file 1655 are decrypted. Key file 1653 is supplied as an input to decryption operation 1657 which is keyed with key 1659. Likewise, machine identification file 1655 is supplied as an input to decryption operation 1663 which is keyed with key 1661. Decryption operations 1657, 1663 generate a clear text version of key file 1653 and machine identification file 1655. Once the clear text

is obtained, the export counter which is contained within key file 1653 is modified in accordance with block 1661. For example, if this is the seventh permitted export operation out of ten permissible operations, the counter might read "7:10". The clear text version of key file 1653 is supplied as an input to encryption operation 1669. Encryption operation 1669 may be any conventional encryption operation (such as the DES algorithm), which is keyed with the transfer key 1665 which was previously obtained. The same operation is performed for the clear text of machine identification file 1655. Transfer key 1671 is utilized as a key for encryption operation 1675, which may comprise any conventional encryption operation, such as the DES operation. Finally, the output of encryption operations 1669 and 1675 are supplied as inputs to copy operations 1679, 1681 which copy the encrypted key file 1653 and machine identification file 1655 to memory media 1677.

Figure 35 is a block diagram depiction of an import operation. Memory media 1677 (of Figure 34) is physically removed from source computer 1651 (of Figure 34) and physically carried over to computer 1707 (of Figure 35); alternatively, in a distributed data processing system, this transfer may occur without the physical removal of memory media 1677. With reference now to Figure 35, in accordance with block 1683, the machine identification of the target machine is copied to memory media 1677 to maintain a record of which particular target computer received the key file and machine identification file. Then, in accordance with blocks 1685, 1693 the encrypted key file 1653 and machine identification file 1655 are copied from the memory media to target computer 1707. The encrypted key file 1653 is supplied as an input to decryption operation 1689 which is keyed with key 1687. Decryption operation 1689 reverses the encryption operation of block 1669, and provides as an output a clear text version of key file 1653. Likewise, machine identification file 1655 is supplied as an input to decryption operation 1697, which is keyed with key 1695. Decryption operation 1697 reverses the encryption of encryption operation 1675 and provides as an output the clear text of machine identification file 1655. In accordance with block 1691, the machine identification of the source computer 1651 is retrieved and recorded in memory in the clear text of key file 1653. Next, the clear text of key file 1653 is supplied as an input to encryption operation 1699. Encryption operation 1699 is a conventional encryption operation, such as the DES operation, which is keyed with a target computer unique attribute, such as the machine identification or modified machine identification for the target computer 1707. The clear text of machine identification file 1655 is supplied as an input to encryption operation 1703. Encryption operation 1703 is any conventional encryption operation, such as the DES encryption operation, which is keyed with a unique target computer attribute 1705, such as machine identification or modified machine identification of target computer 1707. The output of encryption operation 1699 produces an encrypted key file 1709 which includes a product key (which is the same temporary product key of key file 1653 of source computer 1651), a customer number (which is the same customer number of key file 1653 of source computer 1651), and clear machine identification (which is the machine identification for target computer 1707, and not that of source computer 1651), trial interval data (which is identical to the trail interval data of key file 1653 of source 1651), and an identification of the machine identification of the source computer 1651. The output of encryption operation 1703 defines machine identification file 1711, which includes the machine identification of the target computer 1707 (and not that of the source computer 1651), and the trial interval data (which is identical to that of machine identification file 1655 of source computer 1651).

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

Claims

1. A method of passing encrypted files between data processing systems, comprising:
   at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
   providing a transfer memory medium;
   at said source computer, decrypting said at least one file;
   at said source computer, encrypting said at least one file with a key which is derived from at least one unique transfer memory media attribute;
   at said source computer, copying said encrypted file to said transfer memory media;
   at a target computer, decrypting said at least one file;
   at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

2. A method of passing encrypted files between data processing systems, comprising:

at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
providing a transfer memory medium;
at a target computer copying a transfer encryption key which is unique to said target computer to said transfer memory media;
at said source computer, decrypting said at least one file;
at said source computer, encrypting said at least one file with said transfer encryption key;
at said source computer, copying said encrypted file to said transfer memory media;
at a target computer, decrypting said at least one file;
at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

3. A method of passing encrypted files according to Claims 1 or 2, further comprising:
providing a export counter in said source computer which defines a maximum number of permissible transfer operations; and
actuating said export counter for each transfer operation.

4. A method of passing encrypted files according to one of Claims 1 to 3, further comprising:
identifying each one of said permissible transfer operations to a particular target computer.

5. A method of passing encrypted files according to one of Claims 1 to 4, further comprising:
recording the occurrence of all transfer operations involving said transfer memory medium by obtaining identifying information from each target computer.

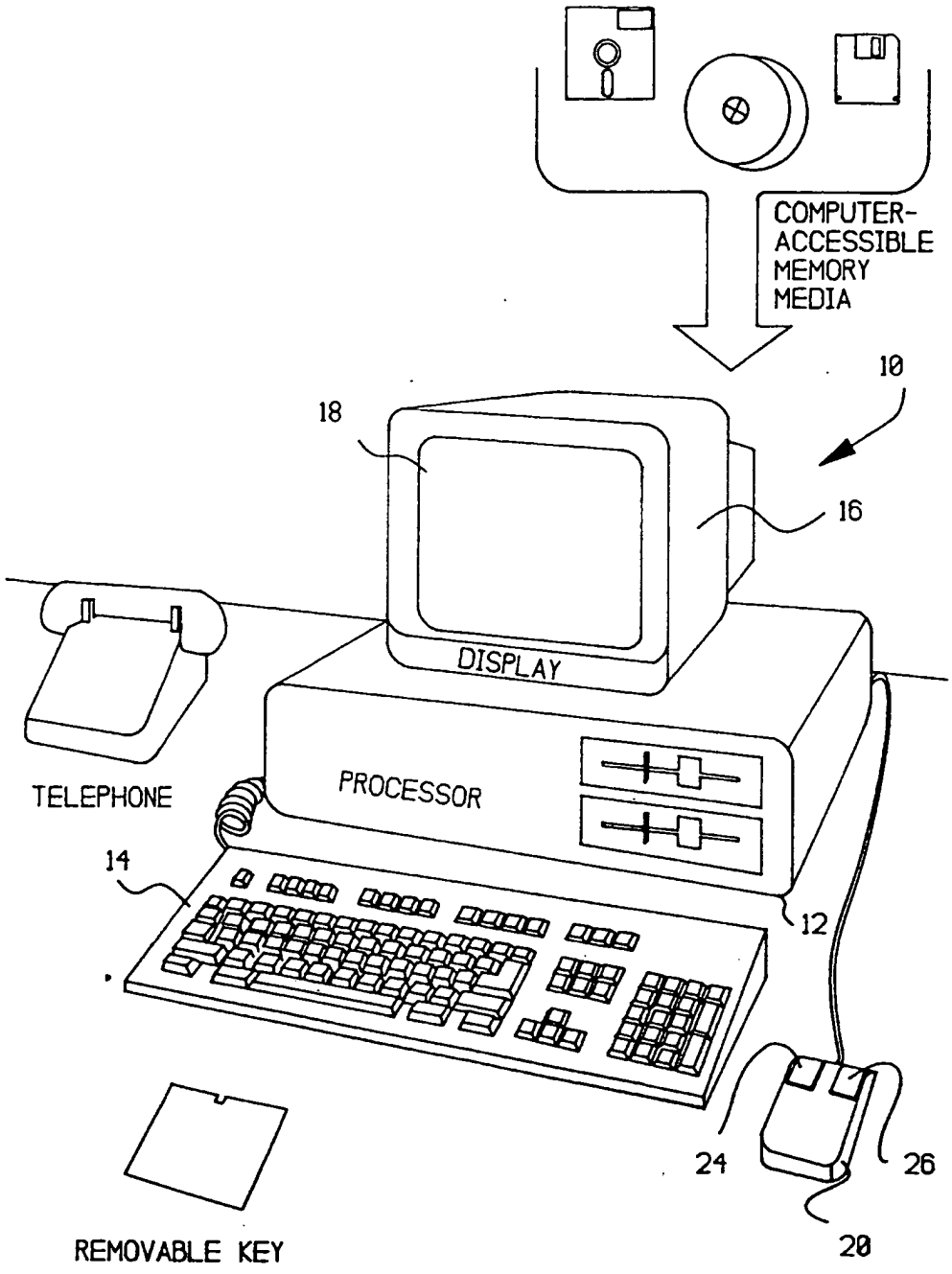
6. A method of passing encrypted files between data processing systems, comprising:
at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
providing a transfer memory medium;
initiating a particular transfer operation;
at said source computer, decrypting said at least one file;
including in said at least one file a transfer identifier which uniquely identifies said particular transfer operation;
at said source computer, encrypting said at least one file with a key which is derived from at least one unique transfer memory media attribute;
at said source computer, copying said encrypted file to said transfer memory media;
at a target computer, decrypting said at least one file;
at said target computer, encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

7. A method of passing encrypted files according to Claim 6, further comprising:
at said target computer, passing a unique target computer identification to said transfer memory media.

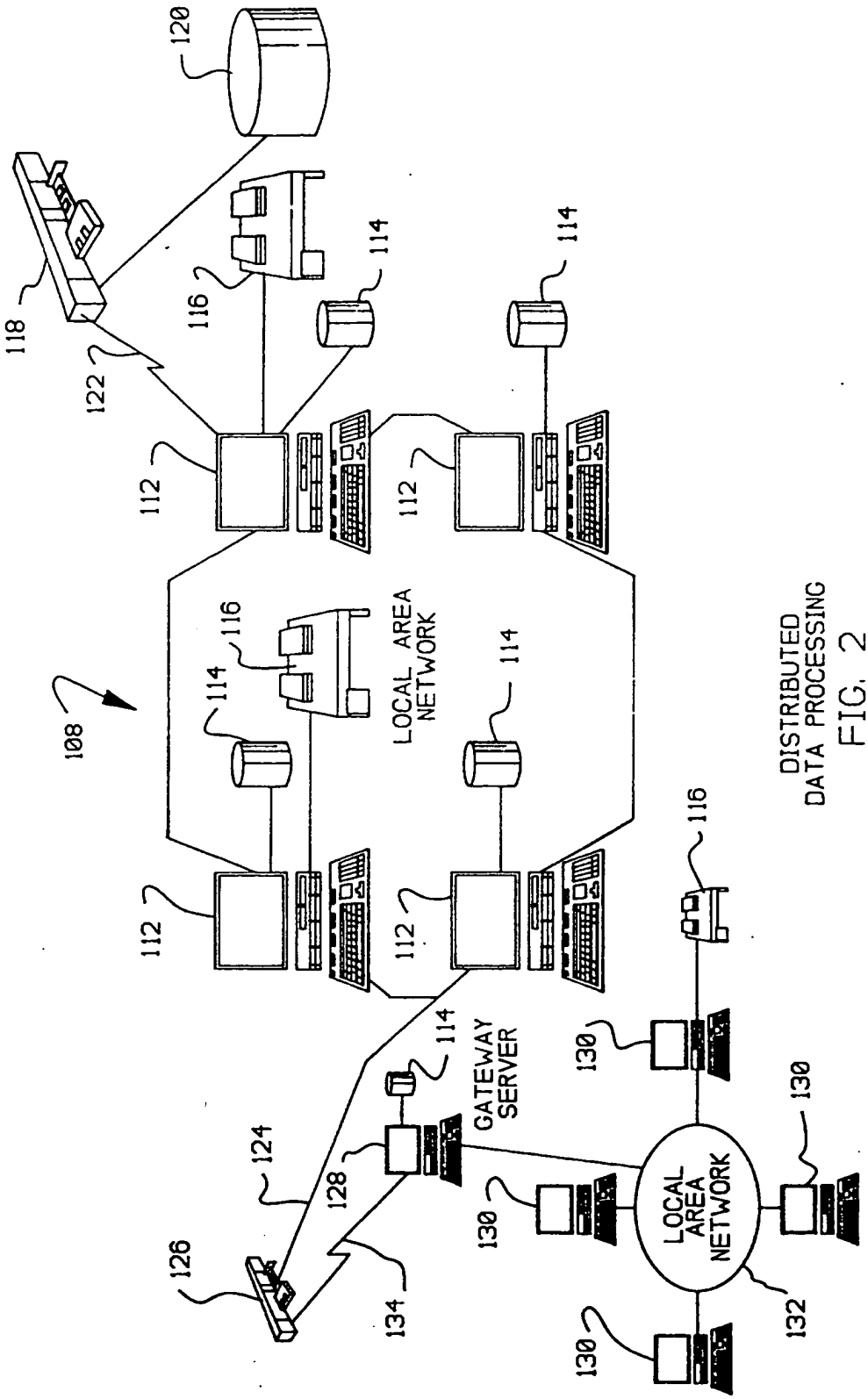
8. A method of passing encrypted files according to Claim 6 or 7, further comprising:
at said target computer, updating said at least one file to provide an identification of said source computer.

9. An apparatus passing encrypted files between data processing systems, comprising:
at least one file in a source computer which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
a removable transfer memory medium having a unique attribute;
an export program for decrypting said at least one file and encrypting said at least one file with a key which is derived from said unique attribute and copying said encrypted file to said transfer memory media;
an import program at a target computer for decrypting said at least one file, and encrypting said at least one file with a key which is at least partially derived from at least one target computer system attribute.

10. An apparatus for passing encrypted files according to Claim 9, further comprising:
a export counter in said export program in said source computer which defines a maximum number of permissible transfer operations, and for counting each transfer operation.



STAND ALONE PC
FIG. 1



DISTRIBUTED DATA PROCESSING
FIG. 2

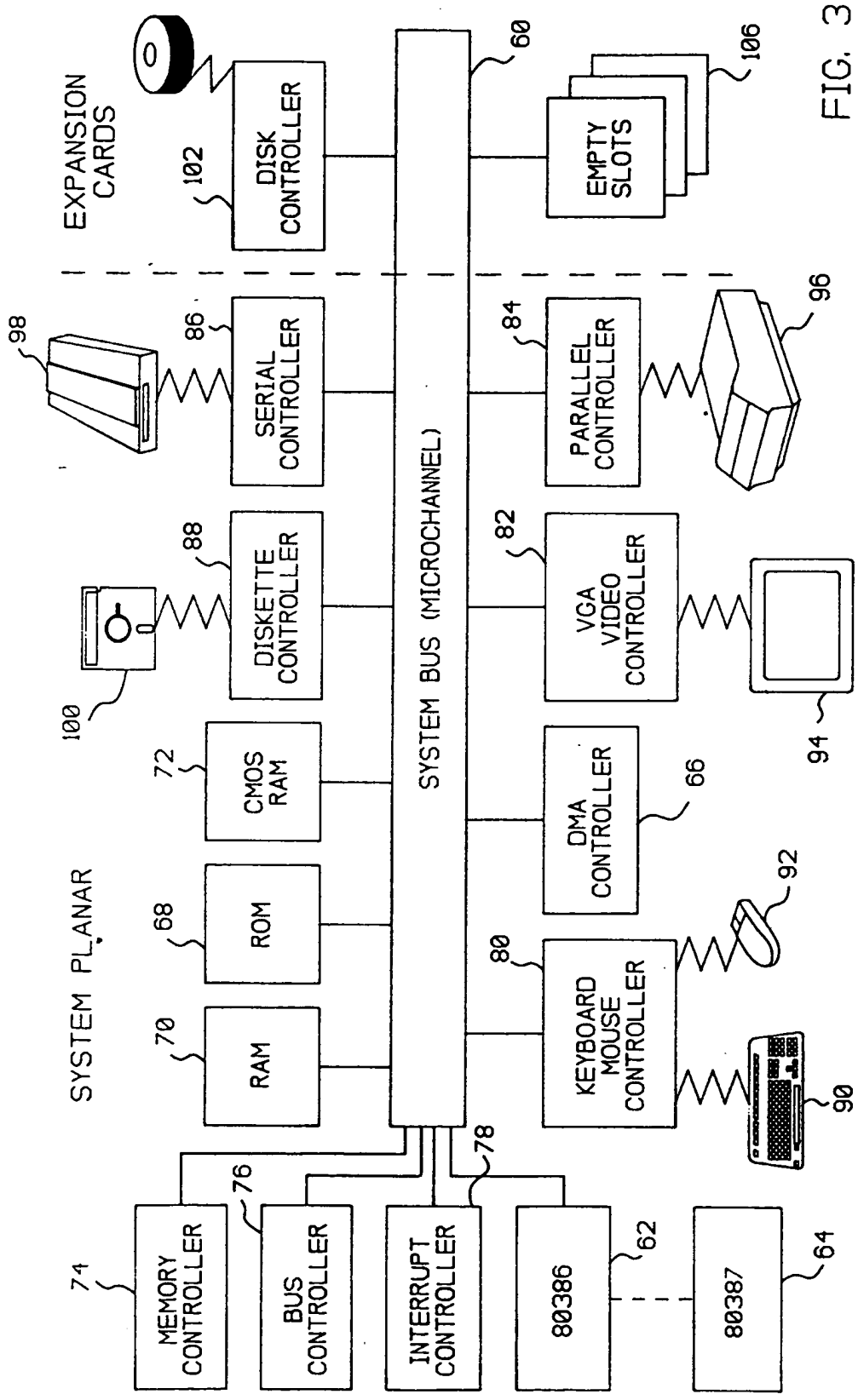


FIG. 3

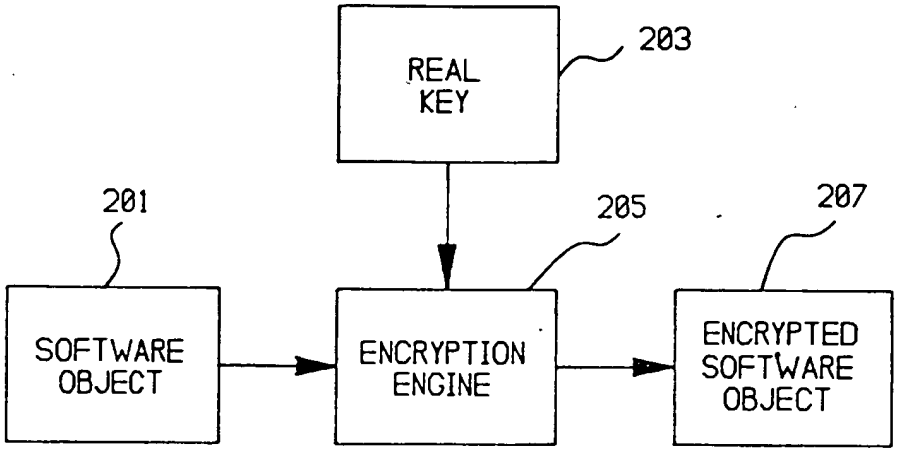


FIG. 4

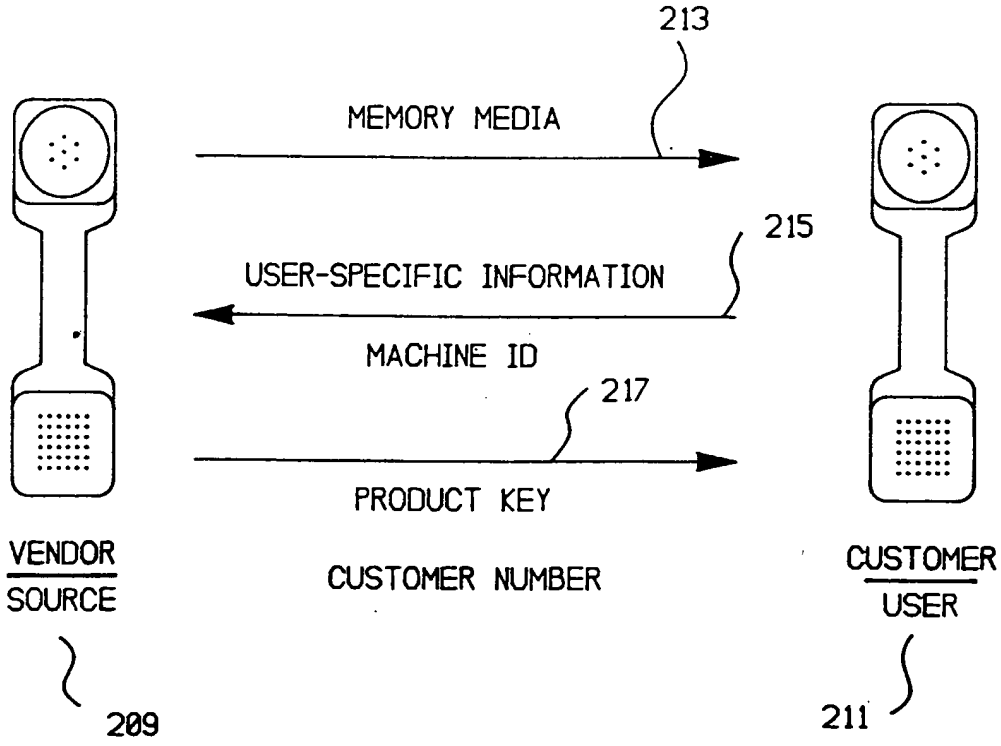
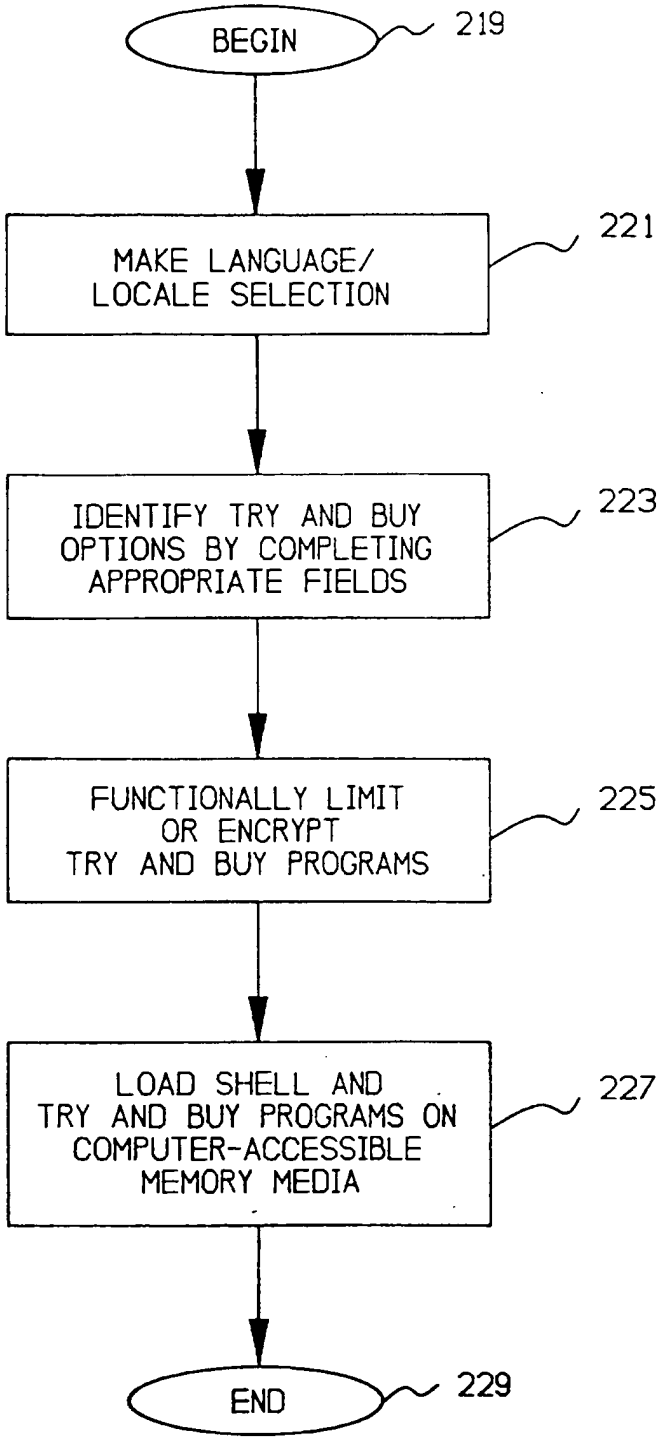


FIG. 5



BUILDING THE SHELL

FIG. 6

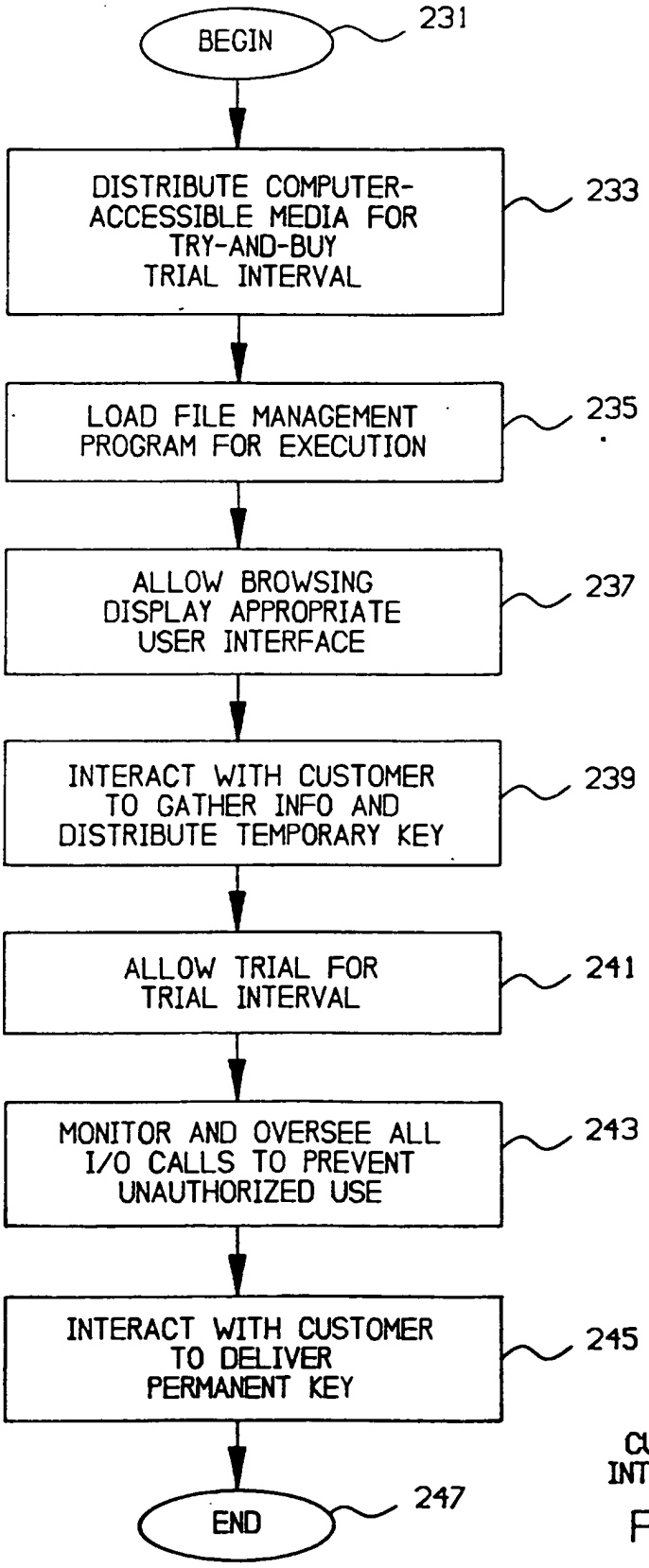


FIG. 7

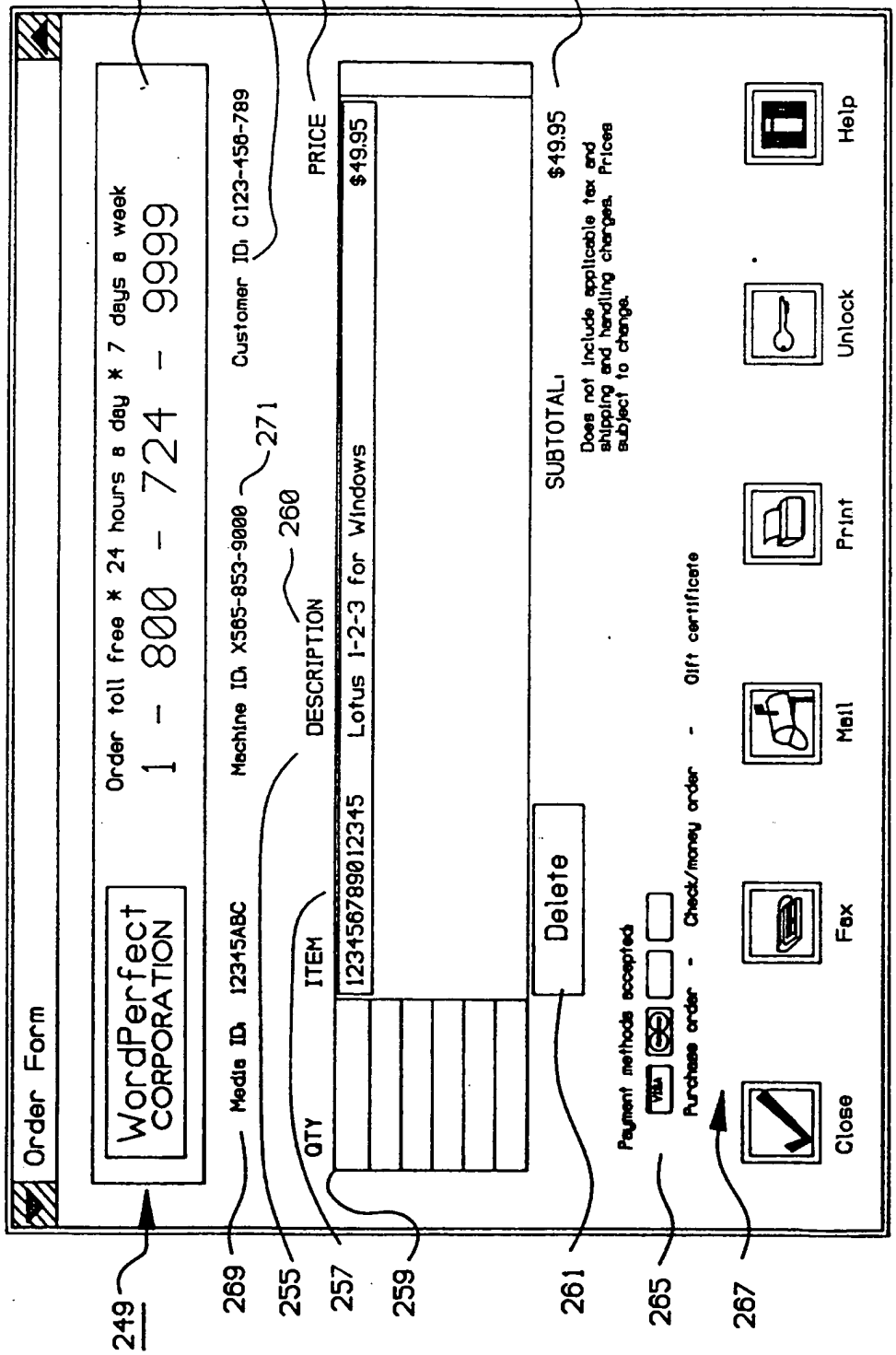


FIG. 8

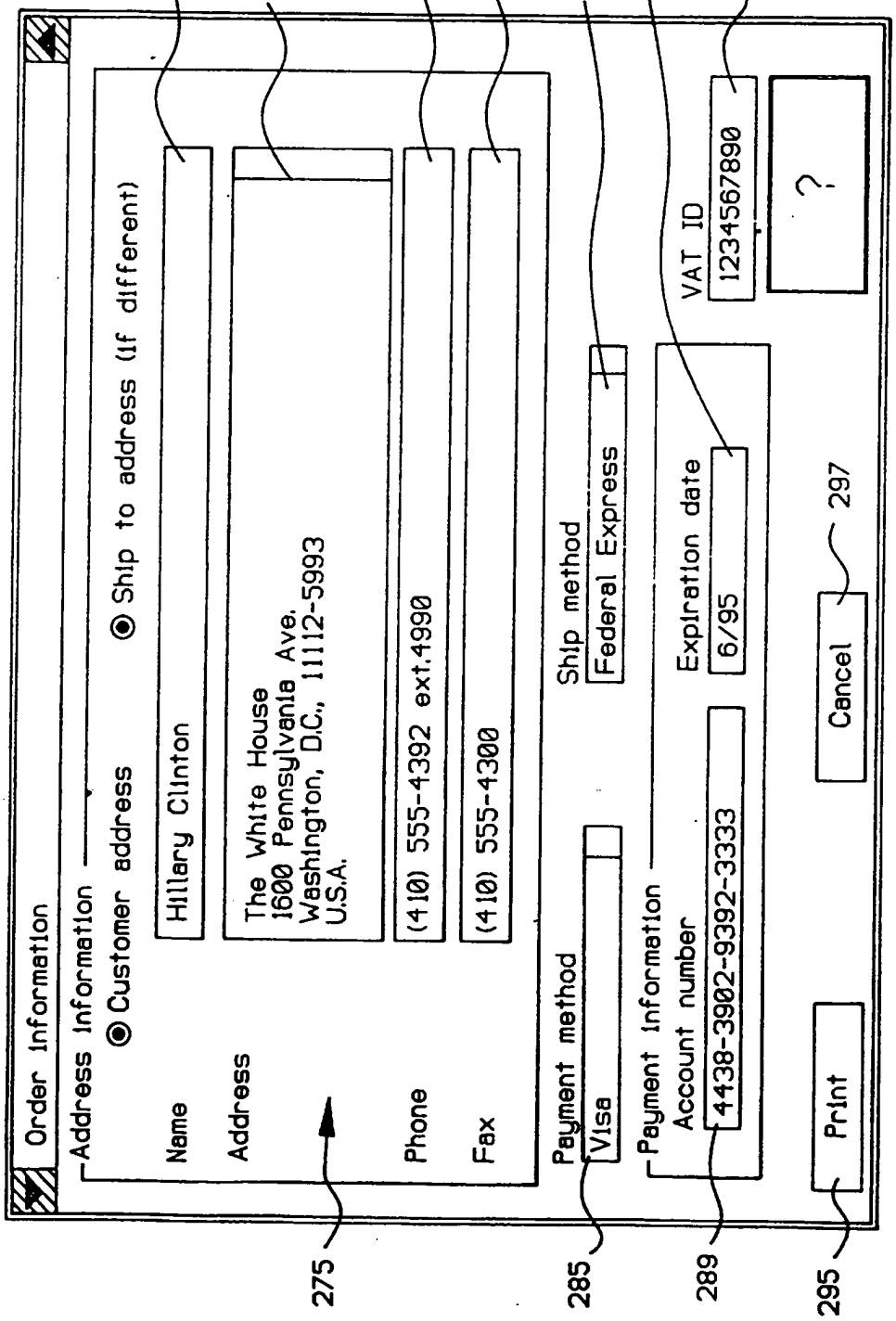


FIG. 9

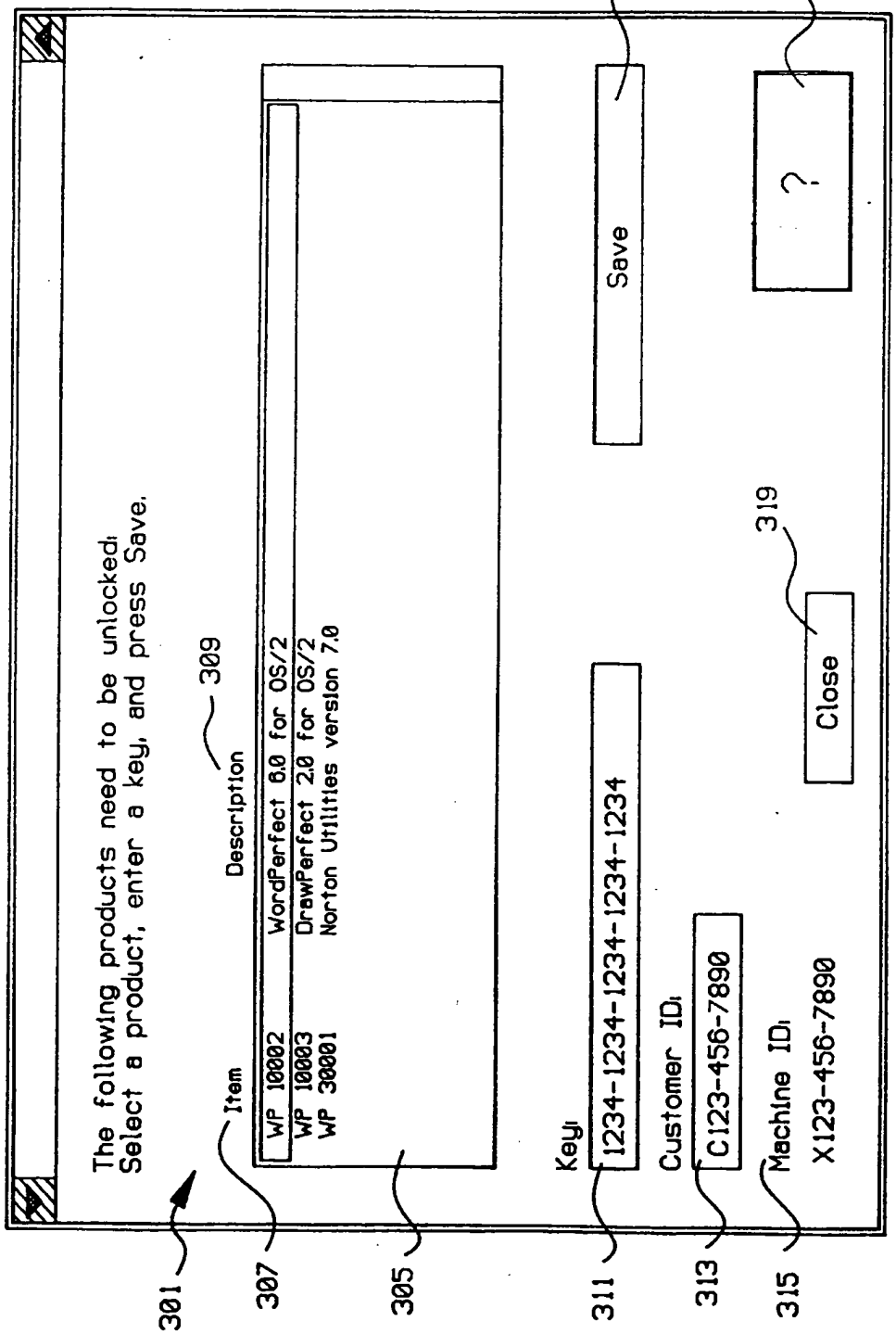


FIG. 10A

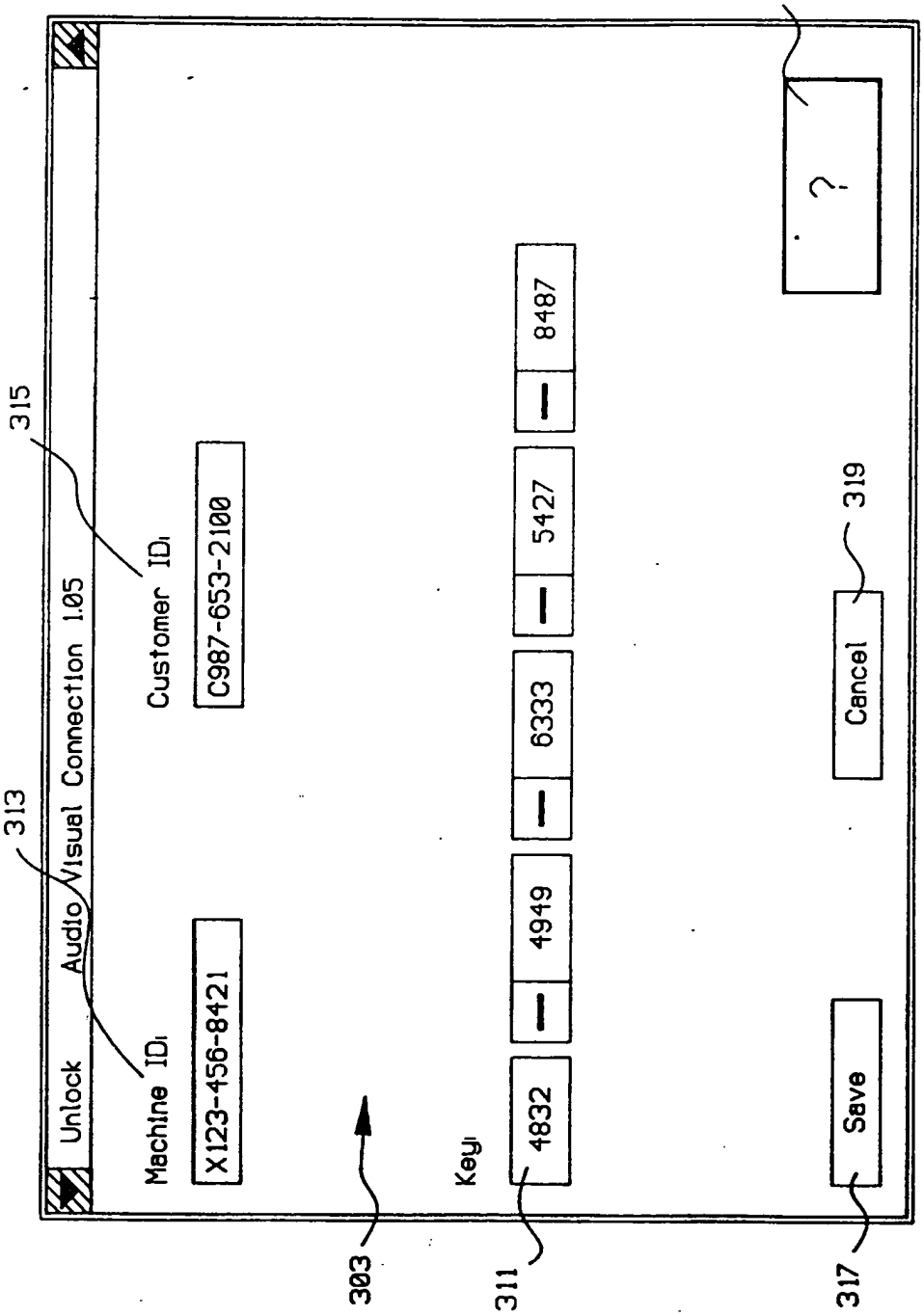


FIG. 10B

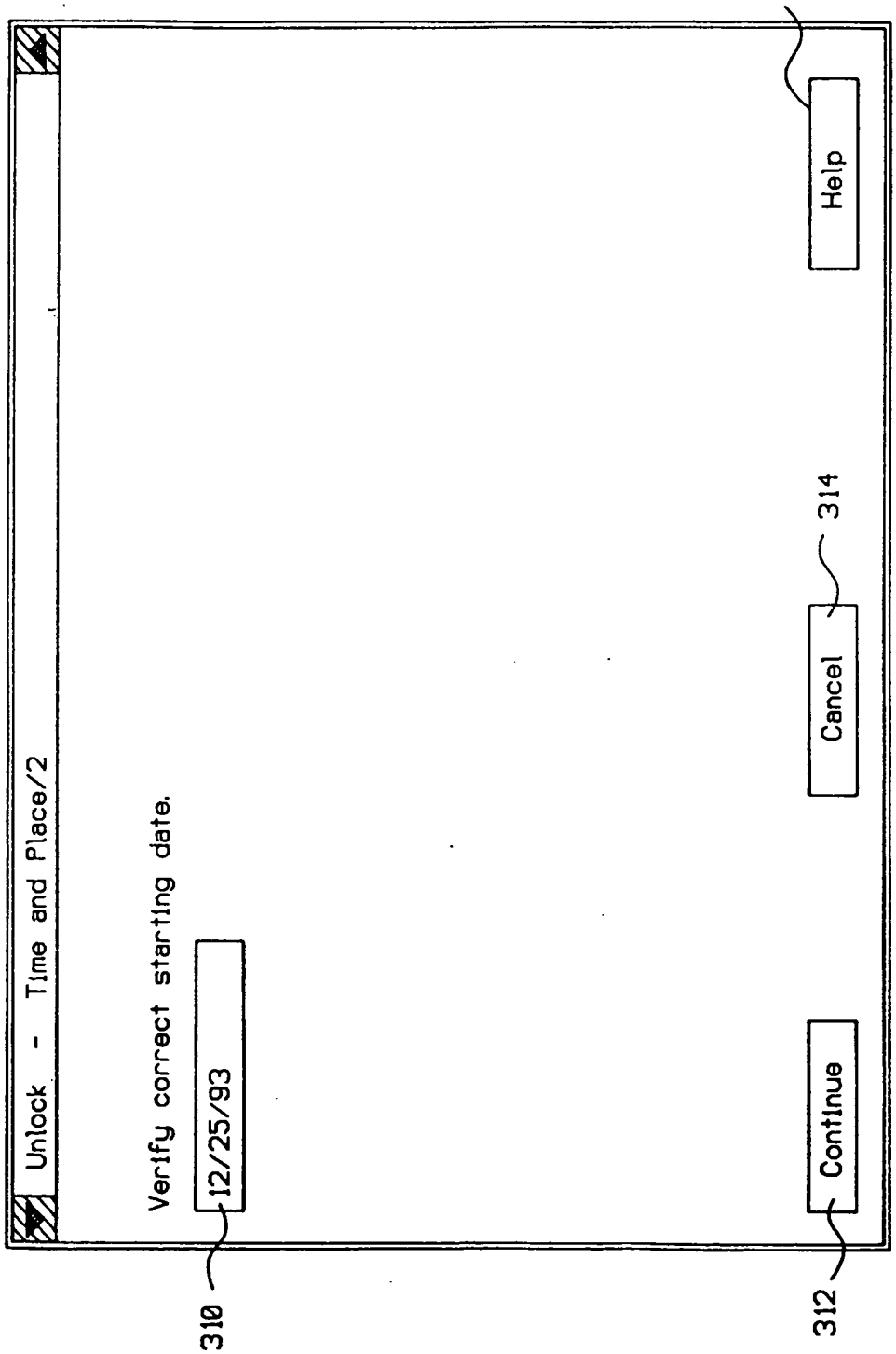


FIG. 11

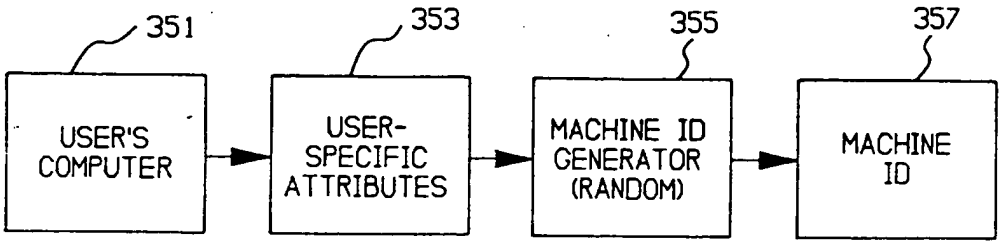
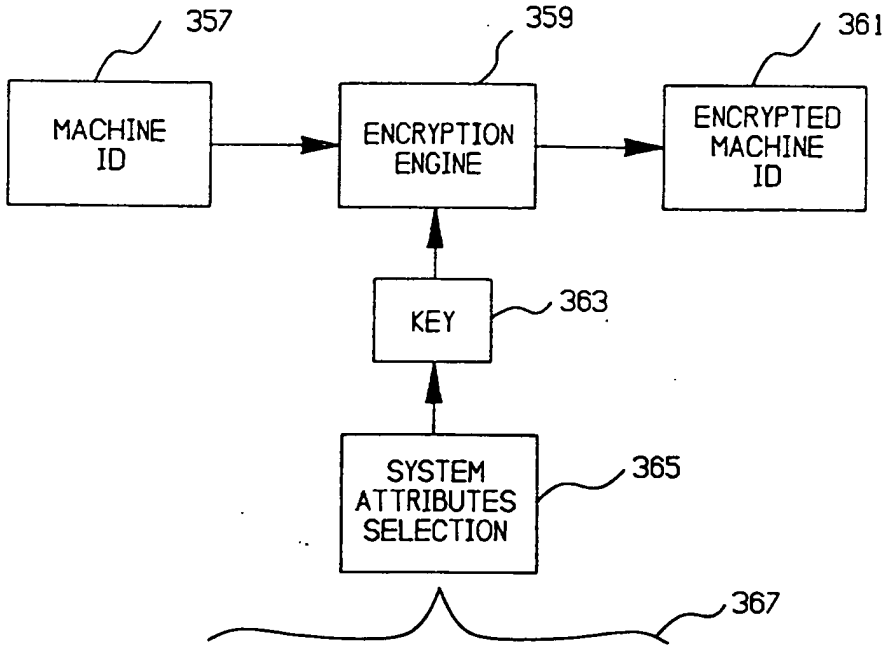
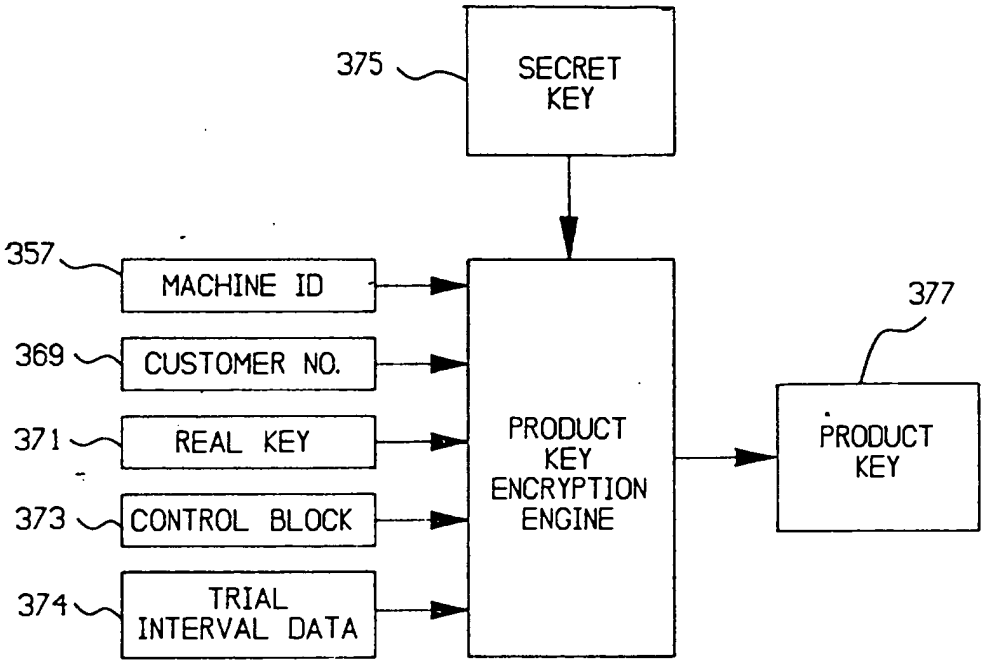


FIG. 12

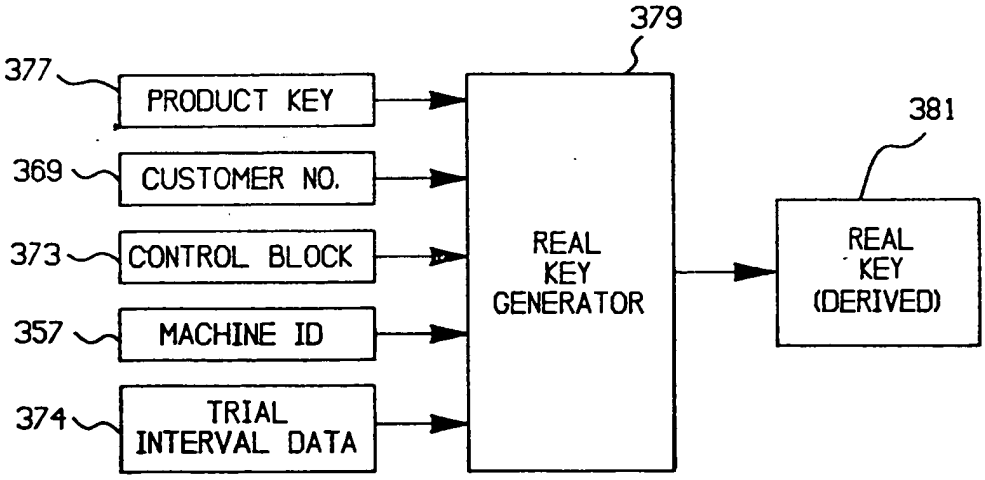


1. HARD DISK SERIAL NO.
2. SIZE OF HARD DISK
3. FORMAT OF HARD DISK
4. SYSTEM MODEL NO.
5. HARDWARE INTERFACE CARD
6. HARDWARE SERIAL NO.
7. CONFIGURATION PARAMETERS

FIG. 13



GENERATION OF PRODUCT KEY
FIG. 14



VALIDATION OF PRODUCT KEY
FIG. 15

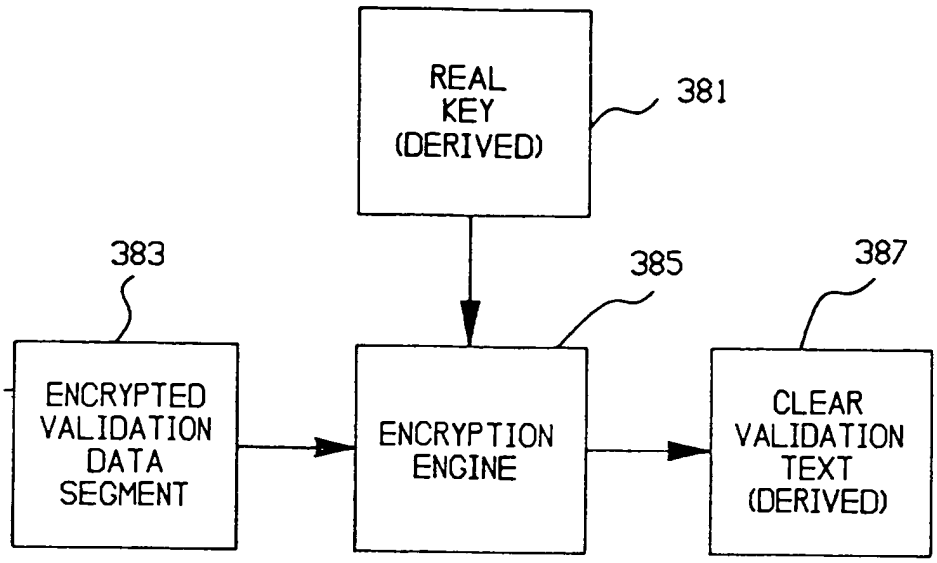


FIG. 16

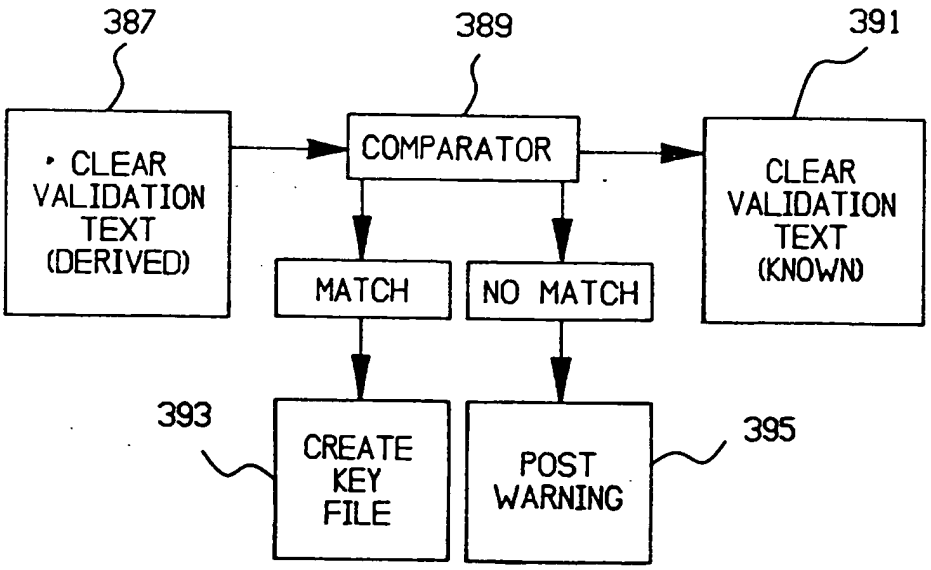


FIG. 17

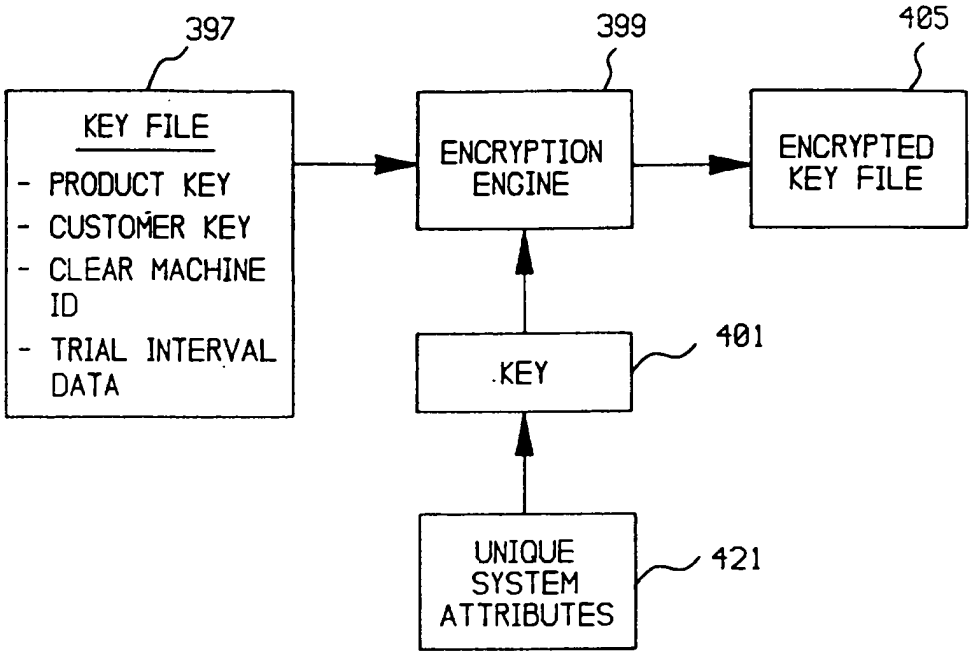


FIG. 18

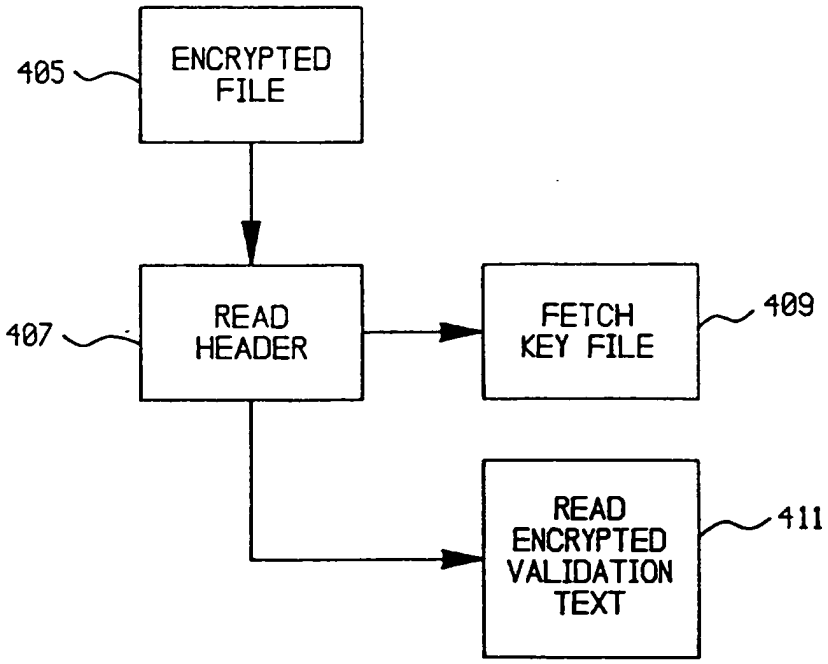


FIG. 19

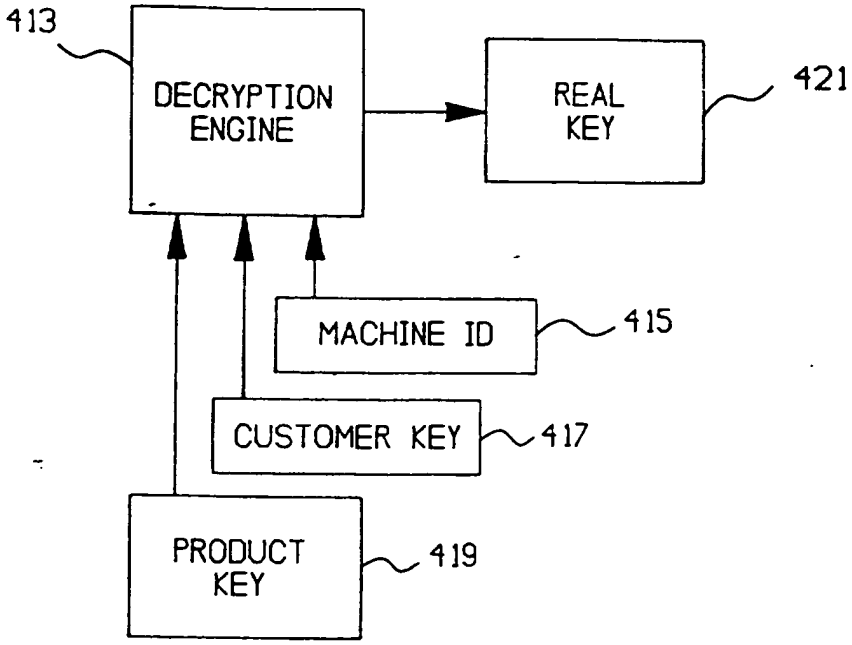


FIG. 20

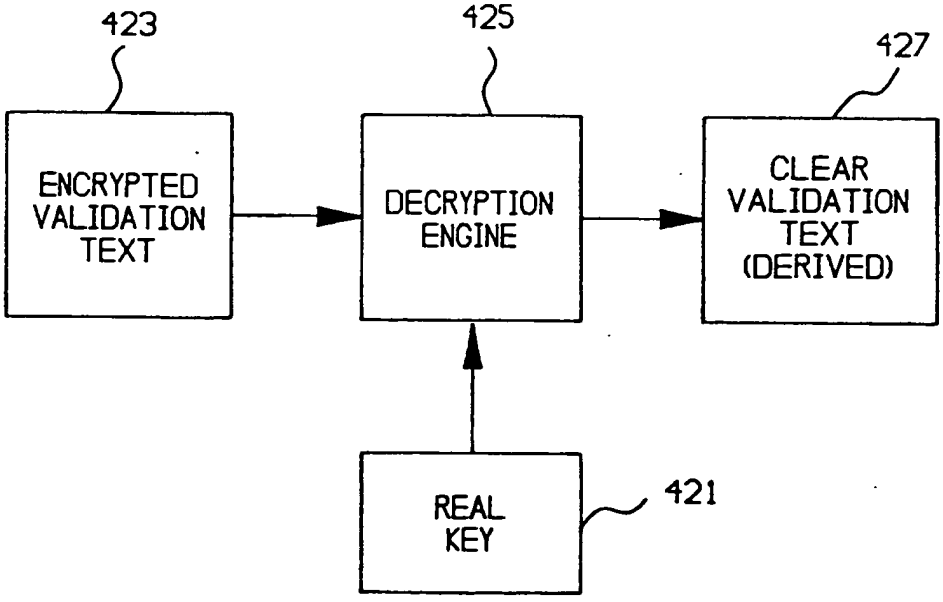


FIG. 21

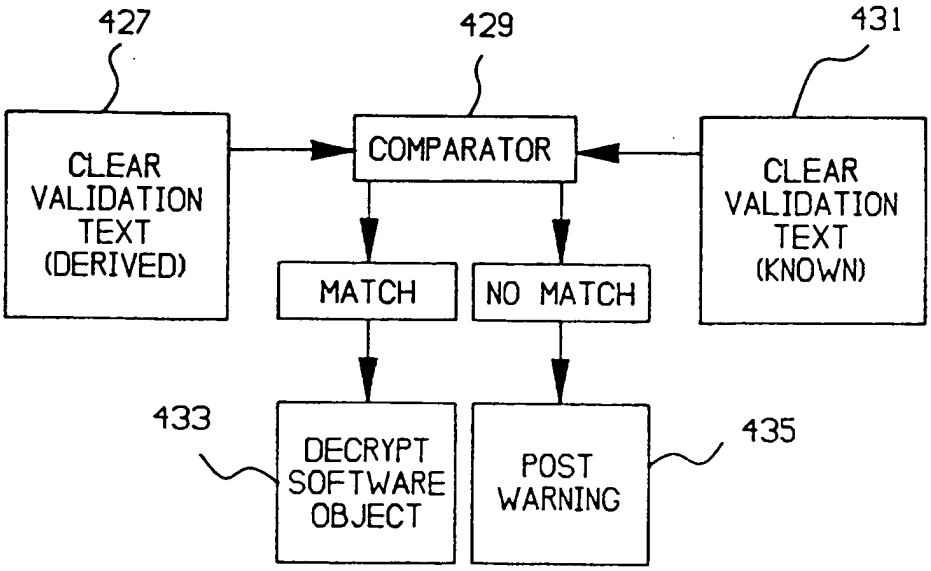


FIG. 22

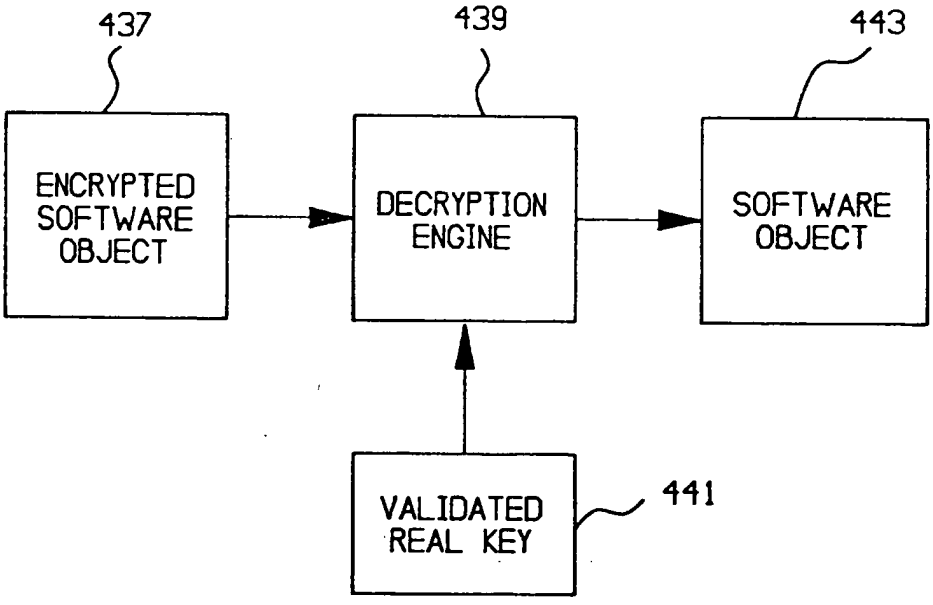
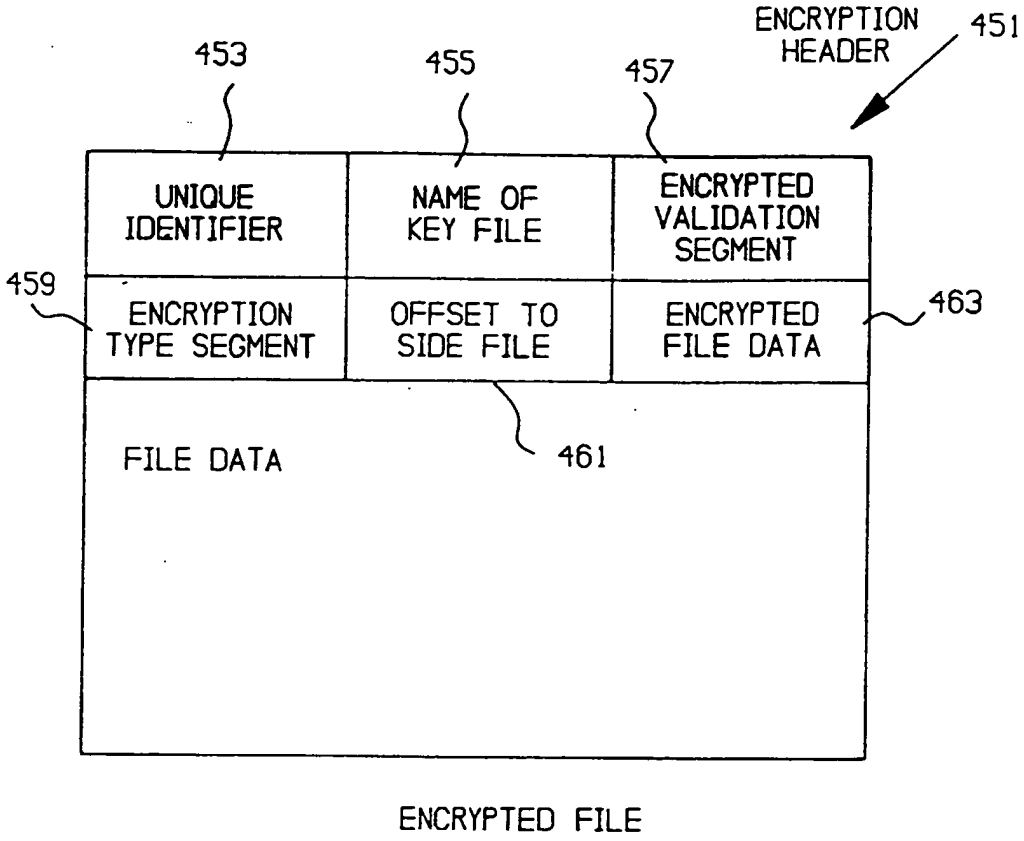


FIG. 23



ENCRYPTED FILE

FIG. 24

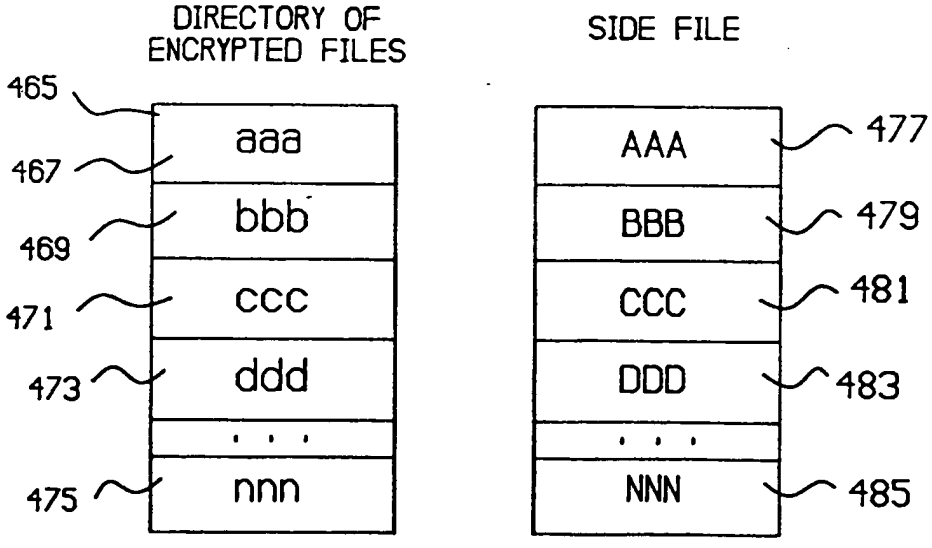


FIG. 25

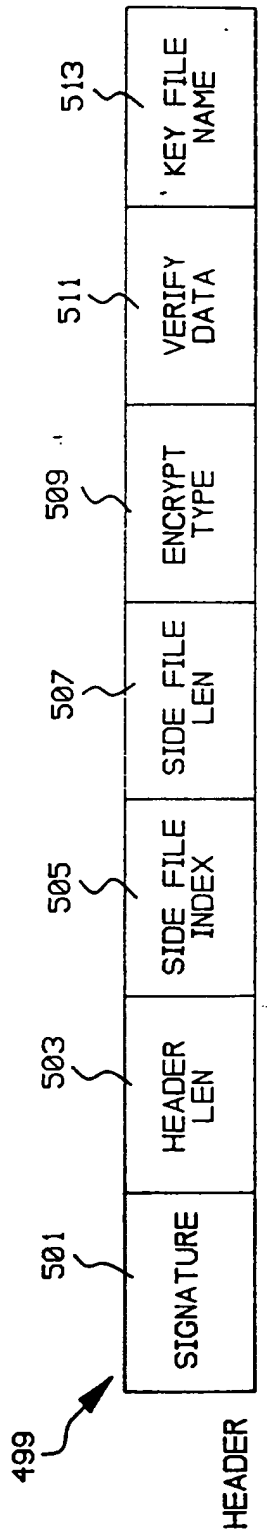


FIG. 26

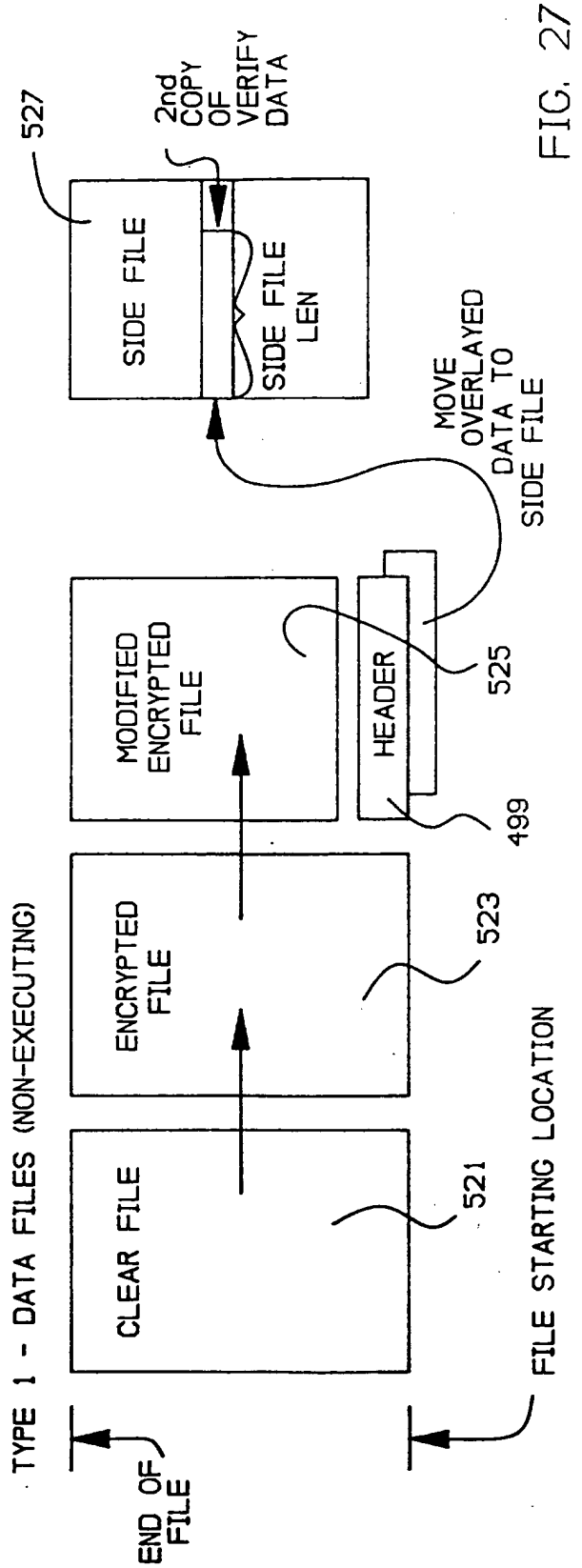


FIG. 27

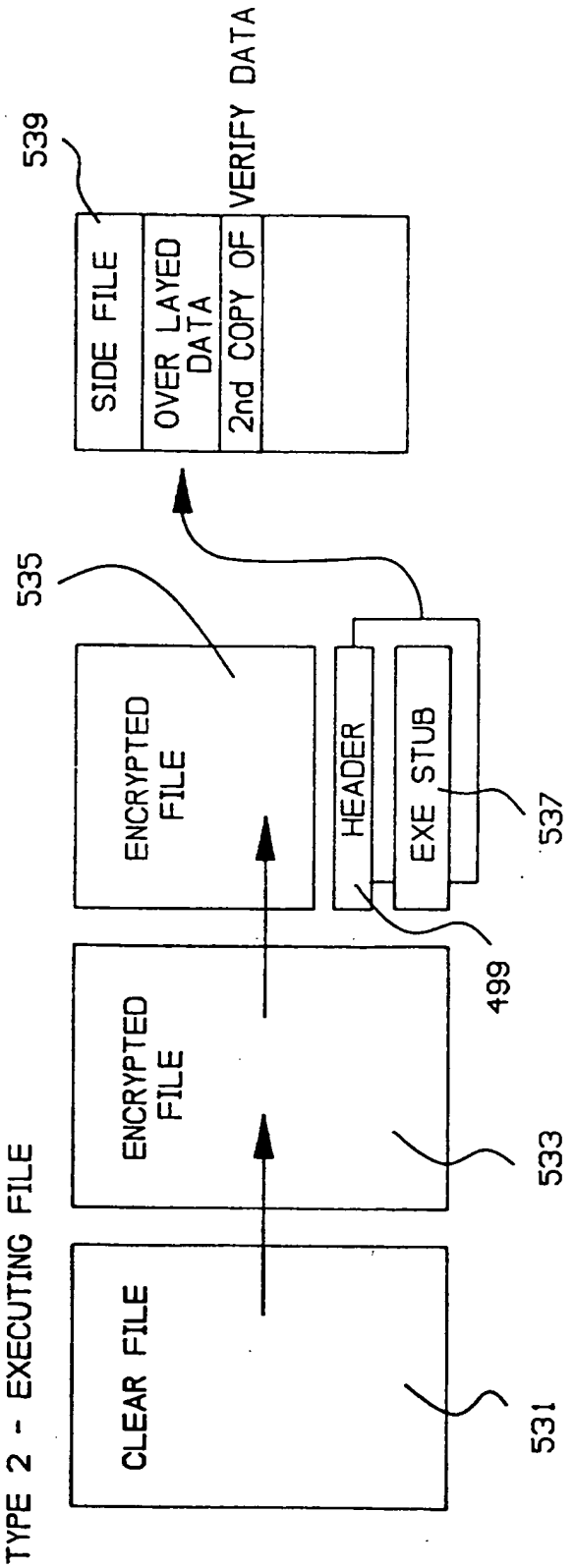
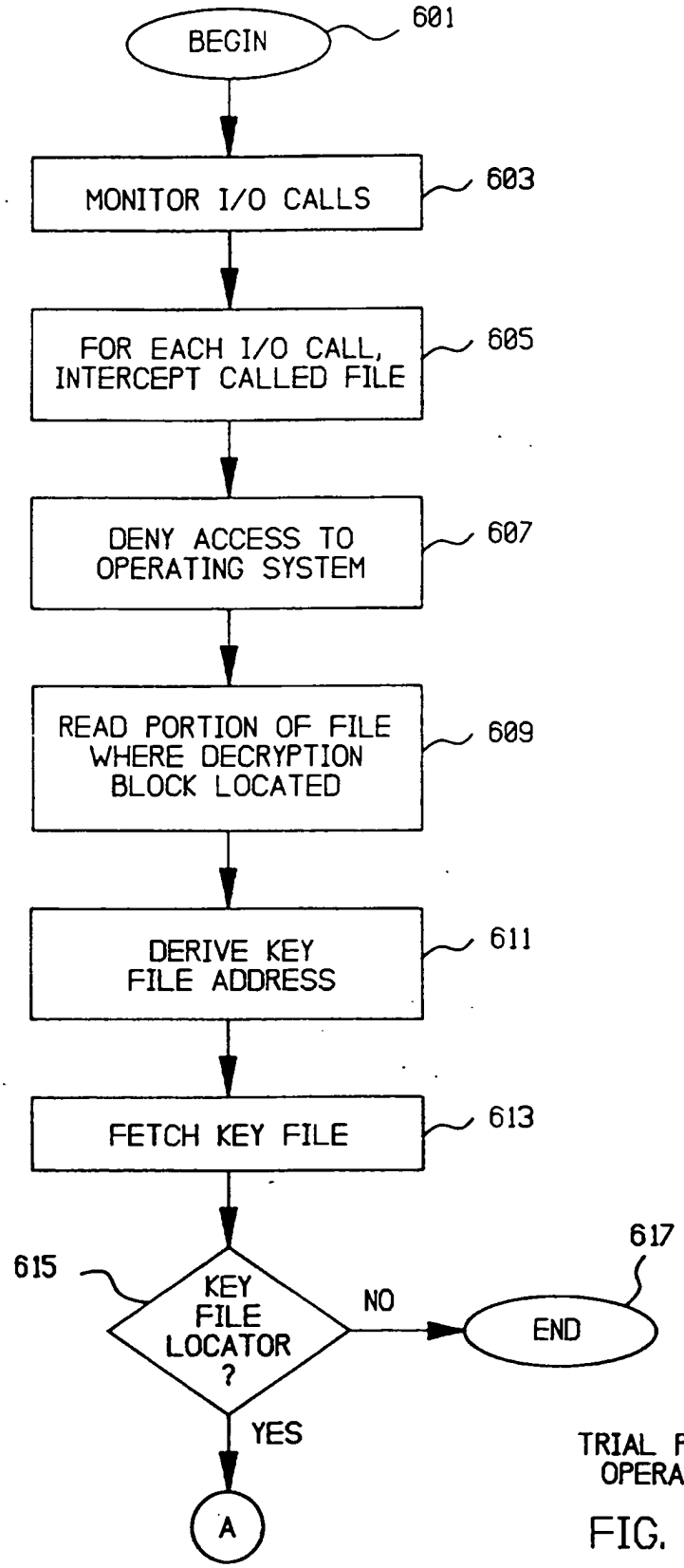
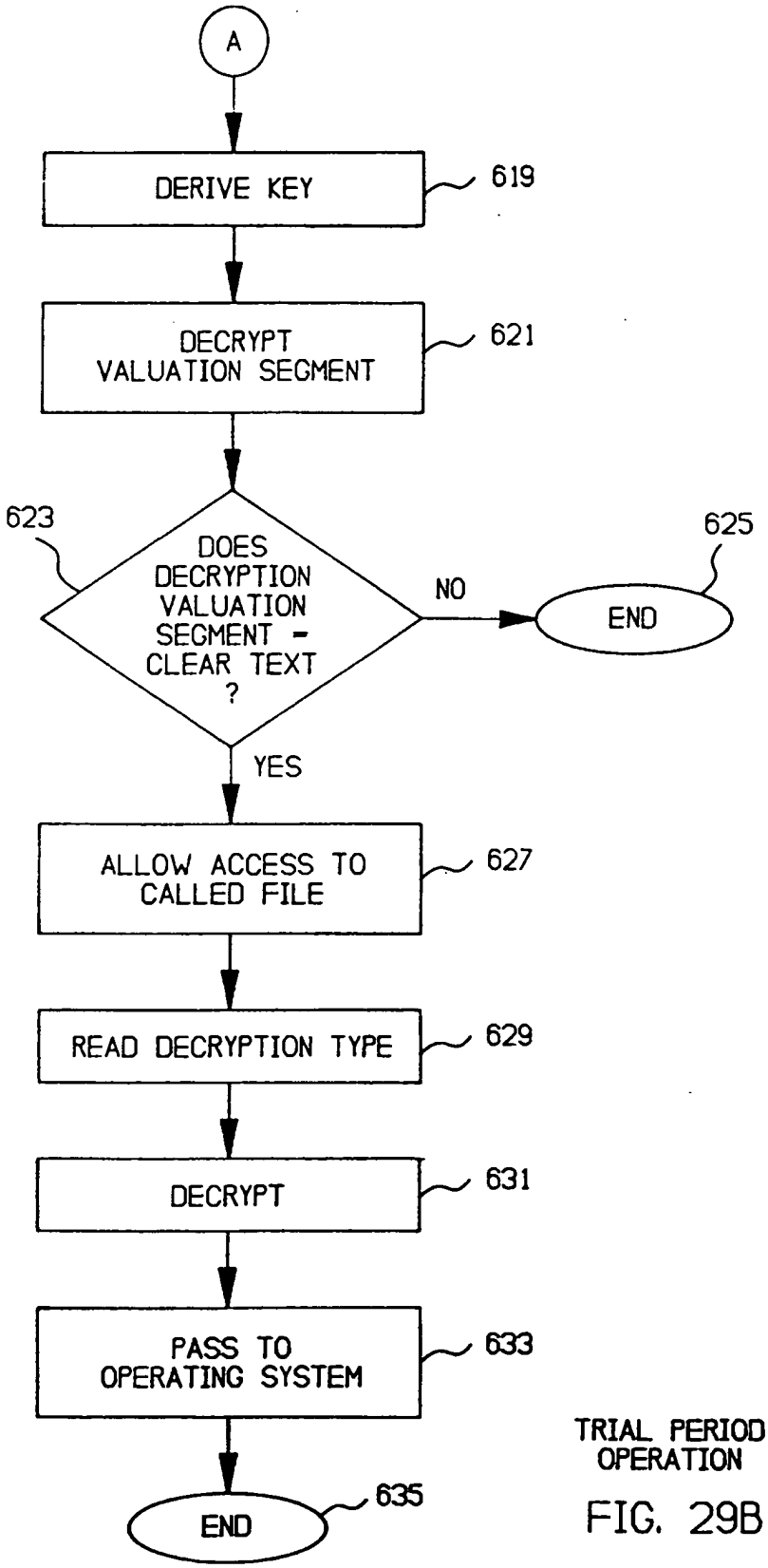


FIG. 28



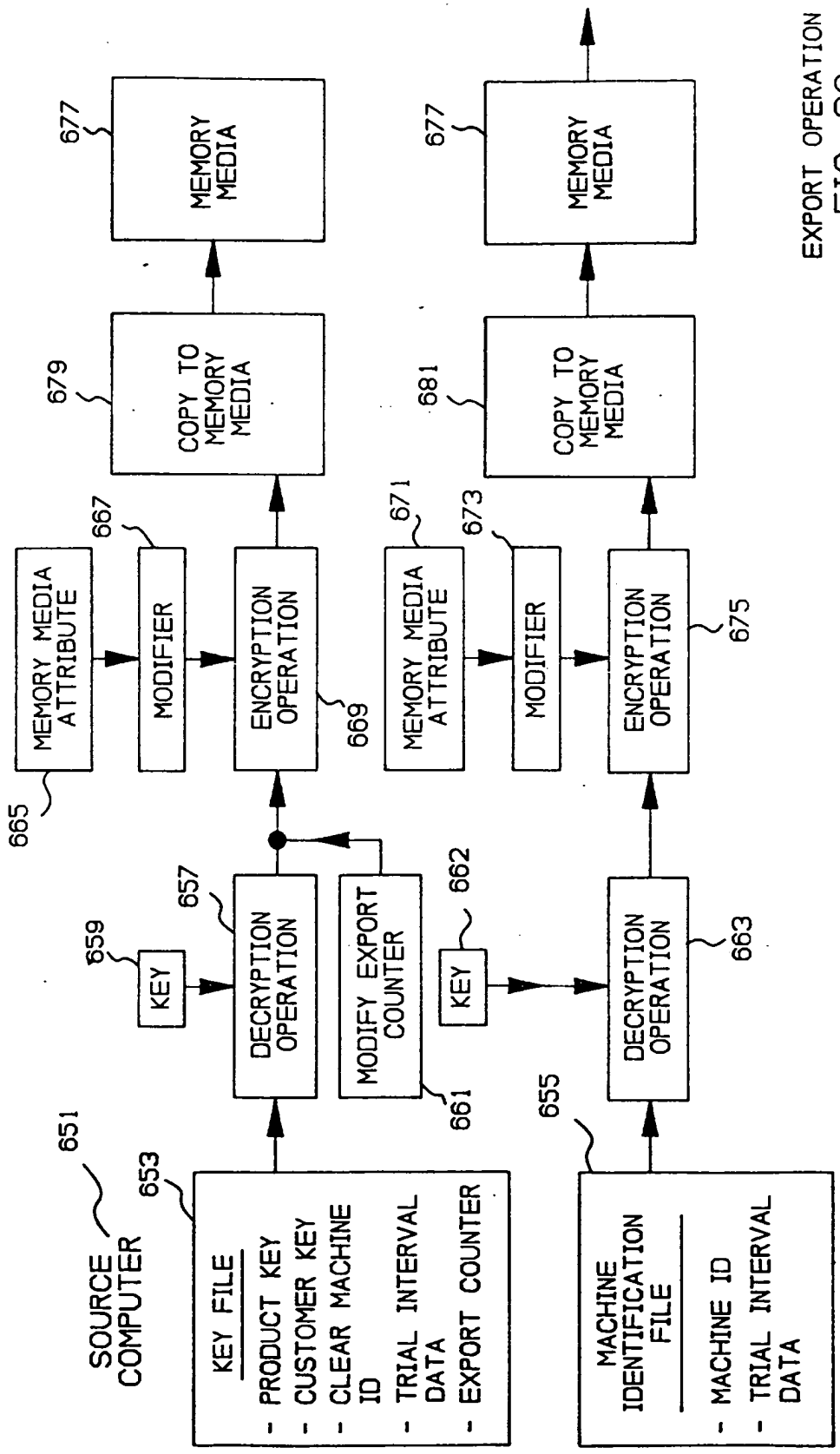
TRIAL PERIOD OPERATION

FIG. 29A



TRIAL PERIOD OPERATION

FIG. 29B



EXPORT OPERATION
FIG. 30

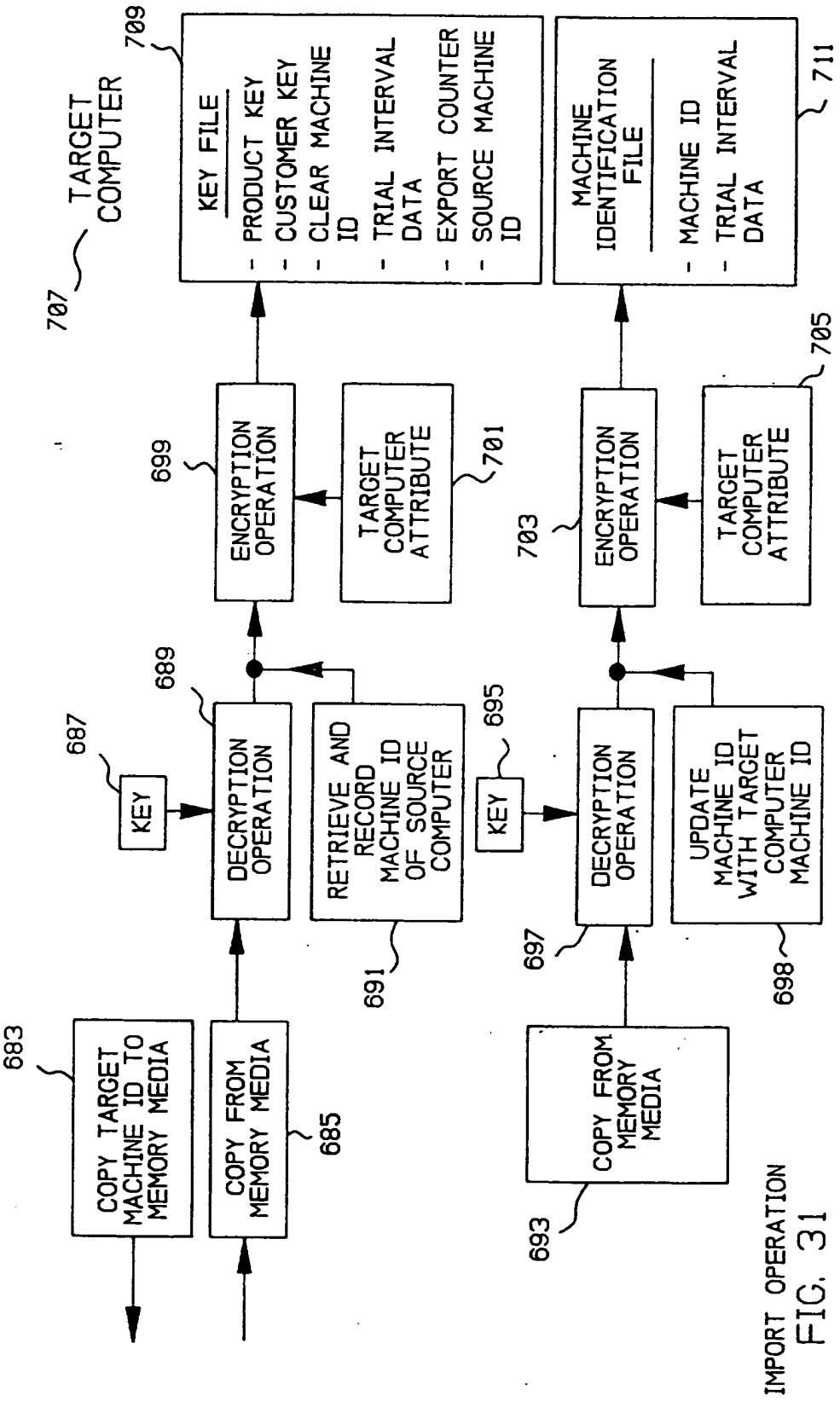


FIG. 31

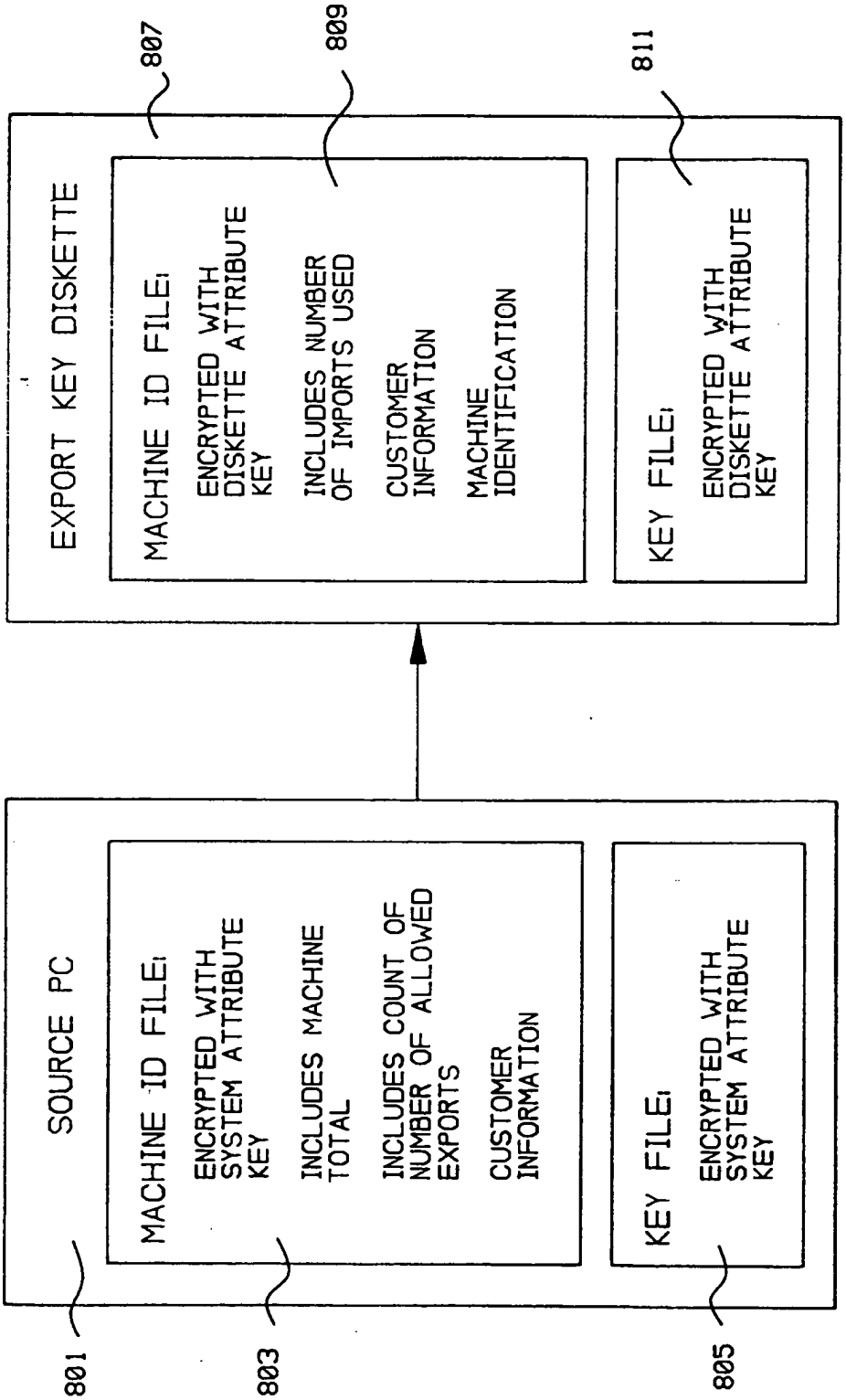


FIG. 32

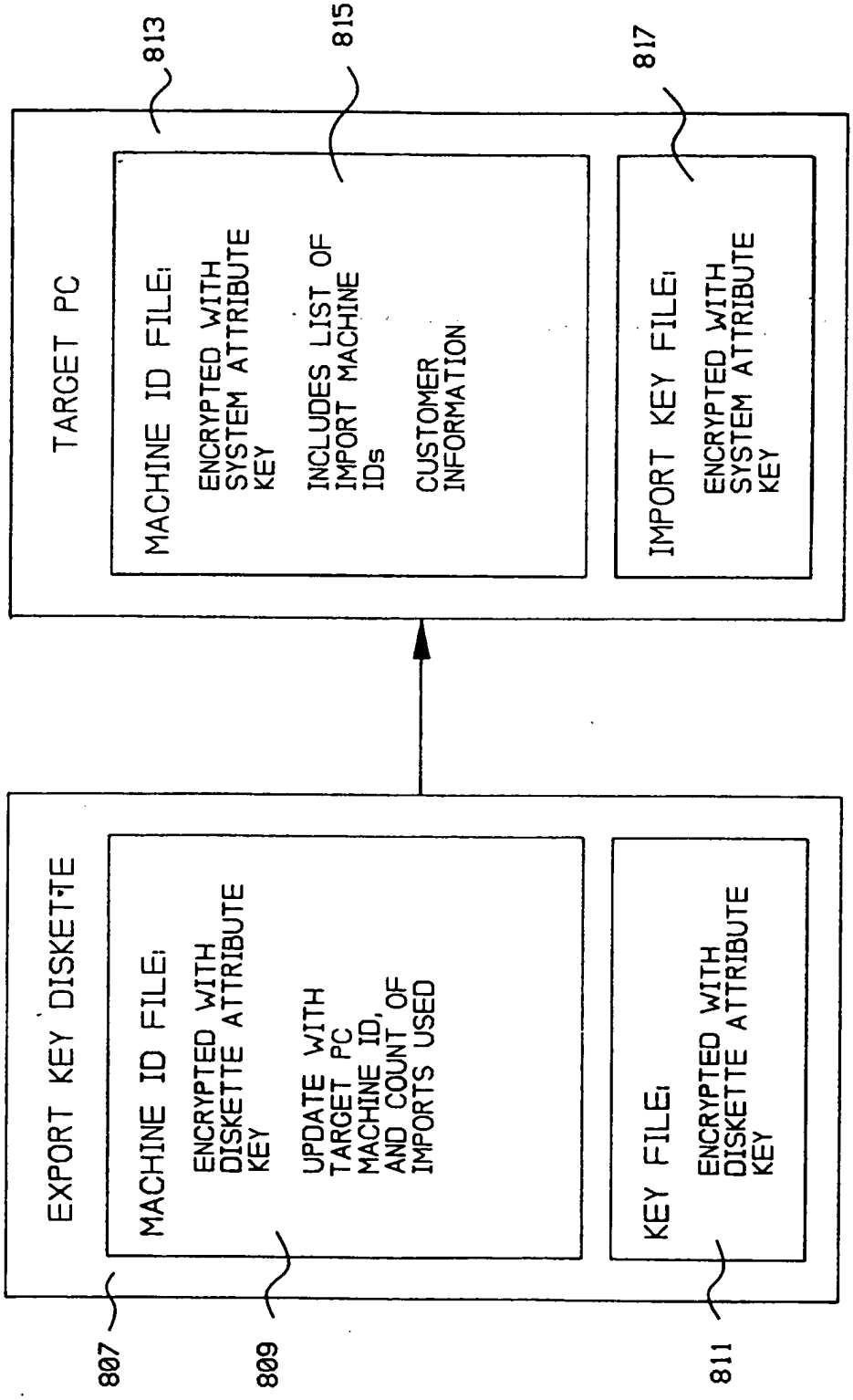
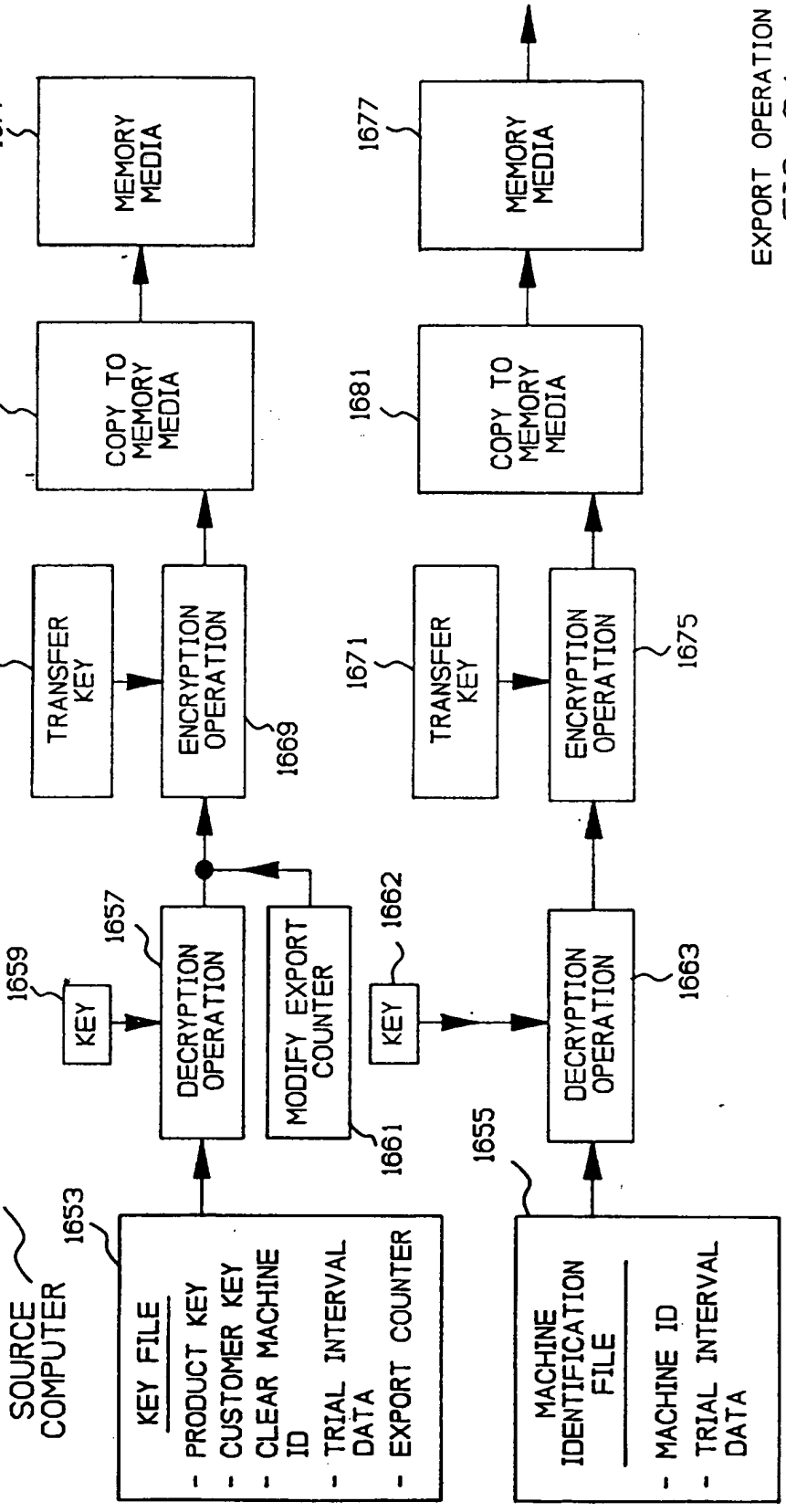


FIG. 33



EXPORT OPERATION
FIG. 34

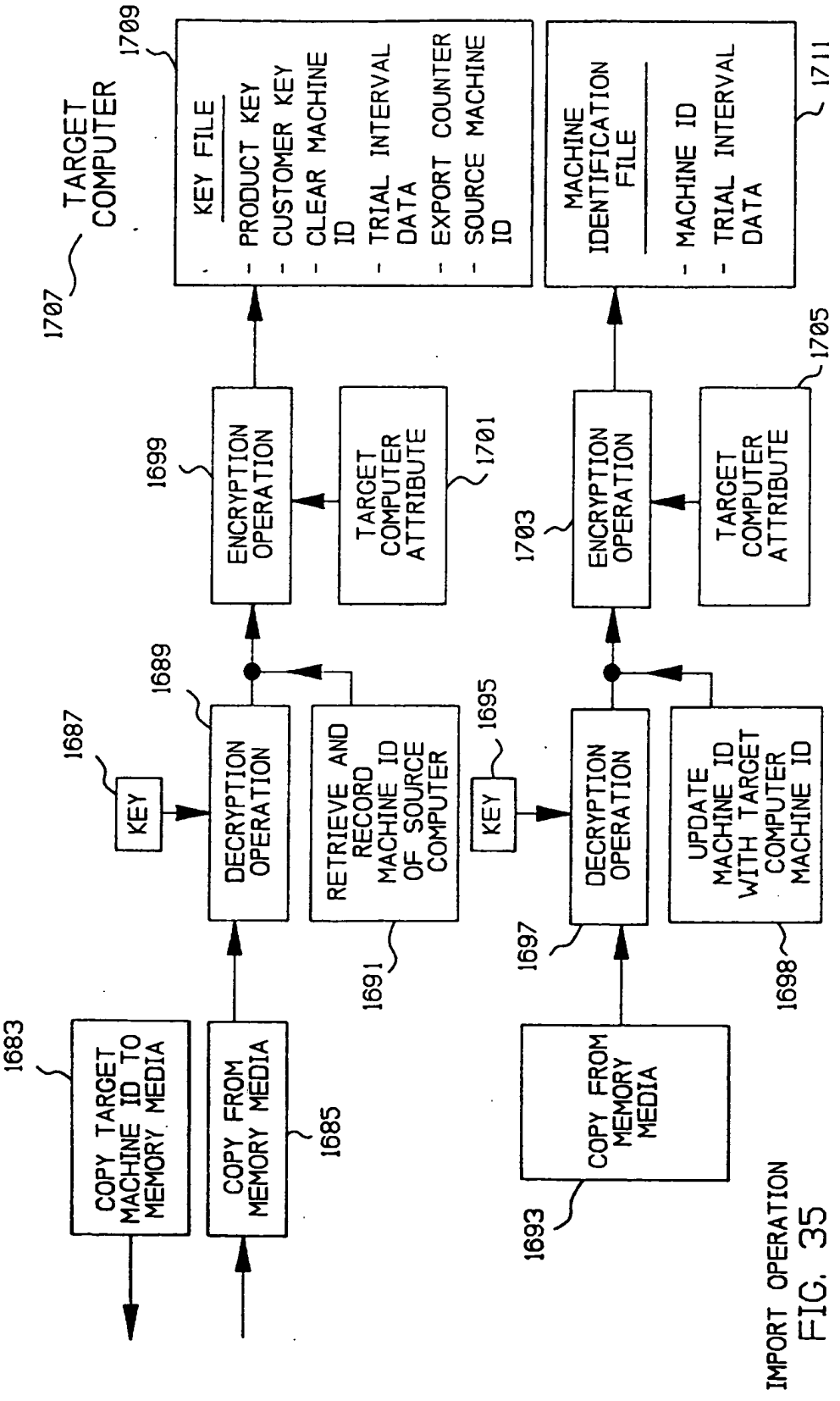


FIG. 35

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 10 5400

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	WO-A-94 07204 (UNILOC) * abstract; figures 41,2,8 * * page 6, line 11 - page 9, line 5 * * page 10, line 3 - line 10 * * page 12, line 7 - page 17, line 13 *	1,9	G06F1/00 G06F12/14
Y		2,4-8,10	
Y	GB-A-2 136 175 (ATALLA) * the whole document *	2	
Y	EP-A-0 268 139 (IBM) * column 1, line 1 - column 3, line 1 * * column 6, line 7 - column 7, line 50 * * column 9, line 20 - line 29 * * column 19, line 9 - line 50 * * column 21, line 6 - line 18 * * claims 2,9 *	4-8,10	
A		3	
A	EP-A-0 561 685 (FUJITSU) * the whole document *	9	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 July 1995	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 243 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**, G06F 17/60

(21) Application number: **95308414.2**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 344773**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventors:
- **Stefik, Mark J.**
  Woodside, California 94062 (US)
- **Piroll, Peter L.T.**
  El Cerrrito, California 94530 (US)
- **Merkle, Ralph C.**
  Sunnyvale, California 94087 (US)

(74) Representative: **Goode, Ian Roy et al**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works having a fee reporting mechanism

(57) A fee accounting mechanism for reporting fees associated with the distribution and use of digital works. Usage rights and fees are attached to digital works. The usage rights define how the digital work may be used or further distributed. Usage fees are specified as part of a usage right. The digital works and their usage rights and fees are stored in repositories (201). The repositories control access to the digital works. Upon determination that the exercise of a usage right requires a fee, the repository generates a fee reporting transaction (302). Fee reporting is done to a credit server (301). The credit server collects the fee information and periodically transmits it to a billing clearinghouse (303).

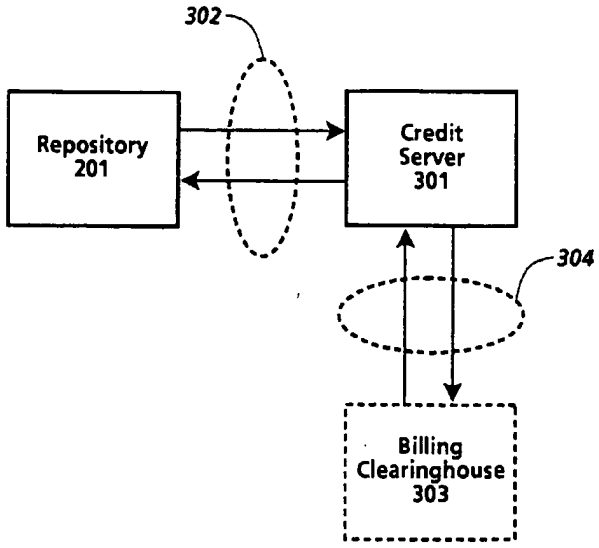


Fig. 3

EP 0 715 243 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

In a system for the control of distribution and use of digital works, a fee reporting mechanism for reporting fees associated with such distribution and use is disclosed. The system includes a means for attaching usage rights to a digital work. The usage rights define how the digital work may be used or further distributed by a possessor of the digital work. Usage fees are specified as part of a usage right. The ability to report usage fees may be a condition to the exercise of a usage right. Further, different fees may be assigned to different usage rights.

The present invention enables various usage fee scenarios to be used. Fees may be assessed on a per use basis, on a metered basis or based on a predetermined schedule. Fees may also be discounted on a predetermined schedule, or they can be marked-up a predetermined percentage (e.g. as a distributor fee). Fee reporting may also be deferred to a later time, to accommodate special deals, rebates or some other external information not yet available.

The present invention supports usage fees in an additive fashion. Usage fees may be reported for a composite digital work, i.e. a digital work comprised of a plurality of discrete digital works each having their own usage rights, and for distributors of digital works. Accordingly, fees to multiple revenue owners can be reported.

Usage fee reporting is done to a credit server. The credit server collects the fee information and periodically transmits it to a billing clearinghouse. Alternatively, the credit server may have a pre-allocated credit which is decremented as fees are incurred. In this alternative embodiment, the credit server would have to be periodically reallocated with credits to enable further use.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of

the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. Assuming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository

2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multifunction" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned

for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system.Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repositery has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.

Continuation of the Table on the next page

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset thresholds that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy

or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a cardsized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a|b|c]" is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces {} are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x^*) is used to indicate a variable length list, that is, a list containing one or more items of type x. The notation $(x)^*$ is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases,

the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticket, etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights: = (Rights*)"** define the digital work rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digital work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-CodeI Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play : {Player: Player-ID} I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play — A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print — To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan (Remaining-Rights: Next-Set-of-Rights)] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy — Make a new copy of a work
- Transfer — Moving a work from one repository to another.
- Loan — Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code : = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}I Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders

which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code : [Extract I Embed I Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code: = Install I Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights : = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer I 0 I unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable I Unrestrictable} {Unchargeable I Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as "forever", then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval : = From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If "SC: " is not specified, the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers. For such access tests, the communication would need to be established and authorization obtained before the right

could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec I Scheduled-Fee-Spec I Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: I Incentive: } [Per-Use-Spec I Metered-Rate-Spec I Best-Price-Spec I Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if **Fee**: is specified. Incentives are paid by the revenue-owner to the user if Incentive: is specified. If the **Min**: specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the **Max**: specification is given, then there is a maximum fee to be charged per time-spec for its use. When **Fee**: is specified, **Account-ID** identifies the account to which the fee is to be paid. When Incentive: is specified, **Account-ID** identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec := Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec := Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates, and pricing that depends on information that is not available to the repository. All fee specifications can be combined

with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the **Max**: field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec : = Call-For-Price "** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. **A Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The

second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository- 1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOG IN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOG IN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To sim-

plify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets --the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the

requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right speci fi c steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work, unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.

- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for "printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage,

such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted, the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the directory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not return the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However,

it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.
- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A system for controlling the distribution and use of digital works having a mechanism for reporting fees based on the distribution and use of digital works, said system comprising:

   means for attaching usage rights to a digital work, each of said usage rights specifying how a digital work may be used or distributed, each of said usage rights specifying usage fee information, said usage fee information

comprising a fee type and fee parameters which define a fee to be paid in connection with the exercise of said usage right;
a communication medium for coupling repositories to enable communication between repositories; and
a plurality of repositories, each of said repositories comprising:
an external interface for removably coupling to said communications medium;
storage means for storing digital works having attached usage rights and fees;
requesting means for generating a request to access a digital work stored in another of said plurality of repositories, said request indicating a particular usage right; and
processing means for processing requests to access digital works stored in said storage means and for generating fee transactions when a request indicates a usage right that is attached to a digital work and said usage right specifies usage fee information;
each of said plurality of repositories being removably coupled to a credit server, said credit server being arranged for recording fee transactions from said repository and subsequently reporting said fee transactions to a billing clearinghouse.

2. The fee reporting system as recited in Claim 1 wherein said fee type of said fee information is a metered use fee, a per use fee, a best price fee, a scheduled fee,or a mark-up fee.

3. A method for reporting fees associated with the distribution and use of digital works in a system for controlling the distribution and use of digital works, said method comprising the steps of:

a) attaching one or more usage rights to a digital work, each of said one or more usage rights comprising an indicator of how said digital work may be distributed or used and a usage fee to be paid upon exercise of said right;
b) storing said digital work and attached one or more usage rights in a server repository, said server repository controlling access to said digital work;
c) said server repository receiving a request to access said digital work from a requesting repository;
d) said server repository identifying a usage right associated with said access request;
e) said server repository determining if said identified usage right is the same as one of said one or more usage rights attached to said digital work;
f) if said identified usage right is not the same as any one of said one or more usage rights attached to said digital work, said server repository denying access to said digital work;
g) if said usage right is included with said digital work, said server repository determining if a usage fee is associated with the exercise of said usage right;
h) if a usage fee is associated with usage right, said server repository calculating said usage fee;
i) said server repository transmitting a first assign fee transaction identifying said requesting repository as a payer for said usage fee to a first credit server;
j) said requesting repository transmitting a second assign fee transaction identifying said requesting repository as a payer for said usage fee to a second credit server;
k) said server repository transmitting said digital work to said requesting repository;
l) said server repository transmitting a first confirm fee transaction to said first credit server; and
m) said requesting repository transmitting a second confirm fee transaction to said second credit server.

4. The method as recited in Claim 3 wherein said digital work is comprised of a plurality of independent digital works and said step of said server calculating said usage fee is further comprised of the step of reporting the usage fees for each of the plurality of independent digital works.

5. A method for reporting fees associated with the distribution and use of digital works in a system for controlling the distribution and use of digital works, said method comprising the steps of:

a) attaching one or more usage rights to a digital work, each of said one or more usage rights comprising an indicator of how said digital work may be distributed or used and a usage fee to be paid for exercise of said right;
b) storing said digital work and said attached one or more usage rights in a server repository, said server repository controlling access to said digital work;
c) said server repository receiving a request to access said digital work from a requesting repository;
d) said server repository identifying a usage right associated with said access request;
e) said server repository determining if said digital work has attached thereto said identified usage right;
f) if said identified usage right is not attached to said digital work, said server repository denying access to

said digital work;
g) if said usage right is attached to said digital work, said server repository determining if a usage fee is associated with the exercise of said usage right;
h) if a usage fee is associated with said usage right, said server repository determining a fee type;
i) said server repository transmitting a first fee transaction identifying said requesting repository as a payee for said usage fee to a credit server, said first fee transaction being dependent on said determined fee type; and
k) said server repository transmitting said digital work to said requesting repository.

6. A system for controlling the distribution and utilization of digital works having a mechanism for reporting usage fees, said system comprising:

digital works comprising a first part for storing the digitally encoded data corresponding to a digital work and a second part for storing usage rights and fees for said digital work, said usage rights specifying how a digital work may be used or distributed and said usage fees specifying a fee to be paid in connection with the exercise of a corresponding usage right;
a plurality of repositories, each of said repositories comprising:
communication means for communicating with another of said plurality of repositories;
storage means for storing digital works;
requesting means for generating a request to access a digital work stored in another of said plurality of repositories, said request indicating a particular usage right;
processing means for processing requests to access digital works stored in said storage means and granting access when said particular usage right corresponds to a stored usage right stored in said digital work, said processing means generating fee transactions when said access is granted and said stored usage right specifies a fee;
each of said plurality of repositories being removably coupled to a credit server, said credit server being arranged for recording fee transactions from said repository and subsequently reporting said fee transactions to a billing clearinghouse.

7. The system as recited in Claim 6 wherein said storage means is further comprised of a first storage device for storing said first part of said digital work and a second storage device for storing said second part of said digital work.

8. A method for reporting fees associated with use of rendering digital works by a rendering device in a system for controlling the rendering of digital works by a rendering system, said rendering system comprised of a rendering repository and a rendering device, said rendering device utilizing a rendering digital work for rendering a digital work, said method comprising the steps of:

a) storing a first digital work in a server repository, said digital work specifying a first usage fee to be reported for a use of said first digital work;
b) storing a rendering digital work in said rendering repository, said first rendering digital work specifying a second usage fee to be reported for a use of said rendering digital work;
c) said server repository receiving a request to use said first digital work from said rendering repository;
d) said server repository determining if said request may be granted;
e) if said server repository determines that said request may not be granted, said server repository denying access to said first digital work;
f) if said server repository determines that said request may be granted, said server repository transmitting said digital work to said rendering repository;
g) said server repository transmitting a first fee transaction identifying said rendering repository as a payee for said first usage fee for use of said first digital work to a first credit server;
h) said rendering device rendering said first digital work using said rendering digital work; and
i) said rendering repository transmitting a second fee transaction identifying said rendering repository as a payee for said second usage fee for use of said rendering digital work to a second credit server.

9. The method as recited in Claim 8 further comprising the step of said rendering repository transmitting a third fee transaction identifying said rendering repository as a payee for said first usage fee for use of said first digital work to said second credit server.

10. The method as recited in Claim 9 wherein said rendering digital work is a set of coded rendering instructions for controlling said rendering device.

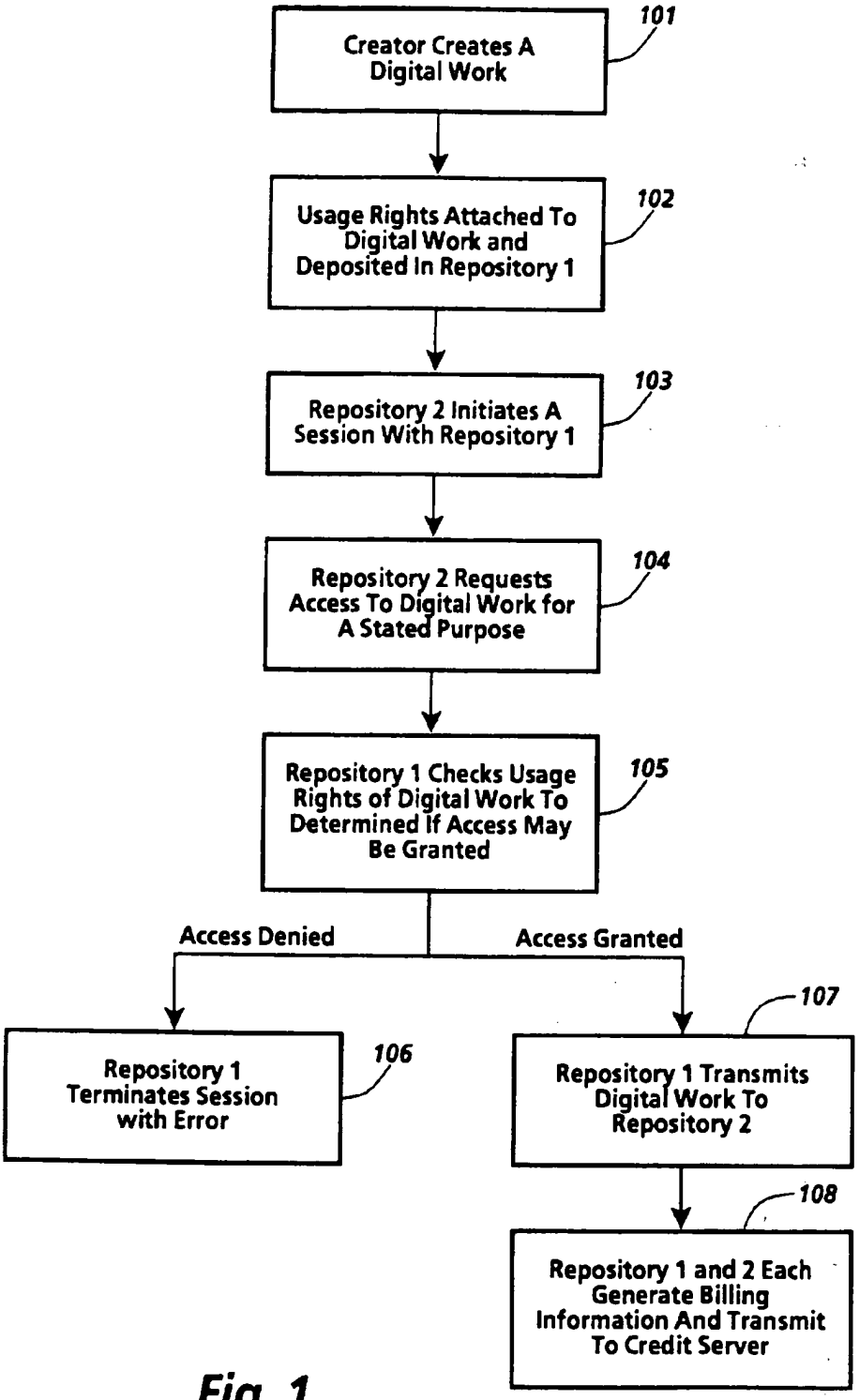


Fig. 1

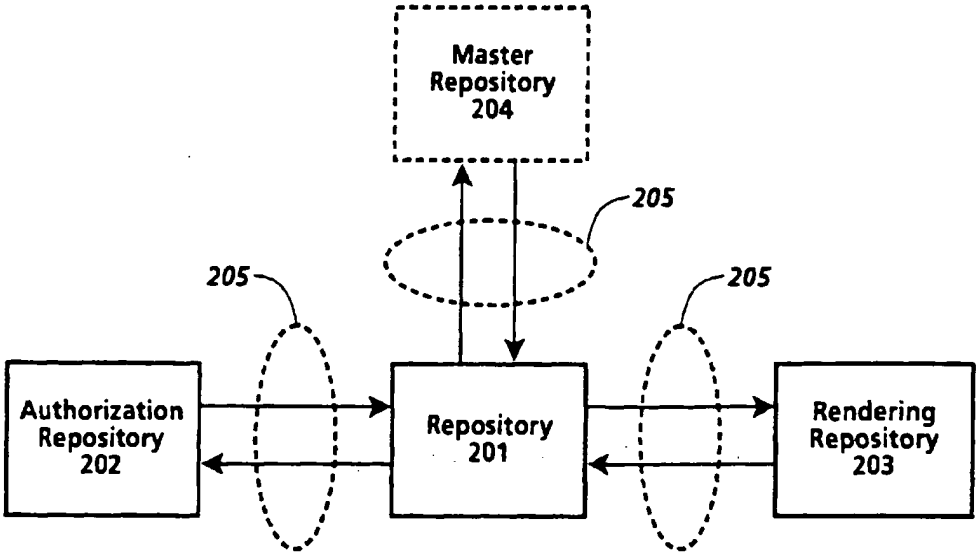


Fig. 2

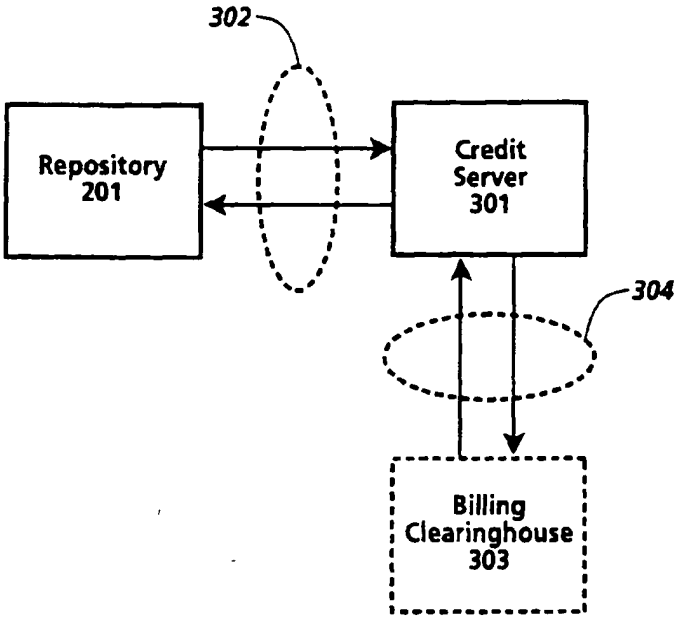


Fig. 3

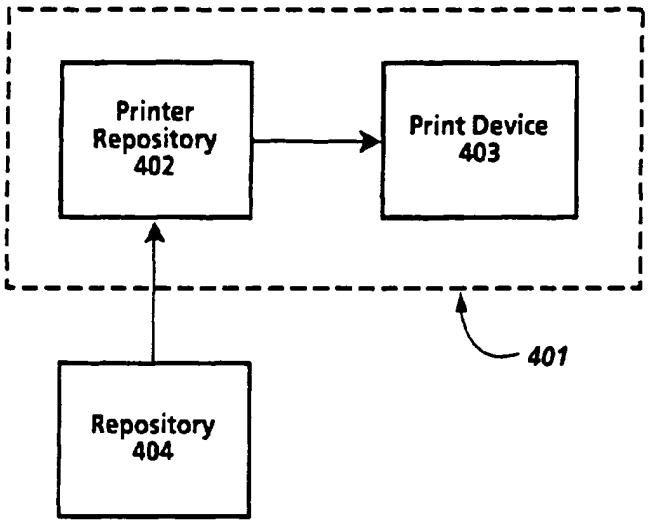


Fig. 4a

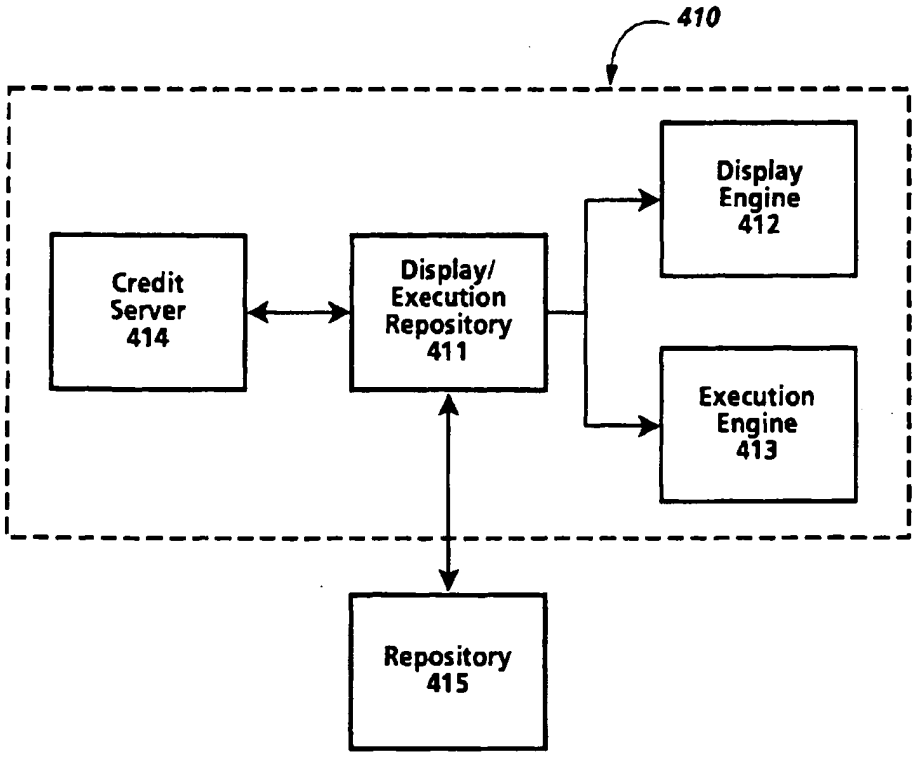


Fig. 4b

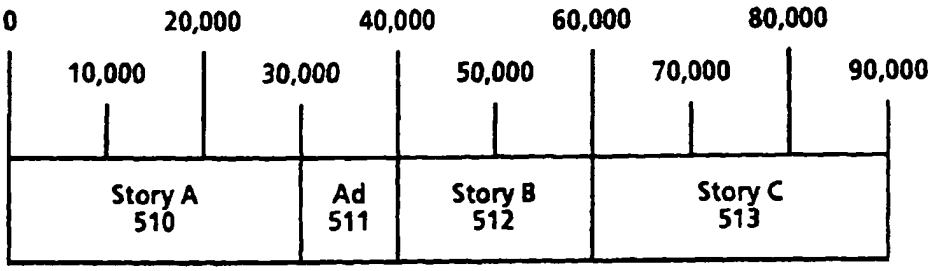


Fig. 5

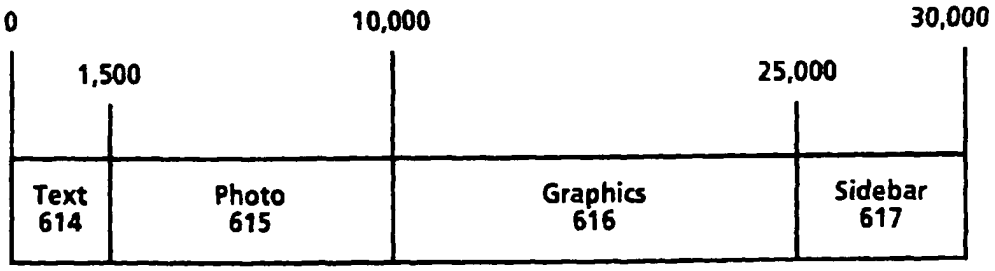


Fig. 6

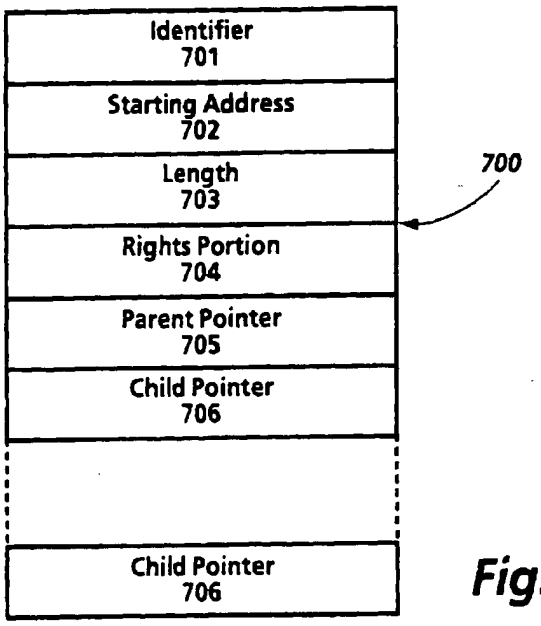


Fig. 7

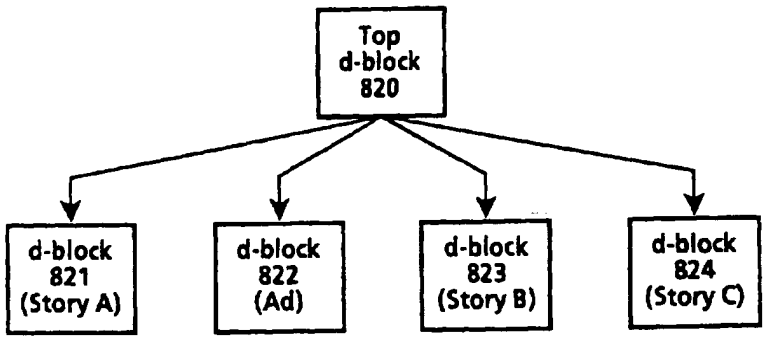


Fig. 8

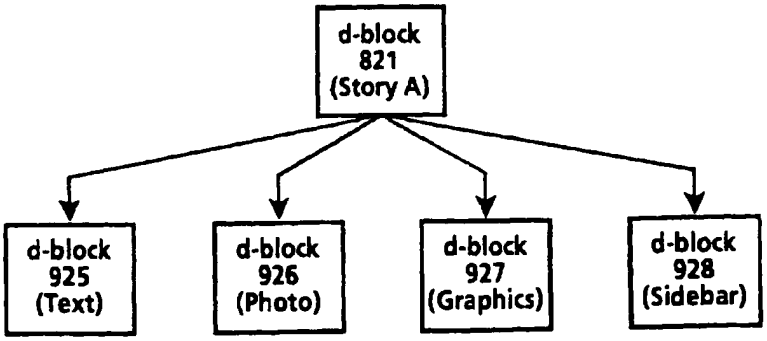


Fig. 9

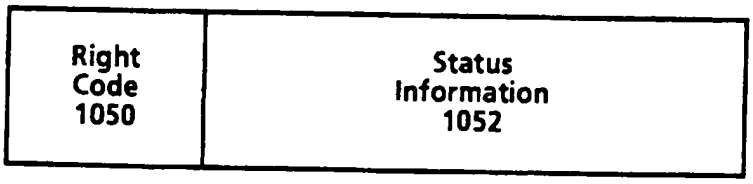


Fig.10

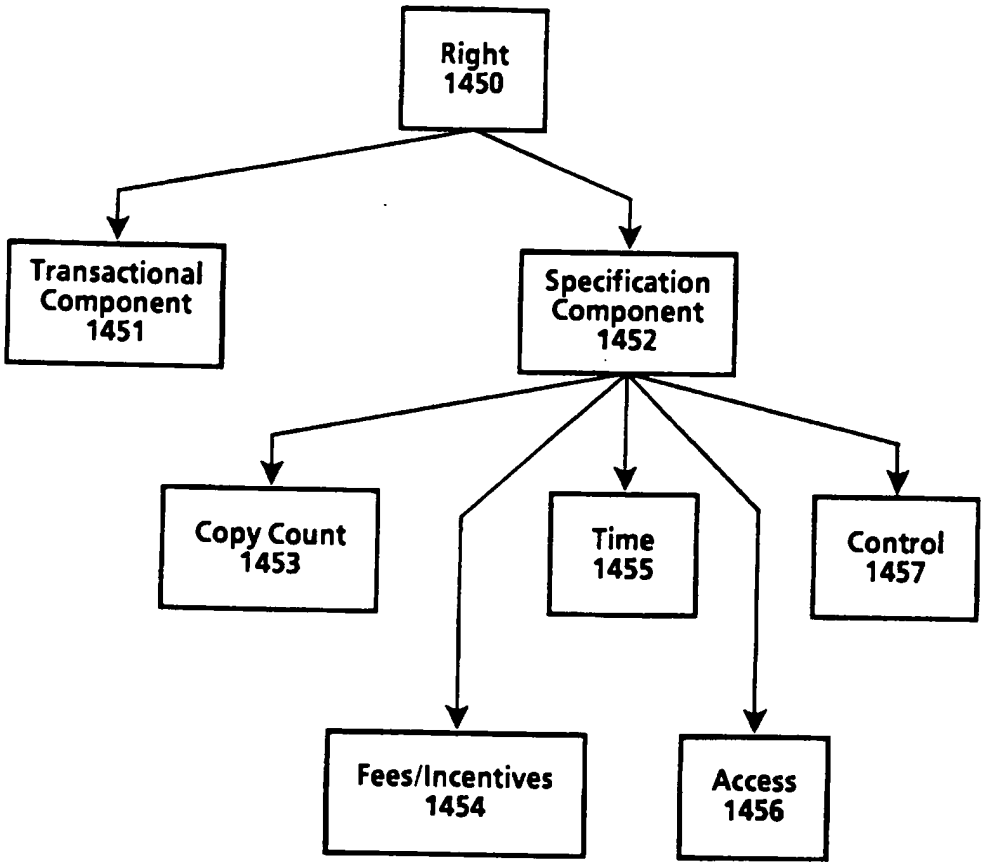


Fig.14

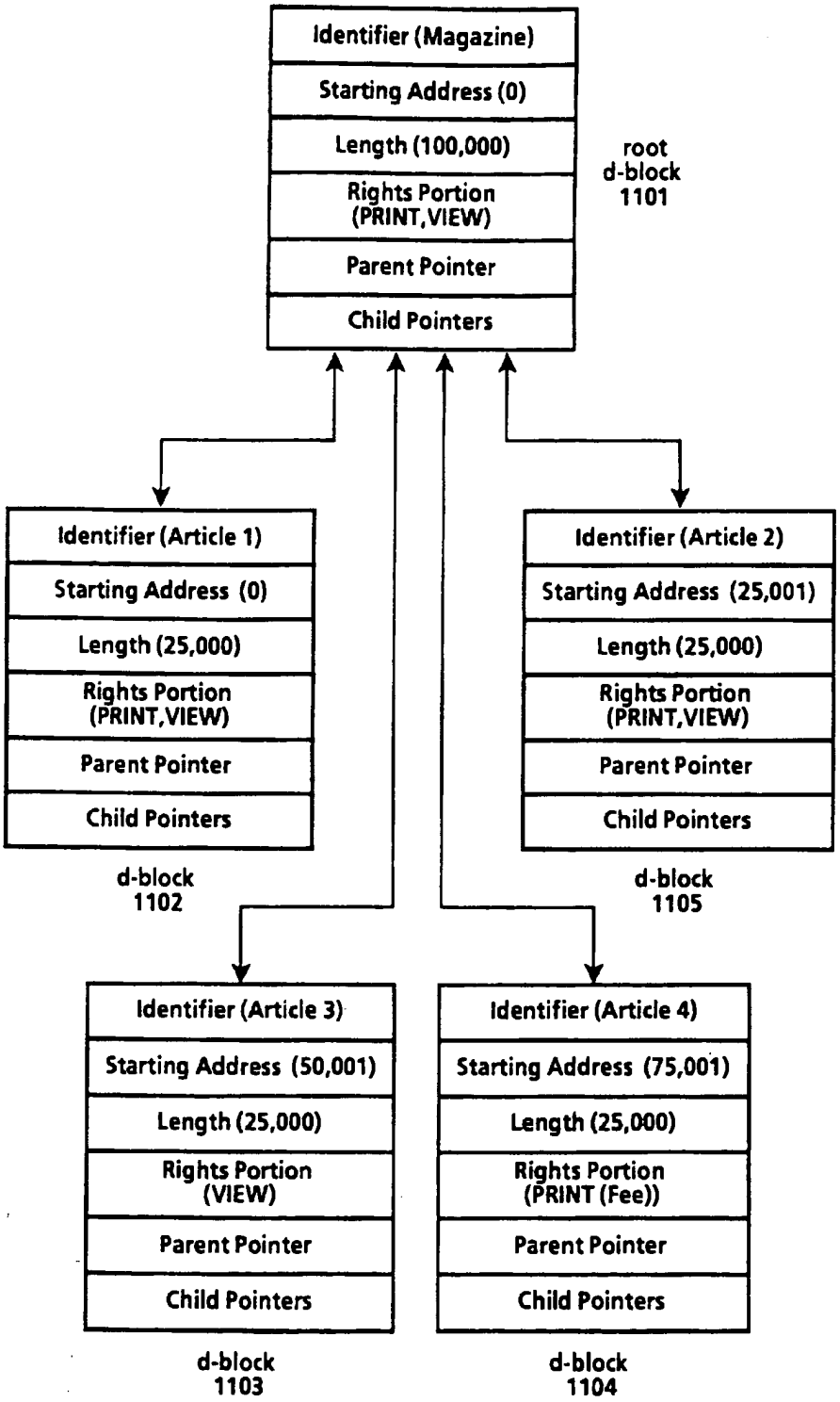


Fig.11

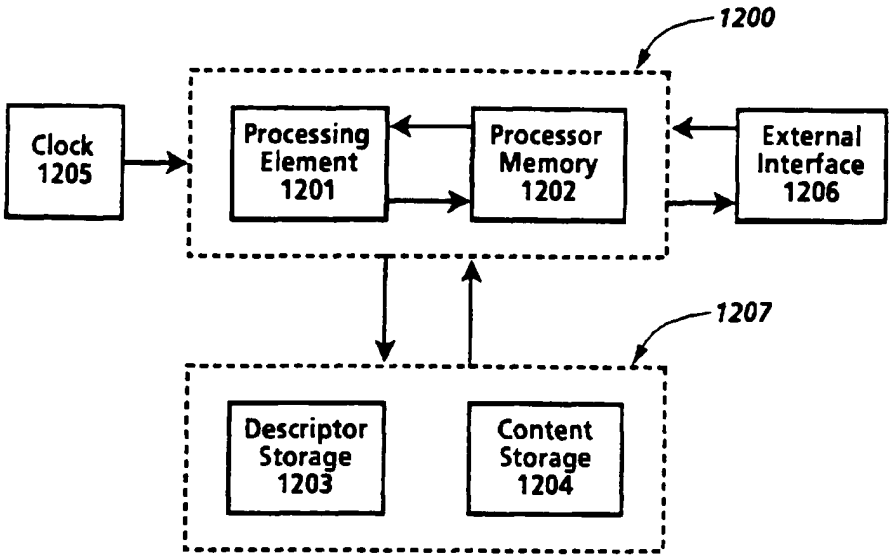


Fig.12

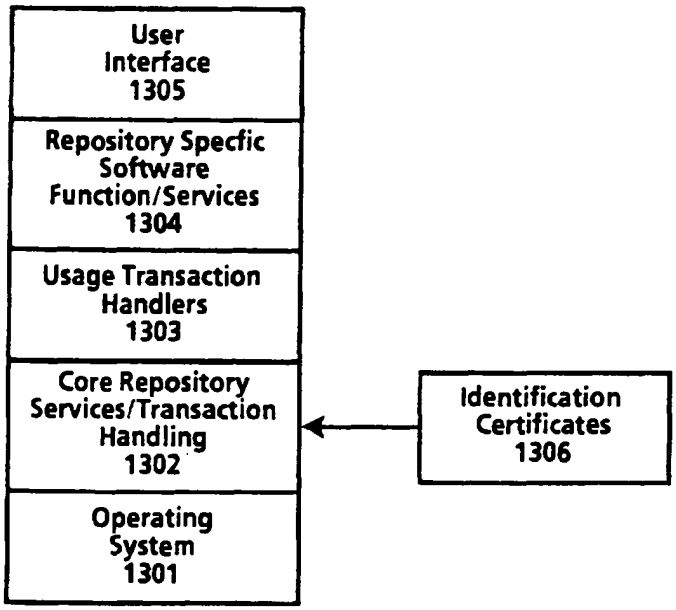


Fig.13

1501 ~ **Digital Work Rights**: = (Rights*)
1502 ~ **Right** : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec } {Access-Spec} {Fee-Spec})
1503 ~ **Right-Code** : = Render-Code | Transport-Code | File-Management-Code| Derivative-Works- Code | Configuration-Code
1504 ~ **Render-Code** : = [Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]
1505 ~ **Transport-Code** : = [Copy | Transfer | Loan {Remaining-Rights: Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}
1506 ~ **File-Management-Code** : = Backup {Back-Up-Copy-Rights: Next-Set-of-Rights} | Restore | Delete | Folder | Directory {Name: Hide-Local | Hide-Remote} {Parts: Hide-Local | Hide-Remote}
1507 ~ **Derivative-Works-Code** : = [Extract | Embed | Edit{Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}
1508 ~ **Configuration-Code** : = Install | Uninstall
1509 ~ **Next-Set-of-Rights** : = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)}{(Keep: Set-Of-Rights)}
1510 ~ **Copy-Count** : = (Copies:positive-integer | 0 | Unlimited)
1511 ~ **Control-Spec** : = (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})
1512 ~ **Time-Spec** : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)
1513 ~ **Fixed-Interval** : = From: Start-Time
1514 ~ **Sliding-Interval** : = Interval: Use-Duration
1515 ~ **Meter-Time**: = Time-Remaining: Remaining-Use
1516 ~ **Access-Spec** : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})
1517 ~ **Fee-Spec**: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec
1518 ~ **Scheduled-Discount**: = Scheduled-Discount: (Scheduled-Discount: (Time-Spec Percentage)*)
1519 ~ **Regular-Fee-Spec** : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)
1520 ~ **Per-Use-Spec**: = Per-Use: Money-unit
1521 ~ **Metered-Rate-Spec** : = Metered: Money-Unit Per: Time-Spec
1522 ~ **Best-Price-Spec** : = Best-Price: Money-unit Max: Money-unit
1523 ~ **Call-For-Price-Spec** : = Call-For -Price
1524 ~ **Scheduled-Fee-Spec**: = (Schedule: (Time-Spec Regular-Fee-Spec)*)
1525 ~ **Markup-Spec**: = Markup: percentage To: Account-ID

Fig.15

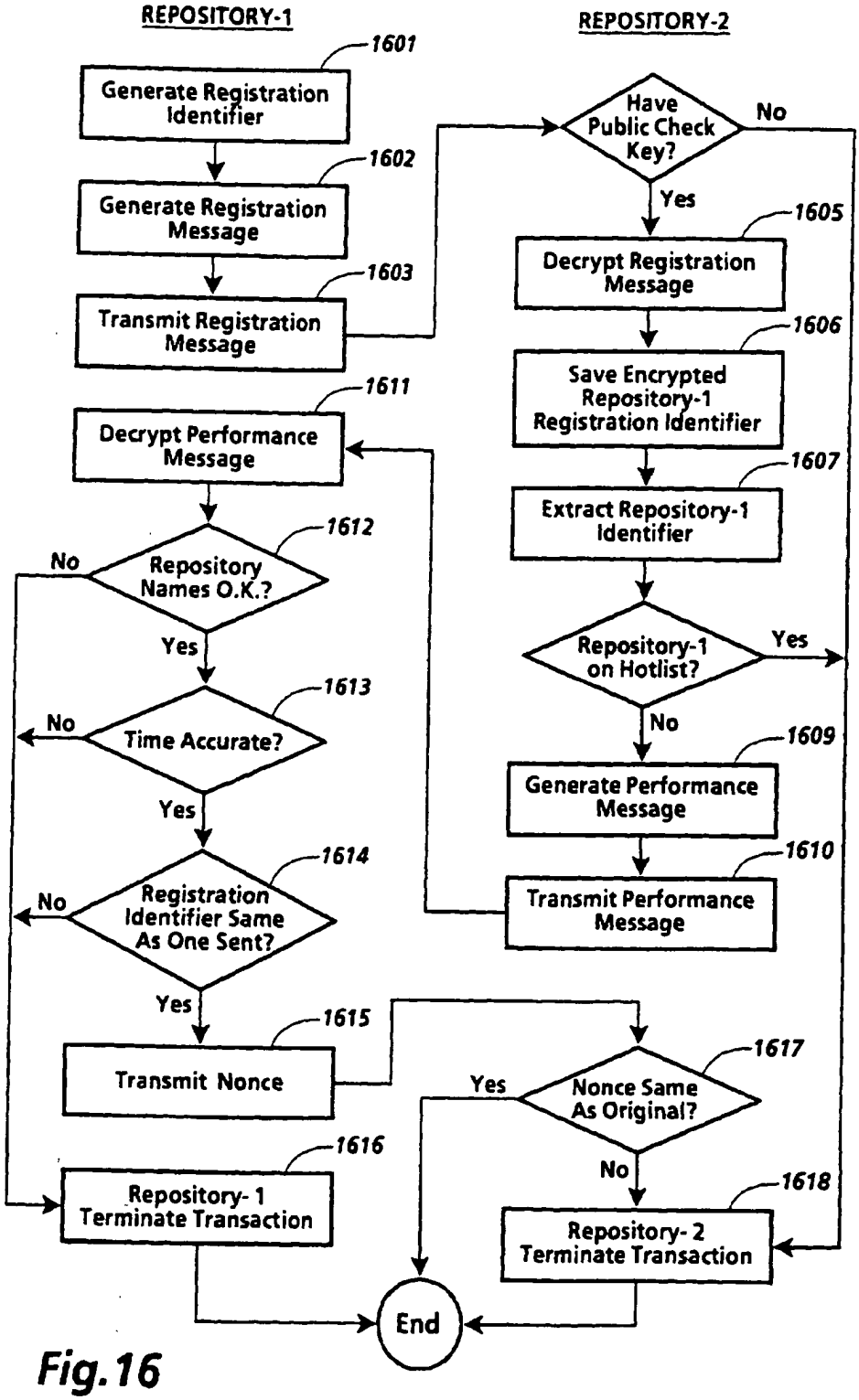


Fig.16

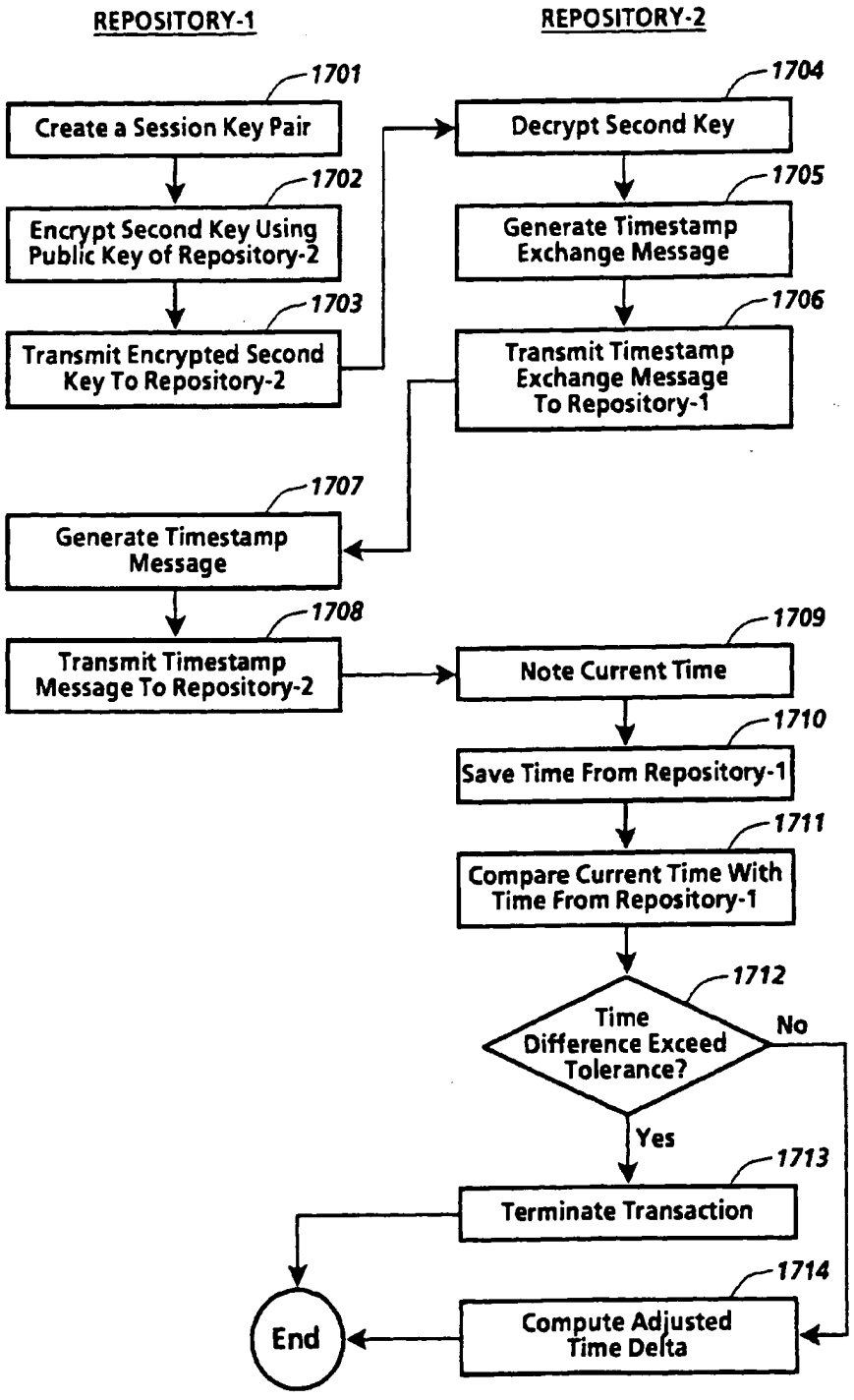


Fig.17

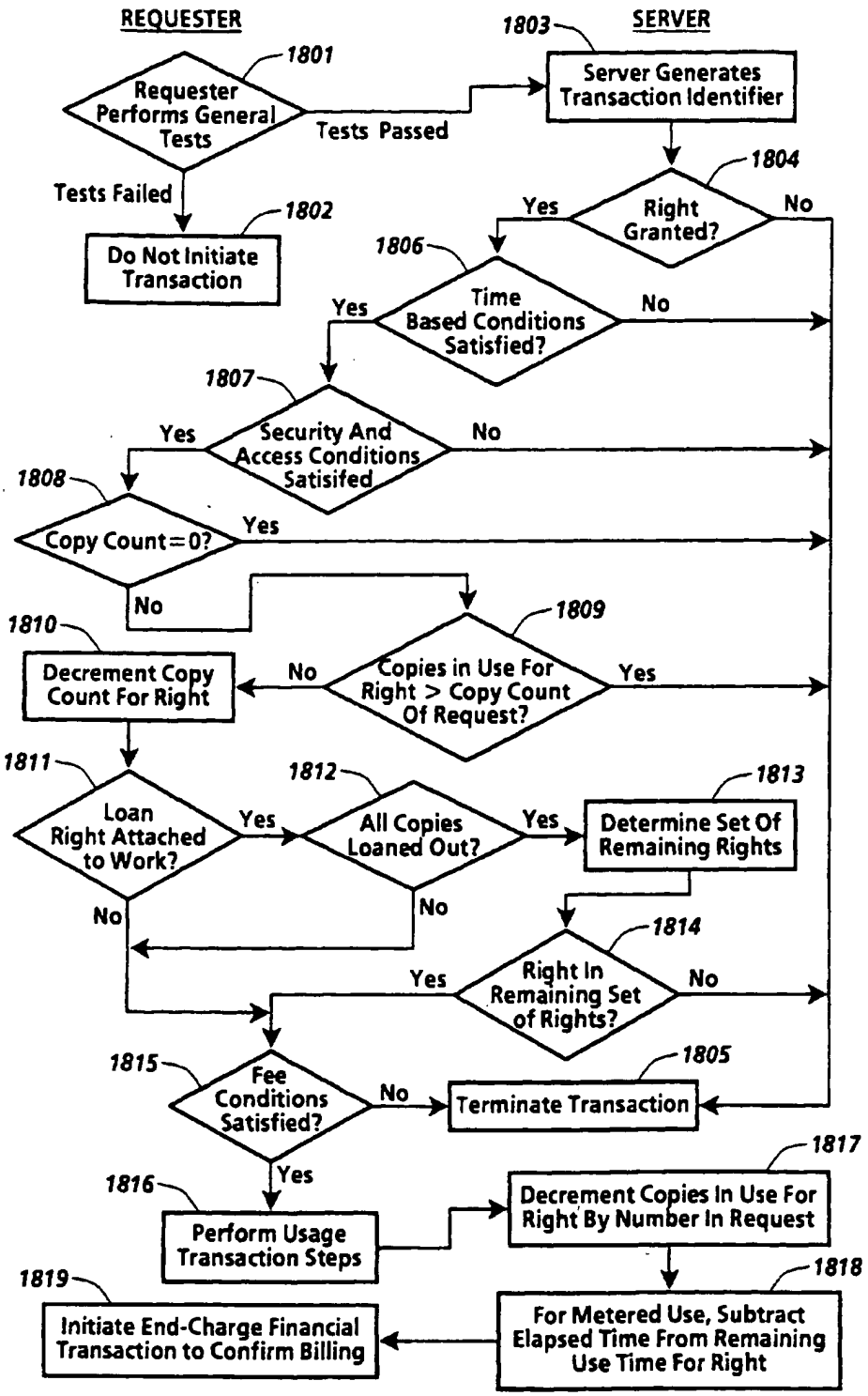


Fig.18

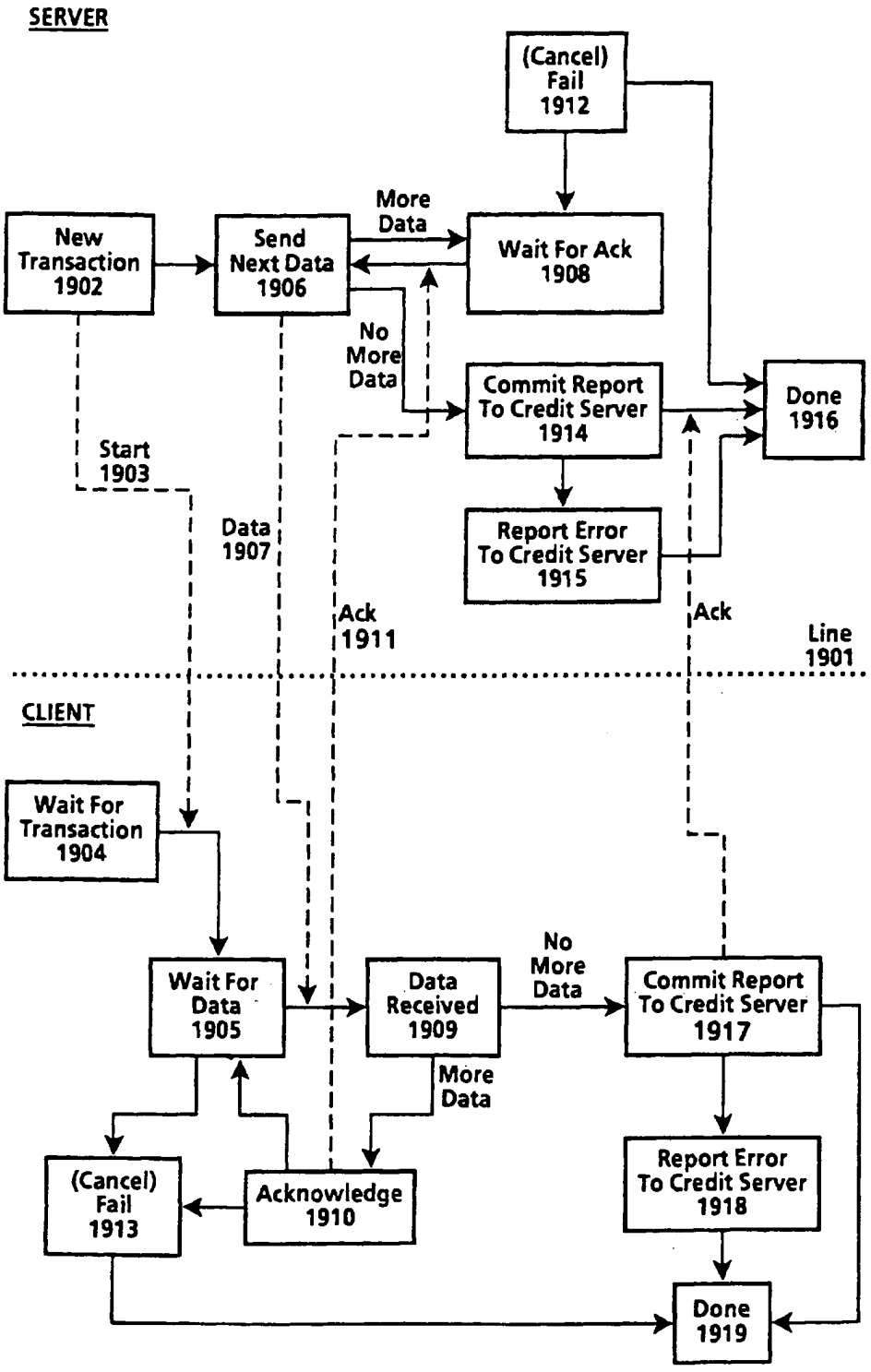


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8414

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 64, line 17 *	1,3,5,6,8	G06F1/00 G06F17/60
A	TRANSACTIONS OF THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS OF JAPAN, vol. E73, no. 7, July 1990 TOKYO JP, pages 1133-1146, XP 000159229 MORI ET AL. 'SUPERDISTRIBUTION: THE CONCEPT AND THE ARCHITECTURE' * page 1135, left column, line 17 - page 1136, left column, line 40 *	1,3,5,6,8	
A	US-A-5 291 596 (MITA) * the whole document *	1,3,5,6,8	
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 *	1,3,5,6,8	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 244 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**

(21) Application number: **95308417.5**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 334041**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventor: **Stefik, Mark J.**
Woodside, California 94062 (US)

(74) Representative: **Goode, Ian Roy**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works utilizing a usage rights grammar

(57) A system for controlling use and distribution of digital works. The present invention allows the owner of a digital work to attach usage rights (1450) to their work. The usage rights define how the individual digital work may be used and distributed (1451). Instances of usage rights are defined using a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label associated with a predetermined behavior and conditions to exercising the right. The behavior of a usage right is embodied in a predetermined set (1452) of usage transactions steps. The usage transaction steps further check all conditions (1453-1457) which must be satisfied before the right may be exercised. These usage transaction steps define a protocol for requesting the exercise of a right and the carrying out of a right.

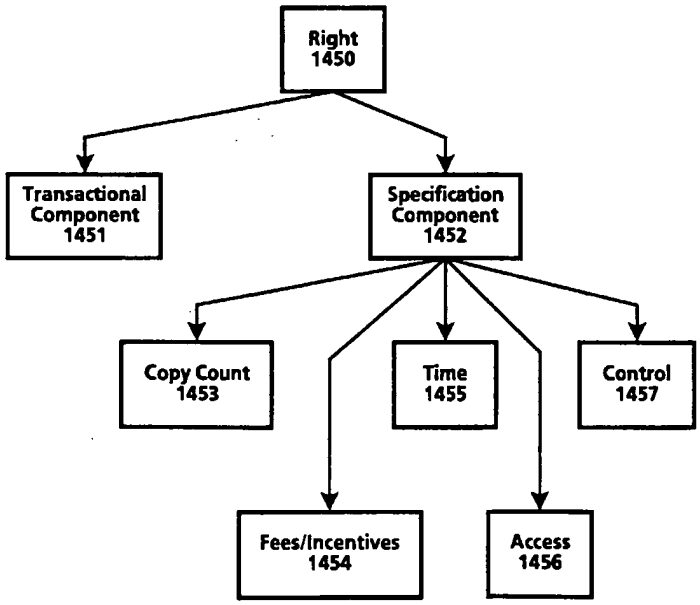


Fig.14

EP 0 715 244 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

A system for controlling use and distribution of digital works is disclosed. A digital work is any written, aural, graphical or video based work that has been translated to or created in a digital form, and which can be recreated using suitable rendering means such as software programs. The present invention allows the owner of a digital work to attach usage rights to their work. The usage rights define how the digital work may be used and distributed. These usage rights become part of the digital work and are always honored.

Instances of usage rights are defined using a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label associated with a predetermined behavior and conditions to exercising the right. For example, a COPY right denotes that a copy of the digital work may be made. A condition to exercising the right is that the requester must pass certain security criteria. Conditions may also be attached to limit the right itself. For example, a LOAN right may be defined so as to limit the duration of which a work may be LOANed.

In the present invention a usage right is comprised of a right code along with the various conditions for exercising the right. Such conditions include a copy-count condition for limiting the number of times a right can be concurrently exercised (e.g. limit the number of copies on loan to some predetermined number), a security class condition for insuring that a repository has an appropriate level of security, access conditions for specifying access tests that must be passed, a time specification for indicating time based constraints for exercising a right and a fee specification for indicating usage fees for the exercise of a right. A digital work may have different versions of a right attached thereto. A version of a right will have the same right code as other versions, but the conditions (and typically the fees) would be different.

Digital works and their attached usage rights are stored in repositories. Digital works are transmitted between repositories. Repositories interact to exchange digital works according to a predetermined set of usage transactions steps. The behavior of a usage right is embodied in a predetermined set of usage transactions steps. The usage transaction steps further check all conditions which must be satisfied before the right may be exercised. These usage transaction steps define a protocol used by the repositories for requesting the exercise of a right and the carrying out of a right.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. As-

suming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository 2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multi-

function" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used for reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such

rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on

a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system. Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repository has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset threshold that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous party. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy

or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a card-sized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a|b|c]" is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces {} are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x^*) is used to indicate a variable length list, that is, a list containing one or more items of type x. The notation $(x)^*$ is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases,

the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticke etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights:= (Rights*)"** define the digital wor rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digita work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right: = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-CodeI Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play:{Player:Player-ID}I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan (Remaining-Rights: Next-Set-of-Rights)] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy Make a new copy of a work
- Transfer Moving a work from one repository to another.
- Loan Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code: = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}I Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders

which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code: [Extract | Embed | Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code: = Install | Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights: = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer | 0 | unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as "forever", then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval : = From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If **"SC: "** is not specified, the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers.

For such access tests, the communication would need to be established and authorization obtained before the right could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: | Incentive:} [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if **Fee:** is specified. Incentives are paid by the revenue-owner to the user if **Incentive:** is specified. If the **Min:** specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the **Max:** specification is given, then there is a maximum fee to be charged per time-spec for its use. When **Fee:** is specified, **Account-ID** identifies the account to which the fee is to be paid. When **Incentive:** is specified, **Account-ID** identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec : = Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates,

and pricing that depends on information that is not available to the repository. All fee specifications can be combined with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the **Max:** field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec: = Call-For-Price "** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. A **Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others. Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The

second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks. Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository- 1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOGIN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOGIN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To sim-

plify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets -- the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the

requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right specific steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work , unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be

played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.

- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for " printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage, such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted , the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the di rectory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not retum the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However,

it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.
- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A distribution system for distributing digital works, said digital works having one or more usage rights attached thereto, said distribution system comprising:

   a grammar for creating instances of usage rights indicating a manner by which a possessor of an associated digital work may transport said associated digital work;

means for creating usage rights from said grammar;
means for attaching created usage rights to a digital work;
a requester repository for accessing digital works, said requester repository having means for generating usage transactions, each said usage transaction specifying a usage right;
a server repository for storing digital works with attached created usage rights, said server repository having means for processing usage transactions from said requester repository to determine if access to a digital work may be granted.

2. The distribution system as recited in Claim 1 wherein said grammar further specifies a default plurality of conditions for an instance of a usage right, wherein said one or more conditions must be satisfied before said usage right may be exercised.

3. The distribution system as recited in Claim 2 wherein said means for creating usage rights from said grammar is further comprised of means for changing said default plurality of conditions for an instance of a usage right.

4. The distribution system as recited in Claim 1 wherein said digital work is a software program.

5. The distribution system as recited in Claim 1 wherein said grammar is further for creating a first version of a usage right having a first set of conditions and a second version of said usage right having a second set of conditions.

6. A computer based system for controlling distribution and use of digital works comprising:

a usage rights grammar for creating instances of usages rights which define how a digital work may be used or distributed, said usage rights grammar comprising a first plurality of grammar elements for defining transport usage rights and a second plurality of grammar elements for defining rendering usage rights;
means for attaching usage rights to digital works;
a plurality of repositories for storing and exchanging digital works, each of said plurality of repositories comprising :
means for storing digital works and their attached usage rights;
transaction processing means having a requester mode of operation for requesting access to a requested digital work, said request specifying a usage right, and a server mode of operation for processing requests to access said requested digital work based on said usage right specified in said request and the usage rights attached to said requested digital work; and
a coupling means for coupling to another of said plurality of repositories across a communications medium.

7. The computer based system for controlling distribution and use of digital works as recited in Claim 6 wherein said first plurality of grammar elements is comprised of:

a loan grammar element for enabling a digital work to be loaned to another repository;
a copy grammar element for enabling a copy of a digital work to be made and transported to another repository; and
a transfer grammar element for enabling a digital work to be transferred to another repository.

8. The computer based system for controlling distribution and use of digital works as recited in Claim 6 or Claim 7 wherein said second plurality of grammar elements is comprised of:

a play grammar element for enabling a digital work to be rendered on a specified class of player device; and
a print grammar element for enabling a digital work to be printed on a specified class of printer device.

9. The computer based system for controlling distribution and use of digital works as recited in any one of Claims 6 to 8 wherein said grammar comprises one or more further pluralities of grammar elements, for defining file management usage rights, for enabling a digital work to be used in the creation of a new digital work,for enabling the secure installation and uninstallation of digital works comprising of software programs, or for providing a set of creator specified conditions which must be satisfied for each instantiation of a usage right defined by a grammar element.

10. A method for controlling distribution and use of digital works comprising the steps of:

a) creating a set of usage rights from a usage rights grammar, each of said usage rights defining a specific instance of how a digital work may be used or distributed, each of said usage rights specifying one or more conditions which must be satisfied in order for said usage right to be exercised;
b) attaching said set of usage rights to a digital work;
c) storing said digital work and its attached usage rights in a first repository;
d) a second repository initiating a request to access said digital work in said first repository, said request specifying a usage right;
e) said first repository receiving said request from said second repository;
f) said first repository determining if said specified usage right is attached to said digital work;
g) said first repository denying access to said digital work if said identified usage right is not attached to said digital work;
h) if said identified usage right is attached to said digital work, said first repository determining if conditions specified by said usage right are satisfied;
i) if said conditions are not satisfied, said first repository denying access to said digital work;
j) if said conditions are satisfied, said first repository transmitting said digital work to said second repository.

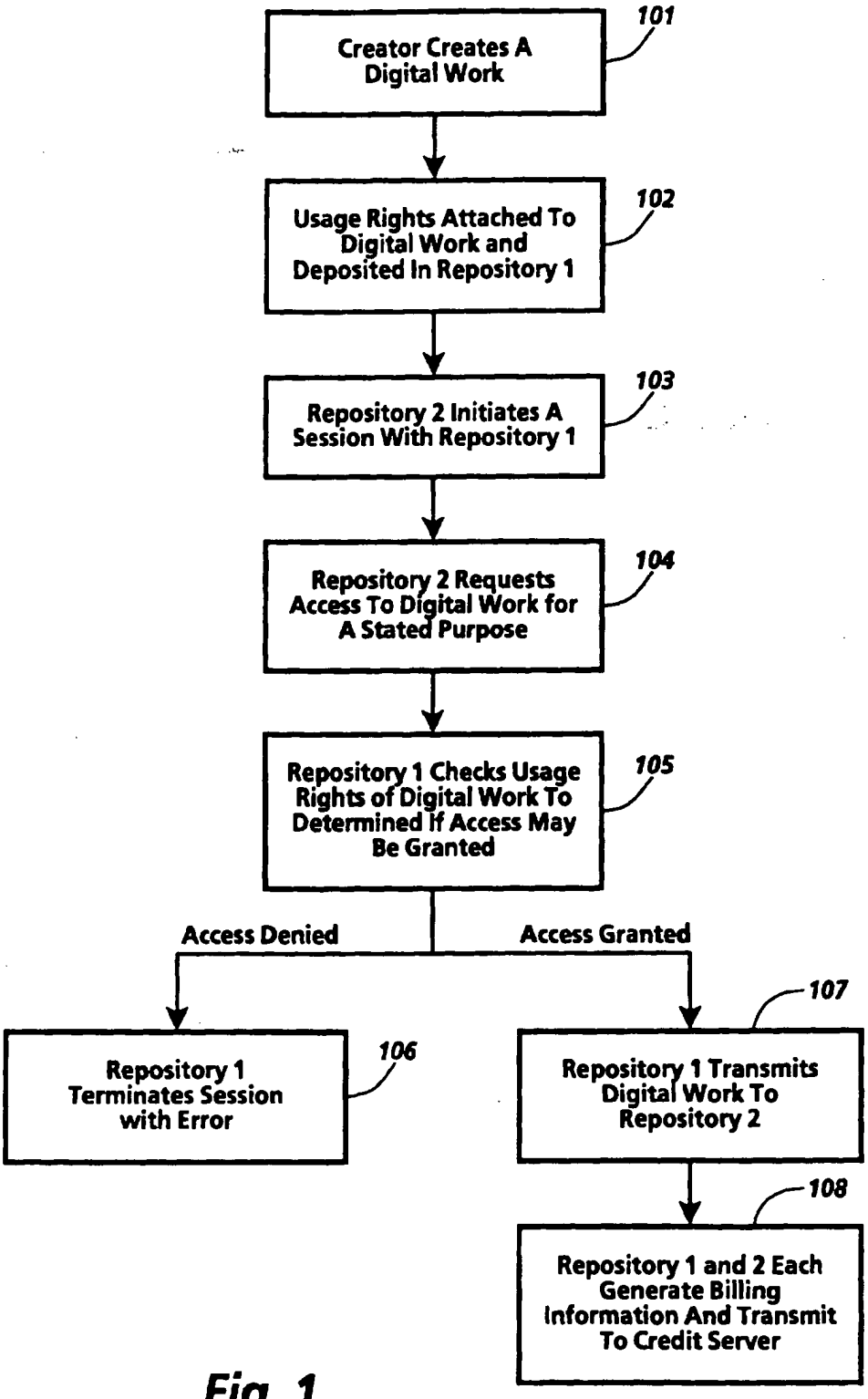


Fig. 1

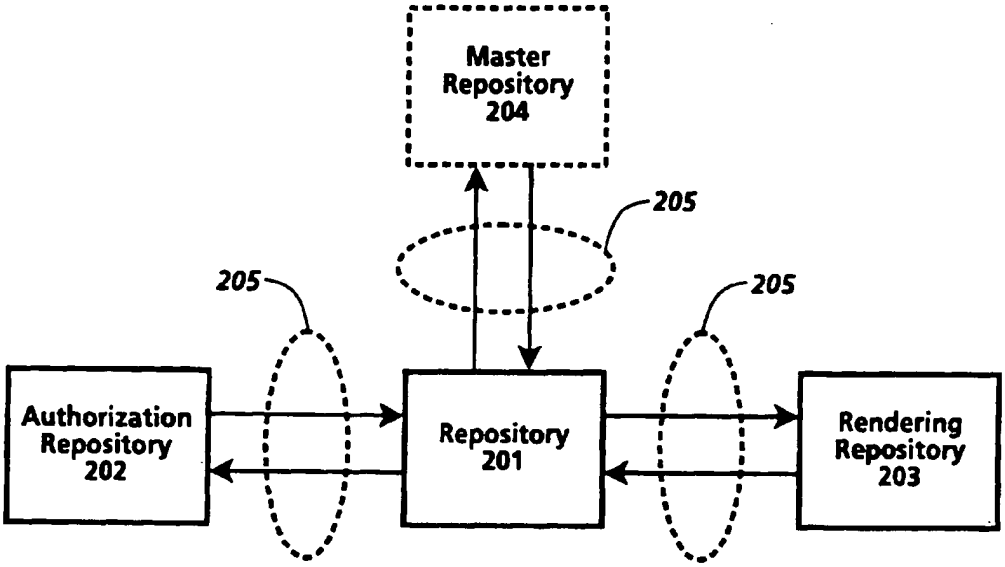


Fig. 2

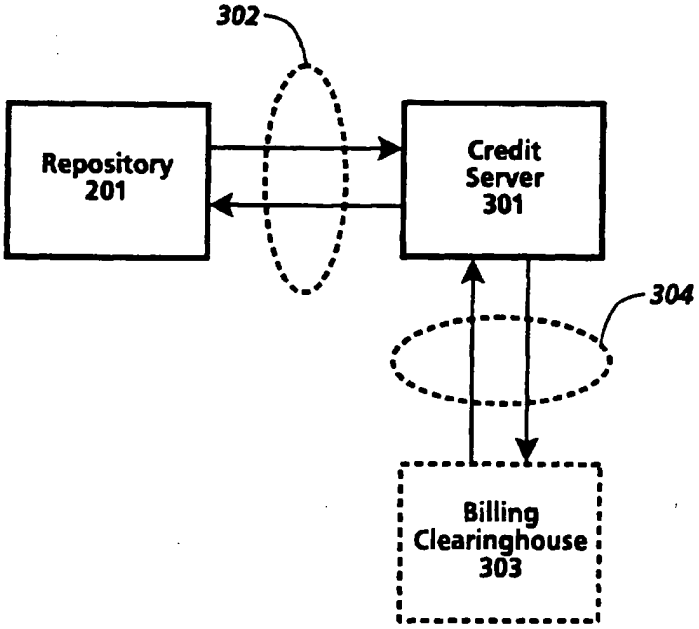


Fig. 3

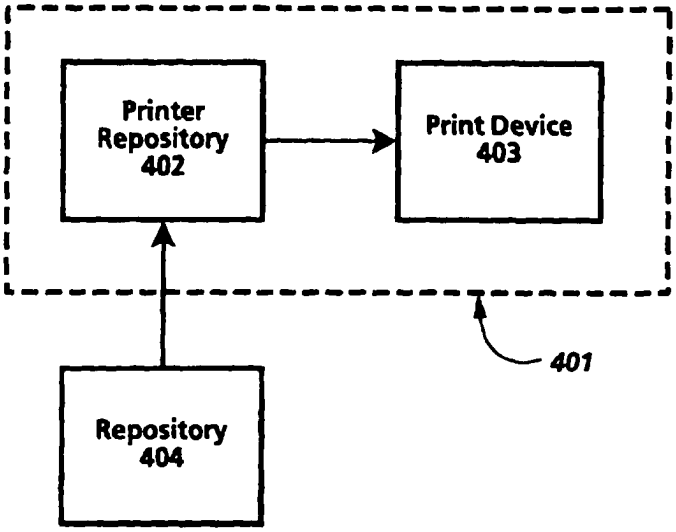


Fig. 4a

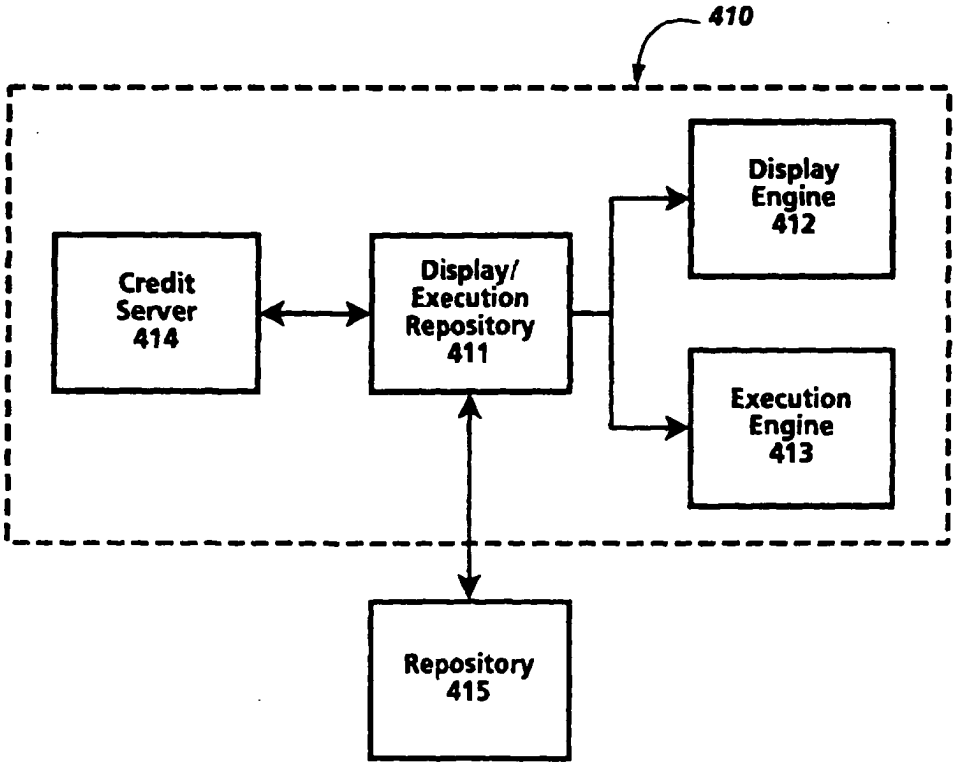


Fig. 4b

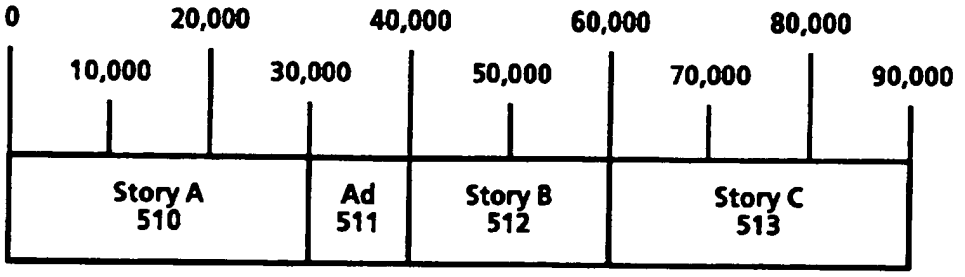


Fig. 5

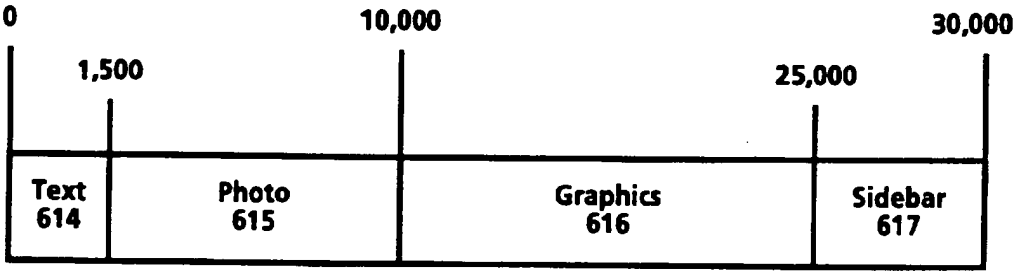


Fig. 6

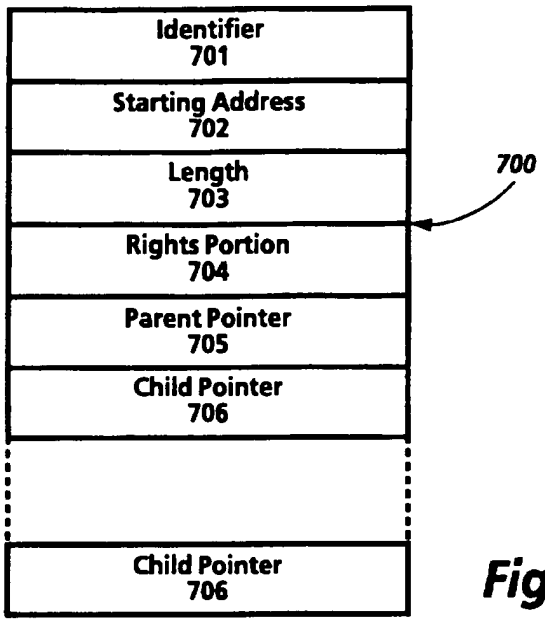


Fig. 7

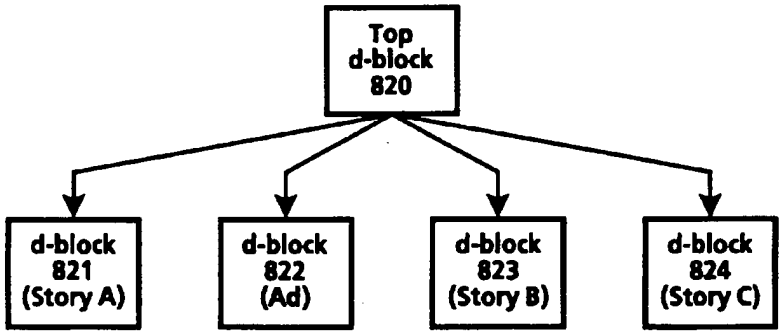


Fig. 8

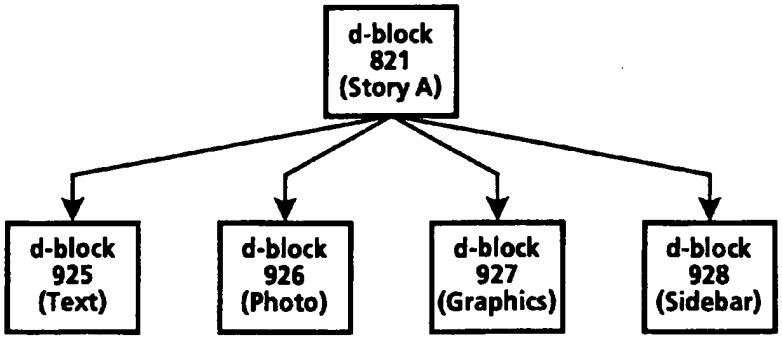


Fig. 9

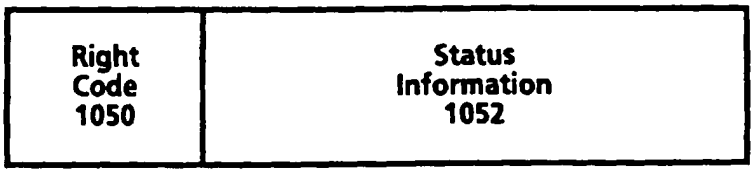


Fig.10

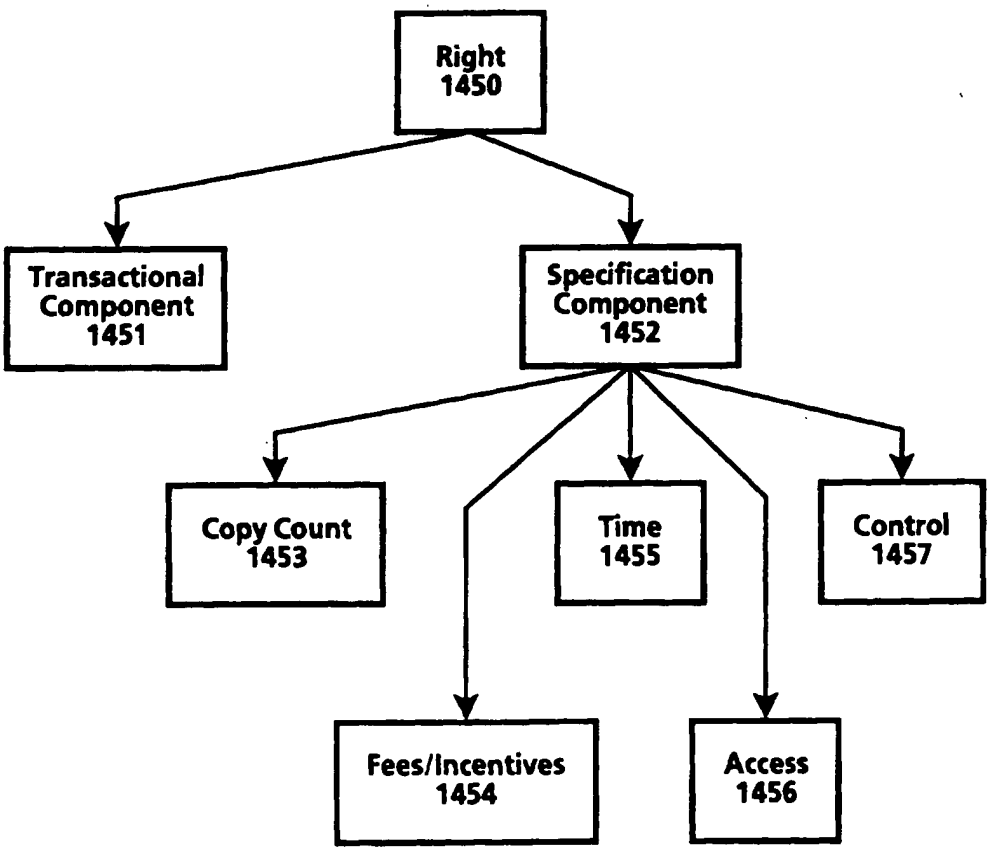


Fig.14

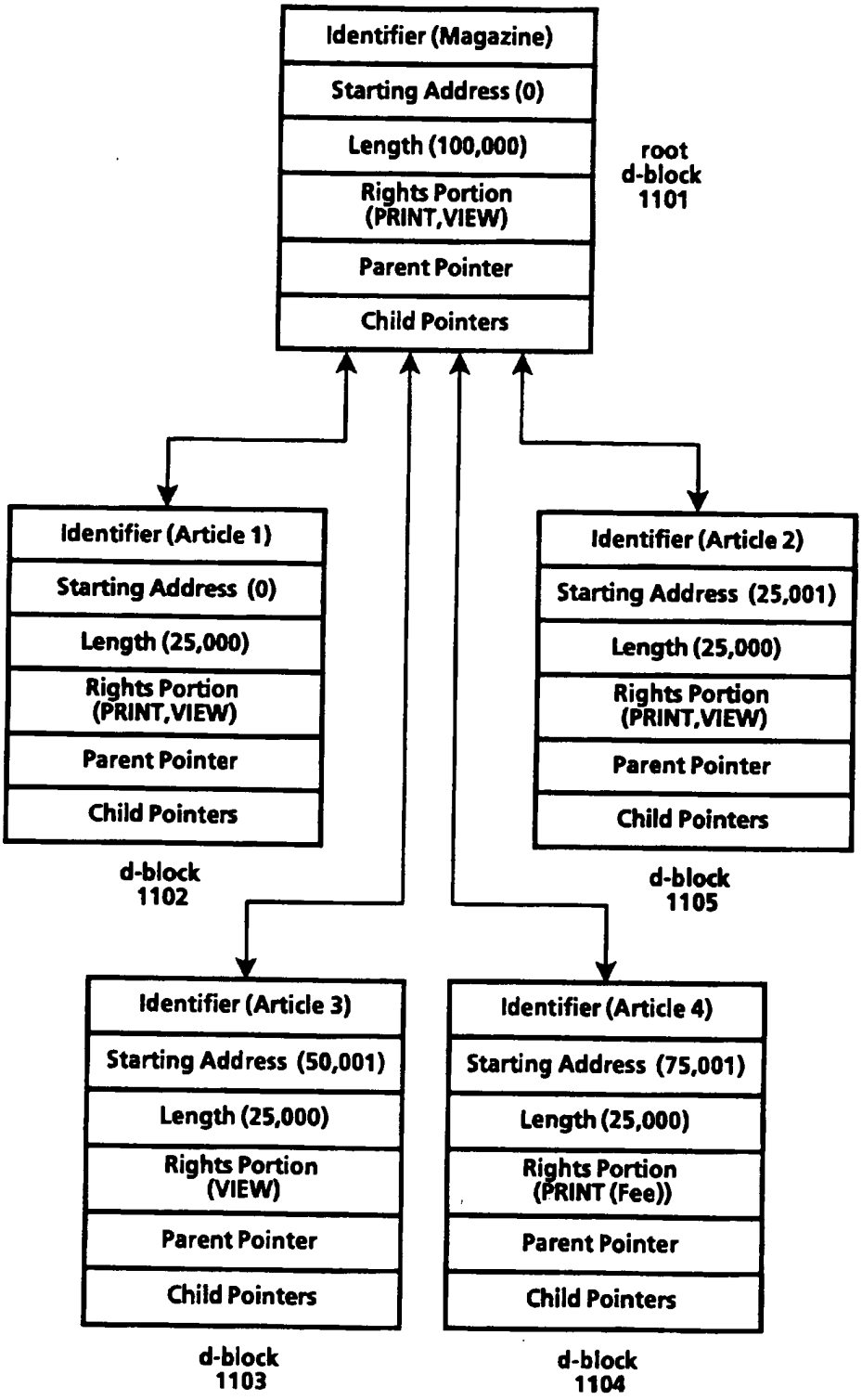


Fig.11

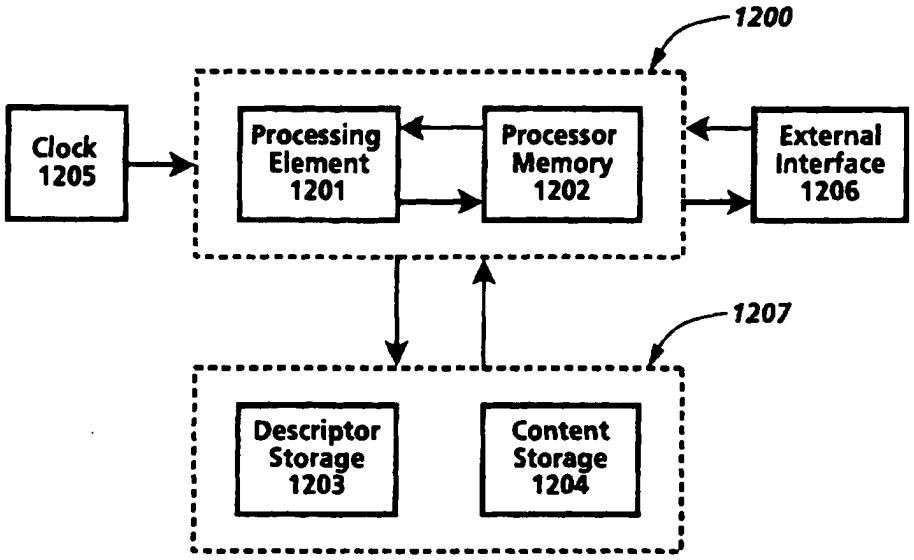


Fig.12

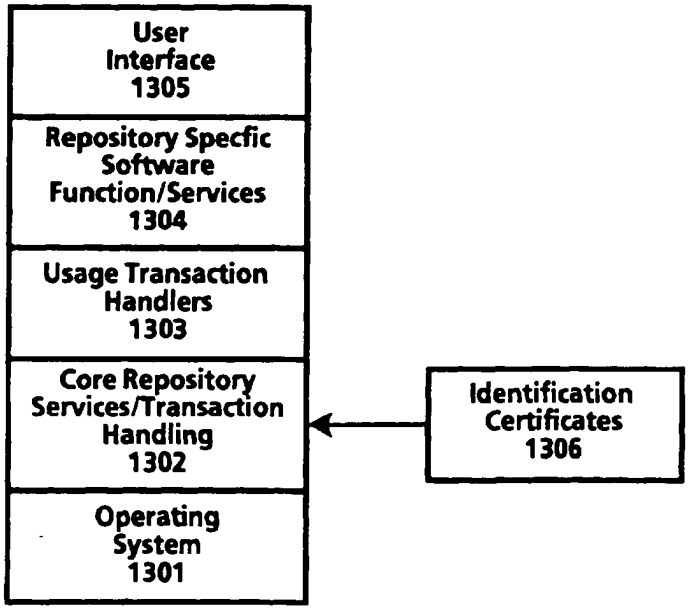


Fig.13

1501 ~ **Digital Work Rights**: = (Rights*)
1502 ~ **Right** : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec } {Access-Spec} {Fee-Spec})
1503 ~ **Right-Code** : = Render-Code | Transport-Code | File-Management-Code| Derivative-Works- Code | Configuration-Code
1504 ~ **Render-Code** : = [Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]
1505 ~ **Transport-Code** : = [Copy | Transfer | Loan {Remaining-Rights: Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}
1506 ~ **File-Management-Code** : = Backup {Back-Up-Copy-Rights: Next-Set-of-Rights} | Restore | Delete | Folder | Directory {Name: Hide-Local | Hide-Remote} {Parts: Hide-Local | Hide-Remote}
1507 ~ **Derivative-Works-Code** : = [Extract | Embed | Edit{Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}
1508 ~ **Configuration-Code** : = Install | Uninstall
1509 ~ **Next-Set-of-Rights** : = {(Add: Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)}{(Keep: Set-Of-Rights)}
1510 ~ **Copy-Count** : = (Copies:positive-integer | 0 | Unlimited)
1511 ~ **Control-Spec** : = (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})
1512 ~ **Time-Spec** : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)
1513 ~ **Fixed-Interval** : = From: Start-Time
1514 ~ **Sliding-Interval** : = Interval: Use-Duration
1515 ~ **Meter-Time**: = Time-Remaining: Remaining-Use
1516 ~ **Access-Spec** : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})
1517 ~ **Fee-Spec**: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec
1518 ~ **Scheduled-Discount**: = Scheduled-Discount: (Scheduled-Discount: (Time-Spec Percentage)*)
1519 ~ **Regular-Fee-Spec** : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec}{Max: Money-Unit Per: Time-Spec} To: Account-ID)
1520 ~ **Per-Use-Spec**: = Per-Use: Money-unit
1521 ~ **Metered-Rate-Spec** : = Metered: Money-Unit Per: Time-Spec
1522 ~ **Best-Price-Spec** : = Best-Price: Money-unit Max: Money-unit
1523 ~ **Call-For-Price-Spec** : = Call-For -Price
1524 ~ **Scheduled-Fee-Spec**: = (Schedule: (Time-Spec Regular-Fee-Spec)*)
1525 ~ **Markup-Spec**: = Markup: percentage To: Account-ID

Fig.15

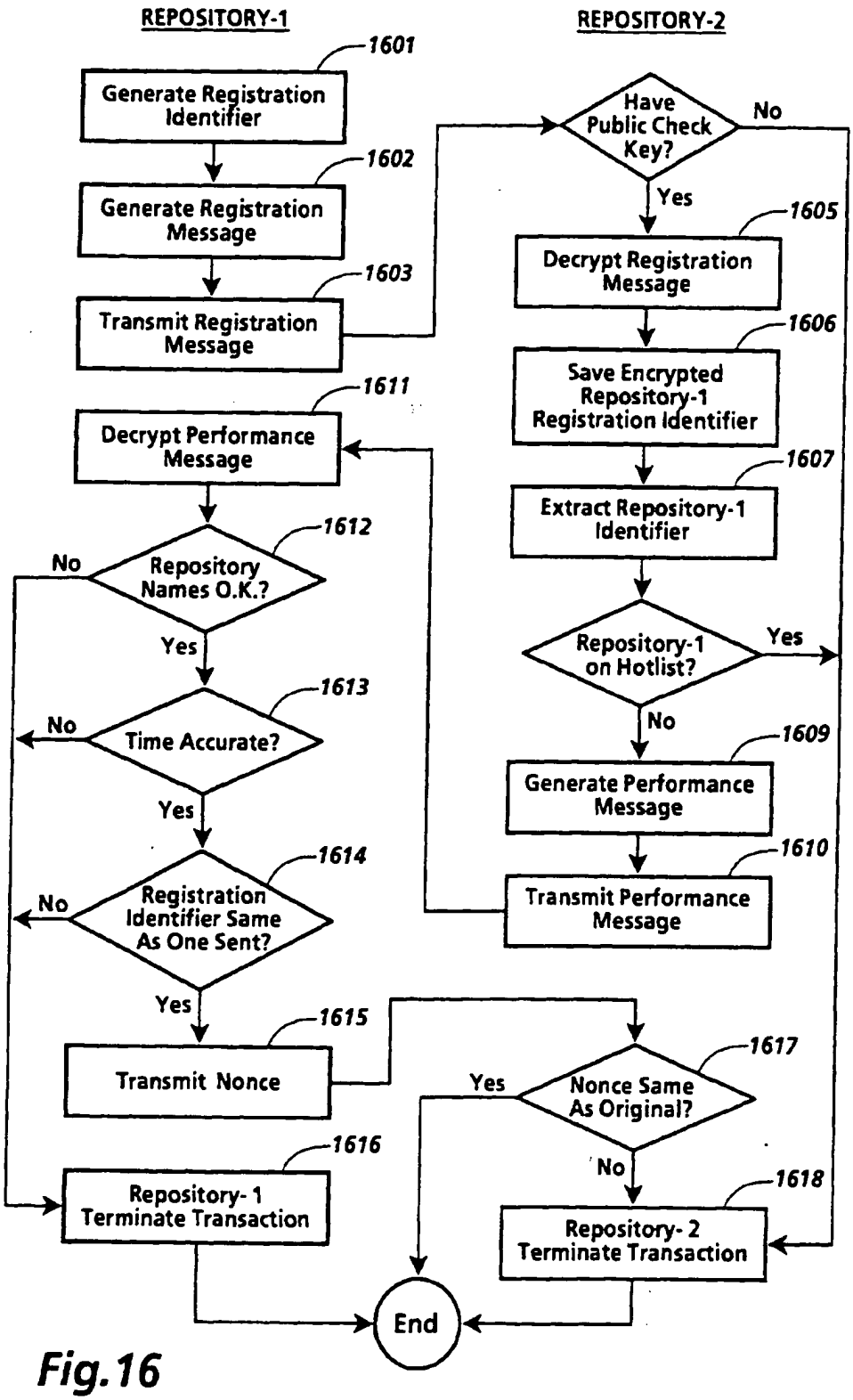


Fig.16

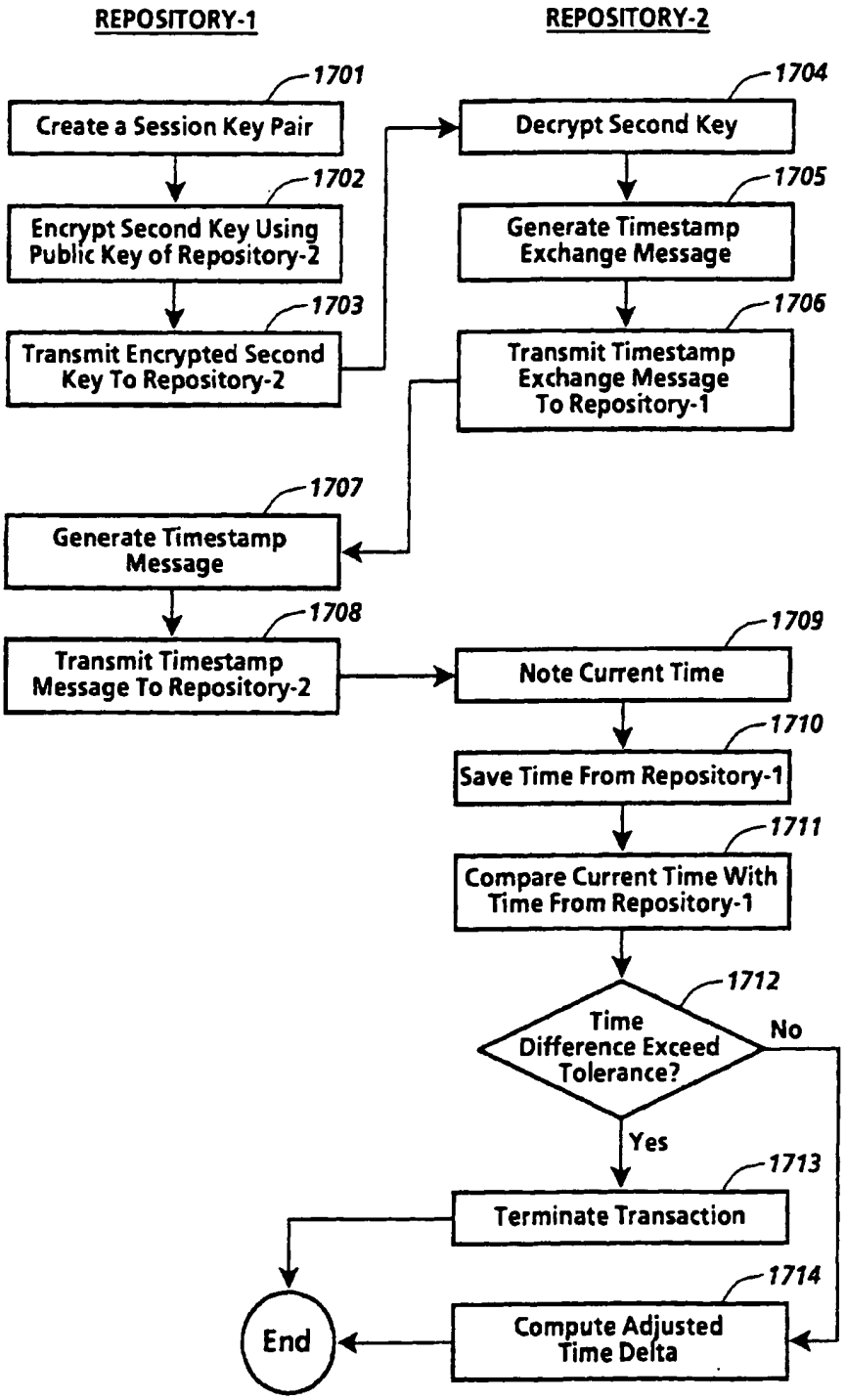


Fig.17

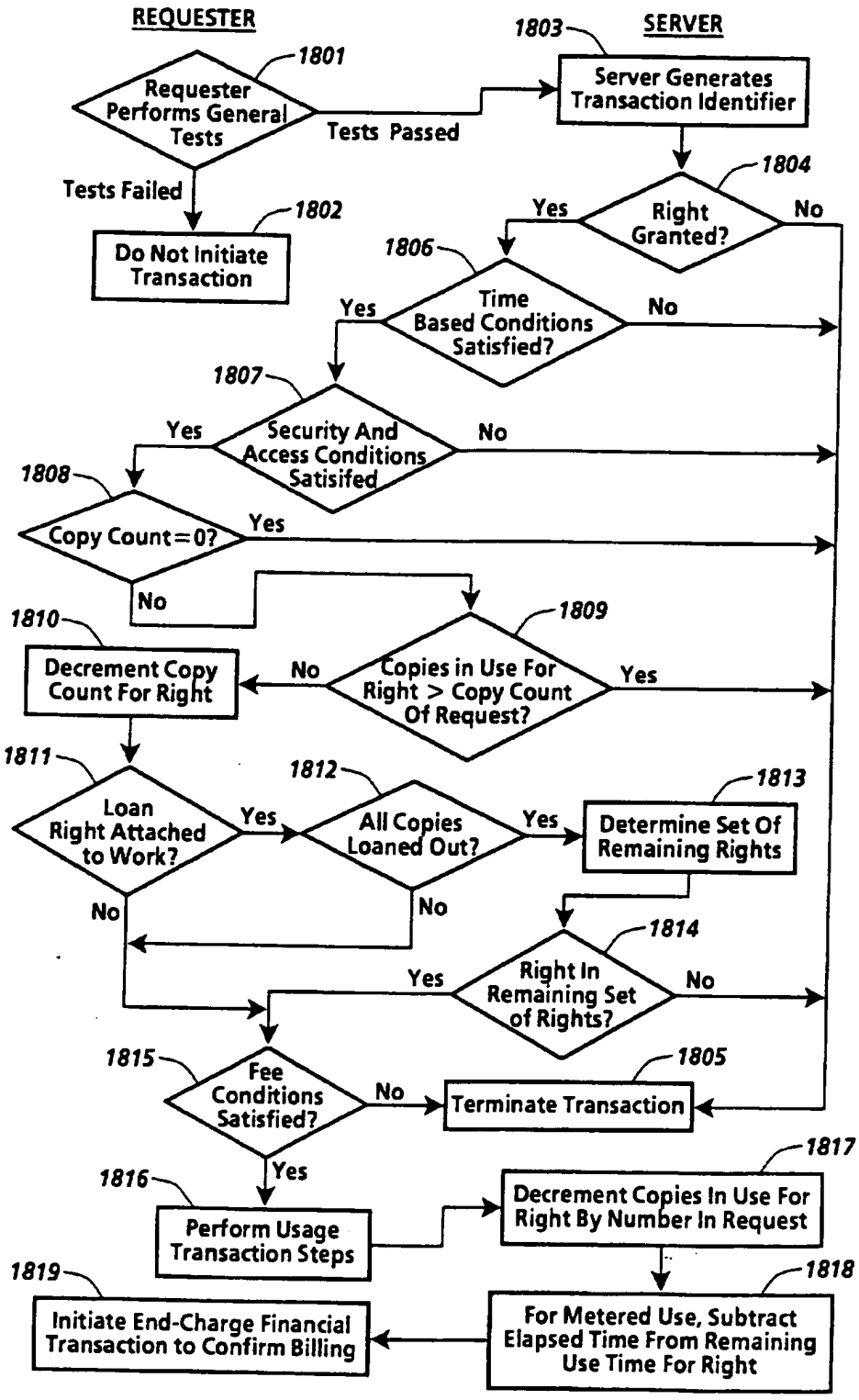


Fig.18

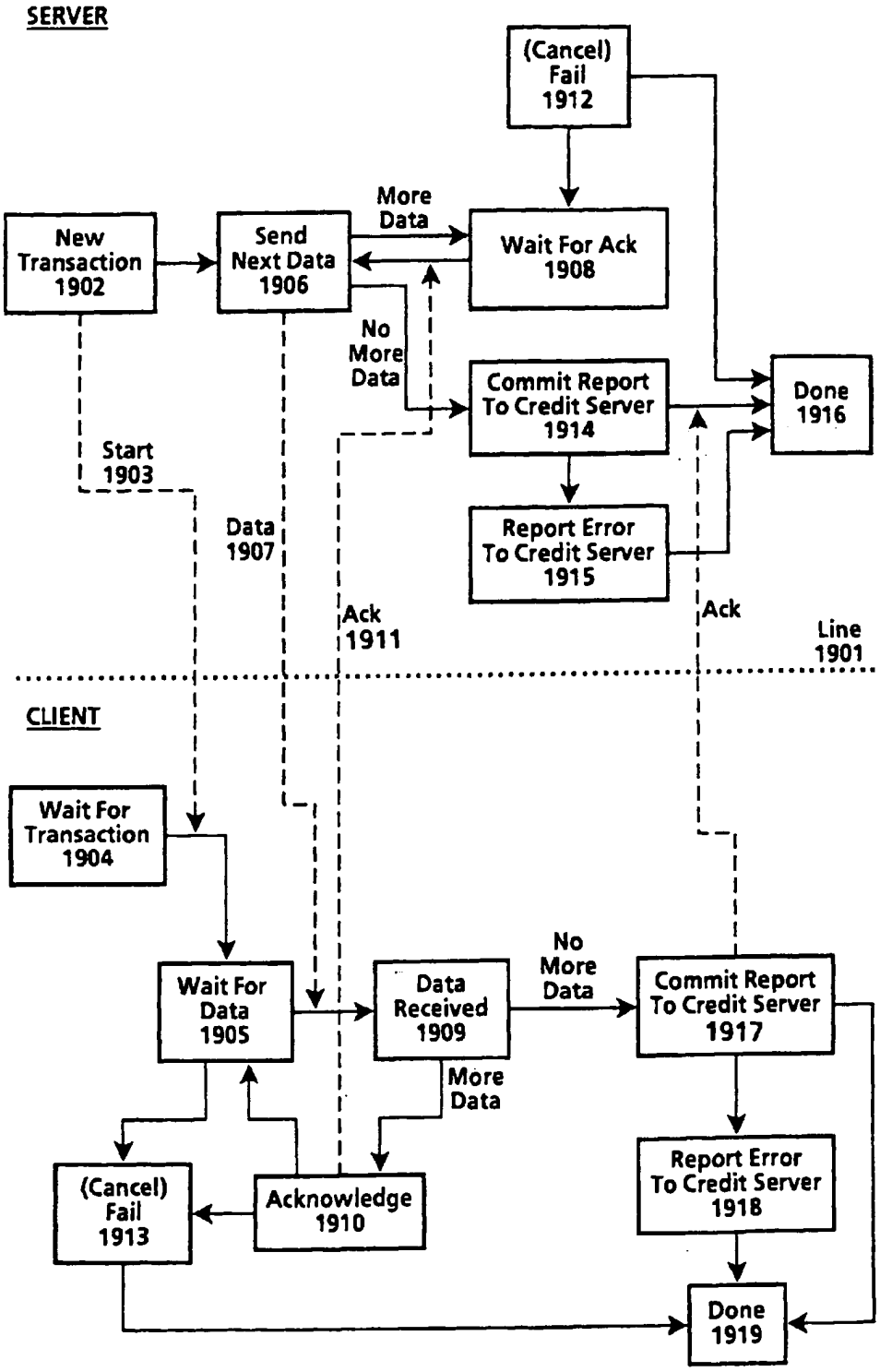


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8417

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 80, line 19; figures 1-43 * ---	1,6,10	G06F1/00
A	US-A-5 291 596 (MIITA) * the whole document * ---	1,6,10	
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 * -----	1,6,10	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 715 245 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
05.06.1996 Bulletin 1996/23

(51) Int Cl.[6]: **G06F 1/00**

(21) Application number: **95308420.9**

(22) Date of filing: **23.11.1995**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **23.11.1994 US 344042**

(71) Applicant: **XEROX CORPORATION**
Rochester New York 14644 (US)

(72) Inventors:
- **Stefik, Mark J.**
  Woodside, California 94062 (US)
- **Casey, Michalene M.**
  Morgan Hill, California 95037 (US)

(74) Representative: **Goode, Ian Roy**
Rank Xerox Ltd
Patent Department
Parkway
Marlow Buckinghamshire SL7 1YL (GB)

(54) System for controlling the distribution and use of digital works

(57) A system for controlling use and distribution of digital works, in which the owner of a digital work (101) attaches usage rights (102) to that work. Usage rights are granted by the "owner" of a digital work to "buyers" of the digital work. The usage rights define how a digital work may be used and further distributed by the buyer. Each right has associated with it certain optional specifications which outline the conditions and fees upon which the right may be exercised. Digital works are stored in a repository. A repository will process each request (103,104) to access a digital work by examining the corresponding usage rights (105). Digital work playback devices, coupled to the repository containing the work, are used to play, display or print the work. Access to digital works for the purposes of transporting between repositories (e.g. copying, borrowing or transfer) is carried out using a digital work transport protocol. Access to digital works for the purposes of replay by a digital work playback device(e.g. printing, displaying or executing) is carried out using a digital work playback protocol. Access is denied (106) or granted (107) depending whether the requesting repository has the required usage rights.

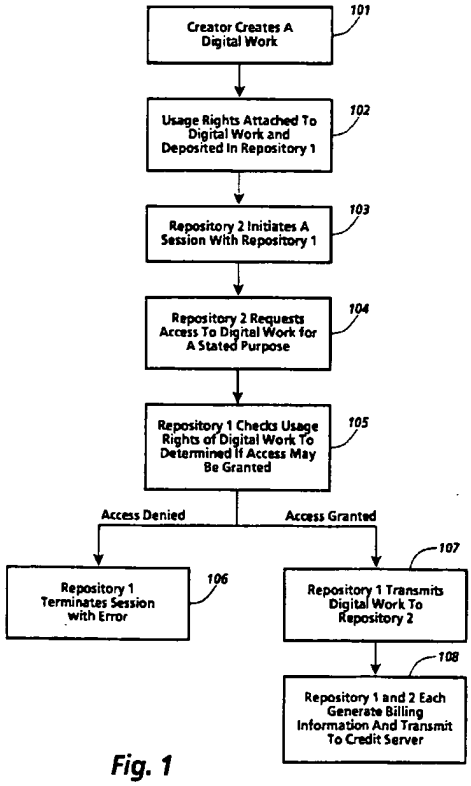


Fig. 1

EP 0 715 245 A1

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to the field of distribution and usage rights enforcement for digitally encoded works.

A fundamental issue facing the publishing and information industries as they consider electronic publishing is how to prevent the unauthorized and unaccounted distribution or usage of electronically published materials. Electronically published materials are typically distributed in a digital form and recreated on a computer based system having the capability to recreate the materials. Audio and video recordings, software, books and multimedia works are all being electronically published. Companies in these industries receive royalties for each accounted for delivery of the materials, e.g. the sale of an audio CD at a retail outlet. Any unaccounted distribution of a work results in an unpaid royalty (e.g. copying the audio recording CD to another digital medium.)

The ease in which electronically published works can be "perfectly" reproduced and distributed is a major concern. The transmission of digital works over networks is commonplace. One such widely used network is the Internet. The Internet is a widespread network facility by which computer users in many universities, corporations and government entities communicate and trade ideas and information. Computer bulletin boards found on the Internet and commercial networks such as CompuServ and Prodigy allow for the posting and retrieving of digital information. Information services such as Dialog and LEXIS/NEXIS provide databases of current information on a wide variety of topics. Another factor which will exacerbate the situation is the development and expansion of the National Information Infrastructure (the NII). It is anticipated that, as the NII grows, the transmission of digital works over networks will increase many times over. It would be desirable to utilize the NII for distribution of digital works without the fear of widespread unauthorized copying.

The most straightforward way to curb unaccounted distribution is to prevent unauthorized copying and transmission. For existing materials that are distributed in digital form, various safeguards are used. In the case of software, copy protection schemes which limit the number of copies that can be made or which corrupt the output when copying is detected have been employed. Another scheme causes software to become disabled after a predetermined period of time has lapsed. A technique used for workstation based software is to require that a special hardware device must be present on the workstation in order for the software to run, e.g., see US-A-4,932,054 entitled "Method and Apparatus for Protecting Computer Software Utilizing Coded Filter Network in Conjunction with an Active Coded Hardware Device." Such devices are provided with the software and are commonly referred to as dongles.

Yet another scheme is to distribute software, but which requires a "key" to enable its use. This is employed in distribution schemes where "demos" of the software are provided on a medium along with the entire product. The demos can be freely used, but in order to use the actual product, the key must be purchased. These schemes do not hinder copying of the software once the key is initially purchased.

It is an object of the present invention to provide an improved system and method for controlling the use and distribution of digital works.

The invention accordingly provides a system and method as claimed in the accompanying claims.

A system for controlling use and distribution of digital works is disclosed. A digital work is any written, aural, graphical or video based work including computer programs that has been translated to or created in a digital form, and which can be recreated using suitable rendering means such as software programs. The present invention allows the owner of a digital work to attach usage rights to the work. The usage rights for the work define how it may be used and distributed. Digital works and their usage rights are stored in a secure repository. Digital works may only be accessed by other secure repositories.

Usage rights for a digital work are embodied in a flexible and extensible usage rights grammar. Conceptually, a right in the usage rights grammar is a label attached to a predetermined behavior and conditions to exercising the right. For example, a COPY right denotes that a copy of the digital work may be made. A condition to exercising the right is the requester must pass certain security criteria. Conditions may also be attached to limit the right itself. For example, a LOAN right may be defined so as to limit the duration of which a work may be LOANed. Conditions may also include requirements that fees be paid.

A repository is comprised of a storage means for storing a digital work and its attached usage rights, an external interface for receiving and transmitting data, a processor and a clock. A repository has two primary operating modes, a server mode and a requester mode. When operating in a server mode, the repository is responding to requests to access digital works. When operating in requester mode, the repository is requesting access to a digital work.

Generally, a repository will process each request to access a digital work by examining the work's usage rights. For example, in a request to make a copy of a digital work, the digital work is examined to see if rights have been granted which would allow copies to be given out. If such a right has been granted, then conditions to exercise of the right are checked (e.g. a right to make 2 copies). If conditions associated with the right are satisfied, the copy can be made. Before transporting the digital work, any specified changes to the set of usage rights in the copy are attached to the copy of the digital work.

Repositories communicate utilizing a set of repository transactions. The repository transactions embody a set of

protocols for establishing secure sessions connections between repositories, and for processing access requests to the digital works.

Digital works are recreated on rendering systems. A rendering system is comprised of at least a rendering repository and a rendering device (e.g. a printer, display or audio system.) Rendering systems are internally secure. Access to digital works not contained within the rendering repository is accomplished via repository transactions with an external repository containing the desired digital work.

A system and method in accordance with the invention will now be described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a flowchart illustrating a simple instantiation of the operation of the currently preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the various repository types and the repository transaction flow between them in the currently preferred embodiment of the present invention.

Figure 3 is a block diagram of a repository coupled with a credit server in the currently preferred embodiment of the present invention.

Figures 4a and 4b are examples of rendering systems as may be utilized in the currently preferred embodiment of the present invention.

Figure 5 illustrates a contents file layout for a digital work as may be utilized in the currently preferred embodiment of the present invention.

Figure 6 illustrates a contents file layout for an individual digital work of the digital work of Figure 5 as may be utilized in the currently preferred embodiment of the present invention.

Figure 7 illustrates the components of a description block of the currently preferred embodiment of the present invention.

Figure 8 illustrates a description tree for the contents file layout of the digital work illustrated in Figure 5.

Figure 9 illustrates a portion of a description tree corresponding to the individual digital work illustrated in Figure 6.

Figure 10 illustrates a layout for the rights portion of a description block as may be utilized in the currently preferred embodiment of the present invention.

Figure 11 is a description tree wherein certain d-blocks have PRINT usage rights and is used to illustrate "strict" and "lenient" rules for resolving usage rights conflicts.

Figure 12 is a block diagram of the hardware components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 13 is a block diagram of the functional (logical) components of a repository as are utilized in the currently preferred embodiment of the present invention.

Figure 14 is diagram illustrating the basic components of a usage right in the currently preferred embodiment of the present invention.

Figure 15 lists the usage rights grammar of the currently preferred embodiment of the present invention.

Figure 16 is a flowchart illustrating the steps of certificate delivery, hotlist checking and performance testing as performed in a registration transaction as may be performed in the currently preferred embodiment of the present invention.

Figure 17 is a flowchart illustrating the steps of session information exchange and clock synchronization as may be performed in the currently preferred embodiment of the present invention, after each repository in the registration transaction has successfully completed the steps described in Figure 16.

Figure 18 is a flowchart illustrating the basic flow for a usage transaction, including the common opening and closing step, as may be performed in the currently preferred embodiment of the present invention.

Figure 19 is a state diagram of server and client repositories in accordance with a transport protocol followed when moving a digital work from the server to the client repositories, as may be performed in the currently preferred embodiment of the present invention.

OVERVIEW

A system for controlling use and distribution of digital works is disclosed. The present invention is directed to supporting commercial transactions involving digital works.

Herein the terms "digital work", "work" and "content" refer to any work that has been reduced to a digital representation. This would include any audio, video, text, or multimedia work and any accompanying interpreter (e.g. software) that may be required for recreating the work. The term composite work refers to a digital work comprised of a collection of other digital works. The term "usage rights" or "rights" is a term which refers to rights granted to a recipient of a digital work. Generally, these rights define how a digital work can be used and if it can be further distributed. Each usage right may have one or more specified conditions which must be satisfied before the right may be exercised.

Figure 1 is a high level flowchart omitting various details but which demonstrates the basic operation of the present

invention. Referring to Figure 1, a creator creates a digital work, step 101. The creator will then determine appropriate usage rights and fees, attach them to the digital work, and store them in Repository 1, step 102. The determination of appropriate usage rights and fees will depend on various economic factors. The digital work remains securely in Repository 1 until a request for access is received. The request for access begins with a session initiation by another repository. Here a Repository 2 initiates a session with Repository 1, step 103. As will be described in greater detail below, this session initiation includes steps which helps to insure that the respective repositories are trustworthy. Assuming that a session can be established, Repository 2 may then request access to the Digital Work for a stated purpose, step 104. The purpose may be, for example, to print the digital work or to obtain a copy of the digital work. The purpose will correspond to a specific usage right. In any event, Repository 1 checks the usage rights associated with the digital work to determine if the access to the digital work may be granted, step 105. The check of the usage rights essentially involves a determination of whether a right associated with the access request has been attached to the digital work and if all conditions associated with the right are satisfied. If the access is denied, repository 1 terminates the session with an error message, step 106. If access is granted, repository 1 transmits the digital work to repository 2, step 107. Once the digital work has been transmitted to repository 2, repository 1 and 2 each generate billing information for the access which is transmitted to a credit server, step 108. Such double billing reporting is done to insure against attempts to circumvent the billing process.

Figure 2 illustrates the basic interactions between repository types in the present invention. As will become apparent from Figure 2, the various repository types will serve different functions. It is fundamental that repositories will share a core set of functionality which will enable secure and trusted communications. Referring to Figure 2, a repository 201 represents the general instance of a repository. The repository 201 has two modes of operation; a server mode and a requester mode. When in the server mode, the repository will be receiving and processing access requests to digital works. When in the requester mode, the repository will be initiating requests to access digital works. Repository 201 is general in the sense that its primary purpose is as an exchange medium for digital works. During the course of operation, the repository 201 may communicate with a plurality of other repositories, namely authorization repository 202, rendering repository 203 and master repository 204. Communication between repositories occurs utilizing a repository transaction protocol 205.

Communication with an authorization repository 202 may occur when a digital work being accessed has a condition requiring an authorization. Conceptually, an authorization is a digital certificate such that possession of the certificate is required to gain access to the digital work. An authorization is itself a digital work that can be moved between repositories and subjected to fees and usage rights conditions. An authorization may be required by both repositories involved in an access to a digital work.

Communication with a rendering repository 203 occurs in connection with the rendering of a digital work. As will be described in greater detail below, a rendering repository is coupled with a rendering device (e.g. a printer device) to comprise a rendering system.

Communication with a master repository 205 occurs in connection with obtaining an identification certificate. Identification certificates are the means by which a repository is identified as "trustworthy". The use of identification certificates is described below with respect to the registration transaction.

Figure 3 illustrates the repository 201 coupled to a credit server 301. The credit server 301 is a device which accumulates billing information for the repository 201. The credit server 301 communicates with repository 201 via billing transactions 302 to record billing transactions. Billing transactions are reported to a billing clearinghouse 303 by the credit server 301 on a periodic basis. The credit server 301 communicates to the billing clearinghouse 303 via clearinghouse transactions 304. The clearinghouse transactions 304 enable a secure and encrypted transmission of information to the billing clearinghouse 303.

RENDERING SYSTEMS

A rendering system is generally defined as a system comprising a repository and a rendering device which can render a digital work into its desired form. Examples of a rendering system may be a computer system, a digital audio system, or a printer. A rendering system has the same security features as a repository. The coupling of a rendering repository with the rendering device may occur in a manner suitable for the type of rendering device.

Figure 4a illustrates a printer as an example of a rendering system. Referring to Figure 4, printer system 401 has contained therein a printer repository 402 and a print device 403. It should be noted that the the dashed line defining printer system 401 defines a secure system boundary. Communications within the boundary are assumed to be secure. Depending on the security level, the boundary also represents a barrier intended to provide physical integrity. The printer repository 402 is an instantiation of the rendering repository 205 of Figure 2. The printer repository 402 will in some instances contain an ephemeral copy of a digital work which remains until it is printed out by the print engine 403. In other instances, the printer repository 402 may contain digital works such as fonts, which will remain and can be billed based on use. This design assures that all communication lines between printers and printing devices are

encrypted, unless they are within a physically secure boundary. This design feature eliminates a potential "fault" point through which the digital work could be improperly obtained. The printer device 403 represents the printer components used to create the printed output.

Also illustrated in Figure 4a is the repository 404. The repository 404 is coupled to the printer repository 402. The repository 404 represents an external repository which contains digital works.

Figure 4b is an example of a computer system as a rendering system. A computer system may constitute a "multi-function" device since it may execute digital works (e.g. software programs) and display digital works (e.g. a digitized photograph). Logically, each rendering device can be viewed as having its own repository, although only one physical repository is needed. Referring to Figure 4b, a computer system 410 has contained therein a display/execution repository 411. The display/execution repository 411 is coupled to display device, 412 and execution device 413. The dashed box surrounding the computer system 410 represents a security boundary within which communications are assumed to be secure. The display/execution repository 411 is further coupled to a credit server 414 to report any fees to be billed for access to a digital work and a repository 415 for accessing digital works stored therein.

STRUCTURE OF DIGITAL WORKS

Usage rights are attached directly to digital works. Thus, it is important to understand the structure of a digital work. The structure of a digital work, in particular composite digital works, may be naturally organized into an acyclic structure such as a hierarchy. For example, a magazine has various articles and photographs which may have been created and are owned by different persons. Each of the articles and photographs may represent a node in a hierarchical structure. Consequently, controls, i.e. usage rights, may be placed on each node by the creator. By enabling control and fee billing to be associated with each node, a creator of a work can be assured that the rights and fees are not circumvented.

In the currently preferred embodiment, the file information for a digital work is divided into two files: a "contents" file and a "description tree" file. From the perspective of a repository, the "contents" file is a stream of addressable bytes whose format depends completely on the interpreter used to play, display or print the digital work. The description tree file makes it possible to examine the rights and fees for a work without reference to the content of the digital work. It should be noted that the term description tree as used herein refers to any type of acyclic structure used to represent the relationship between the various components of a digital work.

Figure 5 illustrates the layout of a contents file. Referring to Figure 5, a digital work is comprised of story A 510, advertisement 511, story B 512 and story C 513. It is assumed that the digital work is stored starting at a relative address of 0. Each of the parts of the digital work are stored linearly so that story A 510 is stored at approximately addresses 0-30,000, advertisement 511 at addresses 30,001-40,000, story B 512 at addresses 40,001-60,000 and story C 513 at addresses 60,001-85K. The detail of story A 510 is illustrated in Figure 6. Referring to Figure 6, the story A 510 is further broken down to show text 614 stored at address 0-1500, soldier photo 615 at addresses 1501-10,000, graphics 616 stored at addresses 10,001-25,000 and sidebar 617 stored address 25,001-30,000. Note that the data in the contents file may be compressed (for saving storage) or encrypted (for security).

From Figures 5 and 6 it is readily observed that a digital work can be represented by its component parts as a hierarchy. The description tree for a digital work is comprised of a set of related descriptor blocks (d-blocks). The contents of each d-block is described with respect to Figure 7. Referring to Figure 7, a d-block 700 includes an identifier 701 which is a unique identifier for the work in the repository, a starting address 702 providing the start address of the first byte of the work, a length 703 giving the number of bytes in the work, a rights portion 704 wherein the granted usage rights and their status data are maintained, a parent pointer 705 for pointing to a parent d-block and child pointers 706 for pointing to the child d-blocks. In the currently preferred embodiment, the identifier 701 has two parts. The first part is a unique number assigned to the repository upon manufacture. The second part is a unique number assigned to the work upon creation. The rights portion 704 will contain a data structure, such as a look-up table, wherein the various information associated with a right is maintained. The information required by the respective usage rights is described in more detail below. D-blocks form a strict hierarchy. The top d-block of a work has no parent; all other d-blocks have one parent. The relationship of usage rights between parent and child d-blocks and how conflicts are resolved is described below.

A special type of d-block is a "shell" d-block. A shell d-block adds no new content beyond the content of its parts. A shell d-block is used to add rights and fee information, typically by distributors of digital works.

Figure 8 illustrates a description tree for the digital work of Figure 5. Referring to Figure 8, a top d-block 820 for the digital work points to the various stories and advertisements contained therein. Here, the top d-block 820 points to d-block 821 (representing story A 510), d-block 822 (representing the advertisement 511), d-block 823 (representing story B 512) and and d-block 824 (representing story C 513).

The portion of the description tree for Story A 510 is illustrated in Figure 9. D-block 925 represents text 614, d-block 926 represents photo 615, d-block 927 represents graphics 616 by and d-block 928 represents sidebar 617.

The rights portion 704 of a descriptor block is further illustrated in Figure 10. Figure 10 illustrates a structure which is repeated in the rights portion 704 for each right. Referring to Figure 10, each right will have a right code field 1050 and status information field 1052. The right code field 1050 will contain a unique code assigned to a right. The status information field 1052 will contain information relating to the state of a right and the digital work. Such information is indicated below in Table 1. The rights as stored in the rights portion 704 may typically be in numerical order based on the right code.

TABLE 1

DIGITAL WORK STATE INFORMATION		
Property	**Value**	**Use**
Copies-in-Use	Number	A counter of the number of copies of a work that are in use. Incremented when another copy is used; decremented when use is completed.
Loan-Period	Time-Units	Indicator of the maximum number of time-units that a document can be loaned out
Loaner-Copy	Boolean	Indicator that the current work is a loaned out copy of an authorized digital work.
Remaining-Time	Time-Units	Indicator of the remaining time of use on a metered document right.
Document-Descr	String	A string containing various identifying information about a document. The exact format of this is not specified, but it can include information such as a publisher name, author name, ISBN number, and so on.
Revenue-Owner	RO-Descr	A handle identifying a revenue owner for a digital work. This is used for reporting usage fees.
Publication-Date	Date-Descr	The date that the digital work was published.
History-list	History-Rec	A list of events recording the repostories and dates for operations that copy, transfer, backup, or restore a digital work.

The approach for representing digtal works by separating description data from content assumes that parts of a file are contiguous but takes no position on the actual representation of content. In particular, it is neutral to the question of whether content representation may take an object oriented approach. It would be natural to represent content as objects. In principle, it may be convenient to have content objects that include the billing structure and rights information that is represented in the d-blocks. Such variations in the design of the representation are possible and are viable alternatives but may introduce processing overhead, e.g. the interpretation of the objects.

Digital works are stored in a repository as part of a hierarchical file system. Folders (also termed directories and sub-directories) contain the digital works as well as other folders. Digital works and folders in a folder are ordered in alphabetical order. The digital works are typed to reflect how the files are used. Usage rights can be attached to folders so that the folder itself is treated as a digital work. Access to the folder would then be handled in the same fashion as any other digital work As will be described in more detail below, the contents of the folder are subject to their own rights. Moreover, file management rights may be attached to the folder which define how folder contents can be managed.

ATTACHING USAGE RIGHTS TO A DIGITAL WORK

It is fundamental to the present invention that the usage rights are treated as part of the digital work. As the digital work is distributed, the scope of the granted usage rights will remain the same or may be narrowed. For example, when a digital work is transferred from a document server to a repository, the usage rights may include the right to loan a copy for a predetermined period of time (called the original rights). When the repository loans out a copy of the digital work, the usage rights in the loaner copy (called the next set of rights) could be set to prohibit any further rights to loan out the copy. The basic idea is that one cannot grant more rights than they have.

The attachment of usage rights into a digital work may occur in a variety of ways. If the usage rights will be the same for an entire digital work, they could be attached when the digital work is processed for deposit in the digital work server. In the case of a digital work having different usage rights for the various components, this can be done as the digital work is being created. An authoring tool or digital work assembling tool could be utilized which provides for an automated process of attaching the usage rights.

As will be described below, when a digital work is copied, transferred or loaned, a "next set of rights" can be specified. The"next set of rights" will be attached to the digital work as it is transported.

Resolving Conflicting Rights

Because each part of a digital work may have its own usage rights, there will be instances where the rights of a "contained part" are different from its parent or container part. As a result, conflict rules must be established to dictate when and how a right may be exercised. The hierarchical structure of a digital work facilitates the enforcement of such rules. A "strict" rule would be as follows: a right for a part in a digital work is sanctioned if and only if it is sanctioned for the part, for ancestor d-blocks containing the part and for all descendent d-blocks. By sanctioned, it is meant that (1) each of the respective parts must have the right, and (2) any conditions for exercising the right are satisfied.

It also possible to implement the present invention using a more lenient rule. In the more lenient rule, access to the part may be enabled to the descendent parts which have the right, but access is denied to the descendents which do not.

An example of applying both the strict rule and lenient is illustrated with reference to Figure 11. Referring to Figure 11, a root d-block 1101 has child d-blocks 1102-1105. In this case, root d-block represents a magazine, and each of the child d-blocks 1102-1105 represent articles in the magazine. Suppose that a request is made to PRINT the digital work represented by root d-block 1101 wherein the strict rule is followed. The rights for the root d-block 1101 and child d-blocks 1102-1105 are then examined. Root d-block 1101 and child d-blocks 1102 and 1105 have been granted PRINT rights. Child d-block 1103 has not been granted PRINT rights and child d-block 1104 has PRINT rights conditioned on payment of a usage fee.

Under the strict rule the PRINT right cannot be exercised because the child d-block does not have the PRINT right. Under the lenient rule, the result would be different. The digital works represented by child d-blocks 1102 and 1105 could be printed and the digital work represented by d-block 1104 could be printed so long as the usage fee is paid. Only the digital work represented by d-block 1103 could not be printed. This same result would be accomplished under the strict rule if the requests were directed to each of the individual digital works.

The present invention supports various combinations of allowing and disallowing access. Moreover, as will be described below, the usage rights grammar permits the owner of a digital work to specify if constraints may be imposed on the work by a container part. The manner in which digital works may be sanctioned because of usage rights conflicts would be implementation specific and would depend on the nature of the digital works.

REPOSITORIES

In the description of Figure 2, it was indicated that repositories come in various forms. All repositories provide a core set of services for the transmission of digital works. The manner in which digital works are exchanged is the basis for all transaction between repositories. The various repository types differ in the ultimate functions that they perform. Repositories may be devices themselves, or they may be incorporated into other systems. An example is the rendering repository 203 of Figure 2.

A repository will have associated with it a repository identifier. Typically, the repository identifier would be a unique number assigned to the repository at the time of manufacture. Each repository will also be classified as being in a particular security class. Certain communications and transactions may be conditioned on a repository being in a particular security class. The various security classes are described in greater detail below.

As a prerequisite to operation, a repository will require possession of an identification certificate. Identification certificates are encrypted to prevent forgery and are issued by a Master repository. A master repository plays the role of an authorization agent to enable repositories to receive digital works. Identification certificates must be updated on a periodic basis. Identification certificates are described in greater detail below with respect to the registration transaction.

A repository has both a hardware and functional embodiment. The functional embodiment is typically software executing on the hardware embodiment. Alternatively, the functional embodiment may be embedded in the hardware embodiment such as an Application Specific Integrated Circuit (ASIC) chip.

The hardware embodiment of a repository will be enclosed in a secure housing which if compromised, may cause the repository to be disabled. The basic components of the hardware embodiment of a repository are described with reference to Figure 12. Referring to Figure 12, a repository is comprised of a processing means 1200, storage system 1207, clock 1205 and external interface 1206. The processing means 1200 is comprised of a processor element 1201 and processor memory 1202. The processing means 1201 provides controller, repository transaction and usage rights transaction functions for the repository. Various functions in the operation of the repository such as decryption and/or decompression of digital works and transaction messages are also performed by the processing means 1200. The processor element 1201 may be a microprocessor or other suitable computing component. The processor memory 1202 would typically be further comprised of Read Only Memories (ROM) and Random Access Memories (RAM). Such memories would contain the software instructions utilized by the processor element 1201 in performing the functions of the repository.

The storage system 1207 is further comprised of descriptor storage 1203 and content storage 1204. The description tree storage 1203 will store the description tree for the digital work and the content storage will store the associated content. The description tree storage 1203 and content storage 1204 need not be of the same type of storage medium, nor are they necessarily on the same physical device. So for example, the descriptor storage 1203 may be stored on a solid state storage (for rapid retrieval of the description tree information), while the content storage 1204 may be on a high capacity storage such as an optical disk.

The clock 1205 is used to time-stamp various time based conditions for usage rights or for metering usage fees which may be associated with the digital works. The clock 1205 will have an uninterruptable power supply, e.g. a battery, in order to maintain the integrity of the time-stamps. The external interface means 1206 provides for the signal connection to other repositories and to a credit server. The external interface means 1206 provides for the exchange of signals via such standard interfaces such as RS-232 or Personal Computer Manufacturers Card Industry Association (PCMCIA) standards, or FDDI. The external interface means 1206 may also provide network connectivity.

The functional embodiment of a repository is described with reference to Figure 13. Referring to Figure 13, the functional embodiment is comprised of an operating system 1301, core repository services 1302, usage transaction handlers 1303, repository specific functions, 1304 and a user interface 1305. The operating system 1301 is specific to the repository and would typically depend on the type of processor being used. The operating system 1301 would also provide the basic services for controlling and interfacing between the basic components of the repository.

The core repository services 1302 comprise a set of functions required by each and every repository. The core repository services 1302 include the session initiation transactions which are defined in greater detail below. This set of services also includes a generic ticket agent which is used to "punch" a digital ticket and a generic authorization server for processing authorization specifications. Digital tickets and authorizations are specific mechanisms for controlling the distribution and use of digital works and are described in more detail below. Note that coupled to the core repository services are a plurality of identification certificates 1306. The identification certificates 1306 are required to enable the use of the repository.

The usage transactions handlers 1303 comprise functionality for processing access requests to digital works and for billing fees based on access. The usage transactions supported will be different for each repository type. For example, it may not be necessary for some repositories to handle access requests for digital works.

The repository specific functionality 1304 comprises functionality that is unique to a repository. For example, the master repository has special functionality for issuing digital certificates and maintaining encryption keys. The repository specific functionality 1304 would include the user interface implementation for the repository.

Repository Security Classes

For some digital works the losses caused by any individual instance of unauthorized copying is insignificant and the chief economic concern lies in assuring the convenience of access and low-overhead billing. In such cases, simple and inexpensive handheld repositories and network-based workstations may be suitable repositories, even though the measures and guarantees of security are modest.

At the other extreme, some digital works such as a digital copy of a first run movie or a bearer bond or stock certificate would be of very high value so that it is prudent to employ caution and fairly elaborate security measures to ensure that they are not copied or forged. A repository suitable for holding such a digital work could have elaborate measures for ensuring physical integrity and for verifying authorization before use.

By arranging a universal protocol, all kinds of repositories can communicate with each other in principle. However, creators of some works will want to specify that their works will only be transferred to repositories whose level of security is high enough. For this reason, document repositories have a ranking system for classes and levels of security. The security classes in the currently preferred embodiment are described in Table 2.

TABLE 2

REPOSITORY SECURITY LEVELS	
Level	Description of Security
0	Open system. Document transmission is unencrypted. No digital certificate is required for identification. The security of the system depends mostly on user honesty, since only modest knowledge may be needed to circumvent the security measures. The repository has no provisions for preventing unauthorized programs from running and accessing or copying files. The system does not prevent the use of removable storage and does not encrypt stored files.
1	Minimal security. Like the previous class except that stored files are minimally encrypted, including ones on removable storage.

TABLE 2 (continued)

REPOSITORY SECURITY LEVELS	
Level	Description of Security
2	Basic security. Like the previous class except that special tools and knowledge are required to compromise the programming, the contents of the repository, or the state of the clock. All digital communications are encrypted. A digital certificate is provided as identification. Medium level encryption is used. Repository identification number is unforgeable.
3	General security. Like the previous class plus the requirement of special tools are needed to compromise the physical integrity of the repository and that modest encryption is used on all transmissions. Password protection is required to use the local user interface. The digital clock system cannot be reset without authorization. No works would be stored on removable storage. When executing works as programs, it runs them in their own address space and does not give them direct access to any file storage or other memory containing system code or works. They can access works only through the transmission transaction protocol.
4	Like the previous class except that high level encryption is used on all communications. Sensors are used to record attempts at physical and electronic tampering. After such tampering, the repository will not perform other transactions until it has reported such tampering to a designated server.
5	Like the previous class except that if the physical or digital attempts at tampering exceed some preset thresholds that threaten the physical integrity of the repository or the integrity of digital and cryptographic barriers, then the repository will save only document description records of history but will erase or destroy any digital identifiers that could be misused if released to an unscrupulous party. It also modifies any certificates of authenticity to indicate that the physical system has been compromised. It also erases the contents of designated documents.
6	Like the previous class except that the repository will attempt wireless communication to report tampering and will employ noisy alarms.
10	This would correspond to a very high level of security. This server would maintain constant communications to remote security systems reporting transactions, sensor readings, and attempts to circumvent security.

The characterization of security levels described in Table 2 is not intended to be fixed. More important is the idea of having different security levels for different repositories. It is anticipated that new security classes and requirements will evolve according to social situations and changes in technology.

Repository User Interface

A user interface is broadly defined as the mechanism by which a user interacts with a repository in order to invoke transactions to gain access to a digital work, or exercise usage rights. As described above, a repository may be embodied in various forms. The user interface for a repository will differ depending on the particular embodiment. The user interface may be a graphical user interface having icons representing the digital works and the various transactions that may be performed. The user interface may be a generated dialog in which a user is prompted for information.

The user interface itself need not be part of the repository. As a repository may be embedded in some other device, the user interface may merely be a part of the device in which the repository is embedded. For example, the repository could be embedded in a "card" that is inserted into an available slot in a computer system. The user interface may be a combination of a display, keyboard, cursor control device and software executing on the computer system.

At a minimum, the user interface must permit a user to input information such as access requests and alpha numeric data and provide feedback as to transaction status. The user interface will then cause the repository to initiate the suitable transactions to service the request. Other facets of a particular user interface will depend on the functionality that a repository will provide.

CREDIT SERVERS

In the present invention, fees may be associated with the exercise of a right. The requirement for payment of fees is described with each version of a usage right in the usage rights language. The recording and reporting of such fees is performed by the credit server. One of the capabilities enabled by associating fees with rights is the possibility of

supporting a wide range of charging models. The simplest model, used by conventional software, is that there is a single fee at the time of purchase, after which the purchaser obtains unlimited rights to use the work as often and for as long as he or she wants. Alternative models, include metered use and variable fees. A single work can have different fees for different uses. For example, viewing a photograph on a display could have different fees than making a hardcopy or including it in a newly created work. A key to these alternative charging models is to have a low overhead means of establishing fees and accounting for credit on these transactions.

A credit server is a computational system that reliably authorizes and records these transactions so that fees are billed and paid. The credit server reports fees to a billing clearinghouse. The billing clearinghouse manages the financial transactions as they occur. As a result, bills may be generated and accounts reconciled. Preferably, the credit server would store the fee transactions and periodically communicate via a network with the billing clearinghouse for reconciliation. In such an embodiment, communications with the billing clearinghouse would be encrypted for integrity and security reasons. In another embodiment, the credit server acts as a "debit card" where transactions occur in "real-time" against a user account.

A credit server is comprised of memory, a processing means, a clock, and interface means for coupling to a repository and a financial institution (e.g. a modem). The credit server will also need to have security and authentication functionality. These elements are essentially the same elements as those of a repository. Thus, a single device can be both a repository and a credit server, provided that it has the appropriate processing elements for carrying out the corresponding functions and protocols. Typically, however, a credit server would be a card-sized system in the possession of the owner of the credit. The credit server is coupled to a repository and would interact via financial transactions as described below. Interactions with a financial institution may occur via protocols established by the financial institutions themselves.

In the currently preferred embodiment credit servers associated with both the server and the repository report the financial transaction to the billing clearinghouse. For example, when a digital work is copied by one repository to another for a fee, credit servers coupled to each of the repositories will report the transaction to the billing clearinghouse. This is desirable in that it insures that a transaction will be accounted for in the event of some break in the communication between a credit server and the billing clearinghouse. However, some implementations may embody only a single credit server reporting the transaction to minimize transaction processing at the risk of losing some transactions.

USAGE RIGHTS LANGUAGE

The present invention uses statements in a high level "usage rights language" to define rights associated with digital works and their parts. Usage rights statements are interpreted by repositories and are used to determine what transactions can be successfully carried out for a digital work and also to determine parameters for those transactions. For example, sentences in the language determine whether a given digital work can be copied, when and how it can be used, and what fees (if any) are to be charged for that use. Once the usage rights statements are generated, they are encoded in a suitable form for accessing during the processing of transactions.

Defining usage rights in terms of a language in combination with the hierarchical representation of a digital work enables the support of a wide variety of distribution and fee schemes. An example is the ability to attach multiple versions of a right to a work. So a creator may attach a PRINT right to make 5 copies for $10.00 and a PRINT right to make unlimited copies for $100.00. A purchaser may then choose which option best fits his needs. Another example is that rights and fees are additive. So in the case of a composite work, the rights and fees of each of the components works is used in determining the rights and fees for the work as a whole.

The basic contents of a right are illustrated in Figure 14. Referring to Figure 14, a right 1450 has a transactional component 1451 and a specifications component 1452. A right 1450 has a label (e.g. COPY or PRINT) which indicates the use or distribution privileges that are embodied by the right. The transactional component 1451 corresponds to a particular way in which a digital work may be used or distributed. The transactional component 1451 is typically embodied in software instructions in a repository which implement the use or distribution privileges for the right. The specifications components 1452 are used to specify conditions which must be satisfied prior to the right being exercised or to designate various transaction related parameters. In the currently preferred embodiment, these specifications include copy count 1453, Fees and Incentives 1454, Time 1455, Access and Security 1456 and Control 1457. Each of these specifications will be described in greater detail below with respect to the language grammar elements.

The usage rights language is based on the grammar described below. A grammar is a convenient means for defining valid sequence of symbols for a language. In describing the grammar the notation "[a | b | c] is used to indicate distinct choices among alternatives. In this example, a sentence can have either an "a", "b" or "c". It must include exactly one of them. The braces { } are used to indicate optional items. Note that brackets, bars and braces are used to describe the language of usage rights sentences but do not appear in actual sentences in the language.

In contrast, parentheses are part of the usage rights language. Parentheses are used to group items together in lists. The notation (x*) is used to indicate a variable length list, that is, a list containing one or more items of type x.

The notation (x)* is used to indicate a variable number of lists containing x.

Keywords in the grammar are words followed by colons. Keywords are a common and very special case in the language. They are often used to indicate a single value, typically an identifier. In many cases, the keyword and the parameter are entirely optional. When a keyword is given, it often takes a single identifier as its value. In some cases, the keyword takes a list of identifiers.

In the usage rights language, time is specified in an hours:minutes:seconds (or hh:mm:ss) representation. Time zone indicators, e.g. PDT for Pacific Daylight Time, may also be specified. Dates are represented as year/ month/day (or YYYY/MMM/DD). Note that these time and date representations may specify moments in time or units of time Money units are specified in terms of dollars.

Finally, in the usage rights language, various "things" will need to interact with each other. For example, an instance of a usage right may specify a bank account, a digital ticket, etc.. Such things need to be identified and are specified herein using the suffix "-ID."

The Usage Rights Grammar is listed in its entirety in Figure 15 and is described below.

Grammar element 1501 **"Digital Work Rights: = (Rights*)"** define the digital work rights as a set of rights. The set of rights attached to a digital work define how that digital work may be transferred, used, performed or played. A set of rights will attach to the entire digital work and in the case of compound digital works, each of the components of the digital work. The usage rights of components of a digital may be different.

Grammar element 1502 **"Right : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec} {Access-Spec} {Fee-Spec})"** enumerates the content of a right. Each usage right must specify a right code. Each right may also optionally specify conditions which must be satisfied before the right can be exercised. These conditions are copy count, control, time, access and fee conditions. In the currently preferred embodiment, for the optional elements, the following defaults apply: copy count equals 1, no time limit on the use of the right, no access tests or a security level required to use the right and no fee is required. These conditions will each be described in greater detail below.

It is important to note that a digital work may have multiple versions of a right, each having the same right code. The multiple version would provide alternative conditions and fees for accessing the digital work.

Grammar element 1503 **"Right-Code : = Render-Code I Transport-Code I File-Management-Code| Derivative-Works- Code Configuration-Code"** distinguishes each of the specific rights into a particular right type (although each right is identified by distinct right codes). In this way, the grammar provides a catalog of possible rights that can be associated with parts of digital works. In the following, rights are divided into categories for convenience in describing them.

Grammar element 1504 **"Render-Code : = [Play: {Player: Player-ID} I Print: {Printer: Printer-ID}]"** lists a category of rights all involving the making of ephemeral, transitory, or non-digital copies of the digital work. After use the copies are erased.

- Play — A process of rendering or performing a digital work on some processor. This includes such things as playing digital movies, playing digital music, playing a video game, running a computer program, or displaying a document on a display.
- Print — To render the work in a medium that is not further protected by usage rights, such as printing on paper.

Grammar element 1505 **"Transport-Code : = [Copy I Transfer I Loan {Remaining-Rights: Next-Set-of-Rights}] {(Next-Copy-Rights: Next-Set of Rights)}"** lists a category of rights involving the making of persistent, usable copies of the digital work on other repositories. The optional Next-Copy-Rights determine the rights on the work after it is transported. If this is not specified, then the rights on the transported copy are the same as on the original. The optional Remaining-Rights specify the rights that remain with a digital work when it is loaned out. If this is not specified, then the default is that no rights can be exercised when it is loaned out.

- Copy — Make a new copy of a work
- Transfer — Moving a work from one repository to another.
- Loan — Temporarily loaning a copy to another repository for a specified period of time.

Grammar element 1506 **"File-Management-Code : = Backup {Back-Up-Copy-Rights: Next-Set -of Rights}| Restore I Delete I Folder I Directory {Name:Hide-Local I Hide - Remote}{Parts:Hide-Local I Hide-Remote}"** lists a category of rights involving operations for file management, such as the making of backup copies to protect the copy owner against catastrophic equipment failure.

Many software licenses and also copyright law give a copy owner the right to make backup copies to protect against catastrophic failure of equipment. However, the making of uncontrolled backup copies is inherently at odds with the ability to control usage, since an uncontrolled backup copy can be kept and then restored even after the authorized copy was sold.

The File management rights enable the making and restoring of backup copies in a way that respects usage rights, honoring the requirements of both the copy owner and the rights grantor and revenue owner. Backup copies of work descriptions (including usage rights and fee data) can be sent under appropriate protocol and usage rights control to other document repositories of sufficiently high security. Further rights permit organization of digital works into folders which themselves are treated as digital works and whose contents may be "hidden" from a party seeking to determine the contents of a repository.

- Backup To make a backup copy of a digital work as protection against media failure.
- Restore To restore a backup copy of a digital work.
- Delete To delete or erase a copy of a digital work.
- Folder To create and name folders, and to move files and folders between folders.
- Directory To hide a folder or its contents.

Grammar element 1507 **"Derivative-Works-Code: [Extract | Embed | Edit {Process: Process-ID}] {Next-Copy-Rights : Next-Set-of Rights}"** lists a category of rights involving the use of a digital work to create new works.

- Extract To remove a portion of a work, for the purposes of creating a new work.
- Embed To include a work in an existing work.
- Edit To alter a digital work by copying, selecting and modifying portions of an existing digital work.

Grammar element 1508 **"Configuration-Code : = Install | Uninstall"** lists a category of rights for installing and uninstalling software on a repository (typically a rendering repository.) This would typically occur for the installation of a new type of player within the rendering repository.

- Install: To install new software on a repository.
- Uninstall: To remove existing software from a repository.

Grammar element 1509 **"Next-Set-of-Rights : = {(Add : Set-Of-Rights)} {(Delete: Set-Of-Rights)} {(Replace: Set-Of-Rights)} {(Keep: Set-Of-Rights)}"** defines how rights are carried forward for a copy of a digital work. If the Next-Copy-Rights is not specified, the rights for the next copy are the same as those of the current copy. Otherwise, the set of rights for the next copy can be specified. Versions of rights after Add: are added to the current set of rights. Rights after Delete: are deleted from the current set of rights. If only right codes are listed after Delete:, then all versions of rights with those codes are deleted. Versions of rights after Replace: subsume all versions of rights of the same type in the current set of rights.

If Remaining-Rights is not specified, then there are no rights for the original after all Loan copies are loaned out. If Remaining-Rights is specified, then the Keep: token can be used to simplify the expression of what rights to keep behind. A list of right codes following keep means that all of the versions of those listed rights are kept in the remaining copy. This specification can be overridden by subsequent Delete: or Replace: specifications.

Copy Count Specification

For various transactions, it may be desirable to provide some limit as to the number of "copies" of the work which may be exercised simultaneously for the right. For example, it may be desirable to limit the number of copies of a digital work that may be loaned out at a time or viewed at a time.

Grammar element 1510 **"Copy-Count : = (Copies: positive-integer | 0 | unlimited)"** provides a condition which defines the number of "copies" of a work subject to the right . A copy count can be 0, a fixed number, or unlimited. The copy-count is associated with each right, as opposed to there being just a single copy-count for the digital work. The Copy-Count for a right is decremented each time that a right is exercised. When the Copy-Count equals zero, the right can no longer be exercised. If the Copy-Count is not specified, the default is one.

Control Specification

Rights and fees depend in general on rights granted by the creator as well as further restrictions imposed by later distributors. Control specifications deal with interactions between the creators and their distributors governing the imposition of further restrictions and fees. For example, a distributor of a digital work may not want an end consumer of a digital work to add fees or otherwise profit by commercially exploiting the purchased digital work.

Grammar element 1511 **"Control-Spec : = (Control: {Restrictable | Unrestrictable} {Unchargeable | Chargeable})"** provides a condition to specify the effect of usage rights and fees of parents on the exercise of the right. A

digital work is restrictable if higher level d-blocks can impose further restrictions (time specifications and access specifications) on the right. It is unrestrictable if no further restrictions can be imposed. The default setting is restrictable. A right is unchargeable if no more fees can be imposed on the use of the right. It is chargeable if more fees can be imposed. The default is chargeable.

Time Specification

It is often desirable to assign a start date or specify some duration as to when a right may be exercised. Grammar element 1512 **"Time-Spec : = ({Fixed-Interval | Sliding-Interval | Meter-Time} Until: Expiration-Date)"** provides for specification of time conditions on the exercise of a right. Rights may be granted for a specified time. Different kinds of time specifications are appropriate for different kinds of rights. Some rights may be exercised during a fixed and predetermined duration. Some rights may be exercised for an interval that starts the first time that the right is invoked by some transaction. Some rights may be exercised or are charged according to some kind of metered time, which may be split into separate intervals. For example, a right to view a picture for an hour might be split into six ten minute viewings or four fifteen minute viewings or twenty three minute viewings.

The terms "time" and "date" are used synonymously to refer to a moment in time. There are several kinds of time specifications. Each specification represents some limitation on the times over which the usage right applies. The Expiration-Date specifies the moment at which the usage right ends. For example, if the Expiration-Date is "Jan 1, 1995," then the right ends at the first moment of 1995. If the Expiration-Date is specified as "forever", then the rights are interpreted as continuing without end. If only an expiration date is given, then the right can be exercised as often as desired until the expiration date.

Grammar element 1513 **"Fixed-Interval :≡ From: Start-Time"** is used to define a predetermined interval that runs from the start time to the expiration date.

Grammar element 1514 **"Sliding-Interval : = Interval: Use-Duration"** is used to define an indeterminate (or "open") start time. It sets limits on a continuous period of time over which the contents are accessible. The period starts on the first access and ends after the duration has passed or the expiration date is reached, whichever comes first. For example, if the right gives 10 hours of continuous access, the use-duration would begin when the first access was made and end 10 hours later.

Grammar element 1515 **"Meter-Time: = Time-Remaining: Remaining-Use"** is used to define a "meter time," that is, a measure of the time that the right is actually exercised. It differs from the Sliding-Interval specification in that the time that the digital work is in use need not be continuous. For example, if the rights guarantee three days of access, those days could be spread out over a month. With this specification, the rights can be exercised until the meter time is exhausted or the expiration date is reached, whichever comes first.

Remaining-Use: = Time-Unit
Start-Time: = Time-Unit
Use-Duration: = Time-Unit
All of the time specifications include time-unit specifications in their ultimate instantiation.

Security Class and Authorization Specification

The present invention provides for various security mechanisms to be introduced into a distribution or use scheme. Grammar element 1516 **"Access-Spec : = ({SC: Security-Class} {Authorization: Authorization-ID*} {Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})"** provides a means for restricting access and transmission. Access specifications can specify a required security class for a repository to exercise a right or a required authorization test that must be satisfied.

The keyword **"SC:"** is used to specify a minimum security level for the repositories involved in the access. If **"SC: "** is not specified; the lowest security level is acceptable.

The optional **"Authorization:"** keyword is used to specify required authorizations on the same repository as the work. The optional **"Other-Authorization:"** keyword is used to specify required authorizations on the other repository in the transaction.

The optional **"Ticket:"** keyword specifies the identity of a ticket required for the transaction. A transaction involving digital tickets must locate an appropriate digital ticket agent who can "punch" or otherwise validate the ticket before the transaction can proceed. Tickets are described in greater detail below.

In a transaction involving a repository and a document server, some usage rights may require that the repository have a particular authorization, that the server have some authorization, or that both repositories have (possibly different) authorizations. Authorizations themselves are digital works (hereinafter referred to as an authorization object) that can be moved between repositories in the same manner as other digital works. Their copying and transferring is

subject to the same rights and fees as other digital works. A repository is said to have an authorization if that authorization object is contained within the repository.

In some cases, an authorization may be required from a source other than the document server and repository. An authorization object referenced by an Authorization-ID can contain digital address information to be used to set up a communications link between a repository and the authorization source. These are analogous to phone numbers. For such access tests, the communication would need to be established and authorization obtained before the right could be exercised.

For one-time usage rights, a variant on this scheme is to have a digital ticket. A ticket is presented to a digital ticket agent, whose type is specified on the ticket. In the simplest case, a certified generic ticket agent, available on all repositories, is available to "punch" the ticket. In other cases, the ticket may contain addressing information for locating a "special" ticket agent. Once a ticket has been punched, it cannot be used again for the same kind of transaction (unless it is unpunched or refreshed in the manner described below.) Punching includes marking the ticket with a timestamp of the date and time it was used. Tickets are digital works and can be copied or transferred between repositories according to their usage rights.

In the currently preferred embodiment, a "punched" ticket becomes "unpunched" or "refreshed" when it is copied or extracted. The Copy and Extract operations save the date and time as a property of the digital ticket. When a ticket agent is given a ticket, it can simply check whether the digital copy was made after the last time that it was punched. Of course, the digital ticket must have the copy or extract usage rights attached thereto.

The capability to unpunch a ticket is inportant in the following cases:

- A digital work is circulated at low cost with a limitation that it can be used only once.
- A digital work is circulated with a ticket that can be used once to give discounts on purchases of other works.
- A digital work is circulated with a ticket (included in the purchase price and possibly embedded in the work) that can be used for a future upgrade.

In each of these cases, if a paid copy is made of the digital work (including the ticket) the new owner would expect to get a fresh (unpunched) ticket, whether the copy seller has used the work or not. In contrast, loaning a work or simply transferring it to another repository should not revitalize the ticket.

Usage Fees and Incentives Specification

The billing for use of a digital work is fundamental to a commercial distribution system. Grammar Element 1517 **"Fee-Spec: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec | Markup-Spec"** provides a range of options for billing for the use of digital works.

A key feature of this approach is the development of low-overhead billing for transactions in potentially small amounts. Thus, it becomes feasible to collect fees of only a few cents each for thousands of transactions.

The grammar differentiates between uses where the charge is per use from those where it is metered by the time unit. Transactions can support fees that the user pays for using a digital work as well as incentives paid by the right grantor to users to induce them to use or distribute the digital work.

The optional scheduled discount refers to the rest of the fee specification--discounting it by a percentage over time. If it is not specified, then there is no scheduled discount. Regular fee specifications are constant over time. Scheduled fee specifications give a schedule of dates over which the fee specifications change. Markup specifications are used in d-blocks for adding a percentage to the fees already being charged.

Grammar Element 1518 **"Scheduled-Discount: = (Scheduled-Discount: (Time-Spec Percentage)*)"** A Scheduled-Discount is a essentially a scheduled modifier of any other fee specification for this version of the right of the digital work. (It does not refer to children or parent digital works or to other versions of rights.). It is a list of pairs of times and percentages. The most recent time in the list that has not yet passed at the time of the transaction is the one in effect. The percentage gives the discount percentage. For example, the number 10 refers to a 10% discount.

Grammar Element 1519 **"Regular-Fee-Spec : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-Spec | Best-Price-Spec | Call-For-Price-Spec] {Min: Money-Unit Per: Time-Spec)(Max: Money-Unit Per: Time-Spec} To: Account-ID)"** provides for several kinds of fee specifications.

Fees are paid by the copy-owner/user to the revenue-owner if Fee: is specified. Incentives are paid by the revenue-owner to the user if Incentive: is specified. If the Min: specification is given, then there is a minimum fee to be charged per time-spec unit for its use. If the Max: specification is given, then there is a maximum fee to be charged per time-spec for its use. When Fee: is specified, Account-ID identifies the account to which the fee is to be paid. When Incentive: is specified, Account-ID identifies the account from which the fee is to be paid.

Grammar element 1520 **"Per-Use-Spec: = Per-Use: Money-unit"** defines a simple fee to be paid every time the right is exercised, regardless of how much time the transaction takes.

Grammar element 1521 **"Metered-Rate-Spec : = Metered: Money-Unit Per: Time-Spec"** defines a metered-rate fee paid according to how long the right is exercised. Thus, the time it takes to complete the transaction determines the fee.

Grammar element 1522 **"Best-Price-Spec := Best-Price: Money-unit Max: Money-unit"** is used to specify a best-price that is determined when the account is settled. This specification is to accommodate special deals, rebates, and pricing that depends on information that is not available to the repository. All fee specifications can be combined with tickets or authorizations that could indicate that the consumer is a wholesaler or that he is a preferred customer, or that the seller be authorized in some way. The amount of money in the Max: field is the maximum amount that the use will cost. This is the amount that is tentatively debited from the credit server. However, when the transaction is ultimately reconciled, any excess amount will be returned to the consumer in a separate transaction.

Grammar element 1523 **"Call-For-Price-Spec:= Call-For-Price"** is similar to a **"Best-Price-Spec"** in that it is intended to accommodate cases where prices are dynamic. **A Call-For-Price Spec** requires a communication with a dealer to determine the price. This option cannot be exercised if the repository cannot communicate with a dealer at the time that the right is exercised. It is based on a secure transaction whereby the dealer names a price to exercise the right and passes along a deal certificate which is referenced or included in the billing process.

Grammar element 1524 **"Scheduled-Fee-Spec: = (Schedule: (Time-Spec Regular-Fee-Spec)*)"** is used to provide a schedule of dates over which the fee specifications change. The fee specification with the most recent date not in the future is the one that is in effect. This is similar to but more general than the scheduled discount. It is more general, because it provides a means to vary the fee agreement for each time period.

Grammar element 1525 **"Markup-Spec: = Markup: percentage To: Account-ID"** is provided for adding a percentage to the fees already being charged. For example, a 5% markup means that a fee of 5% of cumulative fee so far will be allocated to the distributor. A markup specification can be applied to all of the other kinds of fee specifications. It is typically used in a shell provided by a distributor. It refers to fees associated with d-blocks that are parts of the current d-block. This might be a convenient specification for use in taxes, or in distributor overhead.

REPOSITORY TRANSACTIONS

When a user requests access to a digital work, the repository will initiate various transactions. The combination of transactions invoked will depend on the specifications assigned for a usage right. There are three basic types of transactions, Session Initiation Transactions, Financial Transactions and Usage Transactions. Generally, session initiation transactions are initiated first to establish a valid session. When a valid session is established, transactions corresponding to the various usage rights are invoked. Finally, request specific transactions are performed.

Transactions occur between two repositories (one acting as a server), between a repository and a document playback platform (e.g. for executing or viewing), between a repository and a credit server or between a repository and an authorization server. When transactions occur between more than one repository, it is assumed that there is a reliable communication channel between the repositories. For example, this could be a TCP/IP channel or any other commercially available channel that has built-in capabilities for detecting and correcting transmission errors. However, it is not assumed that the communication channel is secure. Provisions for security and privacy are part of the requirements for specifying and implementing repositories and thus form the need for various transactions.

Message Transmission

Transactions require that there be some communication between repositories. Communication between repositories occurs in units termed as messages. Because the communication line is assumed to be unsecure, all communications with repositories that are above the lowest security class are encrypted utilizing a public key encryption technique. Public key encryption is a well known technique in the encryption arts. The term key refers to a numeric code that is used with encryption and decryption algorithms. Keys come in pairs, where "writing keys" are used to encrypt data and "checking keys" are used to decrypt data. Both writing and checking keys may be public or private. Public keys are those that are distributed to others. Private keys are maintained in confidence.

Key management and security is instrumental in the success of a public key encryption system. In the currently preferred embodiment, one or more master repositories maintain the keys and create the identification certificates used by the repositories.

When a sending repository transmits a message to a receiving repository, the sending repository encrypts all of its data using the public writing key of the receiving repository. The sending repository includes its name, the name of the receiving repository, a session identifier such as a nonce (described below), and a message counter in each message.

In this way, the communication can only be read (to a high probability) by the receiving repository, which holds the private checking key for decryption. The auxiliary data is used to guard against various replay attacks to security. If

messages ever arrive with the wrong counter or an old nonce, the repositories can assume that someone is interfering with communication and the transaction terminated.

The respective public keys for the repositories to be used for encryption are obtained in the registration transaction described below.

Session Initiation Transactions

A usage transaction is carried out in a session between repositories. For usage transactions involving more than one repository, or for financial transactions between a repository and a credit server, a registration transaction is performed. A second transaction termed a login transaction, may also be needed to initiate the session. The goal of the registration transaction is to establish a secure channel between two repositories who know each others identities. As it is assumed that the communication channel between the repositories is reliable but not secure, there is a risk that a non-repository may mimic the protocol in order to gain illegitimate access to a repository.

The registration transaction between two repositories is described with respect to Figures 16 and 17. The steps described are from the perspective of a "repository-1" registering its identity with a "repository-2". The registration must be symmetrical so the same set of steps will be repeated for repository-2 registering its identity with repository-1. Referring to Figure 16, repository-1 first generates an encrypted registration identifier, step 1601 and then generates a registration message, step 1602. A registration message is comprised of an identifier of a master repository, the identification certificate for the repository-1 and an encrypted random registration identifier. The identification certificate is encrypted by the master repository in its private key and attests to the fact that the repository (here repository-1) is a bona fide repository. The identification certificate also contains a public key for the repository, the repository security level and a timestamp (indicating a time after which the certificate is no longer valid.) The registration identifier is a number generated by the repository for this registration. The registration identifier is unique to the session and is encrypted in repository-1's private key. The registration identifier is used to improve security of authentication by detecting certain kinds of communications based attacks. Repository-1 then transmits the registration message to repository-2, step 1603.

Upon receiving the registration message, repository-2 determines if it has the needed public key for the master repository, step 1604. If repository-2 does not have the needed public key to decrypt the identification certificate, the registration transaction terminates in an error, step 1618.

Assuming that repository-2 has the proper public key the identification certificate is decrypted, step 1605. Repository-2 saves the encrypted registration identifier, step 1606, and extracts the repository identifier, step 1607. The extracted repository identifier is checked against a "hotlist" of compromised document repositories, step 1608. In the currently preferred embodiment, each repository will contain "hotlists" of compromised repositories. If the repository is on the "hotlist', the registration transaction terminates in an error per step 1618. Repositories can be removed from the hotlist when their certificates expire, so that the list does not need to grow without bound. Also, by keeping a short list of hotlist certificates that it has previously received, a repository can avoid the work of actually going through the list. These lists would be encrypted by a master repository. A minor variation on the approach to improve efficiency would have the repositories first exchange lists of names of hotlist certificates, ultimately exchanging only those lists that they had not previously received. The "hotlists" are maintained and distributed by Master repositories.

Note that rather than terminating in error, the transaction could request that another registration message be sent based on an identification certificate created by another master repository. This may be repeated until a satisfactory identification certificate is found, or it is determined that trust cannot be established.

Assuming that the repository is not on the hotlist, the repository identification needs to be verified. In other words, repository-2 needs to validate that the repository on the other end is really repository-1. This is termed performance testing and is performed in order to avoid invalid access to the repository via a counterfeit repository replaying a recording of a prior session initiation between repository-1 and repository-2. Performance testing is initiated by repository-2 generating a performance message, step 1609. The performance message consists of a nonce, the names of the respective repositories, the time and the registration identifier received from repository-1. A nonce is a generated message based on some random and variable information (e.g. the time or the temperature.) The nonce is used to check whether repository-1 can actually exhibit correct encrypting of a message using the private keys it claims to have, on a message that it has never seen before. The performance message is encrypted using the public key specified in the registration message of repository-1. The performance message is transmitted to repository-1, step 1610, where it is decrypted by repository-1 using its private key, step 1611. Repository-1 then checks to make sure that the names of the two repositories are correct, step 1612, that the time is accurate, step 1613 and that the registration identifier corresponds to the one it sent, step 1614. If any of these tests fails, the transaction is terminated per step 1616. Assuming that the tests are passed, repository-1 transmits the nonce to repository-2 in the clear, step 1615. Repository-2 then compares the received nonce to the original nonce, step 1617. If they are not identical, the registration transaction terminates in an error per step 1618. If they are the same, the registration transaction has successfully completed.

At this point, assuming that the transaction has not terminated, the repositories exchange messages containing session keys to be used in all communications during the session and synchronize their clocks. Figure 17 illustrates the session information exchange and clock synchronization steps (again from the perspective of repository-1.) Referring to Figure 17, repository-1 creates a session key pair, step 1701. A first key is kept private and is used by repository-1 to encrypt messages. The second key is a public key used by repository-2 to decrypt messages. The second key is encrypted using the public key of repository-2, step 1702 and is sent to repository-2, step 1703. Upon receipt, repository-2 decrypts the second key, step 1704. The second key is used to decrypt messages in subsequent communications. When each repository has completed this step, they are both convinced that the other repository is bona fide and that they are communicating with the original. Each repository has given the other a key to be used in decrypting further communications during the session. Since that key is itself transmitted in the public key of the receiving repository only it will be able to decrypt the key which is used to decrypt subsequent messages.

After the session information is exchanged, the repositories must synchronize their clocks. Clock synchronization is used by the repositories to establish an agreed upon time base for the financial records of their mutual transactions. Referring back to Figure 17, repository-2 initiates clock synchronization by generating a time stamp exchange message, step 1705, and transmits it to repository-1, step 1706. Upon receipt, repository-1 generates its own time stamp message, step 1707 and transmits it back to repository-2, step 1708. Repository-2 notes the current time, step 1709 and stores the time received from repository-1, step 1710. The current time is compared to the time received from repository-1, step 1711. The difference is then checked to see if it exceeds a predetermined tolerance (e.g. one minute), step 1712. If it does, repository-2 terminates the transaction as this may indicate tampering with the repository, step 1713. If not repository-2 computes an adjusted time delta, step 1714. The adjusted time delta is the difference between the clock time of repository-2 and the average of the times from repository-1 and repository-2.

To achieve greater accuracy, repository-2 can request the time again up to a fixed number of times (e.g. five times), repeat the clock synchronization steps, and average the results.

A second session initiation transaction is a Login transaction. The Login transaction is used to check the authenticity of a user requesting a transaction. A Login transaction is particularly prudent for the authorization of financial transactions that will be charged to a credit server. The Login transaction involves an interaction between the user at a user interface and the credit server associated with a repository. The information exchanged here is a login string supplied by the repository/credit server to identify itself to the user, and a Personal Identification Number (PIN) provided by the user to identify himself to the credit server. In the event that the user is accessing a credit server on a repository different from the one on which the user interface resides, exchange of the information would be encrypted using the public and private keys of the respective repositories.

Billing Transactions

Billing Transactions are concerned with monetary transactions with a credit server. Billing Transactions are carried out when all other conditions are satisfied and a usage fee is required for granting the request. For the most part, billing transactions are well understood in the state of the art. These transactions are between a repository and a credit server, or between a credit server and a billing clearinghouse. Briefly, the required transactions include the following:

- Registration and LOGIN transactions by which the repository and user establish their bona fides to a credit server. These transactions would be entirely internal in cases where the repository and credit server are implemented as a single system.
- Registration and LOGIN transactions, by which a credit server establishes its bona fides to a billing clearinghouse.
- An Assign-fee transaction to assign a charge. The information in this transaction would include a transaction identifier, the identities of the repositories in the transaction, and a list of charges from the parts of the digital work. If there has been any unusual event in the transaction such as an interruption of communications, that information is included as well.
- A Begin-charges transaction to assign a charge. This transaction is much the same as an assign-fee transaction except that it is used for metered use. It includes the same information as the assign-fee transaction as well as the usage fee information. The credit-server is then responsible for running a clock.
- An End-charges transaction to end a charge for metered use. (In a variation on this approach, the repositories would exchange periodic charge information for each block of time.)
- A report-charges transaction between a personal credit server and a billing clearinghouse. This transaction is invoked at least once per billing period. It is used to pass along information about charges. On debit and credit cards, this transaction would also be used to update balance information and credit limits as needed.

All billing transactions are given a transaction ID and are reported to the credit severs by both the server and the client. This reduces possible loss of billing information if one of the parties to a transaction loses a banking card and

provides a check against tampering with the system.

Usage Transactions

After the session initiation transactions have been completed, the usage request may then be processed. To simplify the description of the steps carried out in processing a usage request, the term requester is used to refer to a repository in the requester mode which is initiating a request, and the term server is used to refer to a repository in the server mode and which contains the desired digital work. In many cases such as requests to print or view a work, the requester and server may be the same device and the transactions described in the following would be entirely internal. In such instances, certain transaction steps, such as the registration transaction, need not be performed.

There are some common steps that are part of the semantics of all of the usage rights transactions. These steps are referred to as the common transaction steps. There are two sets -- the "opening" steps and the "closing" steps. For simplicity, these are listed here rather than repeating them in the descriptions of all of the usage rights transactions.

Transactions can refer to a part of a digital work, a complete digital work, or a Digital work containing other digital works. Although not described in detail herein, a transaction may even refer to a folder comprised of a plurality of digital works. The term "work" is used to refer to what ever portion or set of digital works is being accessed.

Many of the steps here involve determining if certain conditions are satisfied. Recall that each usage right may have one or more conditions which must be satisfied before the right can be exercised. Digital works have parts and parts have parts. Different parts can have different rights and fees. Thus, it is necessary to verify that the requirements are met for ALL of the parts that are involved in a transaction For brevity, when reference is made to checking whether the rights exist and conditions for exercising are satisfied, it is meant that all such checking takes place for each of the relevant parts of the work.

Figure 18 illustrates the initial common opening and closing steps for a transaction. At this point it is assumed that registration has occurred and that a "trusted" session is in place. General tests are tests on usage rights associated with the folder containing the work or some containing folder higher in the file system hierarchy. These tests correspond to requirements imposed on the work as a consequence of its being on the particular repository, as opposed to being attached to the work itself. Referring to Figure 18, prior to initiating a usage transaction, the requester performs any general tests that are required before the right associated with the transaction can be exercised, step, 1801. For example, install, uninstall and delete rights may be implemented to require that a requester have an authorization certificate before the right can be exercised. Another example is the requirement that a digital ticket be present and punched before a digital work may be copied to a requester. If any of the general tests fail, the transaction is not initiated, step, 1802. Assuming that such required tests are passed, upon receiving the usage request, the server generates a transaction identifier that is used in records or reports of the transaction, step 1803. The server then checks whether the digital work has been granted the right corresponding to the requested transaction, step 1804. If the digital work has not been granted the right corresponding to the request, the transaction terminates, step 1805. If the digital work has been granted the requested right, the server then determines if the various conditions for exercising the right are satisfied. Time based conditions are examined, step 1806. These conditions are checked by examining the time specification for the the version of the right. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the time based conditions are satisfied, the server checks security and access conditions, step 1807. Such security and access conditions are satisfied if: 1) the requester is at the specified security class, or a higher security class, 2) the server satisfies any specified authorization test and 3) the requester satisfies any specified authorization tests and has any required digital tickets. If any of the conditions are not satisfied, the transaction terminates per step 1805.

Assuming that the security and access conditions are all satisfied, the server checks the copy count condition, step 1808. If the copy count equals zero, then the transaction cannot be completed and the transaction terminates per step 1805.

Assuming that the copy count does not equal zero, the server checks if the copies in use for the requested right is greater than or equal to any copy count for the requested right (or relevant parts), step 1809. If the copies in use is greater than or equal to the copy count, this indicates that usage rights for the version of the transaction have been exhausted. Accordingly, the server terminates the transaction, step 1805. If the copy count is less than the copies in use for the transaction the transaction can continue, and the copies in use would be incremented by the number of digital works requested in the transaction, step 1810.

The server then checks if the digital work has a "Loan" access right, step 1811. The "Loan" access right is a special case since remaining rights may be present even though all copies are loaned out. If the digital work has the "Loan" access right, a check is made to see if all copies have been loaned out, step 1812. The number of copies that could be loaned is the sum of the Copy-Counts for all of the versions of the loan right of the digital work. For a composite work, the relevant figure is the minimal such sum of each of the components of the composite work. If all copies have been loaned out, the remaining rights are determined, step 1813. The remaining-rights is determined from the remaining

rights specifications from the versions of the Loan right. If there is only one version of the Loan right, then the determination is simple. The remaining rights are the ones specified in that version of the Loan right, or none if Remaining-Rights: is not specified. If there are multiple versions of the Loan right and all copies of all of the versions are loaned out, then the remaining rights is taken as the minimum set (intersection) of remaining rights across all of the versions of the loan right. The server then determines if the requested right is in the set of remaining rights, step 1814. If the requested right is not in the set of remaining rights, the server terminates the transaction, step 1805.

If Loan is not a usage right for the digital work or if all copies have not been loaned out or the requested right is in the set of remaining rights, fee conditions for the right are then checked, step 1815. This will initiate various financial transactions between the repository and associated credit server. Further, any metering of usage of a digital work will commence. If any financial transaction fails, the transaction terminates per step 1805.

It should be noted that the order in which the conditions are checked need not follow the order of steps 1806-1815.

At this point, right specific steps are now performed and are represented here as step 1816. The right specific steps are described in greater detail below.

The common closing transaction steps are now performed. Each of the closing transaction steps are performed by the server after a successful completion of a transaction. Referring back to Figure 18, the copies in use value for the requested right is decremented by the number of copies involved in the transaction, step 1817. Next, if the right had a metered usage fee specification, the server subtracts the elapsed time from the Remaining-Use-Time associated with the right for every part involved in the transaction, step 1818. Finally, if there are fee specifications associated with the right, the server initiates End-Charge financial transaction to confirm billing, step 1819.

Transmission Protocol

An important area to consider is the transmission of the digital work from the server to the requester. The transmission protocol described herein refers to events occurring after a valid session has been created. The transmission protocol must handle the case of disruption in the communications between the repositories. It is assumed that interference such as injecting noise on the communication channel can be detected by the integrity checks (e.g., parity, checksum, etc.) that are built into the transport protocol and are not discussed in detail herein.

The underlying goal in the transmission protocol is to preclude certain failure modes, such as malicious or accidental interference on the communications channel. Suppose, for example, that a user pulls a card with the credit server at a specific time near the end of a transaction. There should not be a vulnerable time at which "pulling the card" causes the repositories to fail to correctly account for the number of copies of the work that have been created. Restated, there should be no time at which a party can break a connection as a means to avoid payment after using a digital work.

If a transaction is interrupted (and fails), both repositories restore the digital works and accounts to their state prior to the failure, modulo records of the failure itself.

Figure 19 is a state diagram showing steps in the process of transmitting information during a transaction. Each box represents a state of a repository in either the server mode (above the central dotted line 1901) or in the requester mode (below the dotted line 1901). Solid arrows stand for transitions between states. Dashed arrows stand for message communications between the repositories. A dashed message arrow pointing to a solid transition arrow is interpreted as meaning that the transition takes place when the message is received. Unlabeled transition arrows take place unconditionally. Other labels on state transition arrows describe conditions that trigger the transition.

Referring now to Figure 19, the server is initially in a state 1902 where a new transaction is initiated via start message 1903. This message includes transaction information including a transaction identifier and a count of the blocks of data to be transferred. The requester, initially in a wait state 1904 then enters a data wait state 1905.

The server enters a data transmit state 1906 and transmits a block of data 1907 and then enters a wait for acknowledgement state 1908. As the data is received, the requester enters a data receive state 1909 and when the data blocks are completely received it enters an acknowledgement state 1910 and transmits an Acknowledgement message 1911 to the server.

If there are more blocks to send, the server waits until receiving an Acknowledgement message from the requester. When an Acknowledgement message is received it sends the next block to the requester and again waits for acknowledgement. The requester also repeats the same cycle of states.

If the server detects a communications failure before sending the last block, it enters a cancellation state 1912 wherein the transaction is cancelled. Similarly, if the requester detects a communications failure before receiving the last block it enters a cancellation state 1913.

If there are no more blocks to send, the server commits to the transaction and waits for the final Acknowledgement in state 1914. If there is a communications failure before the server receives the final Acknowledgement message, it still commits to the transaction but includes a report about the event to its credit server in state 1915. This report serves two purposes. It will help legitimize any claims by a user of having been billed for receiving digital works that were not completely received. Also it helps to identify repositories and communications lines that have suspicious patterns of

use and interruption. The server then enters its completion state 1916.

On the requester side, when there are no more blocks to receive, the requester commits to the transaction in state 1917. If the requester detects a communications failure at this state, it reports the failure to its credit server in state 1918, but still commits to the transaction. When it has committed, it sends an acknowledgement message to the server. The server then enters its completion state 1919.

The key property is that both the server and the requester cancel a transaction if it is interrupted before all of the data blocks are delivered, and commits to it if all of the data blocks have been delivered.

There is a possibility that the server will have sent all of the data blocks (and committed) but the requester will not have received all of them and will cancel the transaction. In this case, both repositories will presumably detect a communications failure and report it to their credit server. This case will probably be rare since it depends on very precise timing of the communications failure. The only consequence will be that the user at the requester repository may want to request a refund from the credit services -- and the case for that refund will be documented by reports by both repositories.

To prevent loss of data, the server should not delete any transferred digital work until receiving the final acknowledgement from the requester. But it also should not use the file. A well known way to deal with this situation is called "two-phase commit" or 2PC.

Two-phase commit works as follows. The first phase works the same as the method described above. The server sends all of the data to the requester. Both repositories mark the transaction (and appropriate files) as uncommitted. The server sends a ready-to-commit message to the requester. The requester sends back an acknowledgement. The server then commits and sends the requester a commit message. When the requester receives the commit message, it commits the file.

If there is a communication failure or other crash, the requester must check back with the server to determine the status of the transaction. The server has the last word on this. The requester may have received all of the data, but if it did not get the final message, it has not committed. The server can go ahead and delete files (except for transaction records) once it commits, since the files are known to have been fully transmitted before starting the 2PC cycle.

There are variations known in the art which can be used to achieve the same effect. For example, the server could use an additional level of encryption when transmitting a work to a client. Only after the client sends a message acknowledging receipt does it send the key. The client then agrees to pay for the digital work. The point of this variation is that it provides a clear audit trail that the client received the work. For trusted systems, however, this variation adds a level of encryption for no real gain in accountability.

The transaction for specific usage rights are now discussed.

The Copy Transaction

A Copy transaction is a request to make one or more independent copies of the work with the same or lesser usage rights. Copy differs from the extraction right discussed later in that it refers to entire digital works or entire folders containing digital works. A copy operation cannot be used to remove a portion of a digital work.

- The requester sends the server a message to initiate the Copy Transaction. This message indicates the work to be copied, the version of the copy right to be used for the transaction, the destination address information (location in a folder) for placing the work, the file data for the work (including its size), and the number of copies requested.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the client according to the transmission protocol. If a Next-Set-Of-Rights has been provided in the version of the right, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted. In any event, the Copy-Count field for the copy of the digital work being sent right is set to the number-of-copies requested.
- The requester records the work contents, data, and usage rights and stores the work. It records the date and time that the copy was made in the properties of the digital work.
- The repositories perform the common closing transaction steps.

The Transfer Transaction

A Transfer transaction is a request to move copies of the work with the same or lesser usage rights to another repository. In contrast with a copy transaction, this results in removing the work copies from the server.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be transferred, the version of the transfer right to be used in the transaction, the destination address information for placing the work, the file data for the work, and the number of copies involved.

- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted.

In either case, the Copy-Count field for the transmitted rights are set to the number-of-copies requested.

- The requester records the work contents, data, and usage rights and stores the work.
- The server decrements its copy count by the number of copies involved in the transaction.
- The repositories perform the common closing transaction steps.
- If the number of copies remaining in the server is now zero, it erases the digital work from its memory.

The Loan Transaction

A loan transaction is a mechanism for loaning copies of a digital work. The maximum duration of the loan is determined by an internal parameter of the digital work. Works are automatically returned after a predetermined time period.

- The requester sends the server a message to initiate the Transfer Transaction. This message indicates the work to be loaned, the version of the loan right to be used in the transaction, the destination address information for placing the work, the number of copies involved, the file data for the work, and the period of the loan.
- The server checks the validity of the requested loan period, and ends with an error if the period is not valid. Loans for a loaned copy cannot extend beyond the period of the original loan to the server.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, the rights of the original are transmitted, as modified to reflect the loan period.
- The requester records the digital work contents, data, usage rights, and loan period and stores the work.
- The server updates the usage rights information in the digital work to reflect the number of copies loaned out.
- The repositories perform the common closing transaction steps.
- The server updates the usage rights data for the digital work. This may preclude use of the work until it is returned from the loan. The user on the requester platform can now use the transferred copies of the digital work. A user accessing the original repository cannot use the digital work , unless there are copies remaining. What happens next depends on the order of events in time.

Case 1. If the time of the loan period is not yet exhausted and the requester sends the repository a Return message.

- The return message includes the requester identification, and the transaction ID.
- The server decrements the copies-in-use field by the number of copies that were returned. (If the number of digital works returned is greater than the number actually borrowed, this is treated as an error.) This step may now make the work available at the server for other users.
- The requester deactivates its copies and removes the contents from its memory.

Case 2. If the time of the loan period is exhausted and the requester has not yet sent a Return message.

- The server decrements the copies-in-use field by the number digital works that were borrowed.
- The requester automatically deactivates its copies of the digital work. It terminates all current uses and erases the digital work copies from memory. One question is why a requester would ever return a work earlier than the period of the loan, since it would be returned automatically anyway. One reason for early return is that there may be a metered fee which determines the cost of the loan. Returning early may reduce that fee.

The Play Transaction

A play transaction is a request to use the contents of a work. Typically, to "play" a work is to send the digital work through some kind of transducer, such as a speaker or a display device. The request implies the intention that the contents will not be communicated digitally to any other system. For example, they will not be sent to a printer, recorded on any digital medium, retained after the transaction or sent to another repository.

This term "play" is natural for examples like playing music, playing a movie, or playing a video game. The general

form of play means that a "player" is used to use the digital work. However, the term play covers all media and kinds of recordings. Thus one would "play" a digital work, meaning, to render it for reading, or play a computer program, meaning to execute it. For a digital ticket the player would be a digital ticket agent.

- The requester sends the server a message to initiate the play transaction. This message indicates the work to be played, the version of the play right to be used in the transaction, the identity of the player being used, and the file data for the work.
- The server checks the validity of the player identification and the compatibility of the player identification with the player specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server and requester read and write the blocks of data as requested by the player according to the transmission protocol. The requester plays the work contents, using the player.
- When the player is finished, the player and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Print Transaction

A Print transaction is a request to obtain the contents of a work for the purpose of rendering them on a "printer." We use the term "printer" to include the common case of writing with ink on paper. However, the key aspect of "printing" in our use of the term is that it makes a copy of the digital work in a place outside of the protection of usage rights. As with all rights, this may require particular authorization certificates.

Once a digital work is printed, the publisher and user are bound by whatever copyright laws are in effect. However, printing moves the contents outside the control of repositories. For example, absent any other enforcement mechanisms, once a digital work is printed on paper, it can be copied on ordinary photocopying machines without intervention by a repository to collect usage fees. If the printer to a digital disk is permitted, then that digital copy is outside of the control of usage rights. Both the creator and the user know this, although the creator does not necessarily give tacit consent to such copying, which may violate copyright laws.

- The requester sends the server a message to initiate a Print transaction. This message indicates the work to be played, the identity of the printer being used, the file data for the work, and the number of copies in the request.
- The server checks the validity of the printer identification and the compatibility of the printer identification with the printer specification in the right. It ends with an error if these are not satisfactory.
- The repositories perform the common opening transaction steps.
- The server transmits blocks of data according to the transmission protocol.
- The requester prints the work contents, using the printer.
- When the printer is finished, the printer and the requester remove the contents from their memory.
- The repositories perform the common closing transaction steps.

The Backup Transaction

A Backup transaction is a request to make a backup copy of a digital work, as a protection against media failure. In the context of repositories, secure backup copies differ from other copies in three ways: (1) they are made under the control of a Backup transaction rather than a Copy transaction, (2) they do not count as regular copies, and (3) they are not usable as regular copies. Generally, backup copies are encrypted.

Although backup copies may be transferred or copied, depending on their assigned rights, the only way to make them useful for playing, printing or embedding is to restore them.

The output of a Backup operation is both an encrypted data file that contains the contents and description of a work, and a restoration file with an encryption key for restoring the encrypted contents. In many cases, the encrypted data file would have rights for "printing" it to a disk outside of the protection system, relying just on its encryption for security. Such files could be stored anywhere that was physically safe and convenient. The restoration file would be held in the repository. This file is necessary for the restoration of a backup copy. It may have rights for transfer between repositories.

- The requester sends the server a message to initiate a backup transaction. This message indicates the work to be backed up, the version of the backup right to be used in the transaction, the destination address information for placing the backup copy, the file data for the work.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester. If a Next-Set-Of-Rights has been provided,

those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.

- The requester records the work contents, data, and usage rights. It then creates a one-time key and encrypts the contents file. It saves the key information in a restoration file.
- The repositories perform the common closing transaction steps.

In some cases, it is convenient to be able to archive the large, encrypted contents file to secure offline storage, such as a magneto-optical storage system or magnetic tape. This creation of a non-repository archive file is as secure as the encryption process. Such non-repository archive storage is considered a form of "printing" and is controlled by a print right with a specified "archive-printer." An archive-printer device is programmed to save the encrypted contents file (but not the description file) offline in such a way that it can be retrieved.

The Restore Transaction

A Restore transaction is a request to convert an encrypted backup copy of a digital work into a usable copy. A restore operation is intended to be used to compensate for catastrophic media failure. Like all usage rights, restoration rights can include fees and access tests including authorization checks.

- The requester sends the server a message to initiate a Restore transaction. This message indicates the work to be restored, the version of the restore right for the transaction, the destination address information for placing the work, and the file data for the work.
- The server verifies that the contents file is available (i.e. a digital work corresponding to the request has been backed-up.) If it is not, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server retrieves the key from the restoration file. It decrypts the work contents, data, and usage rights.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the work. Otherwise, a set of default rights for backup files of the original are transmitted by the server.
- The requester stores the digital work.
- The repositories perform the common closing transaction steps.

The Delete Transaction

A Delete transaction deletes a digital work or a number of copies of a digital work from a repository. Practically all digital works would have delete rights.

- The requester sends the server a message to initiate a delete transaction. This message indicates the work to be deleted, the version of the delete right for the transaction.
- The repositories perform the common opening transaction steps.
- The server deletes the file, erasing it from the file system.
- The repositories perform the common closing transaction steps.

The Directory Transaction

A Directory transaction is a request for information about folders, digital works, and their parts. This amounts to roughly the same idea as protection codes in a conventional file system like TENEX, except that it is generalized to the full power of the access specifications of the usage rights language.

The Directory transaction has the important role of passing along descriptions of the rights and fees associated with a digital work. When a user wants to exercise a right, the user interface of his repository implicitly makes a directory request to determine the versions of the right that are available. Typically these are presented to the user -- such as with different choices of billing for exercising a right. Thus, many directory transactions are invisible to the user and are exercised as part of the normal process of exercising all rights.

- The requester sends the server a message to initiate a Directory transaction. This message indicates the file or folder that is the root of the directory request and the version of the directory right used for the transaction.
- The server verifies that the information is accessible to the requester. In particular, it does not return the names of any files that have a HIDE-NAME status in their directory specifications, and it does not return the parts of any folders or files that have HIDE-PARTS in their specification. If the information is not accessible, the server ends

the transaction with an error.

- The repositories perform the common opening transaction steps.
- The server sends the requested data to the requester according to the transmission protocol.
- The requester records the data.
- The repositories perform the common closing transaction steps.

The Folder Transaction

A Folder transaction is a request to create or rename a folder, or to move a work between folders. Together with Directory rights, Folder rights control the degree to which organization of a repository can be accessed or modified from another repository.

- The requester sends the server a message to initiate a Folder transaction. This message indicates the folder that is the root of the folder request, the version of the folder right for the transaction, an operation, and data. The operation can be one of create, rename, and move file. The data are the specifications required for the operation, such as a specification of a folder or digital work and a name.
- The repositories perform the common opening transaction steps.
- The server performs the requested operation -- creating a folder, renaming a folder, or moving a work between folders.
- The repositories perform the common closing transaction steps.

The Extract Transaction

A extract transaction is a request to copy a part of a digital work and to create a new work containing it. The extraction operation differs from copying in that it can be used to separate a part of a digital work from d-blocks or shells that place additional restrictions or fees on it. The extraction operation differs from the edit operation in that it does not change the contents of a work, only its embedding in d-blocks. Extraction creates a new digital work.

- The requester sends the server a message to initiate an Extract transaction. This message indicates the part of the work to be extracted, the version of the extract right to be used in the transaction, the destination address information for placing the part as a new work, the file data for the work, and the number of copies involved.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and stores the work. It records the date and time that new work was made in the properties of the work.
- The repositories perform the common closing transaction steps.

The Embed Transaction

An embed transaction is a request to make a digital work become a part of another digital work or to add a shell d-block to enable the adding of fees by a distributor of the work.

- The requester sends the server a message to initiate an Embed transaction. This message indicates the work to be embedded, the version of the embed right to be used in the transaction, the destination address information for placing the part as a a work, the file data for the work, and the number of copies involved.
- The server checks the control specifications for all of the rights in the part and the destination. If they are incompatible, the server ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The server transmits the requested contents and data to the requester according to the transmission protocol. If a Next-Set-Of-Rights has been provided, those rights are transmitted as the rights for the new work. Otherwise, the rights of the original are transmitted. The Copy-Count field for this right is set to the number-of-copies requested.
- The requester records the contents, data, and usage rights and embeds the work in the destination file.
- The repositories perform the common closing transaction steps.

The Edit Transaction

An Edit transaction is a request to make a new digital work by copying, selecting and modifying portions of an existing digital work. This operation can actually change the contents of a digital work. The kinds of changes that are permitted depend on the process being used. Like the extraction operation, edit operates on portions of a digital work. In contrast with the extract operation, edit does not affect the rights or location of the work. It only changes the contents. The kinds of changes permitted are determined by the type specification of the processor specified in the rights. In the currently preferred embodiment, an edit transaction changes the work itself and does not make a new work. However, it would be a reasonable variation to cause a new copy of the work to be made.

- The requester sends the server a message to initiate an Edit transaction. This message indicates the work to be edited, the version of the edit right to be used in the transaction, the file data for the work (including its size), the process-ID for the process, and the number of copies involved.
- The server checks the compatibility of the process-ID to be used by the requester against any process-ID specification in the right. If they are incompatible, it ends the transaction with an error.
- The repositories perform the common opening transaction steps.
- The requester uses the process to change the contents of the digital work as desired. (For example, it can select and duplicate parts of it; combine it with other information; or compute functions based on the information. This can amount to editing text, music, or pictures or taking whatever other steps are useful in creating a derivative work.)
- The repositories perform the common closing transaction steps.

The edit transaction is used to cover a wide range of kinds of works. The category describes a process that takes as its input any portion of a digital work and then modifies the input in some way. For example, for text, a process for editing the text would require edit rights. A process for "summarizing" or counting words in the text would also be considered editing. For a music file, processing could involve changing the pitch or tempo, or adding reverberations, or any other audio effect. For digital video works, anything which alters the image would require edit rights. Examples would be colorizing, scaling, extracting still photos, selecting and combining frames into story boards, sharpening with signal processing, and so on.

Some creators may want to protect the authenticity of their works by limiting the kinds of processes that can be performed on them. If there are no edit rights, then no processing is allowed at all. A processor identifier can be included to specify what kind of process is allowed. If no process identifier is specified, then arbitrary processors can be used. For an example of a specific process, a photographer may want to allow use of his photograph but may not want it to be colorized. A musician may want to allow extraction of portions of his work but not changing of the tonality.

Authorization Transactions

There are many ways that authorization transactions can be defined. In the following, our preferred way is to simply define them in terms of other transactions that we already need for repositories. Thus, it is convenient sometimes to speak of "authorization transactions," but they are actually made up of other transactions that repositories already have.

A usage right can specify an authorization-ID, which identifies an authorization object (a digital work in a file of a standard format) that the repository must have and which it must process. The authorization is given to the generic authorization (or ticket) server of the repository which begins to interpret the authorization.

As described earlier, the authorization contains a server identifier, which may just be the generic authorization server or it may be another server. When a remote authorization server is required, it must contain a digital address. It may also contain a digital certificate.

If a remote authorization server is required, then the authorization process first performs the following steps:

- The generic authorization server attempts to set up the communications channel. (If the channel cannot be set up, then authorization fails with an error.)
- When the channel is set up, it performs a registration process with the remote repository. (If registration fails, then the authorization fails with an error.)
- When registration is complete, the generic authorization server invokes a "Play" transaction with the remote repository, supplying the authorization document as the digital work to be played, and the remote authorization server (a program) as the "player." (If the player cannot be found or has some other error, then the authorization fails with an error.)
- The authorization server then "plays" the authorization. This involves decrypting it using either the public key of the master repository that issued the certificate or the session key from the repository that transmitted it. The authorization server then performs various tests. These tests vary according to the authorization server. They

include such steps as checking issue and validity dates of the authorization and checking any hot-lists of known invalid authorizations. The authorization server may require carrying out any other transactions on the repository as well, such as checking directories, getting some person to supply a password, or playing some other digital work. It may also invoke some special process for checking information about locations or recent events. The "script" for such steps is contained within the authorization server.

- If all of the required steps are completed satisfactorily, the authorization server completes the transaction normally, signaling that authorization is granted.

The Install Transaction

An Install transaction is a request to install a digital work as runnable software on a repository. In a typical case, the requester repository is a rendering repository and the software would be a new kind or new version of a player. Also in a typical case, the software would be copied to file system of the requester repository before it is installed.

- The requester sends the server an Install message. This message indicates the work to be installed, the version of the Install right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step certifies the software.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the compatibility-checking script and follows them. If the software is not compatible with the repository, the installation transaction ends with an error. (This step checks platform compatibility.)
- The requester retrieves the instructions in the installation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error. Note that the installation process puts the runnable software in a place in the repository where it is no longer accessible as a work for exercising any usage rights other than the execution of the software as part of repository operations in carrying out other transactions.
- The repositories perform the common closing transaction steps.

The Uninstall Transaction

An Uninstall transaction is a request to remove software from a repository. Since uncontrolled or incorrect removal of software from a repository could compromise its behavioral integrity, this step is controlled.

- The requester sends the server an Uninstall message. This message indicates the work to be uninstalled, the version of the Uninstall right being invoked, and the file data for the work (including its size).
- The repositories perform the common opening transaction steps.
- The requester extracts a copy of the digital certificate for the software. If the certificate cannot be found or the master repository for the certificate is not known to the requester, the transaction ends with an error.
- The requester checks whether the software is installed. If the software is not installed, the transaction ends with an error.
- The requester decrypts the digital certificate using the public key of the master repository, recording the identity of the supplier and creator, a key for decrypting the software, the compatibility information, and a tamper-checking code. (This step authenticates the certification of the software, including the script for uninstalling it.)
- The requester decrypts the software using the key from the certificate and computes a check code on it using a 1-way hash function. If the check-code does not match the tamper-checking code from the certificate, the installation transaction ends with an error. (This step assures that the contents of the software, including the various scripts, have not been tampered with.)
- The requester retrieves the instructions in the uninstallation script and follows them. If there is an error in this process (such as insufficient resources), then the transaction ends with an error.
- The repositories perform the common closing transaction steps.

Claims

1. A system for secure distribution and control of digital works between repositories comprising:

means for creating usage rights, each instance of a usage right representing a specific instance of how a digital work may be used or distributed;
means for attaching a created set of usage rights to a digital work;
a communications medium for coupling repositories to enable exchange of repository transaction messages;
a plurality of general repositories for storing and securely exchanging digital works with attached usage rights, each of said general repositories comprising:
a storage means for storing digital works and their attached usage rights;
an identification certificate for indicating that the associated general repository is secure;
an external interface for removably coupling to said communications medium;
a session initiation transaction processing means for establishing a secure and trusted session with another repository, said session initiation transaction processing means using said identification certificate;
a usage transaction processing means having a requester mode of operation for generating usage repository transaction messages to request access to digital works stored in another general repository, said usage repository transaction message specifying a usage right, said usage transaction processing means further having a server mode of operation for determining if a request for access to a digital work stored in said storage means may be granted, said request being granted only if the usage right specified in said request is attached to said digital work; and
an input means coupled to said usage transaction processing means for enabling user created signals to cause generation of a usage repository transaction message to request access to digital works.

2. The system as recited in Claim 1 further comprising a rendering system, said rendering system comprising:

a rendering repository for securely accessing digital works from a general repository, said rendering repository comprising;
a storage means for storing digital works and their attached usage rights;
an identification certificate, said identification certificate for indicating that the rendering repository is secure;
an external interface for removably coupling to said communications medium;
a session initiation transaction processing means for establishing a secure and trusted session with a general repository, said session initiation transaction processing means using said identification certificate;
a usage transaction processing means for generating usage repository transaction messages to request access to digital works stored in a general repository, said usage repository transaction message specifying a usage right;
an input means coupled to said usage transaction processing means for enabling user created signals to cause generation of usage repository transaction messages to request access to digital works;
a rendering device for rendering digital works.

3. The system as recited in Claim 1 wherein said means for creating usage rights is further for the specification of different sets of usage rights to be attached to digital works when a corresponding usage right is exercised.

4. The system as recited in Claim 1 wherein said usage rights grammar further defines means for specifying conditions which must be satisfied before a usage right may be exercised and said usage transaction processing means in said server mode is further comprised of means for determining if specified conditions for a usage right are satisfied before access is granted.

5. The system as recited in Claim 1 wherein a first usage right enables copying of a digital work and specification of a revenue owner who is paid a fee whenever a copy of said digital work is made.

6. A method for controlling distribution and use of digital works comprising the steps of:

a) attaching a set of usage rights to a digital work, each of said usage rights defining a specific instance of how a digital work may be used or distributed, said usage right specifying one or more conditions which must be satisfied in order for said usage right to be exercised and a next set of usage rights to be attached to a distributed digital work;
b) storing said digital work and its attached usage rights in a first repository;

c) a second repository initiating a request to access said digital work in said first repository, said request identifying a usage right representing how said second repository desires to use said digital work;
d) said first repository receiving said request from said second repository;
e) said first repository determining if the identified usage right is attached to said digital work;
f) said first repository denying access to said digital work if said identified usage right is not attached to said digital work;
g) if said identified usage right is attached to said digital work, said first repository determining if conditions specified by said usage right are satisfied;
h) if said conditions are not satisfied, said first repository denying access to said digital work;
i) if said conditions are satisfied, said first repository attaching a next set of usage rights to said digital work, said next set of usage rights specifying how said second repository may use and distribute said digital work; and
j) said first repository transmitting said digital work and said attached next set of usage rights to said second repository.

7. The method as recited in Claim 6 wherein said step of a second repository initiating a request to access said digital work in said first repository is further comprised of the steps of:

c1) said second repository initiating establishment of a trusted session with said first repository;
c2) said first repository performing a set of registration transaction steps with said second repository, successful completion of said set of registration transaction steps indicating that said first repository is a trusted repository;
c3) said second repository performing said set of registration transaction steps with said first repository, successful completion of said set of registration transaction steps indicating that said second repository is a trusted repository;
c4) if said first repository and said second repository each successfully complete said set of registration steps, said first and second repository exchanging session encryption and decryption keys for secure transmission of subsequent communications between said first and second repository; and
c5) if said first repository or said second repository cannot successfully complete said set of registration transaction steps, terminating said session.

8. A system for controlling distribution and use of digital works comprising:

means for attaching usage rights to said digital work, said usage rights indicating how a recipient may use and and subsequently distribute said digital work;
a communications medium for coupling repositories to enable distribution of digital works;
a plurality of repositories for managing exchange of digital works based on usage rights attached to said digital works, each of said plurality of repositories comprising:
a storage means for storing digital works and their attached usage rights;
a processor operating responsive to coded instructions;
a memory means coupled to said processor for storing coded instruction to enable said processor to operate in a first server mode for processing access requests to digital works and for attaching usage rights to digital works when transmitted to another of said plurality of repositories, a second requester mode for initiating requests to access digital works, and a session initiation mode for establishing a trusted session with another of said plurality of repositories over said communications medium;
a clock;
a repository interface for coupling to said communications medium.

9. The system as recited in Claim 8 further comprising a plurality of rendering systems for rendering of digital works, each of said rendering systems comprising:

a repository for secure receipt of a digital work; and
a rendering device having means for converting digital works to signals suitable for rendering of said digital works.

10. A method for secure access of digital works stored on a server repository, said digital works having associated therewith one or more usage rights for specifying how said digital work may be used or distributed, said method comprising the steps of:

a) a requesting repository performing a first registration transaction with a server repository, said first regis-

tration transaction for establishing to said server repository that said requesting repository is trustworthy;
b) concurrently with step a), said server repository responding with a second registration transaction, said second registration transaction for establishing to said requesting repository that said server repository is trustworthy;
c) if either said first registration transaction or said second registration transaction fails, said server repository denying access to said digital work;
d) if said first registration transaction and said second registration transaction are successful, said requesting repository initiating a usage transaction with respect to a digital work stored in said server repository, said usage transaction indicating a request to access a digital work and specifying a particular usage right;
e) determining if said usage transaction may be completed by comparing said particular usage right specified in said usage transaction and usage rights associated with said digital work;
f) if said particular usage right is not one of said usage rights associated with said digital work, denying access to said digital work; and
g) if said particular usage right is one of said usage rights associated with said digital work, granting access to said digital work and performing usage transaction steps associated with said particular usage right.

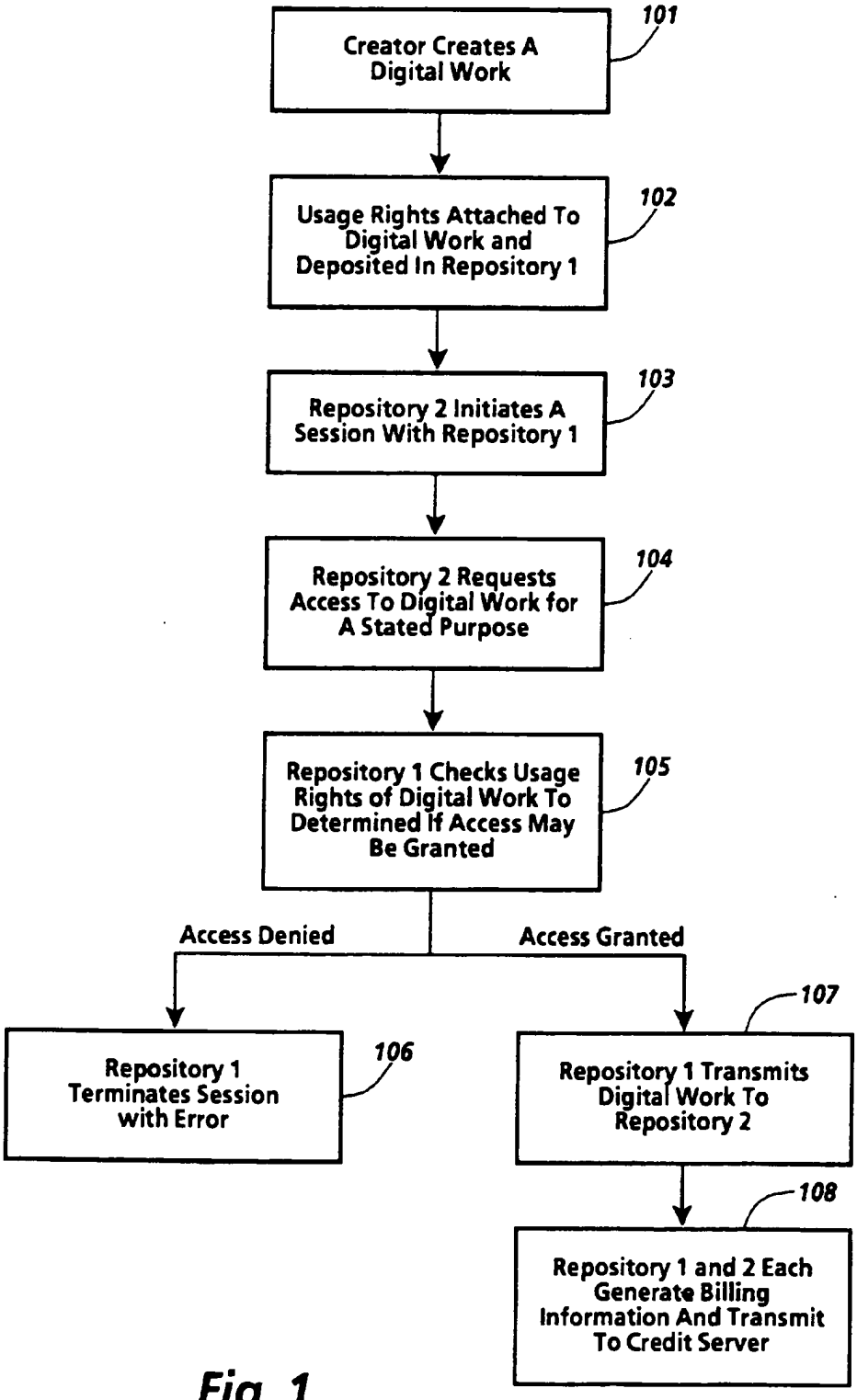


Fig. 1

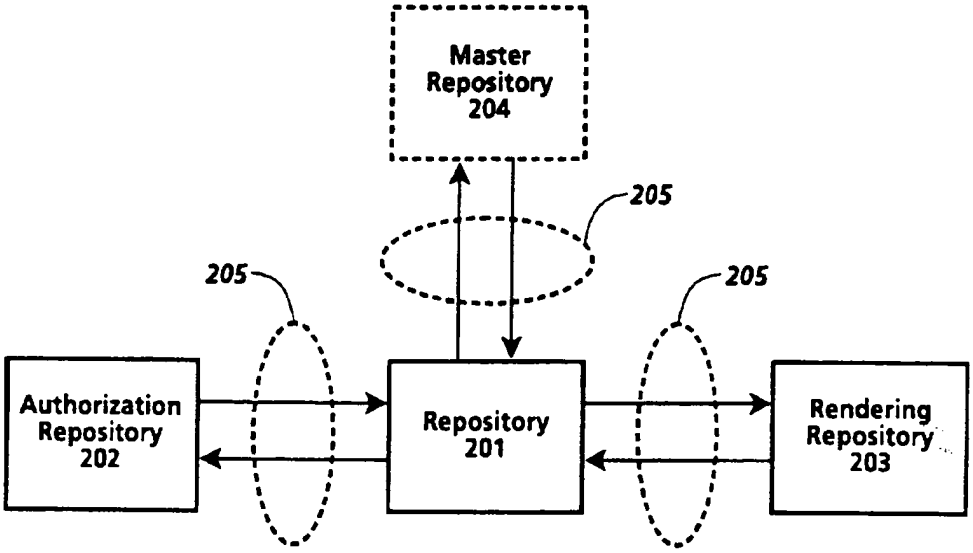


Fig. 2

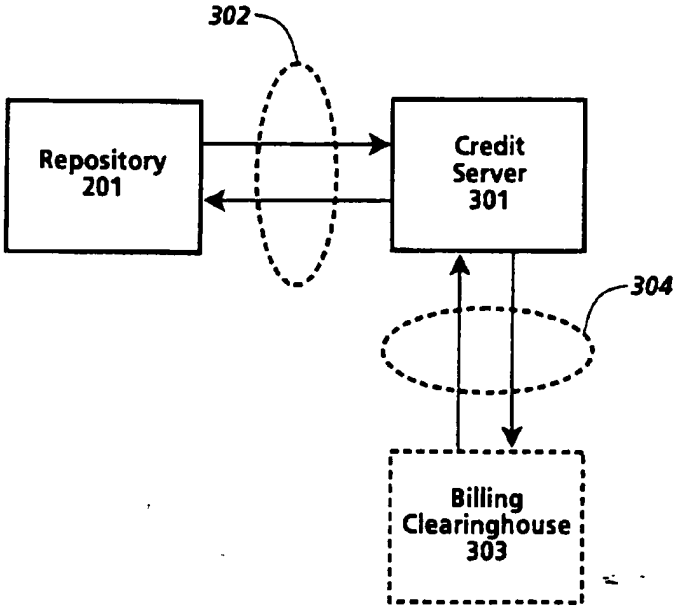


Fig. 3

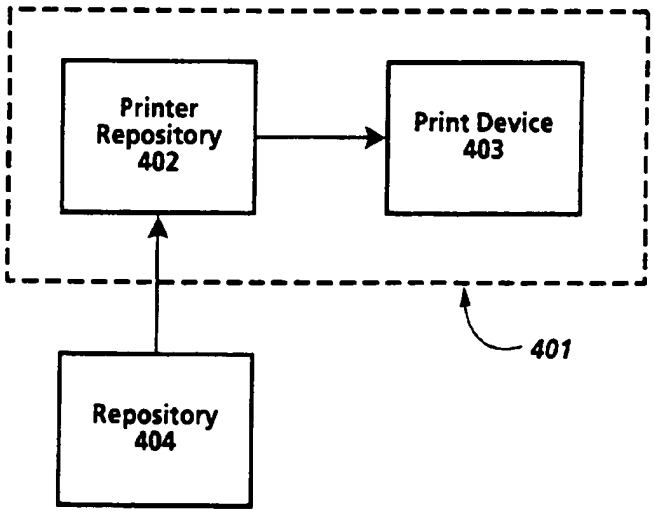


Fig. 4a

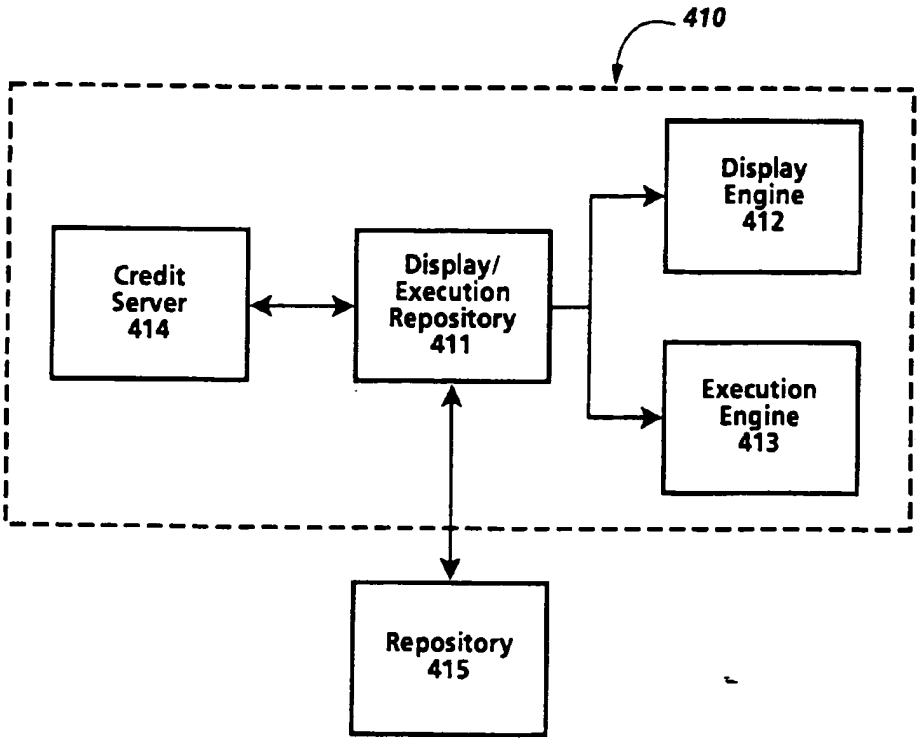


Fig. 4b

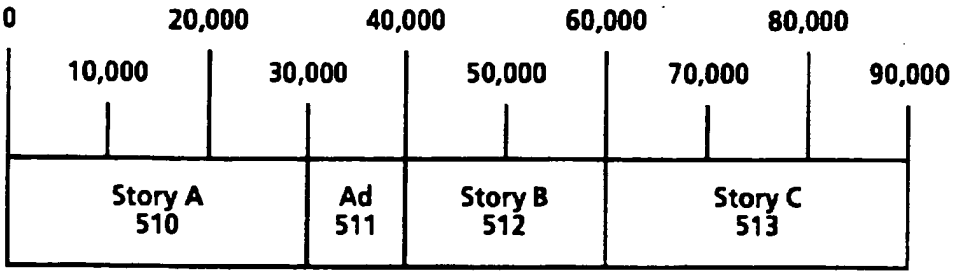


Fig. 5

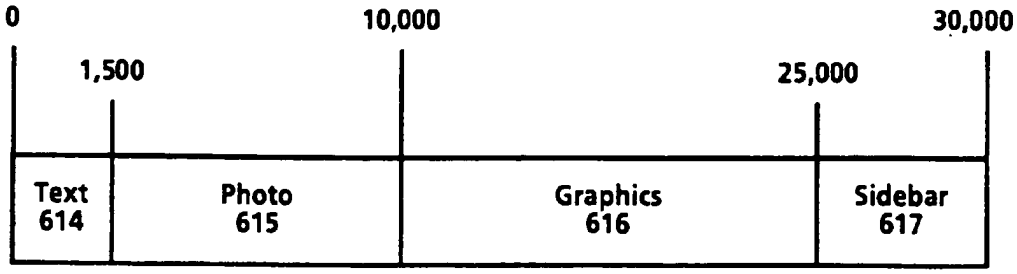


Fig. 6

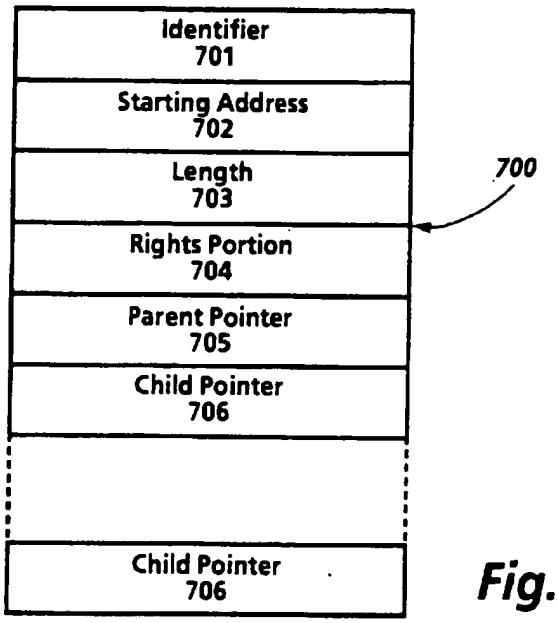


Fig. 7

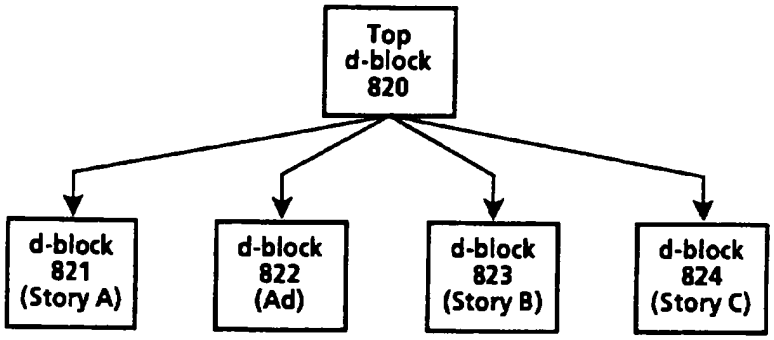


Fig. 8

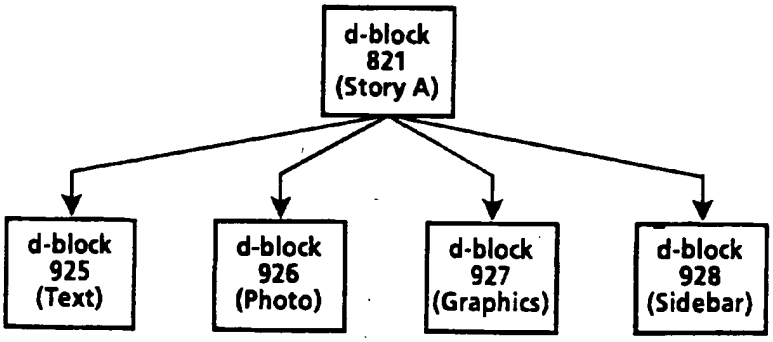


Fig. 9

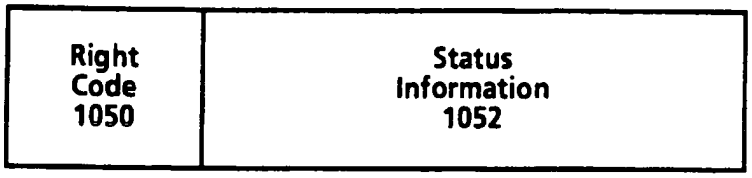


Fig.10

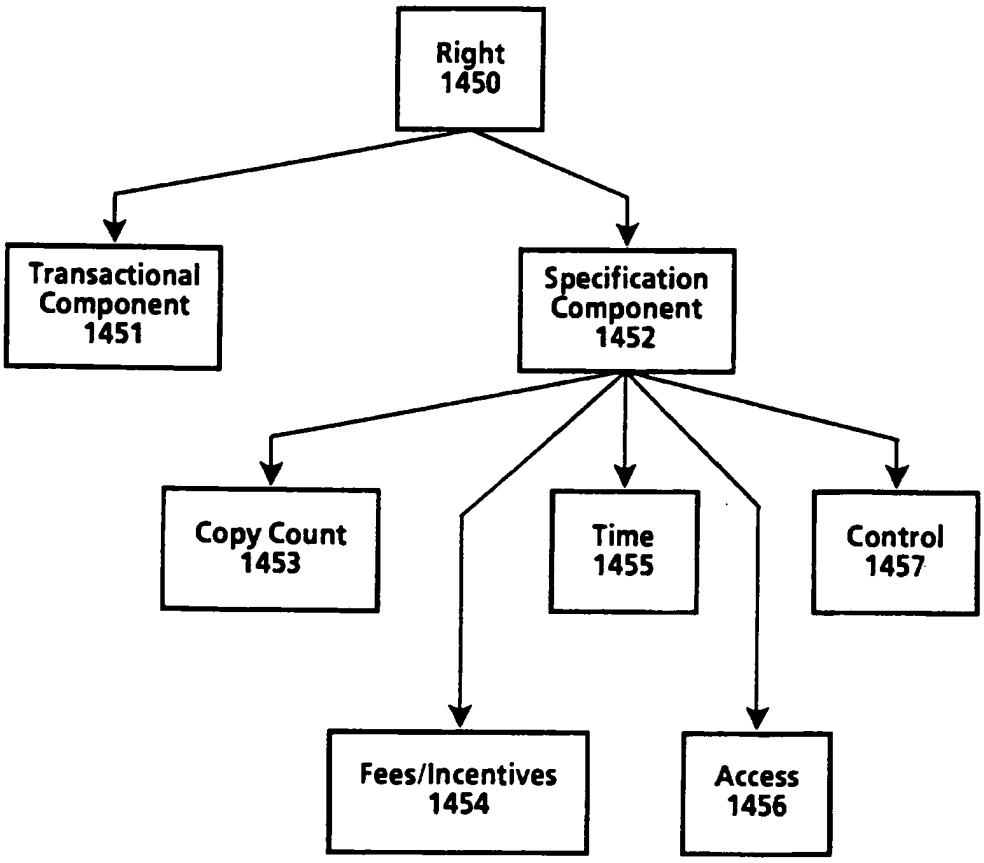


Fig.14

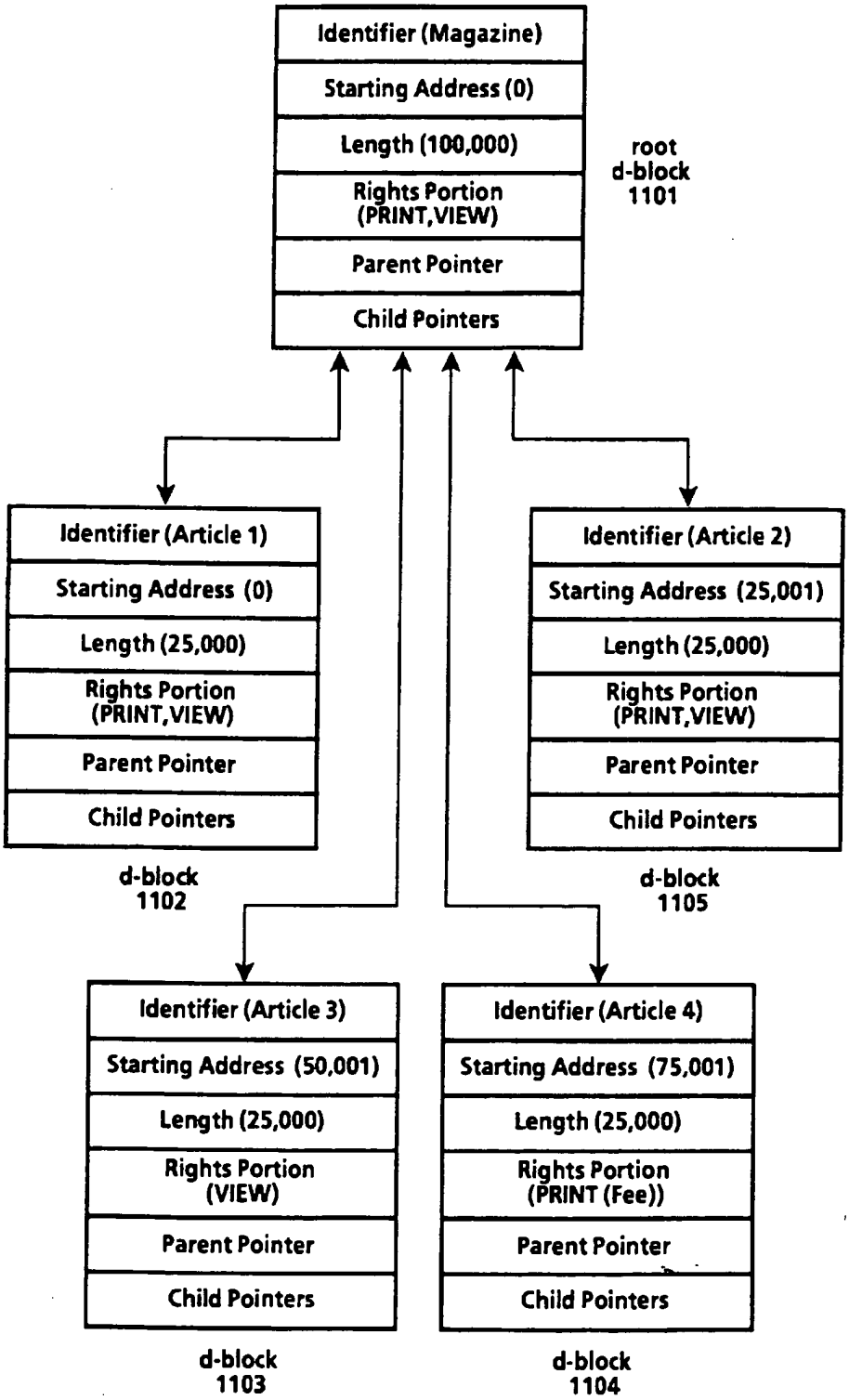


Fig.11

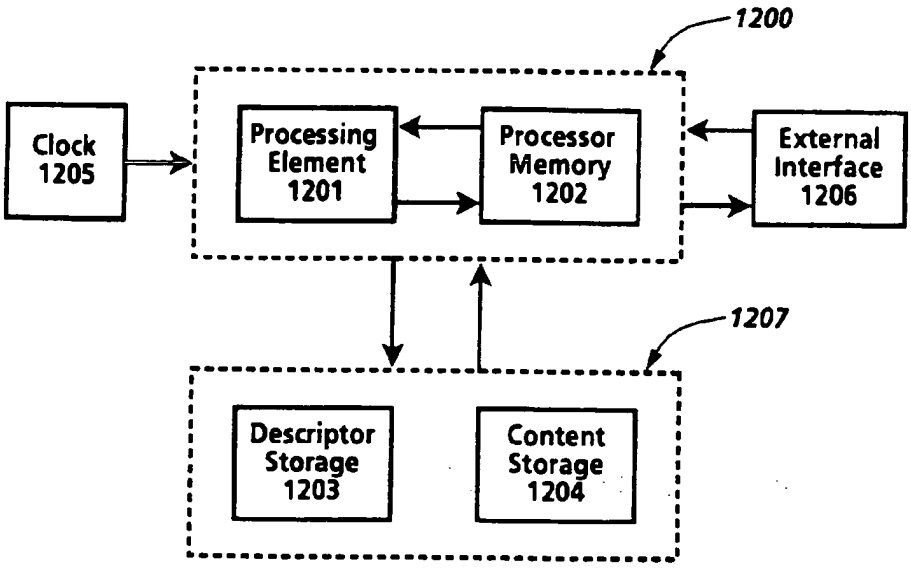


Fig.12

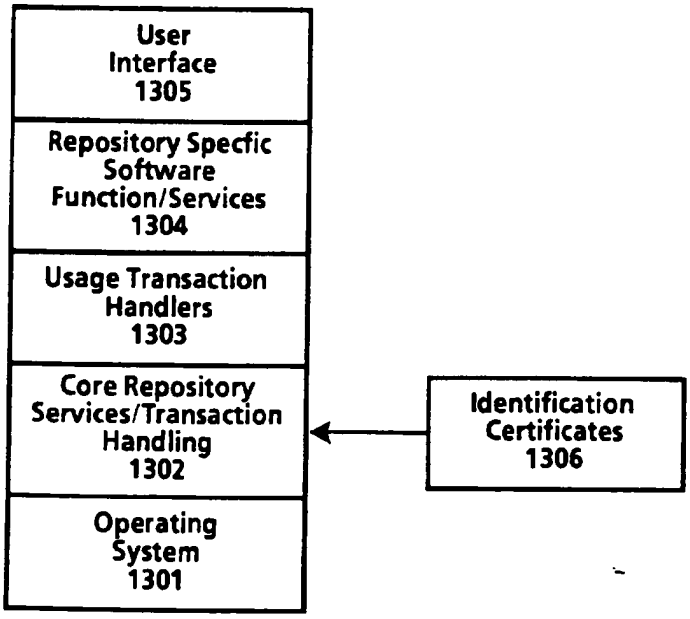


Fig.13

1501 ~ **Digital Work Rights**: = (Rights*)
1502 ~ **Right** : = (Right-Code {Copy-Count} {Control-Spec} {Time-Spec }
{Access-Spec} {Fee-Spec})
1503 ~ **Right-Code** : = Render-Code | Transport-Code | File-Management-
Code| Derivative-Works- Code | Configuration-Code
1504 ~ **Render-Code** : = [Play : {Player: Player-ID} | Print: {Printer: Printer-ID}]
1505 ~ **Transport-Code** : = [Copy | Transfer | Loan {Remaining-Rights:
Next-Set-of-Rights}]{(Next-Copy-Rights: Next-Set-of-Rights)}
1506 ~ **File-Management-Code** : = Backup {Back-Up-Copy-Rights:
Next-Set-of-Rights} | Restore | Delete | Folder
| Directory {Name: Hide-Local | Hide-Remote}
{Parts: Hide-Local | Hide-Remote}
1507 ~ **Derivative-Works-Code** : = [Extract | Embed | Edit{Process:
Process-ID}] {Next-Copy-Rights :
Next-Set-of Rights}
1508 ~ **Configuration-Code** : = Install | Uninstall
1509 ~ **Next-Set-of-Rights** : = {(Add: Set-Of-Rights)} {(Delete:
Set-Of-Rights)} {(Replace: Set-Of-Rights)}{(Keep: Set-Of-Rights)}
1510 ~ **Copy-Count** : = (Copies:positive-integer | 0 | Unlimited)
1511 ~ **Control-Spec** : = (Control: {Restrictable | Unrestrictable}
{Unchargeable | Chargeable})
1512 ~ **Time-Spec** : = ({Fixed-Interval | Sliding-Interval | Meter-Time}
Until: Expiration-Date)
1513 ~ **Fixed-Interval** : = From: Start-Time
1514 ~ **Sliding-Interval** : = Interval: Use-Duration
1515 ~ **Meter-Time**: = Time-Remaining: Remaining-Use
1516 ~ **Access-Spec** : = ({SC: Security-Class} {Authorization: Authorization-ID*}
{Other-Authorization: Authorization-ID*} {Ticket: Ticket-ID})
1517 ~ **Fee-Spec**: = {Scheduled-Discount} Regular-Fee-Spec | Scheduled-Fee-Spec |
Markup-Spec
1518 ~ **Scheduled-Discount**: = Scheduled-Discount: (Scheduled-Discount:
(Time-Spec Percentage)*)
1519 ~ **Regular-Fee-Spec** : = ({Fee: | Incentive: } [Per-Use-Spec | Metered-Rate-
Spec | Best-Price-Spec | Call-For-Price-Spec]
{Min: Money-Unit Per: Time-Spec}{Max:
Money-Unit Per: Time-Spec} To: Account-ID)
1520 ~ **Per-Use-Spec**: = Per-Use: Money-unit
1521 ~ **Metered-Rate-Spec** : = Metered: Money-Unit Per: Time-Spec
1522 ~ **Best-Price-Spec** : = Best-Price: Money-unit Max: Money-unit
1523 ~ **Call-For-Price-Spec** : = Call-For -Price
1524 ~ **Scheduled-Fee-Spec**: = (Schedule: (Time-Spec Regular-Fee-Spec)*)
1525 ~ **Markup-Spec**: = Markup: percentage To: Account-ID

Fig.15

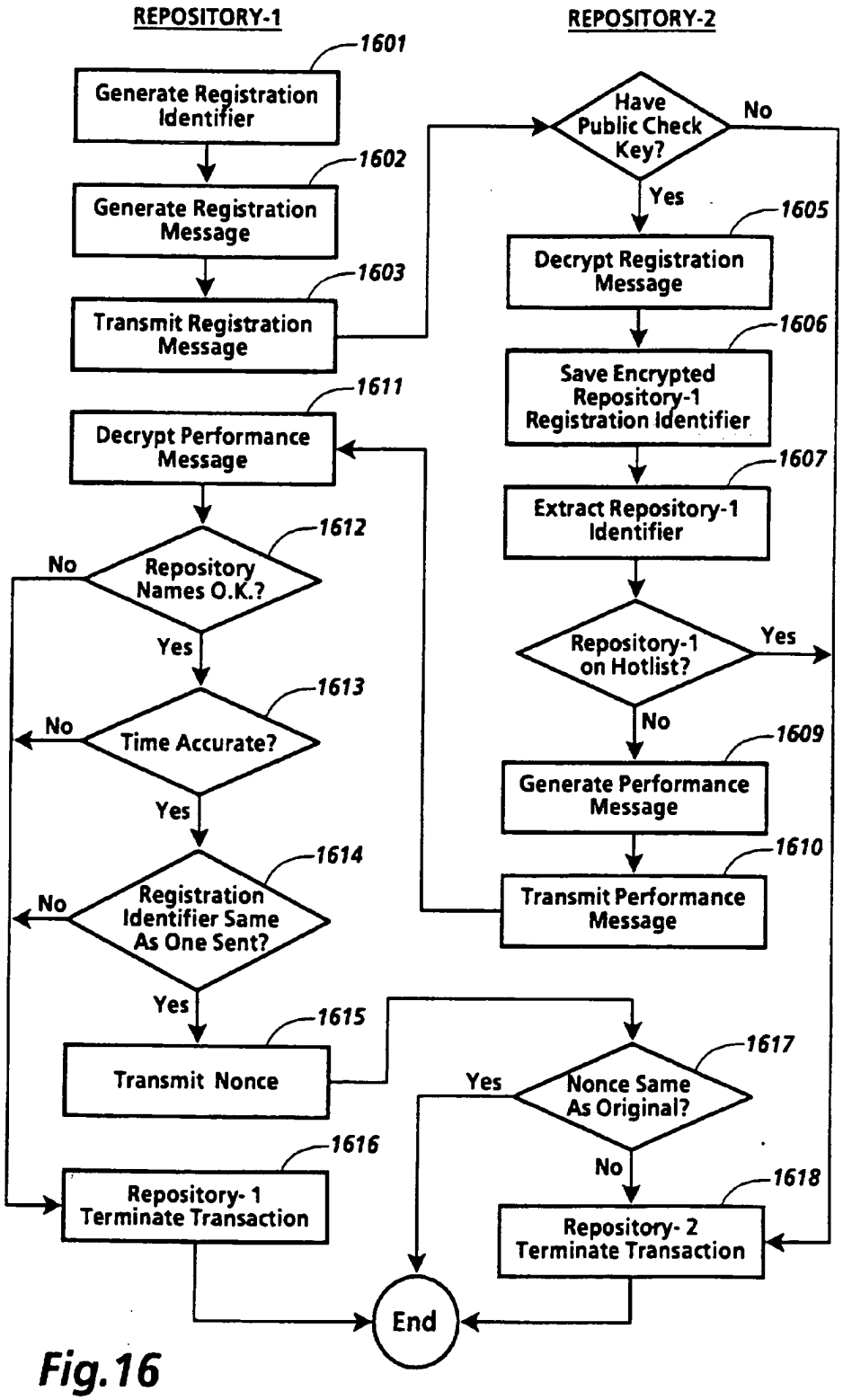


Fig.16

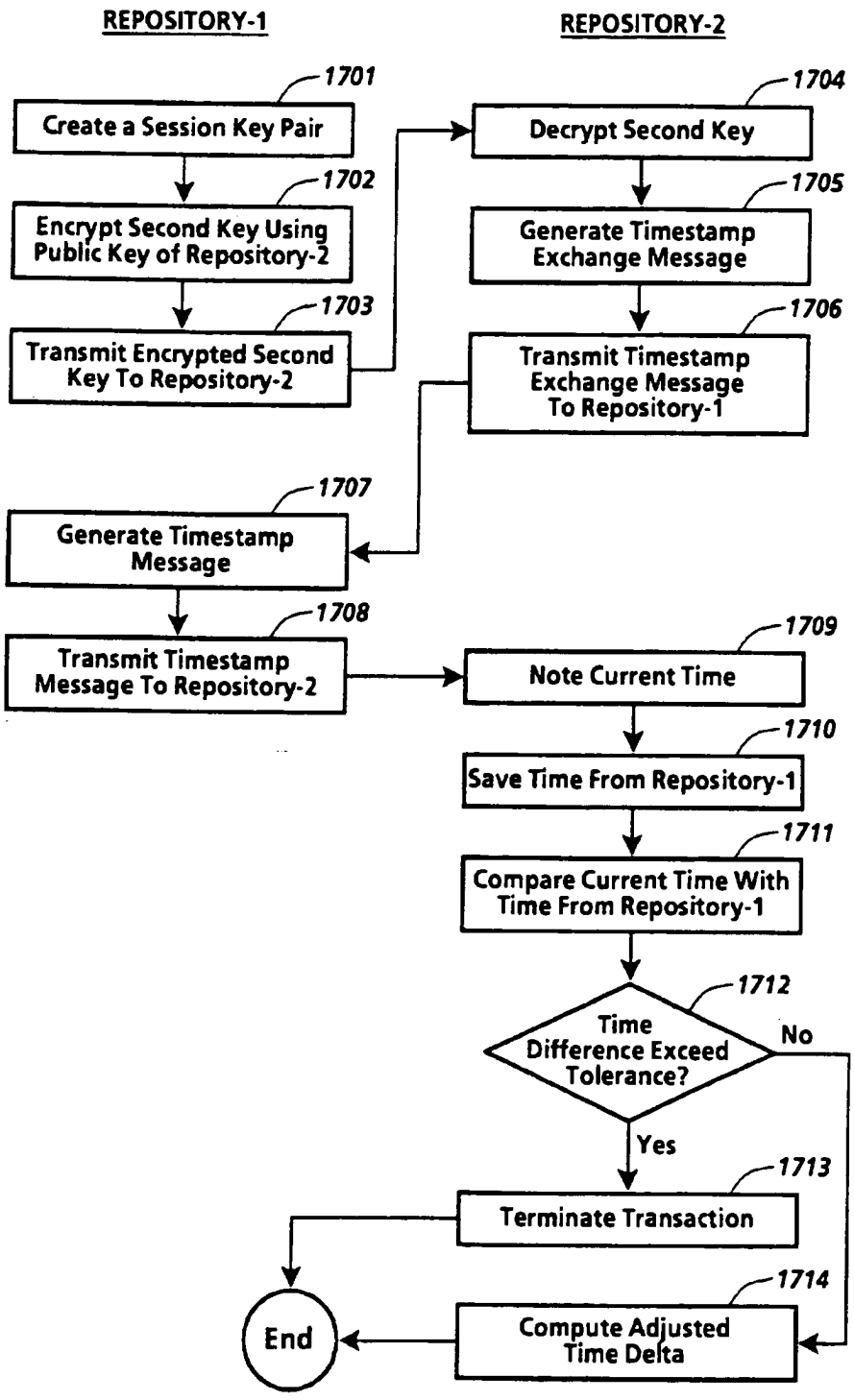


Fig.17

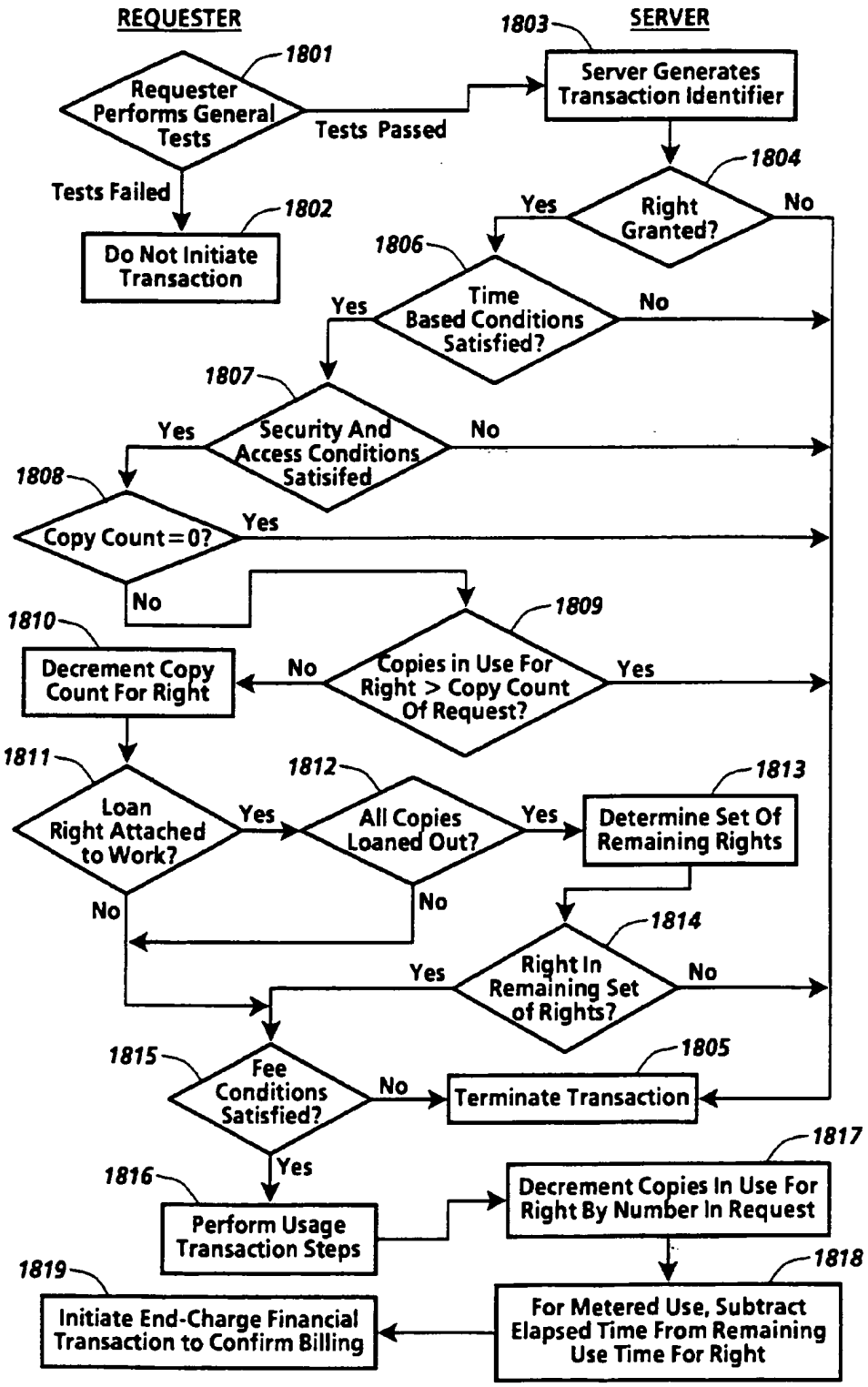


Fig.18

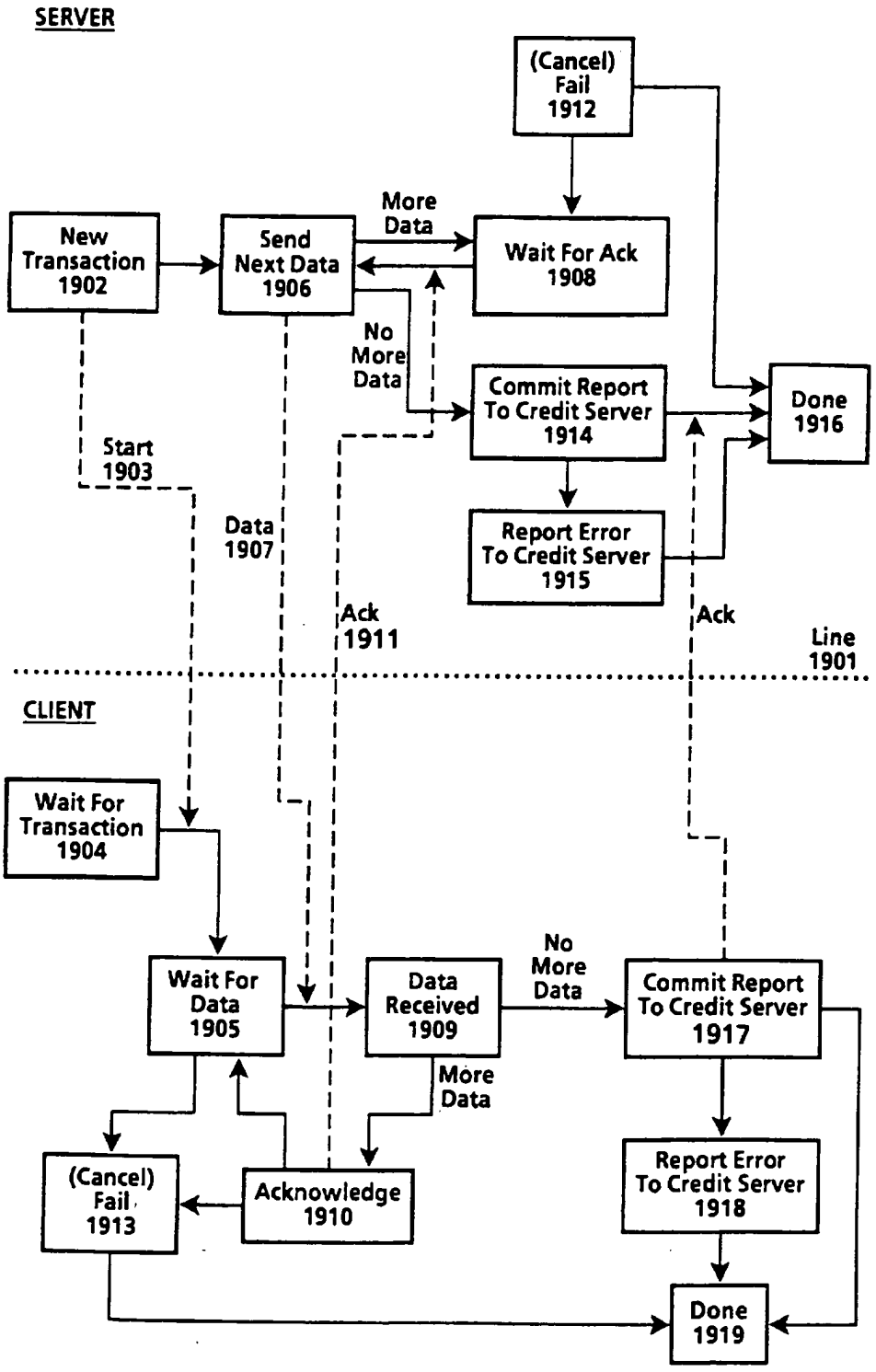


Fig.19

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 8420

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	WO-A-92 20022 (DIGITAL EQUIPMENT CORP.) * page 45, line 10 - page 64, line 17 * ---	1,6,8,10	G06F1/00
A	GB-A-2 236 604 (SUN MICROSYSTEMS INC) * page 9, line 11 - page 20, line 15 * ---	1,6,8,10	
A	US-A-5 291 596 (MITA) * the whole document * -----	1,6,8,10	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	1 April 1996	Moens, R

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 731 404 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
11.09.1996 Bulletin 1996/37

(51) Int. Cl.[6]: **G06F 1/00**, G06F 19/00

(21) Application number: **96100832.3**

(22) Date of filing: **22.01.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **07.03.1995 US 401484**

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

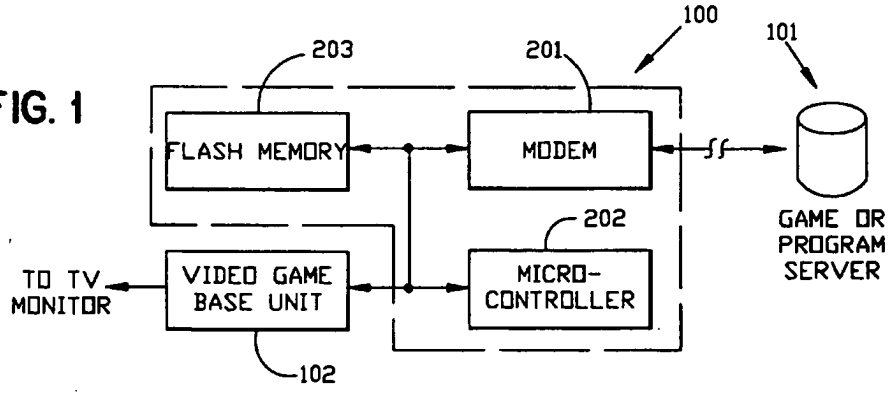
(72) Inventors:
- **Bakoglu, Halil Burhan**
  Ossining, New York 10562 (US)
- **Chen, Inching**
  Wappingers Falls, New York 12590 (US)
- **Lean, Andy Geng-Chyun**
  Merrick, New York 11566 (US)
- **Maruyama, Kiyoshi**
  Chappaqua, New York 10514 (US)
- **Yue, Chung-wai**
  Yorktown Heights, New York 10598 (US)

(74) Representative: **Rach, Werner, Dr.**
IBM Deutschland
Informationssysteme GmbH,
Patentwesen und Urheberrecht
70548 Stuttgart (DE)

(54) A universal electronic video game renting/distributing system

(57) A video game cartridge that can be plugged into a video game machine to enable a user to request and play a video game for a predetermined number of video frames. The cartridge has a receiver for receiving the video game program and the predetermined frame count in response to a request from the user. The program and frame count is then stored in a memory of the cartridge. Finally, the cartridge has a counter which changes its value when the user is actively playing the video game program. The counter ceases to change its value when the user is not playing the video game program. When the counter reaches a predetermined limit, the user is no longer authorized to play the video game program.

FIG. 1



EP 0 731 404 A1

Printed by Rank Xerox (UK) Business Services
2.13.4/3.4

Description

Technical Field

This invention relates to a video game cartridge for receiving video game programs from a remote server.

Description of the Prior Art

Today, there are many video games available for purchase or for rental at stores. Generally, there is no trial or test playing of the games in the stores, and there is no return on purchased games once the game package has been opened. Therefore, a person who is interested in any game has to buy it before playing it and thus may face the risk of not liking the game later. There is no return or refund of the game since the package has been opened. A person who rents a game from a store has to go through the usual VCR tape rental trouble of driving to the store, picking up the game and then later returning the game to the store.

To make video game rental easier for the consumer, Sega has created the Sega Channel. In this service, via cable and using a cable adapter unit which is plugged into the Sega Genesis game machine, people can play games that are downloaded to the cable adapter. It requires the on-line Sega Channel connection as well as the special adapter while the game is being played.

Down loading a software program to a personal computer over the modem connection exists today. Such software can come with a limited life where the life can be specified by expiration date, or time, or the number of times of the software usage. These schemes in limiting the software usage is not applicable to down loading video games to cartridges which are plugged into existing video game base units because these game base units do not have timer device built in. Thus a new scheme for controlling the usage of the game is needed.

The US Patent 4,905,280 to J.D. Wiedemer, et al describes a method for real time down loading of broadcast programs for pay-per-view or for subscription. Descrambling of broadcast programs is done by codes on a replaceable memory module, which is delivered to a subscriber by the service provider. This patent is applicable to the "purchase" of software content or real-time service, but it is not applicable to limiting the life of rented software.

US patent 5,251,909 to Reed et al describes software renting or distributing schemes in which access is granted to a subscriber prior to the actual programs being transmitted. This patent describes an off-line process and is not applicable to delivering software for rental purposes.

Summary of the Invention

It is an object of this invention to provide a portable video game cartridge which can be plugged into a video game machine base unit, such as Nintendos, Sega Genesis&tm. video game machine or Atari's Jaguar&tm. video game machine. The cartridge will allow a video game program to be used by receiving the video program over a telephone network or cable system.

The current invention describes a way of distributing and controlling the usage of a video game program (or any software program) by using a "watchdog mechanism" and by limiting the "life" of a game by limiting the total number of graphic frames that a video machine can generate. It offers a simple and effective way of software renting and distribution where game machines have no timer.

It is also an object of this invention to prevent piracy of video programs and programs in general by storing the frame count in a random location of the memory that is unknown to a potential pirate, especially if the count itself is encrypted. Since the count is part of the video game program or program execution path, the video game or program cannot be used without knowledge of the count.

This invention is generally an apparatus and method for enabling a user to request and use a program where the user receives the program and a frame count indicating the number of frames of the program that the user is authorized to execute or use. This program and the frame count is then stored in a memory. When the user is actively providing input to the program, the frame count changes. The frame count will cease to change when the user is not providing input to the program. When the count reaches a predetermined limit, the user is prevented from continuing use of the program.

This invention is a video game cartridge which can be plugged into a video game machine for enabling a user to receive and play a video game for a predetermined number of frames. The cartridge has a receiver for receiving the video program and for receiving a frame count indicating the number of video frames of the video game program that the user is authorized to play. The video program and frame count is then stored in a memory of the cartridge. The cartridge also has a counter which changes the frame count when the user is actively playing the video game program. When the user is not playing the video game program, the counter ceases to change its count. Finally when the counter reaches a predetermined limit, the user is prevented from further playing the video game program.

Brief Description of the Drawings

FIG. 1 schematically illustrates the major components of the video game cartridge along with a video game machine and a remote server.

FIG. 2 is a functional diagram showing the functions of each of the major components of the video game cartridge.

FIG. 3 schematically illustrates the flow chart for the watch "dog mechanism".

Description of the Preferred Embodiment

FIG. 1 illustrates a sample diagram of a electronic game or program renting system setup. The dotted line encloses the portable and programmable game cartridge unit 100 that can be plugged into a video game machine base unit 102, such as Sega Genesis&tm. video game machine, and remotely be connected to a video game server 101 via a modem connection. The connection to the remote video server can be through cable TV, or other telecommunication facilities.

When a video game base unit 102 is powered on, a user could either play a game (or games) stored in the programmable game cartridge 100 or place an order of a new game (either for rental or for purchase) to the game or program server 101. The cartridge 100 contains screen assistance (and voice assistance) to help place an order for a video game program to the server 101.

FIG. 2 illustrates the components of the video game cartridge unit 100. It consists of modem 201, microcontroller 202, flash memory 203 and an interface 204 to the video game base unit 102. The modem 201 performs the interface to the telephone or cable network. It can optionally perform decompression of received game or software if necessary. The received game is stored in flash memory 203. The game comes with its "life" which is indicated by the total number of graphic frames the video game machine 102 is authorized to generate when the game is actively played. For example, the game machine could render game graphics frame by frame at the rate of thirty framers per second.

After the number of graphic frames is exhausted, further playing of the game is prevented by the following mechanism. The flash memory 203 also stores a "watchdog mechanism" which keeps track of the remaining life of the game. An hourglass routine is embedded in the watchdog mechanism which is executed by microcontroller 202. This watchdog mechanism updates and tracks down a specified register in the flash memory 203 with its location randomly determined by the game server 101 in FIG. 1 during the down loading of the game.

The use of expiration date or time for voiding the game is an obvious approach if the video game base unit 102 comes with a timer. Since this patent application assumes a game base unit 102 which has no timer (which is the case of many existing game machines), the "life" of the rented game is determined by the total number of graphic frames that the base game unit can generate. This "life", or frame count, is what a renter gets when a game is down loaded. It is stored into a location in the flash memory 203. The location into which the frame count is stored in the flash memory is determined randomly by the video server at the time of the game down loading. The video game can resume at any time when it is being turned on, provided there is available frame count stored in the designated random location. The microcontroller 202 can pick up the frame count and allow the renting period, and thus the game or software, to be continued. As the rented game is being played, the frame count is decremented. When the user turns off the power, the hourglass routine in memory 203 will first store the remaining frame count to a random location in the non- volatile memory 203 and then shut down the game. The rental expires when there is no frame count remaining. The microcontroller 202 will not allow any portion of the game to be played by the game base unit 102 when the frame count reaches zero.

FIG. 3 illustrates the watchdog mechanism embedded with the video game program execution path that contains the hourglass routine which serves as part of the watchdog mechanism which can expire the game. When the user starts the game, the frame count is first fetched (305) and checked (306). If the frame count reaches zero, the game is over even though the game unit still has its power on (306N). If the frame count is still greater than zero (306Y), the scanner continues to monitor the game player's input in playing the video game (307). No active input (307) means the player is not playing the video game, and the scanner continues to monitor the player inputs from the key pad connected to the video game. When there is no active input, the video game will not render any game graphic frames. Therefore, the game program execution path will fall through decisions 308 and 309 and immediately return to continue scanning (307). When the game is not actively played and the player leaves the game machine's power on, the game will be sitting idle without rendering any new graphic frames. The frame count will not be consumed until the player becomes active again in playing the game as detected by the scanner (307 and 308).

If the player's input has been detected as active (307), a check is made to see if graphic rendering is required (309). Graphics rendering is required when the game program determines that the input signals from the key pad connected to the video game are valid signals. If rendering is required (309Y), the frame counter will be decremented (301). The hourglass routine (301 and 302) decrements the frame count and checks for any frame count left.

If the count is valid (302Y), then the program flows back to (310) which is the game program main collections, and then at the same time, 302 Y:sup.':esup. branches to check for power-off condition (303).

If the user decides to power-off the game, the watchdog mechanism will go through decision (303) and the shutdown routine (304) to store any remaining frame count in the flash memory. The shutdown routine stores the remaining frame count in the flash memory and exits the game. In summary flowchart components (301-306) and their associated flash memory form the "watchdog mechanism" that contains the hourglass rou-

tine (301 and 302) to keep track of the games "life" (remaining frame count). The watchdog mechanism also insures that the game can be resumed if there is still a valid frame count in the flash memory. Microcontroller (202) can also give advance warning when the rental is about to expire. Rental extension, if desired, can be downloaded again by the server (101) through a telephone or cable connection. Thus, server (101) in FIG. 1 has complete control over the game playing time, which should reflect the user's request for renting the game.

Although this embodiment was described in terms of a video game program in a cartridge, this invention can be extended to software programs in general. As long as the programs monitor user inputs, a scanner and watchdog mechanism can be implemented in similar fashion using a non-volatile memory.

The watchdog mechanism can even be made more secure by encrypting the frame count, which is stored at a random location in the memory. Even if the would-be pirate stumbles across the count in the memory, he/she wouldn't know what he/she found.

Claims

1. An apparatus for enabling a user to request and use a program, said apparatus comprising:

   a. a receiver for receiving the program and a frame count indicating a number of frames of the program that is authorized to be executed by the user;

   b. a memory for storing the program and the frame count received by the receiver; and

   c. a counter for changing the frame count when the user is actively providing input to the program, wherein the counter ceases to change its count when the user is not providing input to the program, and wherein the user is prevented from continuing use of the program when the counter reaches a predetermined limit.

2. An apparatus as recited in claim 1, further comprising:

   means for randomly determining an address in the memory in which the frame count is to be stored, and wherein the address is unknown to the user.

3. A method of enabling a user to request and use a program, said method comprising:

   a. receiving the game program and a frame count indicating a number of frames of the program that is authorized to be used by the user in response to a request;

   b. a memory for storing the program and the frame count; and

   c. changing the frame count when the user is actively using the program, wherein the frame count ceases to change when the user is not using the program and wherein the user is prevented from continuing use of the program when the counter reaches a predetermined limit.

4. A method as recited in claim 3, wherein the frame count is stored in a randomly determined location in the memory.

5. A video game cartridge which can be plugged into, for operation with, a video game machine to enable a user to request and play a video game program which is received from a remotely located server, said video game cartridge comprising:

   a. a receiver for receiving from the server the video game program and a frame count indicating a number of frames of the video game program that is authorized to be played by the user in response to a request;

   b. a memory for storing the video game program and the frame count received by the receiver; and

   c. a counter for changing the frame count when the user is actively playing the video game program, wherein the counter ceases to change its count when the user is not playing the video game program, and wherein the user is prevented from further playing the video game program when the counter reaches a predetermined limit, indicating that the user has played said video game for the number of frames.

6. A video game cartridge as recited in claim 5, further comprising:

   means for randomly determining an address in the memory in which the frame count is to be stored.

7. A video game cartridge as recited in claim 5, further comprising:

   a modem for transmitting to the server the request from the user to play a video game program.

8. A video game cartridge, as recited in claim 5, wherein said memory is a non-volatile memory.

9. A video game cartridge, as recited in claim 8, wherein the frame count indicated in the counter is stored in the memory when power for the video game machine is turned off.

10. A video game cartridge, as recited in claim 9, further comprising:

a means for fetching the frame count stored in the memory when power for said game machine is turned on.

11. A video game cartridge which can be plugged into, for operation with, a video game machine to enable a user to request and play a video game program which is received from a remotely located server, said video game cartridge comprising:

a. a modem for transmitting from the user over a telephone or cable network a request to receive the video game from the server, and for receiving the video game program and frame count from the server over the telephone or cable network, the frame count indicating a predetermined number of frames of the video game program that is authorized to be played by the user in response to the request;

b. a non-volatile memory for storing the video game program and the frame count;

c. a counter for changing the frame count when the player is actively playing the video game;

d. a means for storing the changed frame count of the counter in the memory when the power to the video game machine is turned off; and

e. a means for fetching the changed frame count stored in the memory in step (d) when the player resumes playing the video game, wherein the user is prevented from further playing of the video game program when the frame count of the counter reaches a predetermined limit, indicating that the user has played said video game for the predetermined number of frames.

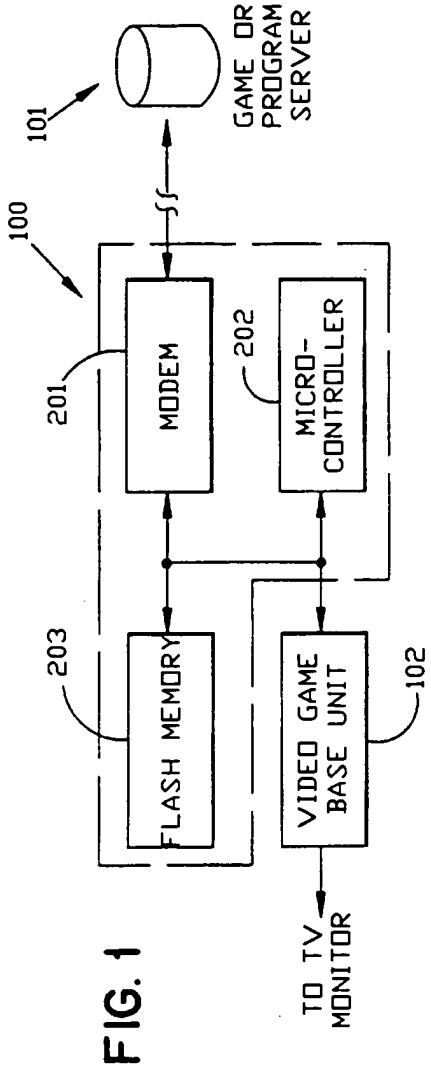


FIG. 1

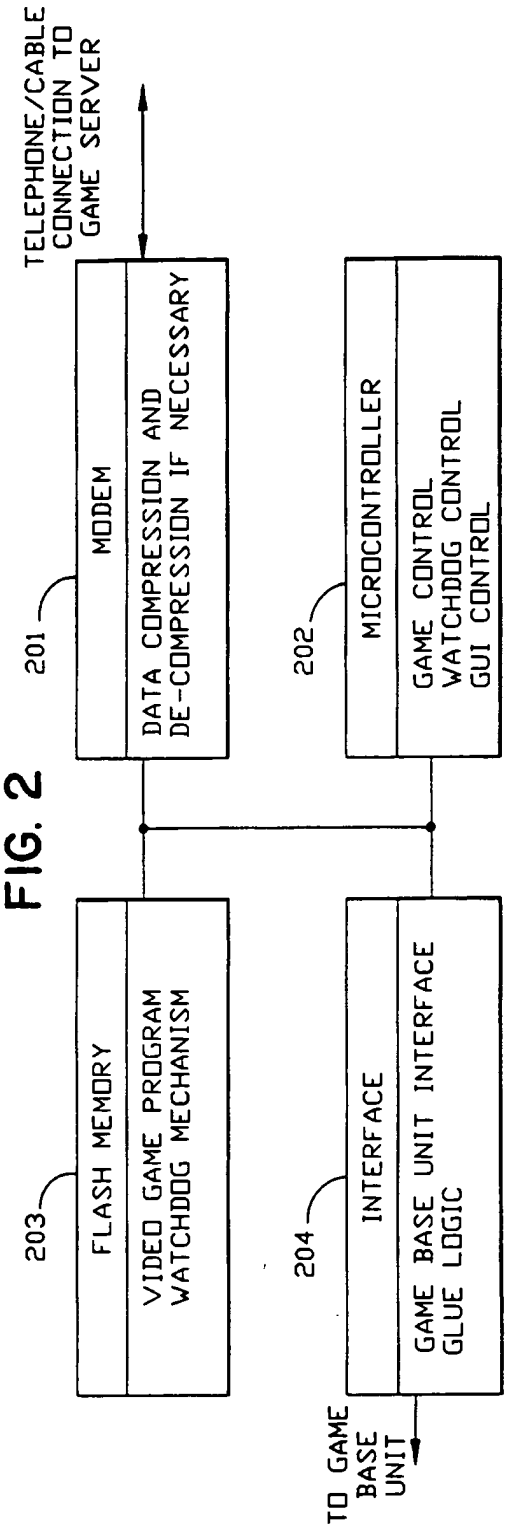
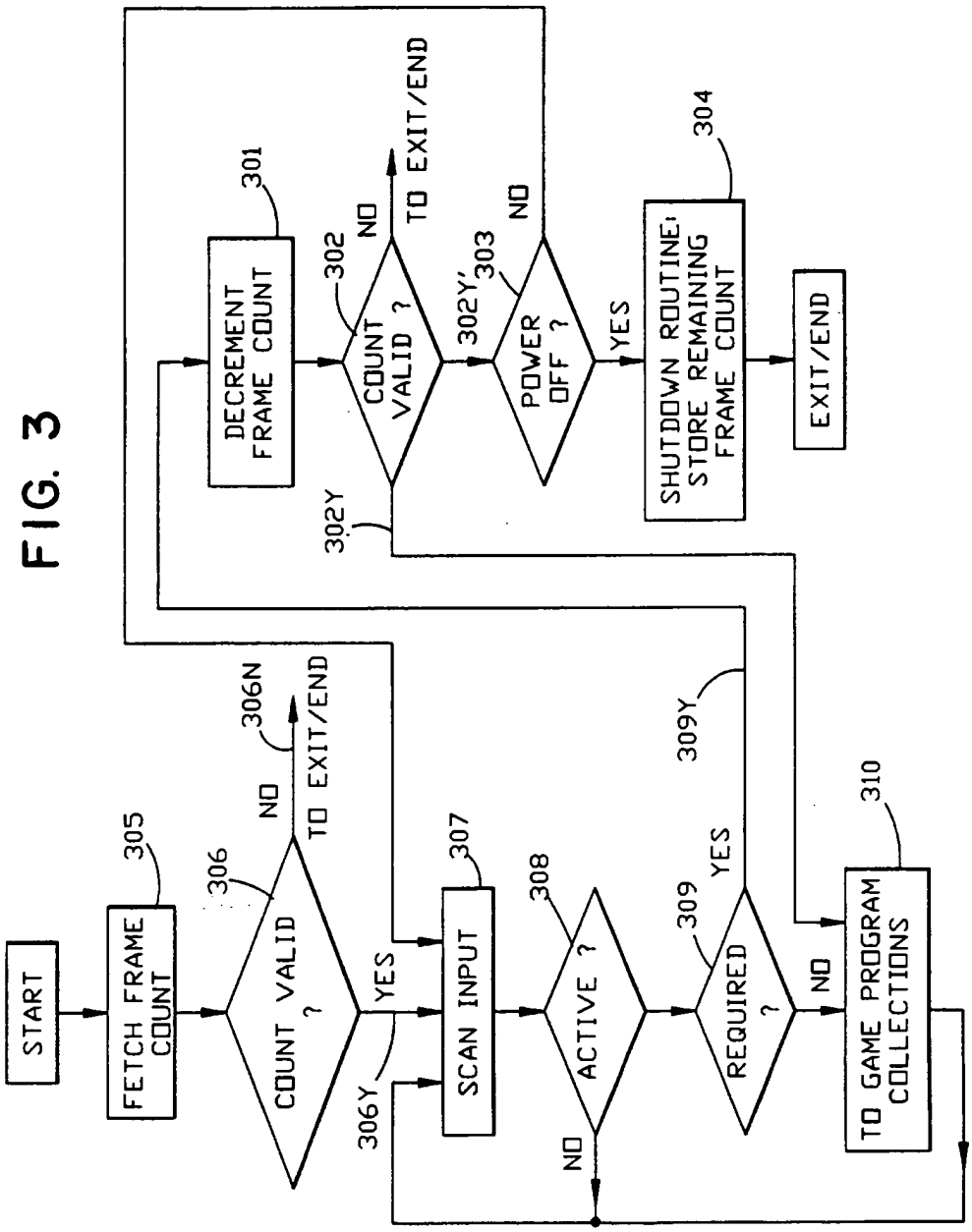


FIG. 2

FIG. 3



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 10 0832

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
P,A	EP-A-0 671 711 (SEGA ENTERPRISES KK) 13 September 1995 * the whole document * ---	1-11	G06F1/00 G06F19/00
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 3, 1 March 1994, pages 413-417, XP000441522 "MULTIMEDIA MIXED OBJECT ENVELOPES SUPORTING A GRADUATED FEE SCHEME VIA ENCRYPTION" * page 413, line 1 - page 414, line 14 * ---	1-11	
A	WO-A-93 01550 (INFOLOGIC SOFTWARE INC) 21 January 1993 * page 1, line 1 - page 8, line 32 * * claims 1-3 * -----	1-11	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 June 1996	Powell, D

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 763 936 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
19.03.1997 Bulletin 1997/12

(51) Int Cl.[6]: H04N 5/913

011

(21) Application number: 96306507.3

(22) Date of filing: 06.09.1996

(84) Designated Contracting States:
DE FR GB

(30) Priority: 18.09.1995 KR 9530444

(71) Applicant: LG ELECTRONICS INC.
Seoul (KR)

(72) Inventors:
- Kim, Yung Gil, c/o LG Elec. Video-Media R&D
Seoul (KR)
- Park, Tae Joon
Seoul (KR)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT
27 Furnival Street
London EC4A 1PQ (GB)

(54) Illegal view/copy protection method and apparatus for digital broadcasting system

(57) An illegal view/copy protection method for a digital broadcasting system is disclosed including an audio/video signal transmission step (200,201) for multiplexing and transmitting audio/video bit stream scrambled in control words (206) and information where the control words and CPTC information for illegal view/copy protection are encrypted (208); and an audio/video reception step (202) for decrypting (210) the transmitted bit stream to analyze the CPTC information and control words (211), deciding whether recording is allowed or not to be recorded on cassette tape, and using the control words, performing descrambling (212) and decoding (213) to output audio/video signals to a monitor, thereby protecting copyright.

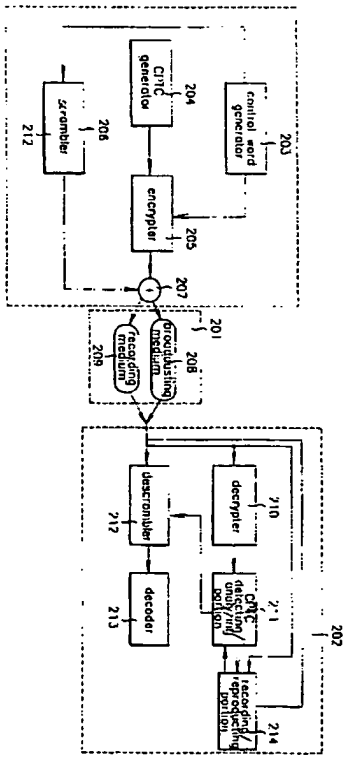


FIG.16

EP 0 763 936 A2

Printed by Jouve, 75001 PARIS (FR)

Description

Background of the Invention

The present invention relates to an illegal view/copy protection method and apparatus for a digital broadcasting system, in which digital broadcasting performed through broadcasting media such as cable, satellite and terrestrial broadcasting, or through prerecorded media such as video cassette tapes, is prevented from being illegally viewed or copied to thereby protect its copyright.

For conventional systems for copyright protection on digital media, there are Macrovision's intellectual property protection system (IPPS), which is disclosed in US Patent No. 5,315,448, and the integrated receiver/decoder (IRD), a conditional receiving system for digital broadcasting media, for receiving DirecTV's satellite broadcasting currently transmitted in the US.

The Macrovision's IPPS disclosed in US Patent No. 5,315,448 is a copy protection system for a hybrid digital VCR having digital recording functions for both a digital input signal and an analog input signal.

As shown in Figs. 1 and 2, in operating its copy protection function, Macrovision's IPPS detects, when a digital signal is input, copy protection control bits from an input signal, and when an analog signal is input, detects the analog copy protection waveform from the input signal.

More specifically, as shown in Fig. 2, a signal in which the analog copy protection waveform generated from an analog copy protection generator is added to the analog video output of the output signals of the digital VCR is output and displayed to be normal on an analog TV but distorted on an analog VCR, as shown in Fig. 1. In digital recording of the input signal, the copy protection control bits are changed to prevent digital copy or to permit one-time digital copy.

Referring to Fig. 3, the IPPS comprises an analog copy protection detector (ACP) 2 for detecting the analog copy protection waveform from an input analog NTSC video signal 1, an A/D converter 3 for A/D-converting analog NTSC video signal 1 input according to the signal output from the ACP detector, an AC bit detector 5 for detecting the AC bit from input digital video signal 4, an SCPS bit detector 6 for detecting the SCPS from input digital video signal 4, an AC bit adder 7 for adding the AC bit to input digital video signal 4 according to the SCPS bit output from SCPS bit detector 6, a switch 8 for outputting a signal output from AC bit adder 7 according to the AC bit output from AC bit detector 5, a switch 9 for selecting and outputting the signal output from A/D converter 3 and switch 8, a digital tape deck mechanism/circuit 10 for digitally recording the signal output from switch 9 and outputting a digital video signal, an AC bit detector 11 for detecting the AC bit from the signal output from digital tape deck mechanism/circuit 10, an ACP signal generator 12 for generating the ACP signal from the signal output from AC bit detector 11, and a D/A converter 13 for adding the ACP signal output from ACP signal generator 12 to the signal output from digital tape deck mechanism/circuit 10 and D/A converting the added result which is output as an analog NTSC video signal.

The operation of the IPPS will be explained below.

The copy protection control bits are made up of the AC and SCPS bits. The AC bit is added to recorded digital video data so that if the AC bit is set, digital copy is prohibited and if the SCPS bit is set, one-time digital copy is allowed.

In playback, when the AC bit is detected by AC bit detector 11, the analog copy protection waveform generated from ACP signal generator 12 is added to the analog video signal, which is output to D/A converter 13. Here, as the position of the copy protection control bits of the digital video data, an area of an MPEC-2 digital copy protection header where one-bit copyright flag and one-bit original-or-copy flag of a PES header are placed is used, or a transport-private-data field area of the transport header of the MPEC-2 is used.

The analog copy protection waveform is a signal which is severely distorted when inserted into the analog NTSC waveform and directly coupled to the analog TV. A method of generating such a signal is presented in US Patents Nos. 4,613,603 and 4,914,694. Using this method, the IPPS generates the analog copy protection waveform.

Referring to Fig. 4, the IRD, as a conditional receiving system for digital broadcasting media, for receiving the DirecTV's satellite broadcasting currently transmitted in US comprises an outdoor unit (ODU) 21 made up of a satellite antenna for receiving 12GHz-satellite broadcasting signals and a low noise block converter (LNB) for converting down the received satellite broadcasting signal into a 1GHz-signal, an IRD 20 for receiving satellite broadcasting from ODU 21 and offering audio and video services to a subscriber's TV or monitor, and an access card 22 required for conditional access (CA) for conditional reception.

Here, IRD 20 performs forward error correction (FEC), decoding, transport demultiplexing, MPEG decoding, NTSC encoding, and audio processing which is a D/A conversion.

Access card 22, whose size is similar to that of a general credit card, has a built-in IC. With this, the card receives CA-related information through a broadcast bit stream and telephone line, that is, a telco MODEM, in order to decide whether a user, subscriber, -selected channel can be viewed or not and to collect its subscription fee.

As shown in Fig. 4, IRD 20 comprises an IR receiver 25 for receiving and processing the subscriber's remote controller input, a telco MODEM 26 which is a general MODEM coupled to the telephone line, a microcomputer 27 made up of an NDC verifier code including software for the CA function and IRD software for IRD driving, a tuner/demodulator/FEC 28 for selecting one channel of

the signal received through ODU 21 and converting the selected channel into a digital bit stream for the purpose of error correction, a transport IC 29 for selecting one program of bit streams output from tuner/demodulator/FEC 28 and multiplexed with various programs, and converting the selected program into a bit stream decodable in the MPEG video decoder and MPEG audio decoder, a card reader interface 23 for data communication between transport IC 29 and access card 22, a system memory 24 coupled to transport IC 29 and for intermediate buffering of data, an MPEG video decoder 30 for expanding a video bit stream compressed in the MPEG format, a frame memory 31 for storing video data expanded in MPEG video decoder 30 in units of frame, an encode/sync/anti-tape/D/A 33 for converting the digital video data expanded in MPEG video decoder 30 into the analog NTSC format and inserting horizontal and vertical sync signals H-Sync and V-Sync and a Macrovision-mode analog copy protection signal in the conversion process, an RF modulator 34 for modulating an NTSC signal of the baseband output from encode/sync/anti-tape/D/A 33 into the RF band, an MPEG audio decoder 32 for expanding the audio bit stream compressed in the MPEG format, and a D/A 35 for converting the expanded digital audio data output from MPEG audio decoder 32 into analog.

Here, in the procedure of conversion into decodable bit stream in the MPEG video and audio decoders from transport IC 29, it is decided whether a program selected through communication with access card 22 can be viewed or not. If the bit stream is scrambled, its descrambling is performed with the access card's permission.

During the process of encode/sync/anti-tape/D/A 33 prior to NTSC video output, the analog copy protection waveform is added to prohibit copying to the analog VCR.

IRD 20 employs a CA system for conditional reception so that a subscriber views programs provided through a broadcasting medium such as satellite broadcasting.

In IRD 20, the NDC verifier code, which is software, and access card 22, which is a smart card for CA, are used to support CA function. A descrambler 36 is contained in transport IC 29.

The detailed block diagram of CA unit 37 and transport IC 29 for operating the CA function in a manner generally used in digital broadcasting is shown in Fig. 5.

More specifically, CA unit 37, included in smart card 22, is made up of smart card 38 for CA and microcomputer 39 operated with CA software.

The CA function is performed when the following two kinds of data are transmitted from a broadcasting station to the IRD. In other words, there are two types of data such as entitlement control message (ECM) or control word packet (CWP), and entitlement management message (EMM) or conditional access packet (CAP).

The EMM is accessed, through the telephone line or satellite broadcasting, to the smart card of the respective IRD at the data rate of 200kbps. The broadcasting station can access all of subscribers' smart cards in a manner that the EMM is transmitted along with ID or address. The EMM has information required to make a control word (CW) for descrambling from the ECM information. The ECM, information in which the control word is encrypted, is transmitted at a speed over 10 per second.

For satellite broadcasting, there are Europe's DVB, Korea's DBS, US' echoster, and the like, aside from DirecTV. Their CA function commonly uses the ECM and EMM information, though different means is provided for the respective broadcastings.

The conventional Macrovision's IPPS is a system having a good performance with respect to the copy protection of analog NTSC video signal. This is an appropriate copyright protection means when a program supplied through a digital medium is converted into analog audio/video signal and recorded or copied through an analog VCR.

However, the IPPS cannot guarantee a satisfactory protection if digital data is recorded or copied using a digital recording medium such as digital VCR. This is because the IPPS uses a method of operating the header's flag bits, without employing, to digital data, encoding methods such as scrambling and encryption. By doing so, hacking is easy to perform only by modulating the flag bits, resulting in very low security.

Summary of the Invention

It would therefore be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which intellectual properties supplied via digital media and protected by copyright are prohibited from being illegally recorded or copied using a digital recording medium such as digital VCR by a user.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which data recorded on a cassette tape is always scrambled to make its hacking difficult and protect its copyright.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which copyright is protected appropriately for respective media which are divided into broadcasting media and pre-recorded media.

It would also be desirable to provide an illegal view/copy protection method and apparatus for a digital broadcasting system in which intellectual properties supplied from a program provider are reproduced to be viewed on screen, copying of the intellectual properties copied and the number of copy are controlled arbitrarily, and fee for recording and copying is collected for the purpose of copyright protection.

According to a first aspect of the present invention, there is provided an illegal view/copy protection method for a digital broadcasting system comprising: an audio/video signal transmission step for multiplexing and transmitting audio/video bit stream scrambled in control words and information where the control words and CPTC information for illegal view/copy protection are encrypted; and an audio/video reception step for decrypting the transmitted bit stream to analyze the CPTC information and control words, deciding whether recording is allowed or not to be recorded on cassette tape, and using the control words, performing descrambling and decoding to output audio/video signals to a monitor.

According to a second aspect of the present invention, there is provided an illegal view/copy protection apparatus for a digital broadcasting system comprising: a program producing portion for multiplexing information encrypted both with the control word for scrambling and the CPTC information for prohibiting illegal view/copy, and the audio/video bit stream scrambled in control words, to thereby make a program; a distribution medium portion for distributing programs made in the program producing portion through a transmission medium; and a program receiving portion for detecting and analyzing the CPTC information from the bit stream transmitted from the distribution medium portion and the bit stream reproduced from cassette tape, and descrambling and decoding the bit stream transmitted from the distribution medium portion.

Brief Description of the Attached Drawings

Figs. 1 and 2 illustrate the operation state of a conventional IPPS;

Fig. 3 is a block diagram of a conventional IPPS;

Fig. 4 is a block diagram of an IRD system;

Fig. 5 shows a configuration of general hardware performing CA function;

Figs. 6A and 6B show formats of CPTC information of an embodiment of the present invention;

Fig. 7 shows a state of generation copy indicating the number of tape recopiable;

Figs. 8A-8D show the recording positions of the CPTC information of an embodiment of the present invention;

Fig. 9 is a flowchart of showing the transmission step of an illegal view/copy protection method embodying the present invention;

Fig. 10 is a flowchart of showing the reception step of an illegal view/copy protection method embodying the present invention;

Fig. 11 is a flowchart of the CPTC information analyzing step of Fig. 10;

Fig. 12 is a flowchart of showing the reproduction/rerecording step of an illegal view/copy protection method embodying the present invention;

Fig. 13 shows the format of an EMM lookup table;

Fig. 14 shows the format of a tape state signal;

Fig. 15 is a flowchart of showing the EMM processing step;

Fig. 16 is a block diagram of the whole configuration of an illegal view/copy protection apparatus embodying the present invention;

Fig. 17 is a block diagram of one embodiment of the program receiving portion of Fig. 16;

Fig. 18 is a block diagram of another embodiment of the program receiving portion of Fig. 16;

Fig. 19 is a block diagram of still another embodiment of the program receiving portion of Fig. 16;

Fig. 20 is a block diagram of yet another embodiment of the program receiving portion of Fig. 16;

Fig. 21 is a block diagram of the IRD shown in Figs. 17, 19 and 20;

Fig. 22 is a block diagram of the IRD and DVCR of Fig. 18;

Fig. 23 illustrates the flow of signals of Fig. 21;

Fig. 24 is a block diagram of one embodiment of the smart card of Fig. 17;

Fig. 25 is a block diagram of another embodiment of the smart card of Fig. 17; and

Fig. 26 is a block diagram of the DVCR of Fig. 17.

Detailed Description of the Invention

An illegal view/copy protection method for a digital broadcasting system embodying the present invention is performed by audio/video signal transmission and audio/video reception steps.

In the audio/video signal transmission step, audio/video bit stream scrambled in control words and information where the control words and CPTC information for illegal view/copy protection are encrypted are multiplexed and transmitted.

In the audio/video reception step, the bit stream transmitted in the audio/video signal transmission step is decrypted to analyze the CPTC information and control words. By doing so, it is decided whether recording is allowed or not. This result is recorded on cassette tape. Using the control words, descrambling and decoding are performed, and then audio/video signals are output to a monitor. Here, the CPTC information separately manages the ECM, EMM and control words, and contains CA information, to thereby control illegal view/copy protection. The CPTC information will be described with reference to Figs. 6A and 6B.

The CPTC information is formatted in a generational copy control field for limiting the number of copy available in order to control the depth of generational copy, and a reproducibility control field for limiting the reproduction of a copied program in order to control the number of copyiable tapes. As shown in Fig. 6A, formatting is performed containing a descrambling information field where part of the control words for descrambling are recorded, or containing a CA field where CA information for conditional access is recorded, as shown in Fig. 6B.

The CPTC information may be encrypted separately to be multiplexed with scrambled digital data, or contained in the ECM information for CA for encryption and multiplexing. Here, the generational copy control field is made up of a permissible generational field for limiting the number of copy permissible and a present generational field for indicating the present generation of a program copied. If the present generation stored in the present generational field is greater than or equal to the permissible generation stored in the permissible generational field, recording or copying is impossible.

A reproduction control field is made up of a reproducible number field for limiting the number of reproducing a copied program, and a maximum reproducible time field for limiting time to reproduce the copied program.

Here, the reproducible number stored in the reproducible number field implements a conditional-number reproducibility function according to the current reproduction number of cassette tape. The maximum reproducible time stored in the maximum reproducible time field implements the conditional-time reproducibility function of copied cassette tape according to the current time information of digital hardware.

The CPTC information may allow the copied cassette tape to be always reproducible, make it never reproducible, allow it to be reproducible as many as a limited number, or make the copied cassette tape reproducible for a limited time after recording or copying.

Using the permissible generational field and present generational field of the generational copy control field, the reproducible number field of the reproduction control field, and data of the maximum reproducible time field, the depth of generation copy, recopying of copied cassette tape, and reproduction time and number are controlled. This process controls the number of copiable cassette tape copied, and reproduction time and number.

In other words, as shown in Fig. 7, information stored in the permissible generational field and present generational field is used to allow first and second generation copy to be perform. Information stored in the reproducible number field and maximum reproducible time field is used to allow reproduction as many as a limited number or for a limited time.

In order to prohibit illegal recording or copy of a program protected by copyright law, collect fee for recording or copy, or arbitrarily control the number of reproducible copied tape to be made from a program supplied by a provider, the depth of generation copy and reproduction of copy tape are controlled to decide how long the first generation recording and copy and second generation copy are made possible.

For this purpose, the copy tape made to be always reproducible, it is made never to be reproducible, it is made to be reproducible as many as a limited number, or it is made to be reproducible for a limited time after recording or copy.

The data recorded on cassette tape contains scrambled audio/video bit stream and CPTC information. The CPTC information is recorded on a recording medium, that is, a rental tape, to prohibit illegal view/copy.

In other words, as shown in Fig. 8A, the CPTC information is overwritten on the scrambled audio/video bit stream for the error effect and recorded on cassette tape. Otherwise, as shown in Fig. 8B, the CPTC information is recorded on a portion of the audio track of cassette tape, on the control track of cassette tape as shown in Fig. 8C, or on the video track of cassette tape as shown in Fig. 8D.

In other words, as shown in Fig. 8A, the CPTC information is overwritten in a predetermined position in the form of error after parities for error correction, that is, inner and outer parities, are added to the scrambled digital data. This method reduces error correction capability but requires no additional tape area for recording the CPTC information. Further, during interleaving and decoding of ECC, the CPTC information is recognized as an error and removed, obtaining the scrambled digital data. Here, the CPTC information is detected separately.

In case that the CPTC information is recorded in part of audio track or control track, as shown in Figs. 8B and 8C, the audio head or control head must be additionally used as the means for detecting the CPTC so that audio track and control track are additionally accessed to detect the CPTC information.

The audio/video signal transmission step using the CPTC information will be explained with reference to Fig. 9.

One embodiment of the audio/video signal transmission step is to transmit an audio/video signal not containing the CA information for conditional access. This, having only the copy protection function, is used in case that a program which can be provided to all viewers is transmitted.

As shown in Fig. 9, the first embodiment of the audio/video signal transmission step comprises the steps of: encoding (100) the audio/video bit stream; generating (105) a control word for scrambling; scrambling (104) for the encoded audio/video bit stream using the generated control word; generating (102) CPTC information for illegal view/copy protection; encrypting (103) for encrypting the control word and CPTC information; and multiplexing and transmitting (106) the scrambled audio/video bit stream and encrypted CPTC information.

In other words, in step 100, the audio/video bit stream is encoded. In step 105, the control word for scrambling is generated. In step 104, the encoded audio/video bit stream is scrambled using the generated control word. In step 102, the CPTC information for illegal view/copy protection is generated. In step 103, the CPTC information and CA information are encrypted using the generated control word. The scrambled audio/video bit stream, encrypted CPTC information and CA

information are multiplexed and transmitted through a transmission medium in step 106. The audio/video signal transmitted through the first embodiment of the audio/video signal transmission step is received through one embodiment of an audio/video reception step.

Referring to Fig. 10, the first embodiment of the audio/video reception step comprises the steps of filtering (110) the transmitted bit stream and decrypting (111) the CPTC information; analyzing (113 and 114) the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream in the control word and outputting an audio/video signal.

In other words, the bit stream transmitted in the first embodiment of the audio/video signal transmission step is filtered and the CPTC information is decrypted in steps 110 and 111. The CPTC information is analyzed to generate the control word and the signal for controlling the protection of copyright, and the CPTC information is updated in steps 113 and 114. Whether to allow recording is determined by the generated signal for controlling the protection of copyright so that the scrambled and transmitted bit stream is recorded on cassette tape in step 115. Then, the transmitted bit stream is descrambled and decoded in control words and output as an audio/video signal in steps 116 and 117. Here, all of the control word is contained in the CPTC information.

Referring to Fig. 11, the CPTC information analyzing step comprises the steps of detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

The copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133, instead of increasing the present generation by '1.'

In order to control generation copy, the permissible generation of the permissible generational field and the present generation of the present generational field are compared in step 130. If the permissible generation is below the present generation, the output disable signal is generated to make copying impossible and the control word is destroyed in step 131. If the permissible generation is not below the present generation, the present generation is increased by '1' and thus recorded on cassette tape in step 132. This enables generation copy. Here, it can be possible that generation copy is limited by updating the CPTC information, instead of increasing the present generation by '1.'

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

In order to control reproduction, the reproducible number of the reproducible number field and the reproduction number of tape are compared in step 134. If the reproducible number is not below the reproduction number of tape, the maximum reproducible time of the maximum reproducible time field and the reproduction time of tape are compared and it is decided whether the maximum reproducible time is below the reproduction time of tape in step 135. In other words, though reproducible, whether it is limited by the reproducible time must be checked. If the maximum reproducible time is not below the reproduction time of tape, the enable erase signal is turned off in step 136 to thereby make the copied program reproducible. If the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, the enable erase signal is turned on to prohibit the reproduction of the copied program. By doing so, part of or the whole program recorded on cassette tape is erased to make copy and reproduction impossible in step 137.

Here, the current time is transmitted to the user by

a provider along with a program. In this case, the copyright protection system implements limited time reproduction using transmitted time information. In this method, the program provider manages the whole users' time so that time modulation by a user cannot occur. Therefore, this is very secure.

The bit stream transmitted in the first embodiment of the audio/video signal transmission step contains ECM and EMM. Part of the control word may be contained in the CPTC information. Its remainder may be contained in the ECM or EMM. The whole control word is contained in the ECM or EMM.

The audio/video signal containing the control word and transmitted according to the audio/video signal transmission step is received according to another embodiment of the audio/video reception step.

Referring to Fig. 10, the second embodiment of the audio/video reception step comprises the steps of filtering (110) the transmitted bit stream and decrypting (111) the CPTC information and control word; filtering (118) the control word; analyzing (113 and 114) the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream in control words and outputting an audio/video signal.

In other words, the bit stream transmitted in the audio/video signal transmission step is filtered and the CPTC information and control word are decrypted in steps 110 and 111. The control word is filtered in step 118. The decrypted CPTC information is analyzed to generate the control word and the signal for controlling the protection of copyright, and the CPTC information is updated in steps 113 and 114. Whether to allow recording is determined by the generated signal for controlling the protection of copyright so that the scrambled and transmitted bit stream is recorded on cassette tape in step 115. Then, the transmitted bit stream is descrambled and decoded in control words and output as an audio/video signal in steps 116 and 117.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating the control words; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

The copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133, instead of increasing the present generation by '1.'

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

Another embodiment of the audio/video signal transmission step is to transmit an audio/video signal containing the CA information for conditional access. This, having the illegal reception and copy protection functions, is used in case that a program which can be provided to limited viewers is transmitted.

As shown in Fig. 9, the second embodiment of the audio/video signal transmission step comprises the steps of: encoding (100) the audio/video bit stream; generating (105) a control word for scrambling; scrambling (104) for the encoded audio/video bit stream using the generated control word; generating (102) CPTC information for illegal view/copy protection; generating (101) CA information for conditional reception; encrypting (103) for encrypting the CPTC information and CA information; and multiplexing and transmitting (106) the scrambled audio/video bit stream and encrypted CPTC information and CA information.

In other words, in step 100, the audio/video bit stream is encoded. In step 105, the control word for scrambling is generated. In step 104, the encoded au-

dio/video bit stream is scrambled using the generated control word. In step 102, the CPTC information for illegal view/copy protection is generated. In step 101, CA information for conditional reception is generated. In step 103, the CPTC information and CA information are encrypted using the generated control word. The scrambled audio/video bit stream, encrypted CPTC information and CA information are multiplexed and transmitted through a transmission medium in step 106. The audio/video signal transmitted through the second embodiment of the audio/video signal transmission step is received through the second embodiment of the audio/video reception step.

Referring to Fig. 10, the second embodiment of the audio/video reception step comprises the steps of: filtering (110) the transmitted bit stream and decrypting (111) the CPTC information; analyzing (112, 113 and 114) the CPTC information and CA information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream and outputting an audio/video signal.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating a control word; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and 137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

In the same manner as the first embodiment of the audio/video reception step, the copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape. If the permissible generation is not below the present generation, the CPTC information is updated in step 133.

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

The bit stream transmitted in the second embodiment of the audio/video signal transmission step contains ECM and EMM. Part of the control word may be contained in the CPTC information. Its remainder may be contained in the ECM or EMM. The whole control word is contained in the ECM or EMM.

The audio/video signal containing the control word and transmitted according to the audio/video signal transmission step is received according to another embodiment of the audio/video reception step. The audio/video signal transmitted in the audio/video signal transmission step containing the control word is received according to still another embodiment of the audio/video reception step.

Referring to Fig. 10, the third embodiment of the audio/video reception step comprises the steps of: filtering (110) the transmitted bit stream and decrypting (111) the CPTC information and CA information; analyzing (112, 113, 114 and 118) the CPTC information and CA information and filtering the control word to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information; deciding (115) whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and descrambling and decoding (116 and 117) the transmitted bit stream and outputting an audio/video signal.

Referring to Fig. 11, in the same manner as the first embodiment of the audio/video reception step, the CPTC information analyzing step comprises the steps of: generating the control words; detecting (130, 131, 132 and 133) the permissible generation of the permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and detecting (134, 135, 136 and

137) the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

The copying number limiting step comprises the steps of: comparing (130) the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation; if the permissible generation is below the present generation, generating (131) an output disable signal to make copying impossible and destroying the control word; and if the permissible generation is not below the present generation, increasing (132) the present invention by '1' and recording the result on cassette tape, and if the permissible generation is not below the present generation, updating the CPTC information in step 133.

The reproduction limiting step comprises the steps of: comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding (134) whether the reproducible number is below the reproduction number of tape; if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding (135) whether the maximum reproducible time is below the reproduction time of tape; if the maximum reproducible time is not below reproduction time of tape, turning off (136) an enable erase signal to thereby enable the copied program to be reproduced; if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on (137) the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

The illegal view/copy protection method for digital broadcasting system embodying the present invention, after the audio/video signal transmission step and audio/video reception step, further comprises a reproduction and rerecording step of: decrypting the bit stream recorded and reproduced on cassette tape, analyzing the CPTC information, deciding whether to allow rerecording, recording the result on cassette tape, filtering the control word, and performing descrambling and decoding to output an audio/video signal.

Referring to Fig. 12, the audio/video reproduction and rerecording step comprises the steps of: filtering (120) the bit stream recorded and reproduced on video tape, and decrypting (121) the CPTC information; analyzing (122 and 123) the CPTC information to generate control words and a signal for controlling the protection of copyright and update the CPTC information; deciding (124) whether to allow recording according to the signal of controlling the protection of copyright, and recording the scrambled and transmitted bit stream on cassette tape; descrambling and decoding (125 and 126) the transmitted bit stream in control words to output an audio/video signal; and deciding whether to allow post-reproduction according to the signal for controlling the protection of copyright to thereby erase part of or the whole data recorded on cassette tape.

Here, EMM may contain information required for decoding information in order to perform the illegal view/copy protection method of a broadcasting system. In this case, a step of storing and processing the EMM is added in the audio/video reproduction and rerecording step.

In the EMM storing and processing step, in case that the EMM is updated by a broadcasting station for the purpose of copyright protection, the EMM having information required to decode the CPTC information is stored in order to continuously reproduce programs of copied cassette tape.

Here, an ID number indicative of updating the EMM is recorded on cassette tape. The EMM is stored to which the updating state and the ID number of cassette tape are mapped.

The EMM storing and processing step comprises the steps of: storing all EMM to be updated and corresponding ID information; selecting the latest EMM in recording cassette tape; recording a corresponding ID number; and selecting an EMM corresponding to the ID number recorded on cassette tape in reproducing the cassette tape.

As shown in Fig. 13, all EMMs (EMM1, EMM2, EMM3,...) to be updated on the EMM lookup table and corresponding ID information (ID1, ID2, ID3,...) are mapped and stored.

Referring to Figs. 14 and 15, in recording a program on cassette tape, that is, when recording is indicated in the recording/reproduction mode, an ID number corresponding to the latest, the final, EMM, is recorded. Thereafter, in reproducing the cassette tape, that is, when reproduction is indicated in the recording/reproduction mode, an EMM corresponding to the ID number recorded on cassette tape is selected from the EMM lookup table so that the recorded program is reproduced according to the reproducible number of the reproducible number field and the reproduction number recorded on the video tape.

Referring to Fig. 16, an illegal view/copy protection apparatus of digital broadcasting system embodying the present invention comprises a program producing portion 200, distribution medium portion 201, and program receiving portion 202.

Program producing portion 200 offers programs, in which information encrypted both with the control word for scrambling and the CPTC information for prohibiting illegal view/copy, and the audio/video bit stream scrambled in control words are multiplexed to make a program.

Distribution medium portion 201 distributes pro-

grams made in program producing portion 200 through a transmission medium.

Program receiving portion 202 detects and analyzes the CPTC information from the bit stream transmitted from distribution medium portion 201 and the bit stream reproduced from cassette tape, and descrambles and decodes the bit stream transmitted from distribution medium portion 201. The descrambled and decoded bit stream is displayed or recorded on cassette tape.

Program producing portion 200 comprises a control word generator 203 for generating a control word for scrambling, a CPTC generator 204 for generating the CPTC information for prohibiting illegal view/copy, a scrambling portion 206 for scrambling the audio/video bit stream using the control word output from control word generator 203, an encrypting portion 205 for encrypting the control word output from control word generator 203 and the CPTC information output from CPTC generator 204, and an adder 207 for multiplexing the signals output from scrambling portion 206 and encrypting portion 205 and transmitting them to distribution medium portion 201.

Distribution medium portion 201 comprises a broadcasting medium 208 for distributing the program made by program producing portion 200 through cable, satellite or terrestrial broadcasting, and a recording medium 209 for distributing the program made by program producing portion 200 through cassette tape.

Program receiving portion 202 comprises a decrypting portion 210 for decrypting the bit stream transmitted from broadcasting medium 208, a CPTC detecting/analyzing portion 211 for detecting and analyzing the CPTC information from the bit stream output from decrypting portion 210 and recording medium 209, and outputting signals for controlling the control word and illegal view/copy, a descrambling portion 212 for descrambling the bit stream transmitted from broadcasting medium 208 and recording medium 209 and the bit stream reproduced from cassette tape, a decoding portion 213 for decoding and displaying the signal output from descrambling portion 212, and a recording/reproducing portion 214 for recording the bit stream transmitted from broadcasting medium 208 and recording medium 209 according to the signal output from CPTC detecting/analyzing portion 211, and reproducing cassette tape, to thereby output the result to descrambling portion 212 and CPTC detecting/analyzing portion 211.

The operation of an illegal view/copy protection apparatus for a digital broadcasting system embodying the present invention will be described below.

Control word generator 203 generates a control word for scrambling, and CPTC generator 204 generates the CPTC information for prohibiting illegal view/copy. Scrambling portion 206 scrambles the audio/video bit stream using the generated control word. Encrypting portion 205 encrypts the CPTC information output from CPTC generator 204 using the generated control word. The audio/video bit stream scrambled in scrambling portion 206 is multiplexed with the encrypted CPTC information in adder 207. The multiplexed result is transmitted to a reception port through distribution medium portion 201.

The signal output from adder 207 is transmitted to program receiving portion 202 through broadcasting medium 208 such as cable, satellite, and terrestrial broadcastings, or through recording medium 209 made of cassette tape such as rental tape.

The bit stream transmitted through broadcasting medium 208 is decrypted in decrypting portion 210. The CPTC information is detected and analyzed in CPTC detecting/analyzing portion 211 so that signals for controlling the control word and illegal view/copy are output. Here, the bit stream transmitted to cassette tape through recording medium 209 is reproduced in recording/reproducing portion 214 and input to descrambling portion 212 and CPTC detecting/analyzing portion 211. The bit stream transmitted from broadcasting medium 208 and the bit stream reproduced from recording medium 209 through recording/reproducing portion 214 are descrambled in descrambling portion 212 according to the control word output from CPTC detecting/analyzing portion 211. The signal output from descrambling portion 212 is decoded in decoding portion 213 and displayed. The bit stream transmitted from broadcasting medium 208 and recording medium 209 is recorded on cassette tape in a recording/reproducing portion 214 according to the signal output from CPTC detecting/analyzing portion 211.

Data received from program receiving portion 202 and recorded on cassette tape is made up of the scrambled audio/video bit stream and CPTC information. The configuration of the program receiving portion having decrypting portion 210, CPTC detecting/analyzing portion 211, descrambling portion 212, decoding portion 213 and recording/reproducing portion 214 will be explained with reference to Figs. 17, 18, 19, and 20.

One embodiment of the program receiving portion of Fig. 17 receives and processes data transmitted via a broadcasting medium. Specifically, this embodiment performs conditional access and copy protection.

Referring to Fig. 17, the first embodiment of the program receiving portion comprises an IRD 222 for receiving, decoding and descrambling the bit stream transmitted from broadcasting medium 208, outputting analog audio/video data to be displayed and outputting scrambled digital audio/video data to be recorded on cassette tape, a smart card 221 for decrypting the bit stream output from IRD 222, detecting/analyzing the CPTC information, and outputting the control word and signals for controlling illegal view/copy to IRD 222 in order to perform conditional access and copy protection, a DVCR 223 for recording the digital audio/video data and CPTC information scrambled and output from IRD 222 on cassette tape, and reproducing the scrambled digital audio/video data and CPTC information recorded on cassette tape to be output to IRD 222, and a lookup table 224 for,

in case that the EMM is updated by a broadcasting station for the purpose of copyright protection, storing EMM having information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction to smart card 221 in order to continuously reproduce the program of copied cassette tape. Here, lookup table 221 is mapped and processed as shown in Figs. 13, 14 and 15.

The operation of the first embodiment of the program receiving portion will be described below.

In case that a bit stream, that is, a program, is received through a broadcasting medium, the received audio/video data is scrambled digital audio/video data.

The received bit stream is decoded in IRD 222 and decrypted in smart card 221. Its CPTC information is detected and analyzed so that a signal for controlling the control word and illegal view/copy is output to IRD 222.

IRD 222 descrambles the decoded bit stream using the bit stream output from smart card 221 and signals for controlling illegal view/copy. The descrambled bit stream is output to display analog audio/video data. IRD 222 outputs the scrambled digital audio/video data and CPTC information to DVCT 223 in order to record them on cassette tape.

The scrambled digital audio/video data and CPTC information output from IRD 222 is recorded on cassette tape in DVCR 223. They are in turn reproduced in DVCR 223 and processed in the same manner that the bit stream transmitted via the broadcasting medium is descrambled and processed in IRD 222 and smart card 221. The processed result is output to be displayed on a monitor, or output to the DVCR and recopied.

Here, reproduction and recopy are made possible by the data stored in the permissible generational field, present generational field, reproducible number field, and maximum reproducible time field contained in the CPTC information.

Updated EMM is mapped and stored in lookup table 224 so that, when the EMM is updated through a broadcasting signal in a broadcasting station in order to protect copyright, the program of cassette tape copied can be continuously reproduced.

Lookup table 224 reads out the EMM containing information required to decode the CPTC information in reproducing the cassette tape. Corresponding CPTC information is output to smart card 221 to enable reproduction.

Another embodiment of the program receiving portion shown in Fig. 18 is to receive and process data transmitted through a recording medium, for instance, rental tape.

The second embodiment of the program receiving portion, as shown in Fig. 18, comprises a DVCR 232 for detecting/analyzing the CPTC information from the bit stream transmitted from the recording medium, outputting a control word and signals for controlling illegal view/copy, and reproducing scrambled digital audio/video data, and an IRD 231 for receiving the control word and signals for controlling illegal view/copy output from DVCR 232, descrambling the scrambled digital audio/video data, and outputting analog audio/video data to be displayed or recorded.

The second embodiment of the program receiving portion is to perform CPTC detection and processing carried out in the smart card of the first embodiment of the program receiving portion shown in Fig. 17. The operation of the second embodiment of the program receiving portion will be described below.

In case that the bit stream is received through the recording medium, the audio/video data reproduced through the DVCR is scrambled digital audio/video data.

The bit stream recorded in DVCR 232 is reproduced. Its CPTC information is detected and analyzed so that the control word and signal for controlling illegal view/copy is output to IRD 231. The bit stream reproduced from DVCR 232 is decoded in IRD 231. The decoded bit stream is descrambled according to the control word and signal for controlling illegal view/copy output from DVCR 232 so that analog audio/video data is output to be displayed.

IRD 231 outputs the scrambled digital audio/video data and CPTC information to DVCR 232 to record them on cassette tape. The scrambled digital audio/video data and CPTC information output from IRD 231 is recorded on cassette tape and recopied in DVCR 223.

Here, reproduction and recopy are made possible by the data stored in the permissible generational field, present generational field, reproducible number field, and maximum reproducible time field contained in the CPTC information.

Referring to Fig. 19, still another embodiment of the program receiving portion is to receive and process data transmitted through a recording medium, performing copy protection (CP).

As shown in Fig. 19, the third embodiment of the program receiving portion comprises a DVCR 243 for reproducing the scrambled digital audio/video data and CPTC information recorded on cassette tape through a recording medium, and outputting them to IRD 242, an IRD 242 for decoding/descrambling the bit stream transmitted from DVCR 243, and outputting analog audio/video data to be displayed, and a smart card 241 for decrypting the bit stream output from IRD 242, detecting/analyzing the CPTC, and outputting the control word and signals for controlling copying to IRD 222 to thereby perform CP. The operation of the third embodiment of the program receiving portion will be explained below.

In case that the bit stream is received via a recording medium, that is, through rental tape, the reproduced audio/video data is scrambled digital audio/video data.

The scrambled digital audio/video data and CPTC information reproduced from DVCR 243 are decoded in IRD 242 and decrypted in smart card 241. The CPTC information is detected and analyzed so that the control word and signal for controlling copying are output to IRD 242.

IRD 242 descrambles the decoded bit stream using the CPTC information output from smart card 241 and signals for controlling copying so that analog audio/video data is output to be displayed.

IRD 242 outputs the scrambled digital audio/video data and CPTC information to DVCR 243 in order to record them on cassette tape. The scrambled digital audio/video data and CPTC information output from IRD 242 are recorded on cassette tape in DVCR 243.

Here, reproduction and recopy are made possible by the data stored in the permissible generational field, present generational field, reproducible number field, and maximum reproducible time field contained in the CPTC information.

Referring to Fig. 20, yet another embodiment of the program receiving portion is to receive and process data transmitted through a recording medium, performing conditional access and CP. This embodiment is made in such a manner that in case of using the same CPTC information as the broadcasting medium, the smart card is commonly used.

As shown in Fig. 20, the fourth embodiment of the program receiving portion comprises a DVCR 253 for reproducing the scrambled digital audio/video data and CPTC information recorded on cassette tape through a recording medium, and outputting them to IRD 252, an IRD 252 for decoding/descrambling the bit stream transmitted from DVCR 253, and outputting analog audio/video data to be displayed, and a smart card 251 for decrypting the bit stream output from IRD 252, detecting/analyzing the CPTC, and outputting the control word and signals for controlling copying to IRD 252 to thereby perform CA and CP. The operation of the third embodiment of the program receiving portion will be explained below.

In case that the bit stream is received via a recording medium, that is, through rental tape and the DVCR, the reproduced audio/video data is scrambled digital audio/video data.

The scrambled digital audio/video data and CPTC information reproduced from DVCR 253 are decoded in IRD 252 and decrypted in smart card 251. The CPTC information is detected and analyzed so that the control word and signal for controlling copying are output back to IRD 252.

IRD 252 descrambles the decoded bit stream using the CPTC information output from smart card 251 and signals for controlling illegal view/copy so that analog audio/video data is output to be displayed.

IRD 252 outputs the scrambled digital audio/video data and CPTC information to DVCR 253 in order to record them on cassette tape. The scrambled digital audio/video data and CPTC information output from IRD 222 are recorded on cassette tape in DVCR 253.

Here, reproduction and recopy are made possible by the data stored in the permissible generational field, present generational field, reproducible number field, and maximum reproducible time field contained in the CPTC information.

IRD 222, 242, or 252 shown in Fig. 17, 19 or 20 is made in the following configuration as shown in Fig. 21.

Referring to Fig. 21, IRD 222, 242 or 252 comprises a recording/digital output controller 262 for decoding the bit stream transmitted from the broadcasting medium and DVCR, outputting to smart card 221, receiving the control word and signals for controlling illegal view/copy output from smart card 221, and controlling the output of the scrambled digital audio/video data for the purpose of recording and displaying; a descrambler 263 for descrambling the scrambled digital audio/video data output from recording/digital output controller 262 according to the control word output from recording/digital output controller 262, and a display processing portion 264 for processing and outputting the digital audio/video data output from descrambler 263 to be displayed. Here, DVCR 265 performs reproduction mainly. DVCR 223 of the program receiving portion of Fig. 18 combines recording therewith. The operation of IRD 266 will be described below.

The signal output to smart card 261 from recording/digital output controller 262 of IRD 266 is ECM, EMM and CPTC information. The signals output from smart card 261 to IRD 266 are the control word used to descramble and display the bit stream, and a signal for controlling copy protection.

Recording/digital output controller 262 communicates with the smart card, performs recording according to the signals of copy protection, outputs them to the digital output port in order to record them in another set, and outputs the control word and bit stream to descrambler 263.

When output to the recording/digital output port, updated ECM, EMM and CPTC information are output in addition to the scrambled data from recording/digital output controller 262 so that a copy different from the original script, that is, the broadcast or rental tape.

The ECM, EMM and CPTC are transmitted in various combinations. For the first combination, the ECM, EMM and CPTC are independently combined. The second combination is that the CPTC is included in the ECM and the EMM is independently combined. The third is that the CPTC is included in the EMM and the ECM is independently combined.

IRD 231 and DVCR 232 of Fig. 18 use the smart card, and additionally requires a CPTC detection and processing portion in the DVCR, which will be shown in Fig. 22.

DVCR 232 comprises a CPTC detecting/processing portion 276 for detecting/analyzing the CPTC information from the bit stream transmitted from recording medium 209, and outputting the control word and signals for illegal view/copy, and a reproducing portion 277 for reproducing the bit stream transmitted from recording medium 209 and outputting it to the IRD.

IRD 231 comprises a digital output controller 272 for receiving the control word and signals for controlling

illegal view/copy output from CPTC detecting/processing portion 276, and controlling the output of the scrambled digital audio/video data output from reproducing portion 277 in order to display them, a descrambler 273 for descrambling the scrambled digital audio/video data output from digital output controller 262 according to the control word output from digital output controller 262, and a display processing portion 274 for processing and outputting the digital audio/video data output from descrambler 273 in order to display them. The operation of IRD 276 and DVCR 275 will be described below.

CPTC detecting/processing portion 276 operates separately when reproducing portion 277 reproduces the scrambled data so that the CPTC information is detected from the cassette tape.

IRD 276 receives the scrambled data, CPTC information and control word from CPTC detecting/processing portion 276 and reproducing portion 277 from DVCR 275. Therefore, for normal descrambling, the scrambled data and control word are supplied to scrambler 273 from digital output controller 272. To the digital output port, only the scrambled data is output. For this reason, in case that the reproduced data is scrambled, copying is made impossible, and vise versa.

Commonly, in order to control tape copying, the depth of generation copy and the reproduction of tape to be copied are used together. As shown in Fig. 7, this yields the effect of controlling the number of copiable tape.

However, in order to allow copying tape to be reproducible as many as a predetermined number or for a predetermined time, it is necessary to perform communication between the smart card and DVCR.

Referring to Fig. 23, tape state information such as the reproduction number of the current tape is transmitted to smart card 261 from DVCR 265. In order to erase the tape, an enable erase signal is transmitted to DVCR 265 from smart card 261, and the erase head of the DVCR operates.

For tape erasing methods, the whole area of tape is erased by the full-width erase head, or only the control track is erased using the control head. In case that the CPTC is contained in the EMM, signals are input and output between the DVCR and smart card.

As the signals input to IRD 266, there are a broadcasting signal transmitted from a broadcasting medium and a signal reproduced from DVCR 265. The broadcasting signal input to IRD 266 is the scrambled digital data and a control signal having the EMM, ECM and CPTC information. The EMM and ECM are required for CA, the CPTC for copyright protection.

The scrambled digital data is input to descrambler 263. The control signal is input to smart card 261 for performing CA and CP. Using the control signal, smart card 261 restores control word CW and outputs it to descrambler 263. Descrambler 263 descrambles it using the control word.

The ECM output from smart card 261 is output to DVCR 265 or to an external port. This ECM is updated from the ECM input for copyright protection. The output disable signal output from smart card 261 is a signal to instruct IRD 266 to prohibit recording or copying. This signal is input to recording/digital output controller 262. The tape state signal is output to smart card 261 from DVCR 265 in order to inform the state of tape.

The signal output to DVCR 265 from smart card 261 for the purpose of a predetermined-number reproduction or predetermined-time reproduction is an erase enable signal. The signal for allowing recorded and copied tape to be reproducible even though the EMM information of the smart card is changed is an ID signal.

The ID signal is mapped and stored with corresponding EMM in the lookup table of smart card 261. If necessary, the EMM corresponding to the ID signal is output.

As shown in Fig. 24, the smart card comprises an ECM filter 301 for filtering the ECM from the bit stream output from the IRD, a CPTC/tape state signal filter 302 for filtering the CPTC information and the tape state signal indicative of the state of tape from the bit stream output from the IRD, an EMM filter 303 for filtering the EMM from the bit stream output from the IRD, a lookup table 304 for, in case that the EMM is updated for copyright protection by a broadcasting station, storing the previous EMM containing information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction in order to continuously reproduce the program of cassette tape copied, an EMM processing portion 307 for processing the EMM using the EMM output from EMM filter 303 and lookup table 304 and the tape state signal output from CPTC/tape state signal filter 302, a CPTC processing portion 306 for processing the CPTC information using the signals output from CPTC/tape state signal filter 302 and EMM processing portion 307, and a CA processing portion 305 for outputting control word CW using the signals output from ECM filter 301 and EMM processing portion 307.

In case that the CPTC information is contained in the EMM, as shown in Fig. 25, smart card 221 comprises an ECM filter 311 for filtering the ECM from the bit stream output from the IRD, an EMM filter 312 for filtering the EMM containing the EMM from the bit stream output from the IRD, a tape state signal filter 313 for filtering the tape state signal output from the IRD, a lookup table 314 for, in case that the EMM is updated for copyright protection by a broadcasting station, storing the previous EMM containing information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction in order to continuously reproduce the program of cassette tape copied, an EMM processing portion 317 for processing the EMM using the EMM output from EMM filter 312 and lookup table 314 and the tape state signal output from tape state signal filter 313, a CPTC processing portion 316 for processing the CPTC information using the signals

output from EMM filter 312 and tape state signal filter 313, to thereby output ECM, enable erase signal and ID signal, and a CA processing portion 315 for outputting control word CW using the signals output from ECM filter 311 and EMM processing portion 317.

ECM filter 301 or 311, CPTC/tape state signal filter 302, EMM filter 303 or 312, and tape state signal filter 313 extract ECM, CPTC, tape state signal and EMM, respectively. CA processing portion 305 or 315 generates a control word and performs CA. EMM processing portion 307 or 317 outputs the EMM information to CA processing portion 305 or 315 and CPTC processing portion 306 or 316, and additionally stores the received EMM to the lookup table.

In case that the scrambled digital data and encoded CPTC information are recorded on tape and that the EMM information required to decode the CPTC information is changed, the reproduction of tape is made impossible. According to this fact, the previous EMM is stored in a memory such as the EEPROM of the smart card as shown in Figs. 13 and 14, which is the same as described before.

Specifically, the lookup table is divided into two fields and stores ID information and EMM information, as shown in Fig. 13. In recording and copying, the ID information is recorded on tape, as shown in Fig. 14 in order to select corresponding EMM from the ID information recorded in the reproduction of tape.

In other words, referring to Fig. 14, EMM processing portion 307 receives a recording/playback signal indicating that the current DVCR mode is recording or playback, ID, and tape state signal having information of reproduction number of tape, selects a proper EMM from the lookup table, outputs it to CPTC processing portion 306 or 316 and CA processing portion 305 or 315, and transmits the ID information for the purpose of recording and copying to record it on tape.

Referring to Fig. 11, CPTC processing portion 306 or 316 performs copyright protection for recording or copying. The CPTC information or ECM containing the CPTC information is input to output the output disable signal, enable erase signal, and the CPTC or ECM containing the CPTC.

In order to control generation copy, CPTC processing portion 306 or 316, in case that the permissible generation of the permissible generational field is greater than the present generation recorded on tape, the present generational field is increased by 1 and encrypted again. If not, the output disable signal is generated to prohibit recording and copying.

In order to control reproduction, in case that the reproducible number of tape is greater than the reproducible number of the reproducible number field or the maximum reproducible time of the maximum reproducible time field is greater than the current time, CPTC processing portion 306 or 316 generates enable erase signal to operate the erase head of the DVCR.

In case that time delay produced when the CPTC or the ECM containing the CPTC is encrypted again becomes a problem to solve, CPTC processing portion 306 or 316 transmits the current generation signal to the DVCR and records it on tape, not modifying the CPTC or the ECM containing the CPTC.

The illegal view/copy protection apparatus for a digital broadcasting system embodying the present invention has means for recording and reproducing the reproduction number information of tape in the DVCR in order to implement the predetermined-number reproducibility of recorded or copied tape. Here, the reproduction number information of tape is updated and recorded again during tape reproduction.

As shown in Fig. 26, the DVCR comprises a deck mechanism 406, a recording/reproducing portion 405 for recording digital data on cassette tape according to the deck mechanism and reproducing the digital data recorded on cassette tape, a reproduction number detecting/updating portion 401 for detecting/updating the reproduction number from the digital data reproduced from recording/reproducing portion 405, and outputting it to the IRD in order to rerecord it in recording/reproducing portion 405, a digital data processing portion 402 for processing the digital data reproduced from recording/reproducing portion 405, outputting it to the IRD, and outputting switching position information for recording and reproducing, a recording/playback switching portion 404 for outputting a switching signal for controlling the reproduction number, the reproduction of digital data and the recording of the updated reproduction number using the switching position information output from digital data processing portion 402, and an error correction encoder/decoder 403 for correcting the error of data output from digital data processing portion 402, and encoding and decoding the data to be output to digital data processing portion 402.

In order to update and rerecord the reproduction number information of tape during playback, the reproduction number information of tape is recorded using an encoding algorithm. Otherwise, the information is recorded as clear data not encoded.

The recording position of the reproduction number information of tape uses part of audio, control and video tracks. For error correction to the reproduction number information of tape, a repetition coding is employed. The operation of the DVCR will be described below.

When reproduced by recording/reproducing portion 405 with the cassette tape loaded on deck mechanism 406, the reproduced digital data is input to reproduction number detecting/updating portion 401 and digital data processing portion 402 so that its reproduction number is detected and the digital data is processed and output.

The reproduction number detected in reproduction number detecting/updating portion 401 is updated, that is, increased by 1, and applied to recording/reproducing portion 405.

Digital data processing portion 402 applies the reproduced digital data output from recording/reproducing

portion 405 to error correction encoder/decoder 403 to perform error correction, encoding and decoding. The result is output to the IRD to be displayed or recorded. At the same time, the switching position information is output to recording/reproducing switching portion 404 in order to output a switching signal.

The switching signal output from recording/reproducing switching portion 404 controls recording/reproducing portion, to thereby record the updated reproduction number output from reproduction number detecting/updating portion 401, that is, the reproduction number added by 1, on tape.

Recording/reproducing switching portion 404 controls the reproduction number, the reproduction of digital data recorded on tape, and the recording of the updated reproduction number.

In another method of implementing the predetermined-number reproducibility of recorded or copied tape, an identifier is given to all tape used for a user to record broadcast programs, and the identifier given to tape and the reproducibility number information of tape corresponding to the identifier are handled together in the smart card.

Here, the smart card has a memory device which can be updated, such as EEPROM. The identifier and corresponding reproducible number information are stored in the memory device. For every reproduction of tape, the reproducible number information is updated and whether to playback is determined.

In conclusion, the described embodiments have the following advantages.

First, by adding CPTC information to data supplied, and by allowing a digital program to be normally viewed only when a CPTC detecting/analyzing means and descrambling/decrypting means are present at the receiving stage, illegal viewing is prohibited.

Second, to enhance copyright protection, data recorded on cassette tape is always scrambled digital data, and its CPTC information is encrypted to be recorded on cassette tape. A code for prohibiting viewable data from being restored from the cassette tape only with the scrambled data and CPTC information, and allowing the data to be viewable is provided in a device excluding the cassette tape. Otherwise, restoring of viewable data is made possible only with the scrambled data and CPTC information, making illegal copy impossible.

Third, using a method of restoring the viewable data only with the scrambled digital data and CPTC, rental tape is made to supply tape. Otherwise, using a method of prohibiting the viewable data from being restored only with the scrambled digital data and CPTC, rental tape is made to supply tape and smart card peculiar to a program provider as one set. Using the smart card for broadcasting medium, the rental tape is made to prohibit the viewable data from being restored only with the scrambled digital data and CPTC. Among the three methods of supplying tape only, one method is selected. Digital hardware for reproducing the data outputs only the scrambled digital data to an external port, making impossible the restoring of viewable data from the output data, without the smart card.

Fourth, the described embodiment prohibits illegal recording and copying of a program protected by copyright law, collects fee for recording or copying, and freely controls the reproducible number of copied tape which can be made from a program supplied by a program supplier, protecting copyright.

Fifth, the described embodiment can be used as a copyright protection system having a high security and multifunction with respect to a program through a broadcasting medium such as satellite and terrestrial broadcastings, or, at the same time, as a copy protection system having a high security to a program through a recording medium such as rental tape.

Sixth, the descibed embodiment is employed to digital hardware such as broadcasting receiver and digital VCR, to thereby perfectly protect a program supplier's copyright and activates digital media because of various software supplied through the digital media.

Claims

1. An illegal view/copy protection method for a digital broadcasting system comprising:

   an audio/video signal transmission step for multiplexing and transmitting audio/video bit stream scrambled in control words and information where the control words and CPTC information for illegal view/copy protection are encrypted; and

   an audio/video reception step for decrypting the transmitted bit stream to analyze the CPTC information and control words, deciding whether recording is allowed or not to be recorded on cassette tape, and using the control words, performing descrambling and decoding to output audio/video signals to a monitor.

2. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1, wherein said CPTC information is formatted in a generational copy control field for limiting the number of copy available, and a reproducibility control field for limiting the reproduction of a copied program.

3. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 2, wherein said CPTC information is formatted further containing a descrambling information field where part of the control words for descrambling are recorded.

4. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 2, wherein said CPTC information is formatted further contain-

ing a CA field where CA information for conditional access is recorded.

5. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 2, wherein said generational copy control field is made up of a permissible generational field for limiting the number of copy permissible and a present generational field for indicating the present generation of a program copied.

6. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 2, wherein said reproduction control field is made up of a reproducible number field for limiting the number of reproducing a copied program, and a maximum reproducible time field for limiting time to reproduce the copied program.

7. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1, wherein the data recorded on cassette tape contains scrambled audio/video bit stream and CPTC information.

8. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 7, wherein said CPTC information is overwritten on the scrambled audio/video bit stream for the error effect and recorded on cassette tape.

9. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 7, wherein said CPTC information is recorded on a portion of any of the audio track of cassette tape, the control track of cassette tape, or the video track of cassette tape.

10. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1, wherein said audio/video signal transmission step comprises the steps of: encoding the audio/video bit stream;

generating a control word for scrambling;
scrambling for the encoded audio/video bit stream using the generated control word;
generating CPTC information for illegal view/copy protection;
encrypting for encrypting the control word and CPTC information; and
multiplexing and transmitting the scrambled audio/video bit stream and encrypted CPTC information.

11. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1, wherein said audio/video signal transmission step comprises the steps of:

encoding the audio/video bit stream;
generating a control word for scrambling;
scrambling for the encoded audio/video bit stream using the generated control word;
generating CPTC information for illegal view/copy protection;
generating conditional access information for conditional reception;
encrypting for encrypting the CPTC information and CA information; and
multiplexing and transmitting the scrambled audio/video bit stream and encrypted CPTC information and conditional access information.

12. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1 or claim 11, wherein said audio/video reception step comprises the steps of:

filtering the transmitted bit stream and decrypting the CPTC information;
analyzing the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information;
deciding whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmitted bit stream on cassette tape; and
descrambling and decoding the transmitted bit stream in the control word and outputting an audio/video signal.

13. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 12, wherein said all of the control word is contained in the CPTC information.

14. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 1, wherein said bit stream transmitted contains ECM and EMM.

15. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 14, wherein said audio/video reception step comprises the steps of:

filtering the transmitted bit stream and decrypting the CPTC information and control word;
filtering the control word;
analyzing the CPTC information to generate a control word and a signal for controlling the protection of copyright and to update the CPTC information;
deciding whether to allow recording according to the signal for controlling the protection of copyright to record the scrambled and transmit-

ted bit stream on cassette tape; and
descrambling and decoding the transmitted bit stream in control words and outputting an audio/video signal.

16. An illegal view/copy protection method for a digital broadcasting system as claimed in any of claims 12, 14 or 15, wherein said CPTC information analyzing step comprises the steps of:

generating a control word;
detecting a permissible generation of a permissible generational field for limiting the available number of copy of a program of the CPTC information and the present generation of the present generational field indicating the present generation of the program copied, to thereby perform copy-impossible and update the CPTC information; and
detecting the reproducible number of the reproducible number field for limiting the number of reproduction of copied programs of the CPTC information, the maximum reproducible time of the maximum reproducible time field for limiting time to reproduce the copied program, and the number and time of reproduction of tape, to thereby process reproduction-impossible.

17. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 12 or claim 16, wherein said copying number limiting step comprises the steps of:

comparing the permissible generation of the permissible generational field and the present generation of the present generational field and deciding whether the permissible generation is below the present generation;
if the permissible generation is below the present generation, generating an output disable signal to make copying impossible and destroying the control word; and
if the permissible generation is not below the present generation, increasing the present invention by '1' and recording the result on cassette tape.

18. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 17, wherein said copying number limiting step further comprises the step of, if the permissible generation is not below the present generation, updating the CPTC information.

19. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 16 or claim 17, wherein said reproduction limiting step comprises the steps of:

comparing the reproducible number of the reproducible number field and the reproduction number of tape and deciding whether the reproducible number is below the reproduction number of tape;
if the reproducible number is not below the reproduction number of tape, comparing the maximum reproducible time and reproduction time of tape, and deciding whether the maximum reproducible time is below the reproduction time of tape;
if the maximum reproducible time is not below reproduction time of tape, turning off an enable erase signal to thereby enable the copied program to be reproduced; and
if the reproducible number is below the reproduction number of tape or the maximum reproducible time is below the reproduction time of tape, turning on the enable erase signal to make the reproduction of the copied program impossible so that part of or the whole program recorded on cassette tape is erased.

20. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 14 or claim 15, wherein part of the control word is contained in the CPTC information.

21. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 20, wherein the remainder of the control word is contained in the ECM.

22. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 20, wherein the remainder of the control word is contained in the EMM.

23. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 14 or claim 15, wherein the whole control word is contained in the ECM.

24. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 14 or claim 15, wherein the whole control word is contained in the EMM.

25. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 14, further comprising a reproduction and rerecording step of: decrypting the bit stream recorded and reproduced on cassette tape, analyzing the CPTC information, deciding whether to allow rerecording, recording the result on cassette tape, filtering the control word, and performing descrambling and decoding to output an audio/video signal.

26. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 25, wherein said audio/video reproduction and rerecording step comprises the steps of:

filtering the bit stream recorded and reproduced on video tape, and decrypting the CPTC information;
analyzing the CPTC information to generate control words and a signal for controlling the protection of copyright and update the CPTC information;
deciding whether to allow recording according to the signal of controlling the protection of copyright, and recording the scrambled and transmitted bit stream on cassette tape; and
descrambling and decoding the transmitted bit stream in control words to output an audio/video signal.

27. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 26, wherein said audio/video reproduction and rerecording step comprises the step of deciding whether to allow post-reproduction according to the signal for controlling the protection of copyright to thereby erase part of or the whole data recorded on cassette tape.

28. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 25, wherein said EMR contains information required for decoding information

29. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 28, further comprising the step of storing and processing EMM in which, in case that the EMM is updated by a broadcasting station for the purpose of copyright protection, the EMM having information required to decode the CPTC information is stored in order to continuously reproduce programs of copied cassette tape.

30. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 29, wherein an ID number indicative of updating the EMM is recorded on said cassette tape.

31. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 30, wherein the EMM is stored to which the updating state and the ID number of cassette tape are mapped.

32. An illegal view/copy protection method for a digital broadcasting system as claimed in claim 31, wherein said EMM storing and processing step comprises the steps of:

storing all EMM to be updated and corresponding ID information;
selecting the latest EMM in recording cassette tape;
recording a corresponding ID number; and
selecting an EMM corresponding to the ID number recorded on cassette tape in reproducing the cassette tape.

33. An illegal view/copy protection apparatus for a digital broadcasting system comprising:

a program producing portion for multiplexing information encrypted both with the control word for scrambling and the CPTC information for prohibiting illegal view/copy, and the audio/video bit stream scrambled in control words, to thereby make a program;
a distribution medium portion for distributing programs made in said program producing portion through a transmission medium; and
a program receiving portion for detecting and analyzing the CPTC information from the bit stream transmitted from said distribution medium portion and the bit stream reproduced from cassette tape, and descrambling and decoding the bit stream transmitted from said distribution medium portion.

34. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein said program producing portion comprising:

a control word generator for generating a control word for scrambling;
a CPTC generator for generating the CPTC information for prohibiting illegal view/copy;
a scrambling portion for scrambling the audio/video bit stream using the control word output from said control word generator;
an encrypting portion for encrypting the control word output from said control word generator and the CPTC information output from said CPTC generator; and
an adder for multiplexing the signals output from said scrambling portion and encrypting portion and transmitting them to said distribution medium portion.

35. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein said distribution medium portion comprises:

a broadcasting medium for distributing the program made by said program producing portion through cable, satellite or terrestrial broadcast-

ing; and
a recording medium for distributing the program made by said program producing portion through cassette tape.

36. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 35, wherein said program receiving portion comprises:

a decrypting portion for decrypting the bit stream transmitted from said broadcasting medium;
a CPTC detecting/analyzing portion for detecting and analyzing the CPTC information from the bit stream output from said decrypting portion and recording medium, and outputting signals for controlling the control word and illegal view/copy;
a descrambling portion for descrambling the bit stream transmitted from said broadcasting medium and recording medium and the bit stream reproduced from cassette tape;
a decoding portion for decoding and displaying the signal output from said descrambling portio; and
a recording/reproducing portion for recording the bit stream transmitted from said broadcasting medium and recording medium according to the signal output from said CPTC detecting/analyzing portion, and reproducing cassette tape, to thereby output the result to said descrambling portion and CPTC detecting/analyzing portion.

37. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein said CPTC information is formatted in a generational copy control field for limiting the number of copy available, and a reproducibility control field for limiting the reproduction of a copied program.

38. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 37, wherein said CPTC information is formatted further containing a descrambling information field where the whole or part of the control words for descrambling are recorded.

39. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 37, wherein said CPTC information is formatted further containing a CA field where CA information for conditional access is recorded.

40. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 37,wherein said generational copy control field is made up of a permissible generational field for limiting the number of copy permissible and a present generational field for indicating the present generation of a program copied.

41. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 37, wherein said reproduction control field is made up of a reproducible number field for limiting the number of reproducing a copied program, and a maximum reproducible time field for limiting time to reproduce the copied program.

42. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein the data recorded on cassette tape contains scrambled audio/video bit stream and CPTC information.

43. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 42, wherein said CPTC information is overwritten on the scrambled audio/video bit stream for the error effect and recorded on cassette tape.

44. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 42, wherein said CPTC information is recorded on a portion of any of the audio track of cassette tape, the control track of cassette tape, or the video track of cassette tape.

45. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein said all of the control word is contained in the CPTC information.

46. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 33, wherein said bit stream transmitted contains ECM and EMM.

47. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 46, wherein part of the control word is contained in the CPTC information.

48. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 47, wherein the remainder of the control word is contained in the ECM.

49. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 47, wherein the remainder of the control word is contained in the EMM.

50. An illegal view/copy protection apparatus for a dig-

ital broadcasting system as claimed in claim 46, wherein the whole control word is contained in the ECM.

51. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 46, wherein the whole control word is contained in the EMM.

52. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 46, wherein said program receiving portion comprises:

an IRD for receiving, decoding and descrambling the bit stream transmitted from said broadcasting medium, outputting analog audio/video data to be displayed and outputting scrambled digital audio/video data to be recorded on cassette tape; and
a smart card for decrypting the bit stream output from said IRD, detecting/analyzing the CPTC information, and outputting the control word and signals for controlling illegal view/copy to said IRD in order to perform conditional access and copy protection.

53. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 52,wherein said program receiving portion further comprises a lookup table for, in case that the EMM is updated by a broadcasting station for the purpose of copyright protection, storing EMM having information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction to said smart card in order to continuously reproduce the program of copied cassette tape.

54. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 52, wherein said program receiving portion further comprises a DVCR for recording the digital audio/video data and CPTC information scrambled and output from said IRD on cassette tape, and reproducing the scrambled digital audio/video data and CPTC information recorded on cassette tape to be output to said IRD.

55. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 54, wherein said smart card comprises:

an ECM filter for filtering the ECM from the bit stream output from said IRD;
a CPTC/tape state signal filter for filtering the CPTC information and the tape state signal indicative of the state of tape from the bit stream output from said IRD;
an EMM filter for filtering the EMM from the bit stream output from said IRD;
a lookup table for, in case that the EMM is updated for copyright protection by a broadcasting station, storing the previous EMM containing information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction in order to continuously reproduce the program of cassette tape copied;
an EMM processing portion for processing the EMM using the EMM output from said EMM filter and lookup table and the tape state signal output from said CPTC/tape state signal filter;
a CPTC processing portion for processing the CPTC information using the signals output from said CPTC/tape state signal filter and EMM processing portion; and
a CA processing portion for outputting control word CW using the signals output from said ECM filter and EMM processing portion.

56. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 54, wherein said smart card comprises:

an ECM filter for filtering the ECM from the bit stream output from said IRD;
an EMM filter for filtering the EMM containing the EMM from the bit stream output from said IRD;
a tape state signal filter for filtering the tape state signal output from said IRD;
a lookup table for, in case that the EMM is updated for copyright protection by a broadcasting station, storing the previous EMM containing information required to decode the CPTC information, and outputting CPTC information corresponding in reproduction in order to continuously reproduce the program of cassette tape copied;
an EMM processing portion for processing the EMM using the EMM output from said EMM filter and lookup table and the tape state signal output from said tape state signal filter;
a CPTC processing portion for processing the CPTC information using the signals output from said EMM filter and tape state signal filter, to thereby output ECM, enable erase signal and ID signal; and
a CA processing portion for outputting control word CW using the signals output from said ECM filter and EMM processing portion.

57. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 54, wherein said DVCR comprises:

a deck mechanism;
a recording/reproducing portion for recording digital data on cassette tape according to said deck mechanism and reproducing the digital data recorded on cassette tape;
a reproduction number detecting/updating portion for detecting/updating the reproduction number from the digital data reproduced from said recording/reproducing portion, and outputting it to said IRD in order to rerecord it in said recording/reproducing portion;
a digital data processing portion for processing the digital data reproduced from said recording/reproducing portion, outputting it to said IRD, and outputting switching position information for recording and reproducing;
a recording/playback switching portion for outputting a switching signal for controlling the reproduction number, the reproduction of digital data and the recording of the updated reproduction number using the switching position information output from said digital data processing portion; and
an error correction encoder/decoder for correcting the error of data output from said digital data processing portion, and encoding and decoding the data to be output to said digital data processing portion.

58. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 35, wherein said program receiving portion comprises:

a DVCR for detecting/analyzing the CPTC information from the bit stream transmitted from said recording medium, outputting a control word and signals for controlling illegal view/copy, and reproducing scrambled digital audio/video data; and
an IRD for receiving the control word and signals for controlling illegal view/copy output from said DVCR 232, descrambling the scrambled digital audio/video data, and outputting analog audio/video data to be displayed or recorded.

59. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 58, wherein said DVCR comprises:

a CPTC detecting/processing portion for detecting/analyzing the CPTC information from the bit stream transmitted from said recording medium, and outputting the control word and signals for illegal view/copy; and
a reproducing portion for reproducing the bit stream transmitted from said recording medium and outputting it to said IRD.

60. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 59, wherein said IRD comprises:

a digital output controller for receiving the control word and signals for controlling illegal view/copy output from said CPTC detecting/processing portion, and controlling the output of the scrambled digital audio/video data output from said reproducing portion in order to display them;
a descrambler for descrambling the scrambled digital audio/video data output from said digital output controller according to the control word output from said digital output controller; and
a display processing portion for processing and outputting the digital audio/video data output from said descrambler in order to display them.

61. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 35, wherein said program receiving portion comprises:

a DVCR for reproducing the scrambled digital audio/video data and CPTC information recorded on cassette tape through a recording medium, and outputting them to said IRD;
an IRD for decoding/descrambling the bit stream transmitted from said DVCR, and outputting analog audio/video data to be displayed; and
a smart card for decrypting the bit stream output from said IRD, detecting/analyzing the CPTC, and outputting the control word and signals for controlling copying to said IRD to thereby perform copy protection and/or conditional access.

62. An illegal view/copy protection apparatus for a digital broadcasting system as claimed in claim 54 or claim 61, wherein said IRD comprises:

a recording/digital output controller for decoding the bit stream transmitted from the broadcasting medium and DVCR, outputting to said smart card, receiving the control word and signals for controlling illegal view/copy output from said smart card, and controlling the output of the scrambled digital audio/video data for the purpose of recording and displaying;
a descrambler for descrambling the scrambled digital audio/video data output from said recording/digital output controller according to the control word output from said recording/digital output controller; and
a display processing portion for processing and outputting the digital audio/video data output from said descrambler to be displayed.

F I G.1
(conventional art)

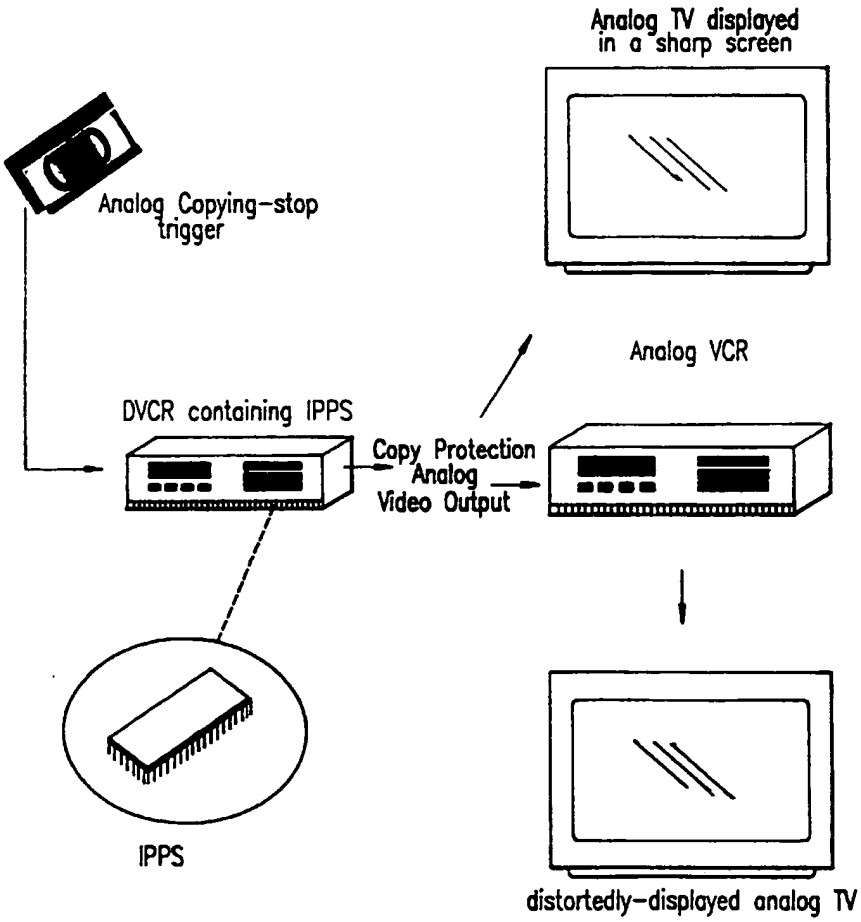
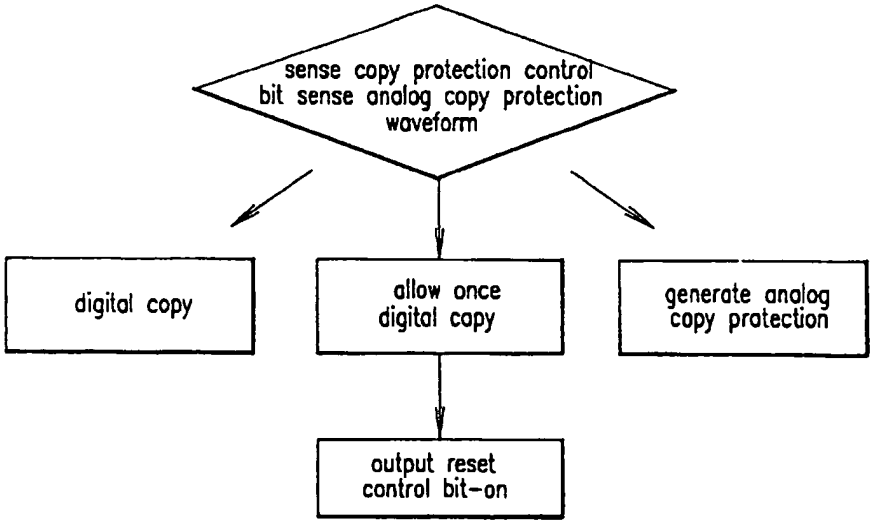
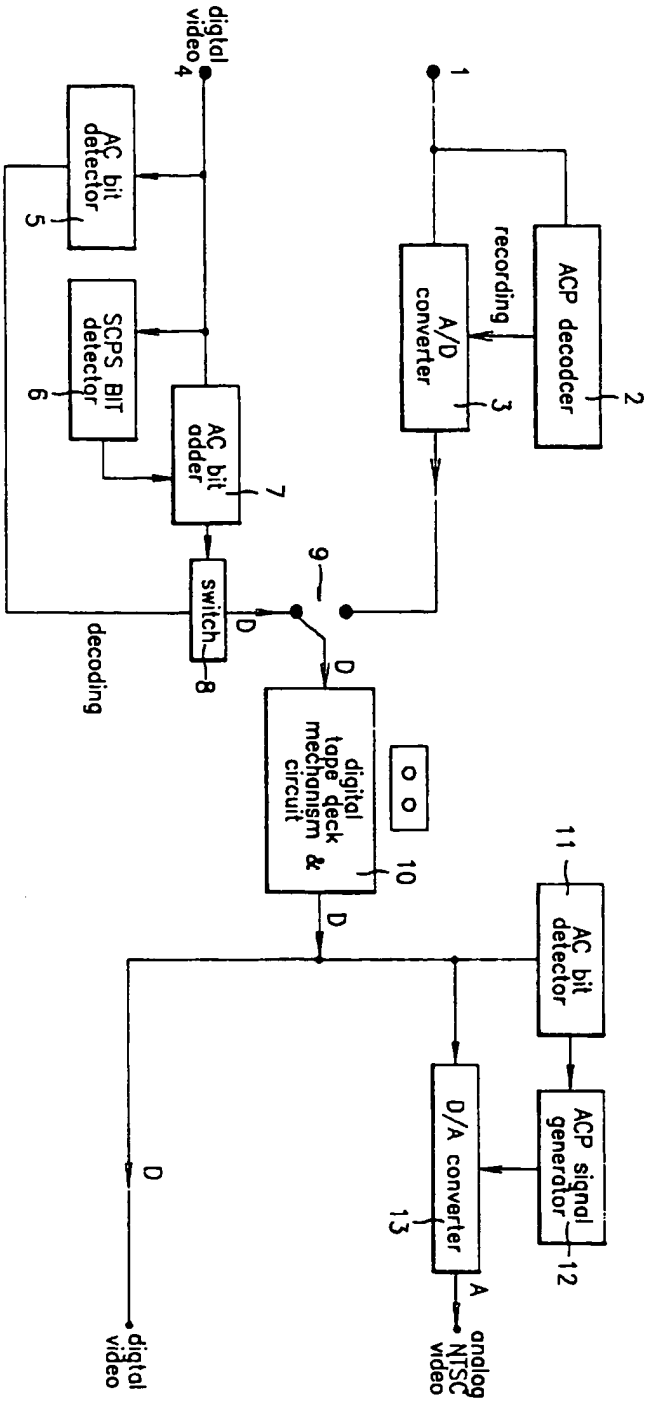


FIG.2
(conventional art)





F I G.3

FIG.4

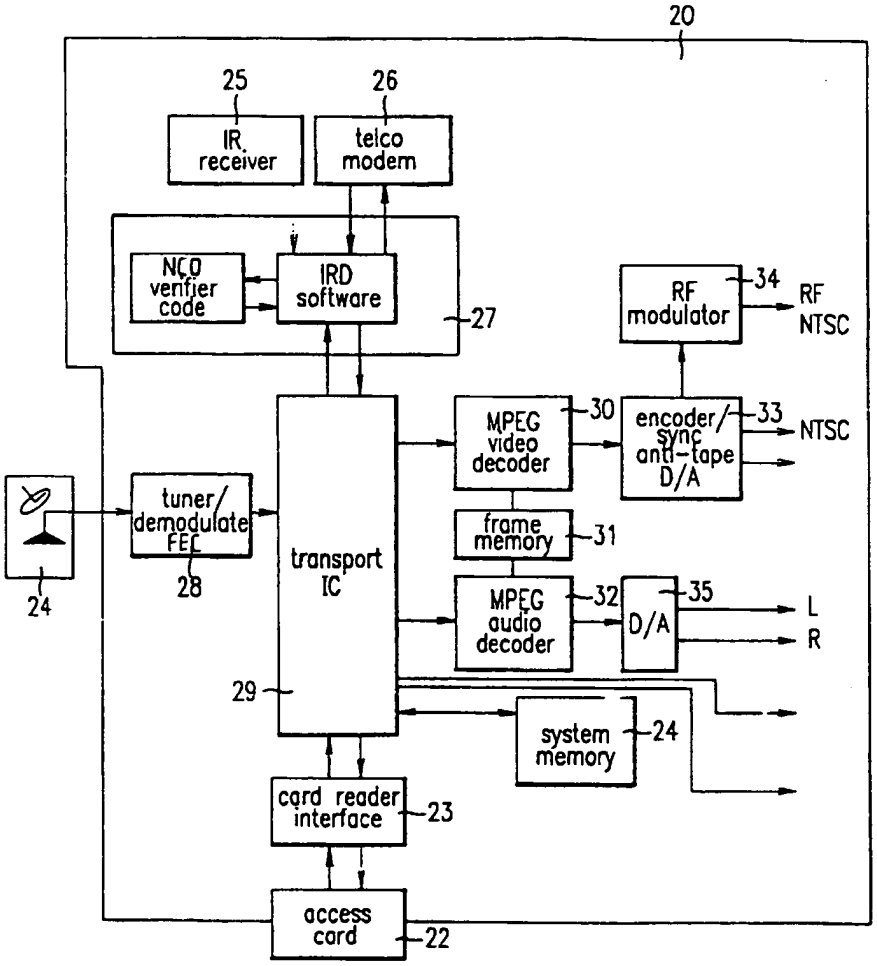


FIG.5

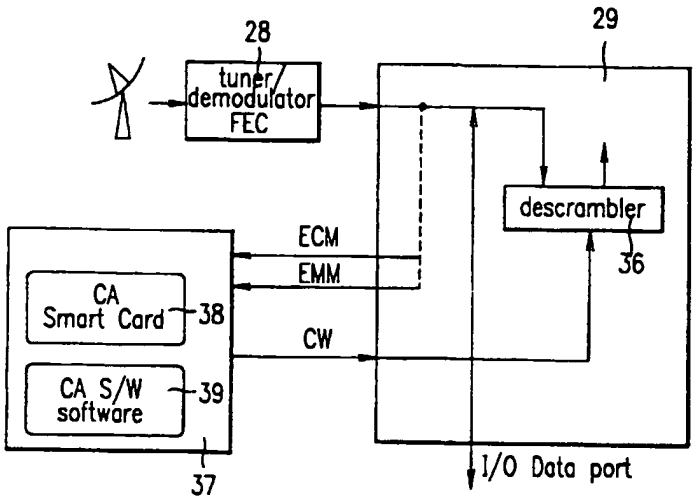


FIG.6a

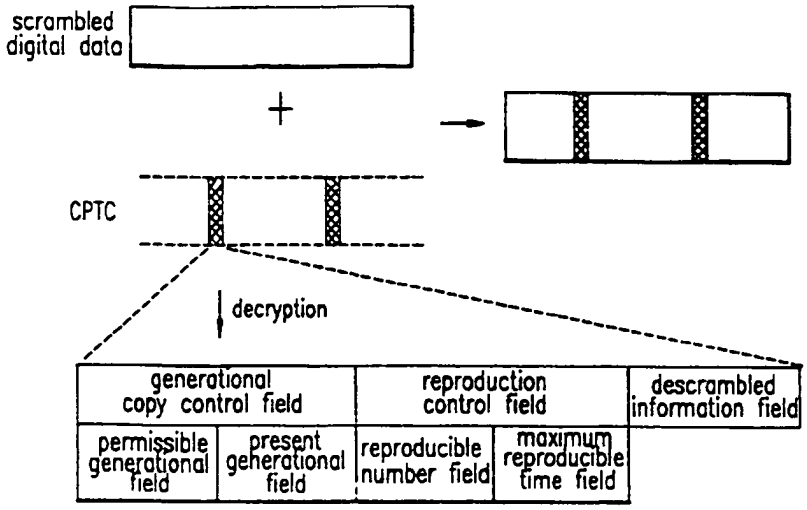


FIG.6b

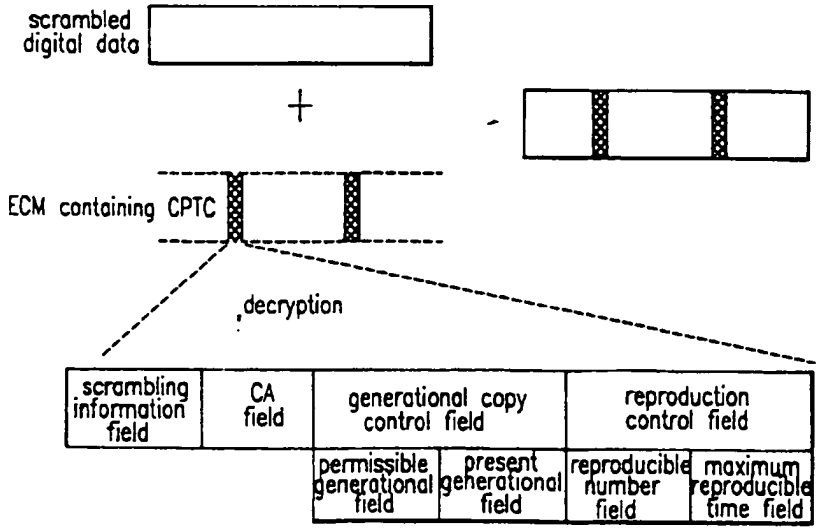


FIG.7

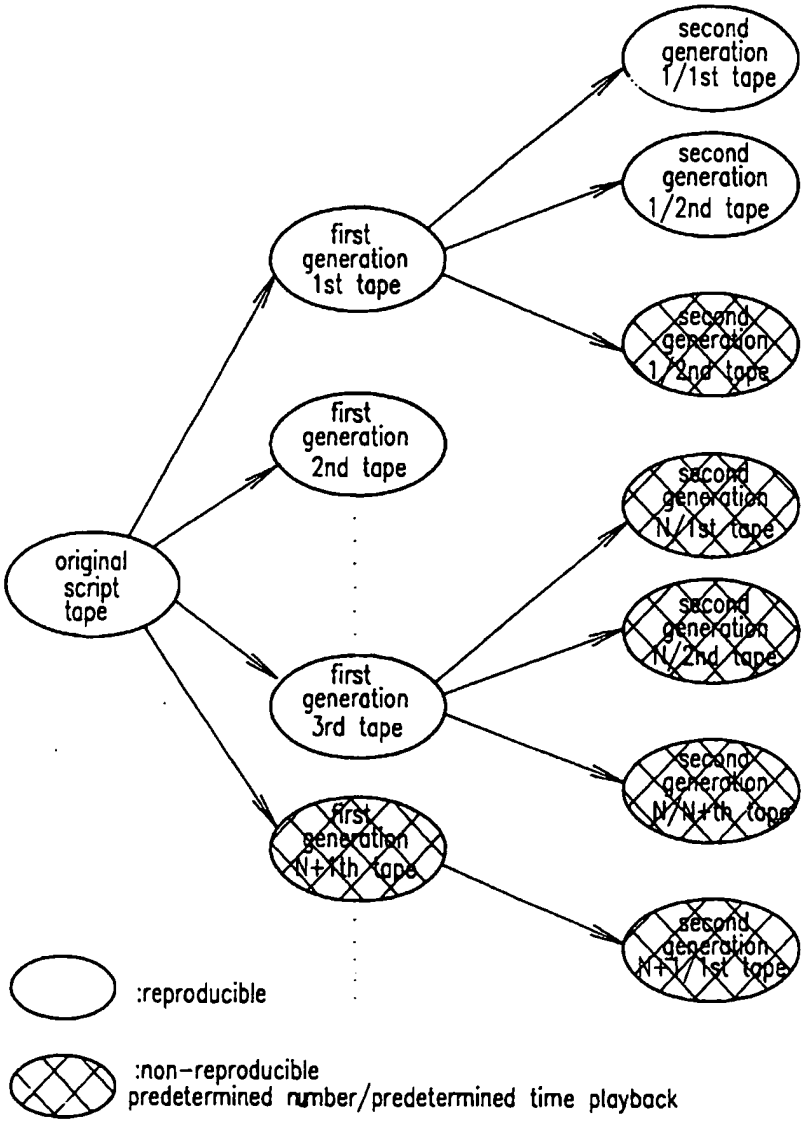


FIG.8a

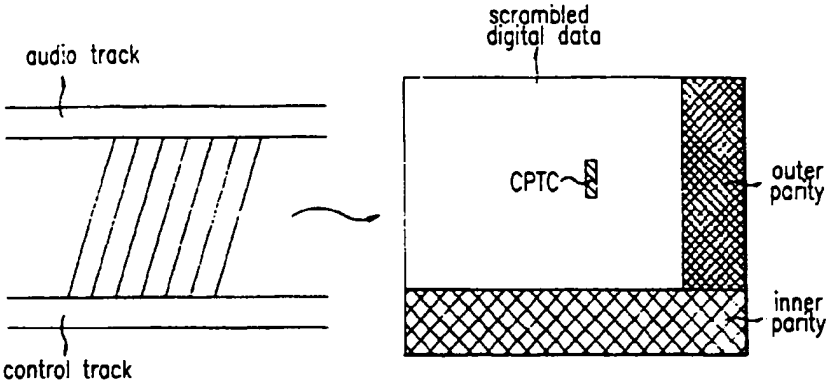


FIG.8b

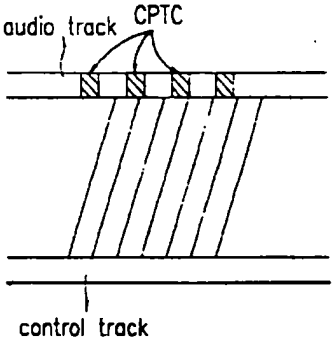


FIG.8c

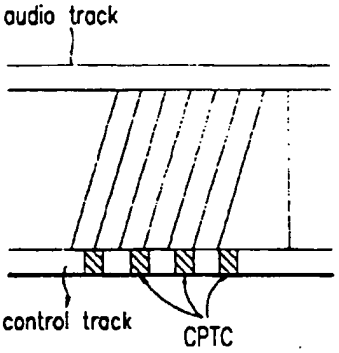


FIG.8d

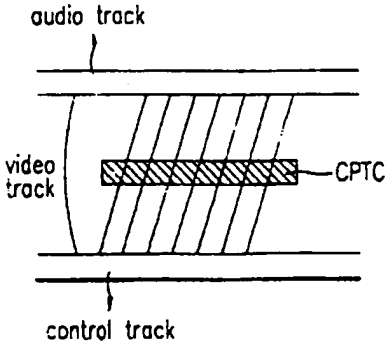


FIG.9

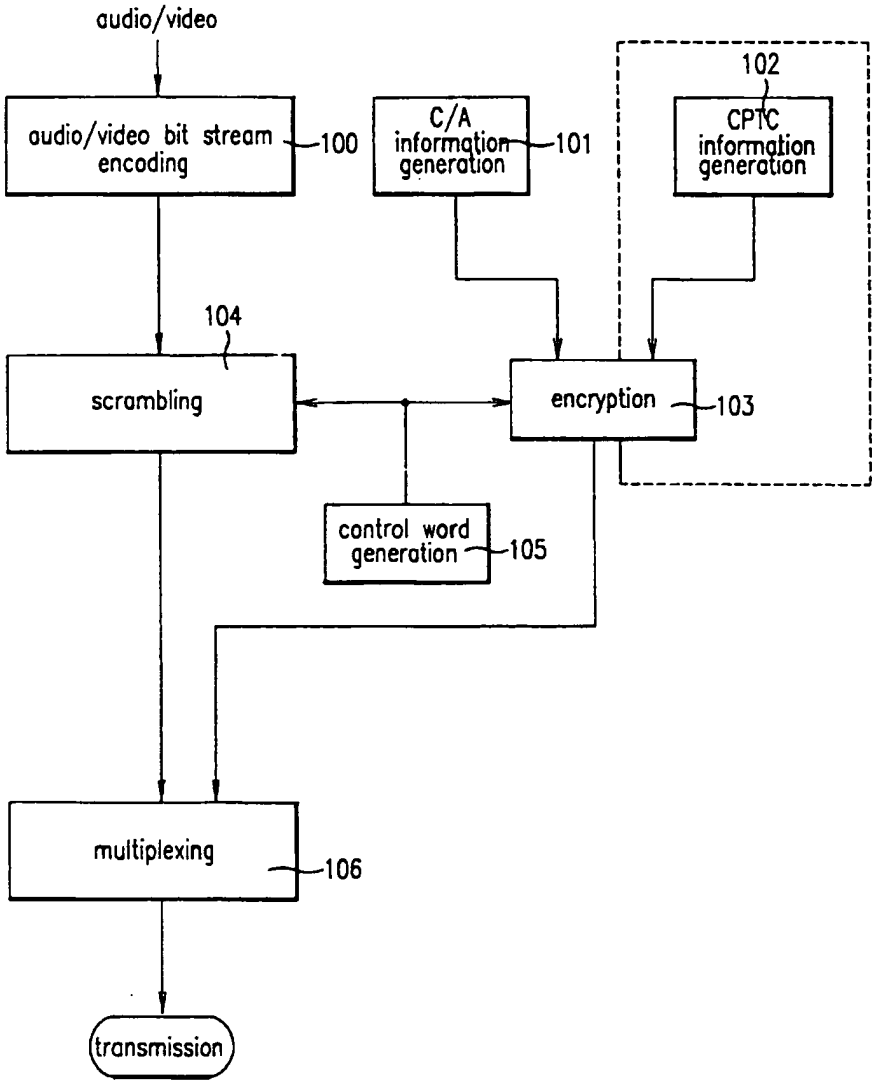
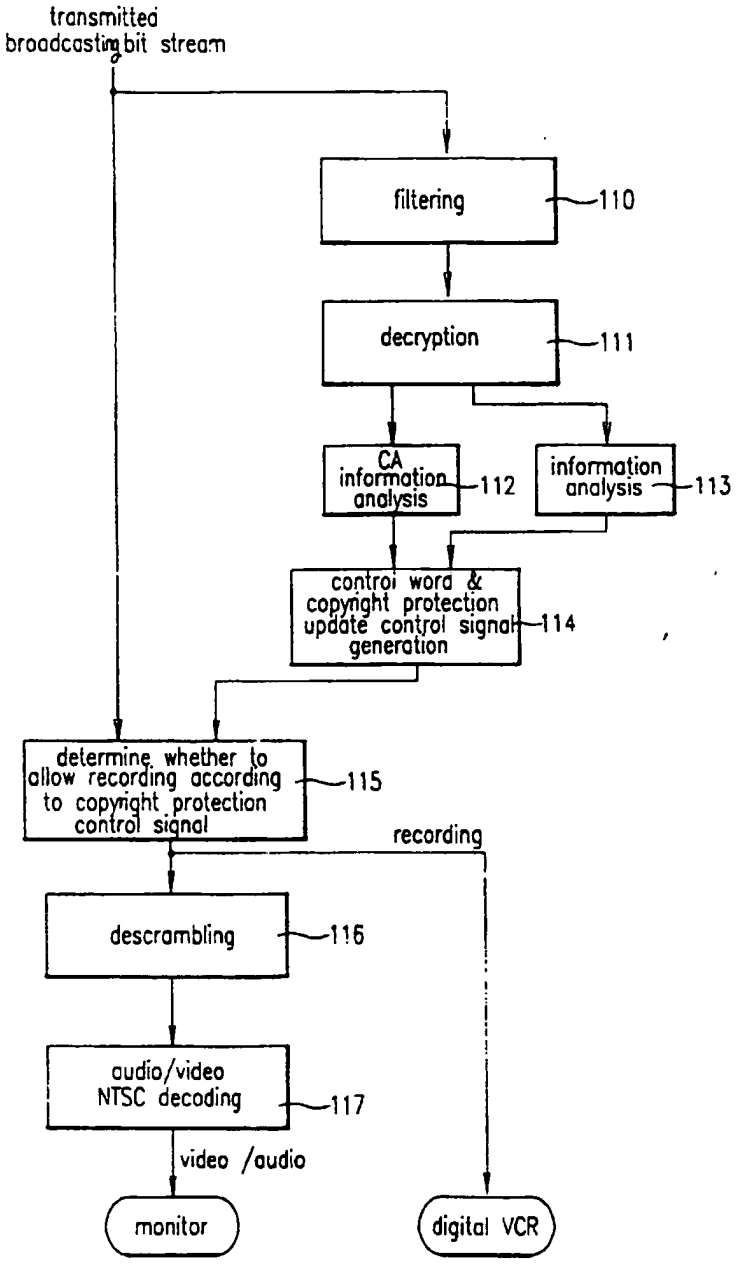


FIG.10



F I G.11

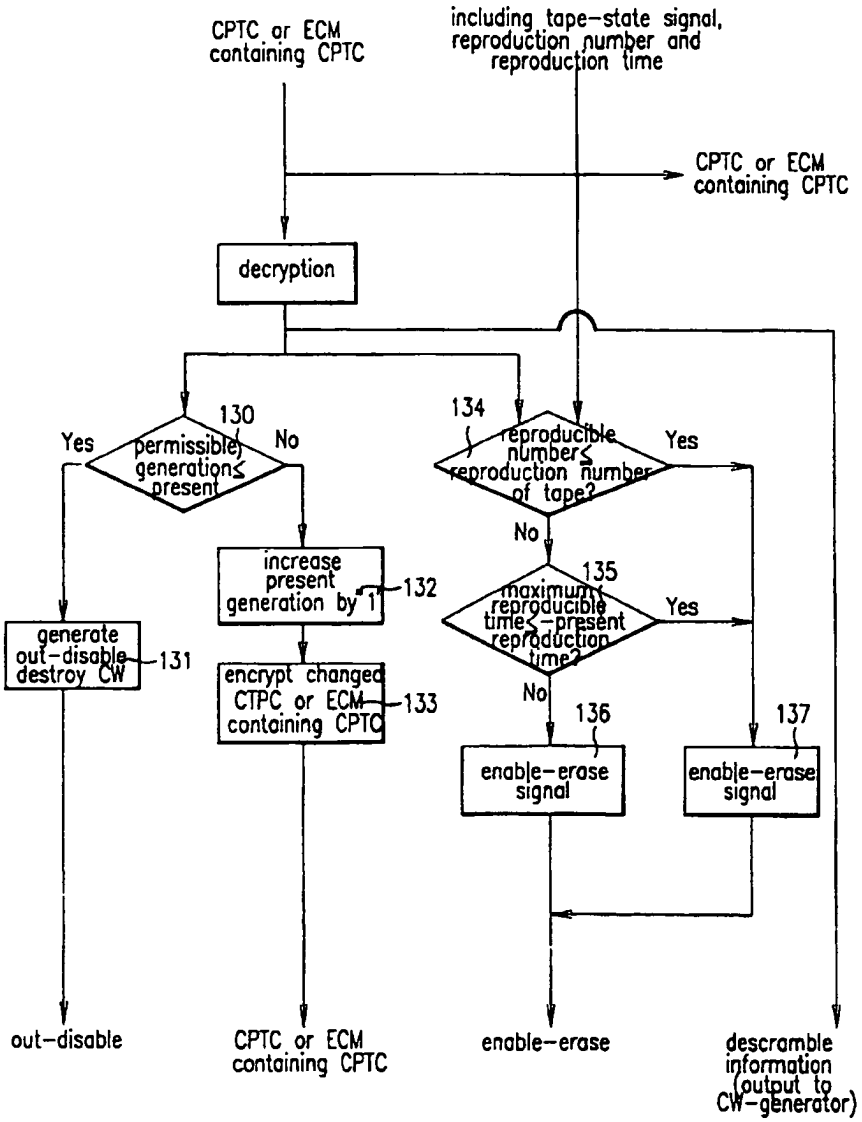


FIG.12

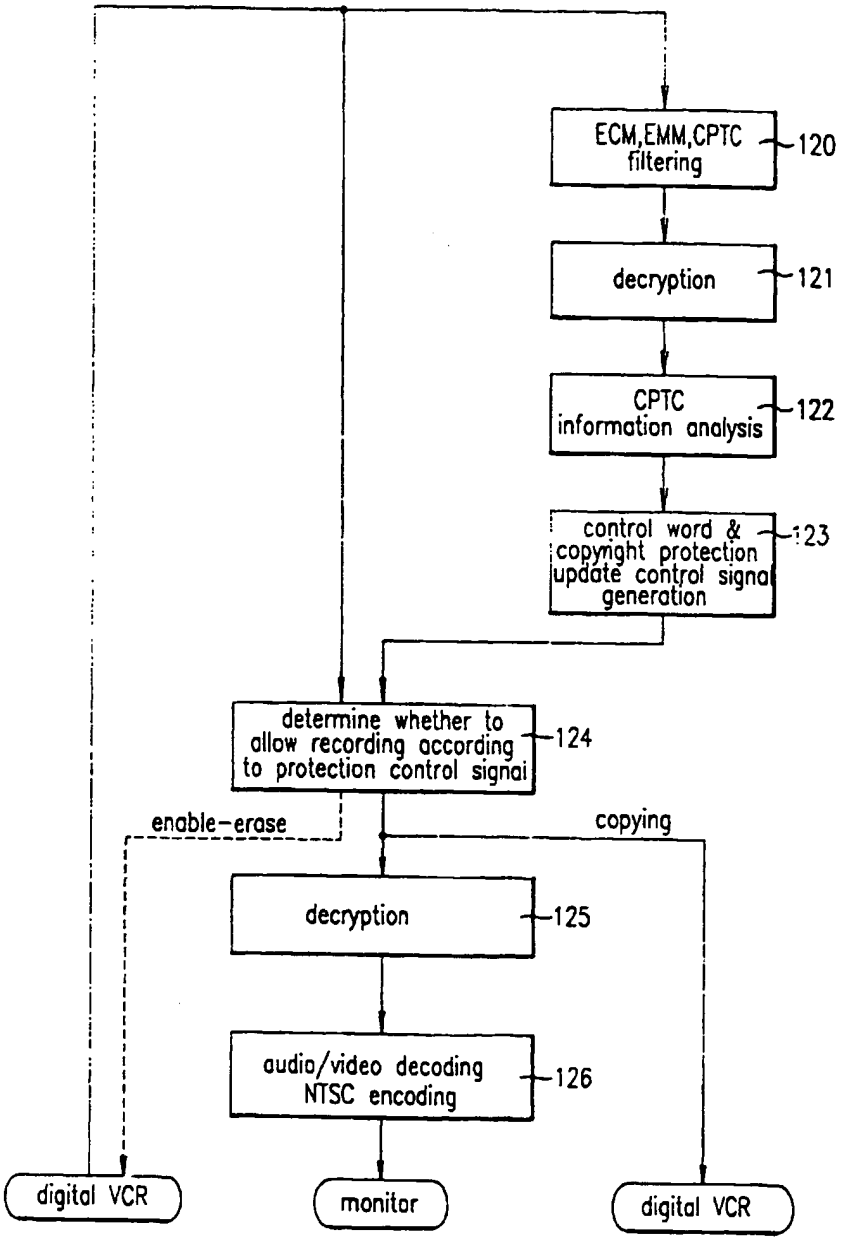


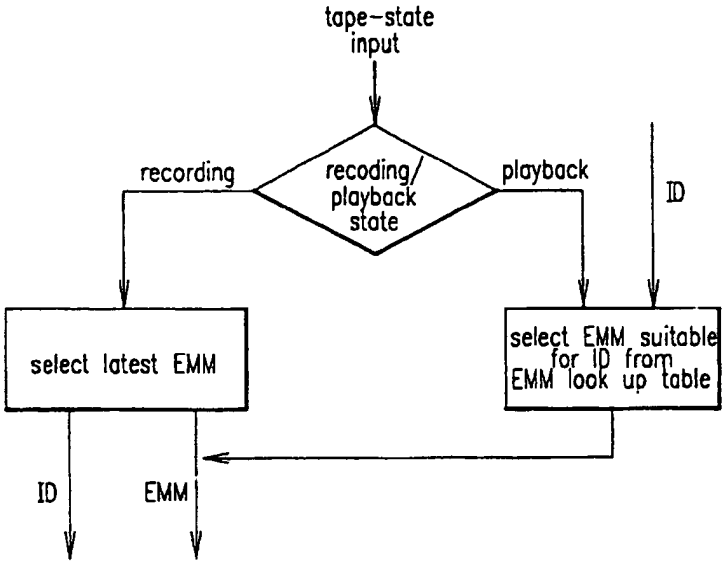
FIG.13

ID_1	EMM_1
ID_2	EMM_2
ID_3	EMM_3
. . .	. . .
ID_n	EMM_n

FIG.14

recording/reproduction state	ID	reproduction number

FIG.15



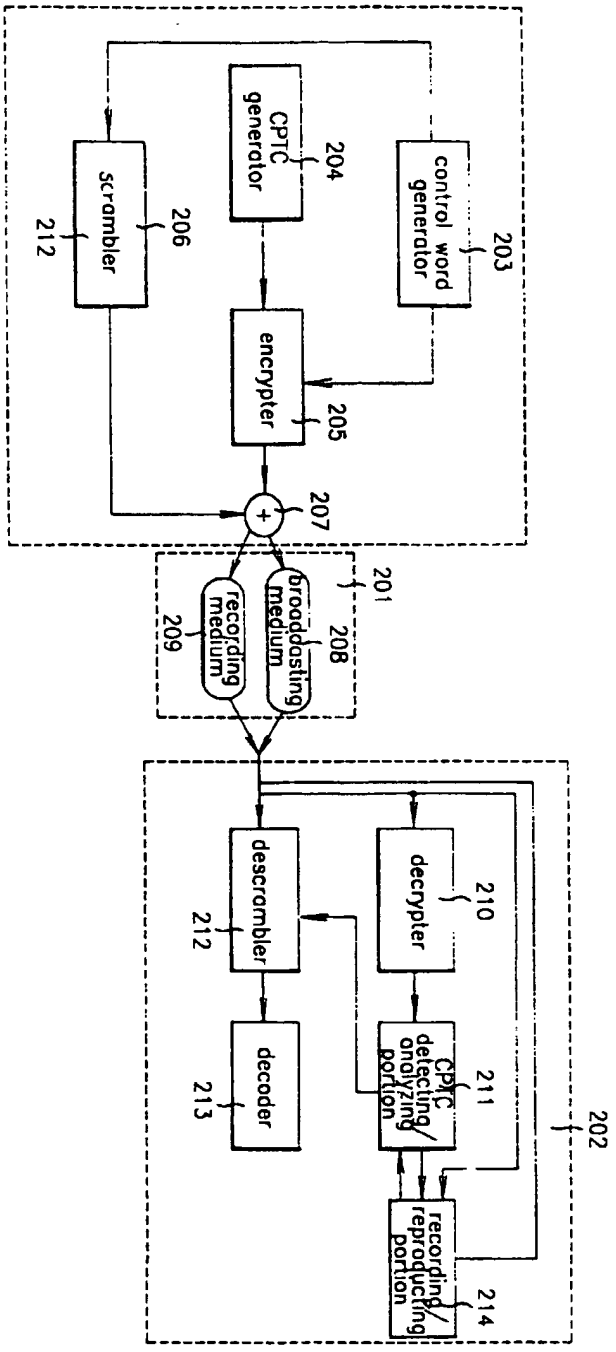


FIG.16

FIG.17a

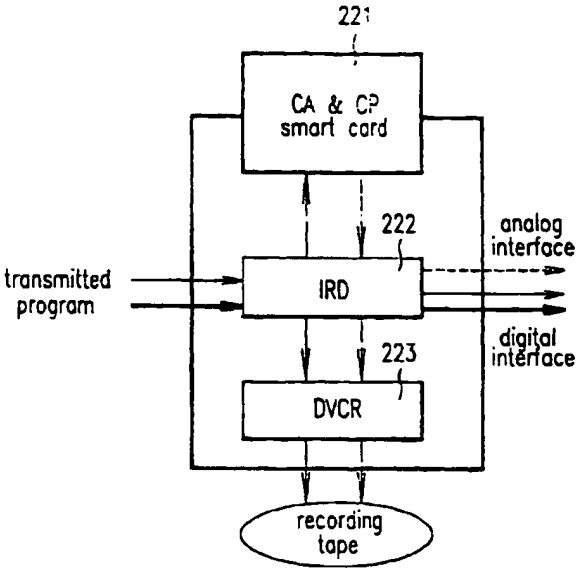


FIG.17b

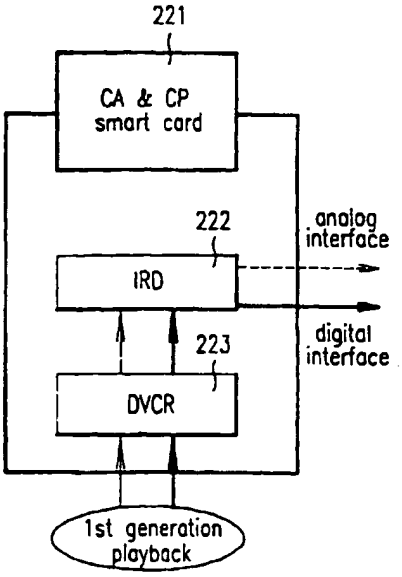


FIG.18

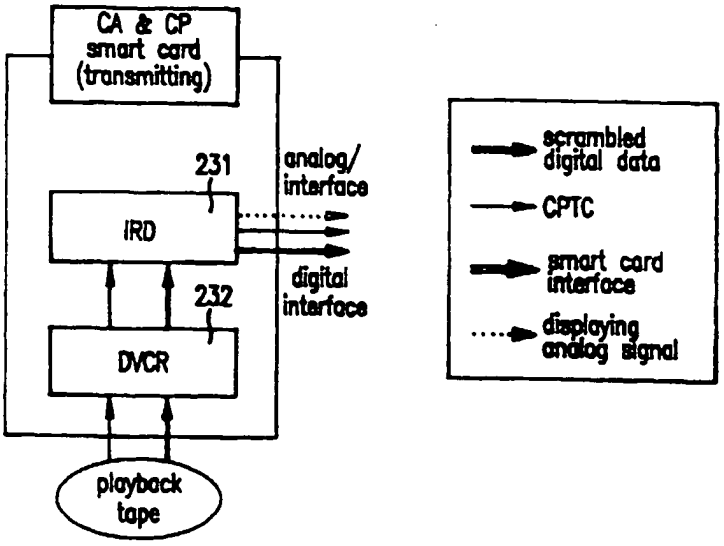


FIG.19

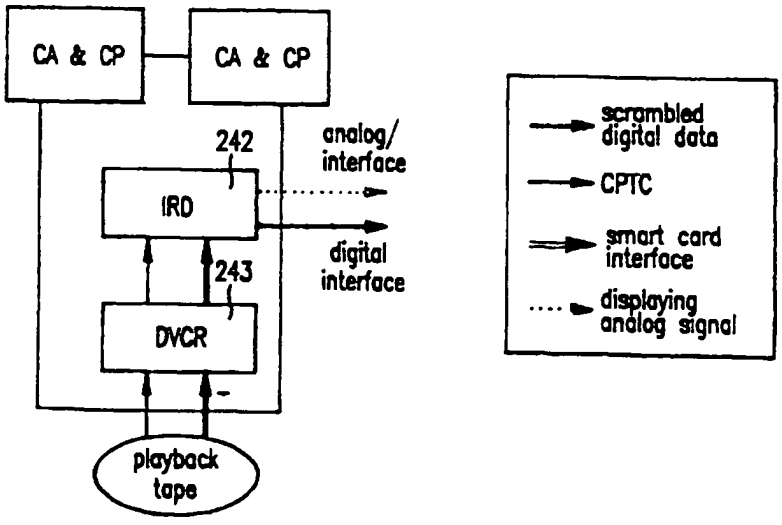
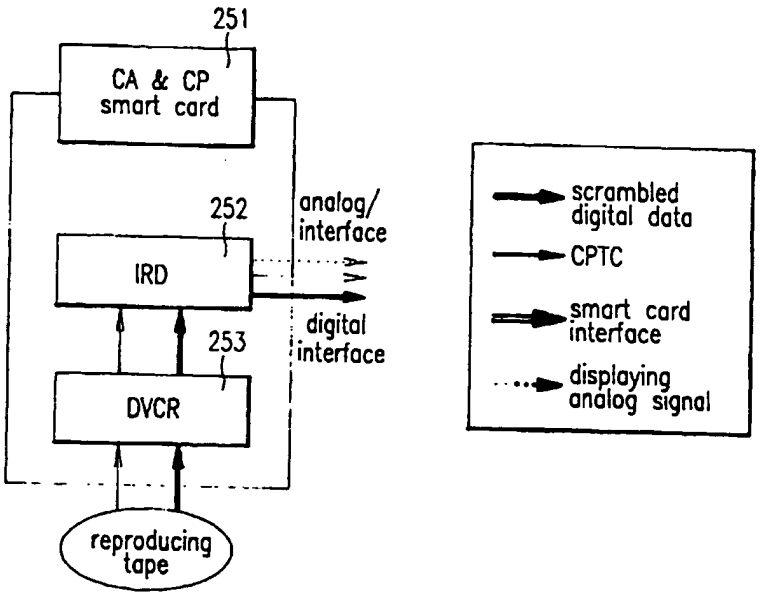
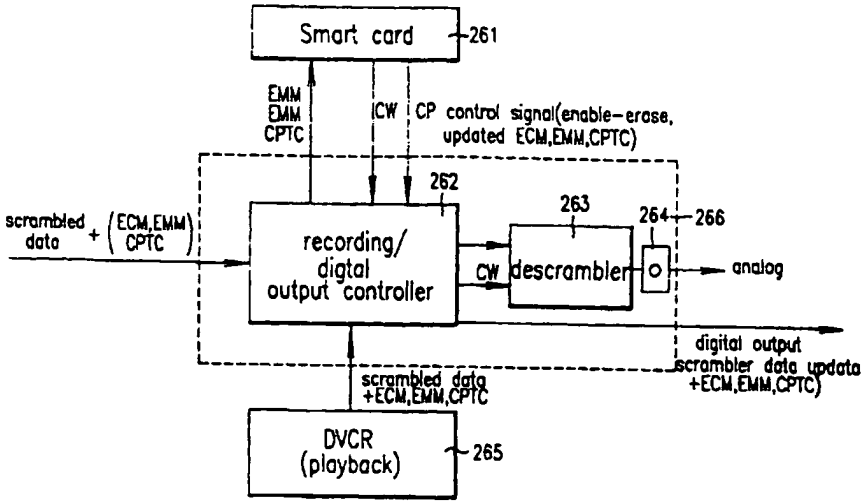


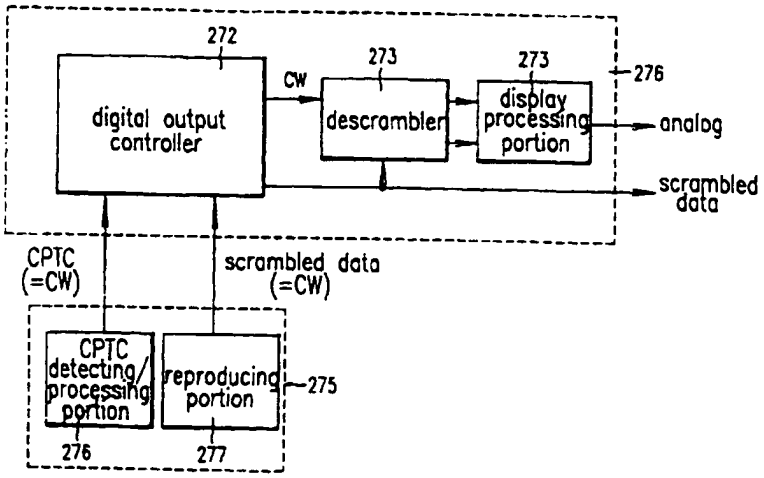
FIG.20



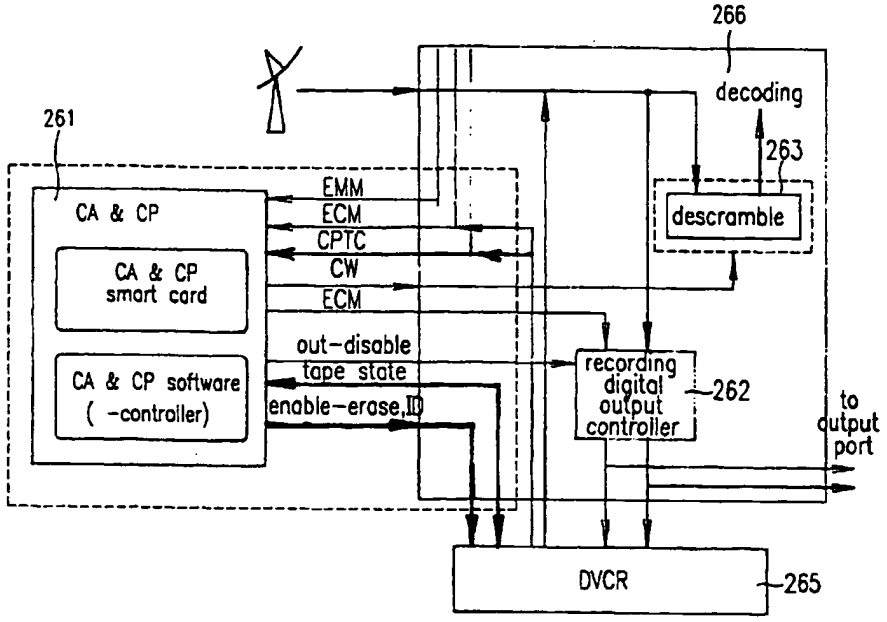
F I G.21



F I G.22



F I G.23



F I G.24

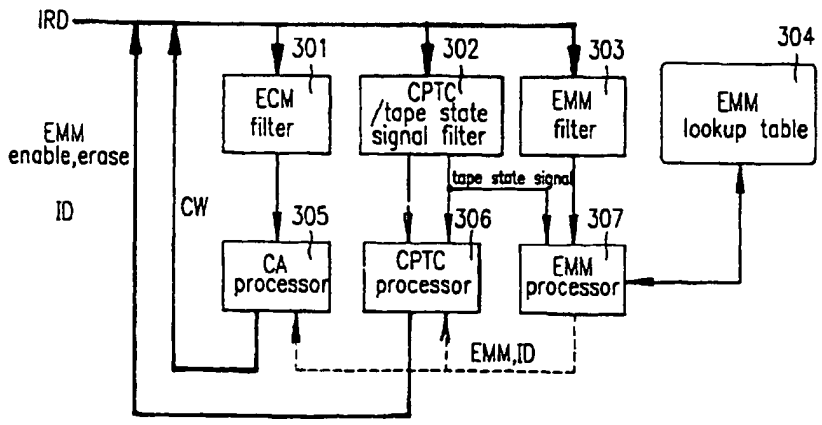


FIG.25

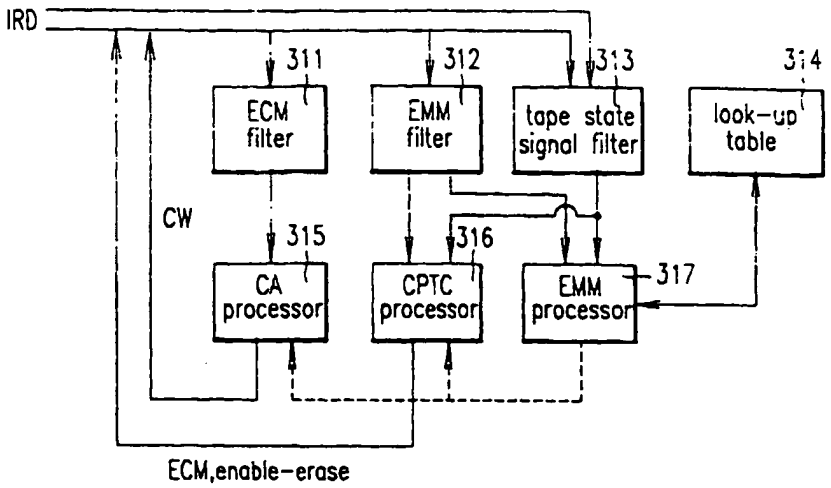
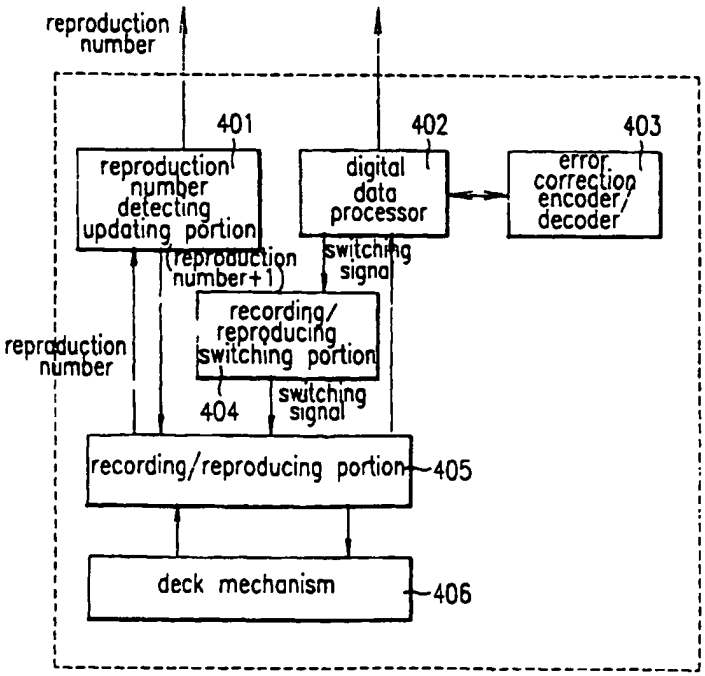


FIG.26



(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 818 748 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
14.01.1998 Bulletin 1998/03

(51) Int Cl.[6]: **G06F 17/60**

(21) Application number: **97304946.3**

(22) Date of filing: **07.07.1997**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **08.07.1996 JP 178130/96**
21.05.1997 JP 130626/97

(71) Applicant: **Murakoshi, Hiromasa**
Koriyama-shi, Fukushima, 963 (JP)

(72) Inventor: **Kanno, Kazuhiro**
Koriyama-shi, Fukushima, 963-02 (JP)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT,
27 Furnival Street
London EC4A 1PQ (GB)

(54) Software management system and method

(57) An operation management system for managing the operation of a managed software product. When a management target function is executed, reference is made to a battery value and, if the value is zero or greater, the function is allowed to be executed. The battery value is decremented as the function is executed. A charge value is supplied on a charge disk, such as a floppy disk, to allow the user to increase the battery value and to extend the usage period of the managed software product. The charge value may be supplied over a communication line.

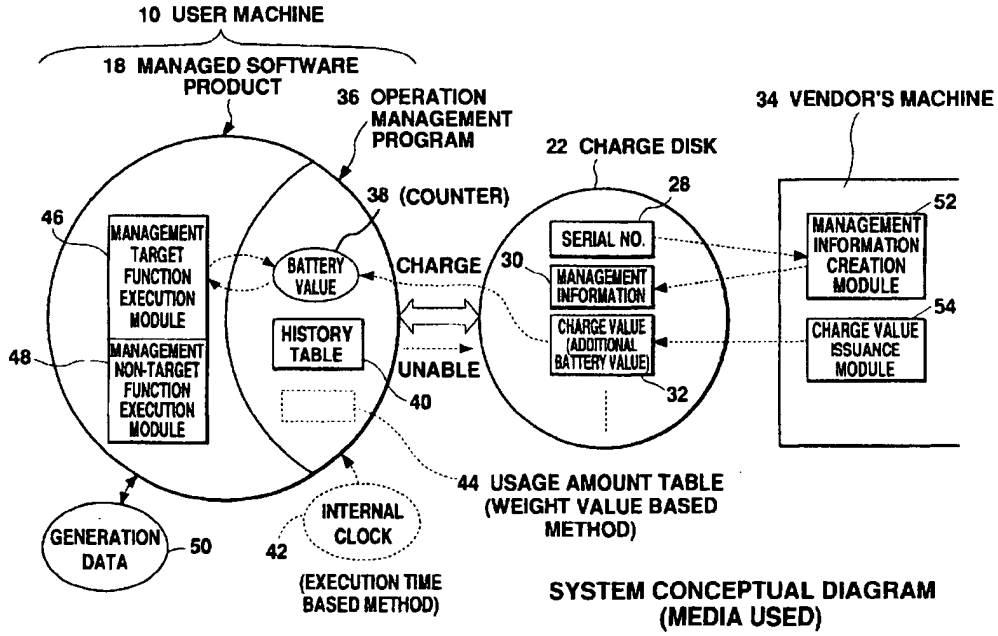


Fig. 3

EP 0 818 748 A2

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to an operation management system and an operation management method, and more particularly to software operation management or execution management.

Description of the Related Art

As computers and computer use become more common, more advanced technology is introduced and a variety of software products are developed for use in various fields. However, in many cases, the user finds it difficult to select a product from among a variety of software products that seem to meet the user's requirements; often, the user cannot find the best tool for his needs.

To reduce such a risk, a service has been available that supplies the user with a trial-use software product free of charge. However, most of these trial-use software products contain only function descriptions or provide the user with limited functions (e.g., save function and/or output function is/are not included). This makes it difficult for the user to evaluate the actual product (all the functions) correctly.

A sales system which charges the user according to how long the user actually uses a software product (including a trial use) would allow him to buy the product anytime he wants, to fully evaluate the product, and to precisely determine the requirements for continued use (including payment for it). Many users would find this type of sales system appealing and economical.

In Japanese Patent Laid-Open Publication No. Sho 59-41061 and Japanese Patent Laid-Open Publication No. Sho 63-153633, a system is disclosed that automatically prevents a program from being used when the usage count reaches a specified value. In Japanese Patent Laid-Open Publication No. Hei 1-147622 a system is disclosed which accumulates program execution time (total program execution time) and prevents the program from being used when the accumulation time reaches a specified amount. However, these systems do not disclose means for extending the program usage period. Japanese Patent Laid-Open Publication No. Hei 5-134949 discloses a system in which a program and expiry of the program are downloaded from a host computer to a user computer via a communication line. Also disclosed is a system in which a new expiry of the program is downloaded from the host computer to the user computer in order to update the expiry. However, the system only measures the execution time taken for executing the entire program, and does not include any means for changing the expiry on the user computer.

In Japanese Patent Laid-Open Publication No. Hei 7-234785, a system is disclosed that relates to a software rental system. This system connects a computer in a rental company to a user computer on which a rental software product is running over a communication line. When the time elapsed from the rental start time reaches the rental limit time, the system makes the program unavailable for use. (For example, the program is deleted.) To allow the user to update the rental period, the rental company sends a rental period extension program to the user's computer over a communication line. The user runs this program to extend the rental period of the program. A drawback of this system is that the user must pay for the software product regardless of whether the user has used it frequently or not. This means that the amount of money the user has to pay depends, not on how often he has used it, but on how long he has used it.

In Japanese Patent Laid-Open Publication No. Hei 7-244585, a system is disclosed that manages the program usage period. This system assigns a usage limit date to a program and, when the current date becomes greater than the limit date, the program product is made unavailable. To extend the usage limit date, the system reads update limit data from a recording medium containing that data and re-assigns a usage limit date based on the update limit data. This system is not reasonable because the amount of money the user has to pay does not depend on whether or not the user actually uses the program.

For example, during execution of a Computer Aided Design (CAD) software product, the user often spends much time thinking without entering data. In the system disclosed by the above mentioned Japanese Patent Laid-Open Publication No. Hei 7-234785 or Japanese Patent Laid-Open Publication No. Hei 7-244585, the user must pay for this thinking time. This places unwanted pressure on the user, especially when he must think carefully during program execution.

SUMMARY OF THE INVENTION

The present invention seeks to solve the problems associated with the art described above. In view of the foregoing, it is an object of the present invention to provide an operation management system and method which reasonably manage the operation of a managed software product.

It is another object of the present invention to provide an operation management system and method which levy a charge according to the actual usage amount of the managed software product (or the amount of the result generated by the managed software product).

It is still another object of the present invention to provide an operation management system and method which manage the operation according to the property of each function of the managed software product.

(1) To achieve the above objects, an operation manage-

ment system for managing the operation of a managed software product according to the present invention comprises: battery value management means for decrementing a battery value according to the operation amount of the managed software product; operation limit means for limiting the operation of the managed software product when the battery value has decreased to a specified limit value; and charge means for adding a charge value to the current battery value when the charge value is entered from external means.

The "battery value" mentioned above is a "virtual battery" which drives a managed software product. This battery value is preferably the value of a counter.

The battery value management means decrement the battery value according to the operation amount of the managed software product. When the battery value has reached a specified limit value (for example, 0), the operation limit means limit all of or a part of the operation of the managed software product. Upon receiving a charge value (additional battery value) from the external means, the charge means add the received value to the current battery value, thus extending the operation period. That is, the battery value is incremented, just as a battery is charged, to allow the continued use of the managed software product.

The managed software product described above is preferably a packaged application software program including a CAD program, game program, video program, language processor, music program, communication program, or a measurement program.

The battery value management means, operation management means, and charge means described above should be implemented preferably as software programs (management software programs) that run on a computer. The managed software product and the management software product may be separate, or the whole or a part of the management software product may be included in the managed software product.

A system according to the present invention is implemented on a general-purpose computer or special-purpose computer having such peripheral units as a disk drive, display, and input unit. The external means described above include recording media such as a magnetic disk or an optical disk and other host computers connected over a network.

(2) An operation management system according to the present invention may be applied to an application software product sales system. The following explains an example:

A vendor sells an application software product containing the operation management program according to the present invention. The operation management program has a battery value defined as the initial value. In addition to this product, the vendor sells recording media containing charge values (e.g., floppy disk (FD)). In this case, it is desirable that a variety of recording media, each containing a unique charge value, be supplied.

On the other hand, a user who bought the application software product may use the product until the battery value reaches zero. This allows the user to fully evaluate and examine the product. A user who wants to use the product after the battery value becomes zero must buy a recording medium containing a charge value to charge the battery. This enables him to add a charge value to the battery value and to use the product continuously.

If the specifications of the application software product do not satisfy the user's request, the user does not buy the recording medium. This prevents additional charges and reduces the cost to the user.

Considering an increase in the sales profit in recording media that will be produced in the future, a combination of a managed software product and the operation management program will lower prices significantly. The operation management system according to the present invention will increase the profits of both the user and the vendor, making it possible to build a very reasonable, economical system.

(3) In a preferred embodiment of the present invention, the battery value management means calculate the operation amount of each function of the managed software product, and subtracts a value corresponding to the operation amount from the battery value.

A continuous decrease in the battery value during execution of a managed software product, as in a conventional system, decrements the value even when the user is idle (input wait time), which places pressure on the user.

Calculating the operation amount of each function during execution of a managed software product, as in a system according to the present invention, decreases the battery value only when the managed software product is actually used, enabling the user to do operation without having to worry about time elapsed while thinking.

(4) In a preferred embodiment of the present invention, function category determination means are also available which determine if an execution instruction from the user activates a management target function or a management non-target function. And, the battery value management means decrement the battery value only when the management target function is executed.

For example, with the data generation function defined as a management target function and with other functions as management non-target functions, a cost can be levied only when new data are generated.

(5) In a preferred embodiment of the present invention, the battery value management means have a weight table containing an operation amount weight value for each of the management target functions. When any of the management target functions is executed, the battery value management means decrement the battery value by the weight value corresponding to the management target function.

In a preferred embodiment of the present invention,

the battery value management means measure the execution time of each of the management target functions and decrement the battery value by the value corresponding to the execution time.

This weight value system is able to calculate the operation amount regardless of the computer speed, which may differ among computers. In addition, by measuring time in this manner, the execution time is directly monitored and therefore the operation mount becomes proportional to the CPU load.

(6) In a preferred embodiment of the present invention, the operation limit means prevent only the management target functions from being executed when the battery value has decreased to a specified limit value; management non-target functions are executed.

For example, forcing a game program used at home to terminate when the battery value has reached a specified value does not cause a serious problem.

However, for a CAD program used in an office, forced termination when the battery value has reached a specified value may make already-produced data unavailable, possibly interrupting a job. Therefore, considering user's advantage and convenience, the embodiment keeps some functions operable even when the battery value has reached a specified value.

(7) A preferred embodiment of the present invention has remainder warning means for issuing a remainder warning message when the battery value has decremented to a specified warning value because a sudden inoperable condition in the managed software product without prior notice may cause the user unexpected damage. The remainder warning means alert the user to that condition before it occurs. In other words, the warning message prompts the user to determine whether to charge the battery value.

A preferred embodiment of the present invention has remainder display means for displaying the battery value on the screen during execution of the managed software product. This remainder display information keeps the user informed of the amount by which the managed software product will be able to continue operation without being charged.

It is also possible to program the system so that, upon detecting that the battery value has been charged to a specified value, the system can automatically disable operation management through the battery value to allow the user to use the product indefinitely.

(8) To achieve the above objects, a method for managing the operation of a managed software product according to the present invention comprises: a count value management step for changing a count value according to the operation amount of the managed software product; an operation limit step for limiting the operation of the managed software product when the count value has reached a specified limit value; and a charge step for charging the current count value or the limit value when a charge value is entered from external means.

The above count value is incremented or decremented according to the operation amount of the managed software product. When the count value is incremented, a charge value is added to the limit value; when the count value is decremented, a charge value is added to the current count value. In either case, the usage period is extended by charging the battery value.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing a user machine used in the operation management system according to the present invention.

Fig. 2 is a diagram showing the data structure of a charge disk.

Fig. 3 is a diagram showing the concept of the operation management system according to the present invention.

Fig. 4 is a diagram showing an example of the history table.

Fig. 5 is a diagram showing an example of the usage amount table.

Fig. 6 is a flowchart showing the processing of the system when a management target function is executed in the execution time based method.

Fig. 7 is a flowchart showing the processing of the system when a management target function is executed in the weight value based method.

Fig. 8 is a flowchart showing the charge disk read processing.

Fig. 9 is a flowchart showing the charge processing.

Fig. 10 is a diagram showing a user machine used in another embodiment.

Fig. 11 is a diagram showing the structure of data sent from the host machine to a user machine.

Fig. 12 is a diagram showing the concept of the system in another embodiment.

Fig. 13 is a diagram showing an example of the user registration table.

Fig. 14 is a flowchart showing the operation of the user machine and a user machine in another embodiment.

Fig. 15 is a diagram showing another configuration of the system.

Fig. 16 is a diagram showing an example of an application according to the present invention.

Fig. 17 is a flowchart showing the function category determination processing.

DESCRIPTION OF PREFERRED EMBODIMENTS

Fig. 1 shows a user machine 10. This user machine 10 is a computer which executes various types of application programs under control of the operation system (OS). The user machine 10 is composed of a system unit 12, display 14, keyboard (not shown in the figure), output unit (not shown in the figure) such as a printer or plotter, and so forth. The system unit 12 contains a CD-ROM disk drive 16 which accesses a CD-ROM and

reads data from it and a floppy disk drive 20 which accesses a floppy disk (FD) and reads data from it.

The CD-ROM shown in Fig. 1 contains a managed software product 18. In this embodiment, the managed software product 18, such as a CAD software product, has an operation management program built in. The operation management program, designed for managing the operation of the managed software product 18, manages the operation using a "battery value" which will be described below. In the example shown in Fig. 1, the managed software product 18 is installed from the CD-ROM to the user machine 10; it may be installed from any other recording medium or via a communication line.

A charge disk 22, containing specified data (including a charge value) on a floppy disk, functions as a battery value charger. Inserting this charge disk 22 into the floppy disk drive 20 causes a charge value to be read and enables the user to extend the allowable operation period of the managed software product 18. In this embodiment, several charge disks 22, each containing a unique charge value, are supplied to allow the user to select or buy a desired charge disk 22 to add a desired charge value to the battery value.

The managed software product 18 and the charge disk 22 are usually supplied from the same vendor. In this embodiment, the managed software product 18 includes the operation management program. Of course, the managed software product 18 and the operation management program may be separately loaded into the user machine 10.

In Fig. 1, the display 14 has a remainder information area 24 where remainder information is displayed and a remainder warning area 26 where a warning message is displayed when the remainder drops below the specified amount. These areas will be described later.

Fig. 2 shows the data structure of the charge disk 22. As shown in Fig. 2, the charge disk 22 contains a serial number 28, management information 30, and charge value (additional battery value) 32. The serial number 28 is a unique identification number that is assigned when the floppy disk is formatted. Usually, this number is not copied when the disk is copied. The management information 30 is created when the serial number 28 is encrypted. This management information 30 is copied when the disk is copied. Therefore, when the disk is copied illegally, the serial number 28 and the management information 30 do not match, thereby making it easy to determine that the disk is copied illegally. Of course, any other conventional security system may also be used instead of this method.

The charge value 32 is an additional charge value to be added to the battery value that is decremented as the user uses the managed software product 18. Charging the battery value with this charge value enables the user to extend the usage period.

When the battery value is managed in the "execution time based method" in which the battery value is decremented by the execution time of each function, an additional time is recorded as the charge value 32. On the other hand, when the battery value is managed in the "weight value based method" in which the battery value is decremented by the weight value of each function, the additional value is recorded as the charge value 32. These methods will be described in more detail later.

Although a floppy disk is used as the charge disk 22 in the embodiment shown in Fig. 1, other types of recording media may also be used. Also, as shown in another embodiment that will be explained later, a charge value may be sent over a communication line.

Fig. 3 shows the concept of the operation management system which uses the charge disk 22. The system is composed primarily of the user machine 10, charge disk 22, and vendor's machine 34. In this embodiment, the managed software product 18 including the operation management program 36 is installed in the user machine 10.

The charge disk 22 is generated on the vendor's machine 34 owned by the vendor which sold the managed software product 18. More specifically, the vendor's machine 34 has two software modules: the management information creation module 52 and the charge value issuance module 54. The management information creation module 52 encrypts the serial number 28 recorded on the charge disk 22, and writes the resulting management information 30 back onto the charge disk 22. Note that the operation management program 36, which contains the encryption condition or the decryption condition, can check whether or not the serial number 28 agrees with the management information 30. The charge value issuance module 54 records the charge value 32, which has been set by the vendor, onto the charge disk 22. In the execution time based method, the charge value 32 is recorded, for example, as 100 hours, 200 hours, or 500 hours. Note that the operation management program 36 contains an initial battery value (for example, 100 hours).

The operation management program 36 has a counter 38 which decrements the battery value (battery value management function). In this embodiment, the operation management program 36 decrements the counter 38 each time a "management target function" provided by the managed software product 18 is executed. When the battery value, i.e., the counter value, has decremented to the limit value of 0, the operation management program 36 prevents management target functions from being executed. That is, in this embodiment, when the battery value has reached a specified limit value, the execution of the managed software product 18 is limited and, when the battery value is charged with the charge value 32 contained on the charge disk 22, the charge value is added to the battery value and the resulting value is used as a new battery value. The usage period of the managed software product 18 is thus extended.

A history table 40 managed by the operation man-

agement program 36 contains history information on charge values recorded on the charge disk 22. Fig. 4 shows an example. As shown in Fig. 4, the history table 40 is composed of three columns: FD serial number column 40A, charge data/time column 40B, and charge value column 40C. The table may have other columns as necessary.

Referring to Fig. 3 again, the following explains how the battery value is managed. When the battery value is managed in the "execution time based method" described above, the execution time of each management target function, measured based on the internal clock 42, is subtracted from the battery value. On the other hand, when the "weight value based method" described above is used, the battery value is managed based on the usage amount table 44. Fig. 5 shows an example of the usage amount table 44. In this embodiment, the table contains entries, each consisting of a function name 44A and the corresponding usage amount 44B. It should be noted that each usage amount is used as a weight value. For example, a weight value is pre-defined according to the processing time of each function. Therefore, when a management target function is executed, the corresponding usage amount (weight value) is subtracted from the battery value.

The managed software product 18 shown in Fig. 3 has many user interface programs as well as many internal functions and common functions used by the programs. These functions are classified roughly into two: management target functions and management non-target functions. Whenever the managed software product 18 attempts to execute a management target function, the operation management program 36 references the battery value and, when it is zero or greater, allows the managed software product 18 to execute that function. When the managed software product 18 attempts to execute a management non-target function, the operation management program 36 does not check the battery value. For example, when input/output function for processing generated data 50 from the managed software product 18 is defined as a management non-target function, the input/output processing is always executed on the generated data 50, even if the usage period of the managed software product 18 has expired. This ensures that the generated data 50 are always processed, thus protecting user assets. Examples of management non-target functions include the data display function, data print function, and data plotter output function.

Management target functions include the data generation function. For example, when the managed software product is a CAD software product, the data generation function includes the straight-line drawing function, curved-line drawing function, circle drawing function, area fill-in function, area hatching function, and character insertion function.

Fig. 3 conceptually shows management target function execution module 46 which executes management target functions and management non-target function execution module 48 which executes management non-target functions. In this embodiment, the battery value is decremented only when a management target function is activated. Note that the battery may be decremented when both a management target function and a management non-target function are activated.

In addition to the data described above, the charge disk 22 may contain other types of data. For example, it may contain the name of the managed software product 18 which accepts a charge value. In this case, the name of the managed software product 18 is used as follows. When the charge disk 22 is read, the operation management program 36 checks whether or not the name of the managed software recorded on the charge disk 22 matches that of the managed software product 18 installed in the user machine 10 and, only when they match, accepts the charge value 32.

The battery value described above is stored on the hard disk and then copied into the computer's RAM. The battery value in the RAM is decremented whenever a management target function is executed. Also, at an interval or as necessary, the battery value in the RAM replaces the battery value on the hard disk. This means that, even when the computer fails, the battery value is not erased. The battery value may also be maintained in some other way.

Fig. 17 is a flowchart showing how the operation management program operates when it accepts an instruction requesting the execution of a managed software product function. The following explains this processing in more detail.

Upon receiving from a user an instruction requesting the execution of a function of the managed software product while the managed software product is in execution (S601), the operation management program checks whether the requested function is a management target function or a management non-target function (S602). When the function is a management target function (S603), the operation management program performs the processing shown in Fig. 6 or Fig. 7 (S604). When the function is a management non-target function (S603), the program executes the function immediately. (S605). This processing is repeated whenever an execution instruction is received.

Next, referring to Fig. 3, the execution of a management target function in the execution time based method is explained with the use of Fig. 6.

When the user requests the execution of a management target function while the managed software product 18 shown in Fig. 3 is in execution, the routine shown in Fig. 6 is started. First, the management target function execution module 46 or the operation management program 36 reads the battery value to check if it is greater than zero. If the battery value is zero or less, the routine is terminated. That is, the requested management target function cannot be started. Note that a management non-target function is started even if the battery value is

zero.

In S102, the routine gets the start time from the internal clock 42 before starting the requested management target function and, in S103, starts the management target function. In S104, the routine gets the end time from the internal clock 42 and, in S105, subtracts the start time from the end time to calculate the processing time (execution time) of the processing executed in S103.

In S106, the routine subtracts the processing time calculated in S105 from the battery value. In S107, the routine checks if the resulting battery value is equal to or less than the warning value and, if so, displays a message in the remainder warning area 26 shown in Fig. 1. If the resulting battery value is greater than the warning value, the routine does not display the message. As shown in Fig. 1, the remainder information area 24 is displayed during execution of the managed software product 18 (see Fig. 1) to allow the user to check the remaining amount. This helps the user determine how long he can execute the managed software product 18.

Fig. 7 shows the processing of a management target function in the weight value based method.

When the execution of a management target function is requested as described above, the routine references the battery value in S201 to check if it is equal to or greater than 0. If it is, the routine executes the requested management target function in S202 and, in S203, references the usage amount table 44 shown in Fig. 5 to find the usage amount (weight value) of the executed management target function. Then, in S204, the routine subtracts the processing amount found in S203 from the battery value to find a new battery value. In S205, the routine checks if the battery value is less than the warning value and, if so, displays a message in the remainder warning area 26 in S206.

The "execution time based method" shown in Fig. 6 allows the user to manage operation using a physical amount that is easy to understand. In addition, the user can manage operation in a relatively simple configuration. On the other hand, the "weight value based method" shown in Fig. 7 gives the user the same result regardless of the CPU speed of the user's machine.

Next, referring to Fig. 3, the charge disk 22 read processing is explained with the use of Fig. 8.

This processing is started when the charge disk 22 is inserted into the floppy disk drive 20 as shown in Fig. 1. The routine reads the serial number in S301, and the management information in S302, both from the charge disk 22. In S303, the routine encrypts the serial number according to the encryption condition, or decrypts the management information according to the decryption condition, and compares the serial number with the management information. This comparison determines whether or not the charge disk 22 is legal. For example, when the disk is illegally copied, the management information 30 is copied, but the serial number 28 is not copied but replaced. This results in a mismatch between the serial number 28 and the management information 30, thereby making it possible to find an illegal copy.

In S304, the routine checks if the charge disk 22 is valid and, if it is not valid, terminates processing in S308. If it is valid, the routine references the history table 40, containing past charge history data, in S305 to check the validity of the charge value 32 recorded on the charge disk 22. To do so, the routine first checks to see if the serial number 28 of the charge disk 22 is in the history table 40. If the serial number is found, the routine takes the following steps to check if the charge value 32 recorded on the charge disk 22 is valid. The routine finds the charge value initially recorded on the charge disk 22 and, from that initial value, subtracts the actual charge value to find the remainder. The next time the battery value is charged, the routine compares the remainder with the charge value currently recorded on the charge disk. If the charge value on the charge disk 22 is greater than the remainder, the routine determines in S306 that the charge disk is not valid and terminates processing in S308. If the routine finds that the charge value 32 on the charge disk 22 is valid, it performs the charge processing, shown in Fig. 9, in S307.

Fig. 9 shows an example of charge processing. In S401, the routine references the counter 38 to read the current battery value and, in S402, reads the charge value from the charge disk 22. In S403, the routine asks the user to type an actual charge value that does not exceed the charge value 32 recorded on the charge disk 22. The user types the charge value, for example, from the keyboard. In S404, the routine checks that the specified charge value is less than the charge value on the charge disk 22. If the specified charge value is greater than the charge value on the charge disk 22, the routine asks the user to retype the charge value.

In S405, the routine adds the specified charge value to the battery value, thus charging the battery value. In S406, the routine subtracts the specified charge value from the initial charge value and writes the resulting value on the charge disk 22 as a new charge value 32 . If the initial charge value 32 is exhausted, the routine writes the value of 0 on the charge disk 22 to virtually erase the charge value. The value of 0 prevents the charge disk 22 from being re-used. In S407, a record relating to the charge processing is added to the history table 40.

In the above embodiment, the user specifies an actual charge value. Instead of having the user specify a value, a pre-defined charge value may be added to the battery value at that time.

Fig. 10 shows another embodiment according to the present invention. In the embodiment described above, the battery value is charged using a recording medium. In this embodiment, the battery value is charged via a communication line 60. For the same components as those used in the above embodiment, the same numbers are assigned and their descriptions are omitted.

The user machine 10 in Fig.10 is connected to the

host machine 62 via the communication line 60. From this host machine 62, send data 64 shown in Fig. 11 are sent to the user machine 10 to charge the battery value.

In Fig. 11, address information 68 specifies the address of the user machine 10. Management information 70 is created by encrypting the serial number on the recording medium containing the managed software product 18. A charge value 72, a value to be added to the battery value as with the above embodiment, is an additional period of time in the execution time based method, and is an additional amount in the weight value based method.

Fig. 12 illustrates the system concept of this embodiment.

As described above, the user machine 10 is connected to the host machine 62 via the communication line 60. That is, this host machine 62 is connected to each of a number of user machines 10 for integrated operation management. This host machine 62 has a management information creation module 76, charge value issuance module 78, user registration table 80, and billing module 82. The management information creation module 76 creates the management information 70 shown in Fig. 11, and the charge value issuance module 78 issues a charge value 72 in response to a request from the user machine 10. As shown in Fig. 13, the user registration table 80 is composed primarily of the user ID column 80A, user name column 80B, and request charge value column 80C. The billing module 82 references the user registration table 80 to automatically issue a bill for a requested amount whenever a charge value is issued, or at some specified interval.

Next, referring to Fig. 12, the operation of this embodiment is explained with the use of Fig. 14. The operation of the user machine 10 is shown in the left side of Fig. 14, while that of the host machine 62 is shown on the right.

First, in S501 and S502, the user machine 10 is connected to the host machine 62 via a communication line. In S503, the user machine 10 generates a request for a charge value that will be sent to the host machine 62. In this case, the request contains at least the serial number of the CD-ROM containing the managed software product 18 and information on the charge value. In S504, the user machine sends the request to the host machine and, in S505, the host machine receives the request.

In S506, the host machine checks the user registration table 80. If the host machine finds, in S507, that the requesting user is registered in the host machine 62, the management information creation module 76 creates management information based on the serial number in S508, and the charge value issuance module 78 generates a charge value in response to the request from the user. In S509, the host machine 62 sends the management information and the charge value to the user machine 10 as the send data 64 shown in Fig. 11. In S510, the user machine 10 receives the send data 64. In S511 and S512, the user machine 10 and the host machine 62 are disconnected.

In S513, the operation management program 36 compares the serial number 74 with the management information 70 to check to see if the data received by the user machine 10 are valid. This prevents the user from illegally charging the battery value. If it is found in S514 that the send data are valid, the charge processing is performed in S515. This charge processing is the same as that in Fig. 9.

As shown in Fig. 12, this embodiment may also use the execution time based method or the weight value based method in order to manage the battery value.

Although the battery value is charged over a communication line such as a telephone line in the above embodiment, it may also be charged over a communication satellite (satellite line).

In the above embodiments, the operation management program 36 is included in the managed software product 18. Of course, an external program can manage the operation of the managed software product 18. Fig. 15 shows the concept of such an embodiment.

As shown in Fig. 15, the operation system (OS) 83 is located between the hardware 81 and each of application programs 84, 86, and 88. The operation management program 36 according to the present invention may be located between the operation system 83 and the application program 84.

Operation management program 36 therefore functions as an interface program. Messages are exchanged between the operation management program 36 and the application program 84 according to some specific rule. Messages are also exchanged between the operation management program 36 and the operation system 83 according to a specific rule.

To execute a management target function in this configuration, the operation management program 36 references the battery value when it receives an execution request from the application program 84. If the battery value is not zero, the operation management program 36 sends an instruction to the operation system 83 while simultaneously decrementing the battery value by a value corresponding to the function. If the battery value is zero, the operation management program 36 sends a message back to the application program 84, indicating that the instruction cannot be executed.

To execute a management non-target function, the operation management program 36 does not reference the battery value when it receives an execution request from the application program 84 but instead sends the instruction directly to the operation system 83.

The battery value is decremented as management target functions are executed. Charging the battery value allows the user to extend the usage period of the application program 84, which may be supplied separately from the application program 84.

In the above embodiments, one operation management program manages one operation management program. It is also possible for one operation manage-

ment program to manage several application programs.

Fig. 16 shows an application of the present invention. The system shown in Fig. 16 is composed of one host machine 90 and several user machines 92. Within each user machine 92 are a managed software product 18 and the operation management program 36, which, in turn, contains the counter 38 where the battery value to be decremented is stored. In other words, the operation of the managed software product 18 is controlled by the value stored in the counter 38. To execute the managed software product 18 in this system, it is necessary to insert a battery disk 96 into the user machine 92 and to move the battery value from the battery disk 96 into the counter 38. The battery value is decremented as the operation of the managed software product 18 proceeds. When the user finishes the managed software product 18, a sequence of operations are executed to move the current counter value from the counter 38 to the battery disk 96. This initializes the counter 38 to zero just as it was before the battery disk 96 was inserted.

The host machine 90 has several disk drives into which a battery disk 96 is inserted to read the battery value that was returned to the battery disk 96. This host machine 90 is also used to charge the battery value on the battery disk 96.

Integrated management of the battery values on several battery disks 96 through the host machine 90 brings a benefit of integrally managing several managed software products 18.

This type of system may be used, for example, in a school or a business where many computers are installed. With an individual carrying his or her own portable battery disk 96, it is possible to check and control the software usage amount of each person. In this case, either the "execution time based method" or the "weight value based method" may be used.

Claims

1. An operation management system for managing the operation of a managed software product, comprising:

   battery value management means for decrementing a battery value according to the operation amount of said managed software product;
   operation limit means for limiting the operation of said managed software product when said battery value has decremented to a specified limit value; and
   charge means for adding a charge value to the current battery value when the charge value is entered from external means.

2. An operation management system according to claim 1, wherein said battery value management means find the operation amount for each execution of a function owned by said managed software product and subtract a value corresponding to said operation amount from said battery value.

3. An operation management system according to claim 2, further comprising:

   function category determination means for determining if a function to which an execution instruction is issued is a management target function or a management non-target function, wherein said battery value management means decrement said battery value only when said management target function is executed.

4. An operation management system according to claim 3, wherein

   said battery value management means has a weight table containing pairs of said management target function and a weight value representing said operation amount thereof, and
   said battery value management means subtract a weight value corresponding to said management target function from said battery value when said management target function is executed.

5. An operation management system according to claim 3, wherein, when said management target function is executed, said battery value management means measure the execution time and subtracts the execution time from said battery value.

6. An operation management system according to claim 3, wherein said operation limit means prevent said management target function from being executed but allows said management non-target function to be executed when said battery value has reached a limit value.

7. An operation management system according to claim 3, wherein said managed software product has a data generation function and a data output function and wherein said function category determination means determine said data generation function as said management target function and determine said data output function as said management non-target function.

8. An operation management system according to claim 1, further comprising remainder warning means for issuing a remainder warning when said battery value has decremented to a warning value.

9. An operation management system according to claim 1, further comprising remainder display

means for displaying said battery value during execution of said managed software product.

10. An operation management system for managing the operation of a managed software product, comprising:

battery value management means for decrementing a battery value according to the operation amount of said managed software product;
operation limit means for limiting the operation of said managed software product when said battery value has decremented to a specified limit value;
read means for reading a charge value from a recording medium containing the charge value thereon; and
charge means for adding said charge value to the current battery value.

11. An operation management system according to claim 10, further comprising erase means for erasing the charge value from said recording medium after said charge value is added.

12. An operation management system according to claim 10, further comprising:

specification means for allowing a user to specify an actual charge value by which the current battery value is to be actually charged, the actual charge value not exceeding the charge value recorded on said recording medium; and
rewrite means for rewriting the charge value on said recording medium with a remainder value after said actual charge value is added to the current battery value.

13. An operation management system according to claim 10, in which said recording medium contains not only said charge value, but also the identification number of the recording medium and management information generated through encryption of the identification number, said operation management system further comprising:

validity determination means for comparing said identification number with said management information considering the condition of said encryption to determine the validity of said recording medium.

14. An operation management system comprising:

a managed machine containing a managed software product; and
a managing machine connected to said managed machine with a communication line, wherein
said managed machine comprises:
battery value management means for decrementing a battery value according to the operation amount of said managed software product;
operation limit means for limiting the operation of said managed software product when said battery value has decremented to a specified limit value;
charge value receive means for receiving a charge value from said managing machine; and
charge means for adding said charge value to the current battery value, and wherein
said managing machine comprises:
charge value send means for sending said charge value to said managed machine.

15. An operation management system according to claim 14, wherein said managed machine further comprises:

notification means for notifying said managing machine of the identification number of a portable recording medium initially containing said managed software product; and
validity determination means for comparing management information sent from said managing machine with said identification number to determine the validity of the recording medium; and wherein said managing machine further comprises:
management information creation means for creating said management information generated by encrypting said notified identification number and for sending the management information to said managed machine.

16. An operation management system comprising:

at least one managed machine containing a managed software product; and
a managing machine for managing the operation of said managed machine, wherein said managed machine comprises:
a counter containing a battery value changing according to the operation amount of said managed software product;
first charge means for reading a battery value from a portable recording medium to store the battery value into said counter; and
first return means for writing the current battery value on said recording medium, and wherein, said managing machine comprises:
second charge means for writing said battery value on said recording medium; and
second return means for reading said battery value from said recording medium.

17. An operation management method comprising:

a count value management step for changing a count value according to the operation amount of a managed software product;
an operation limit step for limiting the operation of said managed software product when said count value has reached a specified limit value; and
a charge step for charging the current count value or said limit value when a charge value is entered from external means.

18. A medium containing a management software product for managing the operation of a managed software product, wherein said managed software product and said management software product are executed on computers, said management software product comprising:

a module for changing a count value according to the operation amount of said managed software product;
a module for limiting the operation of said managed software product when said count value has reached a specified limit value; and
a module for charging the current count value or said limit value when a charge value is entered from external means.

19. A medium containing a charge value read by a management software product for use in managing the operation of a managed software product, wherein said managed software product and said management software product are executed on computers, said management software product comprising:

a module for changing a count value according to the operation amount of said managed software product;
a module for limiting the operation of said managed software product when said count value has reached a specified limit value; and
a module for charging the current count value or said limit value when said charge value is entered.

20. A computer system having an interface software product between an operation system and at least one application software product, wherein said interface software product comprises:

a module for changing a count value according to the operation amount of said application software product;
a module for limiting the operation of said application software product when said count value has reached a specified limit value; and
a module for charging the current count value or said limit value when a charge value is entered from external means.

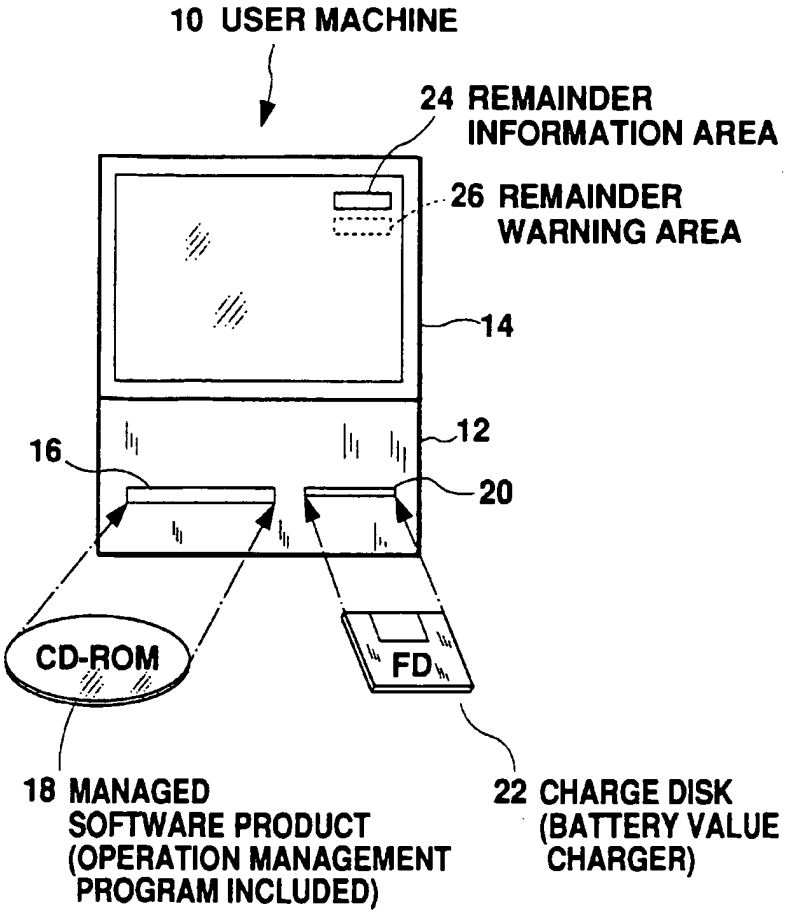


Fig. 1

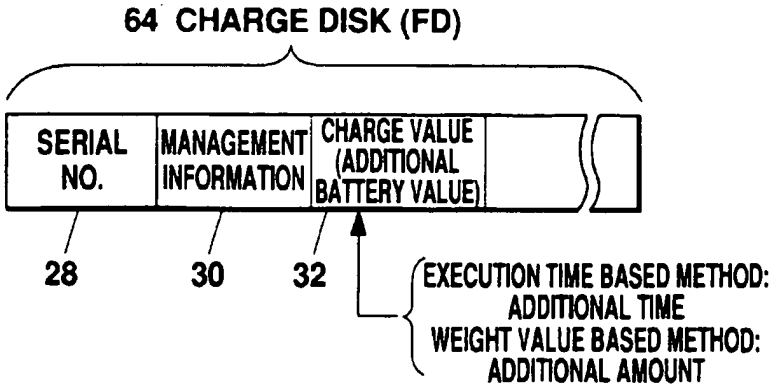


Fig. 2

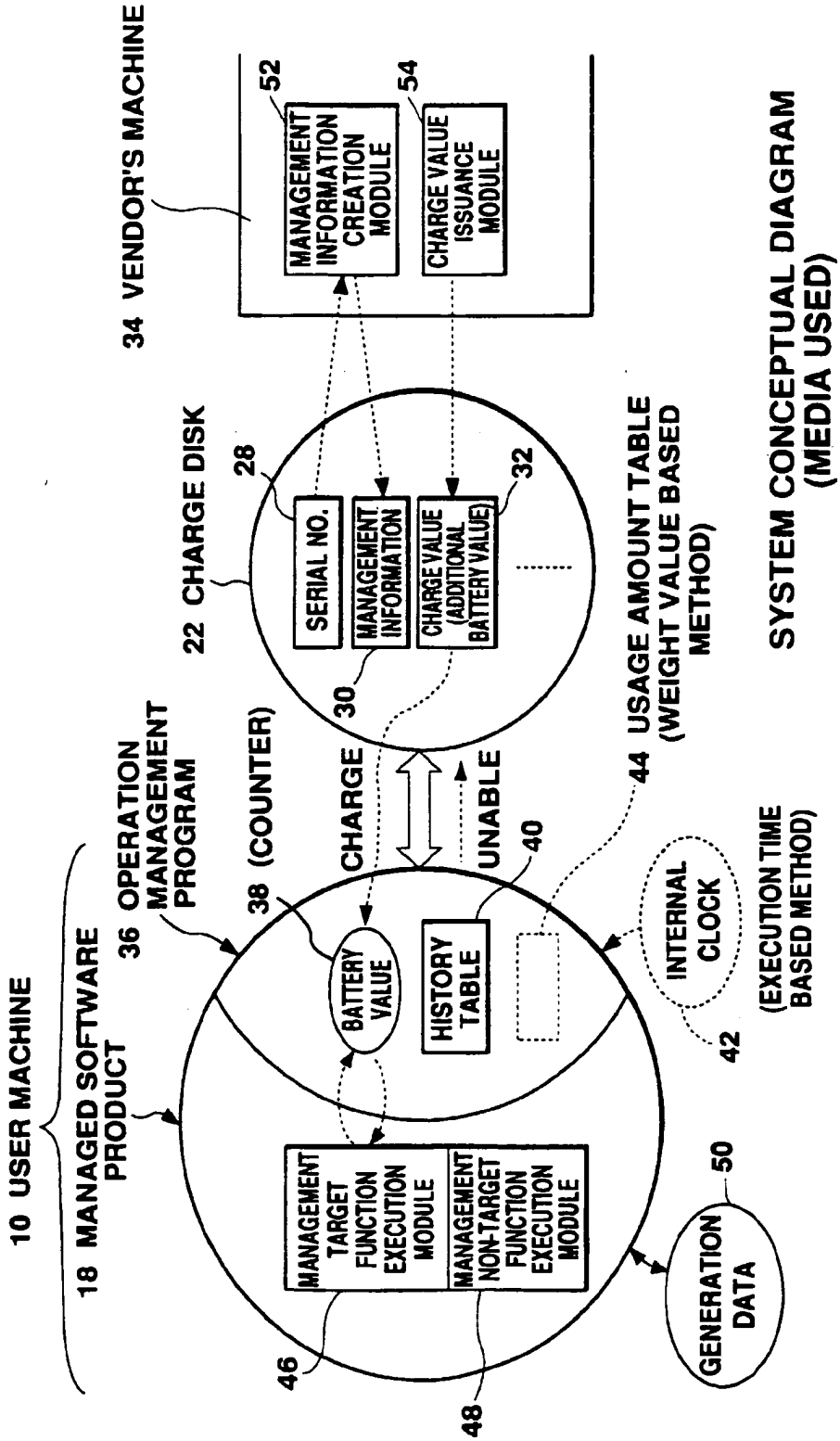


Fig. 3

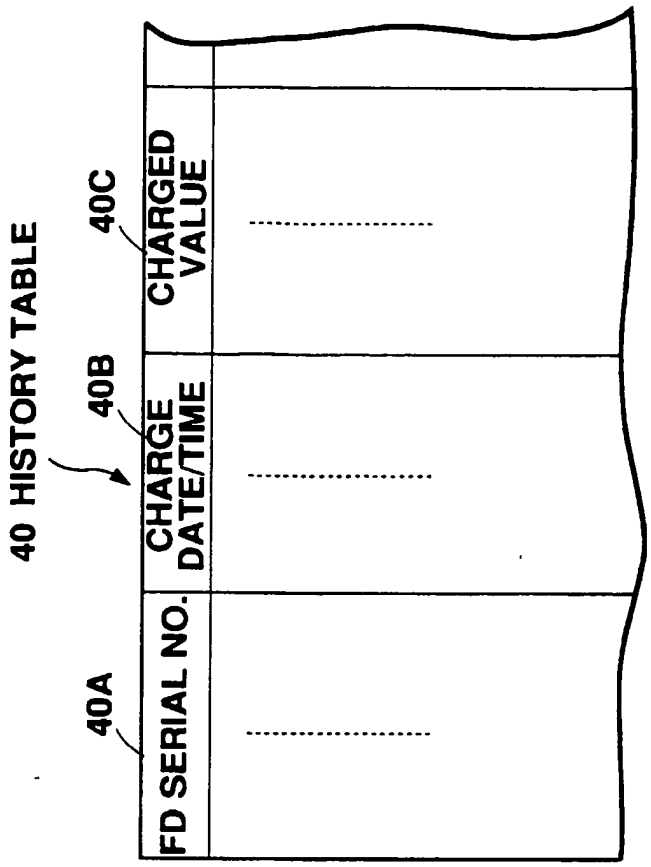


Fig. 5

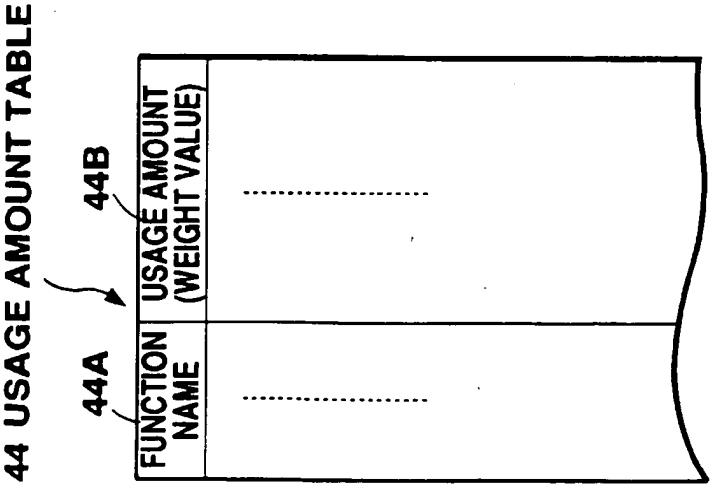


Fig. 4

MANAGEMENT TARGET FUNCTION EXECUTION
(EXECUTION TIME BASED METHOD)

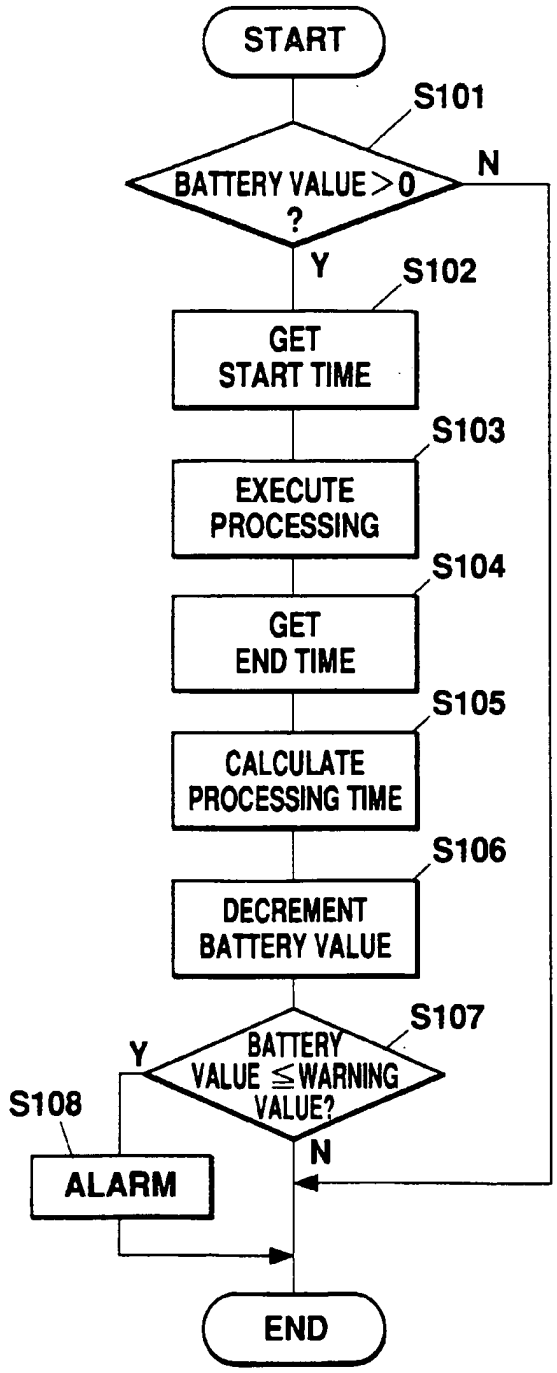


Fig. 6

MANAGEMENT TARGET FUNCTION EXECUTION
(WEIGHT VALUE BASED METHOD)

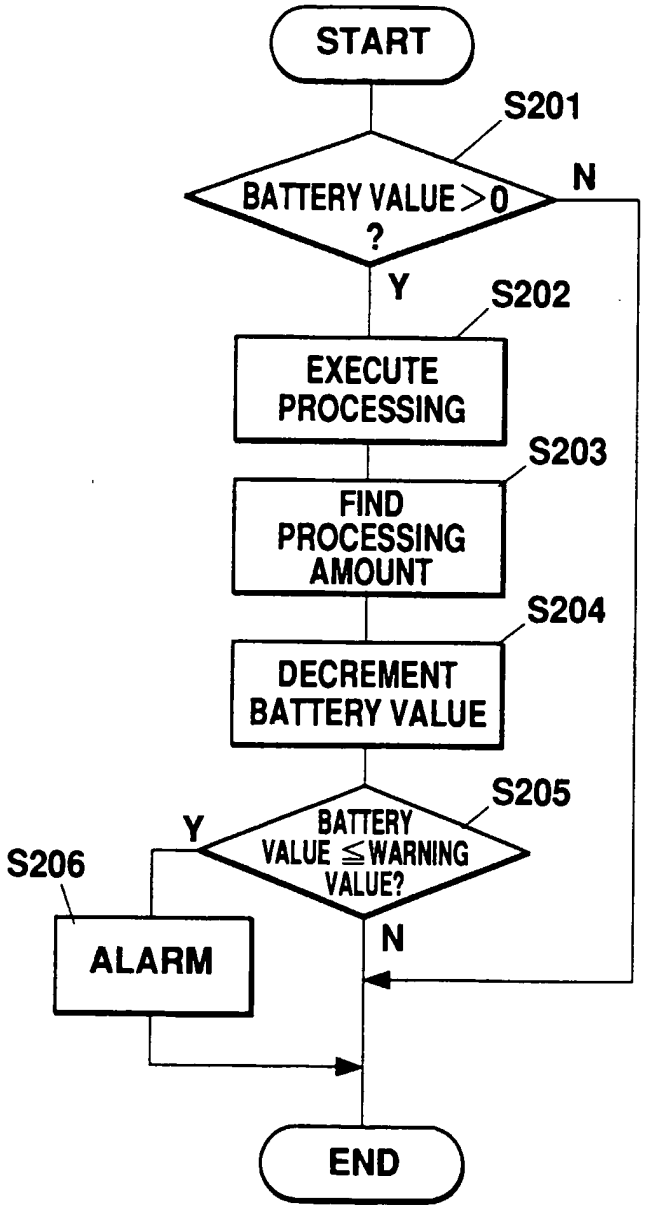


Fig. 7

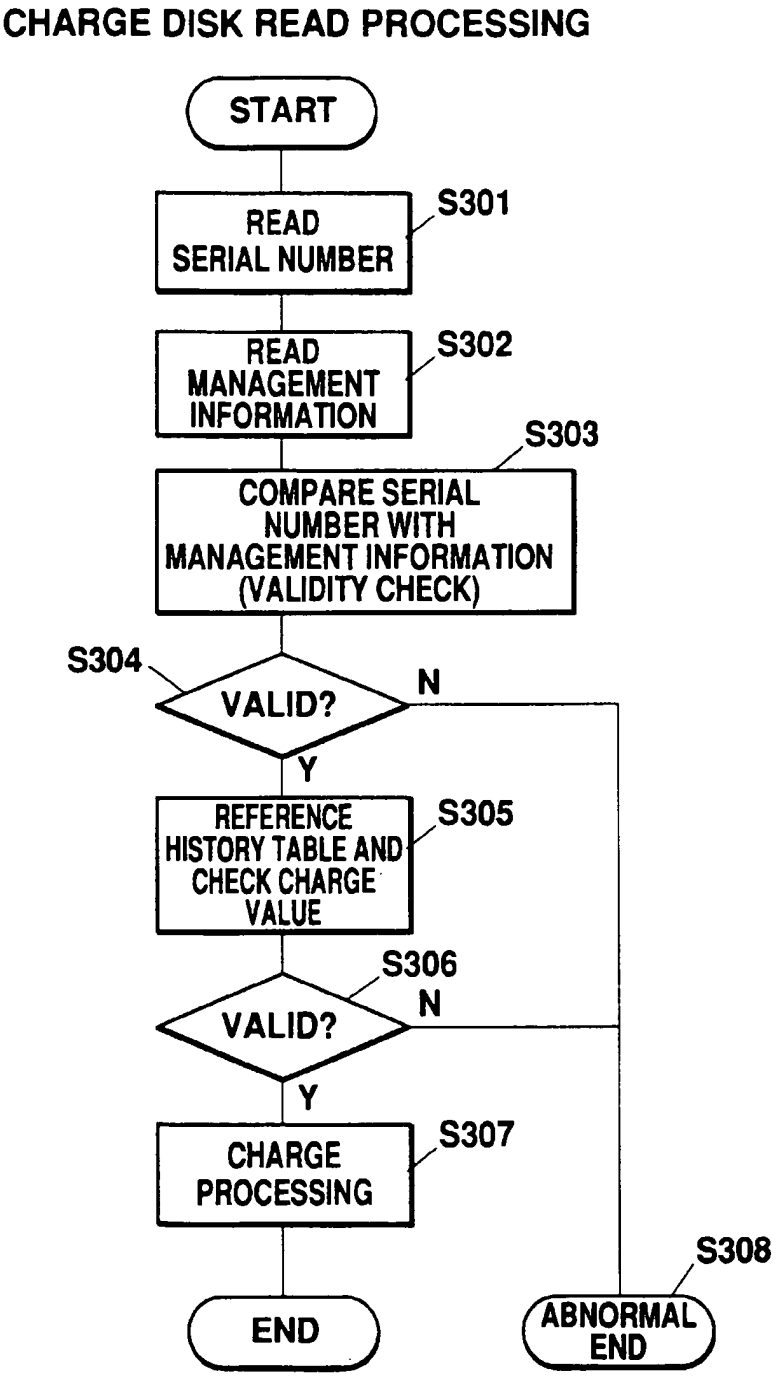


Fig. 8

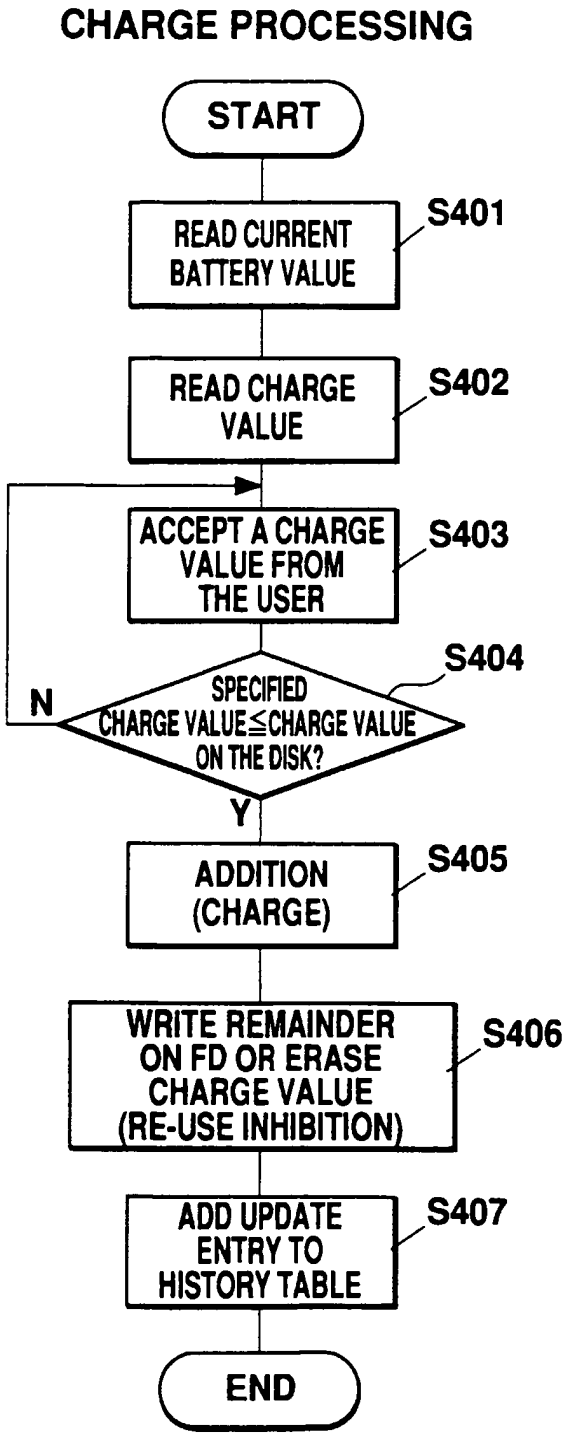


Fig. 9

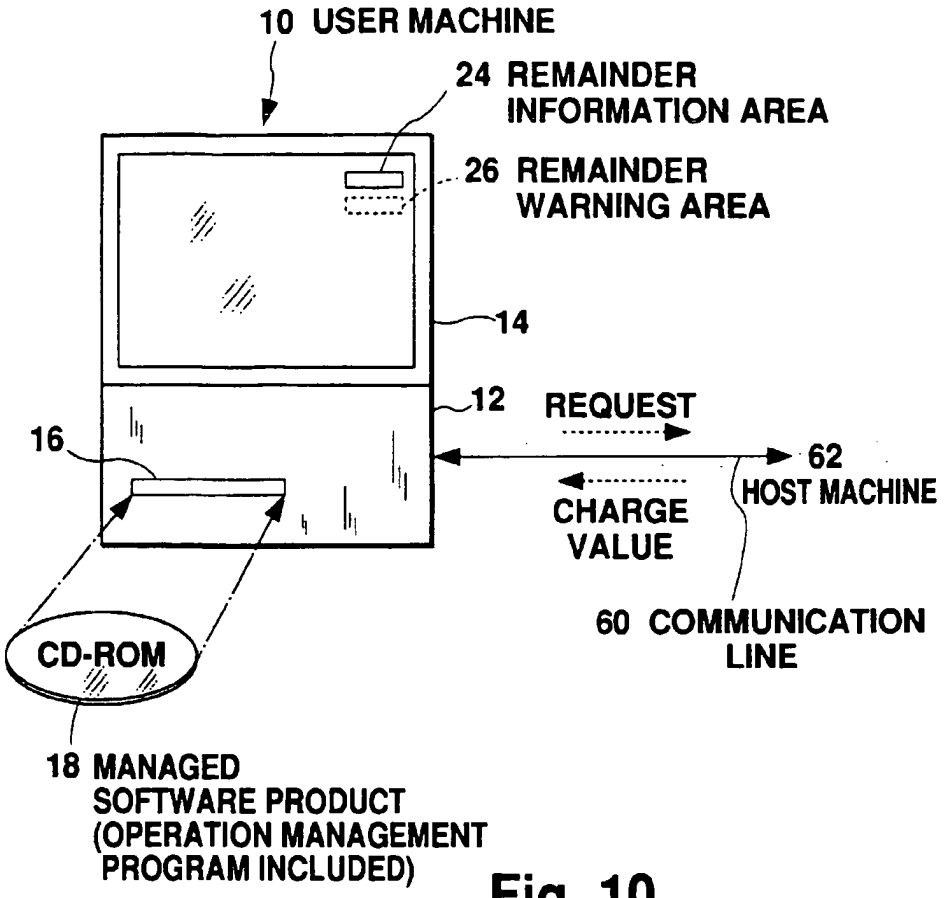


Fig. 10

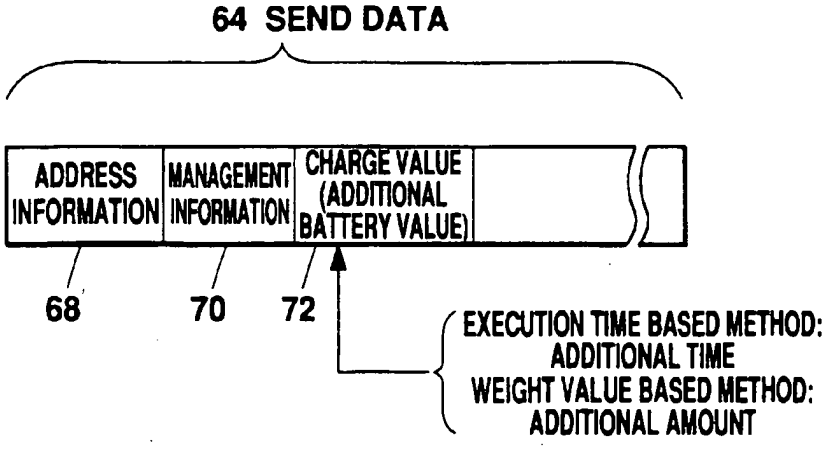


Fig. 11

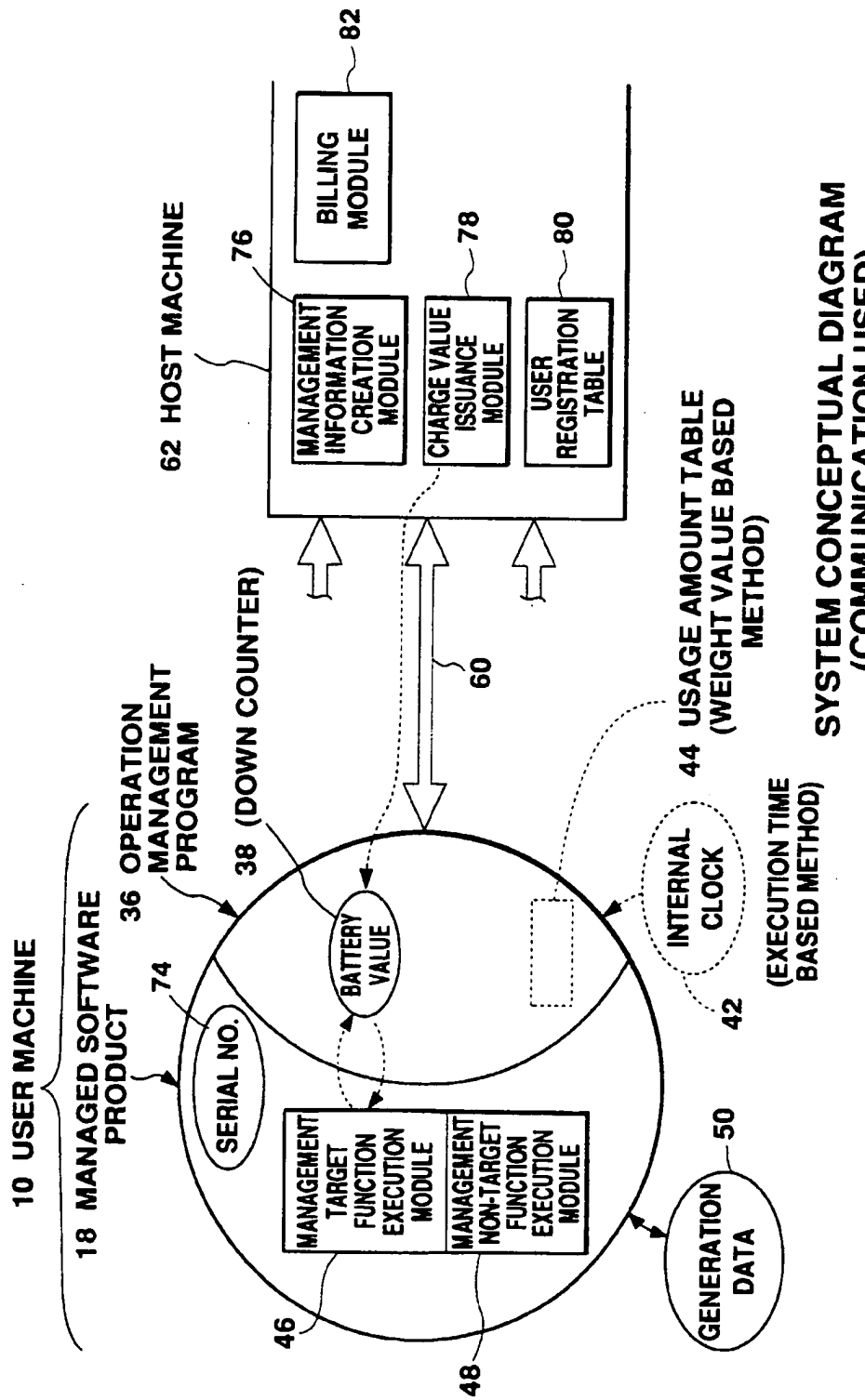


Fig. 12

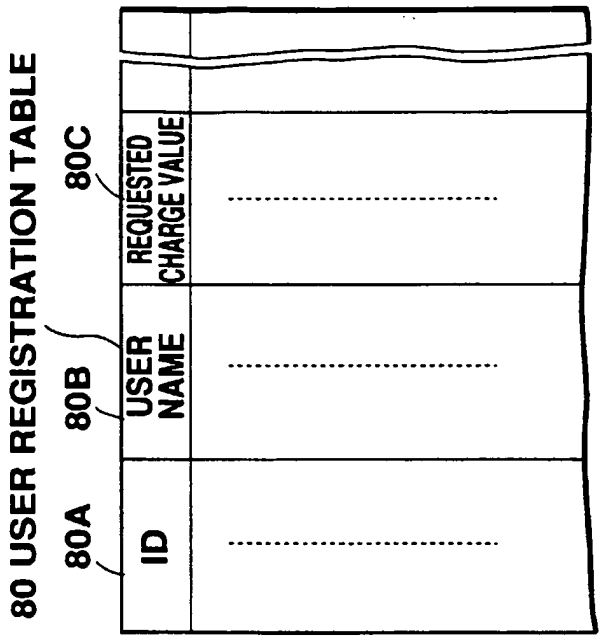


Fig. 13

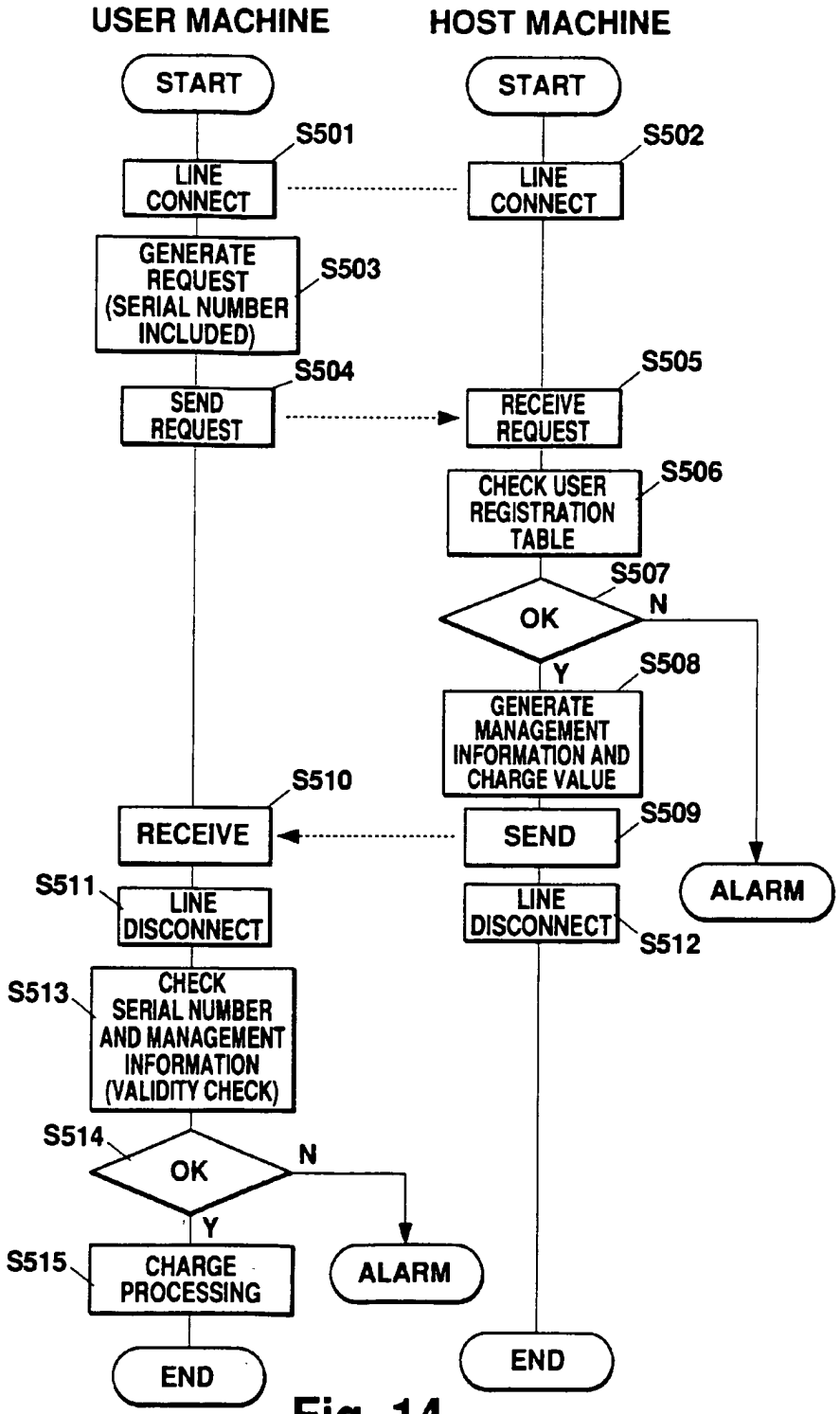


Fig. 14

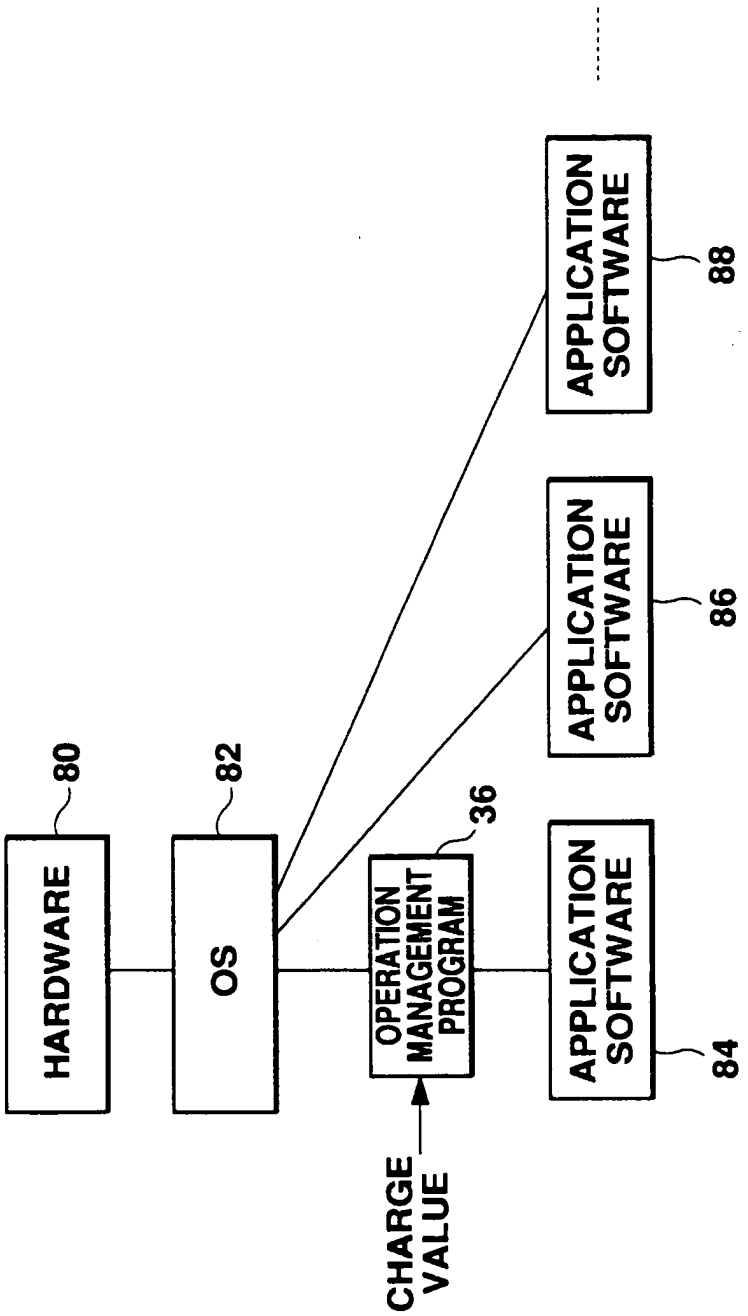


Fig. 15

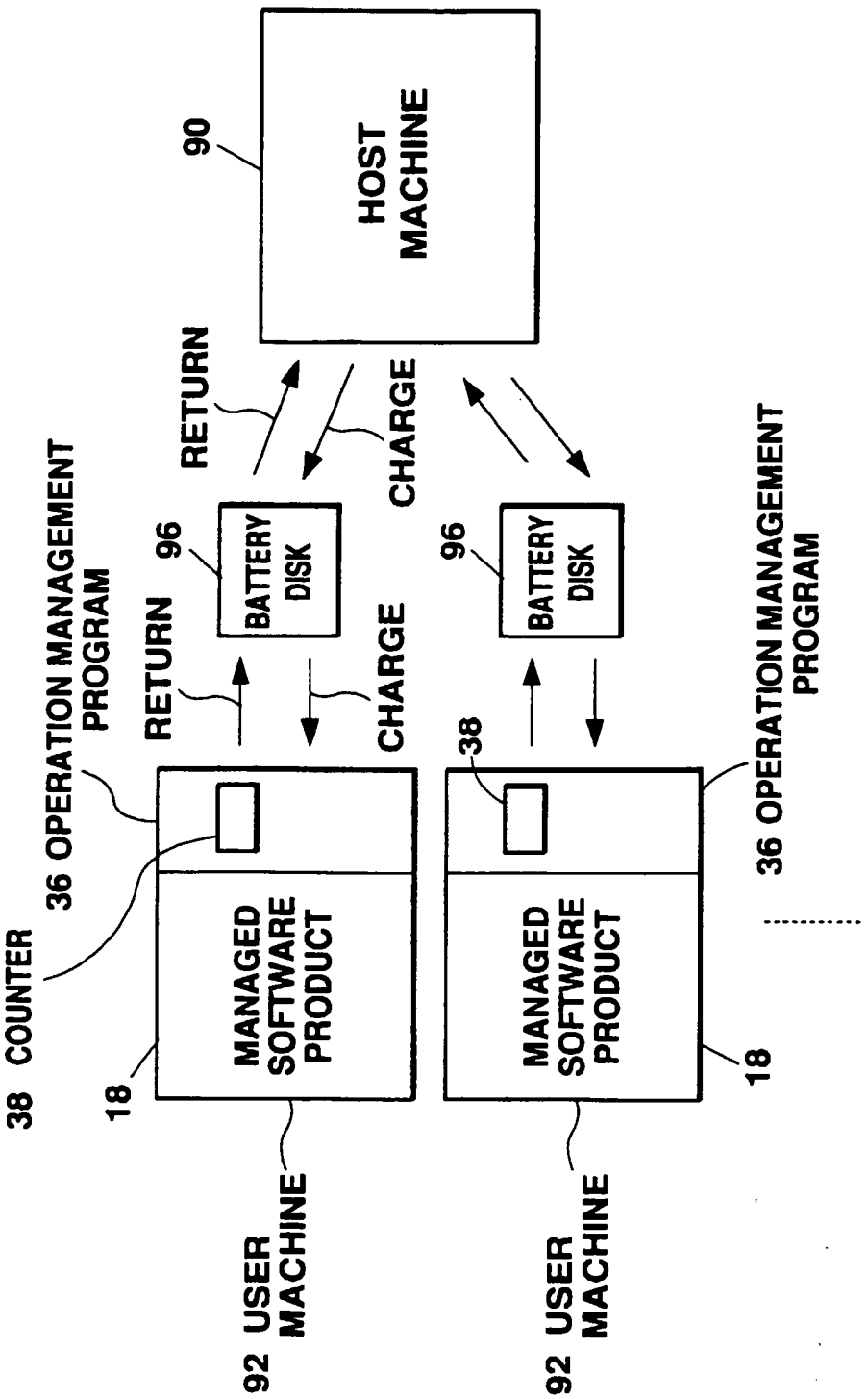


Fig. 16

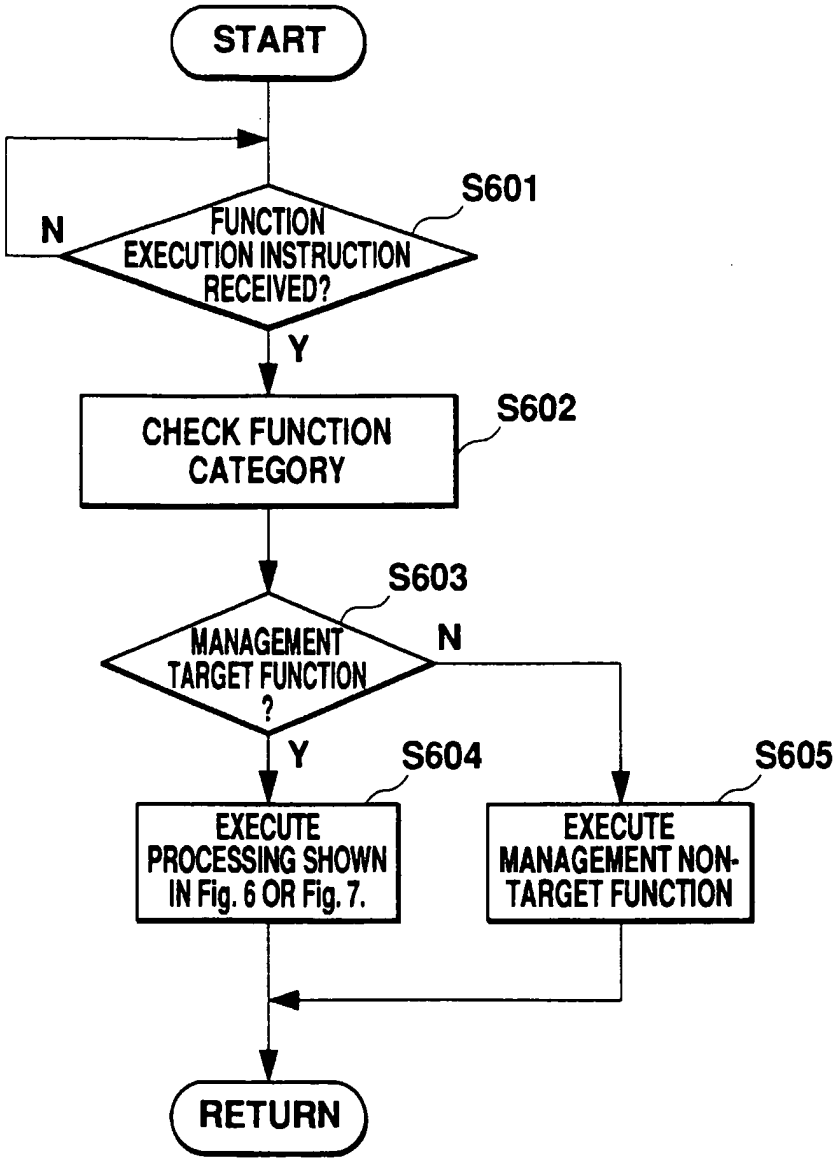


Fig. 17

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 840 194 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
06.05.1998 Bulletin 1998/19

(51) Int. Cl.[6]: G06F 1/00

(21) Application number: 97108754.9

(22) Date of filing: 02.06.1997

(84) Designated Contracting States:
DE FR GB
Designated Extension States:
AL LT LV RO SI

(30) Priority: 29.10.1996 JP 286345/96

(71) Applicant:
MATSUSHITA ELECTRIC INDUSTRIAL CO., LTD.
Kadoma-shi Osaka (JP)

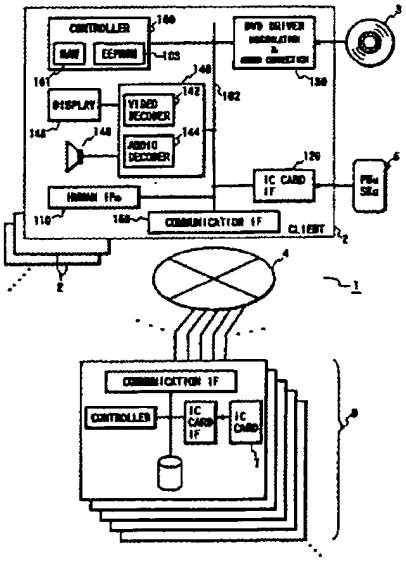
(72) Inventors:
- Uranaka, Sachiko
Tokyo (JP)
- Kiyono, Masaki
Kamakura-shi, Kanagawa-ken (JP)

(74) Representative:
Pellmann, Hans-Bernd, Dipl.-Ing. et al
Patentanwaltsbüro
Tiedtke-Bühling-Kinne & Partner
Bavariaring 4
80336 München (DE)

(54) System and method for controlling the use of a package of distributed application software

(57) A system for permitting only an authentic user to play a desired application contained in a distributed application package in one of predetermined operation, e.g., free play mode, charged mode, limit-attached play mode, etc. The system comprises a client for playing an application under the control of a server connected with the client through a communication network. The application package (the volume) includes a distribution descriptor which contains mode codes assigned to the volume and the applications of the volume. The data of distribution descriptor is decided and stored in the descriptor at the time of distribution of the volume. This feature makes the system flexible. There is also disclosed a system operatable without communicating with a server.

FIG. 1



EP 0 840 194 A2

Printed by Xerox (UK) Business Services
2.16.1/3.4

Description

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention generally relates to a security system and, more specifically, to a method and system for permitting an authentic user to use charged information which has been distributed via package or transmission media while charging and controlling the use of distributed charged information.

2. Description of the Prior Art

In order to use charged information such as music, movies, games, etc. provided by information providers that provide various programs of such charged information, a user has generally to take two steps. In the first step (or obtaining step), the user obtains a desired program from one of the information providers by purchasing a package media such as an FD (floppy disc), an optical disc (e.g., CD-ROM (compact disc read only memory) and DVD (digital versatile disc or video disc)), etc. on which the desired program is recorded (off-line distribution or obtaining) or by down loading the desired program from the server computer of an information provider through a predetermined procedure (on-line distribution or obtaining). In case of the on-line obtaining, the user may either play the program while obtaining it (i.e., the two steps are executed in parallel) or store the program while obtaining it in the first step and execute the program later as the second step (or using step). In case of the off-line obtaining, in the second step the user loads the obtained recording media into an appropriate device and directly plays (or executes) the program or once stores the program into the memory of the device and then plays the program.

Japanese Patent unexamined publication No. Hei7-295674 (1995) discloses a security system for use in the second or using step for a CD-ROM. In this system, the user can use encrypted information which is recorded together with a public key of a toll center (a center public key) on a CD-ROM by encrypting with the center public key and sending a code of desired program included in the information and a user-generated key to the information provider and by decrypting the information with an encryption key which has been encrypted with the user-generated key and sent by the information provider. However, the identity of the user is not verified, permitting a mala fide user who have obtained other person's CD-ROM to use it. Further, the center public key is pressed together with the encrypted information on the CD-ROM. This makes it difficult to change the center public key. Also, this causes different providers who probably want to use different center public keys to force the CD-ROM manufacturer to use different masters (or stampers) in pressing the CD-ROMs.

Japanese Patent unexamined publication No. Hei7-288519 (1995) discloses a security system for use in both the first and second steps. However, this system is only applicable to a system in which charged information is distributed on line.

Japanese Patent unexamined publication No. Hei8-54951 (1996) discloses a system in which the quantity of used software is monitored, and further software use by the user is impeded if the quantity exceeds a predetermined quantity. Since a dedicated hardware is necessary for impeding of software use, this system is only suitable for the use in a server in a on-line distribution system.

There is also a system for permitting a user to use, only for a trial period, software which has been distributed with data defining the trial period. In this system, a mala fide user may make the software reusable by installing the software again or setting the user system clock for a past time.

There are these and other programs in the art. It is an object of the invention to provide a system for permitting only an authentic user (a user who have legally obtained charged information either on line or off line from an information provider) to use the charged information without any limitation, charging for each time of its use, or within the tolerance of a use-limiting factor (e.g., the quantity used, the days elapsed since the day of its purchase or the current date) according to the type of the charged information.

SUMMARY OF THE INVENTION

According to the principles of the invention, it is assumed that charged information or an application package is distributed, either via package (or recording) media or via transmission media, together with at least control information such as a media title and a media code, etc. However, an illustrative embodiment will be described mainly in conjunction with charged information recorded on and distributed by means of the DVD.

For any type of charged information, charged information has been encrypted with a key and recorded on a DVD when obtained by a user. If distributed charged information to be played is of the limitlessly playable type, the charged information processing is achieved in the following way: the key is first obtained in a user public key-encrypted form from

the DVD on which the key has been recorded at the time of selling the DVD; the user public key-encrypted key is decrypted with a user secret key stored in a IC card into a decrypted key; and the encrypted charged information is decrypted with the decrypted key and consumed (that is, played or executed). The user-public key-encrypted key may be obtained on line from the server serving the client (device).

If distributed charged information to be played is of the usage-sensitive charging type, the user is charged for each time of using the information. In this case, prior to processing the charged information, the client double-encrypts and sends a user's credit card number to one of the to 11 servers of the provider of the information; the server adds an amount (e.g., play time or duration) used associated with the information to the value in a total amount (software meter) field in a volume data table, and sends the updated total amount value to the client; and the client displays the updated total amount. Then the client starts the charged information processing.

If distributed charged information to be played is of the limit-attached type, that is, the use of the information is to be limited by the tolerance of a certain limiting factor concerning the information consumption, then the client is permitted to consume the charged information only if the use-limiting factor is within the preset limit. In case of this type of charged information, prior to processing the charged information, the client sends the identifier (ID) code of a user specified application which is recorded on the DVD to the server; on receiving the ID code the sever tests if the use-limiting factor associated with the user specified application is within the preset limit; if not, then the server informs the client of the test result, and the client displays the test result; if the test was successful, then the server updates the meter (or integrated value) of the use-limiting factor and sends the updated value to the client; and in response to the reception of the updated value the client displays the updated value. Then the client starts the charged information processing.

BRIEF DESCRIPTION OF THE DRAWING

Further objects and advantages of the present invention will be apparent from the following description of the preferred embodiments of the invention as illustrated in the accompanying drawings. In the drawing,

FIG. 1 is a block diagram showing an arrangement of a system for permitting a user to use a distributed application package on the terms of use of the package with a higher security according to a first illustrative embodiment of the invention;

FIG. 2 is a diagram showing an exemplary structure of an application (or a charged information) package recorded on a DVD used in the inventive system;

FIGs. 3 and 4 are diagrams showing, in a detailed form, exemplary data structures of the volume descriptor 22 and the distribution descriptor 23, respectively;

FIG. 5 is a flow chart of a volume control program for playing the application(s) recorded on the DVD according to the principle of the invention;

FIG. 6A is a diagram showing an exemplary structure of a volume data table stored in a server shown in FIG. 1;

FIG. 6B is a diagram showing an exemplary structure of a application data table stored in a server 8;

FIG. 7 is a diagram showing a structure of a server table 75 stored in the EEPROM 103 of the client 2;

FIGs. 8A and 8B are flow charts of initial routines executed interactively by the client 2 and the server 8, respectively, at the beginning of the processes 650, 700 and 800.

FIG. 9 is a flow chart showing a procedure of a free play process shown as step 650 in FIG. 5, wherein connecting adjacent blocks by two flow lines indicates that each block is executed interactively by a client and an associated server;

FIGs. 10A and 10B are flow charts jointly showing a procedure formed of exemplary expected play time informing routines interactively executed;

FIGs. 11A and 11B are flow charts jointly showing a procedure formed of exemplary timed play and metered usage report routines interactively executed for playing an application while timing the duration and displaying a timed play duration after the play;

FIGs. 12A and 12B are flow charts jointly showing a procedure formed of exemplary timed application-play subroutines interactively executed for playing the application while timing the duration;

FIGs. 13A and 13B are flow charts jointly showing a procedure formed of alternative timed application-play subroutines interactively executed in which timing of play time is achieved with a timer in the client;

FIG. 14 is a flow chart of an exemplary application play subroutine called in steps 612 and 622 of FIGs. 12A and 13A, respectively, and executed by the controller 100;

FIG. 15 is a flow chart showing a procedure of a charged play process 700 shown as step 700 in FIG. 5,

FIGs. 16A and 16B are flow charts jointly showing a procedure formed of exemplary expected charge informing routines interactively executed;

FIGs. 17A and 17B are flow charts jointly showing a procedure formed of routines interactively executed in block 650 of FIG. 15;

FIGs. 18A and 18B are flow charts jointly showing a procedure formed of exemplary timed play and metered charge report routines interactively executed for playing an application while timing the duration and displaying a charge and a total amount of charges after the play;

FIG. 19 is a flow chart showing a procedure interactively executed by the client 2 and the server 8 in the operation block 800 of FIG. 5, wherein blocks connected with two flow lines indicates that operation of the blocks is done by the two elements 2 and 8;

FIGs. 20A and 20B are a key-encrypting key table and a user's public key table, respectively, stored in the server; and

FIG. 20C is a flow chart of a process for obtaining the application encrypting key K_v from the server 8;

FIG. 21 is a block diagram of an exemplary decipherer-built-in IC card IF according to the invention;

FIG. 22 is a diagram showing a K_v decoder used in place of the K_v decoder 126 of FIG. 21 in a system 1 using the cryptosystem of FIG. 20C;

FIG. 23 is a diagram for explaining the meanings of the terms-of-use (TOU) codes and the corresponding limit values;

FIG. 24 is a block diagram showing an arrangement of a system for playing a distributed application package on the terms of use of the package without communicating with any server according to a second illustrative embodiment of the invention;

FIG. 25 is a flow chart schematically showing an exemplary control program executed by the controller 100a shown in FIG. 24;

FIGs. 26 and 27 are flow charts showing an operation of a free play mode shown in step 650a of FIG. 25 in a detailed form and a further detailed form, respectively; and

FIG. 28 is a flow chart showing an operation of a limit-attached play mode shown in step 800a of FIG. 25.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

For the sake of better understanding of the following description, it will be useful to define some terms to be used.

Charged information provided by an information provider may be distributed off-line (in off-line distribution) or on-line (in on-line distribution). In off-line distribution, the charged information is recorded on package media or recording media, and distributed through the sales network of the provider, that is, sold at stores in the sales network. The package media include all sorts of portable recording media such as various types of magnetic discs, a variety of optical memory discs (e.g., CD, CD-ROM, DVD), and magnetic tapes and cartridges. In on-line distribution, the charged information is transmitted via transmission media from the servers at the service points of the provider and the distributors aligned with the provider to the client device (e.g., PC (personal computer)) of the user who requested the charged information, and stored in a recording media of the client (device). The transmission media include any telecommunication channels which permit data communication between the servers and the client device. The package media and the transmission media are hereinafter referred to en bloc as "distribution media".

The charged information may be any type of software such as music, movies, games, etc. which are each referred to as an "application" without discrimination. The distribution unit of charged information is referred to as a "charged information package" or an "application package". There may be included one or more applications in an application package.

The present invention relates to a system for permitting a user to use a distributed application package on the terms of use of the package with a higher security.

Embodiment I

For the purpose of simplicity, a first illustrative embodiment will be described in which package media, among other things, DVDs are used as distribution media.

FIG. 1 is a block diagram showing an arrangement of a system for permitting a user to use the application(s) recorded on a DVD on the terms of use of the DVD with a higher security according to the first illustrative embodiment of the invention. In FIG. 1, the system 1 comprises a client or DVD player 2 which plays a DVD 3, a telecommunication network 4, and a server 8 at a toll center of the provider 6 which provides the application package of the DVD 3.

FIG. 2 is a diagram showing an exemplary structure of an application (or a charged information) package 20 recorded on the DVD 3 used in the inventive system 1. In FIG. 2, the application package 20 comprises at least one application 21, a volume (or package) descriptor 22 comprising data concerning the application package 20, and a distribution descriptor 23 comprising data which is determined mainly at the time of, e.g., distribution or sales after the pressing of the DVD 3. (The volume descriptor 22 and the distribution descriptor 23 constitutes the volume control data of the volume 20.) In this embodiment it is assumed that a volume (or package) control program which controls the use of the application package 20 in cooperation with the server 8 is included in and distributed with the application package

20. Thus, the application package 20 further comprises the package control program 24 suited for the terms of use of the package 20. The application(s) 21, the volume descriptor 22 and the package (or volume) control program 24 are recorded in the data area of the DVD 3 at the time of manufacturing the DVD 3, while the distribution descriptor 23 is recorded in the burst cutting area at the time of, e.g., sales of the DVD 3.

FIGs. 3 and 4 are diagrams showing, in a detailed form, exemplary data structures of the volume descriptor 22 and the distribution descriptor 23, respectively. In FIG. 3, the volume descriptor 22 at least contains a volume identifier (VID_v) 25 which the title of the application package 20 is probably used for and which is the same as the application identifier if the package or volume 20 contains only one application; a provider identifier 26; volume creation date and time 27 which may be used for the base point by which volume expiration data and time as described later is determined; and volume effective date and time 28 indicative of date and time until which the volume 20 is available. If the volume 20 contains more than one applications, the volume descriptor 22 further contains application identifiers (AID_a's) 29.

In FIG. 4, the distribution descriptor 23 comprises the fields of: a volume issue number (NO_{v-i}) 30 which contains a serial number given to each of the distributed application packages of an identical volume identifier (volume ID or title) VID_v in the order of distribution; a server public key (PK_s) 31 the data of which is given by the server 6 at a toll center of the provider 6; a PK_u (user-public-key)-encrypted application-encrypting key (K_v) 32; and sales date and time 33. The key PK_s 31 field contains a key which has been used in encrypting each application 21 in the package 20 and which has been encrypted with a user public key (PK_u) of the user who has legally obtained the package 20. Appropriate data are recorded in all of the fields 30 through 34 at the time of distribution of the package 20, i.e., at the time of sales of the DVD 3 in this embodiment.

The distribution descriptor 23 further comprises the field 34 of terms-of-use code (mode code) plus limit value for the volume (the volume limit value field) and, for each of the application IDs 29, the fields 35 of terms-of-use code plus limit value for the application ID 29 (application limit value field). If terms of use are set only to the volume 20, there is no need of the field 35. If terms of use are set to each application, the field is empty.

FIG. 23 is a diagram for explaining the meanings of the terms-of-use (TOU) codes and the corresponding limit values. In FIG. 23, the terms-of-use code may be, e.g., one byte in length. The higher digit (X) of the TOU code indicates the target to which the terms of use is applied as shown in table 36. That is, higher digits of 0, 1, 2,... indicate that the TOU codes beginning with those digits are for the entire volume, application 1, application 2 and so on. The lower digit (Y) of the above mentioned terms-of-use code indicates the terms of use of the package 20 or the application 21 to which the code is set, and is directly followed by a corresponding limit value as shown in table 37 of FIG. 23. Specifically, the terms-of-use code (or TOU code)of 00H means, for example, that the volume 20 is usable freely after distribution. The value '31H' means, for example, that the application 3 to which the TOU code is set can be used by paying per unit of play duration. The lower digit of 2H or more means that the volume 20 or the application to which the TOU code is set can be used freely until the corresponding limit value are reached, which disables further use. As seen from the table, the use-limiting factors determined by the TOU codes whose lower digits are 2H to 5H are the current date and time, the expiration date and time, the amount of used period, and the access count, respectively.

Since the data of the distribution descriptor 23 can be set as described above, this provides both the providers and the users with more flexibility than conventional system can provide.

Again in FIG. 1, the DVD player 2 comprises a controller 100 for controlling the entire DVD player 2; data bus 102 connected with the not-shown CPU (central processing unit), not-shown ROM (read-only memory), RAM (random access memory) 101, and EEPROM (electrically erasable programmable ROM) 103 included in the controller 100; human interfaces (IFs) 110 including input devices such as a keyboard, a voice recognition device, a mouse, a remote controller, etc. ; an IC card interface (IF) 120 for connecting the bus 102 with the ROM (not shown) in a IC card 5; a DVD driver 130 for reading out the data recorded on the DVD 3 and for demodulating and error-correcting the read data; a video and audio output IF 140 for receiving a MPEG 2 bit stream and outputting a video and audio output signals; a display device 146; a loudspeaker 148, and a communication IF 150 for communicating through the public telecommunication network 4. The IC card 5 stores a user's password PW_u and a user's secret key SK_u which corresponds to the user's public key PK_u mentioned in conjunction with the PK_u-encrypted AP-encrypting key (K_v) contained in the field 32 of the distribution descriptor 23 recorded in the burst cutting area of the DVD 3. The video and audio output IF 140 includes a MPEG 2 video decoder 142 and a MPEG 2 audio decoder 144.

As for obtaining the DVD 3, there may be some ways. If one is to buy a DVD 3, e.g., at some book store or through mail order, he or she has to have the PK_u-encrypted version of an application-encrypting key (K_v) recorded in the burst cutting area of the desired DVD 3 by notifying his or her public key PK_u which corresponds to his or her secret key SK_u stored in the IC card 5. If one is a member of a DVD distribution service, he or she can obtain a DVD with a PK_u-encrypted AP-encrypting key recorded without notifying the PK_u each time of obtaining because he or she must have notified the PK_u when he or she applied for the service.

In operation, the user first sets a desired DVD 3 in the DVD driver 130 of the DVD player 2, and issues a start command to the DVD player 2 through an appropriate human IF 110. In response to a receipt of the start command, the

controller 100 reads the volume control program 24 from the data area of the DVD 3 through the DVD driver 130 while loading the read program 24 into the RAM 101 of the controller 100, and then executes the volume control program 24.

FIG. 5 is a flow chart of the volume control program 24 for playing the application(s) 21 recorded on the DVD 3 according to the principle of the invention. In FIG. 5, the controller 100 first checks the AID1 field to see if the volume 20 contains a single application in step 500. If not, then the controller 100 displays the application IDs in the field 29 and prompts the user to select a desired one of the applications in step 502, and waits for the selection in step 504. If any application is selected in step 504, the controller 100 registers the application ID of the application as the application to be played in step 506 and proceeds to step 508 to check the field 35 of the terms-of-use (TOU) code plus limit value for the selected application to see if the field is empty. If so, the controller 100 proceeds to step 510 to read the volume limit field 34.

On the other hand, if the test result is YES in step 500, then the controller 100 registers the volume ID as the application to be played in step 512, and reads the volume limit value 34 in step 510.

If the step 510 is completed or the test result of step 508 is NO, then the controller 100 checks the terms-of-use (TOU) code to see if the lower digit of the TOU code is 0 in step 514. If so, then the controller 100 plays an application free of charge in step 650, and otherwise makes another check to see if the lower digit of the TOU code is 1 in step 516. If so, the controller 100 plays an application in a usage-sensitive charging in step 700, and otherwise (if the lower digit of the TOU code is 2 or more) play an application only when the software meter of a use-limiting factor is under a preset value in step 800. On completing any of the steps or processes 650 through 800, the controller 100 ends the program 24. Thus, the DVD player 2 plays a program specified by the user according to the terms of use determined by the TOU code which has been set to either the application package or the specified application.

The processes 650, 700 and 800 are executed interactively with an associated server 8. The servers 8 need various data for executing these processes, and store such data in the form of tables.

FIG. 6A is a diagram showing an exemplary structure of a volume data table stored in a server 8. In FIG. 6A, Each of the records of the volume data table 60 comprises volume ID (VID_v) and issue No. (NO_{v-i}) fields. The combination of VID_v and NO_{v-i} serves as the user ID of the user of the application package 20 or the DVD 3. For this reason, the table 60 has, for the members or subscribers of DVD distribution service or the like, personal data fields which contains, for example, a member ID, a name, an address, etc. Each record further comprises a volume minute meter field ($VM\text{-}METER_{v-i}$) containing a software meter of play duration in minute which is attached to (or associated with) the volume 20; a volume charge meter ($VC\text{-}METER_{v-i}$) containing a software charge meter which is attached to the volume 20; a limit value (LV_{v-i}) containing a limit value associated with the TOU code (e.g., the effective date and time, the allowable expiration date and time, the allowable access, etc.); a limit value meter ($LV\text{-}METER_{v-i}$); an application ID (AID_{v-i-a}) field containing the title of the application; an application minute meter ($AM\text{-}METER_{v-i-a}$) field containing a software meter of play duration in minute which is attached to the application of AID_{v-i-a}; an application charge meter ($AC\text{-}METER_{v-i-a}$) field for a software meter of play duration in minute which is attached to the application of AID_{v-i-a}; a limit value (LV_{v-i-a}) containing a limit value associated with the TOU code; and a limit value meter ($LV\text{-}METER_{v-i}$).

FIG. 6B is a diagram showing an exemplary structure of a application data table stored in a server 8. In FIG. 6B, the application data table 70 comprises the fields of, for example, an application code ($ACODE_n$), an application title (AID_n), a duration (D), a rate-per-access (RATE/ACCESS), an access count, a minute meter, etc. The duration is a period of time what it takes to play the application. The rate per access is a charge for a play of the whole application, which is used for informing the user of an expected play duration prior to a play. The rate per unit time is a charge for a unit time of play, which is used for the calculation of a charge for an actually timed play duration. The access count and minute meter fields contains the number of accesses to the application and a total amount of play time, which are not necessary for the present invention but will be used in statistical calculations for the analysis of, e.g., the tastes.

FIG. 7 is a diagram showing a structure of a server table 75 stored in the EEPROM 103 of the client 2. In FIG. 7, the fields of the table 75 comprises a server public key (PK_s), a server ID (SID_s), a server network address ($SADD_s$), etc. this table 75 is used for associating the sever public key (PK_s) contained in the distribution descriptor 23 recorded in the burst cutting area of the DVD with the ID and the network address.

Play an Application Free of Charge

The initial routines of the processes 650, 700 and 800 are the same.

FIGs. 8A and 8B are flow charts of initial routines 80a and 80b which are executed interactively by the client 2 and the server 8, respectively, at the beginning of the processes 650, 700 and 800. In FIG. 8, the controller 100 of the client or the DVD 2, in step 82, sends a service request with the network address $CADD_c$ of the client or DVD 2, the TOU code plus limit value, the volume ID (VID_v), the issue number (NO_{v-i}), the application ID (AID_{v-i-a}), and other data to the associated server 8 the ID of which is SID_s (SID_s is obtained from the table 75 in FIG. 7 by using the public key recorded on the DVD 3), and in step 92 waits for a response from the server (SID_s) 8. If there is a response from the server (SID_s), the client 2 proceeds to the next step through a circle with "A" therein.

On the other hand, in FIG. 8B, the server 8 of SID_s receives the message from the client 2, that is, the service request and the accompanying data and stores date in a predetermined location for subsequent use in step 84. Then, the server 8 searches the table 60 for a record which contains VID_v and NO_{v-i} in the volume ID and issue No. fields thereof, respectively in step 86. If the search is unsuccessful, then the server 8 adds the record for VID_v and NO_{v-i} and fills relevant fields with AID_{v-i-a} and a limit value, if any, in the table 60 in step 88, and proceeds to step 90. Also, if the search in step 86 is successful, the server 9 proceeds to step 90, where the server 8 selects a routine to execute next according to the value of the TOU code and enters the selected routine through a circle with "B" therein. In this case, if the TOU code = x0H (x: an arbitrary HEX number, the letter H in the last position indicates that the preceding number is in hexadecimal), then a routine for playing an application free of charge is selected. If the TOU code = x1H, then a routine for playing an application in usage-sensitive charging is selected. If the TOU code ≥ x2H, then a routine is selected which plays an application only if the software meter of a use-limiting factor is under a preset value.

FIG. 9 is a flow chart showing a procedure of a free play process shown as step 650 in FIG. 5, wherein connecting adjacent blocks by two flow lines indicates that each block is executed interactively by a client of $CADD_c$ and an associated server SID_s as shown in detail later. If the TOU code is 0 in step 514 of FIG. 5, then the server ($CADD_c$) enters the free play process 650 as shown in FIG. 9, and the client and the server (SID_s) execute the initial routine 80 in block 660. In block 670, they executes an expected play time informing routine, that is, displays an expected play time before playing an specified application. In block 680, they execute an application play and metered play time report routine. Since the routine 80 has been detailed in FIG. 8, the expected play time informing routine and the application play and metered play time report routine will be detailed in the following.

FIGs. 10A and 10B are flow charts jointly showing a procedure formed of exemplary expected play time informing routines 97a and 97b interactively executed by the client 2 and the associated server 8, respectively. In FIG. 10B, the server 8 retrieves the duration (D_n) of the application of AID_{v-i-a} from the table 70 in a well known manner in step 91. In the next step 92, the server 8 calculates an expected total amount of play time according to the value of the TOU code. Specifically, if the TOU code is 0xH, then the client adds the duration (D_n) and the value of the $VM\text{-}METER_{v-i}$ field of the record identified by VID_v and NO_{v-i} in the table 60. If the TOU code is axH (a: the application number of the specified application in the volume), then the client adds the duration (D_n) and the value of the $AM\text{-}METER_{v-i-a}$ field of the record identified by VID_v, NO_{v-i}, and AID_{v-i-a} in the table 60. Then the server 8 sends the result to the client whose network address is $CADD_c$ in step 93, and ends the process.

On the other hand in FIG. 10A, the client 2 receives the incoming message or the value of the updated meter in step 94. In the next step 95, the value is displayed as the total amount of usage. Then the client 2 ends the process.

In updating a relevant meter, a predetermined value of duration has been used in the just described routines of FIG. 10 (a preset value metering system). This arrangement is suited mainly for such applications as it takes a constant time to play, and will not cause a problem unless the user discontinues the play. From this point of view, it is preferable to actually measure the playing time in metering (a timed value metering system). However, it is also noted that the preset value metering system is useful in informing the user of expected play time prior to an actual playing.

FIGs. 11A and 11B are flow charts jointly showing a procedure formed of exemplary timed play and metered usage report routines 675a and 675b interactively executed by the client and the server, respectively, for playing an application while timing the duration and displaying a timed play duration after the play. In the routine 675, the client and the server call a timed application-play subroutine for playing the application while timing the duration (play time) in step 200.

Then the server 8 proceeds to step 210, where the client updates a relevant meter according to the TOU code in the same manner as in step 92 of FIG. 10B. Specifically, if the TOU code is 0xH, then the play time is added to the value of the $VM\text{-}METER_{v-i}$ field of the record identified by VID_v and NO_{v-i} in the table 60. If the TOU code is axH (a: the application number of the specified application in the volume), then the play time is added to the value of the $AM\text{-}METER_{v-i-a}$ field of the record identified by VID_v, NO_{v-i}, and AID_{v-i-a} in the table 60. Then the server 8 sends the play time and the value of the updated meter (*i.e.*, the total amount of play time) to the client whose network address is $CADD_c$ in step 212, and ends the process.

On the other hand, the client 2, after step 200, make a test to see if there is a response from the server of SID_s in step 214. This step is repeated until the client 2 receives a call from the server 8, when the client 2 receives the incoming message or the value of the updated meter in step 216. In the next step 218, the client 2 displays the play time and the total amount of play time, and then ends the routine 675.

FIGs. 12A and 12B are flow charts jointly showing a procedure formed of exemplary timed application-play subroutines 205a and 205b executed by the client 2 and the server 8, respectively, for playing the application while timing the duration. The server 8 of SID_s waits for a notice in step 611 to see if the client has started playing the application. On the other hand, the client 2 of $CADD_c$ informs the server of a start of play in step 610 and immediately call an application play subroutine in step 612. This, causes the server 8 to start a timer in step 613, and waits for a notice of a stop of play from the client 2 in step 615. On completing the step 612, the client informs the server 8 of the stop of play in step 614. In response to this notice, the server 8 stops and reads the timer as the play time in step 617. After steps 614 and 617, the client and the server return.

Though the above described arrangement has used a timer of the server, it may be possible to use a timer of the client.

FIGs. 13A and 13B are flow charts jointly showing a procedure formed of alternative timed application-play subroutines 205ac and 205bc interactively executed by the client 2 and the server 8, respectively, in which timing of play time is achieved with a timer in the client. In the alterative subroutine 205a, the client 2 starts a timer in step 620, calls an application play routine in step 622, stops the timer in step 624, sends the play time to the server 8 in step 626, and then returns. On the other hand, the server 8, on entering the subroutine 295b, waits for a call from the client of $CADD_c$ in step 621. If there is a call from the client 2, then the server 8 receives the play time in step 623 and then returns.

However, the arrangement of FIG. 13 has a possibility of permitting a mala fide user to manipulate the timer of the client 2. From this point of view, the arrangement shown in FIG. 12 is preferable to that of FIG. 13.

FIG. 14 is a flow chart of an exemplary application play subroutine called in steps 612 and 622 of FIGs. 12A and 13A, respectively, and executed by the controller 100.

Prior to the description of the flow chart, we define some notation concerning encryption and decryption. If encrypting X with a key EK according to an encrypting algorithm e yields Y, then it is expressed as:

$$e(EK, X) = Y.$$

Similarly, if decrypting Y with a key DK according to a decrypting algorithm d yields Z, then it is expressed as:

$$d(DK, Y) = Z.$$

Assuming that the algorithms e and d and the keys EK and DK correspond each other, that is, $d(DK, Y) = X$, it follows that

$$d(DK, e(EK, X)) = X.$$

Returning now to FIG. 14, the controller 100 read the PK_u-encrypted application-encrypting (AP-encrypting) key (K_v) or $e1(PK_u, K_v)$ from the filed 32 of the distribution descriptor 23 of the DVD in step 602. Here,

$$v = 1, 2, \ldots V,$$

where V is the number of kinds of the application package. This indicates that different application-encrypting keys K1 through K_v is assigned to respective kinds of applications, that is, volume VID1 through VID_v.

In the next step 604, the user secret key SK_u is read from the IC card 5. In the next step 606, the PK_u-encrypted AP-encrypting key $e1(PK_u, K_v)$ is decrypted with the user secret key SK_u to obtain the application encrypting key K_v. Then in the next step 608, the K_v-encrypted application (AP), i.e., $e(K_v, AP)$ which is recorded on the DVD 3 is decrypted with the obtained AP-encrypting key K_v to obtain $d(K_v, e(K_v, AP)) = AP$, while passing the obtained application data to the video and audio output IF 140. The obtained application data has the form of an MPEG 2 bit stream. The video and audio output IF 140 converts the MPEG 2 bit stream of the application data into video and audio output signals through MPEG 2 video and audio decoding. The video and audio output signals are applied to the display device 146 and the loudspeaker 148, respectively.

Play an Application in Usage-sensitive Charging system

FIG. 15 is a flow chart showing a procedure of a charged play process 700 shown as step 700 in FIG. 5, wherein connecting adjacent blocks by two flow lines indicates that each block is executed interactively by a client of $CADD_c$ and an associated server of SID_s. In FIG. 15, the client 2 enters the process 700 via step 516 of FIG. 5 and proceeds to block 630, where the client 2 and the associated server 8 execute the initial routine 80. In the next block 640, the client 2 displays an expected charge and a total amount of charges received from the server 8, and let the user decide whether to play the desired application.

FIGs. 16A and 16B are flow charts jointly showing a procedure formed of exemplary expected charge informing routines 640a and 640b interactively executed by the client 2 and the associated server 8, respectively. The routines 640a and 640b are very similar to the routine 97 except that in the routine 640, the DURATION (D_n) or "play time" has been replaced with RATE PER ACCESS and "charge"; between steps 92a and 93a, there has been added a step 641 of the server generating and storing a pseudo random number R in a memory location R'; in step 93a, the server sends the pseudo random number R as well; between steps 94 and 95a there has been added a step 643 of the client storing the received pseudo random number R in a memory location R" for subsequent use. The replacement of DURATION (D_n) with RATE PER ACCESS is achieved by accessing a RATE PER ACCESS field 74 instead of a DURATION field

73 in table 70. Further, in the routine 640 there have been added the following steps: in step 644 following the step 96a, the client 2 makes a check to see if the user decides to play the application; if not, the client 2 sends a quit message to the server of $SADD_s$ in step 645, and ends the routine 640; on the other hand, in step 642 following the step 93a, the server 8 of SID_s waits for a call from the client 2 of $CADD_c$; on receiving a call from the client, the server makes another check in step 646 to see if what has been received is a quit message; if so, the client ends the routine 640; and if the user decided to play the application in step 644, which means that what the server has received is not a quit message but an encrypted credit card number as seen from the description below, then the client 2 and the server 8 proceed to the step 650 of FIG. 15.

In the next block 650, the server 8 obtains a user's credit card number (CCNOu) through the client 2 keeping the security of the card number as shown in FIGs. 17A and 17B. In step 647, the client 2 encrypts the credit card number of the user which has been input by the user through a human IF 110 with a key, i.e., the pseudo random number R which has been stored in a memory location R" in step 643 of FIG. 16A to obtain e2(R, CCNOu). In the next step 648, the client 2 further encrypts R + e2(R, CCNOu) with another key or a server public key read from the distribution descriptor 23 recorded in the burst cutting area of the DVD to obtain

$$e1(PK_s, R + e2(R, CCNOu)).$$

In the next step 649, the client 2 sends the encrypted data to the server 8. Through step 646 of FIG. 16B, the server proceeds to step 650, where the server 8 finds that what was received from the client $CADD_c$ is encrypted data. In the next step 651, the server 8 reads a server secret key SK_s from an IC card 7. in the next step, the server 8 decrypts the received encrypted data with the server secret key SK_s as follows;

$$d1(SK_s, \text{encrypted data}) = d1(SK_s, e1(PK_s, R + e2(R, CCNOu)) = R + e2(R, CCNOu).$$

In step 653, the server 8 makes a check to see if the just obtained pseudo random number R coincides with the random number R which has been stored in a memory location R' of the server. if so, the server 8 sends an enable message to the client of $CADD_c$, and in step 655 decrypts e2(R, CCNOu) with the pseudo random number R to obtain the user's credit card number CCNOu. On the other hand, in response to a reception of the enable message in step 657, the client 2 exits from the process. After step 655, the server also exits from the process. If the result is NO in step 653, then the server 8 sends a disable message to the client in step 656, and ends the process. In response to a reception of the disable message in step 657, then the client displays a message to this effect in step 658, and then ends the process.

After operation of block 650, the client 2 waits, in step 663, for a report from the server on whether the credit card for the transmitted card number (CCNOu) is valid or not, while the server 8 refers to the credit company associated with the card number in step 661 to see if the credit card is valid. if not, the server 8 informs the client 2 of the invalidity of the credit card in step 662, and ends the process. If the card is valid in step 661, the server 8 informs the client of the validity in step 667. If the client 2 receives a report from the server in step 663, the client makes another check in step 664 to see if the report indicates the validity of the card. If not, the client display a message to indicate the invalidity in step 665, and ends the process. If the report indicates the validity in step 664, which means the completion of step 667, then the client 2 and the server 8 proceed to the next block 670.

In step 670, the client 2 and the server 8 execute timed play and metered charge report routine. FIGs. 18A and 18B are flow charts jointly showing a procedure formed of routines 675ac and 675bc interactively executed for playing an application while timing the duration and displaying a charge and a total amount of charges after the play. In FIG. 18, the routines 675ac and 675bc are identical to the routine 675a and 675b in FIGs. 11A and 11B except that "time" has been replaced with "charge", and accordingly VM-METER and AM-METER have been replaced with VC-METER and AC-METER.

The operation, in the client 2, of playing an application on usage-sensitive charging is completed by block 675 of FIG. 15 or step 218a of FIG. 18A. After step 212a, the server 8 charges the play to the credit card number CCNOu obtained in step 655 of FIG. 17B in step 680. This completes the whole of the charged application play process of FIG. 15.

In this process, only information on charge is given to the user. It is very easy to provide information on both time and charge by adding steps 91 through 93 and 95 to the routines 640b and 640a, and by adding steps 210 and 218 to the routines 675bc and 675ac.

As described above, expected time and/or charge are (is) displayed before playing a user specified application. This is helpful for the user to decide whether to play the application. Additionally, charging is done based on the actually timed play duration. This makes the charging reasonable.

In the above description, the arrangement is such that the user has to input his or her credit card number CCNOu each time he or she wants to play an application. However, instead of doing this, the credit card number CCNOu may be stored in non-volatile memory or EEPROM 103 in a PW_u-encrypted form. In this case, CCNOu is obtained by decrypting PW_u-encrypted CCNOu (e.g., $e(PW_u, CCNOu)$) with a password entered by the user. That is, d(entered password, $e(PW_u, CCNOu)$) = CCNOu.

Permit the Play Within a Preset Limit

FIG. 19 is a flow chart showing a procedure interactively executed by the client 2 and the server 8 in the operation block 800 of FIG. 5, wherein blocks connected with two flow lines indicates that operation of the blocks is done by the two elements 2 and 8. In this case, it is assumed that a preset limit is recorded in or on the application package and is transmitted from client 2 to server each time of play. On entering the process 800 via step 516 of FIG. 5. the client 2 proceeds to step 801, where the client 2 and the server 8 executes the initial routines 80. It is noted that in routine 80b, if there is a record for VID_v and NO_{v-i}, then the limit value (LV_{v-i}) field of the table 60 of FIG. 6A contains the limit value transmitted from the client 2, otherwise, the received limit value is stored in the LV_{v-i} field when the record for VID_v and NO_{v-i} is added in step 88.

In step 810, the server 8 makes a check if a meter associated with the TOU code received from the client 2 is under the limit value. This check is made by comparing an LV field and LV-meter field associated with the TOU code in table 60. if the value of the LV-meter is equal to or greater than the LV field value, then the server returns an over limit massage to the client 2 in step 820. If not, the server 8 returns an underlimit message to the client 2 in step 822, and proceeds to step 828. If the client 2 receives the overlimit message in step 824, then the client 2 displays a message to this effect. If not, the client 2 proceeds to the step 828.

Since the expected play time informing routines 97a and 97b and the application play subroutine 600 has been described above, the description of steps 828 and 830 are omitted.

According to this feature of the invention, it is possible to limit the use of charged information. This feature is especially useful in case when a user who have paid in advance for the use of the application package is permitted to use the application package within a limit value.

Though it has been assumed that the limit values are included in the application package, the limit values may be kept in the servers of the provider or distributer from the beginning. In this case, the limit values are fixed. However, if limit values are permitted to be set and recorded in the application package at the time of distribution or sales, the limit values are advantageously set according to an amount paid.

As is apparent from the foregoing, as a limit value, any use-limiting factors will do that can be measured in quantity. Such limit values are, for example, the effective date and time, the allowable expiration date and time, the maximum amount of play time, the allowable access count.

It is also possible to combine this feature with a charged application play feature. That is, an arrangement may be such that the user is permitted to use an application package on usage-sensitive charging only if the value of an LV-meter associated with the TOU is under the value of the corresponding LV or the value recorded in a field 33 or 34 of the distribution descriptor 23.

Modification I

In the above embodiment, applications, if more than one, in one volume are encrypted by an identical application encrypting key K_v. However, the applications APa in one volume may be encrypted with respective AP-encrypting keys K_a, where a lower case "a" following AP and K is a serial number assigned to each application ID. In this case, each of the AP-encrypting keys K_a are encrypted with the user public key PK_u, and stored in the PK_u-encrypted AP-encrypting key (K_a) fields 32a in the distribution descriptor 23.

Modification II

It has been assumed that the user of the DVD 3 is limited to the purchaser thereof who have had the PK_u-encrypted AP-encrypting key (K_v) recorded on the DVD 3. However, the system may be so arranged that predetermined people, e.g., family members FM_1, FM_2,..., FM_N of the purchaser can use the DVD (N is the number of the family members). One of the ways to realize this is to encrypt the AP-encrypting key K_v with a public key PK_{u-n} of each member FMn (n = 1, 2,...,N) to obtain $e1(PK_{u-1}, K_v)$, $e1(PK_{u-2}, K_v)$,..., $e1(PK_{u-n}, K_v)$ and to record them in the PK_{u-n}-encrypted AP-encrypting key $e1(PK_{u-n}, K_v)$ fields 32 of the distribution descriptor 23 at the time of purchase of the DVD.

Modification III: K_v Retrieval From Server

In the above description, the AP-encrypting key K_v has been recorded in a PK_u-encrypted form on the DVD 3. However, the AP-encrypting key K_v may be managed by the server 8 and transmitted to the client or the DVD player 2 in response to a request issued from the DVD player 2 each time of use of the DVD 3. In this case, there is no need of providing the distribution descriptor 23 with the PK_u-encrypted AP-encrypting key field 32. Instead each of the servers has to store an AP-encrypting key table (or K_v table) and a PK_u table (shown in FIGs. 20A and 20B) in the hard disc. As shown in FIG. 20A, the K_v table a volume ID (VID_v) field (as the entry of record) and an AP-encrypting key (K_v) field in

each record. In FIG. 20B, each record of the PK_u table comprises a volume ID (VID_v) field (as the entry of record), a volume issue number (NO_{v-i}) field and a PK_u field (Successive same values in the first field are shown by showing only the first appearing one). Further, the process (or step) 610 of obtaining the AP-encrypting key K_v, that is, a group of the steps 602, 604 and 606 in the application play routine 600, has to be replaced with a process of FIG. 20C.

FIG. 20C is a flow chart of a process in which the client DVD player 2 obtains the application encrypting key K_v from the server 8. In step 616, the server 8 retrieves a key K_v from the K_v table by using VID_v. In the next step 618, the key K_v is encrypted with an arbitrary number used only in the current process, e.g., a pseudo random number R to obtain $e2(R, K_v)$. In the next step 620, the server 8 retrieves a key PK_u from the PK_u table by reading the PK_u field of the record which contains VID_v and NO_{v-i} in the VID_v and NO_{v-i} fields, respectively. In the next step 622, $R + e2(R, K_v)$ is encrypted with the retrieved key PK_u to obtain a double encrypted AP-encrypting key

$$e1(PK_u, R + e2(R, K_v)),$$

which is returned to the client with a client network address $CADD_c$ in the next step 624.

On the other hand, the controller 100 of the client 2 waits for a response from the server 8 of SID_s in step 626. If there is any response from the server 8 of SID_s in step 626, then the client DVD 3 receives the data $e1(PK_u, R + e2(R, K_v))$ from the server 8 in step 628. In the next step 630, the received data is decrypted with the user secret key SK_u read from the IC card 5. Specifically, the following calculation is done.

$$d1(SK_u, e1(PK_u, R + e2(R, K_v))) ==> R + e2(R, K_v)$$

In the next step 632, $e2(R, K_v)$ is decrypted with the obtained pseudo random number R. Specifically, the following calculation is done.

$$d2(R, e2(R, K_v)) ==> K_v$$

Thereafter, the controller 100 proceeds to the step 608 of FIG. 14.

In this modification, the applications APa in one volume may be encrypted with respective AP-encrypting keys K_a. In this case, the K_v table has to be replaced with K_a table in which each record comprises an application ID (AID_a) field and an AP-encrypting key (K_a) field. Further in step 612, the controller 100 of the DVD player 2 has to also send the application ID of the application to be played to the server.

Also in this modification, the system may be, again, so arranged that predetermined people, e.g., family members FM_1, FM_2,..., FM_N of the purchaser can use the DVD (N is the number of the family members). In this case, for each member FMn (n = 1, 2,...,N), the server 8 has to use the member's own public key PK_{u-n} in encrypting the AP-encrypting key K_v. One way to realize this is to issue a volume issue number NO_{v-i-n} to each member FMn at the time of sales of the DVD, provide the non-volatile memory (not shown) of the DVD player 2 with a table for associating the user's password PWn with the volume issue number NO_{v-i-n}, send the volume issue number (NO_{v-i-n}) associated with the user's password in step 612, and use not the PK_u table but a PK_{u-n} table in which each of the records has the following fields:

$$VID_v, NO_{v-i-n}, PK_{u-n}.$$

Another way is to issue and record not only a volume issue number NO_{v-i} but also family member numbers FMNn for all members at the time of sales of the DVD, provide the non-volatile memory (not shown) of the DVD player 2 with a table for associating the user's password PWn with the corresponding family member number FMNn, send the volume issue number (NO_{v-i}) and the family member number FMNn associated with the user's password in step 612, and use another PK_{u-n} table in which each of the records has the following fields:

$$VID_v, NO_{v-i}, FMNn, PK_{u-n}.$$

In the process of FIG. 20C, the server 8 may be authenticated by means of a public-key cryptosystem using a pair of server secret and public keys (SK_s, PK_s). In this case, the server 8 signs the double-encrypted AP-encrypting key

$$e1(PK_u, R + e2(R, K_v))$$

with a signing key or the server secret key SK_s after step 622. While the client or DVD player 2 tests the signature by the server 8 with a test key or the server public key PK_s contained in the PK_s field 31 of the distribution descriptor 23 recorded in the burst cutting area of the DVD 2 before step 630.

However, even if just described authentication of the server 8 is omitted, an attacker will never go to any greater length than a steal of TOU code plus limit value, a volume ID VID_v, a volume issue number NO_{v-i}, and the client network address $CADD_c$. This is not a serious problem.

In the process of FIG. 20C, a pseudo random number R has been used as a pseudo variable which takes a different value each time of execution of the process. However, as the pseudo variable, any thing will do if the result of encryption with it takes a different value each time of execution of the process.

Modification IV

In the first illustrative embodiment, the decryption of application is achieved by software. For this purpose, the controller 100 has to read the user secret key SK_u from the IC card 5 through the bus 102, which leaves the possibility of permitting a breaker to easily steal the user secret key SK_u through the bus 102. In order to prevent this, the process

achieved by the steps 604 through 608 may be realized by hardware as shown in FIG. 21, which is a block diagram of an exemplary decipherer-built-in IC card IF. In FIG. 21, the decipherer-built-in IC card IF 120a comprises an IC card receptacle 121 and a printed wiring board 122 extending from and fixed with the receptacle 121. An IC 123 is mounted on the printed wiring board 122. The IC 123 comprises a memory IF 125 which usually connects the memory of the IC card 5 with the bus 102 and, in response to an instruction from the controller 100, reads and passes the key SK_u to the next stage; a K_v decoder 126 for receiving the key Sk_u and encrypting $e1(PK_u, K_v)$ with the key SK_u to yield K_v; and an AP decoder 127 for receiving the key K_v and encrypting $e(K_v, AP)$ to yield application data (AP). The printed wiring board 122 portion may be preferably molded together with the IC card receptacle 121 portion so as to make the whole a single body. By doing this, leaking of the user secret key SK_u can be prevented.

This modification can be also applied to a system 1 using the cryptosystem of FIG. 20C. In this case, the K_v decoder 126 of FIG. 21 has to be replaced with a K_v decoder 126a as shown in FIG. 22. In FIG. 22, the K_v decoder 126a decrypts the input data, $e1(PK_u, R + e2(R, K_v))$, from the bus 102 by using the user secret key SK_u passed by the memory IF 125 to obtain $R + e2(R, K_v)$, while decrypting the obtained data $e2(R, K_v))$ with the obtained random number R and outputting the key K_v.

Embodiment II

FIG. 24 is a block diagram showing an arrangement of a system capable of playing a distributed application package, e.g., a DVD on the terms of use of the DVD without communicating with any server according to a second illustrative embodiment of the invention. In FIG. 24, the system 1a is identical to the client 2 of FIG. 1 except that the communication IF 150 has been eliminated because of no need of communication with a server and the controller 100 has been replaced with a controller 100a. In the controller 100a, a not-shown ROM for storing a control program as described later and the EEPROM 103 have been also replaced with a new ROM (not shown) and an EEPROM 103a. In order to play a role of the server 8, the system 1a has to have table 60 of FIG. 6A in any non-volatile memory, e.g., the EEPROM 103a and an application duration (play time) for each application as defined in table 70 of FIG. 6B has to be included in the control data of each application package.

FIG. 25 schematically shows an exemplary control program executed by the controller 100a shown in FIG. 24. The control program of FIG. 25 is also identical to that of FIG. 5 except that the decision step 516 and the step 700 has been eliminated because the limit-attached play mode is not supported by the system 1a in this embodiment, and the steps 650 and 800 are replaced with steps 650a and 800a. Accordingly, operation after step 514 will be described in the following.

If the lower digit of the terms-of-use (TOU) code is 0 in the decision step 514, then in step 650a the controller 100a plays, in the free play mode, the application stored in the selected application in step 506 or 512 and ends the operation. It should be noted that since the system 1a does not have the charged play mode, the lower digit of the TOU code is defined as follows.

Higher digit of terms-of-use code (Hexadecimal)	Corresponding limit value	Play mode
0	None	Free play mode
2	Effective date and time	Limit-attached play mode
3	Allowable expiration date and time	
4	Maximum amount of used period	
5	Allowable access count	
:	:	
:	:	

Accordingly, if the lower digit of the TOU code is not 0 in the decision step 514, then in step 800a the controller 100a plays, in the limit-attached play mode, the application stored in the selected application in step 506 or 512 and ends the operation.

FIGs. 26 and 27 show an operation of a free play mode shown in step 650a of FIG. 25 in a detailed form and a further detailed form, respectively. In FIG. 26, the controller 100a executes an initial routine 80a in step 660a, in step 670a executes an expected play time informing routine, and in step 680a executes an application play and metered play time report routine.

As shown in FIG. 27, in the initial routine 80c, the controller 100a searches the table 60 for a record which contains VID_v and NO_{v-i} in the volume ID and issue No. fields thereof, respectively in step 86. If the search is unsuccessful, then the controller 100a adds the record for VID_v and NO_{v-i} and fills relevant fields with AID_{v-i-a} and a limit value, if any, in the table 60 in step 88, and proceeds to step 90. Also, if the search in step 86 is successful, the server 9 proceeds to step 90, where the controller 100a selects a routine to execute next according to the value of the TOU code and enters the selected routine. In this case, if the TOU code = x0H (x: an arbitrary HEX number, the letter H in the last position indicates that the preceding number is in hexadecimal), then a routine for playing an application free of charge is selected. If the TOU code ≥ x1H, then a routine is selected which plays an application only if the software meter of a use-limiting factor is under a preset value.

The expected play time informing routine 670a is identical to the routines 97 (FIG. 10) minus communication steps 93 and 94, comprising the above described steps 91, 92 and 95. Similarly, it is seen from FIGs. 11 and 13A that the above described steps 620, 622, 624, 210 and 218 are executed in this order in the timed play and metered usage report routine 680a. In this way, the system 1a permits the user to play the application stored in the selected application (steps 506 and 512 of FIG. 25) free of charge.

FIG. 28 is a flow chart showing an operation of a limit-attached play mode shown in step 800a of FIG. 25. Since this operation is very similar to that of FIG. 19, only the flow is briefly described, omitting the details of each step. In FIG. 28, controller 100a first makes a check if a meter associated with the TOU code has reached the limit value obtained with the TOU code. If so, then the server returns an overlimit massage to controller 100a in step 820. Otherwise, the controller 100a proceeds to the expected play time informing routine 828a (= 670a), where the controller 100a executes the above described steps 91, 92 and 95, and then calls the application play subroutine 600 in step 830, thereby completing the operation. Since the application play subroutine 600 has been detailed above, further description is omitted. In this way, the system 1a permits the user to play the application stored in the selected application (steps 506 and 512 of FIG. 25) only if the limit value associated with the TOU code assigned to the volume or the user-specified application has not been reached.

According to the second embodiment, the system 1a can operate in either of the free play mode and the limit-attached play mode without the need of communication with a server. For this, the system 1a may be made portable.

Modifications

In the above description, the illustrative embodiment has been described in conjunction with the DVD. The same discussion can be applied to such package media as permit write once or more.

Further, the present invention is also applicable to application packages distributed via transmission media. In this case, the distributed application packages are stored in a bulk storage in the user's device. An application package comprises one or more application and application control data, that is, an application descriptor and distribution descriptor. One volume is stored as a file. Since a plurality of application package may be stored in a single storage, each application package does not have to contain a control program. One control program, which may be distributed via either package or transmission media, is enough for one user device. The folder or directory in which the application packages are stored is set for a user specified one in the control program when the control program is installed. The data to be recorded in the distribution descriptor is included in the application package by the provider according to the information given by the user.

As described above, one who is permitted to use an application package is limited to an owner of the IC card which stores a user secret key SK_u corresponding to the user public key PK_u used for encryption of the AP-encrypting key K_v in the application package. For this, even if someone has unjustly obtained an application package, for example, by copying the whole volume from the DVD on which the volume is recorded, he or she can not use it without the IC card of the owner of the DVD. Thus the inventive system can prevent unjust use of an application package (DVD in this case) by any other person than the regular owner of the application package.

Also, the inventive system is so arranged that most part of the application package is recorded by pressing in manufacturing process of the DVDs, whereas at least a part of the volume control data (i.e., the distribution descriptor) can be determined at the time of, e.g., distribution of each of the DVDs after the manufacturing process. This makes the system flexible because control data can be easily changed without changing the stamper.

In the initial routines 80a and 80b in FIG. 8A and 8B, the data transmitted with the service request may be encrypted in the same manner as in case of the transmission of user's credit card number shown in FIG. 17. However, in case of the initial routines, there are a plurality of data. These data may be encrypted in the following way.

If the data to be encrypted are D1, D2,... then they are first encrypted with a key R as follows:

e2(R, D1), e2(R, D2),...

Then further encryption is made with a server public key PK_s as follows:

e1(PK_s, R + e2(R, D1) + e2(R, D2),....).

In the process of FIG. 17, the user may be authenticated by means of a public-key cryptosystem using a pair of

user secret and public keys (SK_u, PK_u). In this case, the client 2 signs the double-encrypted credit card number

$$e1(PK_s, R + e2(R, CCNOu))$$

with a signing key or the user secret key SK_u after step 648. While the server tests the signature by the client 2 with a test key or the user public key PK_u before step 650.

Instead of storing a single server public key in the distribution descriptor 23, a plurality of server public keys or all the server public keys may be recorded. By doing this, it is possible, for example, to setting a different charge depending on the server public key which the user have selected by appropriately combining the tables 70 and 75.

Also, application packages with an identical volume ID can have different server public keys recorded. A plurality of toll center may be advantageously provided for application packages of the same title.

In order to prevent any use of IC card by other person than the owner of the IC card, it is possible to add, before the Sku reading step 604, the steps of prompting the user to enter a password through a human IF 110 and proceeding to step 604 only if the entered password coincides with the user password PW_u stored in the IC card.

Though the IC card 5 is used in the above embodiment, the IC card IF 120 may be replaced with a magnetic card reader to permitting the use of the magnetic card. Alternatively, the arrangement may be such that the user enters his or her password each time the user uses the DVD.

Instead of storing the user secret key SK_u in the IC card 5, the key Sku may be stored in non-volatile memory in a PW_u-encrypted form. In this case, the key SK_u is obtained by decrypting PW_u-encrypted SK_u with a password entered by the user.

The discussion of three preceding paragraphs are applied to the IC card used for storing the server secret key in the server. However, in this case the user has to be taken as the administrator of the toll server.

Many widely different embodiments of the present invention may be constructed without departing from the spirit and scope of the present invention. It should be understood that the present invention is not limited to the specific embodiment described in the specification, except as defined in the appended claims.

A system for permitting only an authentic user to play a desired application contained in a distributed application package in one of predetermined operation, e.g., free play mode, charged mode, limit-attached play mode, etc. The system comprises a client for playing an application under the control of a server connected with the client through a communication network. The application package (the volume) includes a distribution descriptor which contains mode codes assigned to the volume and the applications of the volume. The data of distribution descriptor is decided and stored in the descriptor at the time of distribution of the volume. This feature makes the system flexible. There is also disclosed a system operatable without communicating with a server.

Claims

1. An application package for use in a system for playing an application contained in the application package (the volume), the application package comprising:

   application data for at least one application; and
   volume control data for use in controlling said system, wherein said volume control data at least comprises:
   a volume ID for identifying the kind of said application package (said volume);
   an issue number assigned in order of issue to each of the volumes of said kind; and
   application IDs each assigned to one of said at least one application contained in said volume, and wherein:
   at least a part of said volume control data is to be added to said volume after the creation of said volume; and
   said at least a part of said volume control data includes said issue number.

2. An application package as defined in claim 1, wherein:

   said application data has been encrypted with an encrypting key; and
   said at least a part of said volume control data includes a user's public key-encrypted version of said encrypting key used.

3. An application package as defined in claim 1, wherein said at least a part of said volume control data includes mode codes which are assigned to said volume or said at least one application and each indicate a play mode associated with one of said volume or said at least one application to which the mode code is assigned.

4. A package media on which an application package as defined in claim 1 has been recorded.

5. A package media of a write-once type on which an application package as defined in claim 1 has been recorded.

6. A package media on which an application package as defined in claim 1 has been recorded wherein said at least a part of said volume control data is recorded in an area different from data area where said application data is recorded on the package media.

7. A method for sending data with a raised security from a first device to a second device through a public telecommunication network, comprising the steps of:

in said second device,

generating a pseudo random number;
transmitting said pseudo random number to said first device;

in said first device,

encrypting said data with said transmitted pseudo random number into encrypted data;
encrypting concatenated data consisting of said pseudo random number and said encrypted data with a public key of said second device into double-encrypted data;
sending said double-encrypted data to said second device; in said second device,
decrypting said double-encrypted data with a secret key of said second device which corresponds to said public key into decrypted data consisting of a decrypted random number portion and another decrypted portion; and
decrypting said another decrypted portion with said transmitted random number to obtain said data.

8. A method for sending a plurality of pieces of data with a raised security from a first device to a second device through a public telecommunication network, comprising the steps of:

in said second device,

generating a pseudo random number;
transmitting said pseudo random number to said first device;

in said first device,

encrypting each of said pieces of data with said transmitted pseudo random number into an encrypted piece of data;
encrypting concatenated data consisting of said pseudo random number and said encrypted pieces of data with a public key of said second device into double-encrypted data;
sending said double-encrypted data to said second device; in said second device,
decrypting said double-encrypted data with a secret key of said second device which corresponds to said public key into decrypted data consisting of a decrypted random number portion and said plurality of decrypted data portions; and
decrypting each of said decrypted portions with said transmitted random number to obtain said pieces of data.

9. A method as defined in claim 7 or 8, further comprising the steps, executed after said step of decrypting said double-encrypted data, of:

proceeding to a next step only if said decrypted random number portion coincides with said transmitted pseudo random number; and
said second device informing said first device of a failure in decryption if said decrypted random number portion does not coincide with said transmitted pseudo random number.

10. In a system provided with means for playing an application contained in an application package, a method for permitting a user to play an encrypting key-encrypted application contained in a distributed application package which further contains, as volume control data, a user's public key-encrypted encrypting key so encrypted as to be able to be decrypted with a secret key of the user into said encrypting key, the method comprising the steps of:

reading said user's public key-encrypted encrypting key from said distributed application package (said vol-

ume);
obtaining said secret key;
decrypting said user's public key-encrypted encrypting key with said secret key to obtain said encrypting key; and
decrypting said encrypting key-encrypted application with said obtained encrypting key into application data while passing said application data to said means for playing an application.

11. In a system comprising a client provided with means for playing an application contained in an application package and a server connected with the client through a communication network, a method for permitting a user to play one of encrypting key-encrypted applications contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order and application IDs, the method comprising the steps of:

said client reading said volume ID, said issue number and an application ID for said one of encrypting key-encrypted applications (said encrypting key-encrypted application) from said volume and sending to said server;

in said server,

retrieving said encrypting key by using said volume ID;
retrieving a public key of said user by using said volume ID and said issue number;
generating a pseudo random number;
double-encrypting said encrypting key with said pseudo random number and said public key into a double encrypted data;
sending said double-encrypted data to said client; in said client,
obtaining a secret key of said user which corresponds to said public key;
obtaining said encrypting key by decrypting said double-encrypted data with said secret key;
decrypting said encrypting key-encrypted application with said obtained encrypting key into application data while passing said application data to said means for playing an application.

12. A method as defined in claim 10 or 11, wherein said means for obtaining a secret key comprises means for reading said secret key from a portable memory of said user.

13. A method as defined in claim 12, wherein said portable memory is an IC card.

14. In a system comprising a client provided with means for playing an application package and a server connected with the client through a communication network for controlling the client, the application package (the volume) containing, as volume control data, a volume ID and an issue number issued to each of the volumes of said volume ID in an issued order, a method for controlling the amount of play time comprising the steps of:

said client sending said volume ID and said issue number to said server;
said server retrieving an expected play time associated with said volume ID and said issue number; and
said server adding said expected play time to the value of a total play time associated with said volume ID and said issue number.

15. In a system comprising a client provided with means for playing an application contained in an application package and a server connected with the client through a communication network for controlling the client, the application package (the volume) containing, as volume control data, a volume ID, an issue number issued to each of the volumes of said volume ID in an issued order and an application ID for the application, a method for controlling the amount of play time comprising the steps of:

said client sending said volume ID, said issue number and said application ID to said server;
said server retrieving an expected play time associated with said volume ID, said issue number and said application ID; and
said server adding said expected play time to the value of a total play time associated with said volume ID and said issue number.

16. In a system comprising a client provided with means for playing an application contained in an application package and a server connected with the client through a communication network for controlling the client, the application package (the volume) containing, as volume control data, a volume ID and an issue number issued to each of the volumes of said volume ID in an issued order, a method for controlling the amount of play time comprising the steps of:

said client and said server interactively measuring, as a measured play time, a play time of said application; and
said server adding said measured play time to the value of a total play time associated with said volume ID and said issue number.

17. A method as defined in claim 16, wherein said step of measuring a play time comprises the step of using a timer of said server.

18. A method as defined in claim 16, wherein said step of measuring a play time comprises the step of using a timer of said client.

19. In a system comprising a client for playing an application package and a server connected with the client through a communication network wherein the application package (the volume) comprises application data and control data and at least a part of the control data has been added to the volume after the creation of said volume, a method for sending desired data from one side of said client and said server to the other side, the method comprising the steps of:

including a secret key of said other side in said at lest a part of said control data;

in said other side,

generating a pseudo random number;
transmitting said pseudo random number to said one side;

in said one side,

encrypting said desired data with said transmitted pseudo random number into encrypted data;
encrypting concatenated data consisting of said pseudo random number and said encrypted data with said public key of said other side into double-encrypted data;
sending said double-encrypted data to said other side;

in said other side,

decrypting said double-encrypted data with a secret key of said other side which corresponds to said public key into decrypted data consisting of a decrypted random number portion and another decrypted portion; and
decrypting said another decrypted portion with said transmitted random number to obtain said desired data.

20. A method as defined in claim 19, wherein said generating a pseudo random number includes storing said pseudo random number in memory, and wherein the method further comprises the step, executed prior to said decrypting said another decrypted portion, of:

in response to a determination that said decrypted random number portion does not coincide with said pseudo random number stored in said means for storing said pseudo random number stored in said memory, informing said one side of a failure in decryption instead of passing the control to next means.

21. In a system comprising a client provided with means for playing an application contained in an application package and a server connected with the client through a communication network, a method for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order, and an application ID for said application, the method comprising the steps of:

proceeding to a next step only if the value of a meter field associated with said volume ID, said issue number and said application ID is under the value of a limit value field associated with said volume ID, said issue number and said application ID in a volume data table; and

displaying a message informing an overlimit on a display device of said client and quit the operation otherwise.

22. In a system comprising a client provided with means for playing an application contained in an application package and a server connected with the client through a communication network, a method for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order, an application ID for said application and a limit value for limiting the play of said application, the method comprising the steps of:

proceeding to a next step only if the value of a meter field associated with said volume ID, said issue number and said application ID in a volume data table is under said limit value; and

displaying a message informing an overlimit on a display device of said client and quit the operation otherwise.

23. A method as defined in claim 21, wherein said limit value is one of effective date and time, allowable expiration date and time, a maximum amount of play time, and an allowable access count.

24. A method as defined in any of claims 11, 15 and 16, wherein said step of said client sending to said server comprises the steps of:

said client encrypting at least one of said volume ID, said issue number and said application ID into encrypted data; and

said server decrypting said encrypted data.

25. A system for sending data with a raised security from a first device to a second device through a public telecommunication network, comprising:

means provided in said second device for generating a pseudo random number;
means provided in said second device for transmitting said pseudo random number to said first device;
means provided in said first device for encrypting said data with said transmitted pseudo random number into an encrypted data;
means provided in said first device for encrypting concatenated data consisting of said pseudo random number and said encrypted data with a public key of said second device into double-encrypted data;
means provided in said first device for sending said double-encrypted data to said second device;
means provided in said second device for decrypting said double-encrypted data with a secret key of said second device which corresponds to said public key into decrypted data consisting of a decrypted random number portion and another decrypted portion; and
means provided in said second device for decrypting said another decrypted portion with said transmitted random number to obtain said data.

26. A system for sending a plurality of pieces of data with a raised security from a first device to a second device through a public telecommunication network, comprising:

means provided in said second device for generating a pseudo random number;
means provided in said second device for transmitting said pseudo random number to said first device;
means provided in said first device for encrypting each of said pieces of data with said transmitted pseudo random number into an encrypted piece of data;
means provided in said first device for encrypting concatenated data consisting of said pseudo random number and said encrypted pieces of data with a public key of said second device into double-encrypted data;
means provided in said first device for sending said double-encrypted data to said second device;
means provided in said second device for decrypting said double-encrypted data with a secret key of said second device which corresponds to said public key into decrypted data consisting of a decrypted random number portion and said plurality of decrypted data portions; and
means provided in said second device for decrypting each of said decrypted portions with said transmitted random number to obtain said pieces of data.

27. A system as defined in claim 25 or 26, further comprising:

means, provided in said second device, activated prior to decrypting each of said decrypted portions and responsive to a determination that said decrypted random number portion does not coincide with said transmitted pseudo random number, for informing said first device of a failure in decryption instead of passing the control to next means.

28. A system for playing an encrypting key-encrypted application contained in a distributed application package which further contains, as volume control data, a user's public key-encrypted encrypting key so encrypted as to be able to be decrypted with a secret key of the user into said encrypting key, the system comprising:

means for reading said user's public key-encrypted encrypting key from said distributed application package (said volume);
means for obtaining said secret key;
means for decrypting said user's public key-encrypted encrypting key with said secret key to obtain said encrypting key;
means for decrypting said encrypting key-encrypted application with said obtained encrypting key to provide application data; and
means for using said application data for playing.

29. A system for permitting a user to play an encrypting key-encrypted application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order and application IDs, the system comprising:

a client for playing an application by using application data; and
a server for controlling said client through a communication network, wherein said client comprises:
means for reading and sending said volume ID, said issue number and an application ID for said one of encrypting key-encrypted applications (said encrypting key-encrypted application) from said volume to said server, said server comprises:

means for retrieving said encrypting key by using said volume ID;
means for retrieving a public key of said user by using said volume ID and said issue number;
means for generating a pseudo random number;
means for double-encrypting said encrypting key with said pseudo random number and said public key into a double encrypted data; and
means for sending said double-encrypted data to said client, and said client comprises:
means for obtaining a secret key of said user which corresponds to said public key;
means for obtaining said encrypting key by decrypting said double-encrypted data with said secret key;
means for decrypting said encrypting key-encrypted application with said obtained encrypting key to provide application data; and
means for using said application data for playing.

30. A system as defined in claim 28 or 29, wherein said means for obtaining a secret key comprises means for reading said secret key from a portable memory of said user.

31. A system as defined in claim 30, wherein said portable memory is an IC card.

32. A system for permitting a user to play a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume) and an issue number issued to each volume of the kind in an issued order, the system comprising:

a client for playing said distributed application package; and
a server for controlling said client through a communication network, wherein:
said client comprises means for sending said volume ID and said issue number to said server; and
said server comprises means for retrieving an expected play time associated with said volume ID and said issue number, and means for adding said expected play time to the value of a total play time associated with said volume ID and said issue number.

33. A system for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order and an application ID for the application, the system comprising:

a client for playing said application; and
a server for controlling said client through a communication network, wherein:
said client comprises means for sending said volume ID, said issue number and said application ID to said server; and
said server comprises means for retrieving an expected play time associated with said volume ID, said issue number and said application ID, and means for adding said expected play time to the value of a total play time associated with said volume ID and said issue number.

34. A system for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order and an application ID for the application, the system comprising:

a client for playing said application; and
a server for controlling said client through a communication network, wherein:
said client and said server comprise means for interactively measuring, as a measured play time, a play time of said application; and
said server further comprises means for adding said measured play time to the value of a total play time associated with said volume ID and said issue number.

35. A system as defined in claim 34, wherein said means for interactively measuring a play time comprises means for using a timer of said server.

36. A system as defined in claim 34, wherein said means for interactively measuring a play time comprises means for using a timer of said client.

37. A system for permitting a user to play an application package (the volume) comprising application data and control data wherein at least a part of the control data has been added to the volume after the creation of said volume, the system comprising:

a client for playing said volume; and
a server for controlling said client through a communication network, wherein said server comprises means for storing a secret key of said server and said at least a part of said control data includes a public key corresponding to said secret key, and wherein the system comprises:
means provided in said server for generating a pseudo random number;
means for storing said pseudo random number;
means provided in said server for transmitting said pseudo random number to said client;
means provided in said client for encrypting desired data with said transmitted pseudo random number into encrypted data;
means provided in said client for encrypting concatenated data consisting of said pseudo random number and said encrypted data with said public key into double-encrypted data;
means provided in said client for sending said double-encrypted data to said server;
means provided in said server for decrypting said double-encrypted data with said secret key into decrypted data consisting of a decrypted random number portion and another decrypted portion; and
means provided in said server for decrypting said another decrypted portion with said transmitted random number to obtain said desired data.

38. A system as defined in claim 37, further comprising:

means, provided in said server, activated prior to said decrypting said another decrypted portion and responsive to a determination that said decrypted random number portion does not coincide with said pseudo random number stored in said means for storing said pseudo random number, for informing said client of a failure in decryption instead of passing the control to next means.

39. A system for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order and application IDs, the system comprising:

a client for playing an application by using application data; and
a server for controlling said client through a communication network, wherein said client comprises:
means for reading and sending said volume ID, said issue number and an application ID for said one of encrypting key-encrypted applications (said encrypting key-encrypted application) from said volume to said server, said server comprises:
means for proceeding to next step only if the value of a meter field associated with said volume ID, said issue number and said application ID is under the value of a limit value field associated with said volume ID, said issue number and said application ID in a volume data table; and
means for causing said client to display a message informing an overlimit on a display device of said client and quit the operation otherwise.

40. A system for permitting a user to play an application contained in a distributed application package which further contains, as volume control data, a volume ID for identifying the kind of said distributed application package (said volume), an issue number issued to each volume of the kind in an issued order, application IDs and limit values associated with respective application IDs for limiting the play of respective applications, the system comprising:

a client for playing an application by using application data; and
a server for controlling said client through a communication network, wherein said client comprises:
means for reading and sending said volume ID, said issue number, an application ID for said one of encrypting key-encrypted applications (said encrypting key-encrypted application) and a limit value associated with said application ID from said volume to said server, and wherein said server comprises:
means for proceeding to a next step only if the value of a meter field associated with said volume ID, said issue number and said application ID in a volume data table is under said limit value; and
means for causing said client to display a message informing an overlimit on a display device of said client and quit the operation otherwise.

41. A system as defined in claim 39, wherein said limit value is one of effective date and time, allowable expiration date and time, a maximum amount of play time, and an allowable access count.

42. A system as defined in any of claims 29, 33 and 34, wherein said means for sending to said server comprises means for encrypting at least one of said volume ID, said issue number and said application ID.

43. A method for permitting an authentic user to play a desired one of the applications contained in a distributed application package in a system capable of playing an application, wherein said application package (said volume) contains volume control data including mode codes assigned to said volume and the applications of said volume, the method comprising the steps of:

deciding to use one of predetermined play modes specified by one of said mode codes associated with said desired application; and
playing said desired application in said specified play mode.

44. A method as defined in claim 43, wherein the method further comprises the step of including, in said mode codes, values indicative of a free play mode and at least one limit-attached play mode which correspond(s) to respective limit value(s) used for limiting usage.

45. A method as defined in claim 44, wherein said step of playing said desired application comprises the step of:

in response to a determination that said one of said mode codes associated with said desired application includes a value indicative of said free play mode, simply playing said desired application.

46. A method as defined in claim 44, wherein said step of playing said desired application comprises the step of:

in response to a determination that said one of said mode codes associated with the desired application

includes one of values indicative of said at least one limit-attached play mode, displaying a message to the effect that a limit value associated with said one of values has been reached instead of playing said desired application if said limit value has been reached.

47. A method as defined in claim 43, wherein said volume control data further includes a volume ID, an issue number and an application ID for each of said applications, and wherein said step of deciding to use one of predetermined play modes comprises the steps of:

obtaining said one of said mode codes associated with said desired application and corresponding limit value by using said application ID; and
comparing said one of said mode codes with a meter value associated with said volume ID, said issue number and said application ID.

48. A method as defined in claim 45, wherein each of said applications has been each encrypted with an encrypting key and said volume control data includes a user's public key-encrypted version of said encrypting key (a public key-encrypted version encrypting key), and wherein said step of simply playing said desired application comprises the steps of:

reading said user's public key-encrypted encrypting key from said volume;
obtaining a user's secret key which corresponds to said user's public key;
decrypting said user's public key-encrypted encrypting key with said user's secret key to obtain said encrypting key; and
decrypting said desired application with said obtained encrypting key.

49. A system for permitting an authentic user to play a desired one of the applications contained in a distributed application package, wherein said application package (said volume) contains volume control data including mode codes assigned to said volume and the applications of said volume, the system comprising:

means for deciding to use one of predetermined play modes specified by one of said mode codes associated with said desired application; and
means for playing said desired application in said specified play mode.

50. A system as defined in claim 49, wherein the system further comprises means for including, in said mode codes, values indicative of a free play mode and at least one limit-attached play mode which correspond(s) to respective limit value(s) used for limiting usage.

51. A system as defined in claim 50, wherein said means for playing said desired application comprises:

means, responsive to a determination that said one of said mode codes associated with said desired application includes a value indicative of said free play mode, for simply playing said desired application.

52. A system as defined in claim 50, wherein said means for playing said desired application comprises:

means, responsive to a determination that said one of said mode codes associated with the desired application includes one of values indicative of said at least one limit-attached play mode, for displaying a message to the effect that a limit value associated with said one of values has been reached instead of playing said desired application if said limit value has been reached.

53. A system as defined in claim 49, wherein said volume control data further includes a volume ID, an issue number and an application ID for each of said applications, and wherein said means for deciding to use one of predetermined play modes comprises:

means for obtaining said one of said mode codes associated with said desired application and corresponding limit value by using said application ID; and
means for comparing said one of said mode codes with a meter value associated with said volume ID, said issue number and said application ID.

54. A system as defined in claim 51, wherein each of said applications has been encrypted with an encrypting key and

said volume control data includes a user's public key-encrypted version of said encrypting key (a public key-encrypted version encrypting key), and wherein said means for simply playing said desired application comprises:

means for reading said user's public key-encrypted encrypting key from said volume;
means for obtaining a user's secret key which corresponds to said user's public key;
means for decrypting said user's public key-encrypted encrypting key with said user's secret key to obtain said encrypting key; and
means for decrypting said desired application with said obtained encrypting key.

55. A method for permitting an authentic user to play a desired one of the applications contained in a distributed application package in a system comprising a client capable of playing an application and a server connected with said client through a communication network, wherein said application package (hereinafter referred to as "said volume") contains volume control data including mode codes assigned to said volume and the applications of said volume, the method comprising the steps of:

said client deciding to use one of predetermined play modes specified by one of said mode codes associated with said desired application; and
playing said desired application in said specified play mode by means of cooperation between said client and said server.

56. A method as defined in claim 55, wherein the method further comprises the step of including, in each of said mode code, a value indicative of one of a free play mode, a charged play mode and at least one limit-attached play mode, wherein said volume control data further comprises a limit value associated with each of said at least one limit-attached play mode.

57. A method as defined in claim 55 or 56, wherein said volume control data further includes a volume ID, an issue number, and an application ID for each of said applications, and wherein said step of playing said desired application in said specified play mode includes an application play step of simply playing said specified application.

58. A method as defined in claim 57, wherein each of said applications contained in a distributed application package has been encrypted with an encrypting key and said volume control data includes a user's public key-encrypted version of said encrypting key (a public key-encrypted version encrypting key), and wherein said application play step comprising the steps of:

reading said user's public key-encrypted encrypting key from said volume;
obtaining a user's secret key which corresponds to said user's public key;
decrypting said user's public key-encrypted encrypting key with said user's secret key to obtain said encrypting key; and
decrypting said desired application with said obtained encrypting key.

59. A method as defined in claim 57, wherein each of said applications contained in a distributed application package has been encrypted with an encrypting key and said volume control data includes a user's public key-encrypted version of said encrypting key (a public key-encrypted version encrypting key), and wherein said application play step comprises the steps of:

in said server,

retrieving an encrypting key by using said volume ID;
retrieving a user's public key associated with said volume ID and said issue number;
double-encrypting said encrypting key with a pseudo random number and said user's public key into a double encrypted data;
sending said double-encrypted data to said client; in said client,
obtaining a user's secret key which corresponds to said user's public key;
obtaining said encrypting key by decrypting said double-encrypted data with said user's secret key;
decrypting said desired application with said obtained encrypting key.

60. A method as defined in claim 57, wherein said step of playing said desired application further comprises the steps, executed prior to said application play step, of:

said server retrieving an expected play time associated with said desired application; and
displaying said expected play time on a display device of said client.

61. A method as defined in claim 57, wherein said step of playing said desired application further comprises the steps of:

measuring, as a measured play time, a duration of said application play step;
adding said measured play time to a play time meter associated with said mode code to obtain a total amount of play time; and
displaying said measured play time and said total amount of play time on a display device of said client after said application play step.

62. A method as defined in claim 61, wherein said step of measuring a duration comprises the step of measuring said play time by using a timer of said server.

63. A method as defined in claim 61, wherein said step of measuring a duration comprises the step of measuring said play time using a timer of said client.

64. A method as defined in claim 57, wherein said step of deciding to use one of predetermined play modes comprises deciding to use said charged play mode if said one of said mode codes associated with said desired application includes a value indicative of said charged play mode, and wherein said step of playing said desired application comprises the steps of:

said client obtaining and sending a credit card number of said user to said server;
proceeding to a next step only if the credit card of said number is found to be valid from a reference to an associated credit company;
displaying, on a display device of said client, a charge for play decided based on a measurement of a duration of said application play step and a total amount of play charges after said application play step; and
said server charging said play to said credit card number.

65. A method as defined in claim 64, wherein said step of playing said desired application further comprises the steps, prior to said application play step, of:

displaying, prior to said application play step, an expected charge and an expected total amount of charges on said display device; and
letting the user decide whether to play said desired application.

66. A method as defined in claim 64, wherein said step of said client obtaining and sending a credit card number of said user to said server comprises the steps of:

in said server,

generating a pseudo random number;
storing said pseudo random number in memory;
transmitting said pseudo random number to said client;

in said client,

prompting said user to input said credit card number;
double-encrypting said credit card number first with said transmitted random number and then with a sever's public key included in said volume control data into a double-encrypted number;
sending said double-encrypted number to said server; in said server,
decrypting said double-encrypted number with a server's secret key into a decrypted random number and another decrypted data; and
decrypting said another decrypted data with said transmitted random number to obtain said credit card number.

67. A method as defined in claim 66, wherein said step of said client obtaining and sending a credit card number of said

user to said server further comprises the steps, executed prior to said step of decrypting said another encrypted data, of:

proceeding to a next step only if said decrypted random number coincides with said pseudo random number which has been stored in said memory; and
displaying a message informing a failure in decryption and quitting the operation otherwise.

68. A method as defined in claim 57 wherein said step of deciding to use one of predetermined play modes comprises deciding to use one of said at least one limit-attached play mode if said one of said mode codes associated with said desired application includes a value indicative of said one of said at least one limit-attached play mode, and wherein said step of playing said desired application comprises the step of:

in response to a determination that a meter value associated with said one of said mode codes associated with said desired application in a record identified by said volume ID, said issue number and an application ID of said desired application in a volume data table has reached a limit value associated with said mode code, displaying a message informing an overlimit on a display device of said client instead of executing said application play step.

69. A method as defined in claim 68, wherein said limit value is one of effective date and time, allowable expiration date and time, a maximum amount of play time, and an allowable access count.

70. A system for playing a distributed application package in one of predetermined play modes in concert with a server, wherein the application package contains a data set encrypted with an encrypting key (a K-encrypted data set) for each of at least one application and volume control data for use in controlling operation of the system and the server and the volume control data includes mode codes defining said play modes, the system comprising:

means for permitting a user to select one of said at least one application of said volume;
means for deciding to use one of said predetermined play modes associated with one of said mode codes assigned to said selected application; and
means for playing said selected application in said selected play mode in concert with said server.

71. A system as defined in claim 70, wherein each of said mode codes includes one of values for a free play mode, a charged play mode and at least one limit-attached play mode.

72. A system as defined in claim 70, wherein said volume control data further includes a volume ID, an issue number and an application ID for each of said applications, and wherein said means for playing said selected application in said selected play mode at least comprises:

means for setting said server for said selected play mode by sending to said server said volume ID, said issue number, and the application ID and said mode code associated with said selected application; and
application play means for simply playing said specified application.

73. A system as defined in claim 72, wherein said volume control data further includes a user's public key-encrypted encrypting key, and wherein said application play means comprises:

means for reading said user's public key-encrypted encrypting key from said volume;
means for obtaining a user's secret key which corresponds to said user's public key;
means for decrypting said user's public key-encrypted encrypting key with said user's secret key to obtain said encrypting key; and
means for decrypting the K-encrypted data set of said selected application with said obtained encrypting key.

74. A system as defined in claim 73, wherein means for decrypting said user's public key-encrypted encrypting key and said means for decrypting the K-encrypted data set are realized as an integrated circuit.

75. A system as defined in claim 72, wherein said application play means comprises:

means for receiving double-encrypted data from said server;
means for obtaining a user's secret key which corresponds to said user's public key;

means for obtaining said encrypting key by decrypting said double-encrypted data with said user's secret key; and

means for decrypting the K-encrypted data set of said selected application with said obtained encrypting key.

76. A system as defined in claim 75, wherein means for obtaining said encrypting key and said means for decrypting the K-encrypted data set are realized as an integrated circuit.

77. A system as defined in claim 74 or 76, wherein said integrated circuit is incorporated into said means for obtaining a user's secret key.

78. A system as defined in claim 73, wherein said means for deciding to use one comprises means for deciding to use a free play mode and wherein said means for playing said selected application further comprises: means, prior to said application play means, of:

means for receiving data from said server; and
displaying said data as an expected play time for said selected application.

79. A system as defined in claim 73, wherein said means for deciding to use one of said predetermined play modes comprises means for deciding to use a free play mode, and wherein said means for playing said selected application further comprises:

means for causing said server to obtain, as a measured play time, data of a operation period of said application play means;
means for receiving first and second data from said server; and
means for displaying, just after the completion of operation by said application play means, said first and second data as said measured play time and a total amount of play time. data as said measured play time and a total amount of play time.

80. A system as defined in claim 79, wherein said means for causing said server to obtain data of said operation period comprises means for informing said server of the start and the end of operation by said application play means to utilize a timer of said server.

81. A system as defined in claim 79, wherein said means for causing said server to obtain data of a operation period comprises:

means for measuring said operation period of said application play means; and
means for sending said operation period to said server for use in a calculation of said total amount of play time.

82. A system as defined in claim 72, wherein said means for deciding to use one comprises means for deciding to use a charged play mode and wherein said means for playing said selected application further comprises:

means for obtaining and sending a credit card number of said user to said server;
means responsive to a verification result of said credit card from said server for starting a next process only if said result is positive; and
means for displaying a charge for play decided based on a measured play time of said application play means and a total amount of play charges after operation of said application play means.

83. A system as defined in claim 82, wherein said means for playing said selected application further comprises:

means activated prior to operation of said application play means for displaying an expected charge and an expected total amount of charges and letting the user decide whether to play said selected application.

84. A system as defined in claim 82, wherein said volume control data of said distributed application package further includes a sever's public key, and wherein said means for obtaining and sending a credit card number of said user to said server comprises:

means for prompting said user to input said credit card number;
means for receiving a random number from said server;

means for obtaining said server's public key from said volume;
means for double-encrypting said credit card number first with said random number and then with said server's public key into a double-encrypted data;
sending said double-encrypted number to said server;

85. A system as defined in claim 84, wherein said means for said client obtaining and sending a credit card number of said user to said server further comprises:

means responsive to a positive result of random number check from said server for starting a next process; and
means responsive to a negative result of said random number check from said server for displaying a message indicative of a failure in said random number check and quitting the operation for said selected application.

86. A system as defined in claim 72, wherein:

said means for deciding to use one comprises means for deciding to use a limit-attached play mode; and
said sending to said server includes sending a limit value associated with said mode code, and wherein said means for playing said selected application further comprises:
means operative prior to operation of said application play means for receiving from said server a limit check result indicative of whether a limit value associated with said mode code has been reached; and
means responsive to an over limit case of said result for starting a next operation.

87. A system as defined in claim 86, wherein said limit value is one of effective date and time, allowable expiration date and time, a maximum amount of play time, and an allowable access count.

88. A system for controlling through a communication network a client device to play a distributed application package in one of predetermined play modes, wherein the application package contains a data set encrypted with an encrypting key (a K-encrypted data set) for each of at least one application and volume control data for use in controlling operation of the system and the client and the volume control data includes a volume ID, an issue number, an application ID for each of said applications, and a mode code for said volume or mode codes for said applications, the system comprising:

volume data table for storing, for each volume, said volume ID, said issue number, said mode code for said volume, and said application ID and said mode code for each of said applications;
means for receiving a service request, a volume ID, an issue number, an application ID and a mode code and other data from said client;
means for storing said received application ID, said received mode code and other data in appropriate fields of a record identified by said volume ID and said issue number;
means responsive to a determination that there is no record identified by said volume ID and said issue number in said volume data table for adding said record in said volume data table and storing said received application ID and mode code and said other data in relevant fields of said record; and
means operative on the basis of said received mode code for deciding to subsequently passing the control to means for supporting a play mode associated said received mode code.

89. A system as defined in claim 88, wherein said means for supporting a play mode at least comprises means for supporting application play means, of client, for simply playing an application identified by said received application ID, and wherein said means for supporting said application play means of said client comprises:

first means for associating a given volume ID with a corresponding encrypting key;
second means for associating both a given volume ID and issue number with a corresponding user's public key;
means for retrieving an encrypting key associated with said received volume ID from said first means;
means for retrieving a user's public key associated with said received volume ID and issue number from said second means;
means for double-encrypting said encrypting key with a pseudo random number and said user's public key into a double encrypted data; and
sending said double-encrypted data to said client.

90. A system as defined in claim 89, further comprising an application data table for storing data for each kind of appli-

cation, wherein said received mode code defines a free play mode, and wherein said means for supporting a play mode associated said received mode code comprises:

means, activated prior to an operation of said means for supporting application play means of said client, for retrieving an expected play time associated with said received application ID from said application data table; and
means for sending said expected play time to said client.

91. A system as defined in claim 89, wherein said received mode code defines a free play mode, and wherein said means for supporting a play mode associated said received mode code comprises:

means for measuring, as a measured play time, a duration of application play;
means for adding said measured play time to a play time meter associated with said received mode code in said volume data table to obtain a total amount of play time; and
means for sending said measured play time and said total amount of play time to said client.

92. A system as defined in claim 91, wherein said means for measuring a duration comprises:

means responsive to a notice of the start of operation by said application play means of said client for starting a timer; and
means responsive to a notice of the end of said operation for stopping said timer.

93. A system as defined in claim 91, wherein said means for measuring a duration comprises:

means for receiving a measured duration from said client.

94. A system as defined in claim 88, wherein said received mode code defines a charged play mode, and wherein said means for supporting a play mode associated said received mode code comprises:

means for receiving a credit card number of said user from said server;
means, responsive to a determination, from a verification of said credit card number, that said credit card number is not valid, for informing said client of invalidity and quitting the operation of said means for supporting a play mode;
means, responsive to a determination, from said verification of said credit card number, that said credit card number is valid, for informing said client of a validity and proceeding to a next operation; and
means for charging said play to said credit card number.

95. A system as defined in claim 94, wherein said means for supporting a play mode associated said received mode code further comprises:

means activated prior to operation of said application play means of said client for retrieving an expected charge from said application data table by using said received application ID;
means for calculating a sum of said expected charge and a value of a charge meter associated with said received volume ID or application ID depending on said received mode code;
means operative prior to operation of said application play means for sending said expected charge and said sum to said client; and
means responsive to a receipt of a message of quitting for quitting said means for supporting a play mode.

96. A system as defined in claim 94, wherein said means for receiving a credit card number of said user from said server comprises:

means for generating a pseudo random number;
means for storing said pseudo random number in memory;
means for transmitting said pseudo random number to said client;
means for waiting for a double-encrypted data from said client;
means for obtaining a server's secret key;
means for decrypting said double-encrypted number with said server's secret key into a decrypted random number and another decrypted data; and

means for decrypting said another encrypted data with said transmitted random number to obtain said credit card number.

97. A system as defined in claim 96, wherein said means for obtaining a user's secret key comprises means for reading said user's secret key from a portable memory of said user.

98. A system as defined in claim 96, wherein said means for receiving a credit card number of said user from said server further comprises:

means responsive to a determination, made prior to said decrypting said another, that said decrypted random number coincides with said pseudo random number which has been stored in said memory for sending an enable message to said client and proceeding to a next operation; and
means responsive to a determination, made prior to said decrypting said another, that said decrypted random number does not coincide with said pseudo random number which has been stored in said memory for sending a disable message to said client and quitting said supporting a play mode.

99. A system as defined in claim 88, wherein:

said received mode code defines a limit-attached play mode; and
means for receiving a service request further receives a limit value associated with said mode code, and wherein said means for supporting a play mode associated said received mode code comprises:
means for proceeding to a next operation only if the value of a software meter associated with said mode code in said volume data table is under said limit value; and
means for sending a message informing an over limit to said client and quitting the operation of said means for supporting a play mode associated said received mode code if the value of a software meter associated with said mode code in said volume data table is not under said limit value.

100. A system as defined in claim 99, wherein said limit value is one of effective date and time, allowable expiration date and time, a maximum amount of play time, and an allowable access count.

101. A system as defined in any of claims 54, 73 and 75, wherein said means for obtaining a user's secret key comprises means for reading said user's secret key from a portable memory of said user.

102. A system as defined in claim 28 or 29, wherein said means for obtaining said secret key comprises means for reading said user's secret key from a portable memory of said user.

103. A method as defined in any of claims 10, 11, 19, 21, 22 and 55, wherein said application package is recorded on a package media.

104. A method as defined in claim 103, wherein said package media is of a write-once type, and said client is a system capable of playing said package media of said write-once type.

105. An application package as defined in claim 1, wherein said package media is distributed to a purchaser thereof or a subscriber thereof via a transmission media.

106. A system as defined in any of claims 28, 29, 37, 39, 40, 70 and 88, wherein said application package is recorded on a package media.

107. A system as defined in claim 106, wherein said application package is recorded on a package media of a write-once type.

108. A system as defined in claim 106, wherein at least a part of said volume control data is recorded, after manufacturing said package media, in an area different from a data area where said at least one application is recorded.

109. A system as defined in claim 108, wherein said client is a system provided with means for playing said package media of said write-once type.

110. A system as defined in any of claims 28, 29, 37, 39, 40, 70 and 88, wherein said application package is recorded

on a DVD and at least a part of said volume control data is recorded, after manufacturing said package media, in a BCA (burst cutting area) of the DVD, and wherein said client is a system provided with means for playing said DVD.

111. A method as defined in any of claims 10, 11, 19, 21, 22, 43 and 55, wherein the application package has been distributed to a purchaser thereof or a subscriber via a transmission media and at least a part of said volume control data has been added to said application package after preparing said application package.

112. A system as defined in any of claims 28, 29, 37, 39, 40, 49, 70 and 88, wherein said application package has been distributed to a purchaser thereof or a subscriber thereof via a transmission media and at least a part of said volume control data has been added to said application package after preparing said application package.

FIG. 1

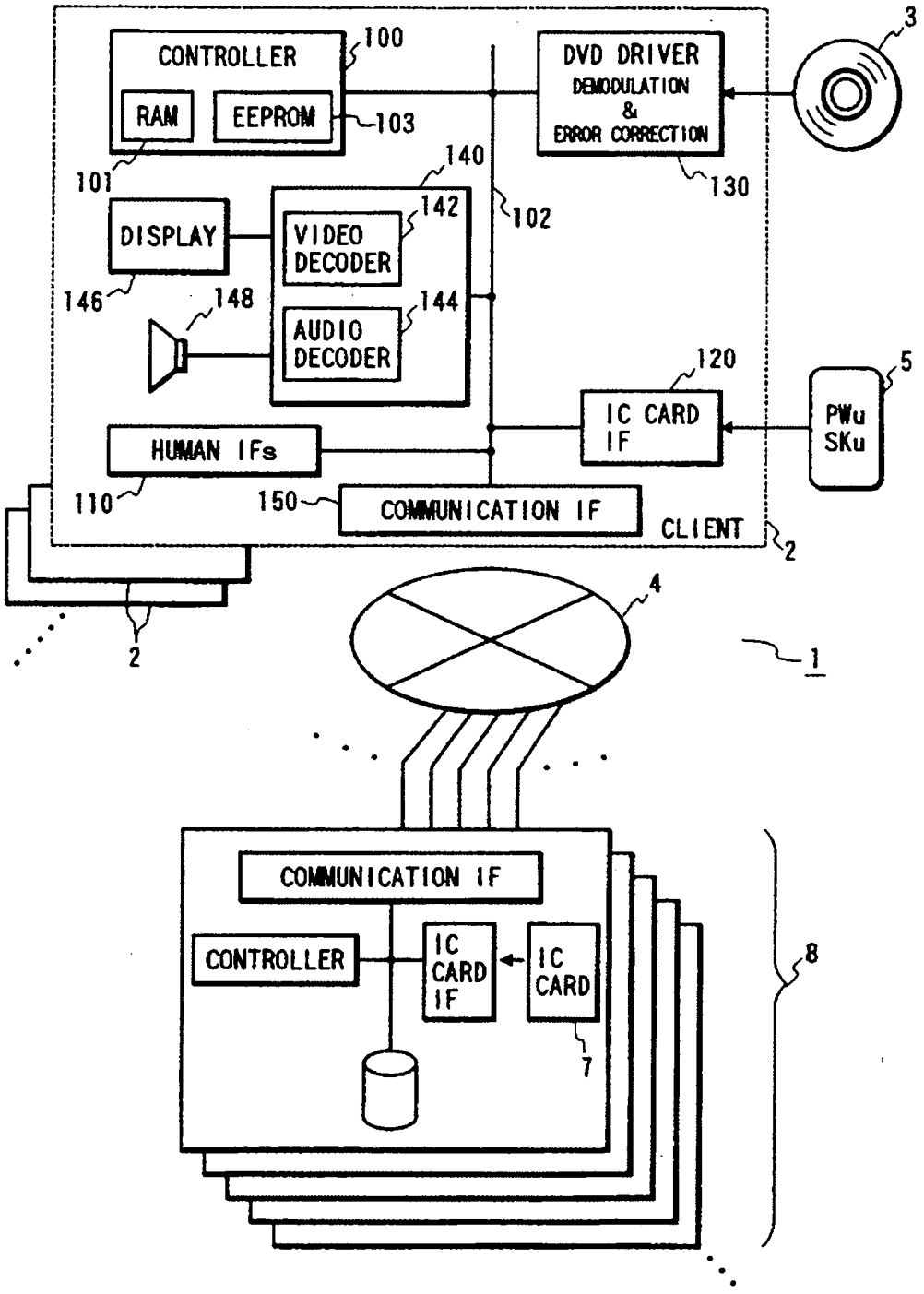


FIG. 2

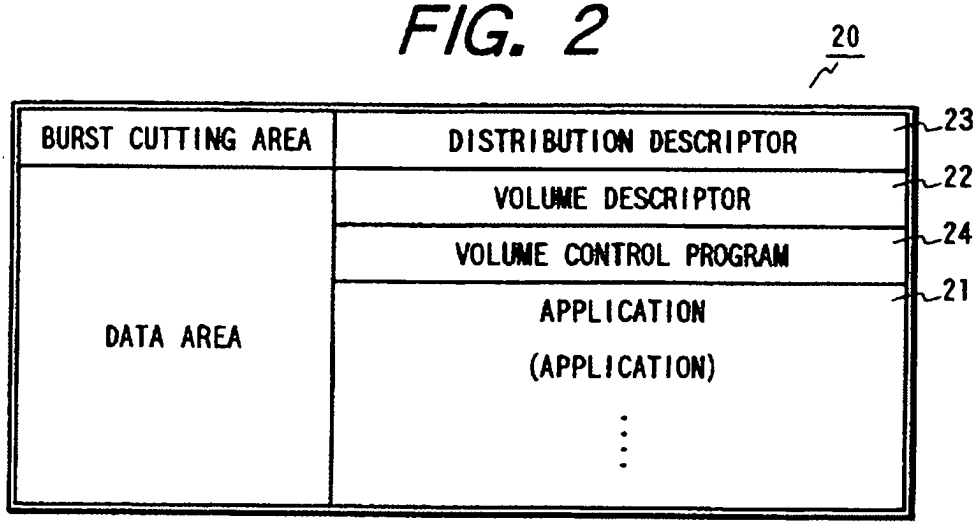


FIG. 3

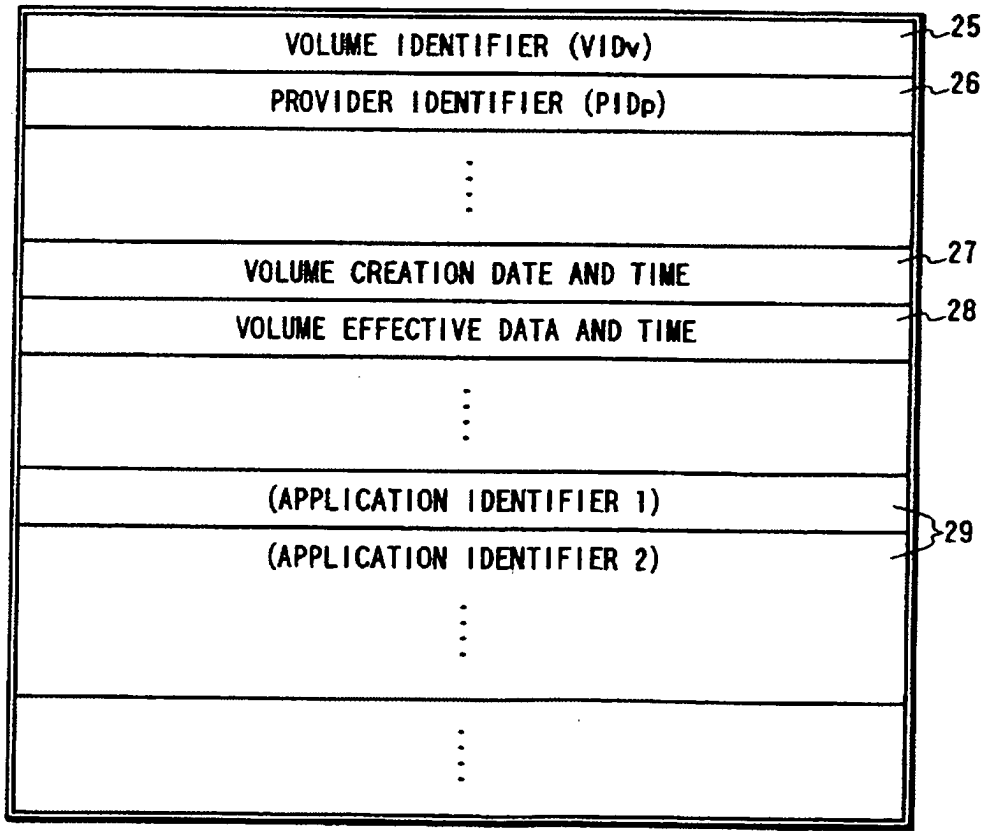


FIG. 4

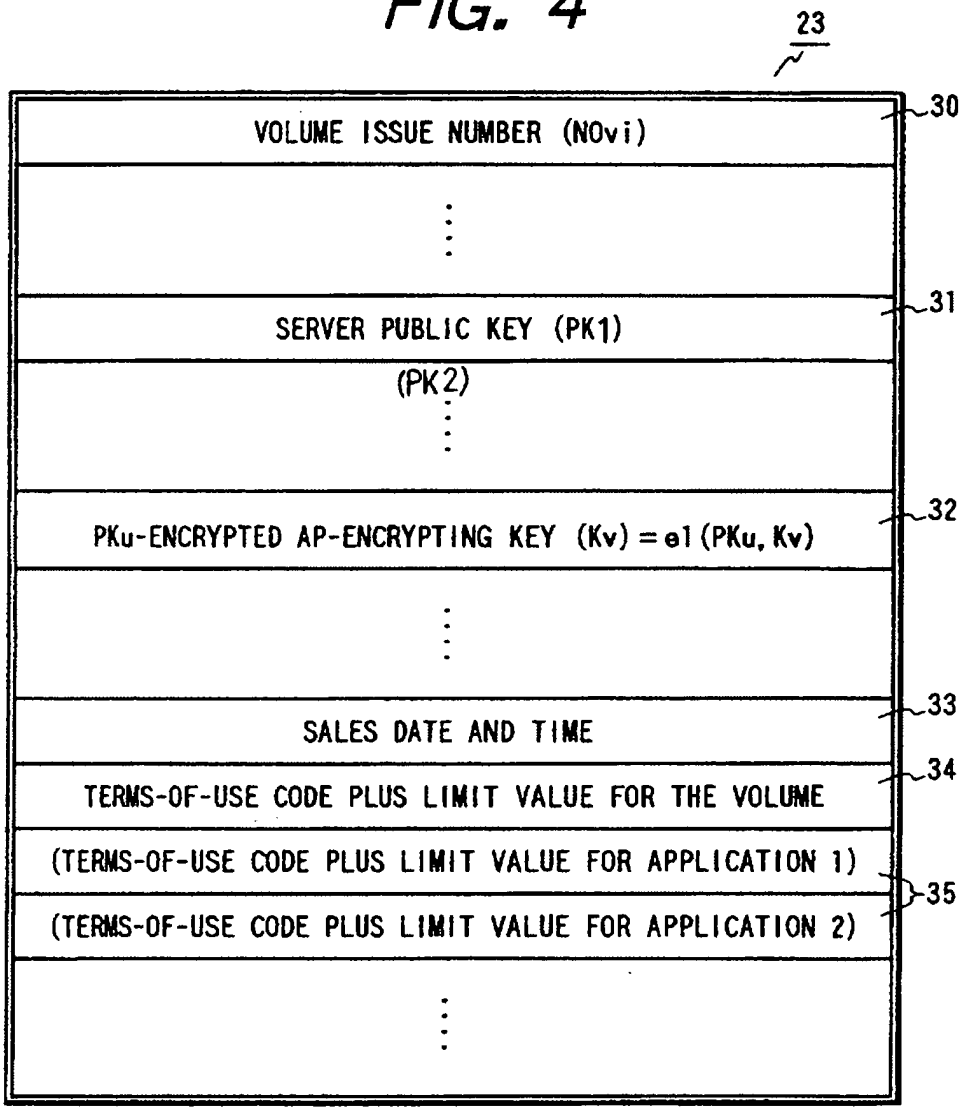


FIG. 5

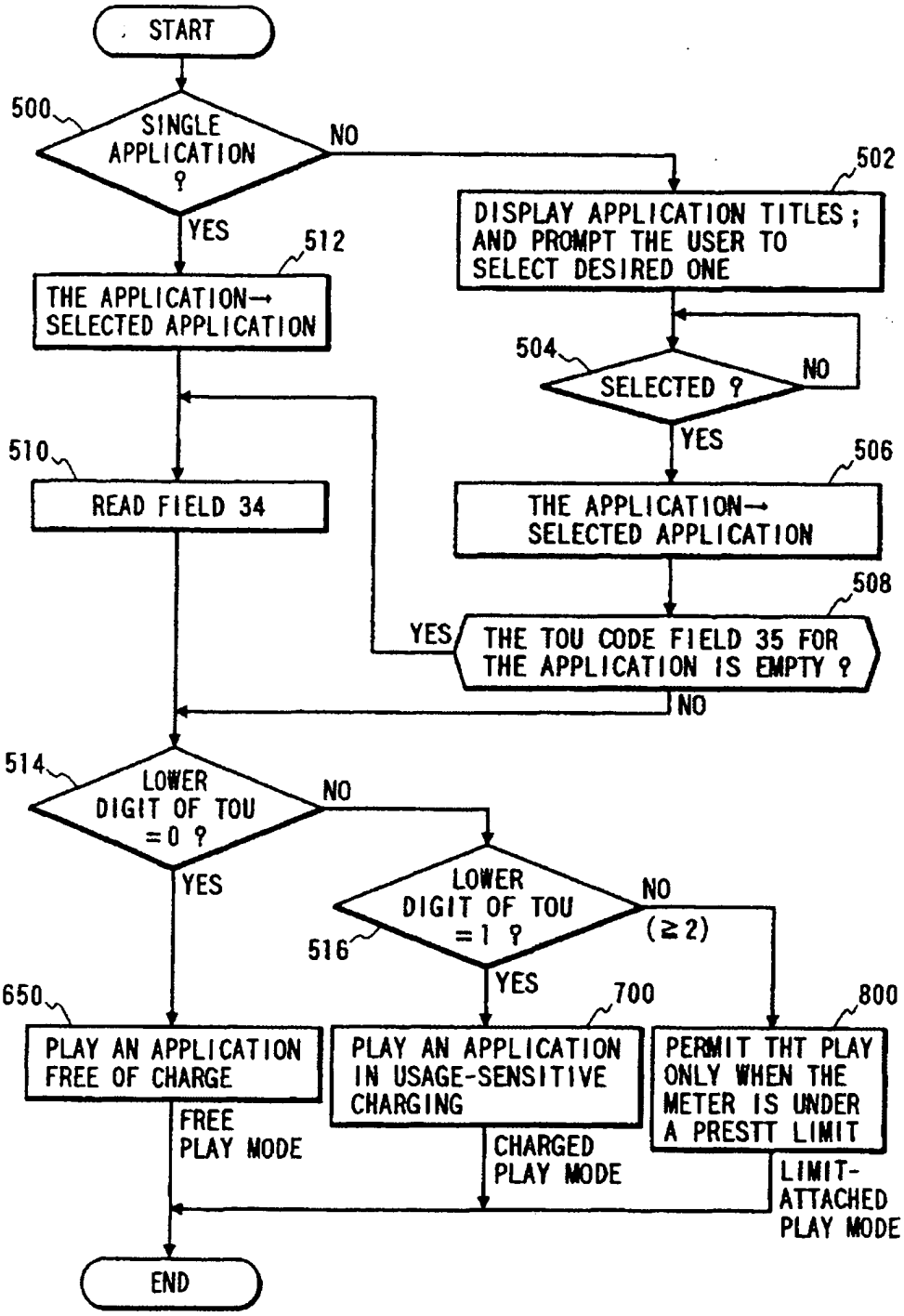


FIG. 6A

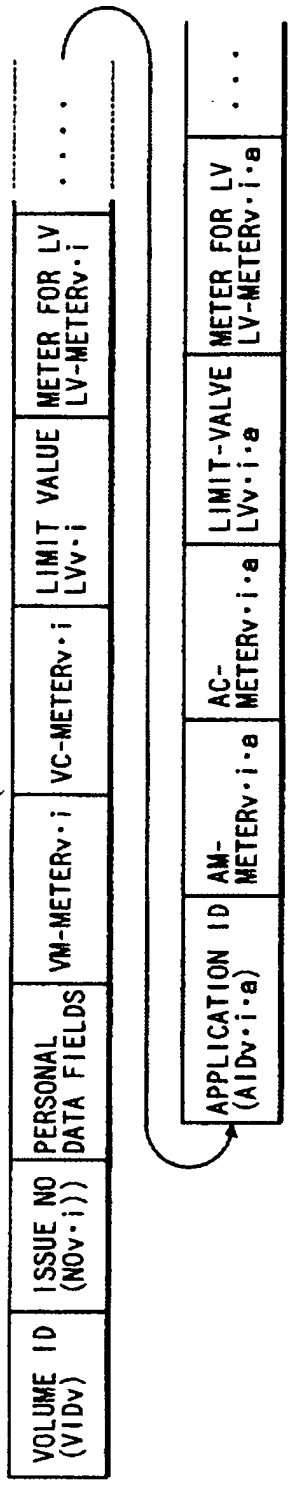


FIG. 6B

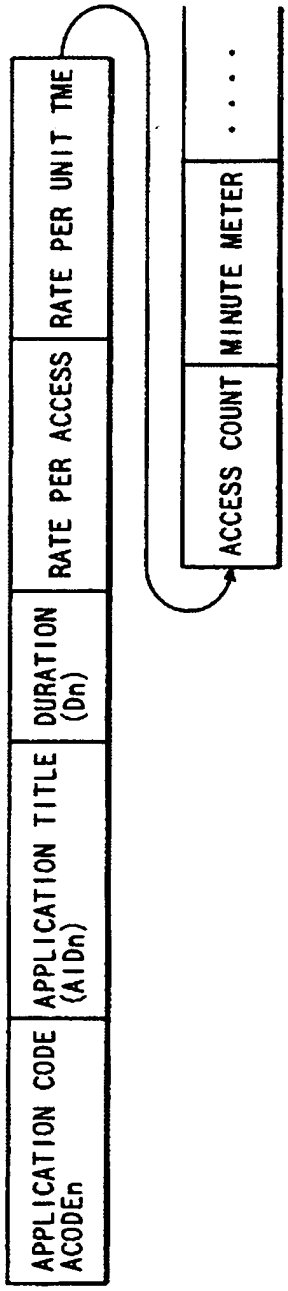


FIG. 7

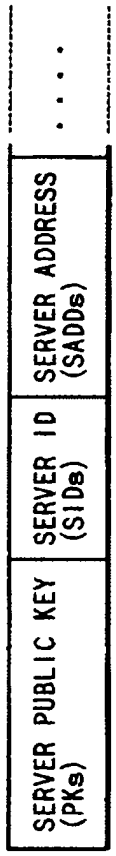


FIG. 8A

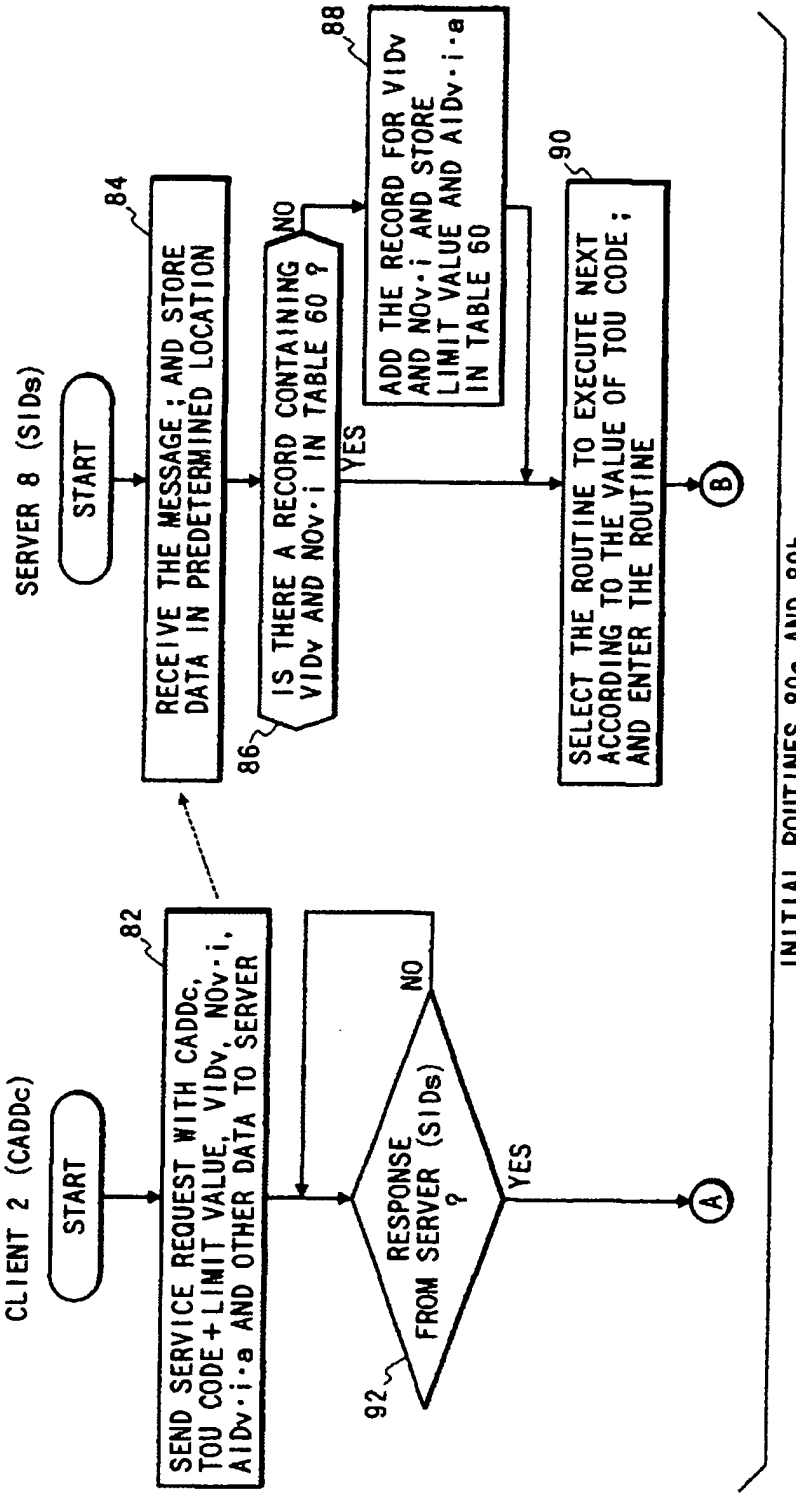


FIG. 8B

FIG. 9

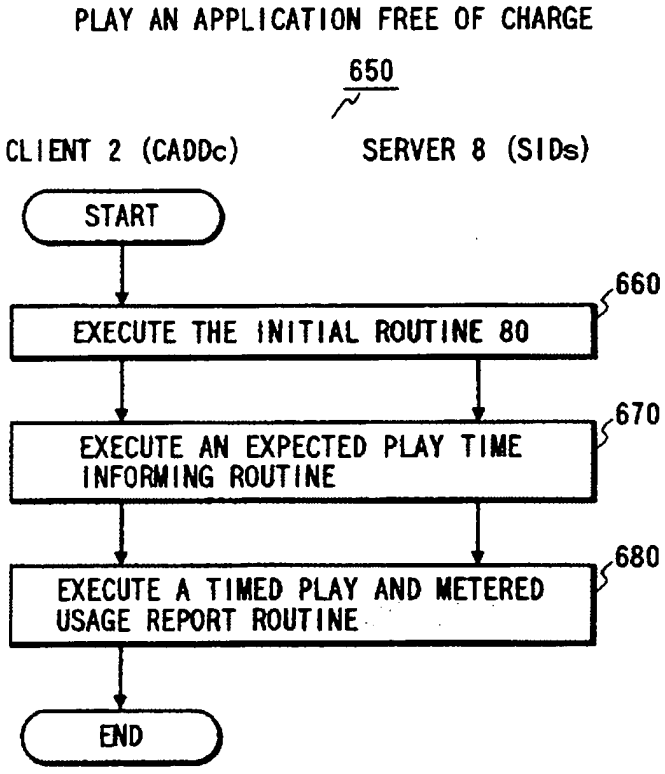


FIG. 10A

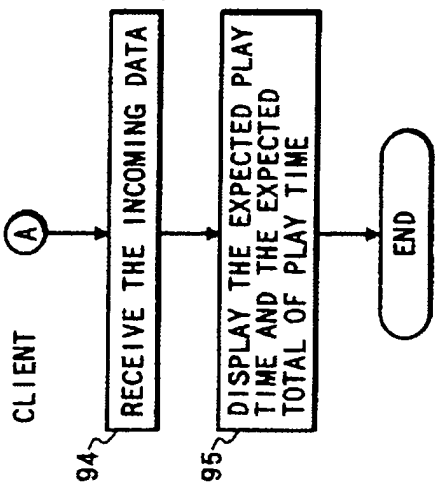


FIG. 10B

FIG. 11A

TIMED PLAY AND METERED USAGE REPORT ROUTINES 675a AND 675b

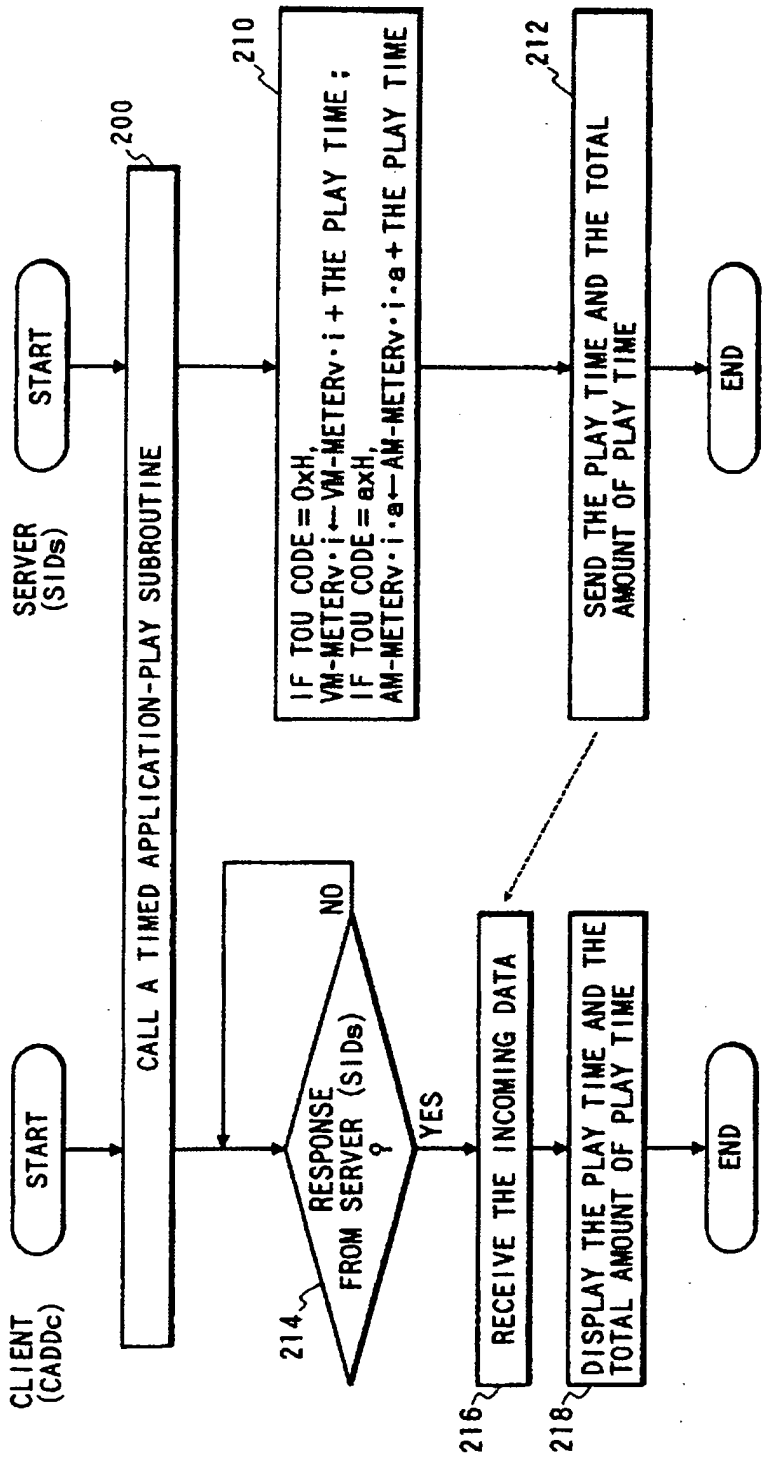


FIG. 11B

FIG. 12A

FIG. 12B

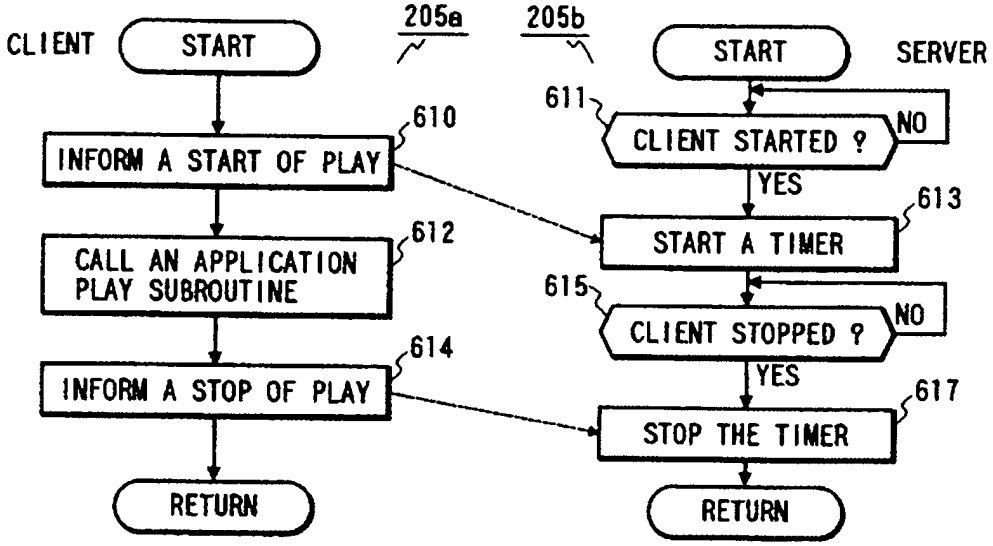


FIG. 13A

FIG. 13B

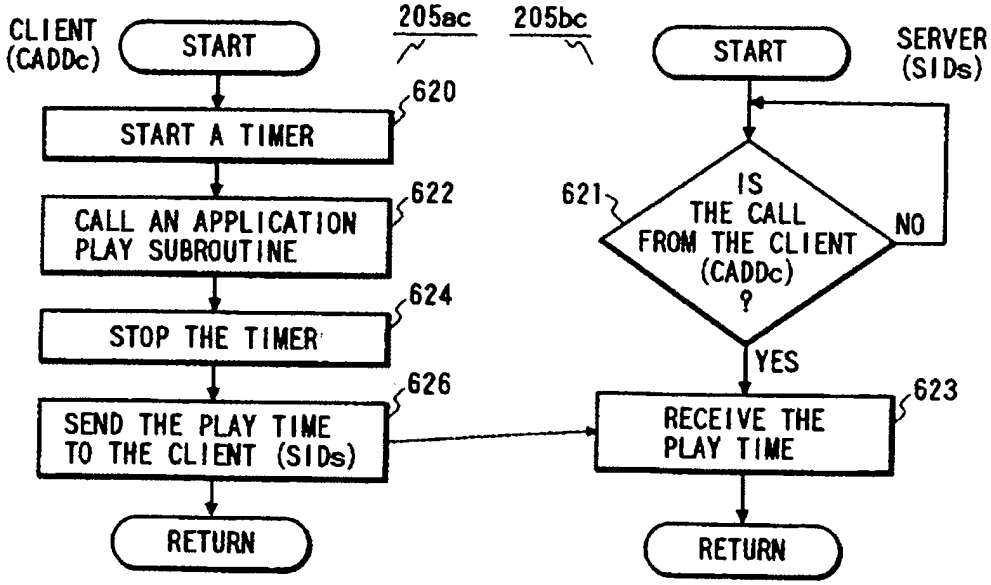


FIG. 14

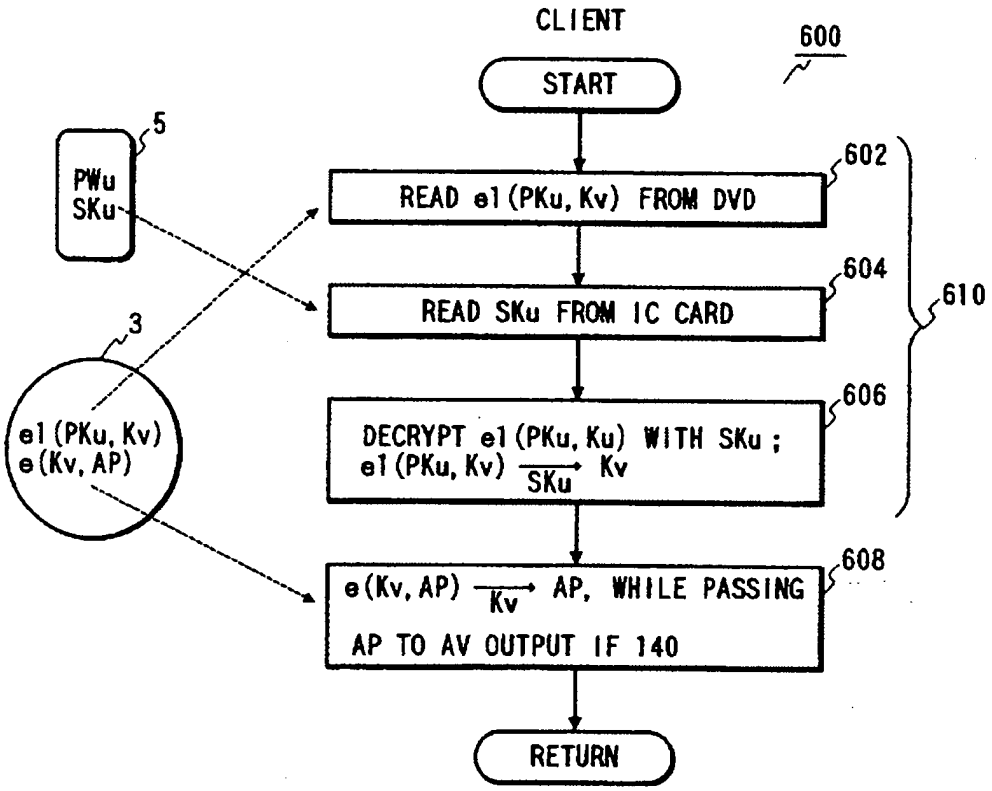


FIG. 15

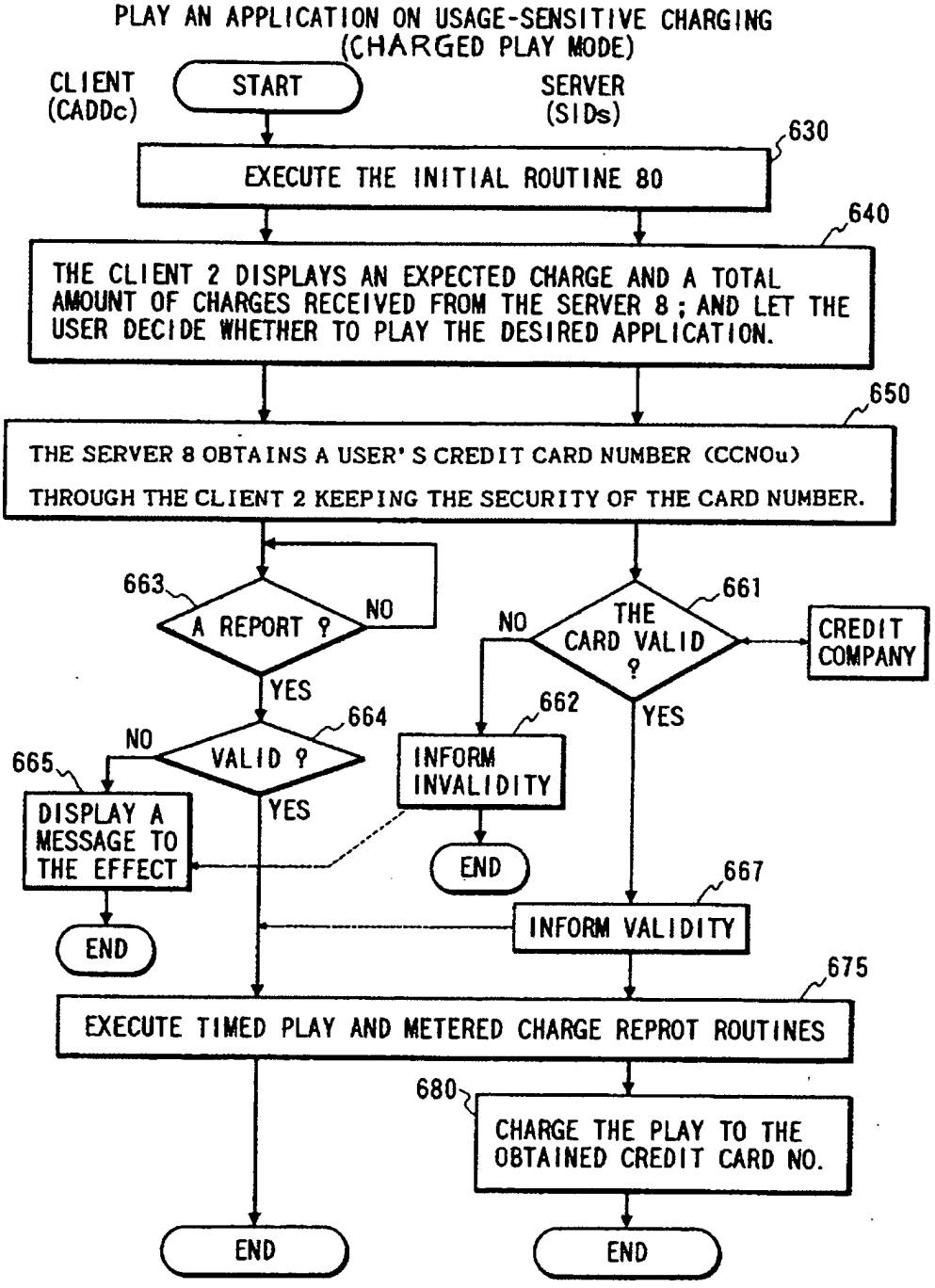


FIG. 16A

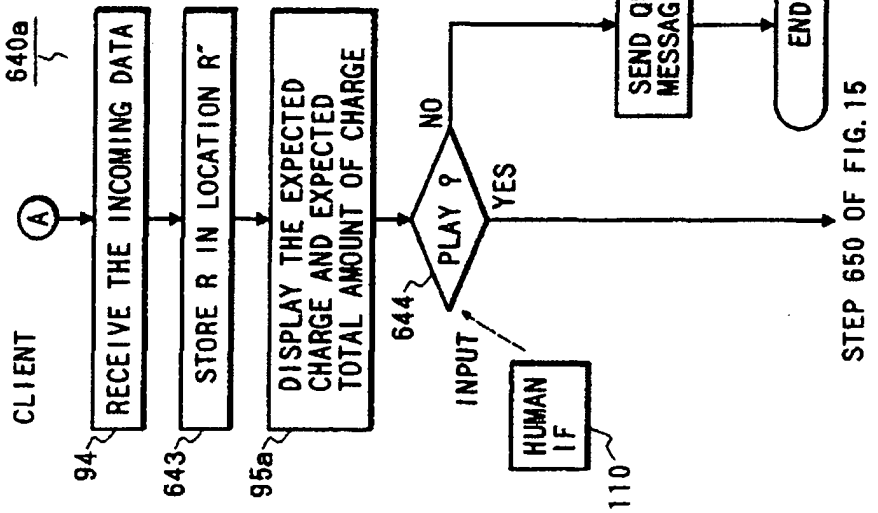


FIG. 16B

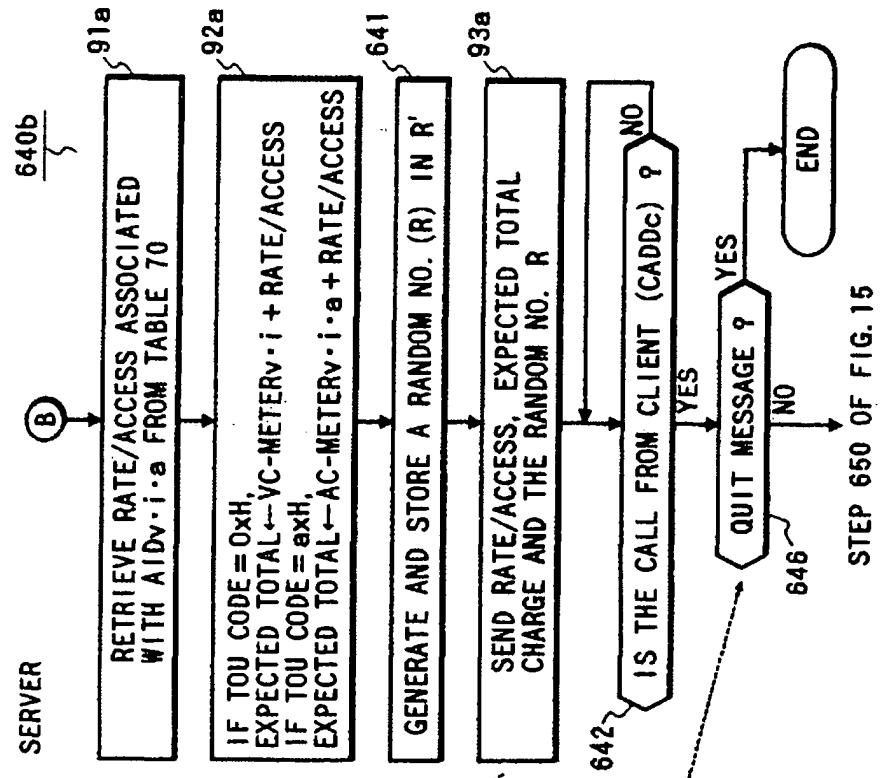


FIG. 17A

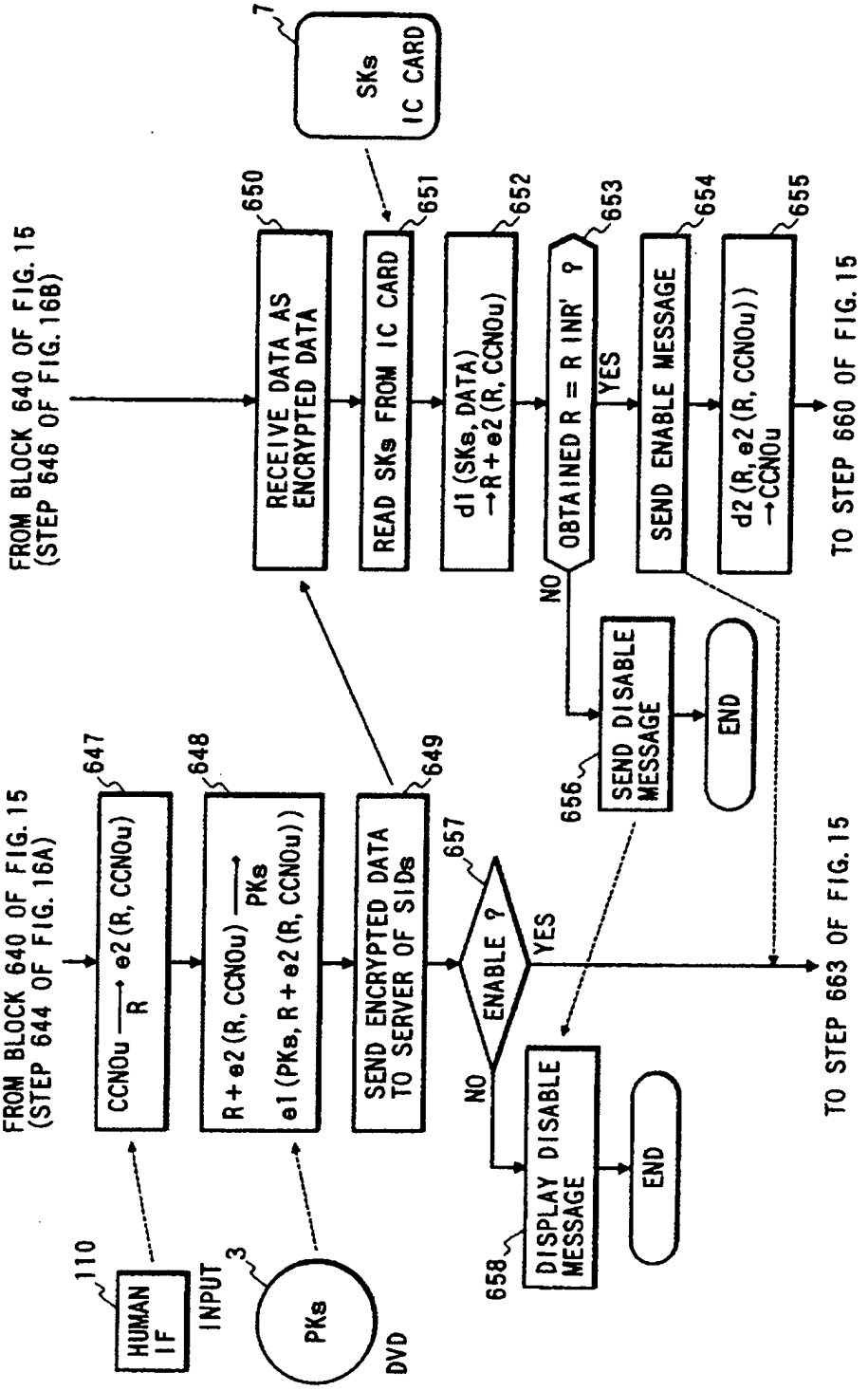


FIG. 17B

FIG. 18A

TIMED PLAY AND METERED CHARGE REPORT ROUTINES

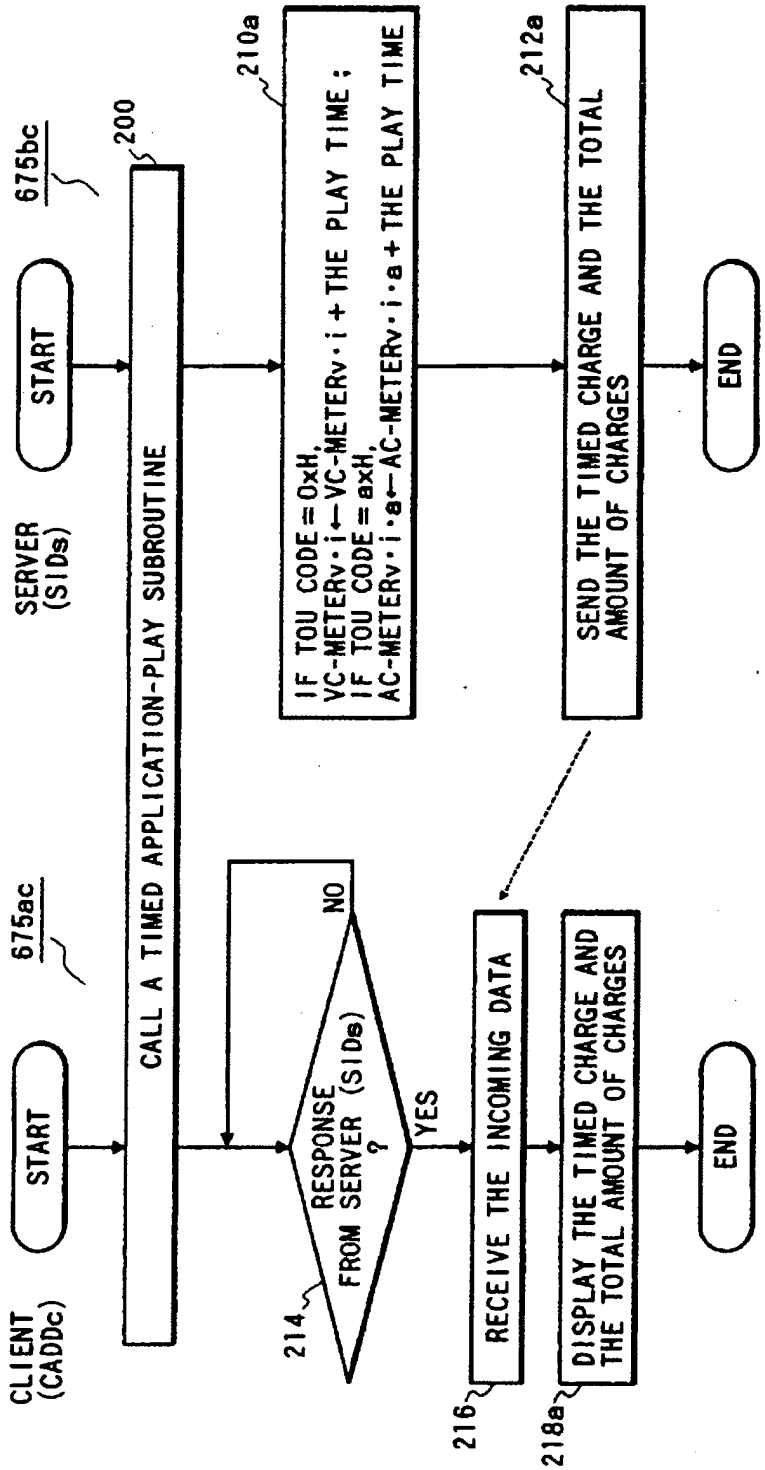


FIG. 18B

FIG. 19

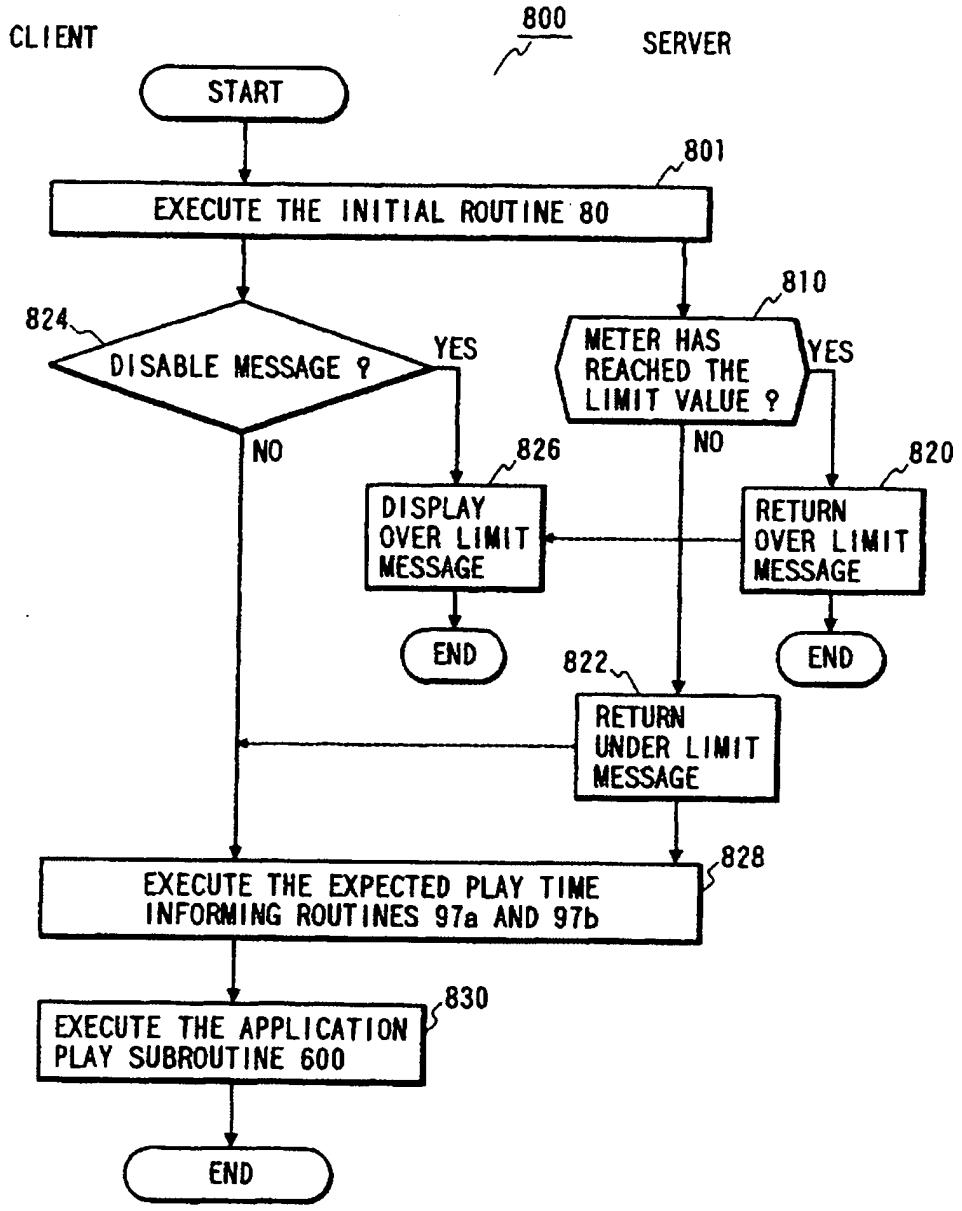


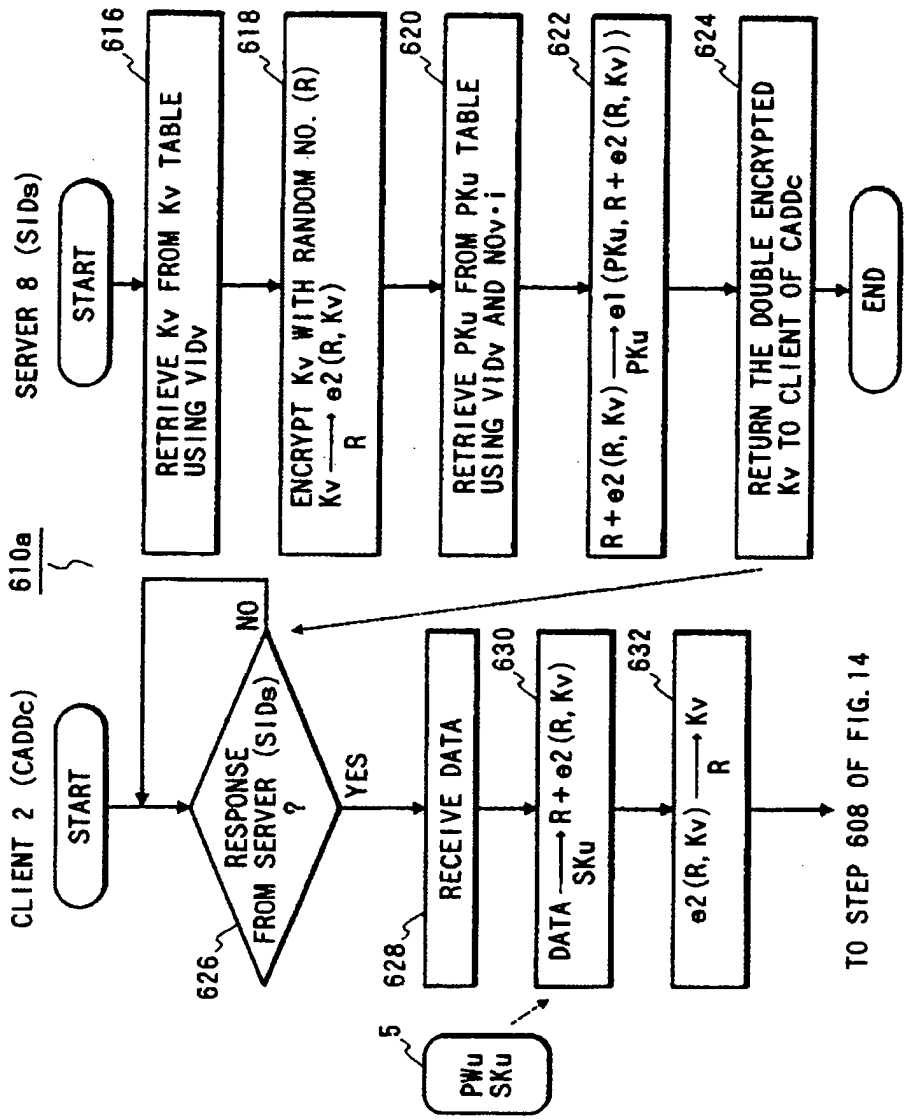
FIG. 20A

VIDv	Kv
VID1	K1
VID2	K2
...	...

FIG. 20B

VIDv	NOv·i	PKu
VID1	NO1·1	PK347020
	NO1·2	PK001031
	...	...
	NO1·365	PK314162
VID2	NO2·1	PK141421
...	...	...
	NO2·77	PK789012
VID3	NO3·1	PK123456
...	...	...

FIG. 20C



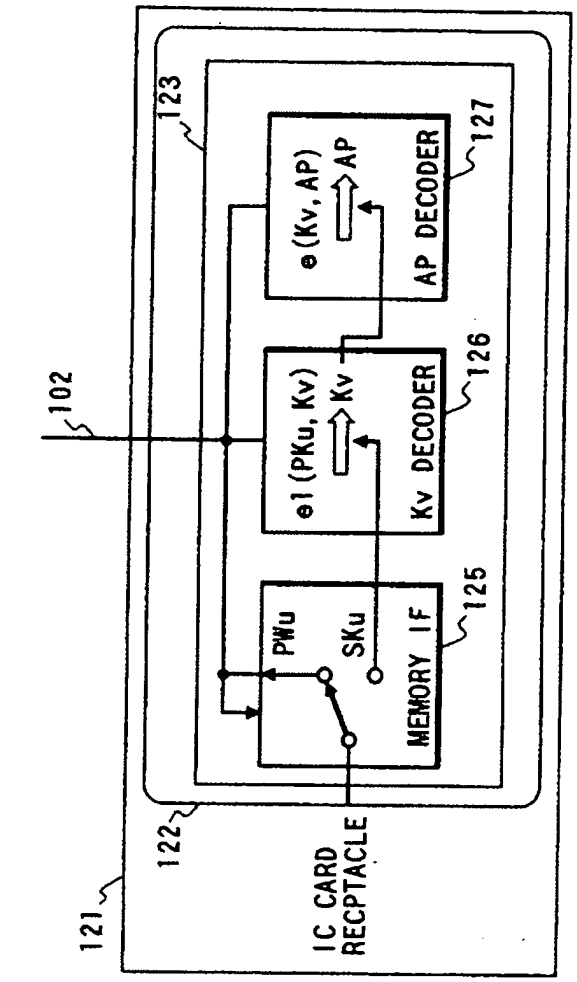
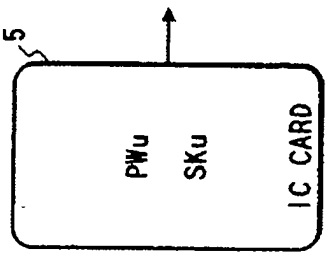


FIG. 21



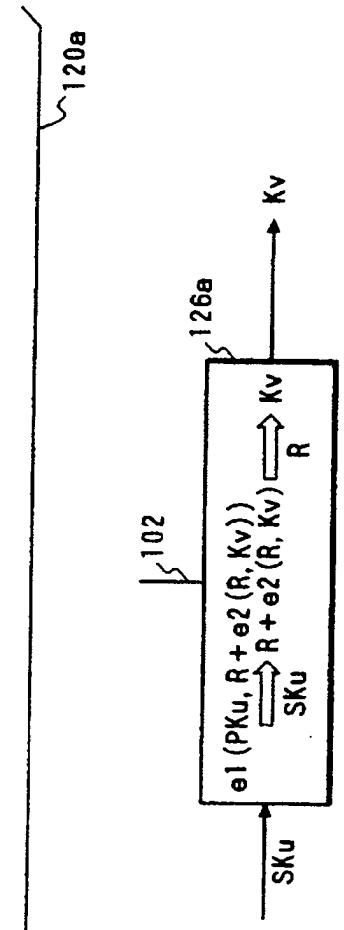


FIG. 22

FIG. 23

THE HIGHER DIGIT OF TERMS-OF-USE CODE (HEXADECIMAL)	THE TERMS-OF-USE CODE IS APPLIED TO :
0	THE ENTIRE VOLUME
1	APPLICATION 1
2	APPLICATION 2
⋮	⋮

36

XYH (X, Y = 1, 2, ···, F)

THE LOWER DIGIT OF TERMS-OF-USE CODE (HEXADECIMAL)	CORRESPONDING LIMIT VALUE	
0	NONE	(FREE)
1	NONE	(CHARGED)
2	THE EFFECTIVE DATE AND TIME	
3	THE ALLOWABLE EXPIRATION DATE AND TIME	
4	THE MAXIMUM AMOUNT OF USED PERIOD	
5	THE ALLOWABLE ACCESS COUNT	
⋮	⋮	

37

FIG. 24

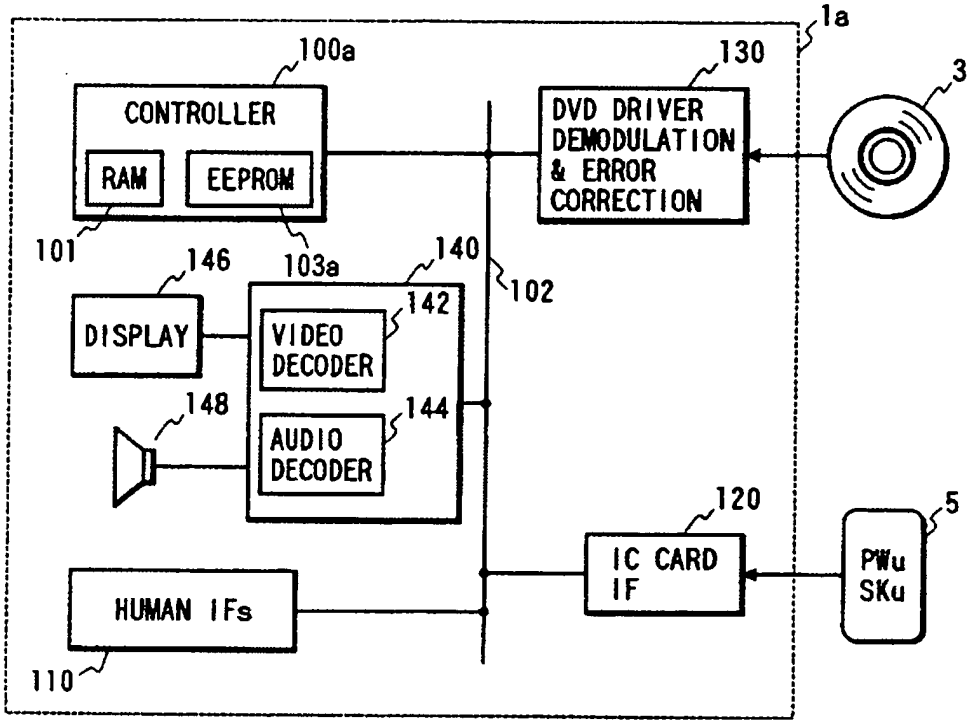


FIG. 26

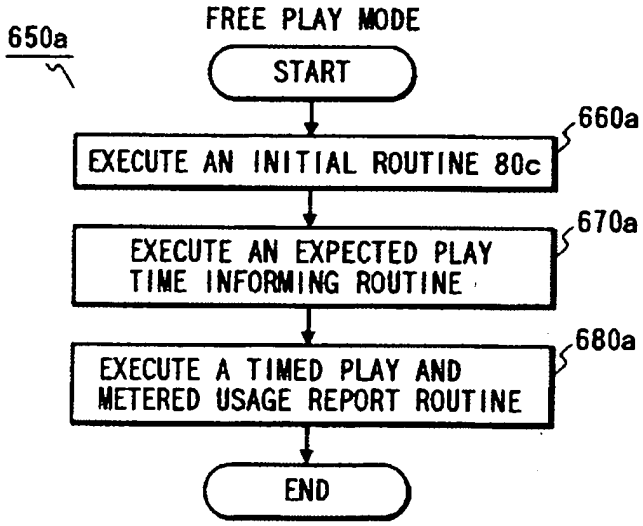


FIG. 25

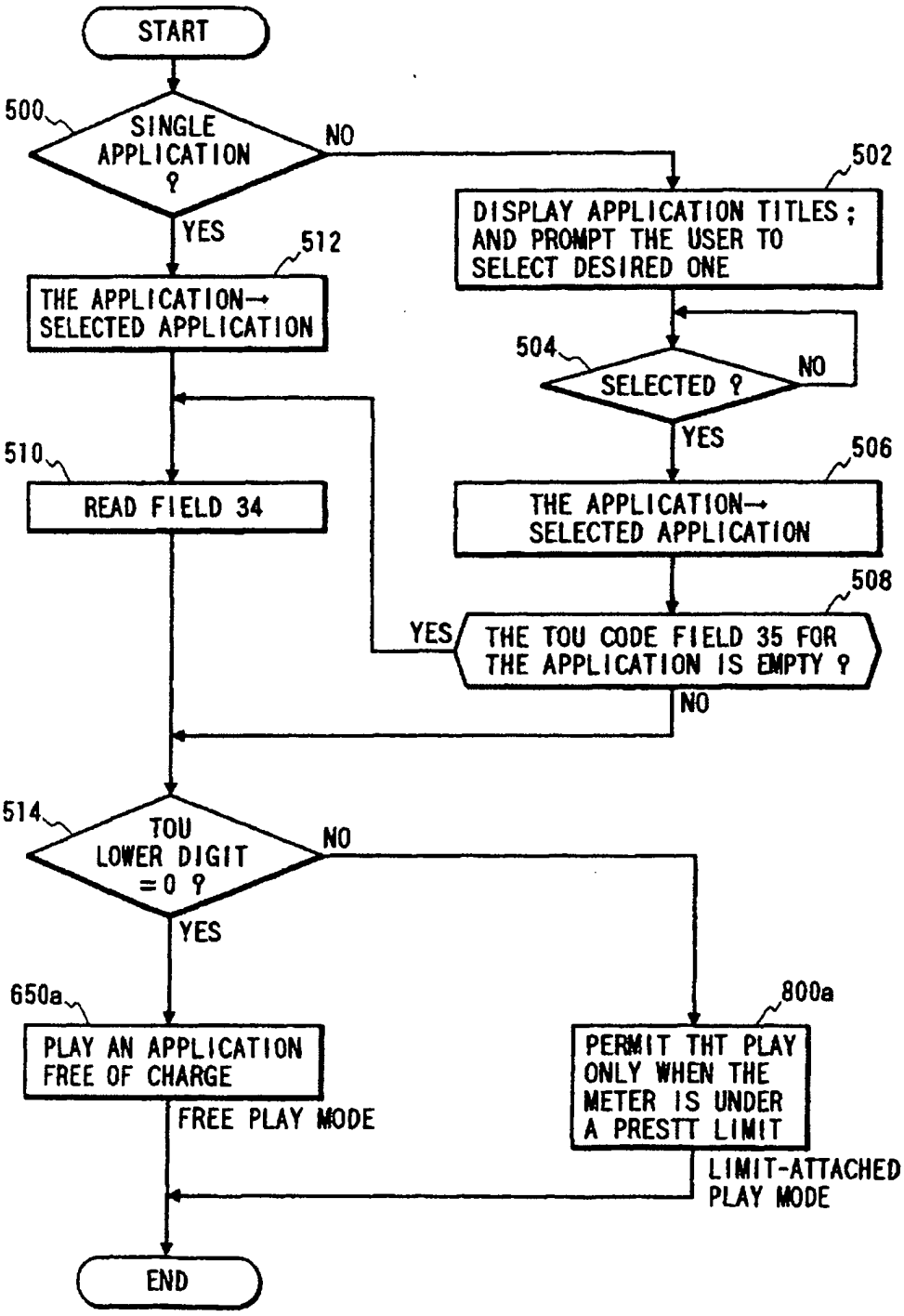


FIG. 27

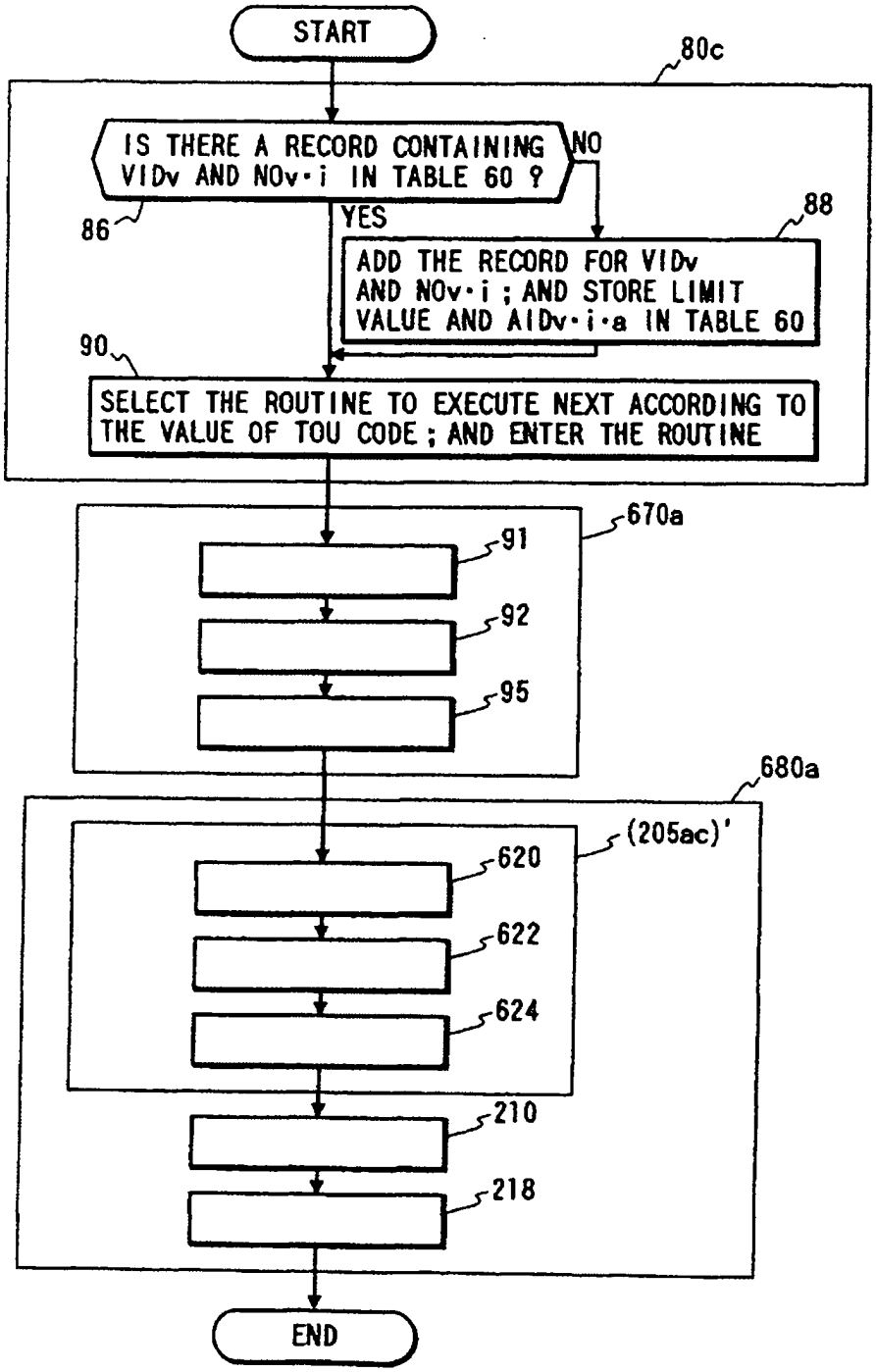
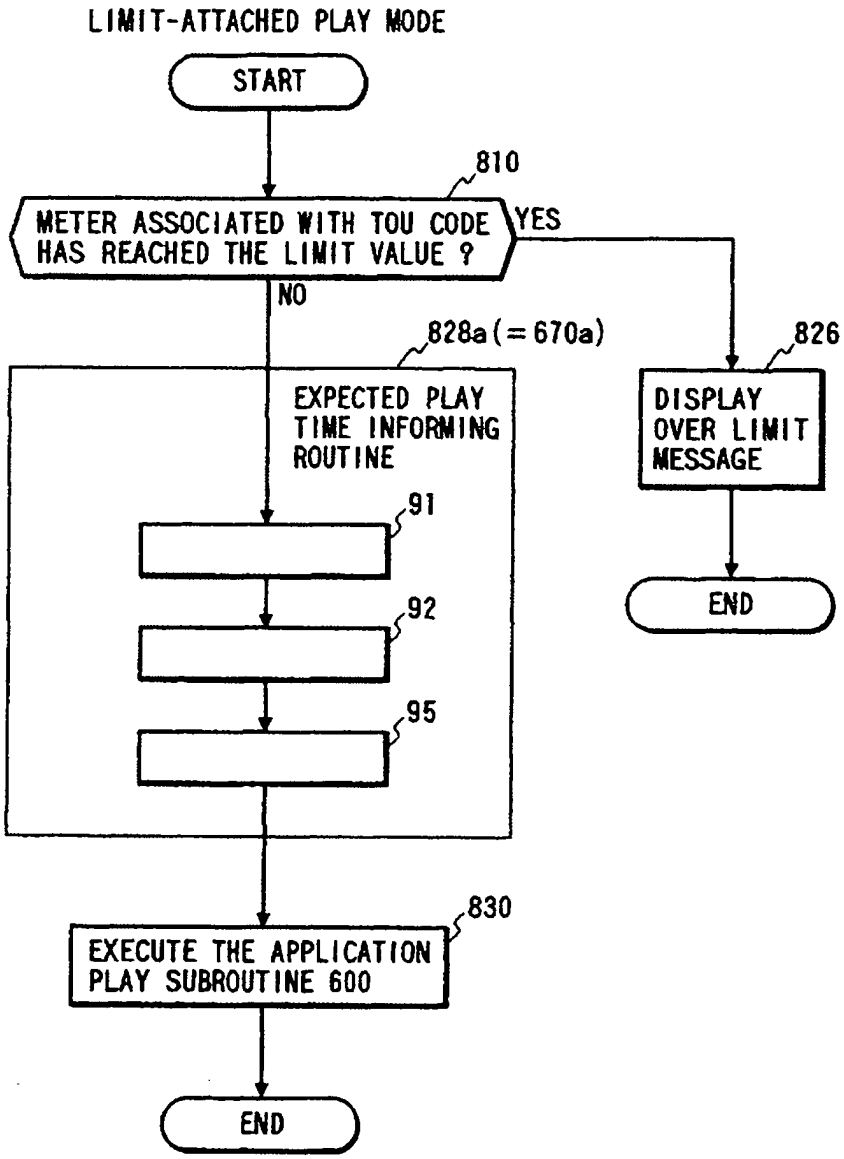


FIG. 28



(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 892 521 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
20.01.1999 Bulletin 1999/03

(51) Int Cl.[6]: **H04L 9/32**

(21) Application number: **98305646.6**

(22) Date of filing: **15.07.1998**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **15.07.1997 US 892792**

(71) Applicant: **Hewlett-Packard Company**
Palo Alto, California 94304 (US)

(72) Inventor: **Zamek, Steven**
Palo Alto, California 94303 (US)

(74) Representative: **Jehan, Robert et al**
Williams, Powell & Associates,
4 St Paul's Churchyard
London EC4M 8AY (GB)

(54) Method and apparatus for long term verification of digital signatures

(57) The time over which a digital signature can be verified is extended well beyond the expiration of any or all of the certificates upon which that signature depends. In a "save state" approach, an archive facility is used to store public key infrastructure (PKI) state, e.g. cryptographic information, such as certificates and certificate revocation lists (CRLs), in addition to non-cryptographic information, such as trust policy statements or the document itself. This information comprises all that is necessary to re-create the signature verification process at a later time. When a user wants to verify the signature on a document, possibly years later, a long term signature verification (LTSV) server re-creates the precise state of the PKI at the time the document was originally submitted. The LTSV server restores the state, and the signature verification process executes the exact process it performed (or would have performed) years earlier. Another embodiment of the invention combines the strength of cryptography with the proven resilience of (non-public key) technology and procedures currently associated with secure data stores by saving the PKI state for future verification; and protecting the PKI state information from intrusion by maintaining it in a secure storage facility which is protected by services, such as firewalls, access control mechanisms, audit facilities, intrusion detection facilities, physical isolation, and network isolation.

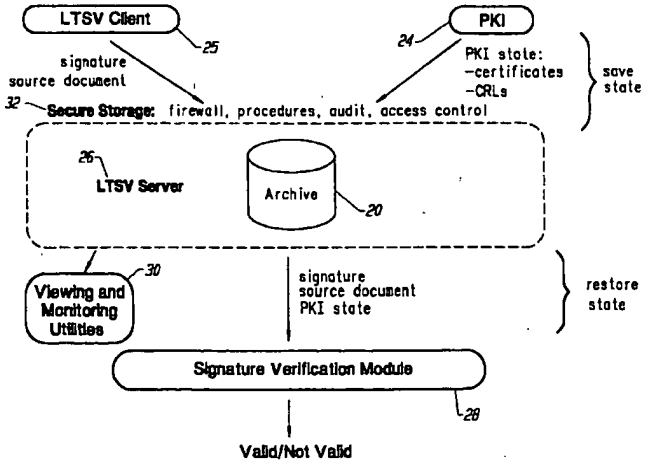


FIG. 3

EP 0 892 521 A2

Printed by Jouve, 75001 PARIS (FR)

Description

This invention relates to a method and apparatus for the long term verification of digital signatures.

The technology of digital signatures opens up the likelihood of increased use of digital networks (including the Internet) for electronic commerce. It is now feasible to send and receive digitally signed documents that represent transactions of some value to one or more parties.

Currently, a digital signature is verifiable only as long as the digital certificates upon which it depends have not expired. Given the expectation that a certificate's life span is in the area of one to two years duration, current technology does not support the emerging needs of the electronic commerce market, where the durability of digital signatures over time is a requirement.

For certain applications, the recipient of digitally signed documents should be able to verify the authenticity of a document years after the document was signed, just as the document's authenticity can be verified at the time of signing. Unfortunately, the current state of the technology does not provide for the verification of these digital signatures after certificate expiration because it is the nature of keys and certificates used for signing and encrypting documents to expire after a specific period of time (typically after a year or two). This is due, at least in part, to the fact that the strength of keys is expected to degrade over time because of such factors as improvements in computing speed and breakthroughs in cryptoanalysis. Moreover, the longer the key is in use, the longer that an adversary has to attempt to crack the key. Therefore, it is standard practice to replace keys periodically. This is why certificates have specific expiration dates.

An examination of the current state of the technology reveals that a digital signature verification module would fail if presented with a request to verify a signed document in which any of the associated certificates had expired. Fig. 1 is a block schematic diagram illustrating certification expiration. This simple example demonstrates that, given a certificate 10 having a two-year life span *(e.g.* from 4/1/96 to 4/1/98), a signature could be successfully verified six months *(e.g.* on 10/1/96) after certificate issuance (100); but this same signature would not be successfully verified three years later *(e.g.* on 4/1/99) (102). This behavior is clearly unacceptable if the duration of a document, for example contract, must extend beyond the duration of the certificates' life.

Further, some current systems use certificate revocation lists (CRLs) to revoke certificates and remove them therefrom, once those certificates expire. This means that a record of those CRLs generally disappears, making long term signature verification impossible using known techniques.

It is known to reconstruct past trust (see A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, pp. 583 (1996)). In this approach, both signature reverification relative to a past point in time and resolution of disputes may require reconstruction of chains of trust from a past point in time. This requires archival of keying material and related information for reconstruction of past chains of trust. Direct reconstruction of such past chains is taught to be unnecessary if a notarizing agent is used. A notarizing agent is defined as a general service capable not only of ascertaining the existence of a document at a certain time, but of vouching for the truth of more general statements at certain points in time. The original verification of the notary is taught to establish the existence of a trust chain at that point in time, and subsequently its record thereof is taught to serve as proof of prior validity. It is taught that details of the original trust chain may be recorded for audit purposes. It is not taught that a document can be verified based upon the existence of expired certificates. Rather, reliance is placed upon the use of the notarizing agent. It is further taught that the archived keying material can be used as evidence at a future time to allow resolution of disputed signatures by non-automated procedures.

It would be advantageous to provide a technique for extending the time over which the authenticity and integrity of digital signatures can be accurately verified beyond the time that any relevant certificates expire.

The present invention seeks to provide improved signature verification.

According to an aspect of the present invention there is provided a method of enabling long term verification of digital signatures as specified in claim 1.

According to another aspect of the present invention there is provided apparatus as specified in claim 11.

The prefered embodiment provides a method and apparatus which effectively extends the time over which a digital signature can be verified, *i.e.* well beyond the expiration of any or all of the certificates upon which that signature depends. The invention can be used for any application domain where users want digital signatures to be applied to long lasting documents *(e.g.* contracts), and be independently verifiable years or decades after signing the document. The preferred embodiment provides two alternative approaches to constructing a solution which delivers long term signature verification (LTSV).

One embodiment of the invention provides an approach for solving the LTSV problem that is referred to herein as the "save state" approach. This embodiment of the invention largely entails the use of cryptographic information and techniques. Thus, an archive facility is used to store the public key infrastructure (PKI) state, *e.g.* cryptographic information, such as certificates and CRLS, in addition to the document itself. This information comprises all that is necessary to re-create the signature verification process at a later time. It may also be desirable to store the source document separately from the cryptographic information (such as the signature, certificates, and CRLs) for reasons of privacy. For ex-

ample, a user may want to have control over the source document. The PKI state information may contain either or both of cryptographically protected information, such as certificates and CRLs, and information that is not cryptographically protected, such as the public key of a root certification authority or policy information.

When a user wants to reverify the signature on a document, possibly years later, an LTSV server re-creates the precise state of the PKI at the time the document was originally submitted. The LTSV server restores the state, and the signature verification process executes the exact process it performed (or would have performed) years earlier. The time used as the basis for re-creation of the signature verification process does not have to be the time of submittal. Rather, the time could be some other relevant time, such as when a document was signed by the originator or when it was verified by a recipient.

Another embodiment of the invention combines the strength of cryptography with the proven resilience of (non-public key) technology and procedures currently associated with secure data stores. An example of this embodiment provides a mechanism that:

- Saves the PKI state for future reverification; and
- Protects the PKI state information from intrusion by either maintaining it in a secure storage facility which is protected by services, such as firewalls, access control mechanisms, audit facilities, intrusion detection facilities, physical isolation, and network isolation; and/or employing a cryptographic protection mechanism, for example using the LTSV server to sign the PKI state information or using a keyed hash algorithm.

In addition, other non-cryptographic features may be added to such approaches to deliver a highly secure and trusted LTSV solution, including, for example utilities for viewing the PKI state information (cryptographic as well as non-cryptographic) and visually monitoring the security of the system. These utilities can be used to provide visual evidence for purposes of dispute resolution.

One enhancement to the secure storage approach herein disclosed maintains certain evidence, such as certificate chains, in an archive. This information need not be used for actual reverification, but merely as supporting evidence in case of a dispute.

An embodiment of the present invention is described below, by way of example only, with reference to the accompanying drawings, in which:

Fig. 1 is a block schematic diagram illustrating certification expiration;

Fig. 2 is a block schematic diagram illustrating a "save state" embodiment of the invention;

Fig. 3 is a block schematic diagram illustrating a "save state" "secure storage" embodiment of the invention;

Fig. 4 is a flow diagram that provides two alternative scenarios that illustrate the applicability of time stamps to the preferred embodiments;

Figs. 5a-5c provide block schematic diagrams that illustrate three long term signature verification usage scenarios;

Fig. 6 is a block schematic diagram that illustrates trust between two entities ; and

Fig. 7 is a block schematic diagram that illustrates a long term signature verification trust model.

The meanings of some of the terms used herein may differ somewhat from common usage. The following definitions are meant to clarify the meaning of each in the context of its usage herein.

Archive: Any facility for the storage and retrieval of electronic information.

Certificate: An artifact upon which digital signatures are based. A certificate securely binds an entity with that entity's public key.

Cryptographic Refresh: A means of solving the key degradation problem when storing cryptographic information for long periods of time. The process involves re-encoding the old cryptographic artifacts *(e.g.* encrypted data, digital signatures, and message digests) with stronger algorithms and/or longer keys.

Document: A document can be any information which can be represented electronically or optically *(e.g.* an arbitrary bit stream).

Key Degradation/Algorithm Degradation: The process whereby the protection afforded a document by encryption under a key loses effectiveness over time. For example, due to factors such as improvements in computing speed and breakthroughs in cryptoanalysis, it is expected that a document securely encrypted today would be crackable years later. This property could affect any cryptographic information, including digital signatures. This problem can be generalized to keyed and non-keyed cryptographic processes and artifacts, such as one-way hash algorithms. The security provided by these are also expected to diminish over time.

LTSV: Long Term Signature Verification. The herein described method and apparatus for verifying a digital signature after the certificates used for such verification have expired.

LTSV client: The entity which requests/utilizes the services of the LTSV server.

LTSV server: The entity which delivers the LTSV services. This does not imply, however, that this entity must be stand-alone component.

LTSV submission: A request from an LTSV client to

an LTSV server to perform the necessary functions required to enable reverification of a digital signature some time in the future *(e.g.* save PKI state).

PKI: Public Key Infrastructure. Refers to all components, protocols, algorithms, and interfaces required to deliver the capabilities to digitally sign and verify documents. For purposes of clarity herein, a PKI does not include a service module for long term signature verification (LTSV server), although in practice a PKI might be designed to encompass such a module.

Signature Reverification: The re-creation of the digital signature verification process after the original verification. This specifically refers to the process associated with the verification process, based upon the restoration of the previously saved PKI state.

Signature Verification: The process by which a digital signature, for a given document, is determined to be authentic or not.

Signature Verification Module: The module which is responsible for performing the verification of digital signatures.

Time stamp: A digital time stamp is an electronic indicator which associates the current date and time with a particular document. Time stamps are useful for proving that a document existed at a particular time. It is desirable that time stamps be secure, durable over time, and trusted by those using them.

The discussion herein assumes an understanding of public key, digital signatures, and PKI infrastructure using X.509 certificates. Practical information concerning application of such techniques is considered to be well known to those skilled in the art. Background information may be found, for example, in B. Schnier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons, Inc. (1996); W. Ford, M. Baum, Secure Electronic Commerce, Prentice Hall PTR (1997); and in the X.509 v.3 specification ([X.509-AM] ISO/IEC JTC1/SC 21, Draft Amendments DAM 4 to ISO/IEC 9594-2, DAM 2 to ISO/IEC 9594-6, DAM 1 to ISO/IEC 9594-7, and DAM 1 to ISO/IEC 9594-8 on Certificate Extensions, 1 December 1996). The system described herein may be built upon the X.509 infrastructure.

The following discussion provides some background on cryptographic techniques. Cryptographic algorithms can generally be divided into two categories: public key *(e.g.* RSA) and secret key *(e.g.* DES). Both types of algorithms transform plain text into cypher text using a key(s) for the encryption and decryption processes.

Both public key and secret key algorithms are considered to be secure. One is not better than another in terms of security. The strength of each algorithm, in terms of it being cracked, is largely a function of the length of the key used. The primary distinguishing characteristic of public key, however, is that it uses two keys (one to encrypt and another to decrypt), while secret key algorithms use only one key (the same key is used for encryption and decryption). For this reason, secret key algorithms are sometime referred to as symmetric algorithms and public key algorithms are called asymmetric.

One problem with secret key algorithms is that a key must be distributed between all participants. This means that some secure channel must be available for the distribution of the keys.

In practice, each entity in a public key-based system has a key pair, *i.e.* one private key and one public key. The private key is known only to its owner, the public key is known to all correspondents. It is computationally infeasible to determine a private key from the public key.

The two primary services provided by public key cryptography are secure exchange of symmetric keys (by using public key techniques to encrypt a symmetric session key), and non-repudiation via digital signatures.

Public key cryptography can be used to solve the key exchange problem associated with secret key algorithms by using this technology to encrypt the secret key under the public key of the recipient. It can then be decrypted by the recipient using his/her private key.

Digital signatures are possible by encrypting data with the private key of the signing entity. Any entity can decrypt it with the signer's publicly available public key and know that no one else could have encrypted it because that private key is only known by that one individual. This particular use of public key provides the non-repudiation service, which is a primary use of public key cryptography. A digital signature is very powerful notion, it generally exhibits the following characteristics:

- Cannot be forged;
- Is independently verifiable;
- Is not reusable or transferable to a different piece of data; and
- Includes data integrity checks, allowing tamper-detection.

The new services provided by public key cryptography do not come for free, however, because these services require the existence of a supporting public key infrastructure. The strength of a public key system depends upon the assurance that all participants know the public key of any entity with whom they wish to correspond. If a secure correspondence between a user and his/her public key cannot be maintained, then it may be possible to impersonate another entity or read encrypted data intended for another.

The standard solution to this problem is the issuance of a digital certificate (X.509 certificate) to each participant. This certificate securely binds its owner's name with his/her public key. It is issued by a trusted third party, called a certification authority (CA), and is signed by that CA, thereby making it tamper proof. Certificates are issued for a limited period of time (start and

stop dates), during which the certificate is considered valid. A certificate is considered expired after the ending validity date.

The public keys of entities (which are embedded in the X.509 certificates) must be publicly available. The distribution or access mechanisms available are numerous.

The secure operation of a public key infrastructure rests upon certain points of trust. Certainly each entity must trust its own CA. However, when a given PKI domain is expanded to encompass relationships with multiple CAs, the number of points of trust are also expanded. The trust placed in a particular end entity *(i.e.* that entity's certificate or signature) is directly related to the trust relationships among the CAs which certify those entities.

CAs can create trust relationships with other CAs by certifying each other. This can be a unidirectional trust relationship, whereby one CA can merely issues a certificate to another CA, just as a CA issues a certificate to an end user. Two CAs can also mutually agree to trust each other (bidirectional trust relationship) by issuing a cross-certificate -- a special form of certificate which contain two individual certificates, one for each direction.

If two entities are in the same CA domain, then there is no concern with respect to CA trust because they both trust the same CA. This is not the case, however, when dealing with the scenario where entities which have been certified by different CAs attempt to conduct a secure transaction. The security of this transaction depends upon the trust between the CAs. More generally, the security provided by the PKI depends upon the trust models embodied in the trust relationships among the various CAs which choose to trust one another. In concrete terms, any change in these trust relationships can cause a signature verification to either succeed or fail.

The preferred method and apparatus effectively extend the time over which a digital signature can be verified, *i.e.* well beyond the expiration of any or all of the certificates upon which that signature depends. They can be used for any application domain where users want digital signatures to be used on long lasting documents *(e.g.* contracts), and be independently verifiable years or decades after signing the document. The preferred embodiment of the invention provides two alternative approaches to constructing a solution which delivers long term signature verification (LTSV).

Fig. 2 is a block schematic diagram illustrating a "save state" embodiment of the invention. This embodiment, largely entails the use of cryptographic information and techniques. Thus, an archive facility 20 is used to store a public key infrastructure (PKI) state 24, *e.g.* cryptographic information, such as certificates and CRLS, in addition to the source document itself. For example, the LTSV client 25 requests the services of an LTSV server 26 to accomplish storage of such information. This step is shown as the "save state" step in Fig. 2. The PKI state information may contain either or both of cryptographically protected information, such as certificates and CRLs, and information that is not cryptographically protected, such as the public key of a root certification authority or policy information.

This information comprises all that is necessary to re-create the signature verification process at a later time, *i.e.* during the "restore state" step, for example, as requested by the LTSV client. It may also be desirable to store the source document separately from the cryptographic information (such as the signature, certificates, and CRLs) for reasons of privacy. For example, a user may want to have control over the source document.

When a user wants to reverify the signature on a document, possibly years later, the LTSV server 26 re-creates the precise state of the PKI at the time the document was originally submitted. The LTSV server restores the state, and the signature verification process 28 executes the exact process it performed (or would have performed) years earlier. The time used as the basis for re-creation of the signature verification process does not have to be the time of submittal. Rather, the time could be some other relevant time, such as when a document was signed by the originator or when it was verified by a recipient.

Fig. 3 is a block schematic diagram illustrating a "save state" "secure storage" embodiment of the invention. This embodiment of the invention combines the strength of cryptography with the proven resilience of (non-public key) technology and procedures currently associated with secure data stores. An example of this embodiment:

- Saves the PKI state for future reverification (as described above in connection with Fig. 2); and

- Protects the PKI state information from intrusion by maintaining it in a secure storage facility which is protected by services, such as firewalls, access control mechanisms, audit facilities, intrusion detection facilities, physical isolation, and network isolation; and/or employing a cryptographic protection mechanism, for example using the LTSV server to sign the PKI state information or using a keyed hash algorithm.

In addition, other non-cryptographic features may be added to such approach to deliver a highly secure and trusted LTSV solution, including, for example utilities 30 for viewing the PKI state information (cryptographic as well as non-cryptographic) and visually monitoring the security of the system. These utilities can be used to provide visual evidence for purposes of dispute resolution.

One enhancement to the secure storage approach herein disclosed maintains certain evidence, such as certificate chains, in an archive. This information need

not be used for actual reverification, but merely as supporting evidence in case of a dispute. See A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, pp. 583 (1996), for one manner in which this enhancement may be implemented in the context of a notary service (discussed above).

There are other embodiments of the invention in which a hybrid LTSV solution could be constructed by combining cryptographic and non-cryptographic techniques. The best combination for a particular application domain depends upon the security requirements of the application(s), in combination with cost constraints.

It is presently preferred to employ the second embodiment of the invention (discussed above) due to the cryptographic strength associated with its ability to recreate the complete digital signature verification process, combined with the trust instilled by more conventional techniques used for providing secure storage, and in conjunction with audit and viewing facilities with which to view evidence and monitor the secure storage controls. In practice, the most useful embodiment of the invention for a particular application may be that which is the least expensive and which still meets the user or application requirements.

Several issues related to the design of a system which implements LTSV are described below. Alternatives for the resolution of the issues are presented, as well as a discussion of the advantages and disadvantages associated with each alternative. The best approach to any given solution depends upon the security requirements of the application(s) using the LTSV services, as well as the cost constraints. There is no best solution for all applications.

When to Save the PKI State

Signature reverification is preferably associated with a particular time because the outcome of this process could change, depending upon the state of the PKI *(e.g.* because of certificate revocations or the creation/removal of cross certificates). There are numerous possibilities with regard to when the PKI state should be saved, including:

- At signature creation time. This approach is used when an individual wants to document the validity of his/her signature at the time it was created. This is the most accurate time to store the PKI state because it reflects the state at the time of signing, which is presumably the critical time in evaluating the authenticity of that signature. Changes to the PKI state occur after that time, some of which could impact the outcome of a signature reverification. Therefore, saving of the PKI state at any time after signing introduces the possibility of inconsistencies between the signer's and recipient's perspectives on a signature's validity.
- At signature verification time. This approach is useful when a recipient wants to document the validity of a signed document received from another individual.
- At archival time. When a user decides that a document should be archived for long term storage is also an appropriate time to save the PKI state.
- When explicitly requested. There may occur certain application specific document life cycle milestones, at which time the user may desire the PKI state to be saved for future reverification.
- Just before changes in PKI state *(e.g.* certificate revocation). This approach requires a tight integration with the PKI because changes in the PKI must be monitored.

The correct time at which to save the PKI state is preferably determined by the constraints and needs of the application using the LTSV services. A robust LTSV solution is able to accommodate the needs of all (or most) applications in this respect.

Contents of the PKI State.

The exact composition of the PKI state varies somewhat from one PKI vendor's product to another's, depending upon the implementation chosen by each vendor. Moreover, certain information is stored in a different format from one vendor to another. In addition, the contents of a PKI state may change over time as well, as new capabilities (and supporting data) are added to the system. Finally, the required contents of the PKI state may change from one application to another, depending upon the needs *(e.g.* level of security and legal requirements) of each application.

Notwithstanding these uncertainties, there are classes of PKI state information which are candidates for saving. These classes include:

- Certificate chain (list of certificates from one entity to another, including certification authorities (CAs) and the end entities).
- CRLs (one for each CA in certificate chain).

  CA policy statements or identifiers.
- Attribute certificates.
- Date and time.
- Trust information (e.g., public key(s) or certificate(s) of trusted root CA(s), policy constraints).

  Policy constraints are, for example, non-crypto-

graphic information stored within the LTSV archive. The public key of the trusted root CA may or may not be cryptographically protected. If it is embedded in a certificate, then it is signed by the CA. However, it could just as well be an isolated public key, in which case it is unprotected by cryptography.

It is possible that the items in the above list may not be supported or available from certain PKI implementations. Further, the PKI state from another implementation might include some additional data. Therefore, the list above is only an example of what might be considered important pieces of PKI state information, given the current state of the technology. An implementation of an LTSV service is preferably tied to the implementation of a specific PKI until such time as the technology evolves and comprehensive standards emerge.

How to Store the PKI State.

Storage of the PKI state is preferably accomplished in either of two general ways:

- Store all of the PKI state relevant to each document separately; and
- Store the PKI state centrally, and only store references to the PKI state information with each document. This approach enables storage efficiencies by eliminating the redundant storage of PKI state information over multiple documents. For example, given two documents submitted to the LTSV server at about the same time, it is possible that the CRLs contained in the PKI state are exactly the same for both submissions. Central storage of this information allows the LTSV server to store this information only once.

The storage requirements for the save state solution for LTSV may be quite large, depending upon the size of the certificates, the length of the certificate chains and -- more importantly -- the size of the CRLs. The choice of storage technique may have a great impact on the total data storage requirements. It is clearly undesirable to store massive CRLs with every document that is stored for long term archival and possible future reverification. For this reason, the second alternative listed above is presently considered to be the preferred approach.

However, this second approach may present certain difficulties in applications where the LTSV server is an entirely separate component from the PKI, and where support of multiple PKIs is a primary design goal of the LTSV server. In this case, it would be advantageous for the PKI state to remain opaque to the LTSV server, thereby providing ease of support of multiple PKI vendors. Given that what constitutes the PKI state for one vendor may be different for another vendor, it is desirable to maintain an opaque interface between the LTSV server and the PKI. On the other hand, storage efficiencies can be derived only if the LTSV server is informed about the contents and format of the PKI state information. These conflicting requirements -- acceptable storage size and opaqueness -- pose a challenge for the design of an LTSV service.

Some of the possible alternatives are listed below:

- Keep the interface opaque and store the PKI state as it currently exists (full certificate chains and CRLs). This option focuses entirely on the opaqueness requirement, and sacrifices the data size requirement. The primary advantage of this solution is simplicity and quick deployment.
- Remove the opaqueness requirement by making the PKI state visible to the LTSV server. This allows the LTSV server to manage the certificates and CRLs manually -- thereby avoiding duplication of these objects in the data store. This solution potentially sacrifices the ease of multi-vendor support at the expense of achieving efficient storage.
- Compromise by making the CRLs visible to the LTSV server, where other PKI state information is opaque. This solution is interesting because it is probable that the CRLs are the largest piece of data comprising the PKI state. Because CRLs are standard across nearly all PKIs, the visibility should not pose a problem in terms of multi-vendor support. This solution address both of the requirements, but does put the burden of management of the CRLs onto the LTSV server.
- An alternative embodiment of the invention provides a variation on the solution above that breaks up the PKI state into multiple pieces, each of which is opaque. The PKI indicates which of these objects is common across multiple signed documents *(e.g.* CRLs and certificates). The PKI labels these objects with unique handles (identifiers), thereby allowing the LTSV server to store these objects and retrieve them efficiently when needed for signature reverification -- all the while maintaining the opaqueness of these objects.
- Encourage PKI vendors to make concise cryptographically protected assertions about the state of revocation, as an alternative to using CRLs. (For example, CRLs indicate who has been revoked. It would be more efficient if the PKI could make a statement that a certificate has not been revoked at a given point in time. This could eliminate the need for storing CRLs.) This approach is non-standard, but acceptable because these PKI-generated assertions are not seen by any application outside the PKI. A major benefit of this approach is that the opaqueness of the state is preserved while some of

the storage inefficiencies of the state information are removed.

For cases where the LTSV server is dedicated to a particular PKI, it is preferred to create a close integration between the two components, thereby allowing the LTSV server to know about the content and format of the PKI state information, and store it in the most efficient manner possible. For cases where the LTSV server must be insulated from the PKI *(e.g.* for portability across multiple PKIs), one of the options listed above (with the possible exception of the first two) may be used.

Location of Source Data.

The source data associated with an LTSV submission can be stored either by the client or by the LTSV server itself. Some LTSV clients do not choose to submit clear text to the LTSV server for storage because of concerns over privacy. (Privacy of the channel between the LTSV client and the LTSV server can be achieved by having the client encrypt the submission under the public key of the LTSV server.) A submission to the LTSV may be encrypted, such that the LTSV is not able to decrypt it. That is acceptable with the LTSV server. However, the client must determine how to decrypt the submission.

Given that the LTSV server views the source data as a bit stream, it is possible that the message could be encrypted by the LTSV client before submission. (The fact that a general purpose LTSV server treats the source document as a bit stream does not preclude the possibility of implementing an application specific LTSV server that is aware of the contents of the submitted data.) The LTSV server treats the encrypted data as the source. Such prior encoding may be sufficient for some applications' needs for privacy. In this case, however, either the client must maintain the decryption key, or the key must be divulged and stored by the LTSV server (which is probably not acceptable).

Alternatively, the LTSV client may submit a message digest (resulting from a one-way hash function) as the source document. The client, in this case, is responsible for maintaining the real source document. If the source document is stored by the client, then only the PKI state information is stored in the LTSV server's archive (along with some reference to the source document or the submitter).

Whether the source data is stored by the client or the LTSV server, it must be produced if and when a reverification of that document is required. It is a required component of any signature verification process.

Key and Algorithm Degradation.

If cryptographically encoded information *(e.g.* digital signatures or encrypted data) is stored for a significant period of time, the issue of key and algorithm degradation must be addressed, *i.e.* the probable loss in effectiveness of a cryptographic key or algorithm over time. Although signed documents are expected to be sealed securely with strong cryptographic algorithms, the strength of an algorithm and associated key length decreases over time with the advent of faster computers and new developments in cryptoanalysis. It is expected that cryptographic algorithms and key lengths have limited life spans. It is generally acknowledged that they should be examined, modified, and/or replaced at periodic intervals. This legitimate security concern increases with the length of time for which a document is valid, and it becomes a very serious threat as the time span approaches multiple decades.

For example, a digital signature performed today, using RSA and a 512-bit key, is considered very strong *(i.e.* it would take years to forge it). But, it is also expected that this same signature may be easily forgeable within ten years or so. This is because of the increased ability to search the key space faster (and thereby find the key used to sign the message) with newer computers or computing techniques. Similarly, there may continue to be developments in techniques for factoring large prime numbers (the difficulty of which is the basis for the strength of the RSA algorithm). It is reasonable for both of these abilities to improve over time (although the pace of these changes is less certain).

It is, therefore, prudent to protect cryptographically encoded documents from this threat when those documents must live beyond a few years. This is the case with the documents expected to be submitted to the LTSV server, and especially so when using the save state approach herein disclosed. Hence, the LTSV facility should address this problem. Not only must the signed documents stored in the archive be protected from this threat, but all other cryptographic data or metadata stored in the archive should be protected. (The cryptographic data primarily include keyed information. That is, any information that is signed or encrypted with a private key. Such information may also include non-keyed cryptographic data, such as the output from a hash algorithm, such as MD5.) This data could also include such items as certificates and CRLs, which are, themselves, digitally signed by the issuing CA.

There are any number of ways that the LTSV facility addresses this problem. For example:

- Periodically countersign all data in need of cryptographic refresh through the use of nested signatures. Under this approach, the LTSV server effectively refreshes the cryptographic strength of the data by signing it with successively longer keys (or stronger algorithms) every few years. Each counter signature has the effect of locking in the cryptographic strength of the enclosed signature(s), thereby extending the cryptographic life of the enclosed document. This countersignature is prefera-

bly performed by the LTSV server using a key owned by that server. Performance shortcuts may be required to avoid the costly unraveling of signatures at reverification time, or the potentially time consuming task of countersigning every document in the archive. Such shortcuts include, for example, removing a previous countersignature before applying a new one, or countersigning the entire archive or portions thereof instead of each individual document.

- A modification of the cryptographic approach suggested above provides for countersigning the information in the archive once, but with an extremely long key, *i.e.* a key which is expected to be unbreakable for decades or more. This eliminates all need for countersigning. This may be merely a theoretical solution because finding an algorithm and key length which is secure for that long is impossible to predict. Therefore, there is still a need to provide some backup mechanism, just in case the original algorithm were cracked, for example.

- Protect the cryptographic information in the archive by insulating the archive itself, rather than the individual documents contained in the archive, thereby eliminating the need for a cryptographic solution. In this approach, the archive is protected via access controls and other procedural controls. If the archive can be effectively insulated from intrusion and modification, then key degradation is not an issue and cryptographic refresh is not necessary.

- Use a time stamp facility to seal the cryptographic information in time. Such a facility is expected to provide all of the necessary characteristics required for solving the key degradation problem. This time stamp facility could use one of the techniques listed above, or it could even be an independent service (see below for a discussion of time stamping).

Relationship to Time stamping.

A secure and comprehensive LTSV solution preferably includes an association with a time stamping mechanism. For long term verification of digital signatures, it is often necessary to know the time at which particular events occurred *(e.g.* time of signing or verifying a signature) to determine if a document was valid at that specific time. If there were uncertainty concerning when a document was signed, then the later reverification of that document could be compromised because of the uncertainty of when it was signed.

Fig. 4 is a flow diagram that provides two alternative scenarios that illustrate the applicability of time stamps.

In scenario 1:

- Alice signs a document at time T1, and sends it to Bob (140).

- Alice's certificate is revoked at time T2 (142).

- Bob verifies Alice's signature at time T3 (144).

In scenario 2:

- Alice's certificate is revoked at time T1 (150).

- Alice signs a document at time T2, and sends it to Bob (152).

- Bob verifies Alice's signature at time T3 (154).

When Bob performs the verification (at time T3), he does not know when Alice signed the document. This is critical, because if Alice's key (certificate) were revoked before signing the message, then the signature verification by Bob should fail, and Bob should not trust the contents of the message. If, on the other hand, the revocation occurred after the act of signing, then the signature can be presumed to be valid and trustworthy. For simplicity, this example does not consider the complicating issue of CRL latency, *i.e.* the time between the initiation of certificate revocation and the time when this information becomes available on a CRL.

This example demonstrates the need for a secure and trusted time stamp mechanism in the domain of digital signatures. The mere recording of the current date and time when creating a digital signature is not sufficient for most application because the source of that time may not be trusted by the recipient. The impact, however, also applies not only to the short term signature verification process, but also to the long term verification of digital signatures. Given the example above, the LTSV server could save the PKI state (at time T1) associated with scenario 1 or scenario 2 (or both). The outcome of a signature verification on this message years later is greatly affected by the PKI state used for this verification process, as well as the target time for the verification.

The problem highlighted above demonstrates the preference that the LTSV service to be cognizant of time. It should:

- Be able to determine in a secure fashion the time at which a document was originally signed;

- Be able to re-create accurately the PKI state which was active at a target time in the past;

- Be able to determine the current date and time accurately; and

- At a minimum, save the PKI state associated with a particular target time.

These requirements establish the preference for the integration of a time stamp facility with the signing and verification (and reverification) process. When a document is signed, it is also preferably time stamped to document in a secure fashion the precise moment at which that event occurred. The LTSV service should know the time for which the PKI state is to be saved, be sure to save the appropriate state (the state active at the target time), and execute its signature reverification process in the context of the correct time.

Usage Scenarios.

Figs. 5a-5c provide block schematic diagrams that illustrate three long term signature verification usage scenarios.

In scenario 1, a client (EntityA) 50 submits a document to a LTSV facility 52 for long term signature verification. This is a simple case where EntityA is interested in documenting that it possessed some piece of information.

In scenario 2, EntityB 56 receives a document from EntityA 54 and submits that document to the LTSV facility 58. In this case, EntityB wants to document that it received some information from EntityB.

In scenario 3, EntityA 60 sends the same document to EntityB 64 and to the LTSV facility 62. This case represents a carbon copy feature, whereby EntityA can document the information it sent to EntityB by additionally filing it with the LTSV facility.

Each of the scenarios described above raises issues with respect to encryption, private key access, and trust models.

Encryption and Private Key Access.

A document can be encrypted and/or signed. Ideally, the LTSV facility accepts any such document. This raises a problem, however, with respect to how the LTSV module works with respect to the encryption. When encrypting under a public key system, the document is effectively encrypted under the public key of the recipient, thereby guaranteeing that the recipient (the possessor of the corresponding private key) is the only entity which can decrypt the information. (For purposes of this discussion, interaction with symmetric keys and algorithms is ignored.)

When applying this principle to scenario 1, it is clear that if the signed message is also encrypted, then it could be encrypted under the public key of the LTSV module. This allows the LTSV component to unwrap the signed document and preserve it for long term verification. This is a useful feature because it provides confidentiality between EntityA and the LTSV service. This scenario does not preclude the possibility that the source document sent signed and encrypted to the LTSV module could itself be encrypted under a key known only to EntityA. That is, it is not necessary that the LTSV have access to the plain text version of the source document. The LTSV module treats that encrypted document as the source. If EntityA does decide to encrypt the document first under a secret key before submitting the document to the LTSV service, then it is the responsibility of EntityA to maintain possession of that key if and when decryption of that document is required.

In Scenario 2, if the message from EntityA to EntityB is encrypted (under the public key of EntityB) and then forwarded -- unchanged -- to the LTSV service by EntityB, then it is unreadable by the LTSV component because it does not possess the private key required to decipher and unwrap the enclosed signed document. This unwrapping (decipherment) is essential for the LTSV module to do its job.

There exist several alternatives for addressing this problem:

- Allow the LTSV facility to have access to EntityB's private key;
- Do not allow EntityA to send encrypted documents to EntityB; or
- Have EntityB strip off the privacy aspect of the signed and encrypted document received from EntityA. Because EntityB wants to preserve EntityA's signature on the document, and be able to verify it at a later time, this stripping process can not alter the validity of the signature. EntityA can then either send the stripped *(i.e.* plain text) document to the LTSV service, or it can re-encrypt it (still preserving the original signature by EntityA) under the public key of the LTSV module.

The latter approach above is presently the preferred approach. The first approach above raises significant security concerns because it requires distribution of an entity's private key. The second approach above is unacceptably restrictive on the usage of the system.

Trust.

Digital signature verification is always performed between two (and only two) entities. The verification process is based upon (among other things) the trust relationship(s) in place between those two entities -- the originator (signer) and the recipient (verifier).

Fig. 6 is a block schematic diagram that illustrates trust between two entities according to the invention. In this situation, EntityA 70 has been issued a certificate by CA1 72, EntityB 74 has been issues a certificate by CA2 76, and CA's 1 and 2 have been cross certified. (A cross-certificate is a special type of certificate which indicates mutual trust between two CAs.) The resulting trust model sets up a path of trust between EntityA and EntityB, enabling them to verify digitally signed docu-

ments from one another successfully. (A trust model is comprised of the trust relationships among PKI entities (CAs and end users), embodied by the certificates and cross-certificates issues among these entities, as well as the underlying policies which enable this trust.) Note that if any of the three paths in this model were not in place, then sufficient trust would be lacking for the successful exchange of digitally signed messages between the two end parties. Signature verification would fail if any entity in this path is not trusted.

This trust path is commonly referred to as the certificate chain because it is composed of the certificates between the two entities. When considering the save state approach to long term signature verification, it is this entire trust path (among other things) which must be archived as part of the PKI state for later signature reverification. Moreover, the trust path stored by the LTSV facility must contain the relevant trust information existing at the time of the request, not at some other time (before or after) where the trust relationships may be different between the entities. For example, a cross certificate between to CAs could either be created or removed at some point in time. This could effect the trust between two entities and affect the outcome of a signature verification.

As discussed above, the time associated with the existing trust model between two entities is extremely important to the LTSV facility, but there are also ramifications with respect to how the LTSV module works -- specifically, what trust information is needed and stored by the LTSV component for later signature verification. This gets complicated when the LTSV component is included, which may or may not be trusted (via some trust path) by some entities.

Consider the three scenarios illustrated in Figs. 5a-5c:

Scenario 1 is fairly straightforward. There are only two entities involved. The trust path stored by the LTSV facility is the path between those two parties (EntityA and LTSV). It is assumed that trust exists between these entities, otherwise EntityA would not submit a request to that service.

Scenario 2, however, raises certain issues. When EntityB sends a request to the LTSV service, what signature does EntityB want to later verify? Most likely, EntityB wants to reverify EntityA's signature at a later time -- it wants the LTSV service to document that the signed document received from EntityA was valid (contained a valid signature) at the time it was received. This raises two general questions:

- Whether the LTSV service is trusted by EntityA. It can be assumed that the communicating parties (EntityA with EntityB, and EntityB with the LTSV) have developed some trust between themselves. But in this case, it is possible that there exists no trust path between EntityA and the LTSV component.
- The trust path that is to be stored by the LTSV facility. There exist three possible trust paths which can be stored by the LTSV, *i.e.* the path between Entities A and B; the path between EntityB and the LTSV component itself; and the path between EntityA and the LTSV component, if it exists.

Fig. 7 is a block schematic diagram that illustrates a long term signature verification trust model. Given scenario 2, where EntityB 84 submits a signed document, received from EntityA 80, to the LTSV component 88, the LTSV can save the trust model embodied in the original signed document (EntityA 80 → CA1 82 → CA2 86 → EntityB 84). Later verification of this signature recreates the verification process originally performed by EntityB when it received this document from EntityA. If, however, the PKI state stored by the LTSV service were to contain only the trust path between the submitter and the service (EntityB 84 -> CA2 86 -> CA3 90 -> LTSV 88), then reverification of the original document, as originally performed, is impossible. In fact, this is exactly the paradigm described in scenario 1, where the trust path between the submitter and the LTSV are of interest.

The above discussion reveals that there are good reasons for the LTSV component to be able to store either trust path, depending upon the requirements of the client.

In scenario 2, the LTSV would most likely store the trust path corresponding to the message from EntityA to EntityB (to reverify the signed document from EntityA to EntityB). In scenario 1, the LTSV would store the trust path corresponding to the submission itself -- from EntityA to the LTSV.

Similarly, scenario 3 represents a case where flexibility in which trust path(s) to store is required. In this case, EntityA's submission to the LTSV facility may be with the intent to either reverify its correspondence with EntityB, or to reverify the submission itself (between EntityA and the LTSV). In fact, both trust paths may be of use to the client. The requirements on the LTSV are determined by the business of the particular application being deployed. For this reason, the interface to the LTSV preferably supports the ability of the client to indicate the needs in terms of trust paths as it impacts the requirements for later reverification.

The disclosures in United States patent application no 08/892,792, from which this application claims priority, and in the abstract accompanying this application are incorporated herein by reference.

Claims

1. A method of enabling long term verification of digital signatures, comprising the steps of:

   submitting a source document or digest thereof to a signature verification entity; and

using an archive facility to store a public key infrastructure (PKI) state relative to said document at a selected archival time.

2. A method as in claim 1, comprising the steps of:

using said archived PKI state to re-create said PKI state relative to said document at a selected time after a certificate associated with said signature has expired;
wherein the time over which a digital signature associated with said document can be verified is extended beyond expiration of any or all of any certificates upon which that signature depends.

3. A method as in claim 1 or 2 comprising the step of:
storing said source document separately from any associated cryptographic information.

4. A method as in claim 1, 2 or 3 wherein the selected archival time used as the basis for subsequent re-creation of a signature verification process is the time of said source document submittal;
is the time when said source document was signed by its originator; or in the time when said source document was verified by a recipient.

5. A method as in any preceding claim, comprising the step of;
protecting said PKI state information from intrusion by maintaining it in a secure storage facility preferably comprising of at least one of a firewall, access control mechanism, audit facility, intrusion detection facility, physical isolation and network isolation; or protecting non-cryptographic PKI state information from intrusion by protecting it in an archive via any of a signature and keyed hash algorithm.

6. A method as in any preceding claim comprising the step of:
providing utilities for viewing said PKI state information and for visually monitoring system security.

7. A method as in any preceding claim, wherein classes of PKI state information may include one or more of certificate chain from one entity to another, including certification authorities (CAs) and the end entities; certificate revocation lists (CRLs), one for each CA in certificate chain; certificate practice statements; attribute certificates; policy constraints; trust information; and date and time.

8. A method as in any preceding claim, comprising the step of:
periodically countersigning all data in need of cryptographic refresh through the use of nested signatures and/or countersigning information in said archive facility once with an extremely long key.

9. A method as in any preceding claim, comprising at least one of the steps of:

protecting said archive facility itself, rather than individual documents contained in said archive; and
employing a cryptographic protection mechanism at said signature verification entity.

10. A method as in any preceding claim, comprising the step of:
using a time stamp facility to seal cryptographic information in time.

11. Apparatus for long term verification of digital signature, comprising:

a source document; and
an archive facility for storing a public key infrastructure (PKI) state relative to said document at a selected archival time.

12. Apparatus as in claim 11, comprising:
either of a signature and a keyed hash system for protecting non-cryptographic PKI state information from undetected modification, wherein said noncryptographic PKI state information is maintained in an archive.

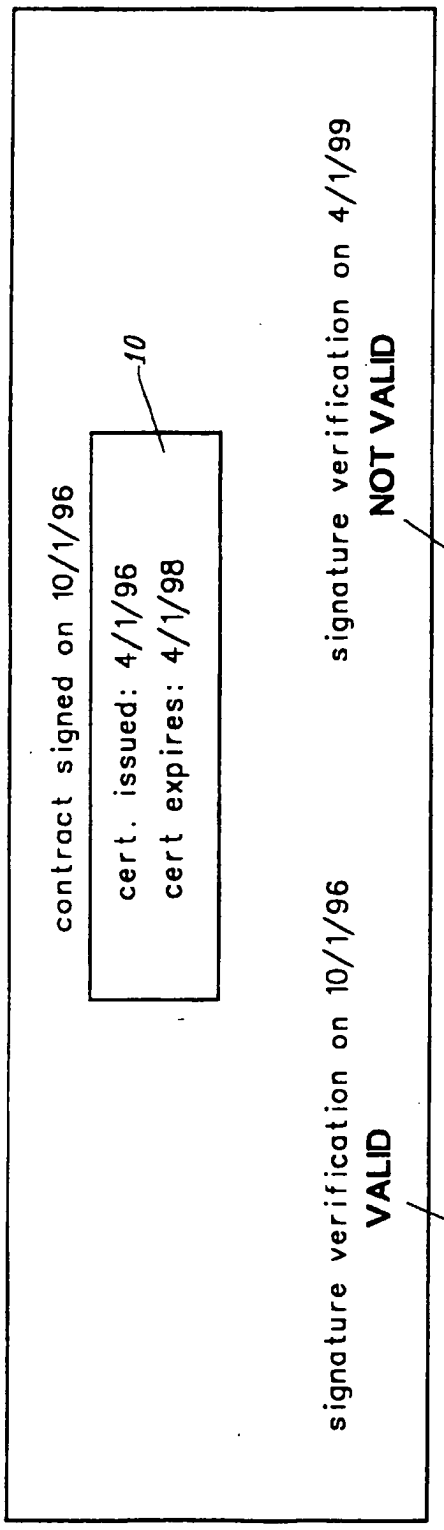


FIG. 1
(PRIOR ART)

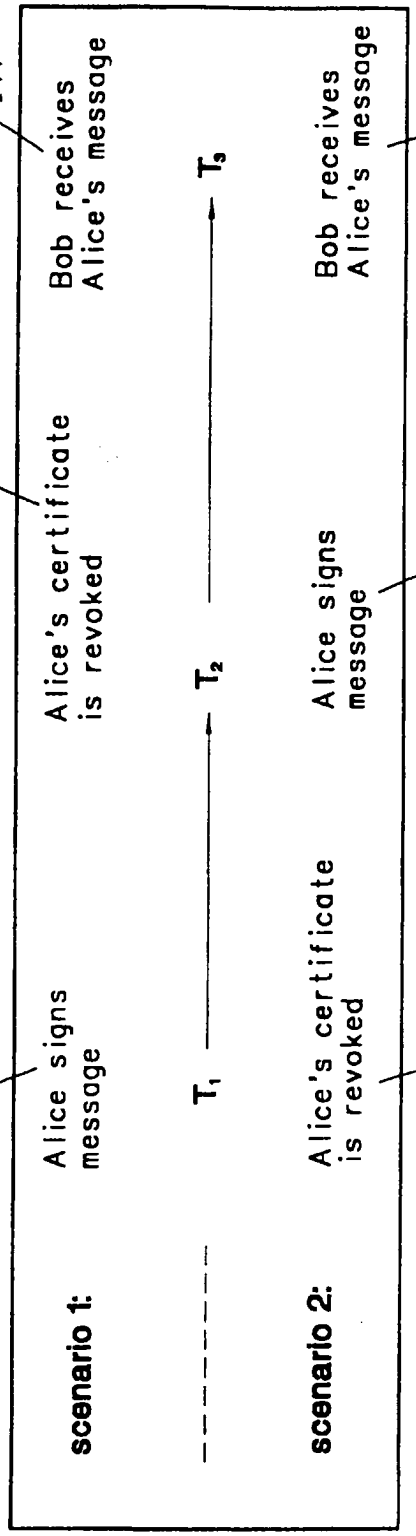


FIG. 4

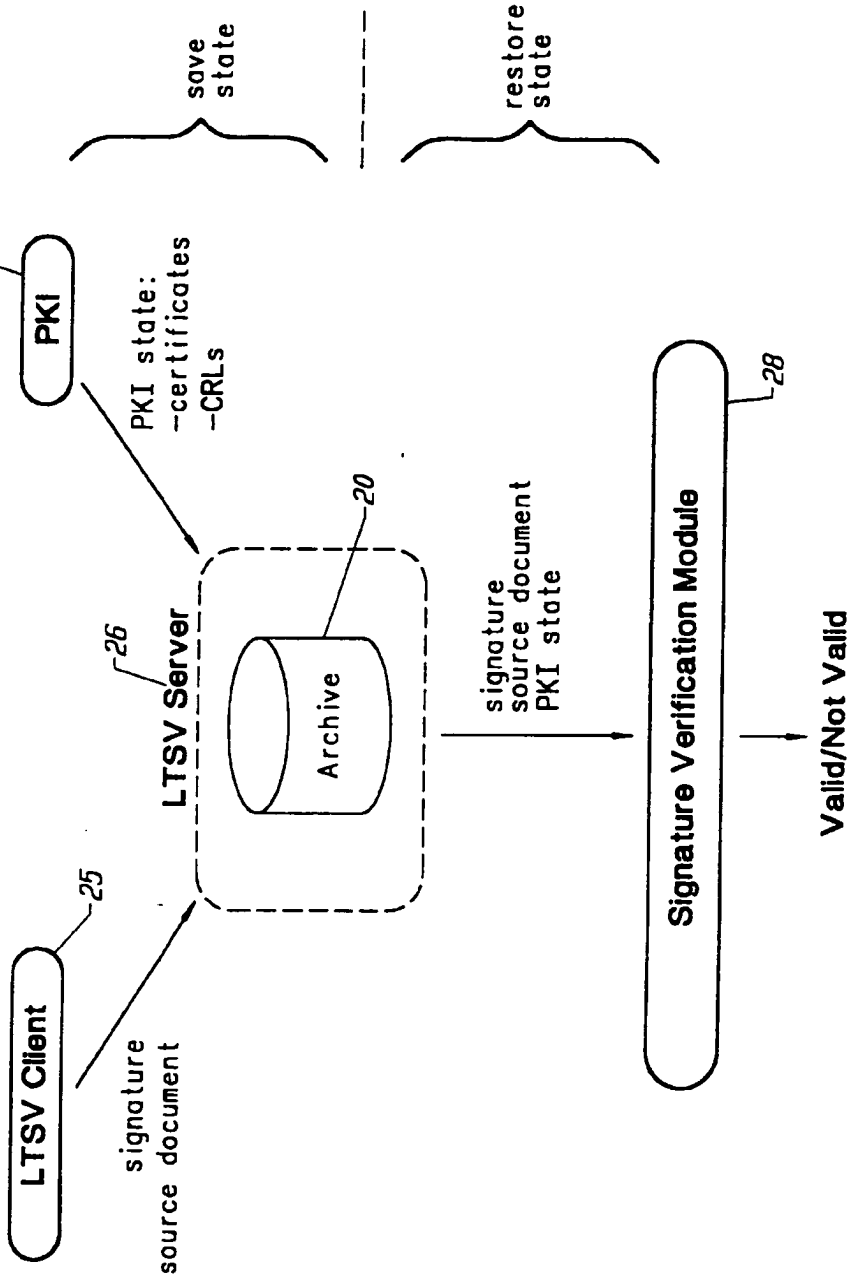


FIG. 2

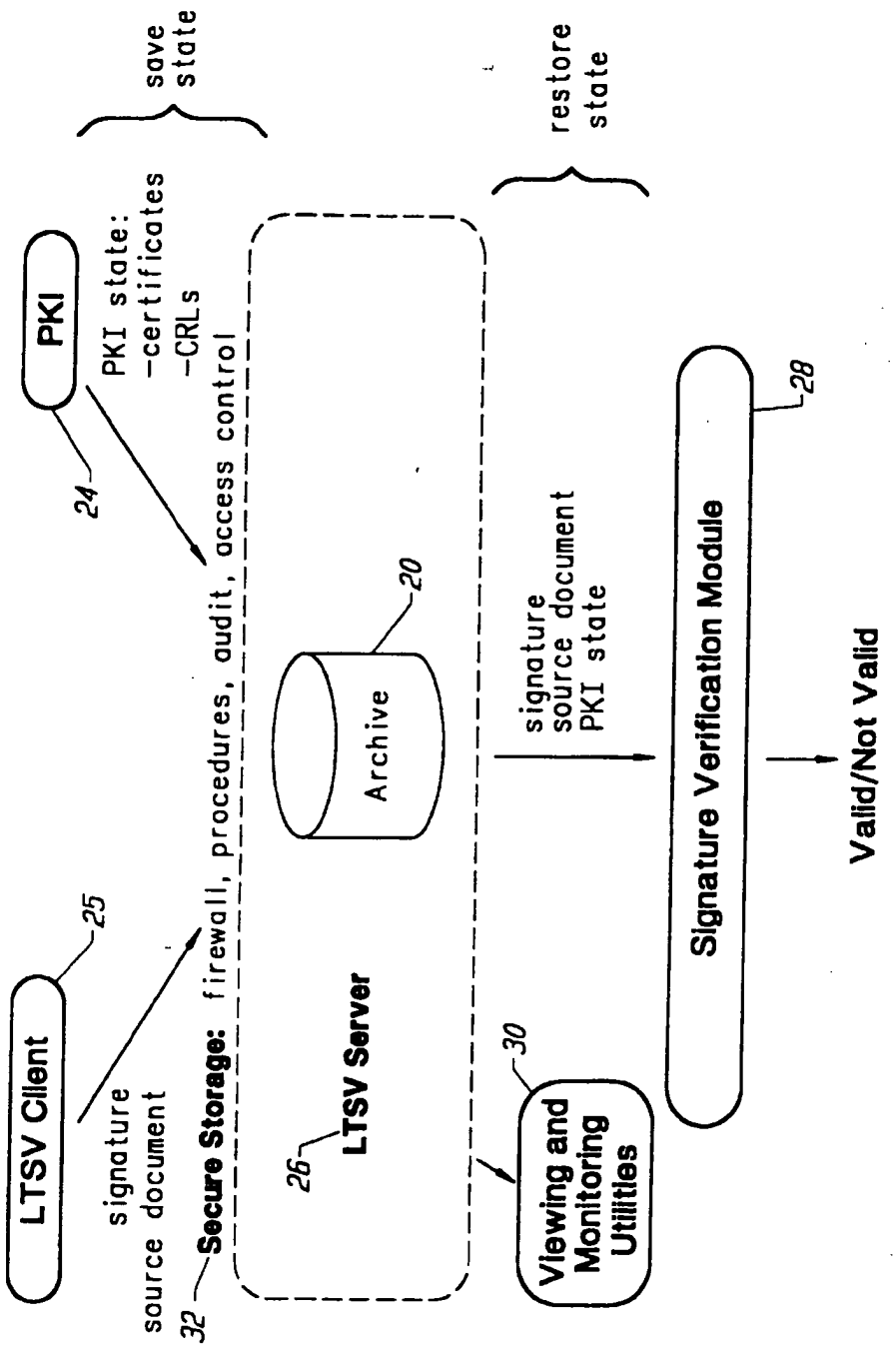


FIG. 3

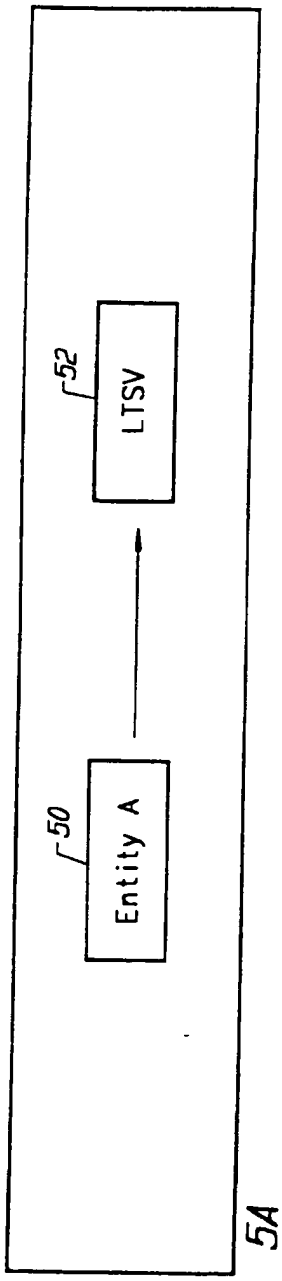


FIG. 5A

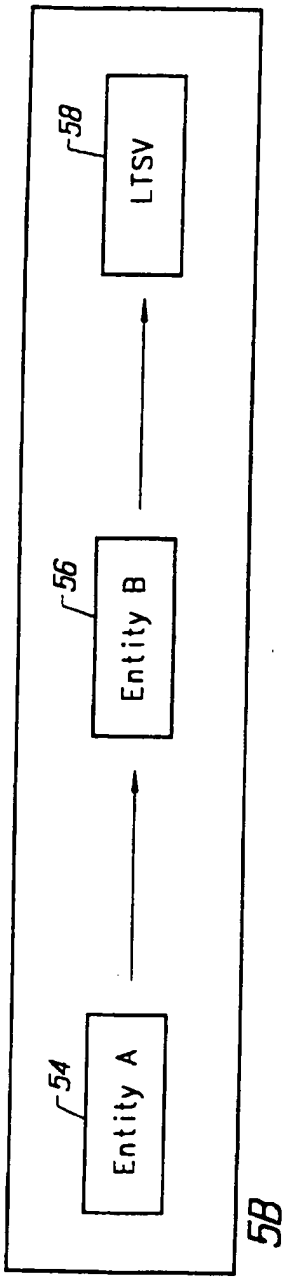


FIG. 5B

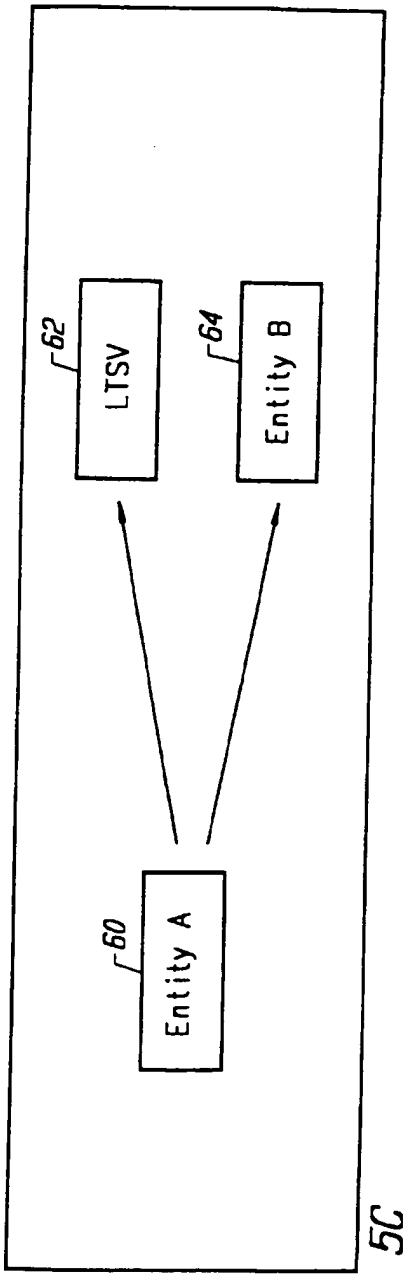
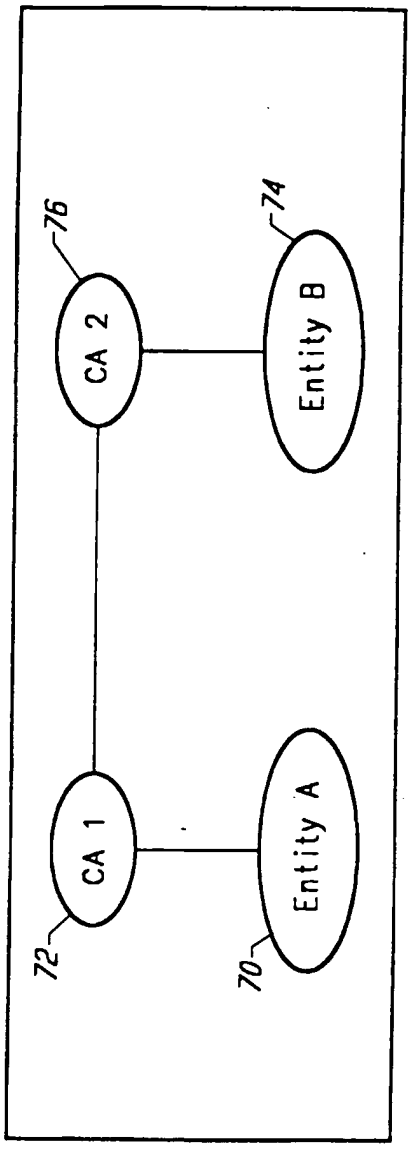


FIG. 5C

FIG. 6



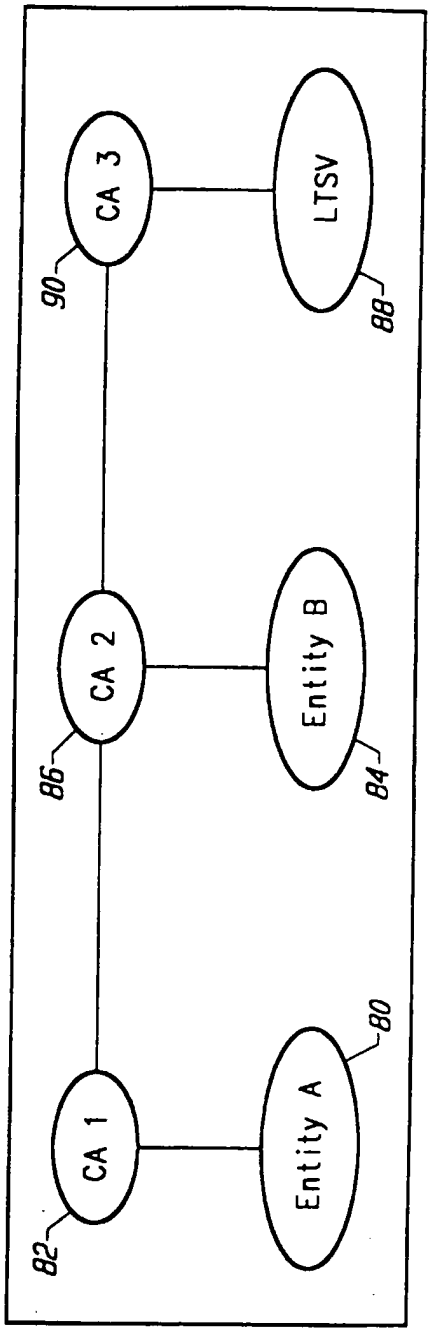


FIG. 7

1 483 282

PATENT SPECIFICATION (11) 1 483 282

(21) Application No. 52131/74 (22) Filed 2 Dec. 1974
(31) Convention Application No. 7342706
(32) Filed 30 Nov. 1973 in
(33) France (FR)
(44) Complete Specification published 17 Aug. 1977
(51) INT CL² G06F 13/00
(52) Index at acceptance
G4A 10EX 13E 13M 17B4 17P 6G 6H 6X AP ND NR

(19)

(54) APPARATUS FOR PROTECTING THE INFORMATION IN AN VIRTUAL MEMORY SYSTEM IN PROGRAMMED DATA PROCESSING APPARATUS

(71) We, COMPAGNIE INTERNATIONALE POUR L'INFORMATIQUE C11-HONEYWELL-BULL (formerly Compagnie Honeywell-Bull), a French Body Corporate, of 94 Avenue Gambetta, Paris 75020, France, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:—

The present invention concerns apparatus for protecting the information in a virtual memory system in programmed data processing apparatus.

Several schemes have been utilized in the past in order to protect information. Some of them are detailed by Robert M. Graham in a paper entitled "Protection in an Information Processing Utility", published in CACM (May 1968).

This type of memory protection is inadequate for present day multiprogramming systems because there is no provision for gradations of privilege or gradations of accessability, and severely limits the control over access to information. There should be provisions for different access rights to the different types of information. A partial answer to this problems is found in the concept of a memory having a segment as the unit of information to which access is controlled (see Patent Application No. 21630/74, (Serial No. 1,465,344), filed on 15 May 1974). Varying degrees of access to each segment is possible by providing for different types of privileges attached to each segment such as master/slave, write/no-write and execute/non-execute. However, this method of protecting the privacy and integrity of information does not take into account the user of the information. Under this type of protection, privilege is not accorded the user but the information being protected. Hence a user if he has access at all to a segment has access similar to all other users who have access to the segment. David C. Evans and Jean Yves LeClerc in a paper entitled "Address Mapping and the Control of Access in an Interactive Computer," SJCC 1967, recognized the problem and attempted a solution. Evans and LeClerc said in that article p. 23, "The user of a computing system should be able to interact arbitrarily with the system, his own computing processes, and other users in a controlled manner. He should have access to a large information storage and retrieval system called the file system. The file system shoufd allow access by all users to information in a way which permits selectively controlled privacy and security of information. A user should be able to partition his computation into semi-independent tasks having controlled communication and interaction among tasks. Such capability should reduce the human effort required to construct, debug, and modify programs and should make possible increased reliability of programs. The system should not arbitrarily limit the use of input/output equipment or limit input/output programming by the user". Evans and LeClerc proposed conditioning access rights on the procedure-in-execution. The segment, under their proposal, is still the unit of information to which access is controlled; however, a segment's access control attributes are recorded substantially in a user-name versus procedure tables whose entries are the access modes. Such a solution, however, has serious drawbacks. For one, the construction and updating of each segment's table of access control attributes presents a formidable task. For another, too many uses of the segment and event occurrences must be foreseen. To overcome this problem access control by procedure-set was suggested. Under this suggestion, related procedures are grouped into "sets of procedures" and access rights to segments is based on the identity of the set to which the procedure seeking access

belongs. This method alleviated the problem of constructing and updating each segment's voluminous tables of access control attributes, but introduced the problem of determining to which set a given procedure belonged, particularly when a procedure was or could be a number of many sets. This ambiguity in defining sets, and the possible transitions between sets makes the implementation of access control based on "sets of procedures" extremely difficult.

To overcome the difficulties encountered with the "set" technique a ring concept was developed. The ring concept groups the sets of procedures into rings that can unambiguously be ordered by increasing power or level of privilege. By assigning a collection of sets to a collection of concentric rings, and assigning numbers to each ring with the smallest ring having the smallest number and each succeeding larger ring having a progressively greater number, different levels of privilege can then be unambiguously assigned to the user of a segment. Under this concept the innermost ring having the smallest number assigned to it has the greatest privilege. Hence it can be postulated that users in the lowest ring number can access information having higher ring numbers, but users in a higher ring number cannot access information having lower ring numbers or can access information in a lower ring number only in a specified manner. This palpable change of power or level of privilege with a change in rings is a concept which overcomes the objections associated to a change of sets.

Multics (*Multi*plexed *I*nformation and *C*omputing *S*ervice) is an operating system developed primarily by Massachusetts Institute of Technology, in cooperation with General Electric Co. and others which first utilized the ring theory of protection in software on a converted Honeywell 635 (Registered Trade Mark) computer and later on a Honeywell 645 (Registered Trade Mark) computer. The Multics philosophy utilizes 64 rings of protection numbered as rings 0—63 and is set forth generally in a paper entitled "Access Control to the Multics Virtual Memory" published by Honeywell Information Systems Inc. in the Multics Technical Papers, Order No. AG95, Rev. O. A more detailed description of Multics ring protection is to be found on chapter 4 of a book entitled "The Multics System: An Examination of its Structure", by Elliott I. Organick, published by MIT Press, and also in the Multics System Programmers Manual 1969, MIT Project MAC. Briefly, the Multics system does not utilize a "pure ring protection strategy" but rather employs the "ring bracket protection strategy" wherein a user's access rights with respect to a given segment are encoded in an access-mode and a triple of ring number (r1, r2, r3) called the user's "ring brackets" for a given segment. A quotation from pages 137—139 from the Multics Technical Paper entitled, "Access Control to the Multics Virtual Memory" sets out the rules and conditions for using and changing rings.

This "ring protection concept" was first implemented with software techniques utilizing 64 separate rings. Subsequently an attempt was made to define a suitable hardware base for ring protection. The Honeywell 645 (Registered Trade Mark) computer represents a first such attempt. The Honeywell 645 (Registered Trade Mark) system differs from the "ringed hardware" concepts described supra in several respects which when taken together, add up to the fact that the Honeywell 645 (Registered Trade Mark) is a 2-ring rather than a 64-ring machine, and has in lieu of a "ring register", a master mode and a slave mode, which imparts greater power to the processor when in master mode than when in slave mode. "The access control field of the 645's SDW (segment descriptor word) contains no information about rings; in particular its does not contain ring brackets. It does, however, contain either:

a) access-mode information possibly including either of the two descriptors;
accessible in master mode only,
master mode procedure;

b) the specification of one of eight special 'directed' faults (traps) which is to occur whenever the segment descriptor word (SDW) is accessed.

"The procedure is only 'in master mode' when executing a procedure whose SDW indicates a 'master mode procedure'. The processor may enter master mode while executing a slave mode procedure by:
faulting,
taking an interrupt".

"The 645 processor's access control machinery interprets the SDW during the addressing cycle and causes the appropriate action to occur depending on the SDW and (usually) on the attempted access, as follows:

a. If the SDW implies a particular "directed fault", then that fault occurs.

b. Otherwise, if the SDW does not permit the attempted access, the appropriate access violation fault occurs.

c. Otherwise, the SDW permits the attempted access and the access is performed.

"When a fault occurs, the 645 enters master mode and transfers control to the

appropriate master mode fault handling procedure". (Access Control to the Multics Virtual Memory, supra pps. 157—158).

Another paper by Michael D. Schroeder and Jerome H. Saltzer entitled "A Hardware Architecture for Implementing Protection Rings" published in Communications of the ACM, March 1972 Vol. 15, No. 3, sets forth background and theory of ring protection and describes a hardware implementation of "ring protection".

Because the Multics and Honeywell 645 version of ring protection was implemented mainly in software, considerable operating system supervisor overhead was entailed particularly when calls to greater or lesser power were made by trapping to a supervisor procedure. What was required was an access control mechanism which had the functional capability to perform effectively its information protection function, was relatively simple in operation, was economic to build, operate and maintain, and did not restrict programming generality. The Honeywell 6000 (Registered Trade Mark) computer system met these requirements by implementing most of the ring protection mechanism in hardware. Hence special access checking logic, integrated with the segmented addressing hardware was provided to validate each virtual memory reference, and also some special instructions for changing the ring of execution. However certain portions of the ring system particularly outward calls and returns or calls to a lesser power and returns therefrom presented problems which required the ring protection function to be performed by transferring control to a supervisor. What is now needed are further improvements in hardware and techniques that will permit a full implementation of ring protection in hardware/firmware and will meet the criteria of functional capability, economy, simplicity and programming generality.

Accordingly the present invention has for an object to provide an improved computer ring protection mechanism.

Accordingly the present invention consists in an internally programmed data processing apparatus CPU having a virtual memory system, and being responsive to internally stored instruction words for processing information and having stored in said virtual memory system a plurality of different types of groups of information each information group-type associated with an address space bounded by a segment having adjustable bounds, and comprising means for protecting the information in said-virtual memory system from unauthorized users by restricting accessability to the information in accordance to levels of privilege, said means comprising in combination with an access checking mechanism:

(a) first means arranged in operation to store in said virtual memory system at least one segment table comprising a plurality of segment descriptors with each segment descriptor being associated with a predetermined one of said segments and each segment descriptor having a predetermined format containing an access information element and a base address element in predetermined positions of said format, said base address element being used for locating in said virtual memory system the starting location of a selected one of said segments, and said access information element for specifying the minimum level of privilege required for a predetermined type of access that is permitted in a selected one of said segments;

(b) a plurality of second means having a predetermined format, communicating with said first means, arranged to store in a predetermined portion of said second means, a segment number SEG for identifying a segment table and the location of a segment descriptor within said segment table, said second means also being arranged to store in a predetermined other portion of said second means, an offset address within the segment identified by said segment descriptor said offset address locating from said segment base the first byte of a word within said segment;

(c) third means responsive to an address syllable element of an instruction being executed for addressing one of said plurality of second means;

(d) fourth means arranged to store a displacement from said address syllable;

(e) fifth means, communicating with said first, second, third and fourth means, arranged to add the displacement D and said base address to said offset; and,

(f) sixth means responsive to said access information element in a selected one of said segment descriptors, restricting the accessability to the segment associated with said selected one of said segment descriptors in accordance to the level of privilege and the type of access specified in said access information element, wherein each group-type of information is associated with a predetermined ring number indicative of a level of privilege said legel of privilege decreasing as the associated ring number increases comprising means for determining the maximum effective address ring number EAR (i.e. minimum level of privilege) of a selected process to access a selected group of information, said means comprising:

(a) first means to store first information indicating the maximum ring number RD (i.e. minimum level of privilege) required to read information from said selected group;

(b) second means to store second information indicating the maximum ring number WR (i.e. minimum level of privilege) required to write information into said selected group;

(c) third means to store third information indicating the maximum ring number MAXR (i.e. minimum level of privilege) required to process information from said selected group; and,

(d) fourth means communicating with said first, second and third means, to determine the maximum of the contents of said first, second and third means whereby the effective address ring number EAR is generated.

The present invention, however, both as to organization and operation thereof may best be understood by reference to the following description which is given by way of example in conjunction with the accompanying drawings in which:

Figure 1 is a block diagram of a computer system utilizing the invention.

Figure 2 is a schematic diagram illustrating the levels of privilege of the invention.

Figure 3 is a flow diagram of the segmented address scheme utilized by the invention.

Figures 4A—4J are schematic diagrams of various novel hardware structures utilized in the invention.

Figure 5 is a schematic diagram of the computer ring protection hardware.

Figure 6 is a schematic diagram of the computer segmented addressing hardware.

Figures 7a—7h and Figures 8a—8c are detailed logic block diagrams of the ring protection hardware.

Figures 9a—9k is a legend of the symbols utilized in the diagrams of the invention.

Figure 10 is a schematic diagram of three stack segments, one each for ring 0, 1 and 3 respectively.

Figure 11A shows the format of the Enter Procedure instruction.

Figure 11B shows the format of a procedure descriptor.

Figure 11C shows the format of a gating procedure descriptor GPD the first word of the segment containing the procedure descriptors.

Figure 11D shows the format of the Exit Procedure instruction.

Figure 12 is a flow diagram of a portion of the Enter Instruction pertaining to ring crossing and ring checking.

Figure 13 schematically shows a segment descriptor and the segment containing procedure descriptors.

Figures 14—16 are flow diagrams showing various operations that are performed when the Enter Procedure instruction is executed.

Figure 17 is a flow chart of the Exit Instruction.

As previously discussed the ring concept of information protection was originated on MULTICS and implemented on various Honeywell (Registered Trade Mark) Computer Systems. The original MULTICS concept required 64 rings or level of privilege and later implementation had the equivalent of two rings on the Honeywell 645 and 8 rings on the Honeywell 6000 (Registered Trade Mark). The embodiment described herein groups data and procedure segments in the system into a hierarchy of 4 rings or classes. (Refer to Figure 2). The 4 rings or privilege levels are identified by integers 0—3; each ring represents a level of privilege in the system with level 0 having the most privilege and level 3 the least. Level 0 is known as the inner ring and level 3 as the outer ring. The basic notion as previously discussed is that a procedure belonging to an inner ring has free access to data in an outer ring. Conversely a procedure in an outer ring cannot access data in an inner ring without incurring a protection violation exception. Transfer of control among procedures is monitored by a protection mechanism such that a procedure execution in an outer ring cannot directly branch to a procedure in an inner ring. This type of control transfer is possible only by execution of a special "procedure-call" instruction. This instruction is protected against misuse in a number of ways. First, a gating mechanism is avilable to ensure that procedures are entered only at planned entry points called gates when crossing rings. The segment descriptor of such a procedure contains a gate bit indicating that procedures in this segment can be entered only via gates; information regarding these gates is contained at the beginning of the segment and is used by the hardware to cause entry at a legal entry-point. The procedure itself must then verify (in a way which, of necessity depends on the function of the procedure) that it is being legitimately called. A further hardward protection mechanism is available in the case that the calling procedure supplies an address as a parameter; it is then possible that the more privileged procedure would invalidly modify information at this address which the less privileged caller could not have done, since the ring mechanism would have denied him access; an address validation instruction is available to avoid this possibility.

An important convention is required

here in order to protect the procedure call mechanism. This states that it is not in general permissible to use this mechanism to call a procedure in a less privileged ring and return to the more privileged one. This restriction is necessary since there is no assurance that the procedure in the higher ring will, in fact, return; that it will not, accidentally or maliciously, destroy information that the more privileged procedure is relying upon; or that it will not, accidentally or maliciously, violate the security of the stack (see GLOSSARY for definition). Any of these could lead to unpredictable results and crash the system.

The level of privilege are quite independent of the process control mechanism and there is no notion here of privileged and non-privileged processes as in the IBM system 360 (Registered Trade Mark). Instead the same process can execute procedures at different levels of privilege (rings) subject to the restrictions imposed by the ring mechanism. In this sense the ring mechanism can be viewed as a method for subdividing the total address space assigned to a process according to level of privilege.

The ring mechanism defined herein permits the same segment to belong to up to 3 different rings at the same time i.e. there are 3 ring numbers in each segment descriptor, one for each type of possible access. Thus the same segment can be in ring one with respect to "write" access, ring two with respect to "execute" access and ring three with respect to "read" access. One obvious use for this is in the case of a procedure segment which can be written only by ring zero (perhaps the loader) but can be executed in ring three.

Of the four available rings, two are allocated to the operating system and two to users. Ring zero, the most privileged ring, is restricted to those operating system segments which are critical to the operation of the whole system. These segments form the hard core whose correctness at all times is vital to avoid disaster. Included would be the system information base, those procedures dealing with the organisation of physical memory or the initiation of physical data transfer operations, and the mechanisms which make the system function, like the "exception supervisor, the scheduler, and the resource management".

Ring one contains a much greater volume of operating system segments whose failure would not lead to catastrophe but would allow recovery. Included herein are the language translators, data and message management, and job and process management. Through the availability of two rings for the operating system, the problem of maintaining system integrity is made more tractable, since the smaller hard core which is critical is isolated and can be most carefully protected.

Rings two and three are available to the user to assign according to his requirement. Two important possibilities are debugging and proprietary packages. Programs being debugged may be assigned to ring two while checked out programs and data with which they work may be in ring two; in this way the effect of errors may be localized. Proprietary programs may be protected from their users by being placed in ring two while the latter occupy ring three. In these and other ways, these two rings may be flexibly used in applications.

The General Rules of the Ring System

1. A procedure in an inner ring such as ring 2 on Figure 2 has free access to data in an outer ring such as ring 3 and a legal access (arrow 201) results. Conversely a procedure in an outer ring such as ring 3 cannot access data in an inner ring such as ring 2 and an attempt to do so results in an illegal access (arrow 202).

2. A procedure in an outer ring such as ring 3 can branch to an inner ring such as ring 1 via gate 204 which results in a legal branch 203, but a procedure operating in an inner ring such as ring 2 may not branch to an outer ring such as ring 3.

3. Each segment containing data is assigned 2 ring values, one for read (RD) and one for write (WR). These ring values specify the maximum ring value in which a procedure may execute when accessing the data in either the read or write mode.

Each time a procedure instruction is executed, the procedure's ring number (effective address ring, EAR) is checked against the ring numbers assigned to the segment containing the referenced data. The EAR is the maximum number of process ring numbers in the processor instruction counter (see later description) and all ring numbers in base registers and data descriptors found in the addressing path. Access to the data is granted or denied based on a comparison of the ring numbers. For example, if a system table exists in a segment having a maximum read/ring value of 3 and a maximum write/ring value of 1, then a user procedure executing in ring 3 may read the table but may not update the table by writing therein.

Procedure Calls and the Stack Mechanism:

The procedure call and stack mechanism is an apparatus being described herein Procedure calls are used to pass from one procedure to another; to allow user procedures to employ operating system services: and to achieve a modular

structure within the operating system. A procedure call is effected by instructions and a hardware recognized entity called a stack.

A stack is a mechanism that accepts, stores and allows retrieval of data on a last-in-first-out basis. Stacks reside in special segments called stack segments. A stack segment consists of a number of contiguous parts called stack frames which are dynamically allocated to each procedure. The first stack frame is loaded into the low end of the segment and succeeding frames are loaded after it. The last frame loaded is considered the top of the stack. A T-register 114 (see Figure 1) locates the top of the stack for the currently active process. A virtual T-register exists in the process control block (PCB) of all other processes in the system.

A stack frame consists of three areas: a work area in which to store variables, a save area in which to save the contents of registers, and a communications area in which to pass parameters between procedures. Prior to a procedure call, the user must specify those registers he wishes saved and he must load into the communications area the parameters to be passed to the called procedure. When the call is made, the hardware saves the contents of the instruction counter and specified base registers to facilitate a return from the called procedure.

Each procedure call creates a stack frame within a stack segment and subsequent calls create additional frames. Each exit from one of these called procedures causes a stack frame to be deleted from the stack. Thus, a history of calls is maintained which facilitates orderly returns.

To ensure protection between procedures executing in different rings, different stack segments are used. There is one stack segment corresponding to each protection ring per process. A process control block (PCB) contains three stack base words (SBW) which point to the start of the stack segment for rings 0, 1 and 2 associated with the process. The ring 3 stack segment can never be entered by an inward call; therefore, its stack starting address is not required in the PCB.

The procedure call is used by users who have written their programs in a modular way to pass from one program module to another. It is used by user programs to avail themselves of operating system services. It is used by the operating system itself to achieve a responsive modular structure. The procedure call as is described in the above referenced patent application is effected by hardware instructions and the hardware recognizable stack mechanism.

The main requirements on a procedure call mechanism are:

1. Check the caller's right to call the caller;
2. Save the status of the caller which includes saving registers, instruction counter (for return), and other status bits;
3. Allow for the passing of parameters;
4. Determine valid entry point for the called procedure;
5. Make any necessary adjustments in the addressing mechanism;
6. Enter the new procedure.

When the called procedure terminates or exits, whatever was done in the call must be undone so that the status of the calling procedure is restored to what it was before the call.

As a preliminary to making a procedure call, the instruction PREPARE STACK is executed. This instruction causes those registers specified by the programmer in the instruction to be saved in the stack. It causes the status register (see Figure 1) to be saved, and provides the programmer with a pointer to parameter space which he may now load with information to be passed to the called procedure.

Another instruction ENTER PROCEDURE permits the procedure call via the following steps corresponding to the requirement specified above:

1. Ring checking—the caller's ring is checked to make sure that this ring may call the new procedure; the call must be to a smaller or equal ring number; and if ring crossing does occur the new procedure must be gated through a gate 204 of Figure
2. The new ring number will then be that of the called procedure.
2. The instruction counter is saved;
3. Base register 0 (see Figure 1) is made to point effectively to the parameters being passed;
4. The entry-point of the called procedure is obtained from a procedure descriptor whose address is contained in the ENTER PROCEDURE INSTRUCTION;
5. A point to linkage information is loaded in base register number 7.
6. The new procedure is entered by loading the new ring number and the address of the entry-point in the instruction counter.

The remainder of the current stack-frame is also available to the called procedure for storage of local variables.

When the called procedure wishes to return, it executes the instruction EXIT PROCEDURE. The registers and the instruction counter are then restored from their saving areas in the stack.

Referring to Figure 1 there is shown a block diagram and a computer hardware

system utilizing the invention. A main memory 101 is comprised of four modules of metal-oxide semi-conductor (MOS) memory. The four memory modules 1—4 are interfaced to the central processor unit 100 via the main store sequencer 102. The four main memory modules 1—4 are also interfaced to the peripheral subsystem such as magnetic tape units and disk drive units (not shown) via the main store sequencer 102 and the 10C (not shown). The main store sequencer gives the capability of providing access to and control of all four memory modules.

Operations of the CPU are controlled by a read only memory ROM, herein called the control store unit 110.

The control store interface adapter 109 communicates with the control store unit 110, the data mangagement unit 106, the address control unit 107 and the arithmetic logic unit 112 for directing the operation of the control store memory. The control store interface adapter 109 includes logic for control store address modification, testing, error checking, and hardware address generation. Hardware address generation is utilized generally for developing the starting address of error sequencers or for the initialization sequence.

The buffer store memory 104 is utilized to store the most frequently used or most recently used information that is being processed by the CPU.

The data management unit 106 provides the interface between the CPU 100 and main memory 101 and/or buffer store memory 104. During a memory read operation, information may be retrieved from main memory or buffer store memory. It is the responsibility of the data management unit to recognize which unit contains the information and strobe the information into the CPU registers at the proper time. The data management unit also performs the masking during partial write operations.

The instruction fetch unit 108 which interfaces with the data management unit 106, the address control unit 107, the arithmetic and logic unit 112 and the control store unit 110 is responsible for keeping the CPU 100 supplied with instructions.

The address control unit 107 communicates with the instruction fetch unit 108, the buffer store directory 105, the main store sequencer 102, the arithmetic logic unit 112, the data management unit 106, and the control store unit 110 via the control store interface adapter 109. The address control unit 107 is responsible for all address development in the CPU.

Interfacing with the address control unit 107, the instruction fetch unit 108 and the control store unit 110 is the arithmetic logic unit 112 which is the primary work area of the CPU 100. Its primary function is to perform the arithmetic operations and data manipulations required of the CPU.

Associated with the arithmetic logic unit 112 and the control store unit 110 is the local store unit 111 which typically is comprised of a 256-location (32 bits per location) solid state memory and the selection and read/write logic for the memory. The local store memory 111 is used to store CPU control information and maintain ability information. In addition, the local store memory 111 contains working locations which are primarily used for temporary storage of operands and partial results during data manipulation.

The central processing unit 100 typically contains 8 base registers (BR) 116 which are used in the process of address computation to define a segment number, an offset, and a ring number. The offset is a pointer within the segment and the ring number is used in the address validity calculation to determine access rights for a particular reference to a segment.

The instruction counter 118 communicates with the main memory local register (MLR) 103 and with the instruction fetch unit 108, and is a 32-bit register which contains the address of the next instruction, and the current ring number of the process (PRN). Also contained in the central processing unit is a T register 114 which also interfaces with the instruction fetch unit 108 and is typically a 32-bit register containing a segment number and a 16-bit or 22-bit positive integer defining the relative address of the top of the procedure stack. The status register 115 is an 8-bit register in the CPU which among other things contains the last ring number—i.e. the previous value of the process ring number (PRN).

The main memory 101 is addressed by the memory address register (MAR) 119, and the information addressed by (MAR) 119 is fetched and temporarily stored in the memory local register (MLR) 103.

Referring now to Figure 3 there is shown a flow diagram of the general rules for segmented address development shown in detail in the above mentioned copending patent application No. 21630/74, Serial No. 1,465,344. Figure 3 when read in conjunction with the above referenced patent application is self-explanatory. There is however one major difference between the address development as shown on Figure 3 to that of the above mentioned application and that is that in the address development of Figure 3 of the instant application as many as 16 levels of

indirection may be utilized in the address development whereas in the above referenced application the levels of indirection were limited to a maximum of two. This of course is a matter of choice with the designer and in no way alters the high level inventive concept.

Referring now to Figures 4A—4J, Figures 4A and 4B show the format of the instruction counter designated by reference numeral 118 on Figure 1. The instruction counter (IC) 118 is a 32-bit register which contains the address of the next instruction, and the current ring number of the process (PRN). Referring specifically to Figures 4A and 4B the TAG is a 2-bit field which corresponds to the TAG field of data descriptors shown and described in the above reference application entitled "Segmented Address Development". PRN is a 2-bit field which defines the current ring number of the process to be used in determination of access rights to main storage. SEG is typically either a 12-bit or a 6-bit field which defines the segment number where instructions are being executed. The OFFSET is typically either a 16-bit or a 22-bit field which defines the address of the instruction within the segment SEG.

Figures 4C—4F show the format of segment descriptors with Figures 4C and 4D showing the first and second word of a direct segment descriptor whereas Figures 4E and 4F show the first and second word of an indirect segment descriptor. Segment descriptors are two words long each word comprised of 32 bits. Referring to Figures 4C—4D which show the first and second word respectively of a direct segment descriptor, P is a presence bit. If P equals one, the segment defined by the segment descriptor is present in main storage. If P equals zero, the segment is not present and a reference to the segment descriptor causes a missing segment exception. All other fields in a segment descriptor have meaning only if P equals one. A is the availability bit. If A equals zero, the segment is unavailable (or locked) and a reference to the segment causes an unavailable segment exception. If A equals one, the segment is available (or unlocked, and can be accessed). I is the indirection bit. If I equals zero, the segment descriptor is direct. If I equals one, the segment descriptor is indirect. U is the used bit. If U equals zero, the segment has not been accessed. If U equals one, the segment has been accessed. U is set equal to one by any segment access. W is the written bit. If W equals zero, no write operation has been performed on the segment. If W equals one, a WRITE operation has been performed on the segment. W is set to one by any WRITE operation. GS is the gating-semaphore bits. When the procedure call mechanism referred to above requires that the segment be a gating segment or when the process communication mechanism (not shown) requires that the segment be a segment descriptor segment (SD) the GS bits are examined. To be a valid gating segment, the GS bits must have the value 10. To be a valid SD segment, the GS bits must have the value 01. If a gating or SD segment is not required, these bits are ignored. The BASE is a 24-bit field which defines the absolute address in quadruple words of the first byte of the segment. This field is multiplied by 16 to compute the byte address of the segment base. The SIZE is a field which is used to compute the segment size. If the segment table number, subsequently referred to as STN, is greater or equal to zero but less than or equal to six, the SIZE field is 18 bits long. The STN is a field indicating the segment table entry STE for selecting a segment descriptor. If the STN is greater than or equal to 8 but less than or equal to 15, the SIZE field is 12 bits long. The number of bytes in the segment is equal to 16 times (SIZE+1). If SIZE equals zero, the segment size is 16 bytes. RD is the read access field. This is a 2-bit field which specifies the maximum EAR (effective address ring number) for which a read operation is permitted on the segment. (A procedure is always permitted to read its own segment if EAR equals PRN). WR is the write access field. This is a 2-bit field which specifies the maximum EAR for which a write operation is permitted on the segment and the minimum PRN at which the segment may be executed. MAXR is the maximum ring number. This is a 2-bit field which specifies the maximum PRN at which the segment may be executed. WP is the write permission bit. This bit indicates whether a WRITE operation may be performed on the segment. If WP equals zero, no WRITE operation may be performed. If WP equals one, a WRITE operation may be performed if EAR is greater than or equal to zero but less than or equal to WR. EP is the execute permission bit. This bit specifies whether the segment may be executed. If EP equals zero, the segment may not be executed. If EP equals one, the segment may be executed at any PRN for which PRN is greater than or equal to WR but less than or equal to MAXR. MBZ is a special field which must be set to zero by software when the field is created, before its initial use by hardware.

Referring to Figures 4E—4F the definitions of the various fields are similar as above however word 0 includes a LOCATION field and word 1 includes a

RSU field. The LOCATION field is a 28-bit field which defines the absolute address of a direct segment descriptor. The value in the LOCATION field must be a muliple of 8. The RSU field is a special field which is reserved for software use.

Figures 4G—4H show the format of the base registers (BR) which are used in the process of address computation to define a segment table number, a segment table entry number, an offset, and a ring number. There are typically 8 base registers as shown by reference numeral 116 on Figure 1. A base register is specified or identified as base register 0 through 7. The size of a base register is 32 bits long. The base register format of Figure 4G is utilized for small segment i.e. where STN is greater or equal to 8 but less than or equal to 15, whereas the format of base register of Figure 4H is utilized for large segments i.e. STN is greater or equal to zero but less than or equal to six. Referring to Figures 4G—4H, TAG is a 2-bit field which corresponds to the TAG of a data descriptor referenced previously. RING is a 2-bit field which contains the ring number associated with the segmented address for protection purposes. SEG is a field previously referred to, which identifies a segment described in a segment table. STN is the segment table number, and STE is the segment table entry number. OFFSET is a 16-bit field or a 22-bit field depending on segment table number, which defines a positive integer. The OFFSET is used in the process of address development as a pointer within a segment.

Referring to Figures 4I—4J there is shown the format of the T-register. The T-register is a 32-bit register containing a segment number and a 16-bit or 22-bit positive integer defining the relative address of the top of the procedure stack previously mentioned. The T-register is shown by reference numeral 114 on Figure 1. The various fields of the T-register have the same definition as described above.

Referring now to Figures 3 and 4A—4J a more defined description of absolute address calculation and access checking is made. In general absolute address calculation consists of fetching a segment descriptor specified by STN and STE and using the segment descriptors in four ways: access checking, computation of the absolute address, bound checking, and updating (U and W flags). As described in copending patent application No. 21630/74, (Serial No. 1,465,344) the absolute address may be direct or indirect and is derived by first deriving an effective address from STN, STE, and SRA (segment relative address). STN is extracted from bits 4 through 8 of the base register BR specified in the address syllable of an instruction. If STN is 7, an out of segment table word array exception is generated. STE is extracted from the base register specified in the address syllable. If STN 4:4 (i.e., beginning at bit 4 and including the next 4 bits) is greater than or equal to zero or less than or equal to six, STE is in a base register bits 8 and 9. If STN 4:4 (i.e. 4 bits beginning at bit 4) is greater than or equal to 8 but less than or equal to 15, STE is in a base register BR bits 8 through 15. The segment relative address SRA for direct addressing is computed by adding the displacement in the address syllable; the offset of the base register BR; and the 32-bit contents of an index register, if specified in the address syllable. The sum of these three quantities is a 32-bit unsigned binary integer which must be less than the segment size appropriate to the segment STN, STE.

Indirect addressing is developed by fetching a data descriptor and developing an address from that descriptor. The effective address of the data descriptor is computed as in the direct addressing case with the exception that the index register contents are not used. In developing the address from the data descriptor the effective address may be computed by an indirection to segment ITS descriptor and an indirection to base ITBB descriptor. If the descriptor is ITS the STN and STE are extracted from the descriptor in the same manner as from a base register. SRA is computed by adding the displacement in the descriptor and the contents of an index register as specified in the syllable. If the descriptor is an ITBB descriptor then STN and STE are extracted from the base register specified in the BBR field (i.e. the base register implied by ITBB descriptor) of the descriptor as in direct addressing. SRA is computed by adding the displacement in the descriptor, the offset of the base register, and the contents of an index register is specified in the address syllable.

As shown on Figure 3 the indirection process may be extended up to 16 levels.

Every effective address contains protection information which is computed in address development and checks for access rights by the ring protection hardware of the absolute address calculation mechanism. The effective address contains protection information in the form of an effective address ring number EAR (see Figures 2J and 2K of above application No. 21630/74, (Serial No. 1,465,344). The EAR is computed from the base register ring number BRN and from the current process ring number PRN by taking the maximum ring number. In developing the EAR for indirect addressing

a somewhat more tedious but essentially similar procedure as indirect addressing is used. In indirect addressing the EAR for extraction of the first descriptor (EAR 1) is once again the maximum of the ring number from the base register specified in the address syllable and the current process ring number PRN in the instruction counter 115 of Figure 1 and stored in 00 register 512 of Figure 5. The EAR for extraction of the second descriptor (EAR 2), of multiple level indirection is the maximum of:

a. EAR 1;

b. The ring number in the first descriptor if indirection is indirection to segment;

c. The ring number from a base register 116 utilized as a data base register BBR if the first descriptor is an indirection to segment descriptor ITBB.

The EAR for extraction of the data of multiple level indirection is the maximum of:

a. EAR 2;

b. The ring number in the second descriptor if it is an indirection segment descriptor ITS;

c. The ring number in one of the base registers utilized as a data base register BBR if the second descriptor is an indirection to base descriptor ITBB.

Referring now to Figures 5 and 6, the transfers and manipulation of the various type ring numbers will be described at the system level. Detailed logic block diagrams for effecting the transfers and operations of Figure 5 will be later described. Referring first to Figure 6 an associative memory 600 is utilized in segmented address development. The associative memory 600 comprises essentially a UAS associator 609 which has circuitry which includes associative memory cells, bit sense amplifiers and drivers, and word sense amplifiers and drivers (not shown). A word or any part of a word contained in UAS associator 609 may be read, compared to another word with a match or no match signal generated thereby, or be written either in whole or in a selected part of the associator 609. For example, US register 607 may contain a segment number which may also be in the associative memory 600. A comparison is made with UAS associator 609 and if a match is found a "hit" results. The match or "hit" signal is provided to encoder 610. The function of encoder 610 is to transform the "hit" signal on one of the match lines to a 4 bit address. Encoder 610 provides this 4 bit address to UAB associator buffer 611 so that the information contained in that particular location of UAB associator buffer 611 is selected. Information in UAB associator buffer 611 may be transferred to UV register 613 for temporary storage or for transfer to QA or QB bus 614 and 615 respectively. By thus locating a prestored segment number of the associative memory 600 (which may have been placed there after a generation of an absolute address) regeneration of the same address is not necessary. In the drawing of Figure 6, UAB associator buffer 611 is shown as storing a first and second word of a segment descriptor; however other types of information may just as well be stored therein. This buffer 611 provides a function similar to that of buffer 104 in the more generalised diagram of Figure 1.

As mentioned supra the development of an absolute address of an operand from an effective address is disclosed in patent application No. 21630/74, (Serial No. 1,465,344). Briefly and with reference to Figure 6 any of 8 base registers 602 are addressed via UG and UH registers 603 and 604 respectively which contain base register addresses from an instruction address syllable or base register specified by the instruction formats. The base register 602 contain such information as TAG, base register ring number BRN, segment table number STN, segment table entry STE and OFFSET as shown or contained by base registers 1 and 2 of the group of base registers 602. Writing into the base registers is performed under micro-op control by UWB logic 601. For example it is shown that information from the UM register 502 of Figure 5 may be written into bit positions (2, 3) of a selected base register; also information from the QA bus may be written into the base registers and provisions are made to clear a selected base register i.e. write all zeroes. Reading out of any of the base registers is performed by UBR logic 605. In general the UBR logic 605 permits the appropriate base register to be strobed out onto bus QA or QB, or into UN register 608. Note that UN register 608 holds bits 8 through 31 of the base registers which is the OFFSET part of the segmented address. Moreover UBR logic 605 when addressed by an address contained in instruction buffer IB (not shown) reads out the segment number SEG (which is comprised of STN and STE) into US register 607 via UBS transfer logic 606. The comparison of the segment number SEG in US register 607 with the associative memory 600 may then be performed as previously described. It will be noted that bits (4—15) of QA bus 614 may also be read into or from US register 607. Similarly bits (8—31) from QA bus 614 may read into UN register 608. Also bits (9—11) of the US register 607 may be read into QA bus 614 as denoted by US (9—11) arrow (the arrows into various register and/or logic circuitry denote the source of data and that followed

by a number denote the bit numbers of that data).

Referring now to Figures 5 and 6, a 2-bit UP register 501 stores the current process ring number PRN. The current process ring numbers PRN is obtained from bits 2 and 3 of the instruction counter (118 or Figure 1) via bits IC (2—3) of the QA bus 614 of Figure 6. Bits IC (2—3) of QA bus 614 are transferred to 2-bit UV register 503 under control of a micro-operation UV9QA0. The micro-operations are obtained from micro-instructions in the control store unit 110. (On Figure 5 the dot surrounded by a circle indicates a micro-operation and the first two letters of the name of the micro-operation indicate the destination of the data to be transferred; the fourth and fifth letters indicate the source of the data transferred; the third character indicates whether a full or partial transfer is made with F indicating a full transfer while the sixth character indicates whether the signal doing the transferring is high or low with even numbers indicating a low signul and odd numbers indicating a high signal. As an example of the use of this convention bits 2 and 3 on QA bus indicating the tail of the arrow QA (2, 3) indicate PRN is the PRN process ring number that is being transferred under control of the micro-op UV9QA0 which says the transfer is made to register UV, is a partial transfer of the bus QA, and the source of the data is the bus QA and is an unconditional transfer as indicated by the sixth character being 0. Transfer to UV register from QA bus source is unconditional. This 0 will be the corresponding seventh character in the logic file name of the subcommand UV9QA1ϕ. Once the process ring number PRN is transferred from the QA bus 614 to the UV register 503 another transfer takes place under control of the micro-operation UM9UV0 from UV register 503 to UM register 502. Finally another transfer takes place from UM register 502 to UP register 501 under control of a micro-operation UP9UM0.

Two bit register UM 502 is utilized to generate the effective address ring number EAR during ITS and ITBB (i.e. indirection to segment and indirection to base), (EAR=MAX (BRN, PRN, DRN/BBR (BRN) etc.) address formation for address syllable 1 and address syllable 2 type instruction format. The EAR is generated according to the rules previously enunciated by utilizing one or more tests shown in block 510 and the maximum of the ring number is obtained and stored in UM register 502 which stores the effective address ring number EAR (detailed logic or making the comparisons of block 510 are later shown and described in detail). The UO register is used to save address syllable 1 effective address ring number EAR in the event the address syllable 2 is being utilized to extract EAR 2.

Two-bit UV register 503, and 2-bit UW register 504 is utilized mainly as storage for various ring numbers that are obtained from the outside of the ring checking hardware of Figure 5 and transferred or processed to other parts of the ring checking hardware. For example the base register ring number BRN is transferred from bit positions 2 and 3 of UBS transfer logic 606 to UV register 503 under control of the micro-operation UVFBS0; the maximum ring number MAXR of word 2 of the segment descriptor (also shown stored in bits 36 and 37 of UAB associator buffer 611) is transferred from UAB buffer 611 to UV register 503 under control of the micro-operation UVFAB1; also bits 34 and 35 of UAB buffer 611 which is the write ring number WR is transferred to UV register 503 under control of micro-operation UVFAB0. UW register 504 has similar transfers of other ring numbers from various parts of the system. For example bits 34 and 35 which are the write ring number WR of UAB buffer 611 may also be transferred to UW register 504 under control of micro-operation UWFAB1; bits 32 and 33, the read RD ring number of UAB buffer 611 may also be transferred to UW register 504 under control of micro-op UWFAB0; also bits 0 and 1 of QA bus 614 may be transferred to UW register 504 under control of micro-operation UW9QA0. Note also several transfer paths of UW register 504 into UV register 503 under control of the micro-operation UV9UW0; the transfer path of UV register 503 into UM register 502 under control of micro-operation UM9UV0; the transfer path of UM register 502 into UP register 501 under control of the micro-operation UP9UM0; the transfer path of UP register 501 into UM register 502 under control of micro-operation UM9UP0; the transfer path of UM register 502 into UO register 512 under control of micro-operation UO9UM0; and finally the transfer path of UO register 512 into UM register 502 under control of the micro-operation UM9UO0.

Briefly therefore UP register 501 holds the current process ring number PRN; UM register 502 and UO register 512 are utilized for transfer operations and also to generate the EAR; UV register 503 may shore for various purposes and at different times the current process ring number PRN, the base register ring number BRN, the maximum ring number MAXR, the write ring number WR, or the read ring number RD. UW register 504 may at various times hold the read ring number RD, the write ring

number WR, and bits 0 and 1 of bus QA. UMR 505 is logic, the details of which are shown on Figure 8d, which compares the contents of registers UM and UV and produces the greater of the two values in the registers and this value is stored in UM register 502 under micro-operation control UMFMR0. This is one way of generating the effective address ring number EAR. UMR logic 505 may also produce the greater value of the contents of register UP or of bits 2 and 3 of UBS logic 606. This is another method and/or additional step in generating the effective address ring number EAR. UMR logic 505 is also utilized to determine whether or not a write violation has occurred by transferring a write ring number WR into UV register 503 and then comparing the contents of the UM register 502 (holding EAR) with the contents of UV register 503 in order to determine which one has the greater contents. Since UM register 502 stores the effective address ring number EAR a comparison of the UM register and the UV register will indicate whether EAR is greater than WR or vice versa. If WP (i.e. write permission bit in the segment descriptor) is equal to 1 and if EAR lies in the range of $0 \leq EAR \leq WR$ then a write operation may be performed into the segment. Note that UMR logic 505 may have inputs directly or indirectly from all registers 501—504, from other logic 506, 507 and also from UBS logic 606.

UWV logic 506 corresponds to the detail logic of Figure 8a. UWV logic 506 has inputs directly or indirectly from registers 501—504 and from logic 505, 507 respectively and generates an execute violation signal when a comparison of UW, UM and UV registers 504, 502, and 503 respectively indicates that the statements that the maximum ring number MAXR is greater or equal to the effective address ring number EAR, and that EAR is greater or equal to the write ring number WR are not true i.e. in order for a procedure to be able to execute in a given segment indicated by the effective address the maximum ring number MAXR must be greater or equal to the effective address ring number and the effective address ring number EAR must be equal or greater than the write ring number WR. UWV logic 506 also performs tests shown in block 510. Indications may be given that the contents of UW register is less than or equal to the contents of the UV register; the contents of the UM register is greater than or equal to the contents of the UV register; the contents of the UV register is equal to the contents of the UM register; the contents of the UV register is greater or equal to the contents of the UM register; and the contents of the UM register is greater than the contents of the UW register. Of course when performing these tests different values of ring numbers may occupy the registers.

UEP logic 507 corresponds to the detail logic of Figure 8b. UEP logic 507 in combination with UWV logic 506 generates the read violation exception. However the read violation exception may be overridden if the effective address ring number EAR equals the current process ring number PRN, since a procedure is always permitted to read its own segment, and if the segment number of the procedure segment descriptor (not shown herein) and the segment number of the address syllable utilized in generation of the effective address are the same.

To illustrate the overriding of the read violation signal assume that the effective address read number EAR is greater than the read number RD which would generate a read violation high signal which would be applied as one input of AND gate 522. However the read violation exception signal may not be generated even though there is a read violation signal if the following two conditions exists:

1. The effective address ring number EAR is equal to the process ring number PRN; i.e. the contents of register UM is equal to the contents of the register UP; and,

2. The segment number contained in the address syllable of the segment in which a procedure desires to read is equal to the segment number of the procedure segment descriptor (not shown) of the current procedure in execution and this is indicated by setting a bit called a P bit and located as the thirteenth bit of UE register 650. (UE register 650 is a store for the contents of UAS associator 609 when a "hit" has resulted by a comparison of the contents of US register 607). Since this example assumes that EAR equals PRN, UEP logic 507 will apply a high signal to AND gate 520 as one input, and since it is also assumed that the segment number SEG of the address syllable of the segment being addressed is equal to the segment number SEG of the procedure segment descriptor (not shown) of the currently executing procedure, then the P bit of the procedure segment descriptor will be set and hence the other input applied to AND gate 520 will be high thus enabling AND gate 520; a high signal is therefore applied to the input of inverter 521 resulting in a low signal at the output of inverter 521 which low signal is then applied as another input of AND gate 522. Since there is a low signal to AND gate 522 no read violation exception signal can be generated by amplifier 523 even if

the third input signal applied to AND gate 522 is high.

To illustrate how a read violation signal is generated and not overridden, assume that the output of UEP logic 507 indicates that the contents of UM register is not equal to the contents of UP register. Then that input to AND gate 520 would be low and hence AND gate 520 would not be enabled and its output would be low and would be applied to the input of inverter 521. Since the input of inverter 521 is low its output would be high which would be applied as one input of AND gate 522. If also the effective address ring number EAR is greater than the read ring number RD (i.e. contents of UM register is greater than contents of UW register) that signal would be high and would be also applied to another input of AND gate 522. AND gate 522 has still a third input which must also be high in order to enable AND gate 522. This third input is high when AND gate 526 is enabled. Since AND gate 526 has one input terminal which is high when the 00 terminal of URV1F flop 524 is low, AND gate 526 is enabled by applying the micro-operation read violation interrogate signal AJERVA to one input terminal of AND gate 526 while the 00 terminal of URV1F flop 524 is low. Thus AND gate 522 will have all input terminals high, generating the read violation exception signal.

The execute violation exception is generated in two ways. It was seen earlier that an execute violation signal results when UWV logic 506 indicates that the inequalities WR is less than or equal to EAR, and EAR is less than or equal to MAXR are not true. This high execute violation signal is applied to a one-legged AND gate 550 which in turn is applied to the input terminal of two-legged AND gate 553 via amplifier 552. When an execute violation interrogate micro-operation signal AJEEVA is applied as another input of two-legged AND gate 553, this gate is enabled which in turn generates the execute violation exception via amplifier 554. The other method by which the execute violation exception is generated by the execute violation hardware 511 is when the execute permission bit EP is not set. When this condition is true it is indicated by the seventh bit of UY register 613 being high; this bit is then applied to the input terminal of one-legged AND gate 551 which is applied as a high signal to one input terminal of AND gate 553 via amplifier 552. When the execute violation interrogate micro-operation signal AJEEVA goes high, AND gate 553 is enabled and generates an execute violation exception via amplifier 554.

The write violation exception is also generated in two ways. It was seen previously how the UMR logic 505 generates a write violation signal when EAR is greater than WR. This write violation signal is applied to one input terminal of AND gate 545. AND gate 545 is enabled when its second input terminal goes high thus generating a write violation exception through amplifier 547. The second input terminal of AND gate 545 goes high when AND gate 542 is enabled. AND gate 542 is enabled when the input signals applied to its input terminals are high. One input signal is high when UWV1F flop 541 is low which in turn applies a low signal to the input terminal of inverter 543 which in turn applies a high signal to one input terminal of AND gate 542; the other input signal is high when the write violation interrogate micro-op signal AJEWVA is high and this happens when it is desired to interrogate a procedure for the write violation exception. (Flip-flops URV1F, URN1F, and UWV1F are set low when any interrupts or softward occurs). (UWV2F, URV2F, and URN2F flip-flops are utilized to store back-up excess checking information for ring checking). The other method for generating a write violation exception is when the write permission bit WP is not set. This condition is indicated by bit 6 of UV register 613 being high. When this condition exists and the high signal (i.e. the sixth bit of UV register) is applied as one input of AND gate 546 and the interrogate signal

AJEWVA is high and applied as another input of AND gate 546, then AND gate 546 is enabled and a write violation exception occurs via amplifier 547.

Logic circuitry 591 comprised of flip-flops 532 and 533 in conjunction with amplifier 530 and AND gate 531 and inverter 530A permit the formation in register UM 502 of the maximum value of ring number (i.e. EAR) under control of a splatter instruction subcommand (not described herein) from the instruction fetch unit IFU. Assuming URN1F flip-flop 532 is set to logical 0 whereas URN2F flip-flop 533 is set to logical 1, then during the execution of the splatter subcommand, input terminal 531A of AND gate 531 will be high; therefore if flip-flop 532 is low (logical 0) then the signal will be inverted by inverter 530A and AND gate 531 will be enabled. Hence the maximum value of the contents of UP register 501 or bits 2 and 3 of logic vector UBS 606 will be strobed into UM register 502. Conversely if flip-flop 532 is a logical 1, then the contents of UM register 502 is not changed via the above mentioned sources and the EAR derived in UM register 502 via the addressing process of indirection is the one utilized. Flip-flop

533 is the back-up store for the EAR of address-syllable 2 when utilized.

Referring now to Figures 7 and 8 and Figure 5 there is a correspondence wherein the detailed logic for hardware in Figure 5 is shown in Figures 7 and 8 as follows: Figure 7a and UW register 504; Figure 7b and UV register 503; Figure 7c and block 590; Figure 7d and block 591; Figure 7e and block 592; Figure 7f and UP register 501; Figure 7g and UO register 512; Figure 7h and UM register 502; Figure 8a and UWV logic 506; Figure 8b and UEP logic 507; and Figure 8d and UMR logic 505.

Referring to Figure 7a, the UW register 504 is comprised of two flip-flops 715a and 720a respectively, each flip-flop capable of holding one bit of information of the UW register. Coupled to flip-flop 715a are 4 AND gates 711a—714a which are OR'ed together, with each gate (except gate 713a) having two input terminals, and with at least one signal applied to each input terminal. AND gate 714a has one of its input terminals coupled to the set terminal OW00010 of the flip-flop 715a. Flip-flop 715a is also coupled to the terminal H27 for receiving from a clock a timing signal called a PDA signal. Flip-flop 720a coupled to AND gates 716a—719a which are OR'ed together. One input terminal of AND gate 716a is coupled to an input terminal of AND gate 711a; one input terminal of AND gate 717a is coupled to one input terminal of AND gate 712a and one input terminal of AND gate 719a is coupled to an input terminal of AND gate 714a, whereas the other input terminal of AND gate 719a is coupled to the set terminal UW00110 of the flip-flop 720a. Flip-flop 720a is also coupled to the H27 terminal for receiving PDA pulses.

AND gates 701a—704a are OR'ed together each having their output terminal coupled to the input terminal of inverter 705a. AND gate 706a is coupled to amplifier 708a; whereas AND gate 707a is coupled to amplifier 709a; one input terminal of AND gate 706a is coupled to one input terminal of AND gate 707a. The output terminal of inverter 705a is coupled to one input terminal of AND gate 714a and 719a; the output terminal of amplifier 708a is coupled to the input terminal of AND gate 713a and the output terminal of amplifier 709a is coupled to the input terminal of AND gate 718a.

The signals applied to the inputs of AND gates and the signals derived as outputs from amplifier, inverters, or flip-flops are designated by letters forming a special code. Since both data signals and control signals are either applied or derived there are two codes, one code for the control signals and one code for the data signals. The code for the control signals are previously described in detail and is summarized here. Briefly the first two characters of a control signal indicate the destination of data to be transferred; the third character indicates whether a full or partial transfer is to be effected with the letter F indicating full transfer and any other character indicating a partial transfer; the fourth and fifth character indicates the source of the data, and if the source is identified by more than two letters only the last two letters need be used; the sixth and seventh characters are usually numerals and indicate whether the signal is high or low i.e. an odd numeral in the sixth position indicates assertion and an even numeral in the sixth position indicates negation; the seventh position indicates whether this is the first, second, third, etc. level of occurrence of the signal. Data, on the other hand, is indicated differently. The first three characters of data indicates the source of the data, the fourth and fifth characters which may be numerals indicate the bit positions where the data is located in the source, and the sixth and seventh position are similar to the control signals in that they indicate whether the signal is high or low and the level of occurrence of the signal. Generally the format itself indicates whether the signal is a control signal or a data signal and by reference to Figures 5 and 6 the source and destination may be determined. There are exceptions to this general rule and they will be spelled out in the specification, and addenum.

As an example of this convention it will be noted on Figure 7a that the following signals are control signals: UWFAB11, UWFAB10, UW9QA10. The following signals are data signals UAB3410, UAB3210, UAB3510, UAB3310, QA00110, and QA00010. The following signals are exception PDARG10 is a timing signal whose source is the PDA clock; UWHOL10 is a hold signal for holding the information in the flip-flops 715a and 720a UW0BK10 and UW1BK10 are back-up logic whose main function is to extend the input capability of flip-flops 715a and 720a by connecting the UW register which is in fact formed by flip-flops 715a and 720a, to bit zero and bit 1 represented by flip-flops 715a and 720a respectively; and finally USCLR10 is the clear signal for clearing and setting the flip-flops to zero.

As an illustration of the above mentioned convention herein adopted the signal UWFAB11 applied to the input of one-legged AND gate 702a is a control signal which transfers data (bits 34 and 35) contained in UAB associator buffer 611 (the U in the signal has been omitted) to UW register 504 and is a full transfer to the

UW register 1; the odd number indicates the signal is assertion. Signal UWFAB10 applied to the input of one-legged AND gate 703a is a control signal with the same source and destination as the signal applied to AND gate 702a except that bits 32 and 33 of UAB are transferred to UW register. The signal UW9QA10 applied to one-legged AND gate 704a is also a control signal wherein data is transferred from QA bus 614 to the UW register and may be a partial transfer. The signal QA00010 applied to AND gate 706a is a data signal where data is on QA bus 614 (the third position is not herein utilized since the first two positions adequately describe where the data is) and this data signal represents the bit identified as 00 on QA bus 614. The signal QA00110 is similar to the previous signal except the data identified by this signal is the data on position 01 of the QA bus 614. Thus by utilizing this convention and Figures 5 through 9 the ring protection hardware is fully defined and may be easily built by a person of ordinary skill in the computer art.

Referring to Figure 7b there is shown the detailed logic block diagram for UV register 503. Signal UVH0L10 is a hold signal for UV register 503 which is generated via inverter 703b when none of the one-legged AND gates 701b—708b has a high signal applied to it. UVH0L10 signal is applied to AND gate 723b and causes information stored in the UV register 503 to be held therein. Signal UVH0L1E coupled to the input of AND gate 704b and to the outputs of AND gates 705b—708b extends the number of control signals that may generate the hold signal UVH0L10. Signal UV0BK10 coupled to the outputs of AND gates 710b—713b and to the input of AND gate 722b is also utilized to extend the number of inputs signals that may be applied to flip-flop 724b. Signal UV1BK10 coupled to the outputs of AND gates 716b—718b and to the input of AND gate 727b similarly extends the number of input signals that may be applied to flip-flop 729b.

Referring now to Figure 7g there is shown the detailed logic block diagram of UO register 512. AND gates 701g—704g are OR'ed together and their output is applied as an input to inverter 705g. AND gates 706g—709g are also OR'ed together and their outputs are coupled to flip-flop 710g. Also one input of AND gate 709g is coupled to the U000010 terminal of flip-flop 710g. AND gates 711g—714g are also OR'ed together and are similarly coupled to flip-flop 715g. It will be noted also that an input of AND gate 706g is coupled to an input of AND gate 711g; an input of AND gate 707g is coupled to an input of AND gate 712g and an input of AND gate 709g is coupled to an input of AND gate 714g. The UOH0L10 signal generated by inverter 705g is also coupled to an input of AND gate 709g and 714g and is utilized to hold information in the UO register 512. XOO represents a ground, whereas XNU means unused input.

Figure 7f is a detailed logic block diagram of UP register 501. It is similar to Figure 7g described supra except that different signals from different destinations and different sources are applied.

Referring now to Figure 7h there is shown the detailed logic block diagram of UM register 502. AND gate 701h—704h are OR'ed together to produce the UMH0L10 hold signal via inverter 705h. AND gates 706h—709h are OR'ed together and are coupled to the input of AND gate 704h in order to extend the range of signals that may be applied to produce the UMH0L10 hold signal. Similarly AND gates 711h—714h are OR'ed together and coupled to the input of AND gate 723h in order to extend the range of signals that may be applied to flip-flop 730h; and also AND gates 716h—719h are OR'ed together and are coupled to the input of AND gate 727h in order to extend the range of signals applied to flip-flop 731h. A line 740h for applying the PDA signals to flip-flop 730h and 731h is coupled at point 734h and 735h respectively. The input of AND gate 703h is also expanded to provide two further inputs URN1F00 and IRNUM10 by coupling the output of amplifier 733h to the input of AND gate 703h.

Referring now to Figures 7c—7e there is shown detailed logic block diagrams of write exception control logic 590, IFU subcommand control logic 591, and read violation exception control logic 592 respectively. Referring first to Figure 7c there is shown flip-flops 705c and 710c which correspond to flip-flops 541 and 540 respectively. Under a micro-operation URW2F10 subcommand the information in flip-flop 710c is transferred to flip-flop 705c. The UWV1H10 hold signal is utilized to hold the information transferred to flip-flop 710c, whereas the UWV2H10 signal is utilized to hold the information transferred to flip-flop 705c. Similarly in Figure 7d information is transferred from flip-flop 710d to flip-flop 705d under micro-operation signal URNSW10, and in Figure 7e information from flip-flop 710e is transferred to flip-flop 709e under control of micro-operation signal URW2F10.

Referring now to Figures 8a, 8b and 8d there is shown detailed logic block diagrams of UWV logic 506, UWEP logic 507, and UMR logic 505 respectively. Referring first to Figure 8a there is shown logic for generating a high signal when one

of the test conditions 510 is true and also for generating the execute violation signal when the contents of UW register is less than or equal to the contents of UM register is less than or equal to the contents of UV register is not true. When the signal UWLEV10 is generated it indicates that the contents of UW register 504 is less than or equal to the contents of UV register 503. The logic for generating this signal was derived pursuant to the following Boolean expression:

$$X_1=\overline{(B\bar{C}\bar{D})+(AB\bar{D})\times(A\bar{C})}$$

Where X_1 represents the output of amplifier 805a and the various letters of the expression represent different input terminals of AND gates 801a—804a.

An indication that the contents of UV register 503 is greater than or equal to the contents of UM register 502 is had when UVGEM10 signal is generated. This signal is generated via inverter 820a in response to various inputs on AND gates 816a—819a which are OR'ed together and coupled to the input of inverter 820a. The logic for generating the UVGEM10 signal is made pursuant to the following Boolean expression:

$$X_2=\overline{(B\bar{C}\bar{D})+(AB\bar{D})+(A\bar{C})}$$

An indication that the contents of UM register 502 is greater than or equal to the contents of UV register 503 is indicated by generating signal UMGEV10 via inverter 810a in response to the various inputs of AND gates 806a—809a which are OR'ed together. The logic for generating this signal is derived from the following Boolean expression:

$$X_3=\overline{(B\bar{C}\bar{D})+(AB\bar{D})+(A\bar{C})}$$

(Wherein X_3 is the generated output signal).

Similarly the UVEQM10 signal is generated pursuant to the following Boolean expression:

$$X_4=\overline{(A\bar{C})+(\bar{A}C)+(B\bar{D})+(\bar{B}D)}$$

Generation of the UVEQUM10 signal indicates that the contents of the UV register 503 is equal to the contents of the UM register 502.

The generation of the UMGEW10 signal indicates that the contents of the UM register 502 is greater or equal to the contents of the UW register 504 and is generated pursuant to logic having the following Boolean expression:

$$X_5=\overline{(B\bar{C}\bar{D})+(AB\bar{D})+(A\bar{C})}$$

Generation of the UMGTW10 signal indicates that the contents of UM register 502 is greater than the contents of UW register 504 and this signal is generated by logic defined by the following Boolean expression:

$$X_6=(AB\bar{D})+\bar{C}(B\bar{D}+A)$$

The generation of the UWGMV00 signal indicates that the contents of UW register less than or equal to the contents of UM register less than or equal to the contents of UV register is not true. It is obtained when the UVGEM10 signal indicating that the contents of UV register is greater than or equal to the contents of the UM register, and the UMGEW10 signal indicating that the contents of the UM register is greater than or equal to the contents of the UW register are both high.

Referring now to Figure 8b a UMEQP10 signal is generated by logic derived from the following Boolean expression:

$$X_7=\overline{(A\bar{C})+(\bar{A}C)+(B\bar{D})+(\bar{B}D)}$$

When this signal is high it indicates that the contents of UM register 502 is greater than the contents of UP register 501.

Referring to Figure 8d there is shown the detailed logic block diagram for performing the operations of UMR logic 505 shown on Figure 5. One of the operations of this logic is to determine the maximum value of the contents of UP register 501 and of bits 2 and 3 of UBS logic 606. In order to do this there must be an indication whether contents of UP is less than the contents of UBS or the contents of UP is greater than the contents of UBS. The generation of UPBEB10 signal indicates that the contents of UP register 501 is less than or equal to bits 2 and 3 of UBS logic 606; whereas the generation signal UPGTB10 indicates that the contents of UP register 501 is greater than bits 2 and 3 of UBS logic 606. These signals are generated by logic which has been defined by the following Boolean expression:

$$X_8=\overline{(B\bar{C}\bar{D})+(AB\bar{D})+(A\bar{C})}$$

Where X_8 is the output of inverter 805d and the letters of the expression are various inputs of the AND gates 801d—803d.

To illustrate how the maximum value of the contents of UP register and UBS logic may be determined by the output signals UMPB010 and UMPB110 of amplifier 814d and 817d respectively, assume first that the contents of register UP are less than or equal to bits 2 and 3 of UBS logic because bit 2 is 1 and bit 3 is 1 whereas UB register

contains 01. This is indicated by the signal UPLEB10 being high and the signal UPGTB10 being low since it is the inverse of signals UPLEB10. This high UPLEB10 signal is applied to one input of AND gate 813d and also one input of AND gate 806d. If bit 2 of UBS logic is a 1 as indicated by signal UBS0210 then AND gate 813d is enabled and signal UMPB010 goes high and indicates that bit 2 on UBS logic is a 1. Moreover if bit 3 of UBS logic is a 1 indicated by input signal UBS0310 being applied as another input of AND gate 816d then AND gate 816d is enabled and signal UMPB110 is high or a 1. Therefore under the assumed conditions where bits (2, 3) UBS logic is greater or equal to the contents of UP register the maximum value of the two quantities is in UBS, and its number is binary 11 or decimal 4. Hence it is seen how a comparison is first made to determine which hardware contains the maximum, and then a determination is made as to the value of that maximum. By similar analysis one may see how the value of the UP register may be determined by signals UMPB010 and signals UMPB110 when the contents of UP register is greater than the second and third bit of UBS logic. Similarly the maximum value of UM register 502 or UV register 503 may be determined by signals UVGEM10 and UMGTV10 respectively, when UV register 503 is greater than or equal to UM register 502, and conversely when UM register 502 is greater than UV register 503.

Referring now to Figures 9a—9i a legend of symbols utilized in Figures 7 and 8 is shown. Figure 9a shows the symbol when there is a connection internally within the logic board. Figure 9b illustrates an output pin connection. Figure 9c indicates an input pin connection and is generally a source outside of the logic board illustrated. Figure 9d is the symbol utilized for an AND gate. Figure 9e is the symbol utilized for an amplifier; whereas Figure 9f is the symbol utilized for an inverter. Figure 9g illustrates three AND gates 901g—903g that are OR'ed together thus causing output 904g to go high when any one of AND gates 901g—903g is high. Figure 9h shows the symbol of a flip-flop having a 00 reset terminal and a 10 set terminal. A PDA line supplies the clock pulse for causing the flip-flop to switch states when other conditions are present on the flip-flop. Figure 9i represents a micro-operation control signal.

In order to enforce the ring protection scheme between procedures executing in different rings, the invention employs push-down stacks for its procedure linkage mechanism wherein a portion of each stack called a stack frame is dynamically allocated to each procedure. Different stack segments are used for each ring with one stack segment corresponding to one ring. Thus when a procedure is executed in ring RN its stack frame is located in the RN stack segment. Referring to Figure 10 there is shown three stack segments 1001—1003, with each stack segment having stack frames S1—S3 respectively. Ring 3 is assigned to stack segment 1001, ring 1 assigned to stack segment 1002 and ring 0 is assigned to stack segment 1003. Within each stack segment there is a procedure 11 associated with stack frame S1 of segment 1001, a procedure P2 associated with stack frame S2 of stack segment 1002 and a procedure P3 associated with stack frame S3 of stack segment 1003. The segmented addresses (i.e. segment number and segment relative address SEG, SRA) of the first bytes of the stack segments for rings 0, 1 and 2 respectively are located in stack base words SBW0—SBW2 respectively which are in turn located in process control block 104. Since the ring 3 stack segment can never be entered by an inward call (i.e. from a ring higher than ring 3) its stack starting address is not needed. Each stack frame S1, S2, S3 is divided into a working area 1005, 1006, 1007 respectively; an unused portion 1008, 1009, 1010, which is utilized for alignment purposes; a register saving area 1011, 1012, and 1013; and a communication area 1014, 1015, and 1016 respectively. The working area is utilized by its procedure as needed and may contain material required by the process such as local variables, etc. The saving area of the stack frame is utilized to save the contents of various registers such as the status register, the T-register and the instruction counter contents ICC. The communications area stores information which is needed to pass parameters between procedures. Prior to a call to a given procedure the user saves those registers he wishes saved and moreover loads into the communication area the parameters to be passed to the called procedure. When the call is made, the hardware saves the contents of the instruction counter and other specified registers to facilitate a return from the called procedure. Each procedure call creates a stack frame within a stack segment and subsequent procedure calls create additional frames. Hence a stack is created and consists of a number of contiguous parts called stack frames which are dynamically allocated to each procedure. These stacks reside in stack segments. Generally the first stack frame is loaded into the beginning of the segment and succeeding frames are loaded after it. The last frame loaded is considered the top

of the stack. A T-register 114 on Figure 1, locates the top of the stack for the currently active process. A procedure such as for example P1 which is executing in ring 3 may call a procedure P2 executing in ring 1 which in turn calls a procedure P3 which is now executing in ring 0. As each procedure is called it creates within its ring stack segment a stack frame (i.e. defining the environment for the procedure execution) and the T-register 114 is loaded which gives the address of the top of the stack for the current active process. The procedure P1 (as previously assumed) may call procedure P2 which in turn may call procedure P3 and since these calls are from a higher ring number to a lower ring number a ring crossing entailing an inward call is required and is accomplished in a manner to be described infra. During each change of procedure the necessary registers and parameters are saved in order to facilitate a return from the called procedure.

A procedure is always accessed through a procedure descriptor 1110 by means of the ENTER PROCEDURE INSTRUCTIONS. The format of the ENTER PROCEDURE INSTRUCTION 1100 is shown on Figure 11a. The operation code (OP) 1101 occupies bit positions 0 through 7. The complementary code 1102 is a one bit code and occupies bit position 8 to 9; if the complementary code is set to logical 1 the instruction is ENT, whereas if the complementary code is logical 0 the instruction is ENTSR and the base register must be base register 0 (BRO). The address syllable AS 1104 occupies bit positions 12 thru 31 and provides the address syllable AS of the procedure descriptor 1110. When an ENTER PROCEDURE INSTRUCTION requires a ring crossing a gating procedure descriptor 1120 is obligatorily accessed. This is indicated by the GS field 1302 of segment descriptor 1301 being set to logical 10. Generally the GS field is set to 10 when one of the ENTER PROCEDURE INSTRUCTIONS is utilized. As described in the application No. 21630/76, Serial No. 1,465,344, the segment descriptor is utilized to point to the base of the segment desired, in this instance the segment 1300 containing gate procedure descriptors GPD 1120. The first word of the segment 1300 containing the gating procedure descriptors (GPD's) is formatted as shown in Figure 11c. The TAG 1121 occupies bit positions 0 and 1 and must indicate a fault descriptor i.e. the TAG field must be set to logical 11. The Caller's Maximum Ring Number CMRN 1122 occupies bit positions 2 and 3, and indicates the maximum ring from which a calling procedure through the gated procedure descriptor GPD is legal. A call violation exception is generated if the caller's ring number is greater than CMRN 1122. The gated procedure descriptor address boundary GPDAB 1124 occupies bit positions 10 through 31 and it must be greater than the segment relative address SRA (i.e. the GPD's displacement in the segment of procedure descriptors 1300), otherwise an illegal GPD access exception occurs. Thus a gating procedure descriptor GPD is utilized as the first word of the segment containing procedure descriptors and is utilized to determine whether the caller has a right to access the segment via the caller's maximum ring number CMRN and whether or not the procedure descriptor called is within the gating procedure descriptor's address boundary. Once it is determined that there is a legal call to the segment and the caller has a right to enter the segment the address is obtained from the address syllable AS 1104 of enter instruction 1100 and the required procedure descriptor 1110 (see also Figure 13) is accessed. The format of procedure descriptor 1110 is shown on Figure 11b and is comprised of two 32 bit words—word 0 and 1 respectively. Word 0 contains the segmented address 1113 of the entry point EP of the procedure desired. The segmented address, as is the case with the segmented address of any operand, is comprised of the segment number SEG and the segment relative address SRA. Word 0 of the procedure descriptor includes an entry point ring number EPRN 1112 and a TAG field 1111. The value of the TAG is interpreted as follows:

a. if the TAG contains logical 00 the procedure descriptor is direct;

b. if the TAG is logical 01 the procedure descriptor is an extended descriptor and includes word 1 making a total of two words;

c. if the TAG is logical 10 the procedure descriptor is indirect and an illegal procedure descriptor exception occurs; and

d. if the TAG is logical 11 it is a fault procedure descriptor and an exception occurs.

Word 1 of the procedure descriptor is 32 bits long and is utilized when the TAG indicates an extended descriptor and contains the segmented address of a linkage section whose contents are loaded in base register BR 7 at procedure entry time.

Referring to Figure 12 a portion of the ENT instruction is shown and more specifically that portion which pertains to the ring crossing and ring checking requirements. The ENT instruction is called, 1201 and a comparison is made 1202 wherein the segmented part of the base register BRn is compared to the segmented part of the address of the T register, and if

they are not equal an illegal stack base register 1208 is indicated. If on the other hand they are equal another comparison 1203 is made wherein the 30th bit including the next two bits (i.e. bits 30 and 31) of base register, BRn is compared to 0 and if it is not equal to 0, then once again an illegal stack base register 1208 is indicated. If it is equal to 0 it indicates that the contents of BRn is aligned with respect to the word boundary and another comparison 1204 is performed to determine that the TAG of BRn (i.e. the two bits starting from bit 0) is equal to 0. A TAG having a logical 0 indicates information is accessed via a direct descriptor which is one of the requirements of the ENT instruction. If the TAG (i.e. bits 0 and 1 of BRn) is equal to 0 then the functions stated in flow charts of Figures 14 through 16 are performed (see flow chart Figure 12 block 1205). If these meet the necessary requirements a further check 1206 is made to determine whether the segment relative address of the entry point which was given (SRA_{EP}) is even, because instructions start on a half-word boundary. If it is not even then an illegal branch address exception is generated 1209 however if it is legal the ENT instruction is executed 1207 via further steps not shown.

Referring now to the flow charts of the access checking mechanism Figures 14—16, generally the following operations are performed each time the instruction ENTER PROCEDURE is issued:

a. the caller's right to call the callee is checked by first determining from the second word of the segment descriptor the call bracket in which the caller is executing. (The call bracket is determined by taking the minimum ring number from the write ring number field WR and the maximum ring number from the maximum ring number field MAXR).

b. a decision is made about the next process ring number by determining whether the caller is in the same call bracket as the callee, which implies don't do anything; whether the caller is in a call bracket requiring that he make an outward call in which case an exception condition is generated which is handled by a mechanism not described herein; or finally whether the caller is in a call bracket which requires an inward call (i.e. going to a call bracket which requires ring crossing from a larger ring number to a smaller ring number in which case the ring crossing must be at a valid entry point EP and the entry point must be validated).

c. a stack frame is created for the callee (i.e. space in the aforementioned format of the appropriate segment is allocated), and the stack frame and the stack frame registers are updated;

d. a branch to the entry point of the procedure pointed to by the procedure descriptor is performed.

Referring now to Figure 14 the access checking is started 1401 by obtaining the address syllable AS containing the effective address ring number EAR, the segment number of the procedure descriptor SEG_{PD}, and the segment relative address of the procedure descriptor SRA_{PD}. Having developed this information the procedure descriptor 1110 is fetched 1403 from (SEG_{PD}, SRA_{PD}) ignoring access rights to scratch pad memory. The procedure descriptor 1110 will yield the TAG which determines whether the descriptor is direct, extended, indirect, or a fault descriptor; the entry point ring number EPRN; the segment (SRA_{EP}) which contains the entry point and the segment relative address (SRA_{EP}) of the entry point. The TAG is tested 1404 to determine whether the descriptor 1110 is direct, extended, indirect or a fault descriptor by checking its field in accordance to the code hereinbefore described. Only a direct or extended procedure descriptor is legal. An indirect or fault descriptor is illegal and upon access invokes an exception mechanism not herein described. Once it is determined that a legal procedure descriptor has been accessed the actual call right checking begins at point A 1405.

Referring now to Figure 15 and continuing from point A 1405 the maximum ring number MAXR, the write ring number WR, and the execute permission bit EP of the segment containing the entry points SEG_{EP} are fetched; this information is contained in the segment descriptor for the segment containing the entry points (SEG_{EP}). The write ring number WR is compared to the maximum ring number MAXR 1503 and if the write ring number WR is greater than the maximum ring number MAXR the segment is nonexecutable and an execute violation exception 1513 occurs. If the write ring number WR is less than or equal to the maximum ring number MAXR then the execute permission bit EP is compared to logical 1 and if the EP bit is not logical 1 then once again an execute violation exception 1513 occurs; however if the EP bit is equal to one the effective address ring number EAR of the calling procedure is maximized with EPRN to give a new EAR_2,—{MAX (EAR_1 EPRN)} where EAR_1 is the maximum of PRN as found in the instruction counter IC, and all ring numbers in base registers and data descriptors, if any, found in the path which leads to the procedure descriptor. The

effective address ring number EAR_1 is then compared 1506 to the maximum ring number MAXR of the MAXR segment descriptor of SEG_{EP} which is the maximum ring number at which a procedure may execute. If EAR_2 is greater than MAXR the procedure call is an inward call which requires that the procedure be entered by a valid entry point and the access checking operation branch to point B 1507. The following checking operations are then performed:

a. the SEG_{EP} is checked to determine if it is a legal gate segment; and,

b. the caller's maximum ring number CMRN is checked to determine if it is greater than or equal to the effective address ring number EAR of the caller.

If these conditions are not true then an illegal gate segment exception 1603 or call violation exception 1615 occurs.

Referring now to branch point B 1507 of Figure 16 the first check 1602 that is made is to determine whether or not the segment which contains the procedure descriptors is a gate segment. This is done by examining the Gating/Semaphore field GS of the segment descriptor pointing to the segment of procedure descriptors, to determine if it is set to logical 10. If the GS field of the segment descriptor of the segment containing procedure descriptors is set to 10 it is then a gate segment and the first word of the segment containing procedure descriptors is a gated procedure descriptor GPD 1120 of Figure 11C and Figure 13. The first word 1120 of the segment containing procedure descriptors is then fetched from address SEG_{PD}, 0 ignoring access rights to scratch pad memory. It will be noted that the TAG field of the first word 1120 of the segment containing procedure descriptor SEG_{PD} 1300 must be a logical 11 (Figure 13) which indicates it is a fault descriptor. Moreover the MBZ field must be set to zero. These conditions are checked by hardware/firmware (arithmetic logic unit) stop 1605 and if these conditions do not hold an illegal gate segment exception 1603 results. However if these conditions do hold a check 1606 is further made to determine that the segment relative address of the procedure descriptor SRA_{PD} 1110 is a multiple of 8. If the condition of step 1606 does not hold an illegal system object address exception 1613 results otherwise the next step 1607 is performed. Step 1607 checks to determine whether or not the segment relative address of the procedure descriptor SRA_{PD} is within the address boundary GPDAB 1124 of the gated procedure descriptor 1120; if it is not within that address boundary it is an illegal procedure descriptor and an illegal GPD gated procedure descriptor access exception 1614 occurs. However if it is within the address boundary of the gated procedure descriptor (i.e. SRA_{PD} is less than GPDAB) then the caller's right to call the callee is checked 1608. This is performed by comparing the effective address ring number EAR_2 to the caller's maximum ring number CMRN 1122 as found in the first word 1120 of the segment of procedure descriptors 1300. If EAR_2 is greater than the caller's CMRN a call violation exception 1615 occurs which indicates that the caller in this particular instance has no right to legally call inward i.e. from a higher ring number to a lower ring number. On the other hand if EAR_2 is equal or less than CMRN, then the inward call is legal and a check is made 1609 to determine that the process ring number PRN which is the current process ring number found in the instruction counter IC just before the call was made is less than the maximum ring number MAXR of SEG_{EP}; and if it is the accessing mechanism branches to point C 1508, otherwise a new process ring number NPRN is calculated and set to a maximum ring number MAXR 1611. Generally the effective address ring number EAR_2 is the same as the process ring number PRN of the caller. Sometimes however, in cases where it is necessary to give maximum assurance that the caller will not be denied access to a given segment the EAR_2 is greater than the PRN. In those cases 1 RN is forced to take the value of EAR_2 in order to make sure that the call is returned to the maximum ring number upon an exit. To this point it will be noted that this checking mechanism was invoked because the EAR_2 was greater than the MAXR hence greater than the top of the call bracket of the procedure and hence an inward call was necessary which necessitated going through a valid gate, and the mechanism included these gating checks. By branching back to C 1508 (Figure 15) a further check 1509 is made to determine then that the process ring number PRN is greater than the write ring number WR of SEG_{EP} which in this context is the minimum ring number at which a procedure may execute. If the write ring number WR is greater than the process ring number PRN an outward call exception 1514 occurs. However if WR is less than or equal to PRN the call is legal and NPRN is set to PRN 1510.

Having made the above checks the inward call is made, and after performance of the desired operation a return back to the original point of the program in execution is made by the EXIT INSTRUCTION. During the ENTER INSTRUCTION the instruction counter IC

was saved in the saving area of the caller's stack frame before making the call. Moreover the caller's ring number was also saved during the ENTER INSTRUCTION and this was saved in base register 0 BRO.

The format of the EXIT INSTRUCTION 1130 is shown on Figure 11D. The operation code OP 1131 is found in bit positions 0—7 and the complementary code C 1133 is found in bit positions 12—15. The complementary code allows other instructions to use the same 8 bit op code. The MBZ field 1132 in bit positions 8—11 must be 0 otherwise an illegal format field exception occurs. (BRO is generally a pointer to the communications area of the caller's stack frame).

In performing the EXIT INSTRUCTION it is necessary to perform predetermined checks in order to ascertain that the caller didn't change his image which would permit him to operate a a different privilege than was intended. Referring to Figure 17 the first check performed 1701 is to determine if the TAG of the instruction counter content (ICC) indicates a direct descriptor. A logical 00 in the TAG field indicates that it is direct if it is not an illegal stack data exception 1702 occurs, whereas if it is equal to 0 the ring field in the instruction counter content ICC is set to the new process ring number NPRN 1703. This sets the new process ring number NPRN to what it used to be when the call was first made. However further checks are made in order to ascertain that there was no further cheating. Hence the base register 0 ring number located at bit position 2 and extending for 2 bit positions from and including bit position 2 must be equal to the new process ring number NPRN 1704. (It will be recalled that when the ENTER INSTRUCTION was called the ring number of the caller before the call was made was stored in bits 2 and 3 of base register 0 (BRO). If check 1704 indicates that the new process ring number NPRN is not equal to the ring number in bit positions 2 and 3 of the base register 0 (BRO) an illegal stack data exception 1702 occurs. The next check 1705 determines whether an inward or an outward return must be performed. Since an inward call was previously performed an outward return is implied in order to reach the original point from which the procedure was called. Moreover since the invention does not permit an outward call there is never a necessity to return inward. Hence the new process ring number NPRN is compared to the process ring number PRN 1705, and if NPRN is less than PRN an inward return is implied and an inward return exception 1706 is generated. However if check 1705 is passed successfully (i.e. NPRN is greater or equal to PRN) then a check is made to determine that a return is made to the segmented address SEGr that called the procedure and a return to the call bracket of the calling procedure is made and moreover that the execute bit EP is set. This is performed by fetching the segment descriptor SEGr of the calling procedure 1707 and making checks 1709, 1711, 1712. In performing checks 1709, 1711, 1712, check 1709 and 1711 determine that the new process ring number NPRN is greater than the minimum ring number WR but less than the maximum ring number MAXR (i.e. that the ring number is in the call bracket of the calling procedure where it should be). Finally check 1712 makes sure that the execute permission bit EP is set to 1. Thus a full cycle is concluded a call was performed via an ENTER INSTRUCTION; the required operation or processing was performed via the called procedure; then a return via an EXIT INSTRUCTION to the calling procedure was performed.

Having shown and described the preferred embodiment of the invention, those skilled in the art will realize that many variations of modifications can be made to produce the described invention and still be within the scope of the claimed invention.

Glossary of Terms

JOB—The job is the major unit of work for the batch user. It is the vehicle for describing, scheduling, and accounting for work he wants done.

JOB STEP—A smaller unit of batch work. It is generally one step in the execution of a job consisting of processing that logically belongs together.

TASK—The smallest unit of user-defined work. No user-visible concurrency of operation is permitted within a task.

PROGRAM—A set of algorithms written by a programmer to furnish the procedural information necessary to do a job or a part of a job.

PROCESS GROUP PLEX—The system's internal representation of a specific execution of a job.

PROCESS GROUP—A related set of processes, usually those necessary for performance of a single job step.

PROCESS—The controlled execution of instructions without concurrency. Its physical representation and control are determined by internal system design or convention.

Glossary of Terms (cont.)

PROCEDURE—A named software function or algorithm which is executable by a computational processor without concurrency. Its physical representation (code plus associated information, invocation, and use are determined by internal system or designed convention).

LOGICAL PROCESS—The collection of hardware resources and control information necessary for the execution of a process.

ADDRESS SPACE (SEGMENTATION)—The set of logical addresses that the CPU is permitted to transform into absolute addresses during a particular process. Although a processor has the technical ability of addressing every single cell of timing memory, it is desirable to restrict access only to those cells that are used during the process associated with the processor.

LOGICAL ADDRESS—An element of the process address space such as for example segment number SEG and Displacement D.

BASIC ADDRESS DEVELOPMENT—A hardware procedure which operates on a number of address elements to compute an absolute address which is used to refer to a byte location in core.

PROCESS CONTROL BLOCK—A process control block PCB, is associated with each process and contains pertinent information about its associated process, including the absolute address of tables defining the segment tables the process may access.

J. P. TABLES—A collection of logical addresses for locating a process control block associated with a process.

SEG_{PD}—The segment which contains the procedure descriptor.

SEG_{EP}—The segment which contains the entry point, as found in the procedure descriptor.

PRN—The process ring number, found in the instruction counter IC just before the call, or calculated by the ENTSR instruction.

EAR—The effective address ring number which is the maximum of:
(a) the process ring number PRN as found in the IC: or
(b) all ring numbers in the base register and data descriptors (if any) found in the path which leads to the procedure descriptor from the call instruction, including the entry point ring number EPRN located in the procedure descriptor itself.

MAXR—The maximum ring number at which a procedure may execute; MAXR is found in the segment descriptor of SEG_{EP}.

WR—The minimum ring number at which a procedure may execute; WR is found in the segment descriptor of SEG_{EP}.

EP—Execution permit bit found in the segment descriptor of SEG_{EP}.

CMRN—The caller's maximum ring number, as found in the first word of the segment SEG_{PD} if this segment is identified as a gate segment (i.e. with the code "gate" set).

NPRN—New process ring number.

EPRN—Entry point ring number (found in the process procedure descriptor).

Addendum

Signal Name	Type	Function
(1) WSCLR	Control	Clears register to which it is connected.
(2) PDARG	Control	Clock Signal PDA.
(3) PDURGIT	Connecting	Pin connected to PDA at one end and resistor at the other.
(4) UWOBK	Connecting	Expands inputs to UW register.
(5) UWHOL	Control	Holds information in register to which it is connected.
(6) UW1BK	Control	Same as UWOBK but is connected to different input terminal of UW register.
(7) UW00000		Reset terminal of one flip-flop of register UW.
(8) UW00010		Set terminal of flip-flop of register UW.
(9) UW00100 UW00110		Same as 7+8 but different flip-flop.
(10) UVSPS	Control	Spare Control Input.

Addendum (cont.)

Signal Name	Type	Function
(11) UVSPD	Data	Spare Data Input.
(12) UVOBK	Expander	Same as UW0BK and UW1BK, but it connects different registers and gates.
(13) UV00000 UV00010 UV00100 UV00110		Same as UW00000, UW00010, UW00100, UW00110, but applies to flip-flop UV.
(14) UWV1S	Control	Control input for UWV1F.
(15) UWV1D	Data	Data input for UWV1F.
(16) UWV2F	F/F	Write control flip-flop.
(17) UWV1S UWV2S	Control	Control unit for UWV1F, UWV2F.
(18) UWV1D	Data	Data input for UWV1F.
(19) UWV1H	Control	Hold UWV1F flip-flop.
(20) UWV1C	Control	Clear UWV1F.
(21) UWV2C	Control	Clear UWV2F.
(22) URN1S URN2S	Control	Control inputs for URN1F, URN2F.
(23) URN1D	Data	Data Input for URN1F.
(24) URNSW	Control	Transfer URN1F to URN2F and URN2F to URN1F.
(25) URN2F	F/F	Control loading max (UP, UBS2, 3 to UM).
(26) URN1H	Control	Hold URN1F flip-flop.
(27) URN2C	Control	Clear URN2F.
(28) URW1S URW2S	Control	Control inputs for URV1F, URV2F.
(29) URW1D	Data	Data Input for URV1F.
(30) URV2F	F/F	Read control flop.
(31) XNU		Indicates terminal not used herein.
(32) XOO		Grounded Input.

WHAT WE CLAIM IS:—

1. An internally programmed data processing apparatus CPU having a virtual memory system, and being responsive to internally stored instruction words for processing information and having stored in said virtual memory system a plurality of different types of groups of information each information group-type associated with an address space bounded by a segment having adjustable bounds, and comprising means for protecting the information in said-virtual memory system from unauthorized users by restricting accessability to the information in accordance to levels of privilege, said means comprising in combination with an access checking mechanism;

(a) first means arranged in operation to store in said virtual memory system at least one segment table comprising a plurality of segment descriptors with each segment descriptor being associated with a predetermined one of said segments and each segment descriptor having a predetermined format containing an access information element and a base address element in predetermined positions of said format, said base address element being used for locating in said virtual memory system the starting location of a selected one of said segments, and said access information element for specifying the minimum level of privilege required for a predetermined type of access that is permitted in a selected one of said segments;

(b) a plurality of second means having a predetermined format, communicating with said first means, arranged to store in a predetermined portion of said second means, a segment number SEG for identifying a segment table and the location of a segment descriptor within said segment table, said second means also being arranged to store in a predetermined other portion of said second means, an offset address within the segment identified by said segment descriptor said offset address locating from said segment base the first byte of a word within said segment;

(c) third means responsive to an address syllable element of an instruction being executed for addressing one of said plurality of second means;

(d) fourth means arranged to store a displacement from said address syllable,

(e) fifth means, communicating with said first, second, third and fourth means, arranged to add the displacement D and said base address to said offset; and,

(f) sixth means responsive to said access

information element in a selected one of said segment descriptors, restricting the accessability to the segment associated with said selected one of said segment descriptors in accordance to the level of privilege and the type of access specified in said access information element, wherein each group-type of information is associated with a predetermined ring number indicative of a level of privilege said level of privilege decreasing as the associated ring number increases comprising means for determining the maximum effective address ring number EAR (i.e. minimum level of privilege) of a selected process to access a selected group of information, said means comprising;

(a) first means to store first information indicating the maximum ring number RD (i.e. minimum level of privilege) required to read information from said selected group;

(b) second means to store second information indicating the maximum ring number WR (i.e. minimum level of privilege) required to write information into said selected group;

(c) third means to store third information indicating the maximum ring number MAXR (i.e. minimum level of privilege) required to process information from said selected group; and,

(d) fourth means communicating with said first, second and third means, to determine the maximum of the contents of said first, second and third means whereby the effective address ring number EAR is generated.

2. Apparatus according to claim 1, wherein said second means for storing the maximum ring number WR additionally indicates the minimum ring number WR (i.e. maximum level of privilege) required to process information from said selected group.

3. Apparatus according to claim 1 or claim 2, wherein said fourth means to generate the effective address ring number comprises a comparator for comparing binary numbers.

4. Apparatus according to any one of claims 1 to 3 wherein the sixth means restricting the accessibility to the segment includes comparator means, communicating with said second means, to compare the effective address ring number EAR with the write ring number WR, and further including means communicating with said comparator means to generate a write-violation-exception signal when EAR is greater than WR.

5. Apparatus according to claim 4, wherein the sixth means restricting the accessibility to the segment includes seventh means, communicating with said second and third means thereof to compare the maximum ring number MAXR and the write ring number WR with the effective address ring number EAR, and further including eighth means, communicating with said seventh means for generating an execute-violation-exception signal when the MAXR is not equal or greater than EAR which in turn is not equal or greater than WR.

6. Apparatus according to claim 5, wherein in that the sixth means restricting the accessibility to the segment includes ninth means, communicating with said first means, for comparing the effective address ring number EAR with the read ring number RD, and further including tenth means, communicating with said ninth means, to generate a read-violation-exeption signal when EAR is greater than RD.

7. Apparatus according to claim 6, wherein in that the sixth means restricting the accessibility to the segment includes eleventh means to store a process ring number PRN of a currently executing process, and also including twelfth means to communicate with said eleventh means, and further including thirteenth means communicating said said twelfth means for overriding said read-violation-exception signal when the effective address ring number EAR is equal to the process ring number PRN of the currently executing process.

8. Apparatus according to any one of the preceding claims wherein the access checking mechanism supervises transfer of control of said CPU from a first selected procedure (i.e. caller) having a first ring number indicative of a minimum level of privilege associated with said caller, to a second selected procedure (i.e. the callee) having a second ring number associated with said callee indicative of a minimum level of privilege associated with said callee, said access checking mechanism comprising

(a) first means for checking the caller's right to call the callee;

(b) second means, communicating with said first means, to compare the caller's ring number to the callee's ring number;

(c) third means responsive to said second means to permit a transfer of control of said CPU from said caller to said callee when the ring number of the caller is greater than the ring number of callee (i.e. inward call); and,

(d) fourth means also responsive to said second means to deny a transfer of control of said CPU from said caller to said callee when the ring number of said caller is less than the ring number of the callee (i.e. outward call).

9. Apparatus according to claim 8, wherein the access checking mechanism includes a plurality of ring stack-segment means each of said ring stack-segment means having associated with it a ring stack-segment number, indicative of the minimum level of privilege required by a selected one of said procedures to access a selected one of said ring stack segments.

10. Apparatus according to claim 9 wherein there are four ring stack segment means having ring numbers 0 to 3 respectively.

11. Apparatus according to claim 9 or claim 10 wherein the access checking mechanism includes stack-frame-element means associated with selected ones of said procedures, said stack-frame-element means being grouped within said ring stack-segment means in accordance with the ring number of the associated procedure of said stack-frame-element means, said stack frame element means to save said register of said caller prior to passing control to said callee.

12. Apparatus according to claim 11, wherein the access checking mechanism includes first sub-element means, responsive to said first, second, third and fourth means, for communicating between a selected one of said stack-frame-means in a first ring stack-segment being associated with one ring number, and a selected other of said stack-frame-means in a second ring stack-segment associated with another ring number.

BARON & WARREN,
16, Kensington Square,
London, W8 5HL.
Chartered Patent Agents.

Printed for Her Majesty's Stationery Office, by the Courier Press, Leamington Spa, 1977
Published by The Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 1

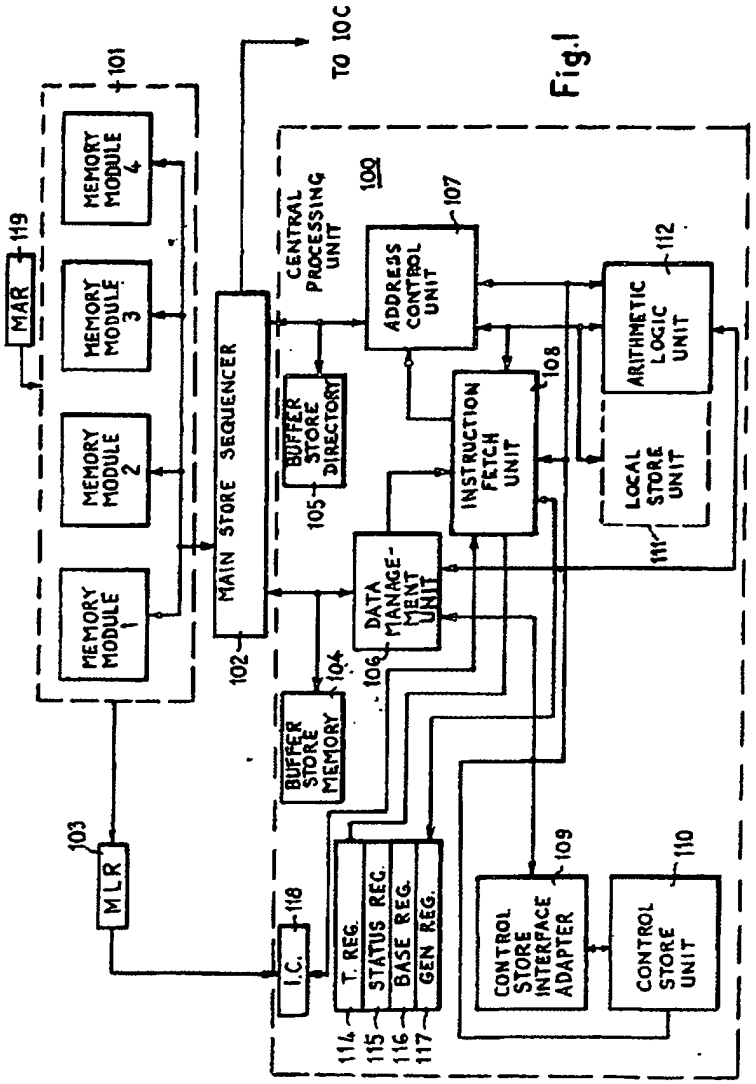


Fig.1

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 2

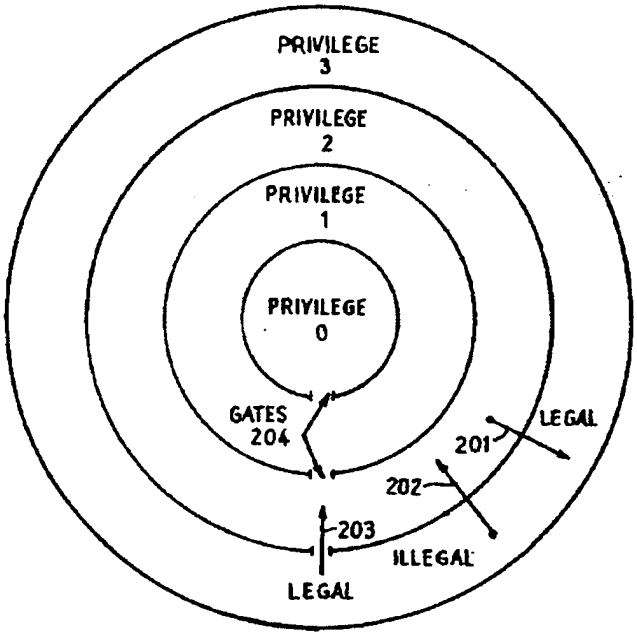
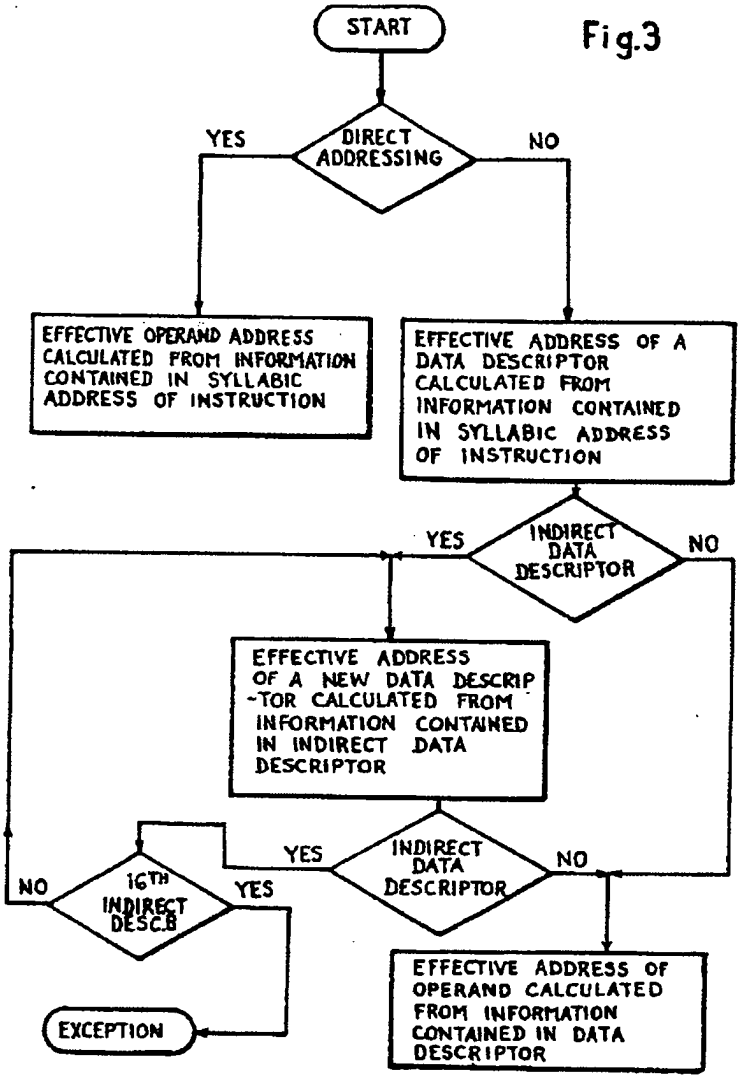


Fig.2

Fig.3



1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 4

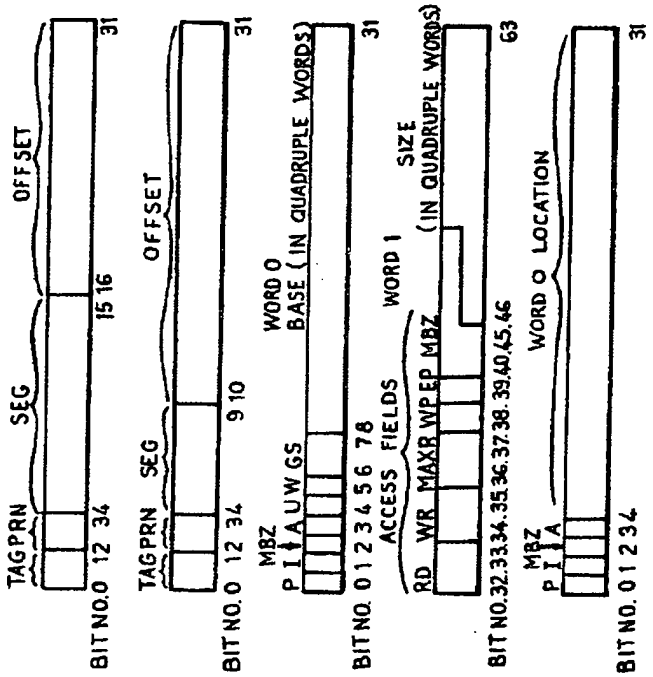


Fig.4A Fig.4B Fig.4C Fig.4D Fig.4E

1483282 COMPLETE SPECIFICATION
29 SHEETS

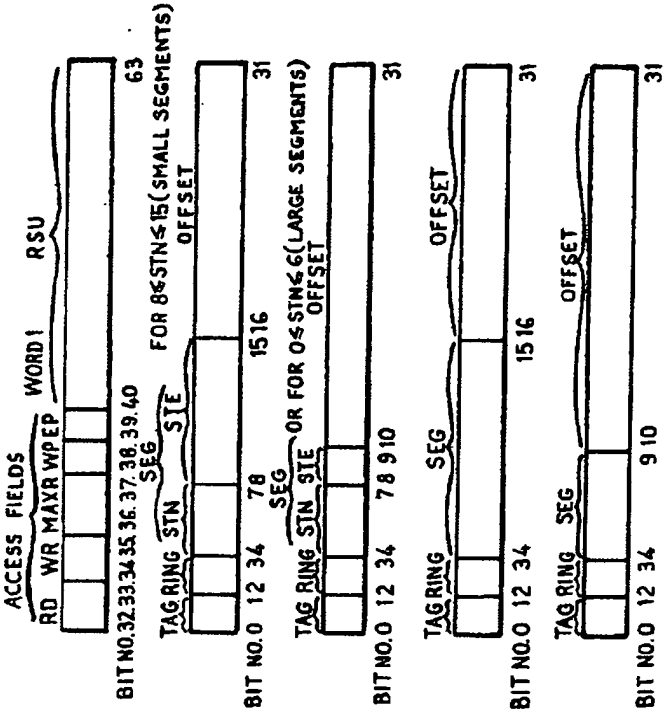


Fig.4F

Fig.4G

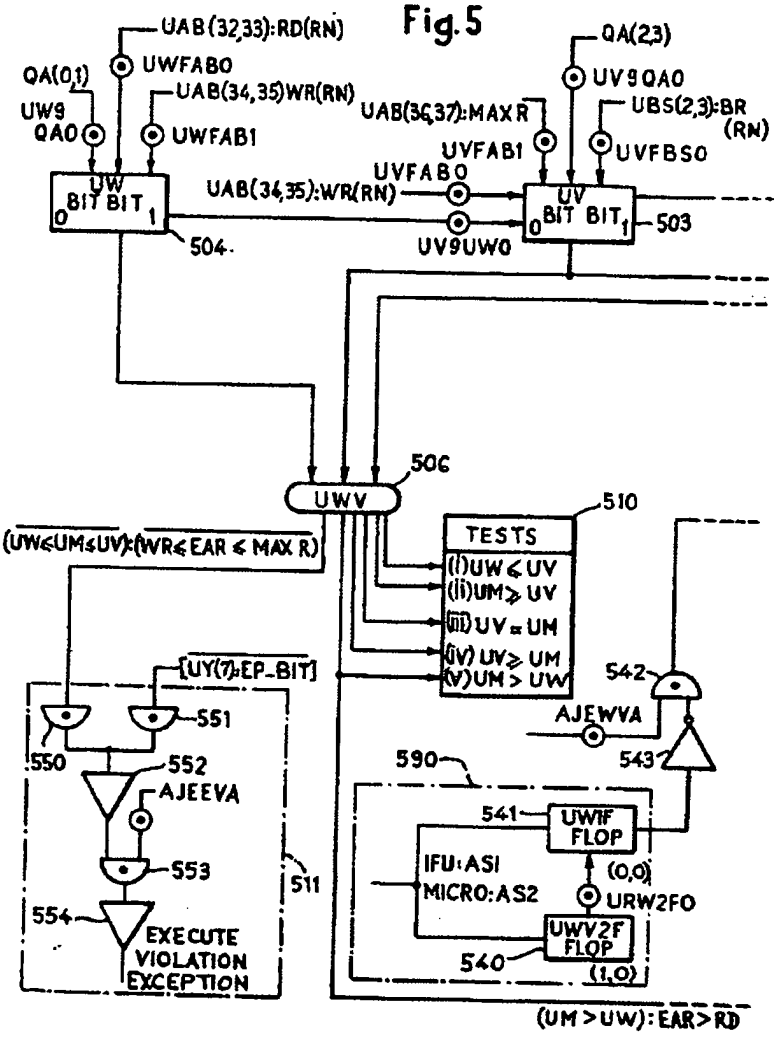
Fig.4H

Fig.4I

Fig.4J

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 6

Fig. 5



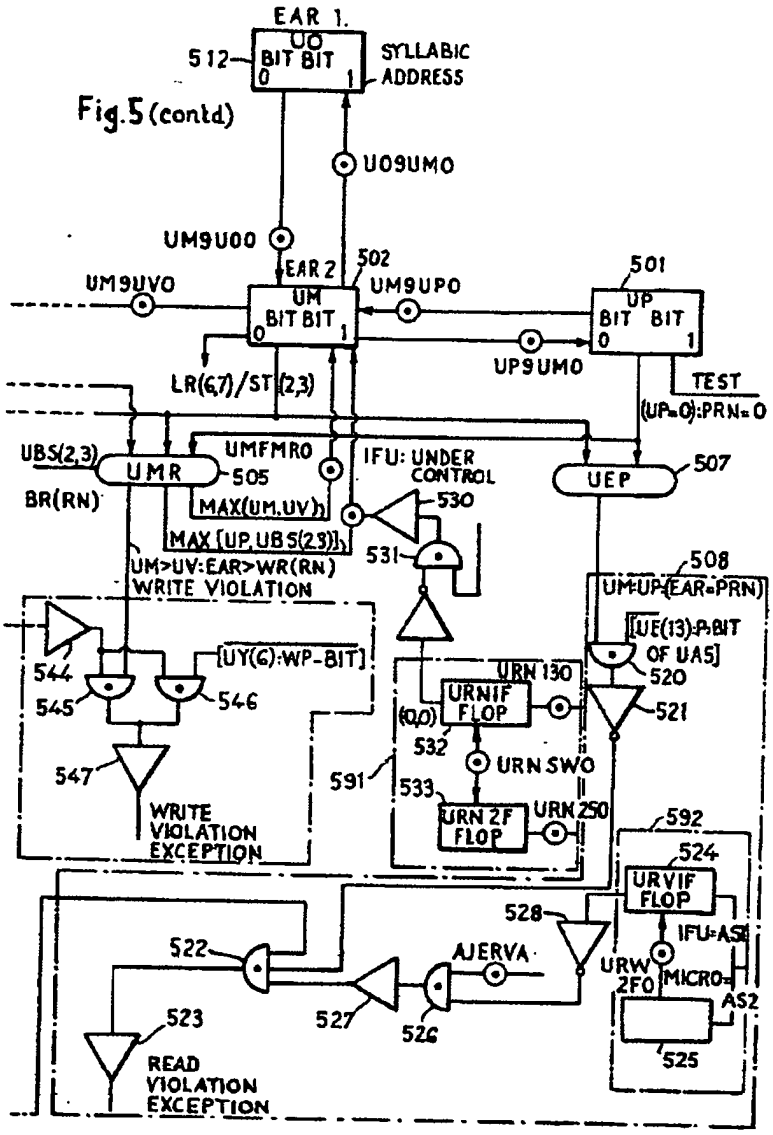

EAR 1.
UO
512
BIT BIT
0 1
SYLLABIC ADDRESS
U09UM0
UM9U00
EAR 2
502
UM9UVO
UM
BIT BIT
0 1
UM9UPO
501
UP
BIT BIT
0 1
UP9UM0
TEST
(UP=0):PRN=0
LR(6,7)/ST(2,3)
UBS(2,3)
UMR
505
UMFMRO
IFU: UNDER CONTROL
UEP
507
BR(RN)
MAX(UM.UV)
530
MAX[UP.UBS(2,3)]
531
UM>UV:EAR>WR(RN) WRITE VIOLATION
508
UM=UP=(EAR=PRN)
[UE(13)=P-BIT OF UA5]
520
[UY(6)=WP-BIT]
544
545
546
547
WRITE VIOLATION EXCEPTION
URN 130
URNIF FLOP
(0,0)
532
URN SWO
591
533
URN 2F FLOP
URN 2S0
521
592
524
URVIF FLOP
528
IFU=AS1
522
AJERVA
URW 2F0
MICRO=
AS2
523
527
526
525
READ VIOLATION EXCEPTION

Fig.5 (contd)

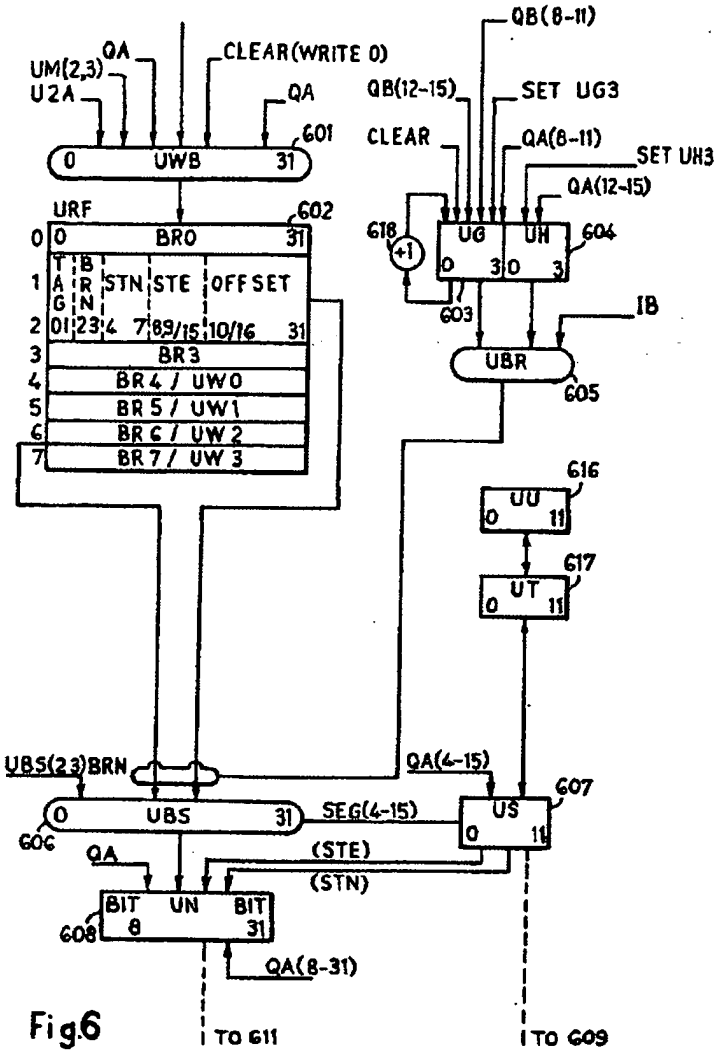

QB(8-11)
UM(2,3)
U2A
QA
CLEAR(WRITE 0)
QA
QB(12-15)
SET UG3
CLEAR
QA(8-11)
SET UH3
QA(12-15)
601
UWB
URF
602
BRO
TAG
BRN
STN
STE
OFFSET
BR3
BR4 / UW0
BR5 / UW1
BR6 / UW2
BR7 / UW3
618
+1
UG
UH
604
603
IB
UBR
605
616
UU
617
UT
UBS(2,3)BRN
QA(4-15)
606
UBS
SEG(4-15)
US
607
(STE)
(STN)
QA
BIT 8
UN
BIT 31
608
QA(8-31)
TO 611
TO 609

Fig.6

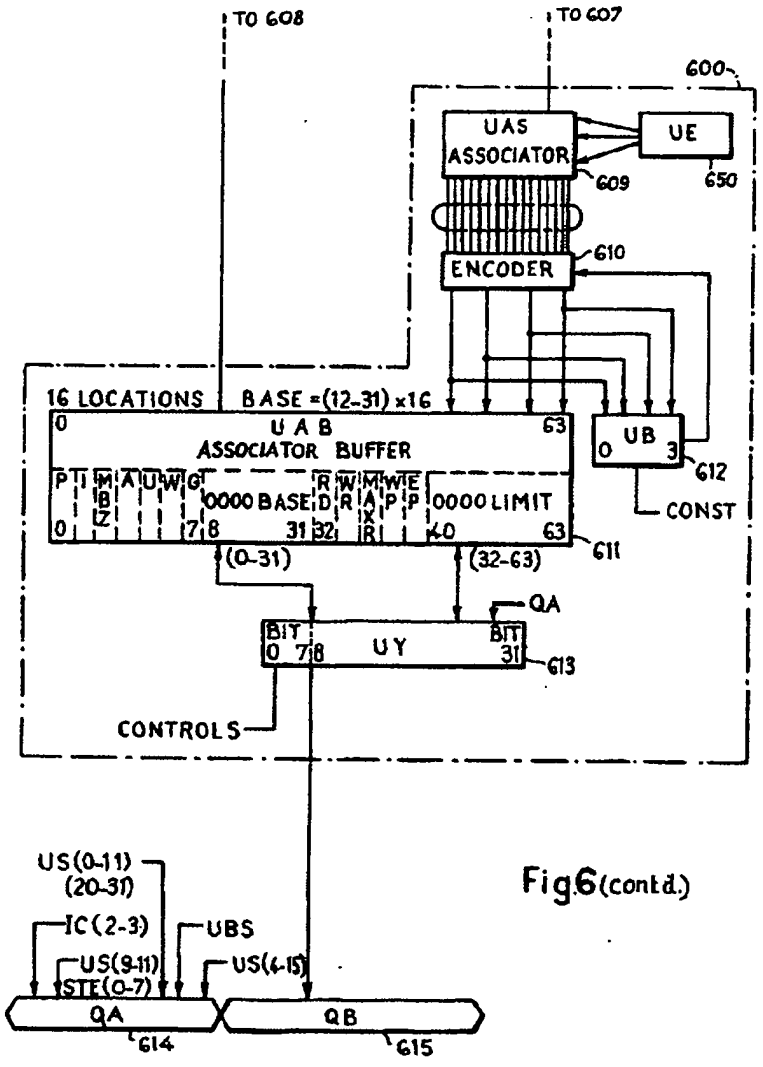

TO 608
TO 607
600
UAS
ASSOCIATOR
609
UE
650
ENCODER
610
16 LOCATIONS
BASE = (12-31) x16
0
U A B
ASSOCIATOR BUFFER
63
UB
0 3
612
CONST
P
M
B
Z
A
U
W
G
0000 BASE
R
W
D
R
M
A
X
R
W
E
P
P
0000 LIMIT
0
7 8
31 32
40
63
611
(0-31)
(32-63)
QA
BIT
0 7 8
UY
BIT
31
613
CONTROLS
US(0-11)
(20-31)
IC(2-3)
US(9-11)
STE(0-7)
UBS
US(4-15)
QA
614
QB
615

Fig.6(contd.)

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 10

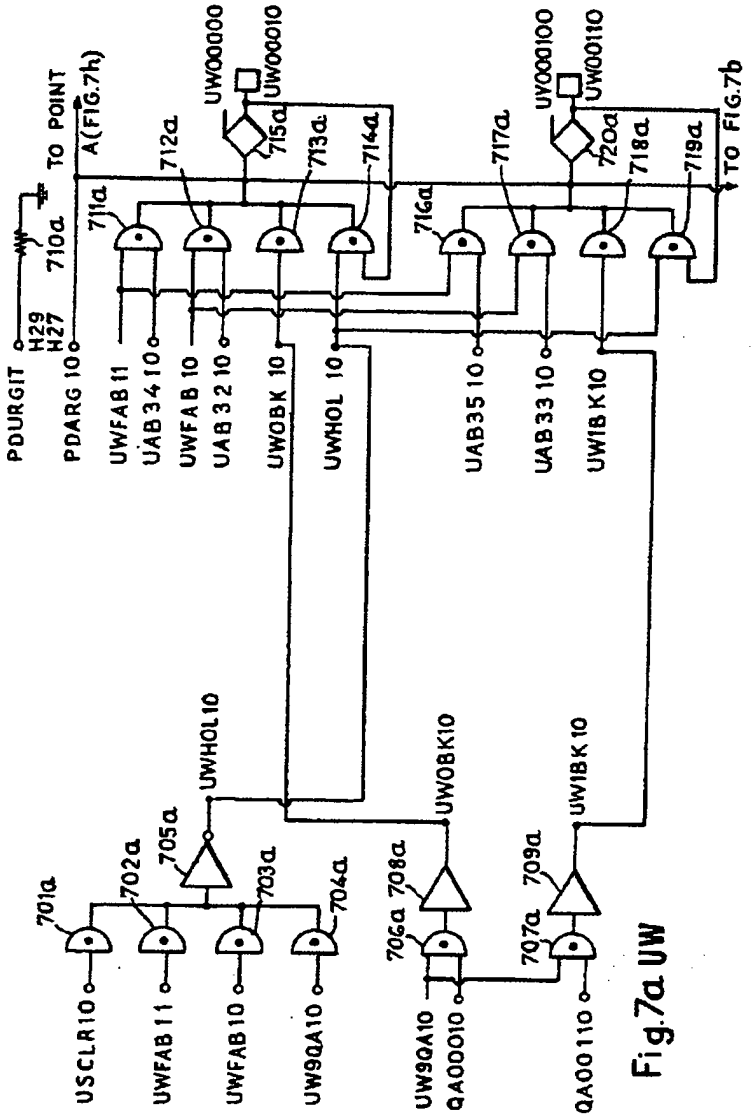
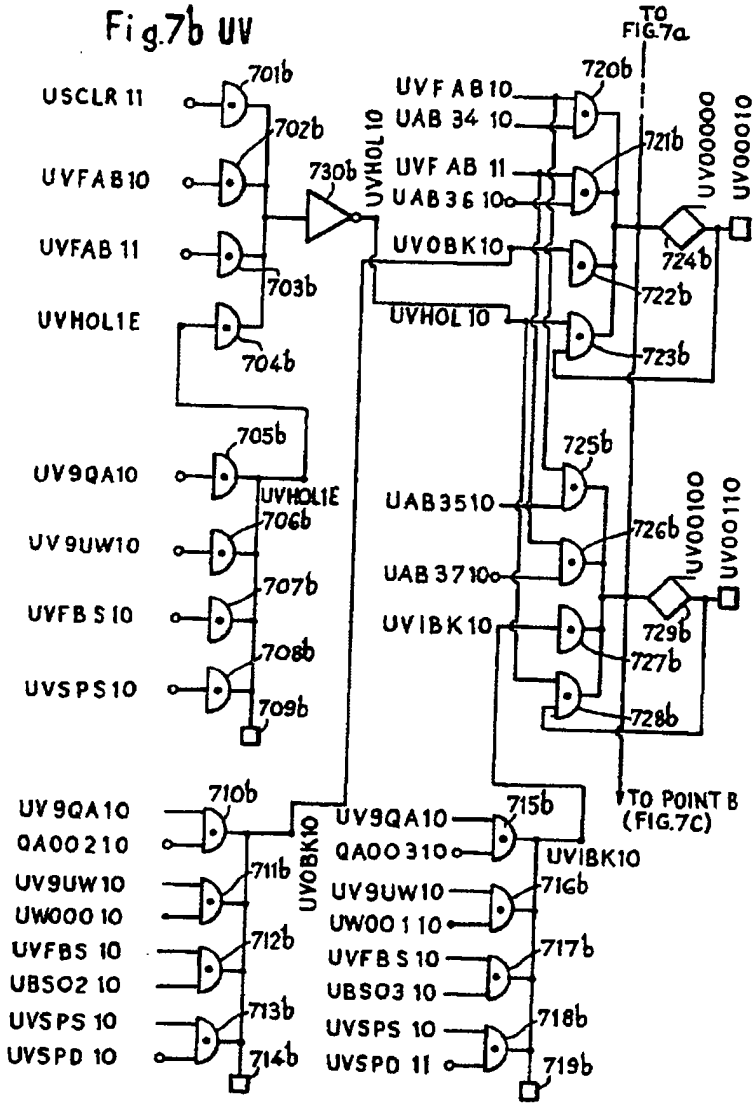


Fig.7a UW

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 11

Fig.7b UV



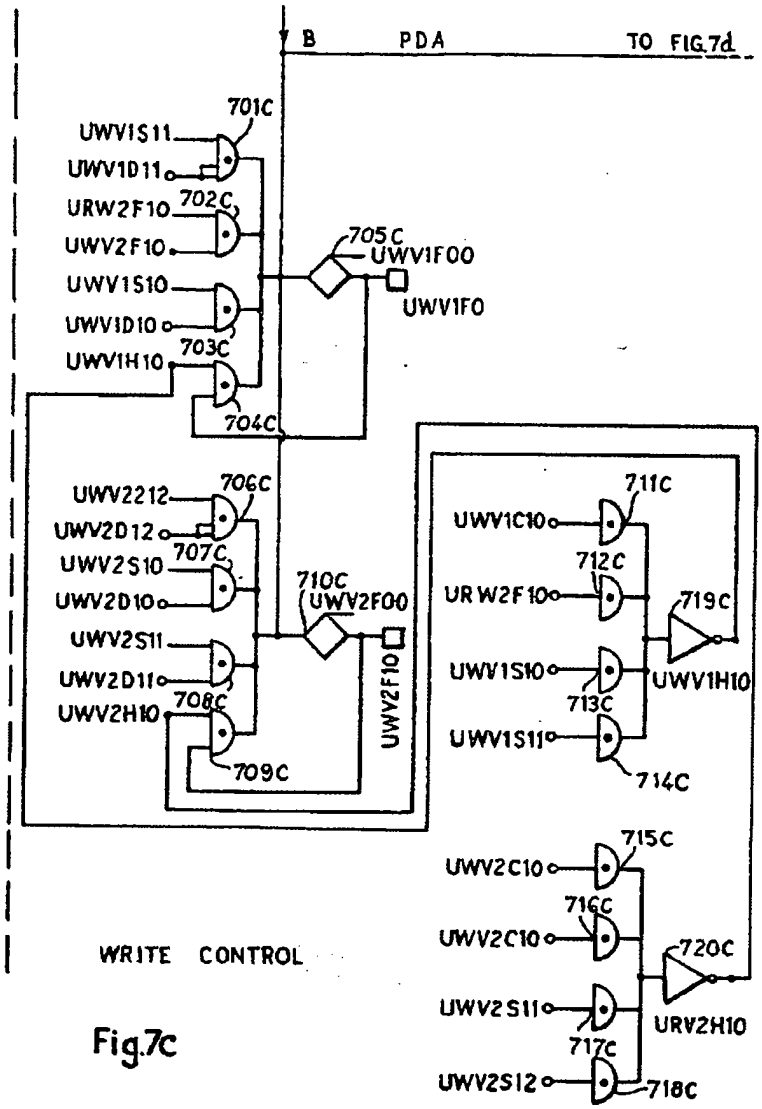

B
PDA
TO FIG.7d
701C
UWV1S11
UWV1D11
URW2F10
702C
UWV2F10
UWV1S10
UWV1D10
UWV1H10
703C
704C
705C
UWV1F00
UWV1F0
UWV2212
706C
UWV2D12
UWV2S10
707C
UWV2D10
UWV2S11
UWV2D11
UWV2H10
708C
709C
710C
UWV2F00
UWV2F10
711C
UWV1C10
712C
URW2F10
719C
UWV1S10
713C
UWV1H10
UWV1S11
714C
715C
UWV2C10
716C
UWV2C10
720C
UWV2S11
717C
URV2H10
UWV2S12
718C
WRITE CONTROL

Fig.7c

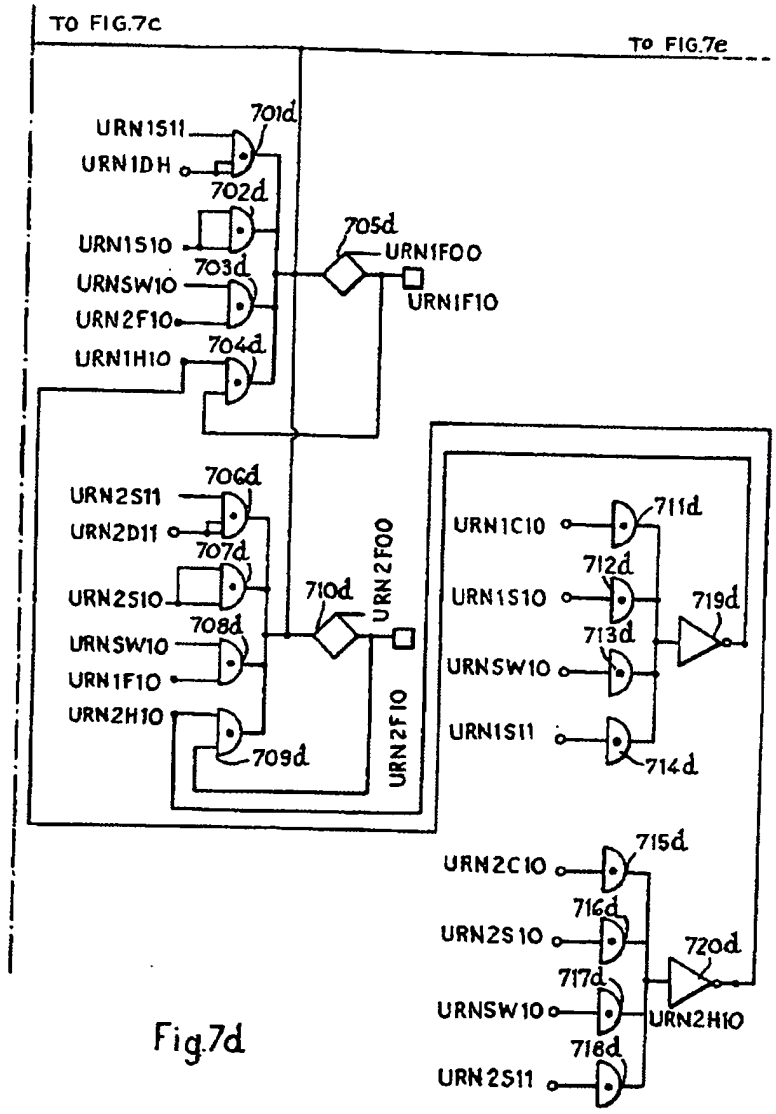

TO FIG.7c
TO FIG.7e
701d
URN1S11
URN1DH
702d
URN1S10
705d
URN1F00
703d
URNSW10
URN1F10
URN2F10
704d
URN1H10
706d
URN2S11
URN2D11
707d
URN2S10
710d
URN2F00
708d
URNSW10
URN1F10
URN2H10
709d
URN2F10
711d
URN1C10
712d
URN1S10
719d
713d
URNSW10
URN1S11
714d
715d
URN2C10
716d
URN2S10
720d
717d
URNSW10
URN2H10
718d
URN2S11

Fig.7d

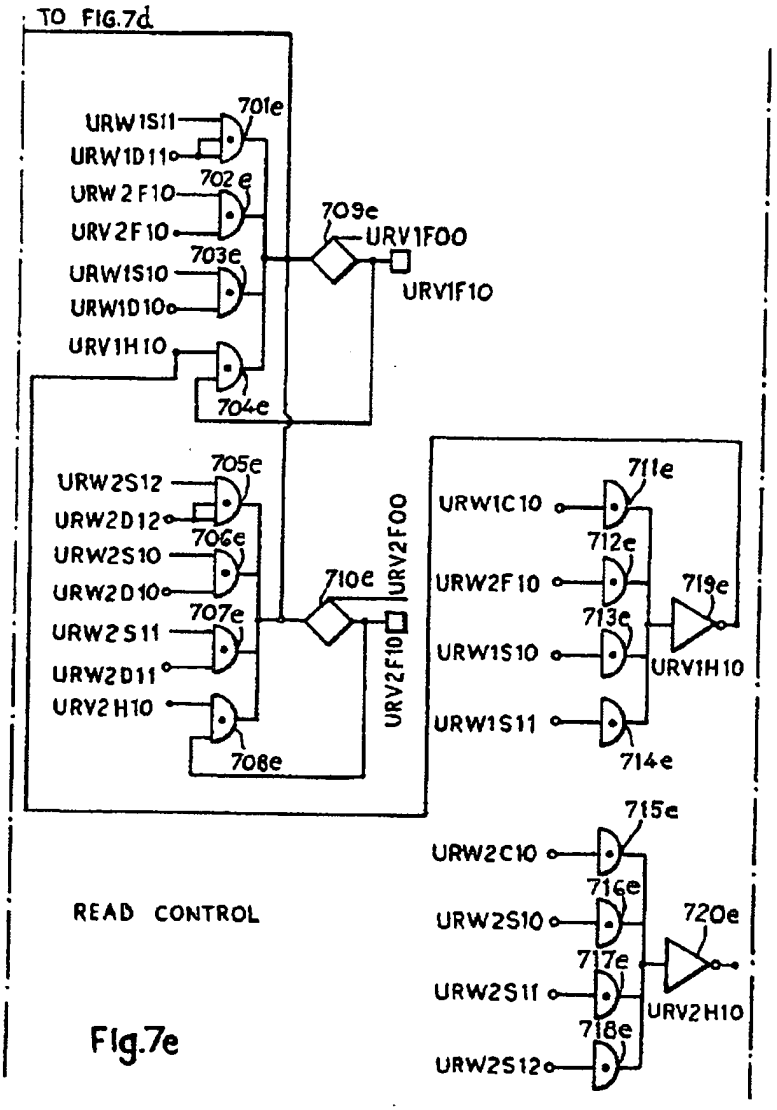
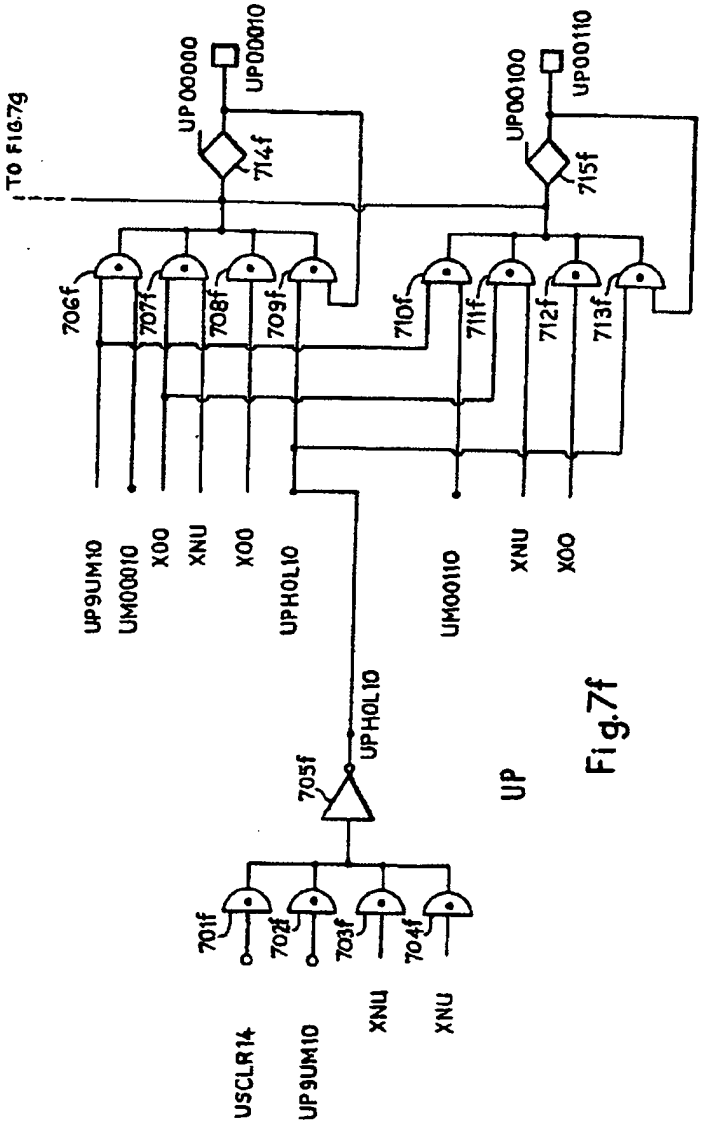


Fig.7e

1483282 COMPLETE SPECIFICATION
29 SHEETS This drawing is a reproduction of the Original on a reduced scale
Sheet 15



UP

Fig.7f

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 16

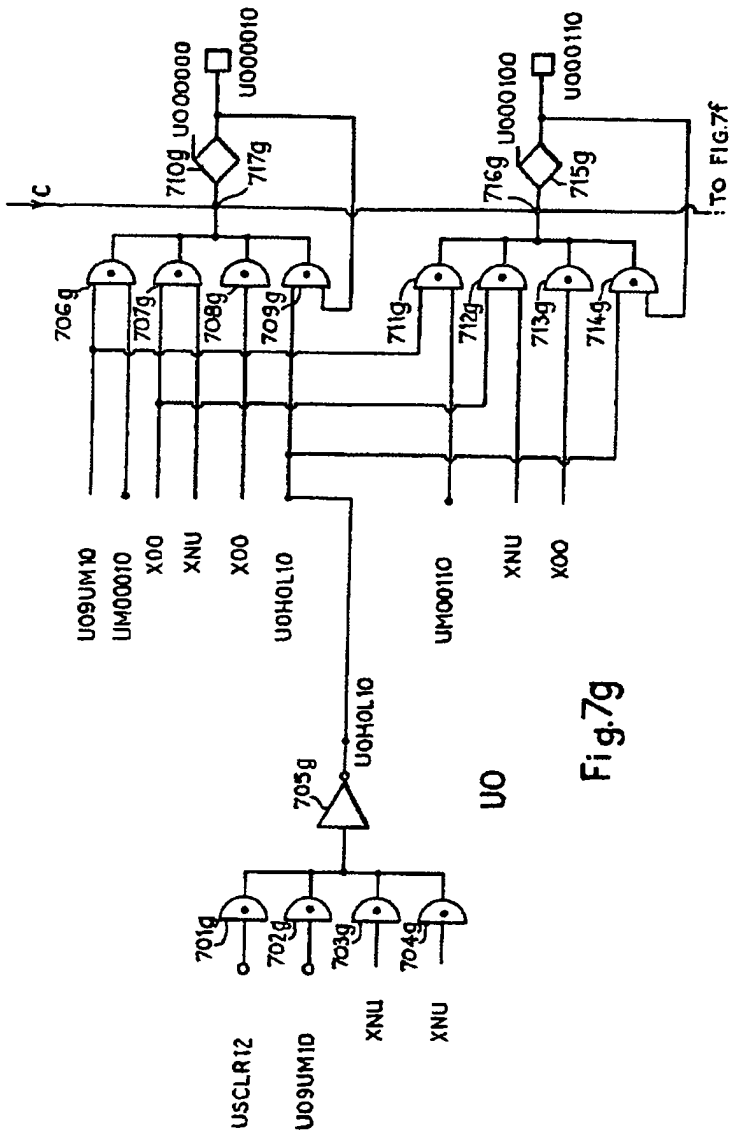


Fig.7g

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 17

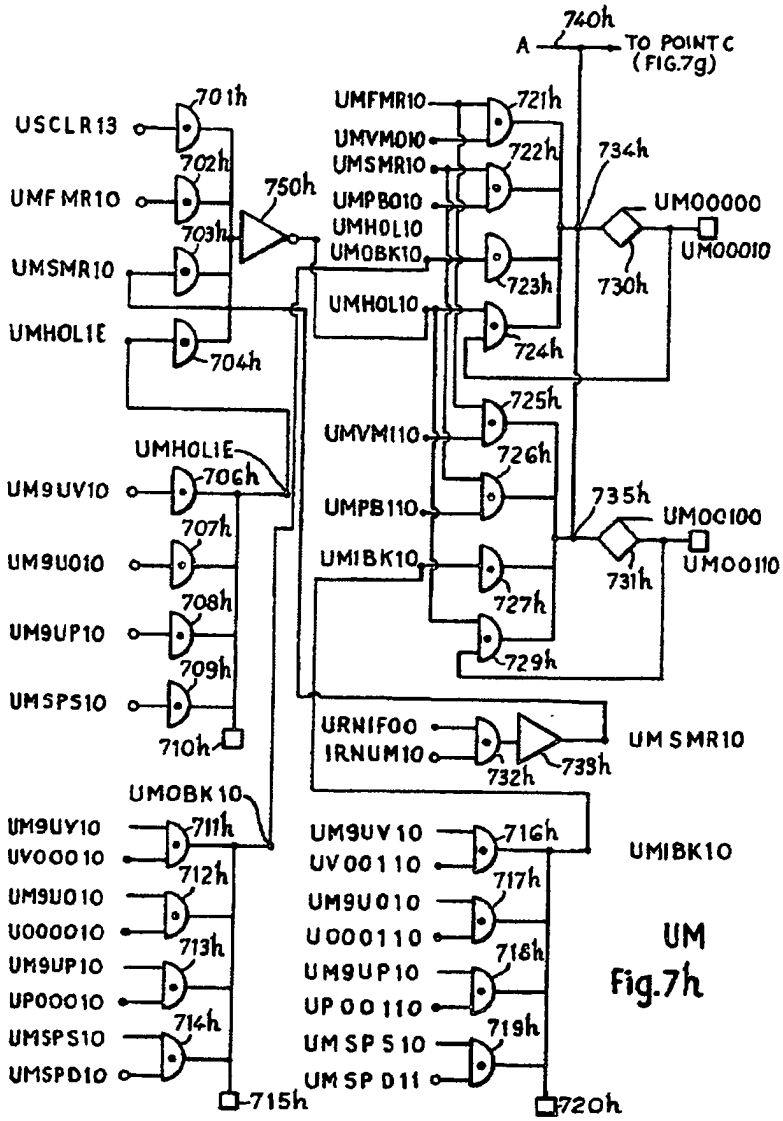


Fig.7h

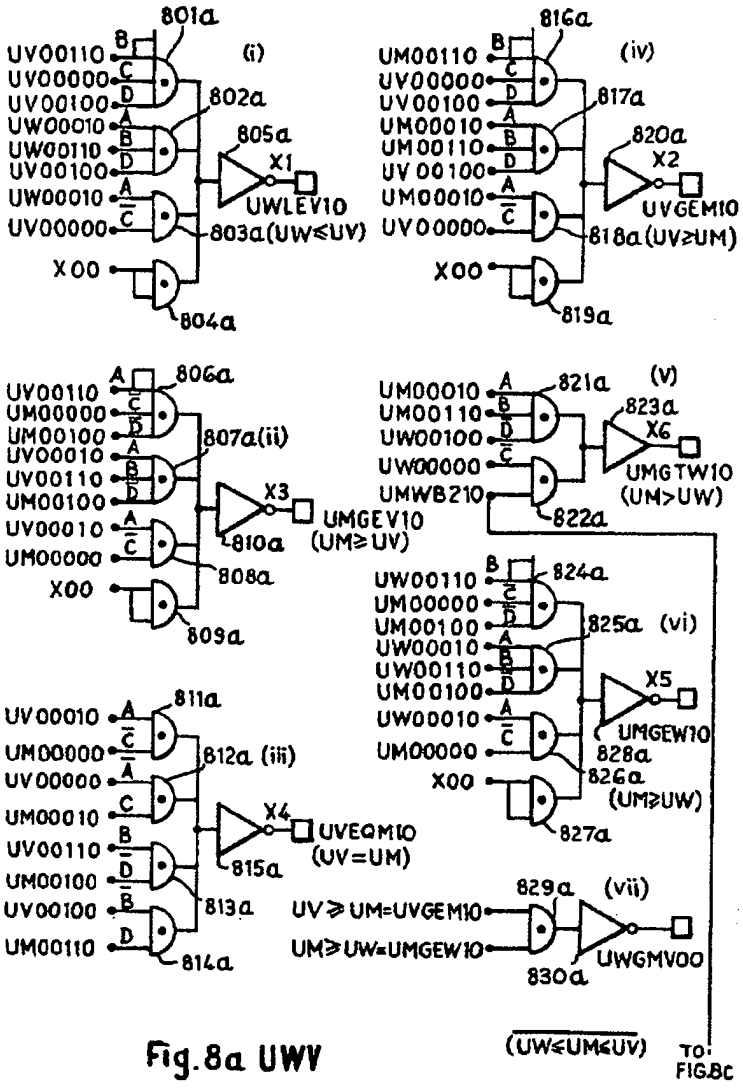

801a
UV00110
UV00000
UV00100
UW00010
UW00110
UV00100
UW00010
UV00000
X00
(i)
802a
805a
X1
UWLEV10
803a(UW≤UV)
804a
816a
UM00110
UV00000
UV00100
UM00010
UM00110
UV00100
UM00010
UV00000
X00
(iv)
817a
820a
X2
UVGEM10
818a(UV≥UM)
819a
806a
UV00110
UM00000
UM00100
UV00010
UV00110
UM00100
UV00010
UM00000
X00
807a(ii)
X3
UMGEV10
(UM≥UV)
810a
808a
809a
821a
UM00010
UM00110
UW00100
UW00000
UMWB210
(v)
823a
X6
UMGTW10
(UM>UW)
822a
811a
UV00010
UM00000
UV00000
UM00010
UV00110
UM00100
UV00100
UM00110
812a (iii)
X4
UVEQM10
(UV=UM)
815a
813a
814a
824a
UW00110
UM00000
UM00100
UW00010
UW00110
UM00100
UW00010
UM00000
X00
825a (vi)
X5
UMGEW10
828a
826a
(UM≥UW)
827a
829a
(vii)
UV≥UM=UVGEM10
UM≥UW=UMGEW10
UWGMV00
830a
(UW≤UM≤UV)
TO FIG.8c

Fig.8a UWV

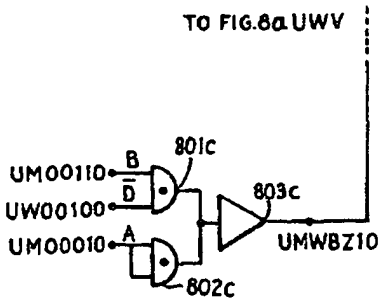


Fig.8c

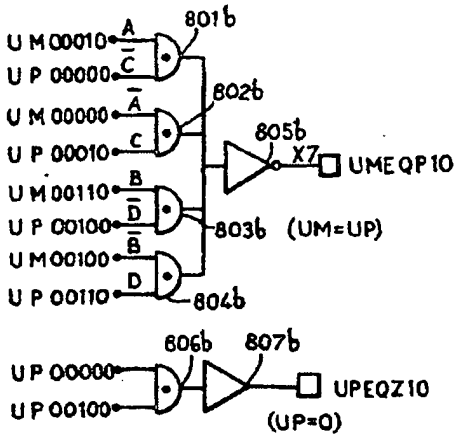


Fig.8b UEP

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 20

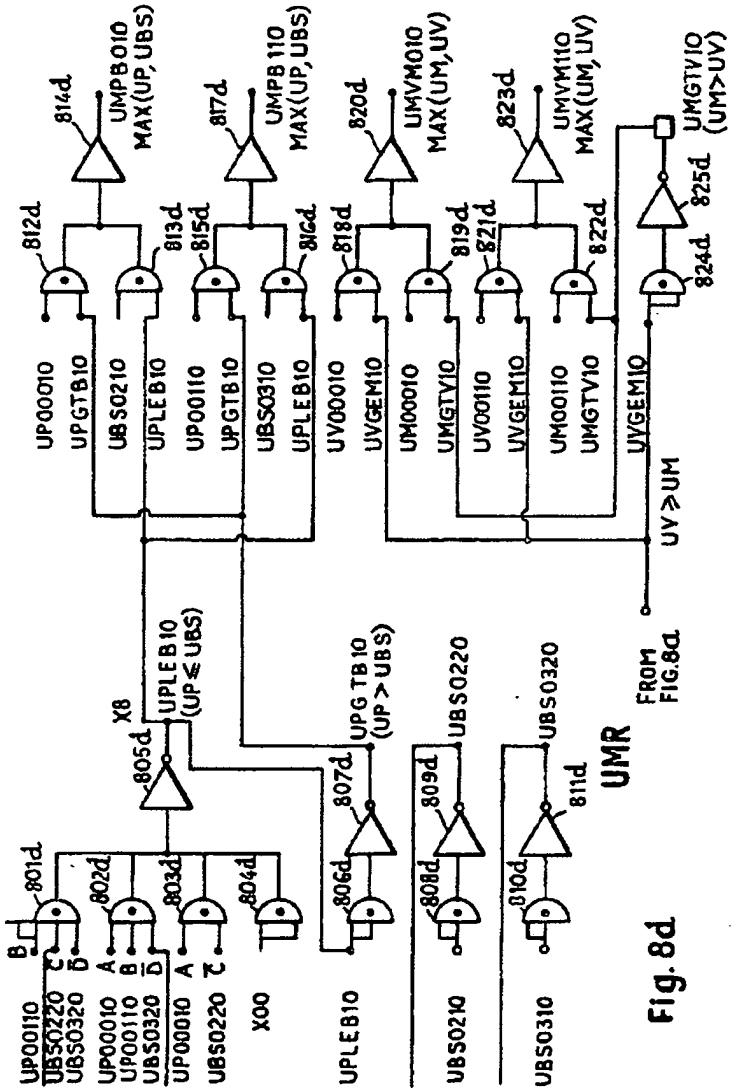


Fig. 8d

KEY TO SYMBOLS

Fig. 9a ——— INTERNAL SIGNAL SOURCE

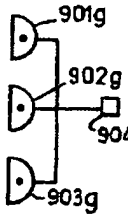
Fig. 9b ——— OUTPUT PIN

Fig. 9c ——— INPUT PIN

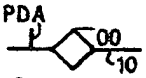
Fig. 9d  AND GATE

Fig. 9e  AMPLIFIER

Fig. 9f  INVERTER

Fig. 9g 

"AND-OR" GATES

Fig. 9h 

FLIP-FLOP

Fig. 9i ⊙——— MICRO-OPERATION

Fig. 9j X::Y——— X::Y

Fig. 9k a::β——— START OF BIT ∝ WHERE THERE ARE β BIT POSITIONS INCLUDING BIT ∝

1483282 COMPLETE SPECIFICATION
29 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 22

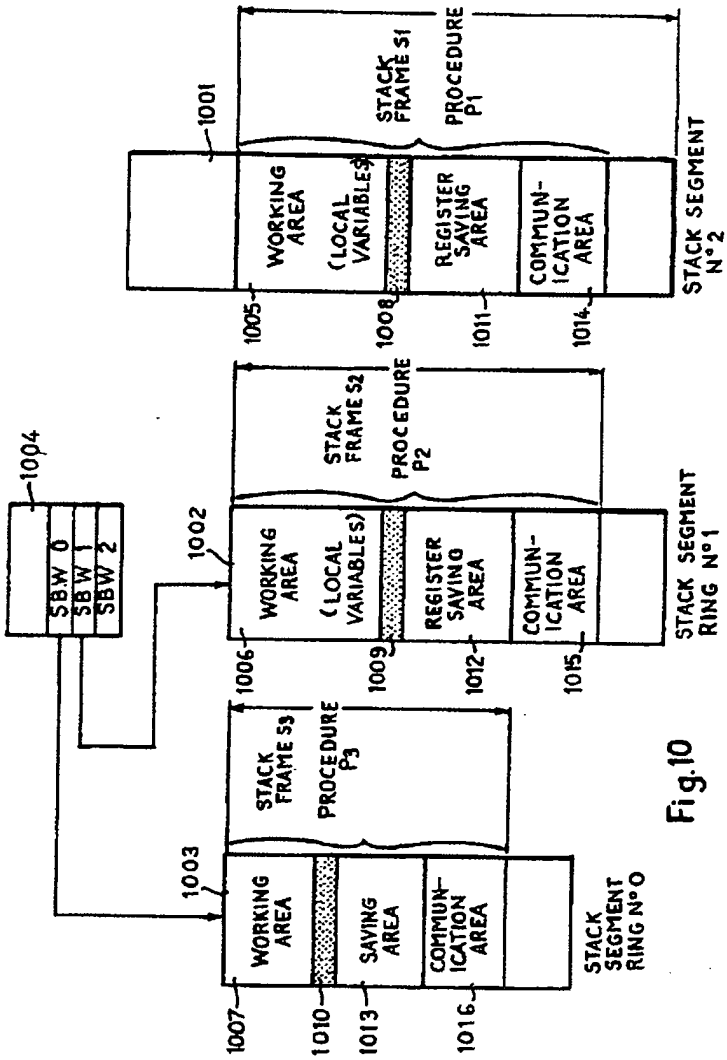


Fig.10

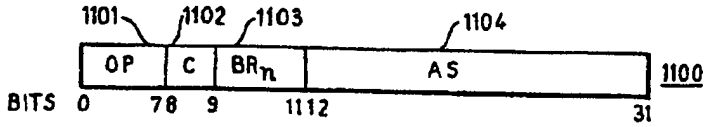


Fig. 11A

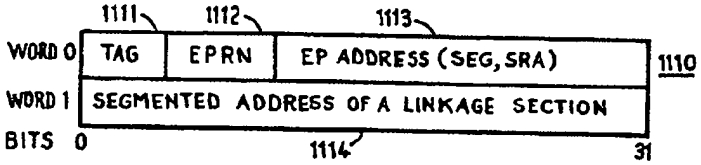


Fig. 11B

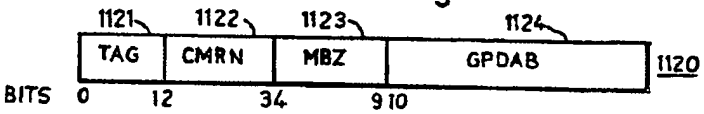


Fig. 11C

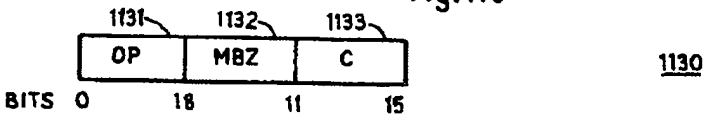


Fig. 11D

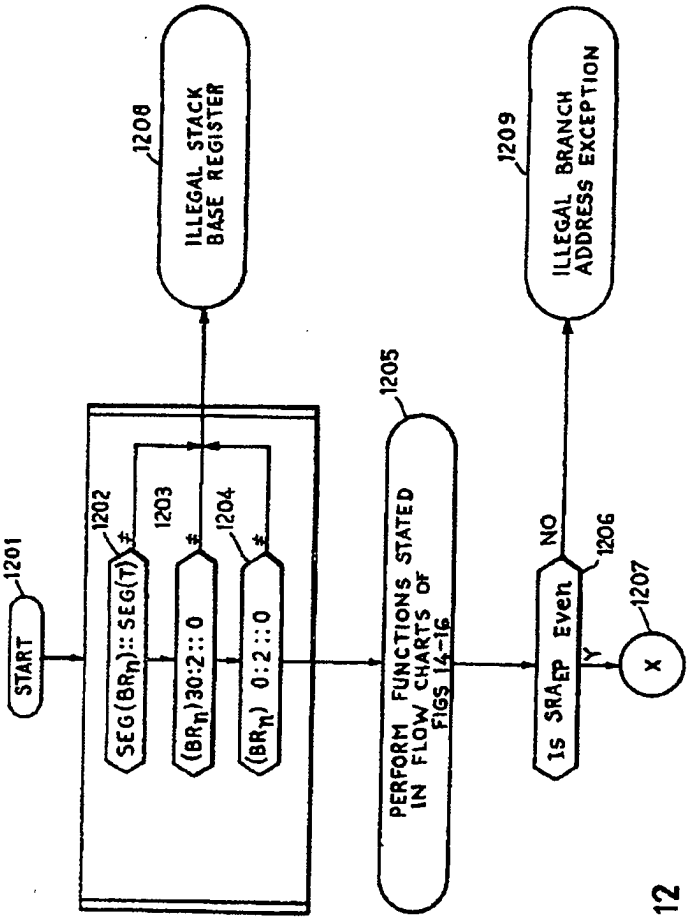

START
1201
SEG(BRn)::SEG(7)
1202
≠
(BRn)30:2::0
1203
≠
(BRn) 0:2::0
1204
≠
ILLEGAL STACK BASE REGISTER
1208
PERFORM FUNCTIONS STATED IN FLOW CHARTS OF FIGS 14-16
1205
Is SRAEP Even
1206
NO
Y
ILLEGAL BRANCH ADDRESS EXCEPTION
1209
X
1207

Fig. 12

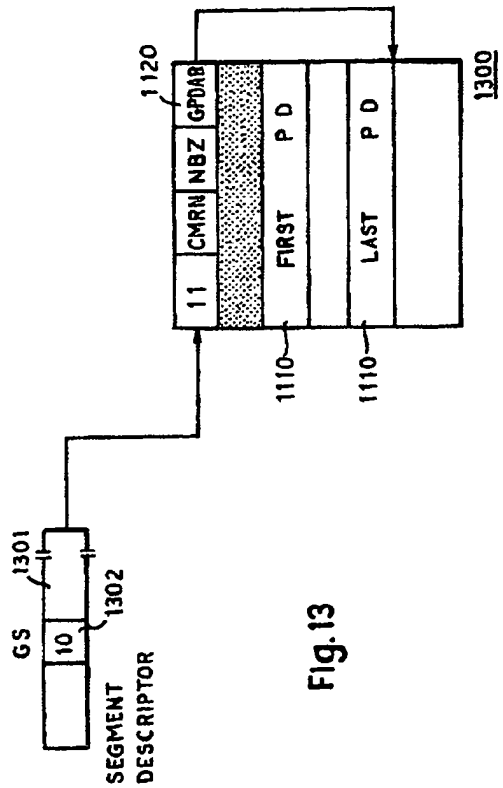

1120
GPDAB
NBZ
CMRN
11
FIRST
P D
LAST
P D
1300
1110
1110
1301
1302
GS
10
SEGMENT DESCRIPTOR

Fig.13

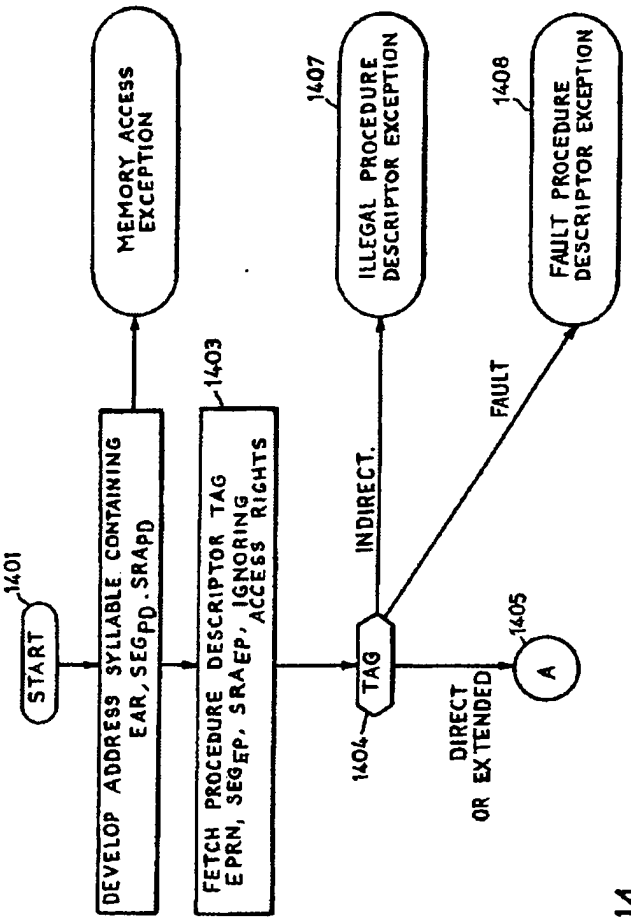

START
1401
DEVELOP ADDRESS SYLLABLE CONTAINING EAR, SEGPD.SRAPD
FETCH PROCEDURE DESCRIPTOR TAG EPRN, SEGEP, SRAEP, IGNORING ACCESS RIGHTS
1403
TAG
1404
MEMORY ACCESS EXCEPTION
INDIRECT.
ILLEGAL PROCEDURE DESCRIPTOR EXCEPTION
1407
FAULT
FAULT PROCEDURE DESCRIPTOR EXCEPTION
1408
DIRECT OR EXTENDED
A
1405

Fig.14

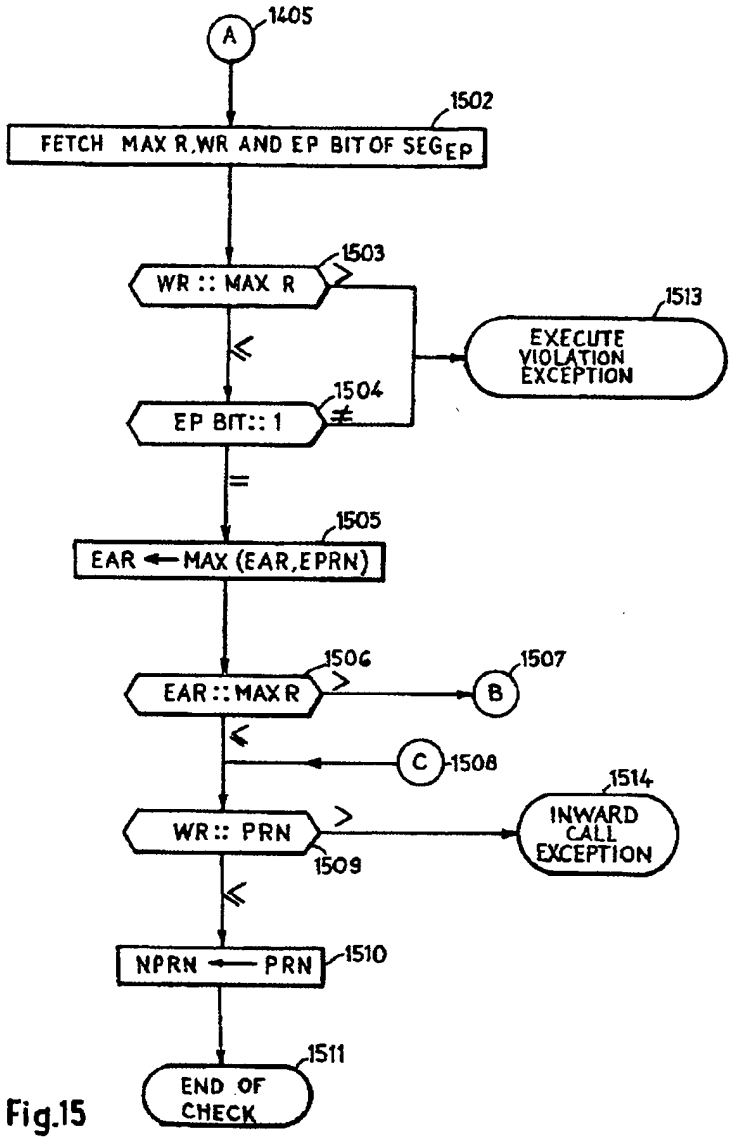

A
1405
1502
FETCH MAX R,WR AND EP BIT OF SEG EP
1503
WR :: MAX R
1513
EXECUTE VIOLATION EXCEPTION
1504
EP BIT:: 1
1505
EAR ← MAX (EAR,EPRN)
1506
EAR :: MAX R
1507
B
C
1508
1514
WR :: PRN
INWARD CALL EXCEPTION
1509
NPRN ← PRN
1510
1511
END OF CHECK

Fig.15

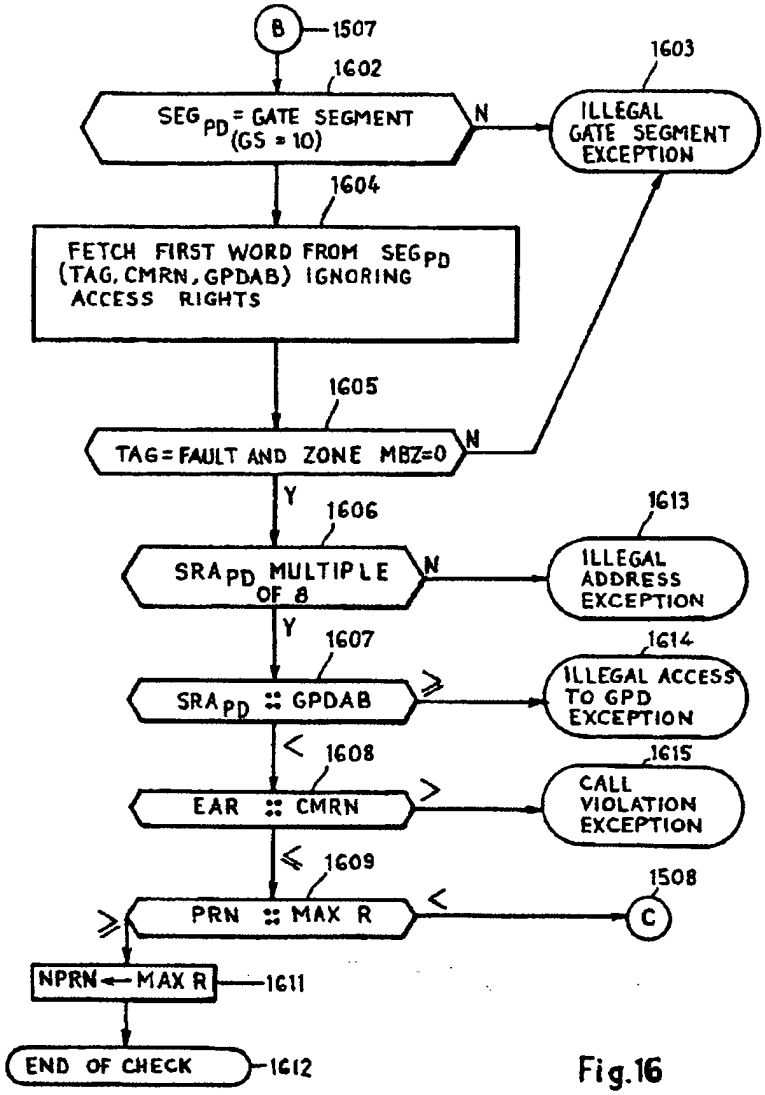

B
1507
1602
SEG PD = GATE SEGMENT (GS = 10)
N
1603
ILLEGAL GATE SEGMENT EXCEPTION
1604
FETCH FIRST WORD FROM SEG PD (TAG, CMRN, GPDAB) IGNORING ACCESS RIGHTS
1605
TAG = FAULT AND ZONE MBZ=0
N
Y
1606
SRA PD MULTIPLE OF 8
N
1613
ILLEGAL ADDRESS EXCEPTION
Y
1607
SRA PD :: GPDAB
≥
1614
ILLEGAL ACCESS TO GPD EXCEPTION
<
1608
EAR :: CMRN
>
1615
CALL VIOLATION EXCEPTION
≤
1609
PRN :: MAX R
<
1508
C
≥
NPRN ← MAX R
1611
END OF CHECK
1612

Fig.16

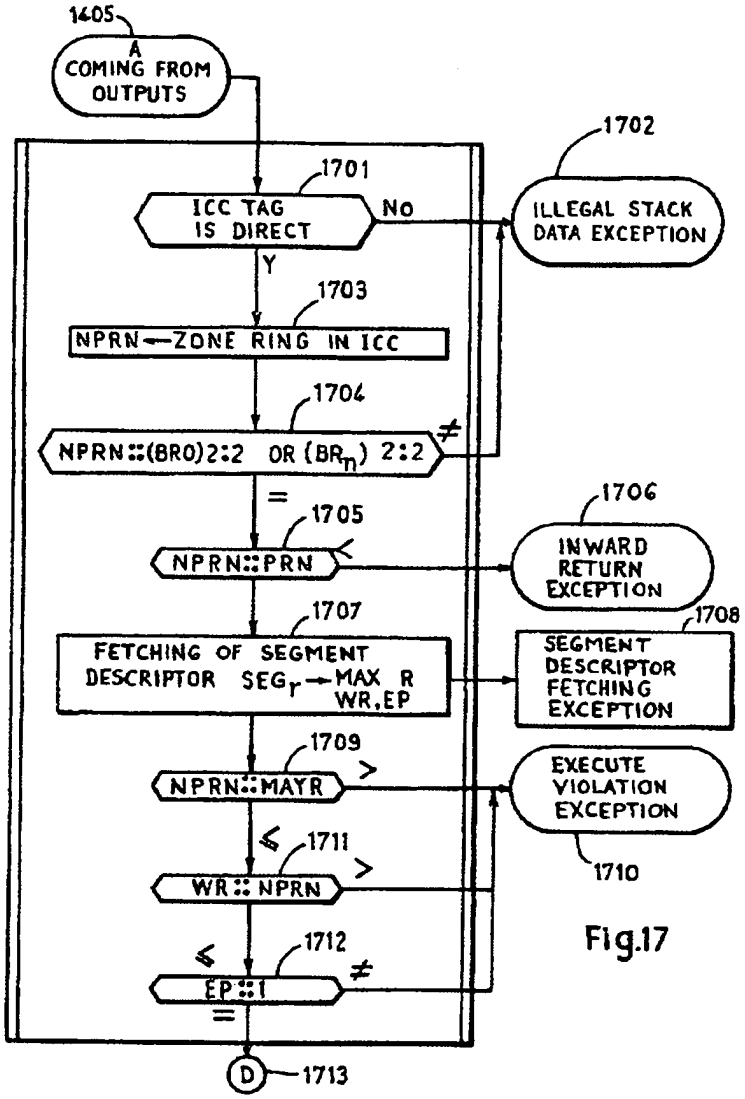

1405
A
COMING FROM
OUTPUTS
1701
ICC TAG
IS DIRECT
No
Y
1702
ILLEGAL STACK
DATA EXCEPTION
1703
NPRN←ZONE RING IN ICC
1704
NPRN::(BRO)2:2 OR (BRn) 2:2
≠
=
1705
NPRN::PRN
<
1706
INWARD
RETURN
EXCEPTION
1707
FETCHING OF SEGMENT
DESCRIPTOR SEGr←MAX R
WR,EP
1708
SEGMENT
DESCRIPTOR
FETCHING
EXCEPTION
1709
NPRN::MAYR
>
EXECUTE
VIOLATION
EXCEPTION
1710
≤
1711
WR::NPRN
>
≤
1712
EP::1
≠
=
D
1713

Fig.17

(12) UK Patent Application (19) GB (11) 2 236 604 (13) A

(43) Date of A publication 10.04.1991

(21) Application No 9009655.3

(22) Date of filing 30.04.1990

(30) Priority data
(31) 415984 (32) 02.10.1989 (33) US

(51) INT CL[5]
G06F 1/00

(52) UK CL (Edition K)
G4A AAP

(56) Documents cited
EP 0002390 A1 WO 88/02202 A1

(58) Field of search
UK CL (Edition K) G4A AAP
INT CL[5] G06F 1/00 12/14
Online database: WPI

(71) Applicant
Sun Microsystems Inc

(Incorporated in the USA - Delaware)

2550 Garcia Avenue, Mountain View, California 94043, United States of America

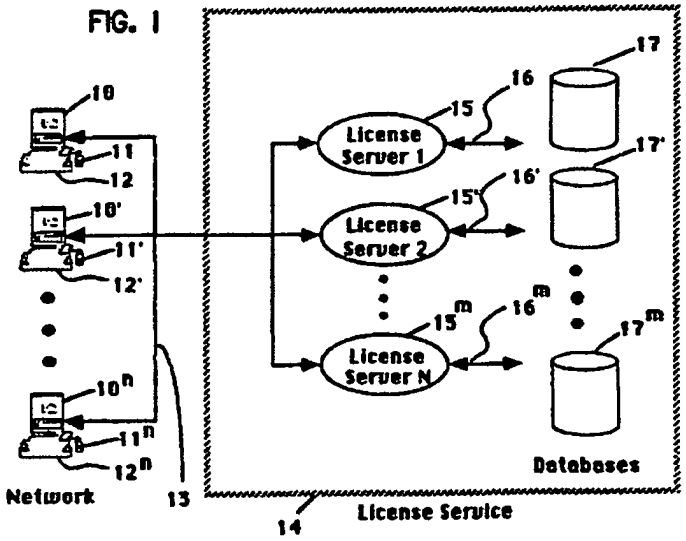
(72) Inventor
John Richard Corbin

(74) Agent and/or Address for Service
Potts Kerr and Co
15 Hamilton Square, Birkenhead, Merseyside, L41 6BR, United Kingdom

(54) **Protecting against the unauthorised use of software in a computer network**

(57) The present invention provides to a software application the verification and licence check out functions which are normally performed by a licence server. The encrypted licence information is contained in a licence token, and is stored in a database 17 controlled by the licence server 15. In contrast to the prior art where the server either grants or denies the request after verifying the user's credentials, the server in the preferred embodiment of the present invention finds the correct licence token for the software application and transmits the token to a licencing library. A licence access module attached to the application decodes the token. Routines in the licencing library coupled to the software application verify the licence information before issuing the licence and updating the token. The access module then encodes the updated token before returning it to the server. Because the verification and issuing function of a token are performed by a software application, the application rather than the server becomes the point of attack by unauthorised users. Reverse engineering the access module is less rewarding than attacking the server because the module reveals the contents of a small fraction of a database of licences.

FIG. 1



At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

GB 2 236 604 A

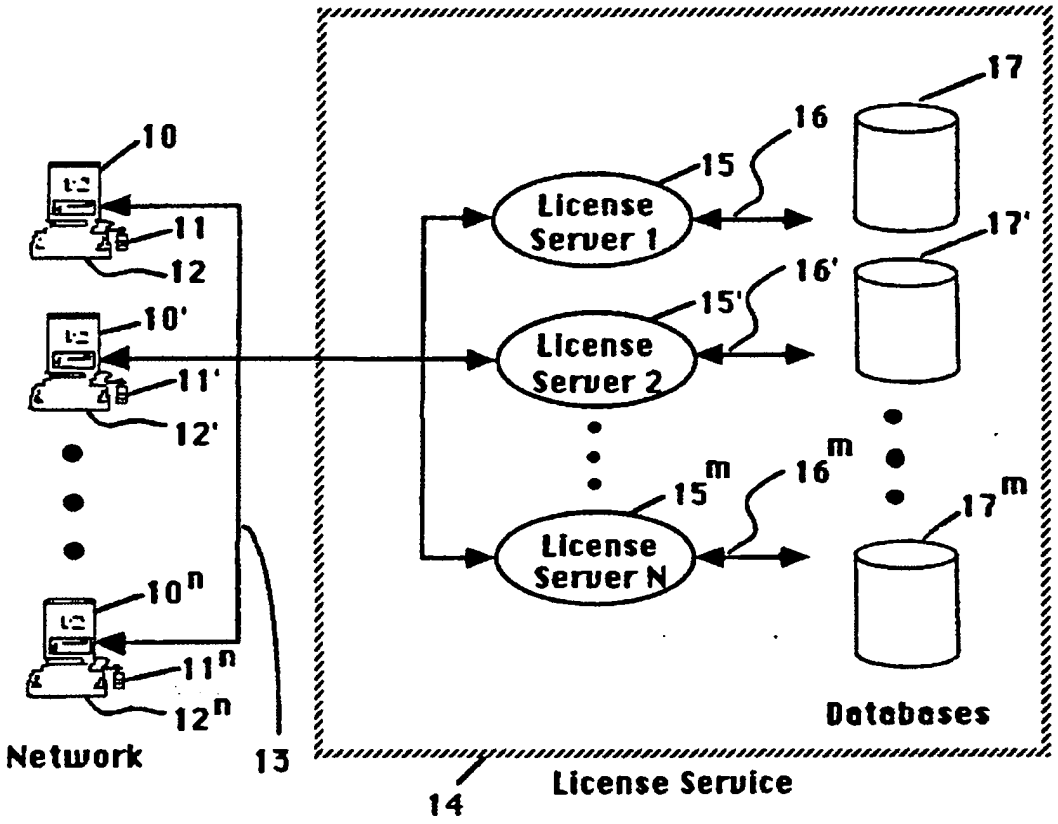

10
11
12
10'
11'
12'
10^n
11^n
12^n
Network
13
14
License Service
License Server 1
License Server 2
License Server N
15
15'
15^m
16
16'
16^m
17
17'
17^m
Databases

FIG. 1

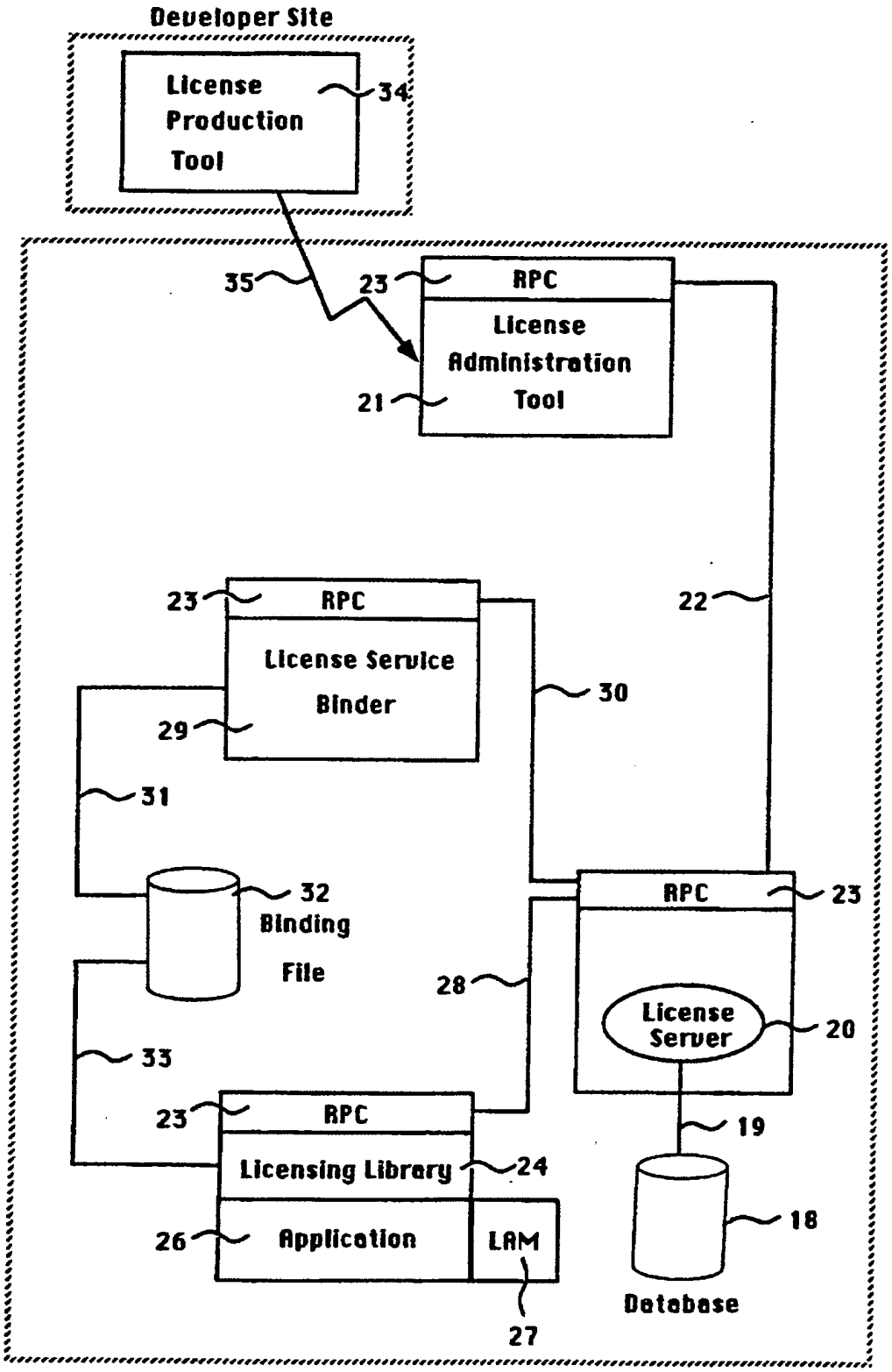

Developer Site
License Production Tool
34
35
23
RPC
License Administration Tool
21
22
23
RPC
License Service Binder
29
30
31
32
Binding File
RPC
23
28
License Server
20
33
23
RPC
Licensing Library
24
19
26
Application
LAM
27
18
Database
Customer Site

FIG. 2

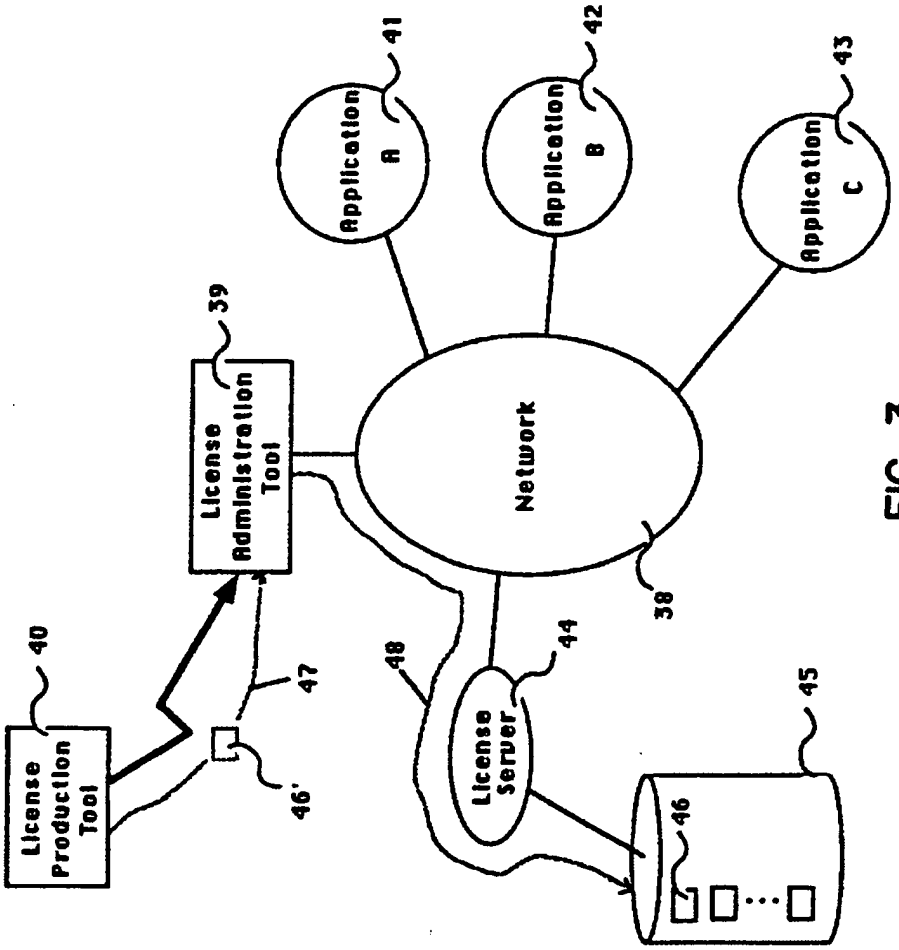

Application A
41
Application B
42
Application C
43
License Administration Tool
39
Network
38
License Production Tool
40
47
46'
48
License Server
44
45
46

FIG. 3

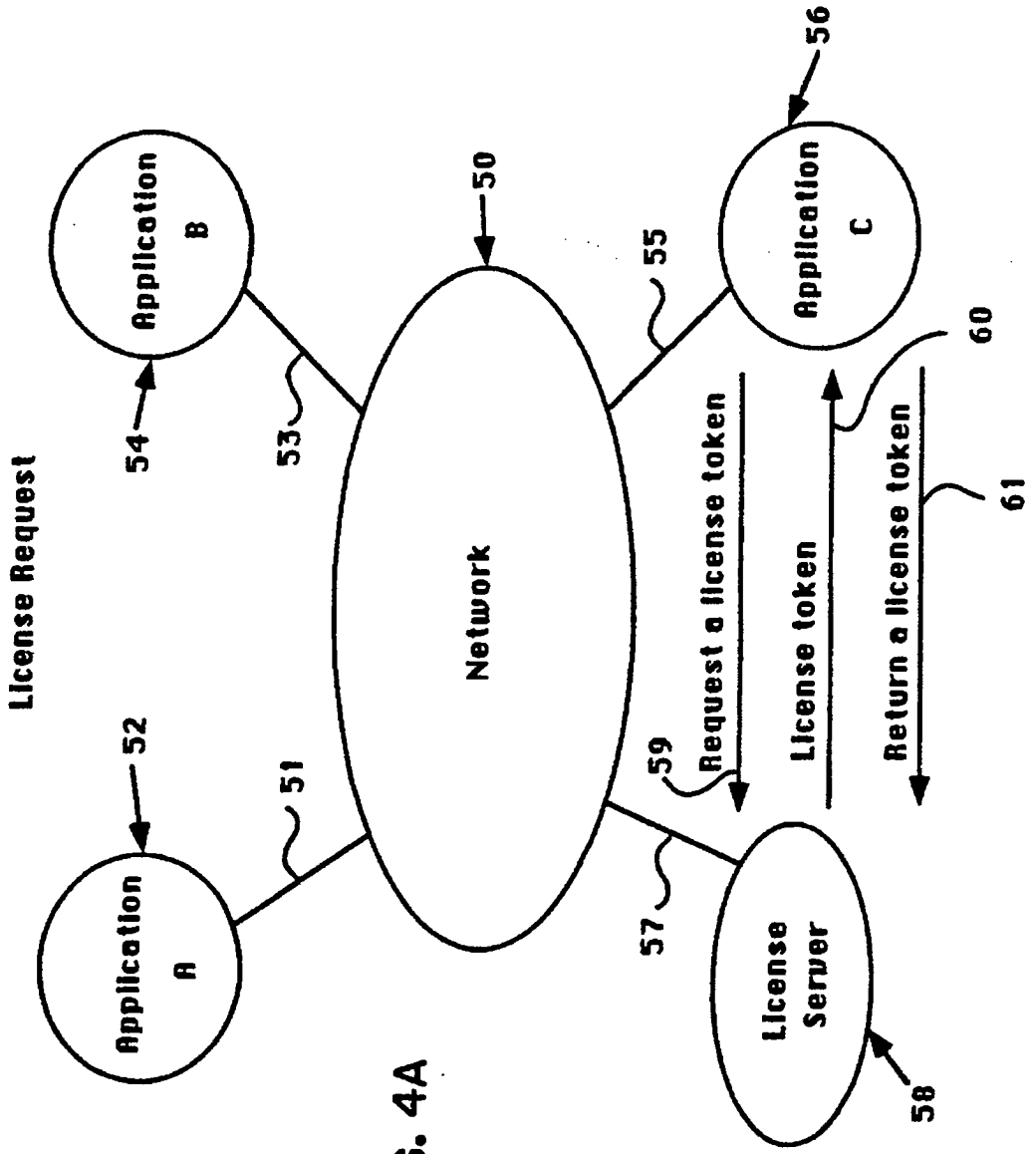

License Request
Application A
Application B
Application C
Network
License Server
50
51
52
53
54
55
56
57
58
59
60
61
Request a license token
License token
Return a license token

FIG. 4A

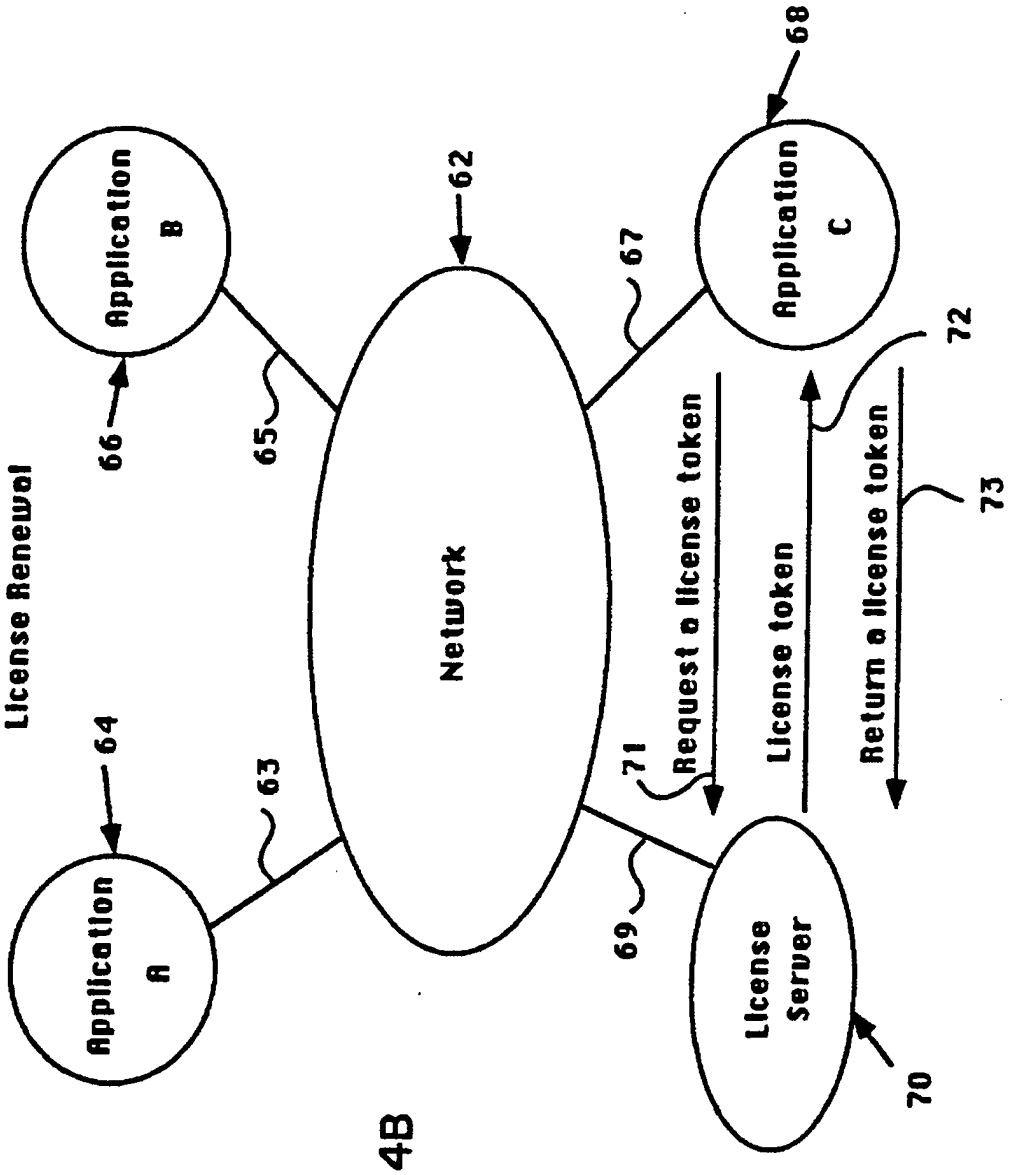

License Renewal
Application A
Application B
Application C
Network
License Server
64
63
66
65
62
67
68
69
70
71
Request a license token
License token
72
Return a license token
73

FIG. 4B

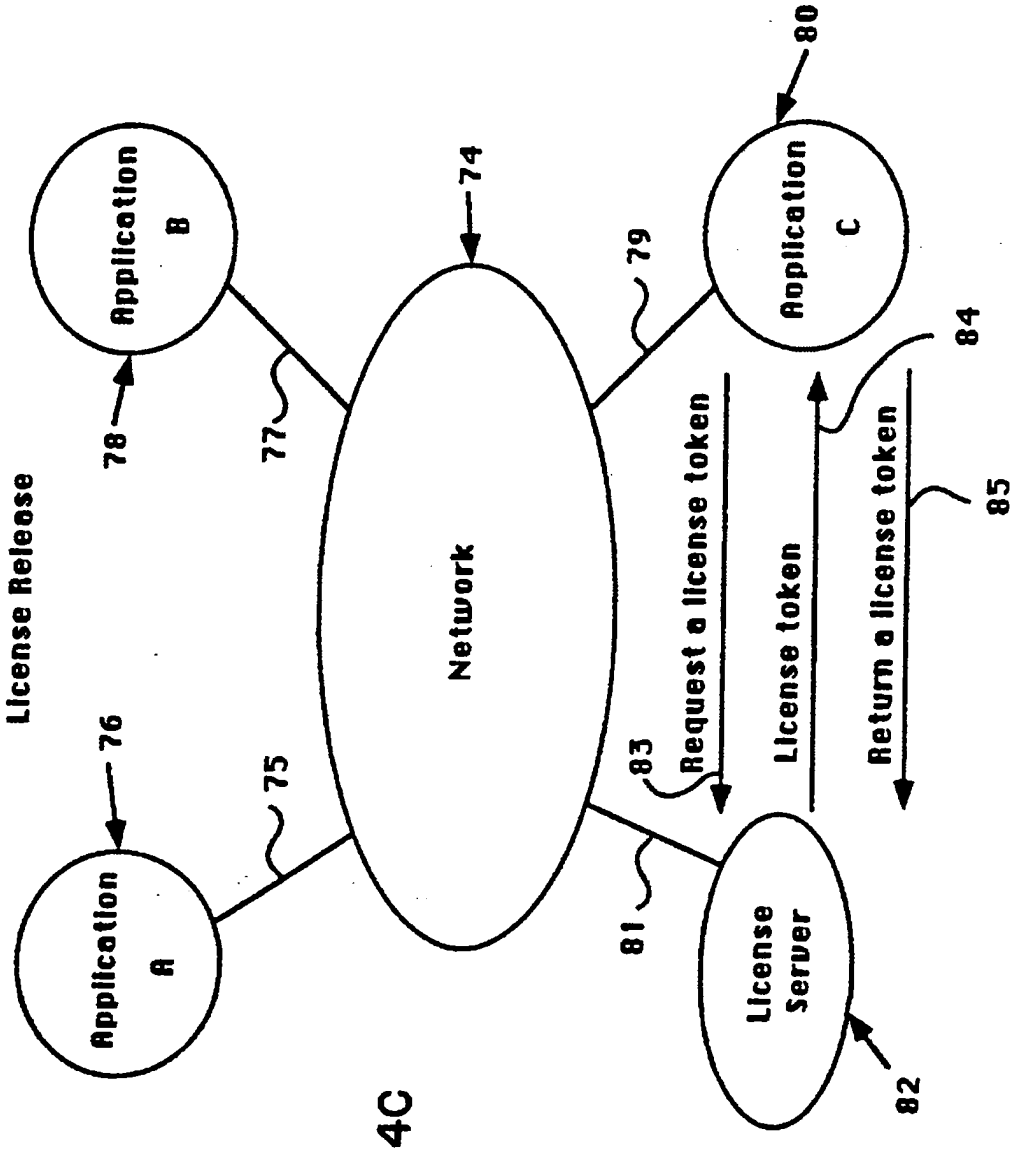

Application B
78
77
74
79
80
Application C
84
Request a license token
83
License token
Return a license token
85
License Release
76
75
Network
Application A
81
License Server
82

FIG. 4C

METHOD FOR PROTECTING AGAINST THE UNAUTHORIZED USE OF SOFTWARE IN A COMPUTER NETWORK ENVIRONMENT

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

The present invention relates to a method for protecting against the unauthorized use of a software application in a computer network environment.

2. ART BACKGROUND

A computer network is typically an interconnection of machines or agents over links or cables. The open access characteristics of a computer network presents opportunities for the unauthorized copying of software, thus eroding the licensing revenue potential of software developers. Traditionally, either the entire network must be licensed (commonly referred to as a site license), or each node where the software is run must be licensed (commonly referred to as a node license). A node refers to a single machine, agent or system in a computer network. A license is an authorization given by a software developer to a customer to use a software application in a specific manner.

A site license lets all users at a designated location or network use the software application, regardless of their position on the network. This flat-fee approach is an overkill for a low usage software application. A node license not only ties a software application to a particular machine in a network, but also is not cost effective for the infrequent use of a software application. See, for example, U.S. Patent No. 4,688,169. Furthermore, if new users of licensed nodes wish to use the software application, they are often required to purchase additional licenses.

An alternative to a site license or a node license is the concept of a concurrent usage license. A concurrent usage license restricts the number of users allowed to use a software application at any given time, regardless of their location on the network. Just as renters check out available copies of a

movie video from a video rental store, users on a network check out a software application from an agent on a first-come-first-serve basis. Thus, a concurrent usage license charges a fee for the use of a software application proportional to its actual use.

Methods to license a software application for concurrent use in a network environment are currently offered by Highland Software, Inc. and Apollo Computer, Inc. See, M. Olson and P. Levine, "Concurrent Access Licensing", *Unix Review*, September 1988, Vol. 6, No. 9. In general, the license for a software application is stored in a database controlled by a license server. A license server is a program that not only stores the license, but also verifies the user's credentials before checking out the license to the authenticated user. To protect against the authorized use, these methods to license concurrent usage rely on secured communications such as public/private key encryption. Under public/private key encryption, each user of the system has two keys, one of which is generally known to the public, and the other which is private. The private transformation using the private key is related to the public one using the public key but the private key cannot be computationally determined from the public key. See Denning, D., *Cryptography and Data Security*, Addison-Wesley, 1982. The encryption key is hidden in the license server to encrypt the database of licenses. Well designed public/private key encryption schemes are difficult to crack, especially if the license server is located in a trusted environment. A trusted environment is one whose access is limited to users having the proper credentials. However, a license server is more likely to be located at a customer's site and hence in an hostile environment. It follows that the license server is vulnerable to sophisticated intruders. Once the private key is decrypted, all sensitive information on the license server such as licenses are compromised.

It is therefore an object of the present invention to provide a more secure method to protect against the unauthorized use of software in a concurrent use licensing environment.

SUMMARY OF THE INVENTION

The present invention provides to the software application the verification and license check out functions which are normally performed by a license server. The preferred embodiment of the present invention comprises a computer network including a plurality of agents running at least one license server and at least one software application. The license server controls a database of an agent containing the license information for the software application. The license information is contained in a license token, and is stored in the database controlled by the license server. The license token is a special bit pattern or packet which is encrypted by the software vendor of the application software. The software application communicates with the license server through a licensing library. The licensing library is a collection of library routines that the software application invokes to request or renew a license from the license server. Before a software application obtains a license, the license token must be decoded by a license access module. The license access module, which is linked with the software application and the licensing library is a program that decodes the license token from a vendor specific format to a licensing library format.

When an user wishes to run a software application, the licensing library invokes a call to request a license token from the license server. In contrast to the prior art where the license server either grants or denies the request after verifying the user's credentials, the license server in the preferred embodiment of the present invention finds the correct license token for the software application and transmits the license token to the licensing library. The license access module attached to the licensing library decodes the licensing token. Routines in the licensing library coupled to the software application verify the license information before checking out the license and updating the license token. The license access module encodes the updated license token before returning it to the license server.

Because the verification and check out function of a license token are performed by a software application, the software application rather than the license server becomes the point of attack by unauthorized users. Reverse engineering the license access module is less rewarding than attacking the license server because the license access module reveals the contents of a fraction of a database of licenses. By the time most attackers crack the license access module, the software vendors would most likely introduce newer versions of the software application and new license access modules for them. Thus the present invention provides a more secure method for protecting against the unauthorized use of a software application in a computer network environment without modifying the underlying computer network.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a network environment employing the present invention.

Figure 2 describes the architecture of a network licensing scheme employing the preferred embodiment of the present invention.

Figure 3 describes the installation of a license token in the preferred embodiment of the present invention.

Figure 4a illustrates the use of a license token to request a license from a license server in the preferred embodiment of the present invention.

Figure 4b illustrates the use of a license token to renew a license from a license server in the preferred embodiment of the present invention.

Figure 4c illustrates the use of a license token to release a license from a license server in the preferred embodiment of the present invention.

NOTATION AND NOMENCLATURE

The detailed description that follows is presented largely in terms of algorithms and symbolic representations of operations on data bits and data structures within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bit patterns, values, elements, symbols, characters, data packages, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms, such as adding or comparing, that are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein that form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases there should be borne in mind the distinction between the method of operations in operating a computer and the method of computation itself. The present invention relates to method steps for operating a computer in processing electrical or other (e.g. mechanical, chemical) physical signals to generate other desired physical signals.

The present invention also relates to an apparatus for performing these operations. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The algorithms presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct a more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given below.

DETAILED DESCRIPTION OF THE INVENTION

The following detailed description is divided into several sections. The first of these sections describes a general network environment for accessing a database of licensed software programs. Subsequent sections discuss the details of a method for protecting against the unauthorized use of a software application.

I. General Network Environment

Referring to **Figure 1**, computer network environment comprises a plurality of data processing devices identified generally by numerals **10** through 10^n (illustrated as **10**, **10'** and 10^n). These data processing devices may include terminals, personal computers, workstations, minicomputer, mainframes and even supercomputers. For the purposes of this Specification, all data processing devices which are coupled to the present invention's network are collectively referred to as "agents". It should be understood that the agents may be manufactured by different vendors and may also use different operating systems such as MS-DOS, UNIX, OS/2, MAC OS and others. Particular examples of suitable agents include machines manufactured by Sun Microsystems, Inc., Mountain View, Calif. Each of the agents has an input device such as a keyboard **11**, **11'** and 11^n or a mouse **12**, **12'** and 12^n. As shown, agents **10** through 10^n (illustrated as **10**, **10'** and 10^n) are interconnected for data transfer to one another by a common cable **13**. It will be appreciated by one skilled in the art that the common cable **13** may comprise any shared media, such as coaxial cable, fiber optics, radio channel and the like. Furthermore, the network resulting from the interconnection of the cable **13** and agents **10** through 10^n (illustrated as **10**, **10'** and 10^n) may assume a variety of topologies, such as ring, star, bus, and may also include a collection of smaller networks linked by gateways or bridges.

Referring again to Figure 1 is a license service 14. The license service 14 is a resource shared by every agent connected to the network. In the preferred embodiment of the present invention, the license service 14 comprises license servers 15 through 15^m (illustrated as 15, 15' and 15^m) and databases 17 through 17^m (illustrated as 17, 17' and 17^m), where m is less than or equal to n. A license server is a program that runs on an agent with a memory storage capability. Each license server 15 (illustrated as 15, 15' and 15^m) communicates with a database 17 stored in memory on the agent over an interface 16 (illustrated as 16, 16' and 16^m). As will be described in detail below, the database 17 stores licensing information for various software applications which are purchased and authorized to run in the computer network environment. The license server is not limited to run on a specific agent, but can operate on any agent including the agent on which the user is to operate the application. Thus, any agent connected to the network may function as a license server as well as a device on which a user may operate application software. As will be described below, the license server does not perform verification of licenses of application software; rather the license server is passive and provides storing, locking, logging, and crash recovering function for the application software.

Figure 2 illustrates the architecture of a network licensing scheme of the present invention. The architecture comprises a database 18, database interface 19, license server 20, licensing library 24, License access module 27, license administration tool 21, license service binder 29, and license production tool 34.

The database 18 stores licensing information and application usage data. Preferably the database 18 comprises a plurality of records which contain the following information:

Database Element	Description
Unique Key Table	Keys for all other tables
Vendor Table	Vendor's ID and name
Product Table	Product number and name
Version Table	Version number and date
License Table	License #, exp date, total units
License Token Table	Stores encoded license token
Unit Group Table	A group's allocation of license
Group List Table	Name of the group
Allowed Users Table	Credentials of allowed users
Current License Use Table	Applications using a license
Lock Table	Locked records in database
Authorized administrator Table	Login names of administrators
License Operation Log Table	Administrator's log information
License Usage Log Table	Request handle plus Client Log
License Queue Log Table	License wait queue
Application Message Log Table	Application specific messages

A database interface 19 provides communication between the license server 20 and the database 18 in order to prevent concurrent access to the same database record by multiple users which can cause the data in the record to become corrupted. Thus, only the owner of the lock can read from and write to the locked record during the usage of the application.

The license server 20 operates on an agent and interfaces the database 18 to license administration tool 21, licensing library 24 and license service binder 29. The license server 20 communicates with the license administration tool 21, licensing library 24 and license service binder 29 via an interface 23. Preferably the interface 23 is a remote procedure call

mechanism which permits a process operating on one device or agent connected to the network to request a resource or service from a remote device or agent connected to the network. See A. Birrell and B. Nelson, "Implementing Remote Procedure Calls," *ACM Transaction on Computer Systems*, February 1984, Vol. 2, No. 1.

Multiple license servers may reside on multiple agents. Preferably the license server 20 operates in a background mode of the agent such that its operation is transparent to a user of that agent. More particularly, as will be described below, the license server 20 provides the following functions: 1) servicing the requests from the licensing library 24 for license token; (2) maintaining a wait queue for requests to the database 18 when no licensing units are available; (3) generating `locks` for exclusive access to database 18; and (4) providing access to information in the database 18.

The licensing library 24 is a set of library routines which enable the application 26 to request licensing service from the license server 20. Upon receiving the request for service from the licensing library 24, the license server 20 retrieves a license token from the database 18 and transmits it to the licensing library 24. The licensing library 24 is linked with the application 26 and communicates with the license server 20 over a path 28 with, preferably, a remote procedure call mechanism 23. Among the major library calls in the licensing library 24 is the application's request for a license from the license server 20. Other important library calls include the request to renew and to release a license. The use of the license token to accomplish the request for the various licensing service will be described in detail below.

The license access module (LAM) 27 is prepared by the software vendor 24 to decode the license token. Once decoded, the application 26 via routines in the licensing library verifies the licensing information in the license token and determines whether a license may be checked out. The LAM 27

also encodes the license token before the application returns it to the database 18 via license server 20. The license access module 27 is described in further detail below.

The license administration tool 21 is utilized by the network administrator to perform administrative functions relevant to the concurrent usage of a software application. The license administration tool 21 may run on any agent connected to the computer network. The license administration tool 21 is primarily used to install the license token into the database 18 through the license server 20. The functionality of the license administration tool 21 includes: (1) starting or terminating a license server, (2) accessing a database controlled by a license server; and (3) generating and printing reports on license usage.

The application 26 may not access the database 18 directly; rather, the request for a license is made through the licensing library 24 to the license server 20 over a path 28. Most network licensing schemes employ secured communication between the licensing library 24 and the license server 20. In contrast, the present invention uses the license access module (LAM) 27 the license library 24 and a plurality of license tokens to protect against the unauthorized use of software application in a computer network.

Referring once again to Figure 2, a license service binder 29 is shown coupled to the license server 20 over a path 30. The license service binder 29 is invoked by means known in the art, such as a network service program. The license service binder 29 locates all agents that are designated as servers on the network, and keeps track of which server is servicing which application. The license service binder 29 contacts each server on its table of available servers and requests a list of products it serves. Finally the license service binder 29 writes the contents of the table of available license servers and the list of products into a binding file 32 over a path 31. In Figure 2, the binding file 32 is coupled to the licensing library 24 over a path 33. The application 26

queries the binding file 32 to see which license server can service its request for a license.

A license production tool 34 is used by the software vendor to create a license token for transmittal to the network administrator. Receiving the license token, the network administrator installs it with the license administration tool 21 into the database 18 through license server 20.

II. License Token

Referring to Figure 3, the creation of a license token in a computer network employing the preferred embodiment of the present invention will be described. A computer network 38 is shown coupled with a license administration tool 39 and a single license server 44. The license server 44 communicates with a database 45. Applications 41, 42, and 43 are shown requesting licensing service from the license server 44. When a customer purchases a license for an application, such as a CAD/CAM program for its research and development department, the software vendor creates a license token with a license production tool, and delivers the license token to the customer's network administrator. A license token is a special bit pattern or packet representing a license to use a software application. The network administrator installs the license token 46 into the database of the license server using the license administration tool 39. Unlike the token used in a token ring which is passed from agent to agent, a license token in the preferred embodiment of the present invention is passed only between a license server and a licensing library for a predetermined amount of time. The predetermined amount of time corresponds to the time the license token is checked out of the license server. Currently, the license token is checked out to an application for no more than ten seconds, and the license token is returned as quickly as possible to the issuing license server. The license token 46 contains information encrypted in the vendor's format such as ,vendor identification, product and version numbers as well as the number of license units purchased

for the license token. A license unit corresponds to the license weighting for an agent connected to the computer network. For example, powerful workstations could require more license units to use a software application than an average personal computer.

The software vendor produces a license token using a license production tool 40. A path 47 illustrates how a license token 46' makes its way to a license administration tool 39 at the customer's site. There, the system administrator installs the license token 46' as license token 46 into the license database 45 of the license server 44. A path 48 indicates the transfer of the license token 46' from the license administration tool 39 to the license server 44 and into the database 45 as license token 46. The license server 44 is now ready to entertain requests from applications 41, 42, and 43 for a license to use the application corresponding to token 46 as well as other applications represented in its database 45.

It should be understood that each network may have a plurality of license servers and each license server may have in its database a plurality of license tokens for a variety of software applications. Referring again to **Figure 3**, if application A 41 requests and checks out the license token 46 for less than ten seconds, applications B and C 42, 43 would be unable to check out the license token 46 if their requests were made during the same time application 41 is checking out a license from the license token 46 because of the locking mechanism provided by database interface 19. Thus, to achieve concurrent license usage in network 38, it is preferred that the network administrator installs more than one license server. To minimize the task of recovering from license server crashes, it is also preferred that the system administrator spreads the license units for any one application among a plurality of strategically located license servers. For instance, if a network has four license servers, the network administrator may want to allocate the twenty license units for a particular popular application among four license tokens with

five license units for each license token. In the event one license server crashes or the license token is checked out, the other three license servers may provide licensing service to other applications.

Figure 4a illustrates the use of a license token to request for a license. As shown, a network 50 is provided, and is coupled to Applications 52, 54 and 56 respectively. Application 56 succeeded in requesting a license token from the license server 58 in step 59. The license token is transmitted to application 56 in step 60. When done, Application 56 returns the license token to the license server 58 in step 61. Aside from the license request function performed with the license token as shown in Figure 4a, the license token is also used in other critical stages of the licensing process. For example, an user may wish to run an application beyond the initial allocated time. As shown in Figure 4b, Application 68 makes a license renewal request 71 from the license server 70 with license token 72. Similarly, in Figure 4c the user makes a license release request 83 when the application no longer needs the license units. As such, the user updates the license token 84 by returning the updated license token to the license server 82 in step 85.

III. License Access Module

In Figure 2, a license access module (LAM) 27 is linked with the application 26 and the licensing library 24 to form the executable code that software vendors ship to the customers. The license access module 27 decodes and encodes the encrypted license token as it is being passed between the license server and the licensing library 24. Thus the level of security of an application from unauthorized use depends heavily upon how secure the license access module is.

Conventional network licensing schemes use public/private key encryption to encode sensitive information. Such a scheme is effective if the license server is in a trusted environment. However, the customer has the

same access to any agent in a network, including the license server. The security of the licensing scheme can be compromised by a user who decrypts the license server's private key. Once the unauthorized user determines the server's private key, he can decrypt all sensitive information on the license server. Should all license servers use the same key, as is frequently done, then all the security of the applications served by all the license servers will be compromised.

The license access module 27 first translates a license token from a vendor specific format to a format usable by the licensing library 24. The license access module accomplishes the translation in two modules. One module translates or decodes a license token from a vendor specific format to a licensing library format. The second module translates or encodes the updated license token from the licensing library format to the vendor specific format. The second module is invoked anytime the licensing library updates the information in a license token.

Upon receiving the license token in the licensing library format, the licensing library invokes routines which verify the correctness of the license by reviewing the following license information stored in the token: (1) flag, (2) maintenance contract date, (3) host name and domain, (4) product name, (5) host id number, (6) license serial number, and (7) expiration date of license. This is compared to the information maintained by the application. If the information matches, the license is verified. After completing the verification process, a routine in the licensing library is initiated which checks out the license by decrementing the license units in license token by the number of licensing units being checked out.

The decoding and encoding routines allow software vendors to implement their own security mechanism to protect their licenses from unauthorized use even though they reside at the customer's site.

Below is an example of a sample application using the licensing library and the license access module written in C language:

```
#define LIC_RENEWAL_TIME (60)              /set renewal time for this session/
#define EST_LIC_RENEWAL_TIME (LIC_RENEWAL_TIME x .9)

NL_vendor_id NL_Vendor_id = 1223;                    /set vendor #/
NL_prod_num NL_Prod_num = "02"                       /set product #/
NL_version NL_Version = ( 12/20/88, "1.0" );         /set version id #/

--
        status = NL_init (vendor_id, NULL, &job_id); /initialize license service/
        if (status != NL_NO_ERROR)     /accept job id if no error/
                {
                fprintf (stderr, "nl_init failed - error =
                %d\n", status );                     /error message if error and
                                                     return/
                return;
                }
        units = 3;
        code_funcs.encode_p = nl_encode;   /pointer to encode function/
        code_funcs.decode_p = nl_decode;   /pointer to decode function/
        if (signal (SIGALRM), alarm_intr ) == (void *) -1)     /set alarm if no
                                                               error/
                {
                perror ("Cannot set SIGALRM"); /otherwise, error message/
                return;
                }
        status = NL_request (job_id, NL_Prod_num,  /request a license/
        &NL_Version,
        units, LIC_RENEWAL_TIME, NL_L2_SRCH,
        &code_funcs, NULL,
        &req_handle, NULL, &app_info);

        if (status != NL_NO_ERROR)             /no error, license checked
                {                              out from license server/
                fprintf (stderr, "nl_request failed - error =
                %d\n", status);                /otherwise, error message/
                return;
                }

        /*
         * We got a license                    /license request successful/
         */

        alarm (EST_LIC_RENEWAL_TIME);          /set alarm for license renewal
                                                time/
--      Application Runs                       /runs application/
--
        status = NL_release (req_handle);      /request to release a license/
        if (status != NL_NO_ERROR)
                {
                fprintf (stderr, "nl_release failed - error =      /otherwise, error
```

```
                %d\n", status);                          messages/
            return;
            }

    int
    alarm_intr ()
            {

    status = NL_confirm (req_handle,      /renew licensing unit with
    LIC_RENEWAL_TIME, NULL);               licensing server/
    /*
     * Verify vendor private information
     */
            }

    if (status!= NL_NO_ERROR)
    fprintf (stderr, "nl_confirm failed - error =      /otherwise, error
    %\n", status);                                     message/
            {
                    puts ("license renewed")           /successful license
            }                                          renewal/
```

The sample application given above is accompanied by self-explanatory annotation to the right margin of the codes. Of particular interest are code_func.encode_p and code_func.decode_p. Encode_p and decode_p are pointers to the software vendor's encode and decode routines, respectively. Taking the pointers in the code_func variable, the licensing library can use the pointers to invoke the decoding and encoding routines in the license access module. The three major licensing library routines, request for a license (NL_request), release a license (NL_release) and renew a license (NL_confirm) invoke the decoding and encoding routines. For example of a license access module, see Appendix 1.

In implementing the license access module, the license server becomes merely a repository for license tokens. The licensing library coupled to the application performs the procedure of authenticating the license token prior to granting a license and therefore access to run the application.

Because the level of security of the system is dictated by the license access module, the software vendors are free to make the license access module as simple or as complex as they desire. In particular, they are free to

adopt any of the encryption schemes as part of their encryption routines. If the security mechanism is broken, and the encryption known to others, then the software vendors can easily remedy the situation by releasing a new version of the product with a new license access module.

While the present invention has been particularly described with reference to Figures 1-4 as well as Appendix 1, and with emphasis on certain language in implementing a method to protect against the unauthorized use of software application in a computer network environment, it should be understood that they are for illustration only and should not be taken as limitation upon the invention. In addition, it is clear that the method of the present invention has utility in any application run in a computer network environment. It is contemplated that many changes and modifications may be made, by one skilled in the art, without departing from the spirit and scope of the invention disclosed above.

CLAIMS

1. In a computer network environment including a plurality of software applications licensed to run on at least one network of agents, said applications located on said agents wherein use of the application on a particular agent is permitted upon the grant of a license, said license being requested by a user from said agent of said applications, a system for protecting against the unauthorized use of said applications comprising:

license token means for storing licensing information of said applications; license server means connected to said agents for communicating with said applications, said license server means having a database which stores said license token means, said license server means further retrieving said license token means from said database upon a request for a license by said applications, said license server means further transmitting said license token means to said applications;

license access means connected to said agents for decoding and encoding said license token means from said license server means, said license access means being integrated with said applications, said license access means receiving said license token means from said license server means; and

licensing library means connected to said agents for verifying said decoded license token means before access to said license is granted, said licensing library means being integrated with said applications.

2. The system as defined in claim 1, wherein each said license token means containing licensing information for at least one version of each said applications.

3. The system as defined in claim 1, wherein the contents of said license token means is encrypted.

4. The system as defined in claim 1, wherein said license token means is passed between said license server means and said licensing library means for a predetermined time period.

5. The license token means as defined in claim 4, wherein during said predetermined time period, only one said applications may check out one said license token means.

6. The system as defined in claim 1, wherein said license server means receives said request for a license from said applications, said license server searches in said database for a license token means storing the license requested by said application before retrieving said license token means.

7. The system as defined in claim 1, wherein said license access means decodes the contents of said license token means before said licensing library means verifies said license token means.

8. The system as defined in claim 1, wherein said license access means encodes said license token means after said licensing library verifies said license token means and prior to returning said license token means to said license server means.

9. The system as defined in claim 1, wherein said licensing library verifies said license token means by

comparing the licensing information stored in said license token means with the licensing information maintained by said application.

10. The system as defined in claim 1, wherein said licensing library means checks out said license of said application in response to a positive comparison of the license information.

11. The licensing library means as defined in claim 10, wherein said license for said application being checked out after said licensing library verifies said license token means.

12. In a computer network environment including a plurality of software applications licensed to run on at least one network of agents, said applications located on said agents wherein use of the application on a particular agent is permitted upon the grant of a license, said license being requested by a user from said agent of said applications, a system for protecting against the unauthorized use of said applications comprising:

license token means for storing licensing information of said applications;

license server means connected to said agents for communicating with said applications, said license server means having a database which stores said license token means, said license server means further retrieving said license token means from said database upon a request for a license by said applications, said license server means further transmitting said license token means to said applications;

license access means connected to said application and accessible from said agents for decoding and encoding said license token means from said license server means, said license access means being integrated with said applications;

licensing library means connected to said application and accessible from said agents for verifying said decoded license token means before access to said license is granted, said licensing library means being integrated with said applications; and

license binding means connected to said license server means and to said licensing library means for constructing a binding file, said binding file informing said licensing library means which of said license server means may grant a license to said application.

13. The system as defined in claim 12, wherein said licensing library means are located on the same agents as said applications.

14. The system as defined in claim 12, wherein said license sever means are located on the same agents as said licensing library means.

15. The system as defined in claim 12, wherein each said license token means contains licensing information for at least one version of each of said applications.

16. The system as defined in claim 12, wherein the contents of said license means is encrypted.

17. The system as defined in claim 12, wherein said license token means is passed between said license server

means and said licensing library means for a predetermined time period.

18. The license token means as defined in claim 17, wherein, during said predetermined time period, only one of said applications may check out one said license token means.

19. The system as defined in claim 12, wherein said license server means further transmit said license token means to said licensing library means.

20. The system as defined in claim 12, wherein said license access means decodes the contents of said license token means before said licensing library means verifies said license token means.

21. The system as defined in claim 12, wherein said license access means encodes said license token means after said licensing library verifies said license token means and prior to returning said license token means to said license server means.

22. The system as defined in claim 12, wherein said license binding means constructs said binding file by contracting each said license server means to request for a list of applications it serves, said binding file containing said list of applications available from said license server means.

23. In a computer network environment including a plurality of software applications licensed to run on at least one network of agents, said applications located on

said agents wherein use of the application on a particular agent is permitted upon the grant of a license, said license being requested by a user from said agent of said applications, a system for protecting against the unauthorized use of said applications substantially as hereinbefore described with reference to the accompanying drawings.

Published 1991 at The Patent Office, State House, 66/71 High Holborn, London WC1R 4TP. Further copies may be obtained from Sales Branch, Unit 6, Nine Mile Point, Cwmfelinfach, Cross Keys, Newport, NP1 7HZ. Printed by Multiplex techniques ltd, St Mary Cray, Kent.

(12) UK Patent Application (19) GB (11) 2 309 364 (13) A

(43) Date of A Publication 23.07.1997

(21) Application No 9700921.1

(22) Date of Filing 17.01.1997

(30) Priority Data
(31) 08588848 (32) 19.01.1996 (33) US

(51) INT CL[6]
H04L 9/30

(52) UK CL (Edition O)
H4P PDCSC
U1S S2204 S2208 S2209

(56) Documents Cited
EP 0328232 A2 WO 95/23468 A1

(58) Field of Search
UK CL (Edition O) H4P PDCSA PDCSC
INT CL[6] H04L 9/30 9/32
Online:WPI,INSPEC

(71) Applicant(s)
Northern Telecom Limited

(Incorporated in Canada - Quebec)

World Trade Center Of Montreal,
380 St Antoine Street West, 8th Floor, Montreal,
Quebec H2Y 3Y4, Canada

(72) Inventor(s)
David Allan
Liam Casey
Adrian Jones

(74) Agent and/or Address for Service
M C Dennis
Nortel Patents, London Road, HARLOW, Essex,
CM17 9NA, United Kingdom

(54) Public/private key encryption/decryption

(57) In a hybrid fiber-coax distribution network, communications between a central station and particular end stations are encrypted using a working key (WK) of a symmetric encryption scheme. The central station has a public and private key (PPK) of a PPK encryption scheme, and some of the end stations can also each have a respective PPK. To provide secure communications for each end station, if the end station has a PPK, then the respective WK is generated in the central station and communicated, encrypted using the end station's public key (PK), to the end station. Otherwise, the WK is generated in the end station and communicated, encrypted using the central station's PK, to the central station. An individual identifier for each end station, and a cryptographic signature at least for end stations not having a PPK, can be communicated to the central station for authentication of the end stations.

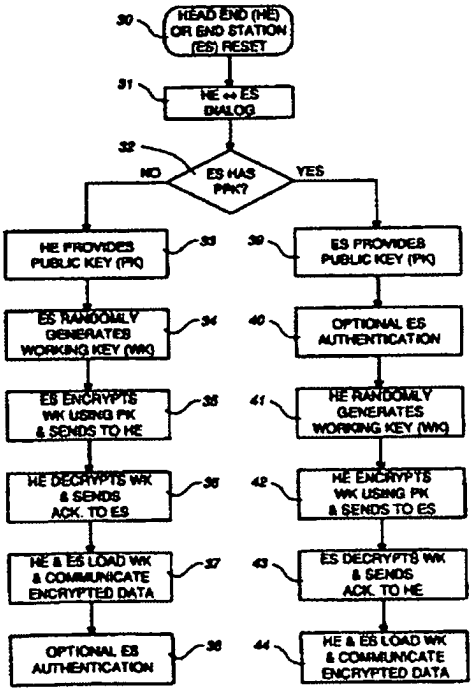


Fig. 2

GB 2 309 364 A

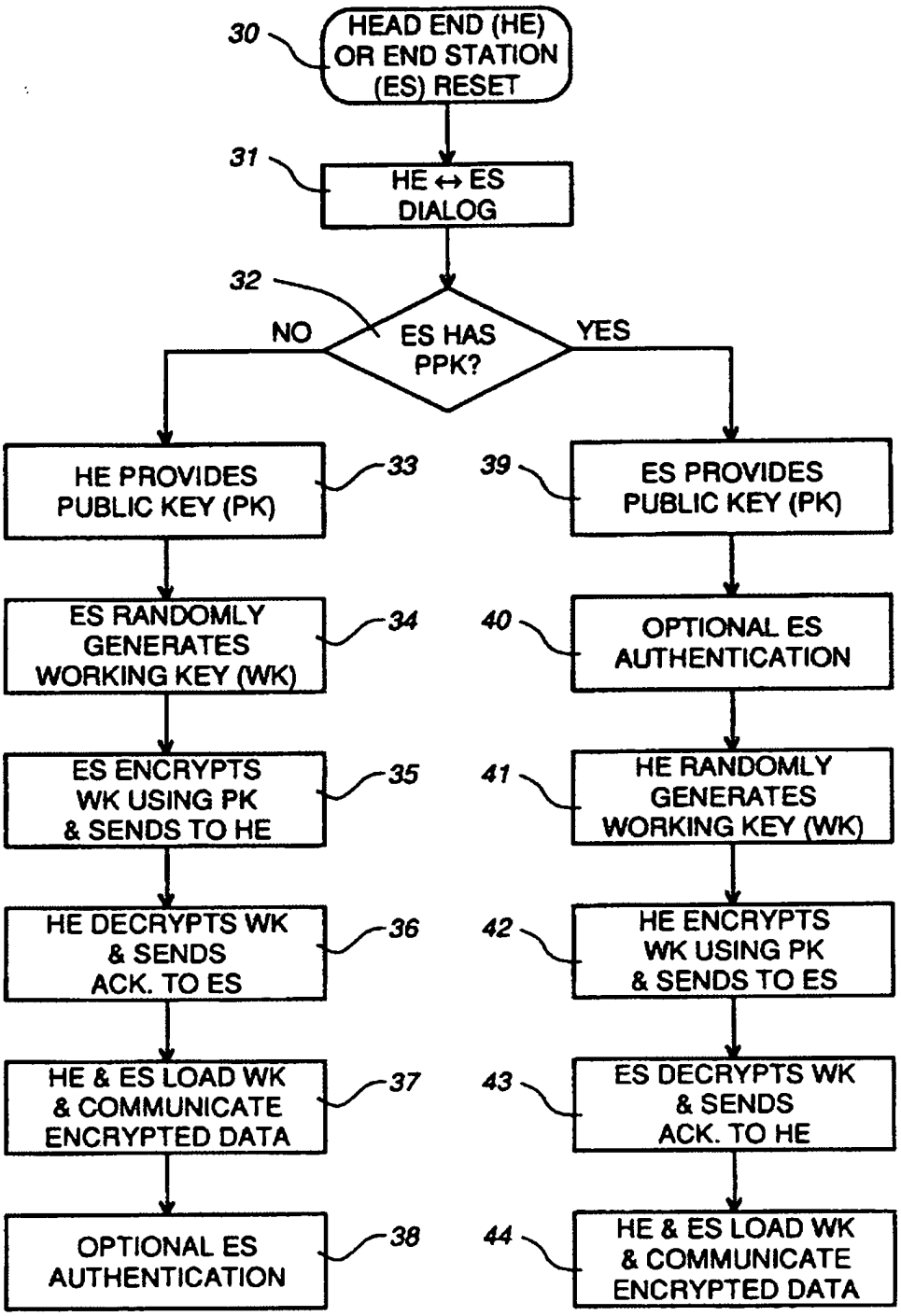

30
HEAD END (HE) OR END STATION (ES) RESET
31
HE ↔ ES DIALOG
32
ES HAS PPK?
NO
YES
HE PROVIDES PUBLIC KEY (PK)
33
ES RANDOMLY GENERATES WORKING KEY (WK)
34
ES ENCRYPTS WK USING PK & SENDS TO HE
35
HE DECRYPTS WK & SENDS ACK. TO ES
36
HE & ES LOAD WK & COMMUNICATE ENCRYPTED DATA
37
OPTIONAL ES AUTHENTICATION
38
39
ES PROVIDES PUBLIC KEY (PK)
40
OPTIONAL ES AUTHENTICATION
41
HE RANDOMLY GENERATES WORKING KEY (WK)
42
HE ENCRYPTS WK USING PK & SENDS TO ES
43
ES DECRYPTS WK & SENDS ACK. TO HE
44
HE & ES LOAD WK & COMMUNICATE ENCRYPTED DATA

Fig. 2

FACILITATING SECURE COMMUNICATIONS IN A DISTRIBUTION NETWORK

This invention relates to methods of facilitating secure communications in a distribution network, such as for example a coaxial cable or hybrid fiber-coax (HFC) network.

Background of the Invention

A distribution network, such as an HFC network in which data is communicated to subscriber end stations via optical fiber and coaxial distribution cables, is a point-to-multipoint network in which data addressed to and intended for any particular subscriber is also inevitably supplied via the network to other subscribers. If the data is not scrambled or encrypted, it can be easily monitored by these other subscribers, leading to a loss of subscriber privacy and a loss of revenues for data suppliers when the data (e.g. television programs) is supplied for a fee. Accordingly, it is important to provide a desired level of security in the data communications in a distribution network.

While various encryption and decryption schemes are known, these have a number of disadvantages associated with them in the environment of a distribution network. A significant factor in this respect is the cost and security of subscriber end stations. As a distribution network will contain large numbers of subscriber end stations, it is commercially necessary that the cost of each end station be kept relatively low. It is therefore desirable to avoid incorporating expensive security schemes in the subscriber end stations. However, subscriber end stations are also easily subject to theft, tampering, and duplication, so that complicated schemes have been considered necessary to provide adequate security.

For example, a security scheme can be implemented using an encryption key which can be stored in the subscriber end station. To prevent access to the encryption key, the store in the subscriber end station, and data lines to and from this store, must also be made physically secure. This leads to extra complexity and costs. Different subscribers may have differing security and privacy needs, which makes it desirable for the network to accommodate differing security schemes and end station costs.

A further security-related desirable aspect of a distribution network is an ability for authentication of subscriber end stations, typically using a unique end station identity which can be physically incorporated (e.g. hard wired) into the end station during manufacture.

Encryption schemes can be divided into those involving public and private keys (PPK) and those involving symmetric keys. In PPK schemes, a first station can distribute its public key, in accordance with which a second station can encrypt data and send the encrypted data to the first station, which decrypts the data using its private key. Because the private key is retained at the first station, and is not practically discoverable

by other parties, PPK schemes are considered to be secure. However, the encryption and decryption processes are relatively slow, so that such schemes are not practical for encryption of real-time high-speed data, such as television program signals, for which distribution networks are primarily intended.

In symmetric key schemes, a single key, referred to as a working key, is used by both of first and second stations to encrypt and decrypt data being communicated between the stations. The nature of the working key is such that encryption of real-time high-speed data, such as television program signals, is practical. However, these schemes require that the working key be present in both stations, and make it desirable for the working key to be periodically changed or updated. Thus symmetric key schemes require generation of a working key in one of the stations or in a third station referred to as a key distribution agent, and communication of the working key to the other station(s).

This communication itself presents a risk of the working key being insecure, and this risk increases with the frequency with which the working key is updated. It is also known to avoid this risk by using a PPK scheme for communication of a working key, and then to use the working key for data encryption.

An object of this invention is to provide a method of facilitating secure communications in a distribution network.

Summary of the Invention

One aspect of this invention provides a method of facilitating secure communications using encryption and decryption processes in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has, and one or more of the end stations can each have, a respective public and private key (PPK) of a PPK encryption scheme, comprising the steps of:

(a) determining in communications between the central station and an end station whether the end station has a PPK, if so proceeding with step (b) and if not proceeding with step (c);

(b) at the central station, determining the public key (PK) of the end station, generating a working key (WK) for encryption of communications to the end station, encrypting the WK using the PK of the end station, and communicating the encrypted WK to the end station; at the end station, decrypting the WK using the private key of the end station; and proceeding with step (d);

(c) at the end station, determining the public key (PK) of the central station, generating a working key (WK) for encryption of communications to the central station, encrypting the WK using the PK of the central station, and communicating the encrypted WK to the central station; at the central station, decrypting the WK using the private key of the central

station; and proceeding with step (d);

(d) using the WK to encrypt at the central station, and to decrypt at the end station, communications from the central station to the end station.

Another aspect of this invention provides a method of facilitating secure communications in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has a public and private key (PPK) of a PPK encryption scheme and each end station has an individual identity (ID) and an individual cryptographic signature encrypted using a private key of a predetermined PPK encryption scheme, comprising the steps of: communicating the ID of an end station to the central station; at the end station, generating a working key (WK) for encryption of communications between the end station and the central station and encrypting the WK using the public key of the central station; communicating the encrypted WK from the end station to the central station; at the central station, decrypting the encrypted WK using the private key of the central station; communicating the cryptographic signature of the end station to the central station; and at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme for authentication of the end station.

Brief Description of the Drawings

The invention will be further understood from the following description with reference to the accompanying drawings, in which:

Fig. 1 illustrates parts of a distribution network to which the invention is applied; and

Fig. 2 is a flow chart illustrating steps of a method for facilitating secure communications in the network in accordance with the invention.

Detailed Description

The invention is described below in the context of a hybrid fiber-coax (HFC) distribution network in which signals are distributed from a central station or head end (HE) to a large number of subscriber end stations (ES) via optical fibers and coaxial cables in known manner. An example of such a network is described in Warwick United States Patent No. 5,408,259 issued April 18, 1995 and entitled "Data Modulation Arrangement For Selectively Distributing Data". Typically in such a network digital data communications are provided between any ES and the HE using asynchronous transfer mode (ATM) cells which are communicated in both directions, i.e. downstream from the HE to the ES and upstream from the ES to the HE, using suitable modulation schemes and carrier frequencies outside the bands used for analog television signals also carried on

the coaxial cables. However, it is observed that the invention is equally applicable to other forms of distribution network.

Referring to Fig. 1, there is illustrated parts of a distribution network in which many end stations, only two of which are shown and are referenced 10 and 12, are connected via branched cables 14 of the distribution network to a head end 16, via which the end stations have access to a network (not shown) which for example supplies digital television program signals subscribed to by end station subscribers. The cables 14 can comprise both optical fiber and coaxial cables forming a hybrid fiber-coax arrangement, on which the digital signals can be communicated in known manner using ATM cells.

As can be appreciated from the illustration in Fig. 1, signals communicated by the head end 16 and intended for any particular end station will actually be delivered via the cables 14 to all of the end stations. For secure and/or private communication of the signals, the head end 16 includes an encryption engine 18 which encrypts the signals in accordance with a working key known only by the head end and the intended end station, which also includes an encryption engine 20 which decrypts the signals for use. These working keys are similarly used for communications in the opposite direction, from the end station to the head end 14. The working keys of this symmetric key encryption scheme are provided in the head end and the end station in a manner which is described in detail below.

The end stations 10 and 12 are of two types, with differing levels of security to enable different security needs of subscribers to be accommodated. The end station 12 represents a relatively secure end station, which includes its own public and private keys of a PPK encryption scheme. As explained in the introduction, such an end station has a relatively high complexity and cost, because of the need for secure storage of the keys and operation of the PPK encryption. Other end stations, which do not have their own public and private keys and accordingly can be provided at a much lower cost, are represented by the end station 10. The network as a whole may have an arbitrary mix of these two types of end station.

Each end station 10 or 12 also has an individual, unique identity number, which is stored (e.g. hard wired) into the ES during its manufacture. This is referred to as a global ID (identity). The global IDs of all of the end stations are stored in a database 22, which can be colocated with the head end 16 or separately from it and with which the head end 16 communicates via a path 24. The head end 16 also has its own public and private keys of a PPK encryption scheme.

Fig. 2 shows steps of a process which is followed in order to set up secure communications between the head end 16 and one of the end stations 10 or 12. This process takes place between the head end and the respective end station without involvement of any other node such as a central key distribution agent, and is described

below as being initiated in each case following any reset (e.g. following a power-up) of either the head end 16 or the respective end station. Consequently, the working key which is used for encrypting the communications between the head end and the end station is changed on any reset. However, the same process can alternatively or additionally be carried out on demand, and/or periodically to provide periodic changes of the working key. It is also observed that the encrypted communications take place between the encryption engines 18 in the head end 16 and 20 in the respective end station 10 or 12, and communications on the network access side of the head end 16 are not subject to the same encryption.

In Fig. 2, a block 30 represents a reset of the head end (HE) or end station (ES), in response to which, as shown by a block 31 in Fig. 2, a dialog or handshake is carried out between the HE and the ES to establish communications between them. These communications are effected using unencrypted ATM cells using addresses of the end station and the head end. As a part of this dialog, as shown by a block 32 in Fig. 2 the head end 16 interrogates the end station to determine whether or not the end station has its own public and private keys. If not, i.e. if the end station is an end station 10 as described above, then the process continues with successive blocks 33 to 38 in Fig. 2. If the interrogation establishes that the end station is an end station 12 having its own public and private keys, then the process instead continues with blocks 39 to 44 in Fig. 2.

In the former case of an end station 10, as shown by the block 33 the head end 16 communicates its public key (PK) to the end station 10; this communication can form part of the dialog block 31. The end station 10 randomly generates (block 34) a working key (WK) for communicating signals in a symmetric key encryption scheme, and encrypts (block 35) this working key in accordance with the supplied public key, sending the encrypted working key in a message to the head end 16. The head end 16 decrypts (block 36) the encrypted working key from this message in accordance with its private key, which is not known to others so that the communication of the working key from the end station 10 to the head end 16 is secure, and optionally but preferably sends an acknowledgement to the end station 10. As shown by the block 37, the head end 16 and the end station 10 then load their encryption engines 18 and 20 respectively with the working key, and thereafter (until this process is repeated, for example in response to a subsequent reset at either end) communications between them take place with data encrypted in accordance with the working key. An optional additional step represented by the block 38 provides for authentication of the end station 10 in a manner described below.

Conversely, in the latter case of an end station 12, as shown by the block 39 the end station 12 communicates its public key (PK) to the head end 16; this communication can form part of the dialog block 31. An optional authentication step for the end station

12 can be carried out by the head end 16 as represented by the block 40 in a manner described below. The head end 16 randomly generates (block 41) a working key (WK) for communicating signals in a symmetric key encryption scheme, and encrypts (block 42) this working key in accordance with the supplied public key of the end station 12, sending the encrypted working key in a message to the end station 12. The end station 12 decrypts (block 43) the encrypted working key from this message in accordance with its private key, which is not known to others so that the communication of the working key from the head end 16 to the end station 12 is secure, and optionally but preferably sends an acknowledgement to the head end 18. As shown by the block 44, the head end 16 and the end station 12 then load their encryption engines 18 and 20 respectively with the working key, and thereafter (until this process is repeated, for example in response to a subsequent reset at either end) communications between them take place with data encrypted in accordance with the working key.

It can be seen from the above description that, in the relatively secure but more expensive situation in which the end station 12 includes its own public and private keys, these are used for communicating a working key generated in the head end, whereas in the other case the end station 10 generates the working key and this is communicated to the head end using the latter's public key.

The optional step of authentication of the end station 12 in the block 40 as described above can make use of the global ID of the end station 12 together with data in the database 22, in which the public key of the end station 12 is stored in association with this global ID. As part of the dialog block 31, the end station communicates its global ID to the head end 16. In the step 40, therefore, the head end 16 can communicate via the path 24 with the database 22 to confirm that the public key which it has received from the end station 12 in the step 39 matches that stored in the database 22 for this end station's global ID, the subsequent steps 41 to 44 only being followed if this authentication step is successful.

Alternatively, or in addition, the optional end station authentication step of block 40 can comprise the steps of the head end sending an unencrypted message to the end station 12 with a request that it be cryptographically signed. In accordance with this request, the end station 12 produces a digest of the message using a known hashing function (thereby reducing the data to be encrypted), encrypts this digest in accordance with its private key, and sends the encrypted message digest to the head end 16. The head end 16 then decrypts this in accordance with the public key of the end station, retrieved from the database 22, to confirm the digest of its original message which the head end also produces using the hashing function.

It can be seen that, alternatively, the steps represented by the blocks 39 and 40 in Fig. 2 could be replaced by a single step in which the head end 16 determines the public

key of the end station 12 from the database 22 in accordance with the global ID of the end station 12 supplied in the dialog 31, without any authentication of the end station or any communication of the public key from the end station 12.

The above sequences provide a particularly strong or secure authentication of the end station 12. For the end station 10 which does not have its own public and private keys, a weaker but still valuable authentication can be provided as shown by the block 38. The authentication block 38 is shown in Fig. 2 as the final block in the process because this enables the exchange of data in the authentication process to be encrypted in accordance with the working key, but this authentication step could alternatively be provided anywhere else in the sequence of steps from the blocks 31 to 37.

For this optional authentication step, the end station 10 is manufactured (e.g. hard wired) with not only its global ID, but also a cryptographic signature. Conveniently, the end station 10 is manufactured with a certificate comprising data including the global ID of the end station and the public key of the manufacturer and a cryptographic signature comprising an encryption, in accordance with the private key of the manufacturer, of a digest of that data produced using a known hashing function. The public key of the manufacturer can also or instead be stored in the database 22. The optional end station authentication step of the block 38 comprises a communication of the cryptographic signature from the end station 10 to the head end 16 (as explained above this could be a part of the dialog 31 or any later step, but the encryption after the block 37 obstructs public observation in the network of cryptographic signatures). The head end 16 then confirms the authenticity of the end station 10 by decrypting the cryptographic signature using the manufacturer's public key, producing a digest from the same data (global ID and public key, both of which can be communicated in the dialog step 31 or later) and the known hashing function, and matching these.

This is a relatively weak authentication, in that identical copies of the end station 10, including duplicated data and cryptographic signatures, could operate at different times on the network without this being detected. However, simultaneous operation of two or more such duplicates would be detected by the fact that two or more end stations would be supplying the same global ID which is supposedly unique. Thus even such a weak authentication is valuable especially in detecting illicit large-scale duplication of end stations.

The processes in accordance with the invention as described above provide a number of significant advantages over known configurations. In particular, requirements for secure storage of public and private keys are minimized in the network as a whole, and eliminated for the end stations 10 which can accordingly be provided at relatively lower cost. At the same time, end stations 12 with greater security can be provided, and the head end 16 can operate simultaneously with both types of end station. This, combined

with optional authentication of the end stations as described above, enables different degrees of security to be easily provided in the network in accordance with service requirements.

Furthermore, renewal of the working keys at reset is simpler than providing time-based schedules for changing encryption keys, and key exchanges take place only between the head end and the end station which use the keys, thereby enhancing security compared with distribution of keys from a key distribution agent. In addition, all of the data flowing between the head end and any particular end station 10 or 12, between successive resets, can be encrypted using a single working key, thereby simplifying the encryption and decryption processes. However, it is observed that different working keys could be generated, communicated, and used in the same manner as described above for encrypting and decrypting different types of information, or different services, for a single end station 10 or 12.

Although particular embodiments of the invention have been described in detail, it should be appreciated that numerous modifications, variations, and adaptations may be made without departing from the scope of the invention as defined in the claims.

WHAT IS CLAIMED IS:

1. A method of facilitating secure communications using encryption and decryption processes in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has, and one or more of the end stations can each have, a respective public and private key (PPK) of a PPK encryption scheme, comprising the steps of:

(a) determining in communications between the central station and an end station whether the end station has a PPK, if so proceeding with step (b) and if not proceeding with step (c);

(b) at the central station, determining the public key (PK) of the end station, generating a working key (WK) for encryption of communications to the end station, encrypting the WK using the PK of the end station, and communicating the encrypted WK to the end station; at the end station, decrypting the WK using the private key of the end station; and proceeding with step (d);

(c) at the end station, determining the public key (PK) of the central station, generating a working key (WK) for encryption of communications to the central station, encrypting the WK using the PK of the central station, and communicating the encrypted WK to the central station; at the central station, decrypting the WK using the private key of the central station; and proceeding with step (d);

(d) using the WK to encrypt at the central station, and to decrypt at the end station, communications from the central station to the end station.

2. A method as claimed in claim 1 wherein each end station has an individual identity (ID) and step (a) includes the step of communicating the ID of the end station to the central station.

3. A method as claimed in claim 2 wherein in step (b) the PK of the end station is determined by the central station from a database using the ID of the end station.

4. A method as claimed in claim 1, 2, or 3 wherein step (b) further comprises an end station authentication step comprising the steps of communicating an unencrypted message from the central station to the end station, producing an encrypted message at the end station using the private key of the end station, communicating the encrypted message to the central station, decrypting the message at the central station using the PK of the end station, and comparing the decrypted message with the original message.

5. A method as claimed in claim 4 wherein in step (b) the end station authentication step is carried out before the step of communicating the encrypted WK to the end station.

6. A method as claimed in any of claims 1 to 5 wherein in step (b) the PK of the end station is communicated to the central station from the end station.

7. A method as claimed in claims 2 and 6 wherein in step (b) the PK of the end station is verified by the central station from a database using the ID of the end station.

8. A method as claimed in any of claims 1 to 7 wherein a plurality of end stations which do not have a PPK each have an individual cryptographic signature encrypted using a private key of a predetermined PPK scheme, step (a) or (c) includes the step of communicating the cryptographic signature of the end station to the central station, and step (c) further comprises an end station authentication step comprising, at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme.

9. A method as claimed in claims 2 and 8 wherein the individual cryptographic signature comprises an encryption of data derived from the ID of the respective end station.

10. A method as claimed in claim 8 or 9 wherein the predetermined PPK scheme uses a private key and a public key of a source of the end station.

11. A method as claimed in claim 8, 9, or 10 wherein the cryptographic signature is communicated to the central station in step (c).

12. A method as claimed in claim 11 and including the steps of encrypting the cryptographic signature at the end station, and decrypting the encrypted cryptographic signature at the central station, using the WK.

13. A method as claimed in any of claims 1 to 12 and further comprising the step of using the WK to encrypt at the end station, and to decrypt at the central station, communications from the end station to the central station.

14. A method of facilitating secure communications in a distribution network comprising a central station and a plurality of addressable end stations, in which communications from the central station addressed to and intended for a particular end station are delivered via the network to a plurality of end stations, wherein the central station has a public and private key (PPK) of a PPK encryption scheme and each end station has an individual identity (ID) and an individual cryptographic signature encrypted using a private key of a predetermined PPK encryption scheme, comprising the steps of:

communicating the ID of an end station to the central station;

at the end station, generating a working key (WK) for encryption of communications between the end station and the central station and encrypting the WK

using the public key of the central station;

communicating the encrypted WK from the end station to the central station;

at the central station, decrypting the encrypted WK using the private key of the central station;

communicating the cryptographic signature of the end station to the central station; and

at the central station, decrypting the cryptographic signature using a public key of the predetermined PPK scheme for authentication of the end station.

15. A method as claimed in claim 14 wherein the individual cryptographic signature comprises an encryption of data derived from the ID of the respective end station.

16. A method as claimed in claim 14 or 15 wherein the predetermined PPK scheme uses a private key and a public key of a source of the end station.

17. A method as claimed in claim 14, 15, or 16 wherein the step of communicating the cryptographic signature of the end station to the central station comprises the steps of encrypting the cryptographic signature at the end station using the WK, communicating the encrypted cryptographic signature from the end station to the central station, and decrypting the encrypted cryptographic signature at the central station using the WK.

18. A method of facilitating secure communications in a distribution network, substantially as hereinbefore described with reference to Figs 1 and 2 of the accompanying drawings.

The Patent Office

12

Application No: GB 9700921.1
Claims searched: 1-13

Examiner: Mr B J Spear
Date of search: 19 March 1997

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): H4P (PDCSC)

Int Cl (Ed.6): H04L 9/30

Other: Online: WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
	NONE	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

The Patent Office

Application No: GB 9700921.1
Claims searched: 14-17

Examiner: Mr B J Spear
Date of search: 21 May 1997

Patents Act 1977
Further Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): H4P (PDCSA)

Int Cl (Ed.6): H04L 9/32

Other: Online: WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP0328232A2 (Fischer)	-
A	WO 95/23468A1 (Merdan)	-

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

09

(12) UK Patent Application (19) GB (11) 2 316 503 (13) A

(43) Date of A Publication 25.02.1998

(21) Application No 9617596.3

(22) Date of Filing 22.08.1996

(51) INT CL[6]
G06F 1/00

(52) UK CL (Edition P)
G4A AAP

(56) Documents Cited
GB 2236604 A EP 0332304 A2 WO 93/11480 A1
US 5375206 A US 4924378 A

(58) Field of Search
UK CL (Edition O) G4A AAP
INT CL[6] G06F

(71) Applicant(s)
ICL Personal Systems Oy

(Incorporated in Finland)

PO Box 458, SF-00101 Helsinki, Finland

(72) Inventor(s)
Tapani Lindgren

(74) Agent and/or Address for Service
S M Dupuy
International Computers Limited, Cavendish Road,
STEVENAGE, Hertfordshire, SG1 2DY,
United Kingdom

(54) Software licence management

(57) A software licence management method and system is for a computer system including at least one server (1,5) and particularly for a plurality of computers connected via a network. Before a service (2) can offer functionality to a user it has to check that the user has a licence for that service. A licensing subsystem (3) has associated with it a ticket database (4) that hold tickets corresponding to existing licences. Tickets, if available, are issued to a service on request, thereby verifying the existence of a licence. The receipt of a ticket allows a service to offer functionality.

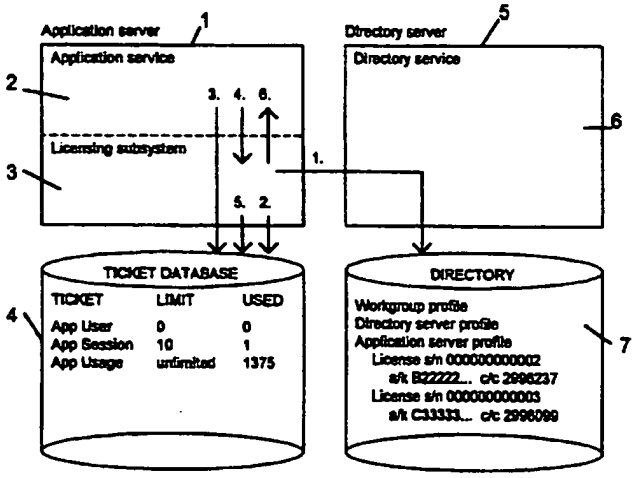


FIG 1

GB 2 316 503 A

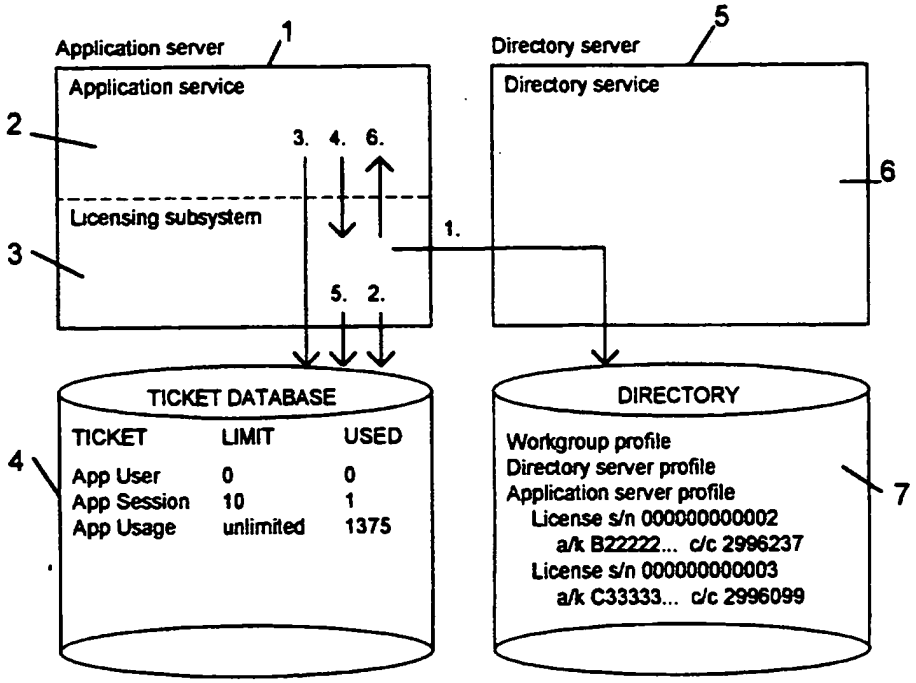

Application server
1
Application service
2
3. 4. 6.
Licensing subsystem
3
1.
5. 2.
Directory server
5
Directory service
6
TICKET DATABASE
TICKET LIMIT USED
4
App User 0 0
App Session 10 1
App Usage unlimited 1375
DIRECTORY
Workgroup profile
Directory server profile
Application server profile
License s/n 000000000002
a/k B22222... c/c 2996237
License s/n 000000000003
a/k C33333... c/c 2996099
7

FIG 1

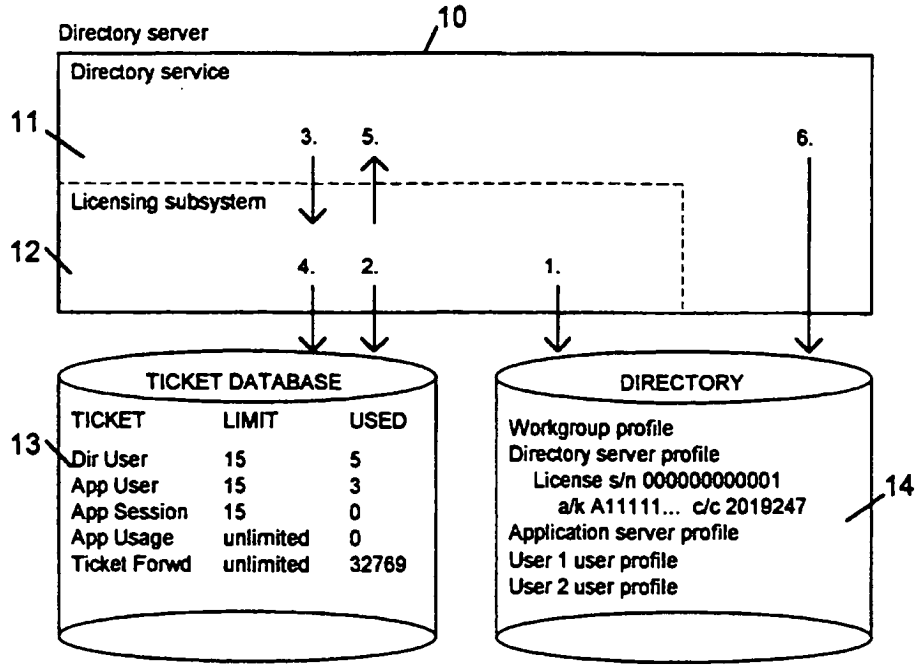

Directory server
10
Directory service
11
3.
5.
6.
Licensing subsystem
12
4.
2.
1.
TICKET DATABASE
TICKET
LIMIT
USED
13
Dir User
15
5
App User
15
3
App Session
15
0
App Usage
unlimited
0
Ticket Forwd
unlimited
32769
DIRECTORY
Workgroup profile
Directory server profile
License s/n 000000000001
a/k A11111... c/c 2019247
14
Application server profile
User 1 user profile
User 2 user profile

FIG 2

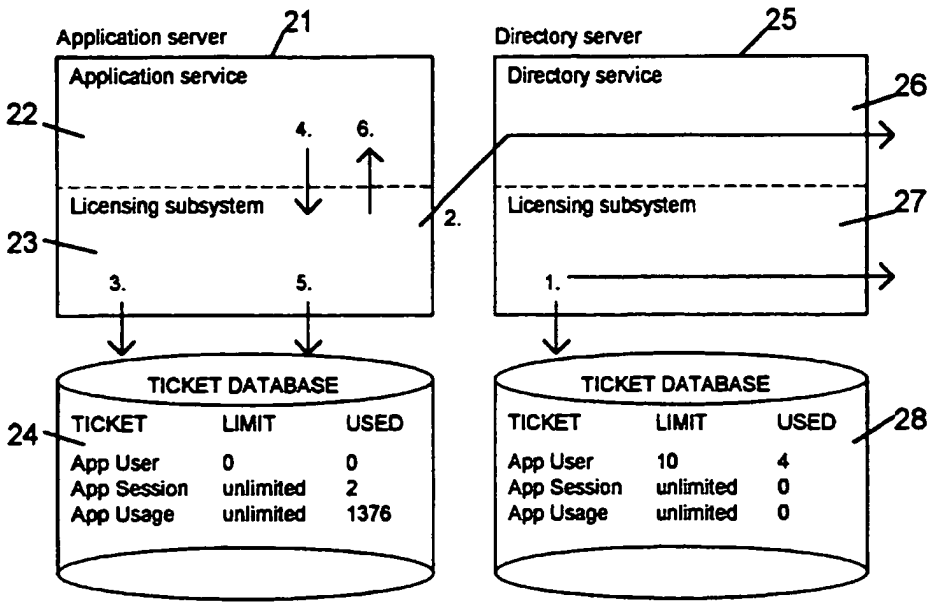

Application server
21
Application service
22
4.
6.
Licensing subsystem
23
2.
3.
5.
TICKET DATABASE
24
TICKET
LIMIT
USED
App User
0
0
App Session
unlimited
2
App Usage
unlimited
1376
Directory server
25
Directory service
26
Licensing subsystem
27
1.
TICKET DATABASE
28
TICKET
LIMIT
USED
App User
10
4
App Session
unlimited
0
App Usage
unlimited
0

FIG 3

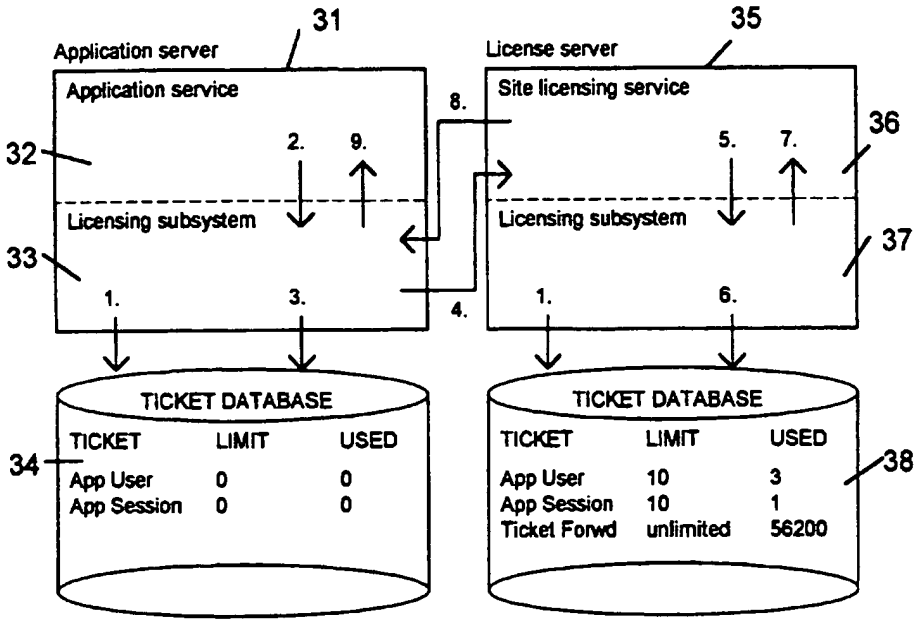

31
Application server
Application service
32
2.
9.
Licensing subsystem
33
1.
3.
TICKET DATABASE
TICKET
LIMIT
USED
34
App User
0
0
App Session
0
0
8.
4.
35
License server
Site licensing service
36
5.
7.
Licensing subsystem
37
1.
6.
TICKET DATABASE
TICKET
LIMIT
USED
38
App User
10
3
App Session
10
1
Ticket Forwd
unlimited
56200

FIG 4

2316503

SOFTWARE LICENCE MANAGEMENT

This invention relates to software licence management and in particular to licence management for software running on a plurality of computers connected via a network.

Conventionally, licences have been provided by software vendors as separate licences for individual workstations or as a single licence for a number of workstations. Various schemes have been proposed in order to try and make unlicensed software unusable, in particular pirated (illegal) copies of software. Other schemes have been proposed such as in order to achieve low initial software costs but licensing royalties consistent with the extent of use, in order not to deter low-usage users from purchasing particular forms of software, and thus to reduce piracy, whilst still enabling a vendor to collect higher dues from high-usage users.

The present invention is, particularly, concerned with a distributed system consisting of various server and client programs running on various computers which are connected via a local or wide area network, and an object is to provide server software licensing which ensures that all software running in the network has been purchased legally.

According to one aspect of the present invention there is provided a software licence management method for use with a computer system including at least one server, the method being such that before a service can offer functionality to a user, the said service shall verify that the user has a licence for said service, and wherein the computer system further includes a licensing subsystem with which are associated service tickets corresponding to existing licences, the method including the steps of the said service requesting a respective service ticket from the licensing

subsystem prior to offering functionality to the user, and the licensing subsystem issuing a said service ticket to the said service, if one is available, thereby verifying the licence exists and allowing the said service to offer functionality.

According to another aspect of the present invention there is provided a computer system including at least one server and a software licence management system, the management system being such that before a service can offer functionality to a user, the service shall verify that the user has a licence for said service, the management system including a licencing subsystem with which are associated service tickets corresponding to existing licences, and the management system being such that a said service ticket is issued to a service, if one is available, upon request by the service, thereby verifying existence of a licence and allowing the said service to offer functionality.

Embodiments of the invention will now be described with reference to the accompanying drawings, in which:

Figure 1 Illustrates obtaining a session or usage ticket for an application,

Figure 2 Illustrates obtaining a user ticket for a directory server,

Figure 3 Illustrates independent licence sharing, and

Figure 4 Illustrates licence sharing with a site licencing service,

Various terms used in the following will first be defined. For the purposes of the description the software is considered to relate to a Groupware Office system which provides various facilities including mail, for example.

Definitions

"Server"	An instance of server software running on a server computer. Usually only one such instance runs on any one computer. Each "server" implements one or more collections of related functions called "function sets", examples of which are directory, mail, library etc. The "directory function set" includes functions to access a database that contains information about the Groupware Office system.
"Client"	Any piece of software that connects to the "server" using a "client-server protocol" to access the functions offered by the "server". A "client" may be a program run by a user on a workstation, or a part of any other program.
"Session"	An instance of client-server dialogue between one "client and one "server". Each "session" allows the "client" to use the functions of one or more "function sets".
"Directory Server"	A server that implements the directory function set.
"Mail Server"	A server that implements the mail function set. [A server may be a directory server and a mail server simultaneously.]
"User"	A person (actual or virtual) listed in the database of a directory server.

"User profile"	The information pertaining to one user stored in a directory database entry, such as the name of the user, user authentication information, the list of servers and function sets the user is permitted to access, etc.
"Server profile"	The name and network address of a server and the list of services offered by it, as stored in a directory database entry.
"Service profile"	Information stored in a directory database entry about one service in one server. If the same type of service is offered by more than one server, each instance has its own profile.
"Site profile"	Information stored in a directory database entry about one site.
"Site"	A set of servers connected to a single directory server. Each server belongs to exactly one site, and each site has exactly one directory server. Other servers in the site are optional, usually unlimited in number, and sometimes called member servers or application servers.
"Enterprise"	A set of sites that share their directory databases. The directory servers in each site replicate directory information to other directory servers in the enterprise. Each directory server contains both "local" and "external" information. One of the directory servers, the "enterprise directory server" controls the others, which are

"site directory servers".

"Subsystems" — Collections of programs and/or subroutines that perform a set of interrelated functions. Some subsystems implement a function set within a server program, while others run independently as stand-alone applications. Many subsystems are collections of common subroutines called by other subsystems. Client programs are also subsystems.

"Subsystem id" — A respective unique number identifying every type of subsystem. Some systems use two ids, a "real subsystem id" when dealing with licensing issues and an "alias subsystem id" when performing a task on behalf of a virtual entity, such as "generic gateway no 9".

Not every subsystem software needs to be purchased individually. Most collections of subroutines can be used freely by other subsystems.

"Services" — Those subsystems that need to be explicitly purchased.

Service types may include directory service, mail service, fax gateway, X.400 gateway, enterprise option, library service, power library option, etc. Each service is located in one server, either as a function set of the server program or a standalone application running in the same computer. Many services of the same type can exist in different servers.

The software licence management system of the present invention proceeds from the premise that before offering any usable functionality to the users, services shall verify from a "licensing subsystem" that a licence for the service exists. To achieve that, it is proposed that the licensing subsystem holds "service tickets" and a service requests a permission to offer its functions to the user by requesting a corresponding "service ticket" be provided from the licensing subsystem. Each service knows that kind of tickets are needed to fulfil the service's functionality. The licensing subsystem has to keep track of the available licences and of the service tickets it has issued. A service ticket may be considered as partially the equivalent of a password in that one must be provided before a service can operate.

A "licence" is a permission to use one or more services within certain limits. Typically these limits are "license duration", which specifies the maximum length of the period when the licence may be used (the "active" period) and the "licence size", which specifies the maximum number of users of the licence. The interpretation of "number of users" varies from service to service. It may, for example, mean the number of users in the local directory that are allowed to use the service, or the number of concurrent sessions that are connected to the service. Licence duration and/or size may also be unlimited.

When a customer purchases a Groupware, for example, software product which employs the software licence management method and system of the invention from a supplier, as well as the media containing the software itself and associated documentation, there is obtained a single licence to one or more services. Each said product has a unique serial number. The license is supplied in the form of a licence agreement document on which the licence information is printed. This licence information consists of the serial number of the product and an "activation key" for the licence. The licence

size and duration and the included services are encoded into the activation key.

The software license management method and system of the invention is such that the Groupware software may be copied and installed by the customer without any technical restrictions, but before any of the services can be used, a corresponding licence must be installed and activated. Licence installation consists of entering the license information (serial number and activation key) into the server profile of a server in the directory server's database, ie in the site directory, in the server profile of the server in question. Licence activation consists of setting the active period of the installed licence so that service tickets can be issued. Typically licence installation and licence activation are performed simultaneously by the server setup program. The license information is stored in the site directory, in the server profile of the server in question.

As will be appreciated, there also exists "evaluation licences" which allow a prospective customer to use a service for a short trial period before actually purchasing it. These licences typically have a very short duration and a relatively small size. The product serial numbers associated with such evaluation licences are not necessarily unique, since the licence information may be distributed on CD-ROMs or via public networks.

As mentioned above, each software product contains just one licence, although that licence may include a large number of services, for example, enough to build a complete Groupware Office site with all of the basic services. Alternatively, the licence may include just one service. Product with that kind of licence could be used to expand the capacity or functionality of an existing Groupware Office System.

The core of the process of designing a product is, therefore, determining what services will be included and the size and duration of the licence. This information is encoded into a number, the covert code, which may be a 7-digit number, for example. The building of the covert code is discussed in more detail hereinafter.

The amount of information that can be encoded into the covert code is limited by the size of the code. Therefore, there are some necessary restrictions on what kinds of licences are possible. The most obvious limitation is that the size and duration can only take certain discrete values. Also, the same size and duration will apply to all services covered by the licence. Another restriction is that only the most common groups of services can be combined freely into a multi-service licence. Other services will have to be licensed individually.

The covert code, which specifies the properties of the software licence, is thus a part of the product description in the logistics database. When the product is manufactured it has the unique serial number, referred to above, assigned to it. The actuation key for the license is calculated as a function of the serial number and the covert code using a secret algorithm. The serial number and activation key may be printed on a label, which is attached to the licence agreement document.

When creating a site, a customer must have a licence that includes a site creation ticket. This licence is installed for the directory server. The customer may also install additional licences for the directory server and for other servers. Each licence may apply to one or more services. Some licences are valid only in that server for which they are installed, whilst other licences may be shared with other servers at the same site (see later). Shared licences would usually be installed in the server profile of the directory

server, although optionally, and with some restrictions, another server may be designated as the licence server. The serial number of the first licence installed in a directory server's profile can be used to identify the site uniquely. Thus the directory server is computer number 1. Other servers in the site will use the same site id but differing computer numbers for identification.

When a service program is about to execute an action which requires that a customer possesses a licence for that service, the service program must first obtain a corresponding service ticket from the licensing subsystem. The actions concerned are ones which are potentially profitable for the customer and may, for example, include namely:

setting up a new Groupware site;

creating a new user account;

setting up an instance of the mail service;

enabling mail usage for a user and creating a user mailbox;

starting a session between a mail UI client and the mail server;

sending a mail message;

relaying a mail message to an X.400 mail network.

Each kind of action requires a specific kind of service ticket. To obtain a ticket the service needs to specify the ticket type and the number of tickets. The service tickets are only identified by ticket type. There is a licensing subsystem in each server and it counts the number of

different tickets in all available licences and keeps track of how many licences of each type are being used in the server.

The steps involved in obtaining various licences will now be described in greater detail. With respect to Figure 1 there will, firstly, be described the case of an application service running in a separate server from the directory server obtaining a session or usage ticket.

Figure 1 illustrates schematically an application server 1, providing an application service 2 and having a licensing subsystem 3, with an associated ticket database 4, a directory server 5 providing a directory service 6 and having an associated directory 7. The ticket database 4 has stored therein details of ticket types, the limit, if any, of the number of such tickets which are available and the number of used tickets for each type. The ticket types as illustrated are "App User" (Application User), "App Session", "App Usage". The directory 7 has stored therein, the "Workgroup profile", the "Directory server profile", the Application server profile. In the example illustrated, the application server has two associated licenses whose serial numbers (s/n) are 000000000002 and 000000000003, respectively, whose activation keys (a/n) are of the form B22222... and C33333..., respectively, and whose covert codes (c/c) are 2996237 and 2996099, for example, respectively.

When the licensing subsystem 3 on the application server starts, it fetches the application server's server profile from the directory service 6, 7 using the directory API (Application Programming Interface) (Step 1 in Figure 1). The licensing subsystem 3 analyses the licences and updates the limits of each ticket type in the local ticket database 4. The numbers of used tickets are not modified at this time, the old accumulated values being maintained (step 2).

When the application service 2 starts, and before any user logs in, it tells the licensing subsystem 3 to set the number of used session tickets to zero. This frees any session tickets that may have been left unreturned at the end of a session because of a system crash etc. (Step 3). The application service 2 then requests a service ticket (session or usage) from the licensing subsystem 3, since without a ticket it cannot proceed. (Step 4). The licensing subsystem checks the ticket availability in the local ticket database 4 and updates the used ticket count (step 5), to take into account the requested ticket, before issuing the ticket to the application service (step 6), which then proceeds since it has determined that there exists the appropriate licence.

In the embodiment of Figure 2, the procedure whereby a directory service obtains a ticket for adding a user to a directory is illustrated.

A directory server 10 provides a directory service 11 and includes a licensing subsystem 12 with an associated ticket database 13, the directory service 11 having an associated directory 14. The ticket database 13 has stored therein details of ticket types, the limit, if any, of the number of such tickets which are available, and the number of used tickets for each type. The ticket types are illustrated as "Dir User" (Directory User), "App User", "App Session", "App Usage" and "Ticket Forwd" (Ticket Forwarding). The directory 14 has stored therein the "Workgroup profile", the "Directory server profile", the "Application Server profile" and the user profile of two users, User 1 and User 2. The Directory server has a license serial number (s/n) 000000000001, with an actuation key (a/) of the form A11111..., and a corresponding covert code (c/c) 2019247, for example.

When the licensing subsystem 12 on the directory server 10 starts, it fetches the directory server's server profile directly from the directory 14 (step 1). The licensing

subsystem 12 analyses the licenses and updates the limits of each ticket type in the ticket database 13. The numbers of used tickets are not modified at this time, the old accumulated values being maintained (step 2).

When it is desired to add a new user to the directory, the directory service 11 requests a user ticket from the licensing subsystem 12 (step 3). The licensing subsystem 12 checks ticket availability in the local ticket database 13 and updates the used ticket count (step 4) to take into account the requested ticket. The licensing subsystem 12 issues the requested user ticket to the directory service 11 (step 5). The directory service then adds the new user to the directory 14, that is it adds its user profile.

To ensure consistency, the directory service 11 may periodically count the number of users in the directory 14 and tell the licensing subsystem 12 to set the used ticket count accordingly.

When a licence is installed, the start time of its active period will be fixed. By default this is the same as the installation time, but any time in the past or in the future may be specified. If the licence has a limited period, the end time will also be set. The licence will be active whenever the current time is after the start time and before the end time.

A customer may wish to deactivate a licence so that it cannot be used. Thus can be done at any time by altering the end time of the licence with the server setup program. The end time may be altered freely, as long as the active period does not exceed the licence duration.

Once installed, limited-duration licences are fixed, ie they cannot be removed, except by remaining the entire site directory, or moved to another server, and their start time

may not be altered. These restrictions, however, do not apply to unlimited-duration licences. They may be removed, reinstalled, moved or altered freely. The only restriction that remains is that a licence may only be installed for one server at a time.

A further restriction applies to the licence that has been used to create a site. This licence cannot be removed or deactivated, except by removing the entire site.

The licensing method described with reference to Figures 1 and 2 applies only to local licences, ie the tickets included in a license can only be issued in one server, the server whose server profile contains the licence. Often there is a need to share a single licence between two or more servers, so that tickets can be issued in all of them. Most commonly, the user tickets for an application are needed in the directory server, and session and usage tickets in the application servers.

If a licence includes an unlimited number of a certain kind of service ticket, sharing the licence is not very complicated. Any server can read the licences in any other server's profile. If the licensing subsystem in a server can verify that another server's licence contains an unlimited supply of freely shareable tickets, it will deduce that these tickets may be issued without limit in any server, independently of other servers. This is independent license sharing.

Not all licences are necessarily shareable, even if they contain an unlimited number of tickets. Whether each licence is shareable or not is a licence-specific property, which is coded in the covert code together with other licence properties.

The first implementation of the licensing subsystem capable

of independent licence sharing will not scan every server profile for available licences. It will only scan its own server profile and the directory server's profile. Therefore, all licenses that are meant to be shared, should be installed for the directory server.

If the tickets to be shared are limited in number, the situation is more complicated. For each "pool" of shareable tickets, there must be a single process that is responsible for keeping track of their usage. It has to co-ordinate the activities of the licensing subsystems in various servers and make sure that no ticket is issued more than once. To achieve this a site licensing service can be implemented. This is an extension to the licensing subsystem that allows the licensing subsystems of various servers to communicate using a client-server protocol. The site licensing service, together with the licensing subsystem in the same server, control the usage of tickets installed for that server. Another server's licensing subsystem may connect to the site licensing service and ask the latter to obtain a service ticket on its behalf.

Licenses that are shareable by independent sharing would also be shareable by the site licensing service, with the addition that also limited-number tickets could be shared. Some types of licences will still be unshareable, since shareability is a licence-specific property. The licensing service could itself require a licence. A site licensing service could be expanded to support also client licensing and enterprise-wide licence sharing.

An example of independent licence sharing will now be described with reference to Figure 3 in which an application server 21 provides an application service 22 and includes a licensing subsystem 23 with an associated ticket database 24. A directory server 25 provides a directory service 26 and includes a licensing subsystem 27, with a associated ticket

database 28, and a directory (not shown but containing information of the type illustrated in Figures 1 and 2). The ticket databases 24 and 28 have details of ticket type, limit and usage as indicated.

The licensing system 27 on the directory server 25 fetches the server profile from the directory (not shown), analyses the licences therein, and updates the ticket limits (step 1). The licensing system 23 on the application server 21 fetches the application server's server profile from the directory (not shown) using the directory API. It also fetches the directory server's server profile (step 2).

The Application server's licensing subsystem 23 analyses the licences in the server's own profile. In this case there are none, since the example is concerned with licence sharing. The licensing subsystem 23 then analyses the directory server's licences. Because there are unlimited session and usage tickets in a shareable licence, the local limit is also set to unlimited. The user ticket limit is set to 0, because they are limited (10 according to ticket database 28) and limited tickets cannot be shared with this method (step 3).

The application service 22 then requests an application session ticket from its licensing subsystem 23 (step 4). The ticket is granted because there are an unlimited supply of them. The used ticket count is updated in the local ticket database 24 (step 5), although it is only needed for statistics as the number is unlimited. The session ticket is then issued to the application service 22, which then proceeds since it has determined that there exists an appropriate licence.

License sharing in the case of a site licensing service will now be described with reference to Figure 4, in which an application server 31 provides an application service 32 and includes a licensing subsystem 33 with an associated ticket

database 34. A license server 35 provides a site licensing service 36 and includes a licensing subsystem 37 with an associated ticket database 38.

The licensing subsystems 33 and 27 of the servers 31 and 35, fetch their corresponding server profiles from a directory (not shown), analyse installed licences and store the ticket limits in the local databases 34 and 38 (step 1). The application server 31 need not have any licences.

The application service 32 requests a service ticket, for example an application session ticket, from the local licensing subsystem 33 (step 2). The local licensing subsystem 33 in the application server 31 will first attempt to issue the ticket locally, but this will fail as there are no licences installed for the application server 31, as indicated by the lack of available tickets in the ticket database 34 (step 3). The licensing subsystem 33 in the application server 31 will then connect to the site licensing service 36 using the client-server protocol and request the ticket remotely (step 4). The site licensing service 36 requests the ticket from the local licensing subsystem 37 and it also request a ticket-forwarding ticket (step 5). The licensing subsystem 37 of the license server 35 checks ticket availability and updates the used ticket counts in the ticket database 38 (step 6). The tickets are issued to the site licensing service 36 (step 7) which forwards the application ticket to the client ie licensing subsystem 33 (step 8), which as a result issues the application ticket to the application service 32, allowing that to proceed (step 9).

Whenever a licensing subsystem issues a service ticket, or a ticket is returned such as because it is an unused ticket (any number can be requested) or because it is a session ticket, which are required to be returned at the end of a session, the transaction can, optionally, be logged to a log file which is separate from other log files in the system.

The information in this separate log file may be used to implement a pay-by-usage licensing scheme (delayed billing). Logging can be enabled or disabled by an administrator. Each server has its own log file and all kinds of tickets issued in the server will be logged the same way. Logging parameters for each kind of ticket could be specified for certain types of licences, although such a licence could not be shared by the independent sharing method.

The proposed licensing method allows for introducing new services while retaining compatibility with old licences. The licensing subsystems will initially support some types of licences and service tickets that are not yet connected to any particular service. New services can be assigned to these items without making any modifications to existing administration programs and the licensing subsystem. The method could be extended further by adding new license/ticket combinations to the licensing subsystem, although all existing combinations would need to be kept unchanged. This would involve updating the licensing subsystem in all servers where the new services would be used. Older subsystems would not accept the new kind of licenses not issue tickets for the new services. The licenses and tickets could be defined statically, as they are now, although there could be other possibilities.

As discussed above, the covert code specifies the licence duration, licence size and included services. An example of a covert code comprises a 7-digit decimal number, with the digits numbered from right to left, starting from zero eg in number 6543210, digit no 0 is "0", digit no 1 is "1" etc.

Licence duration may be encoded in the last digit ie digit 0, as follows:

Digit No 0	Licence Duration
"0"	10 days
"1"	1 month (31 days)
"2"	3 months (92 days)
"3"	6 months (184 days)
"4"	1 year (366 days)
"5"	2 years
"6"	3 years
"7"	Unlimited (small size)
"8"	Unlimited (medium size)
"9"	Unlimited (large size)

Licence size may be coded in the next-to-last digit, digit no 1. However, its interpretation may depend on the licence duration. Limited duration licences may be one of, for example, 30 different sizes; duration digits "7", "8" or "9" select small, medium or large licence sizes respectively.

Digit No 1	Licence size for each duration type			
	Limited	Unlimited Small	Unlimited Medium	Unlimited Large
"0"	1	1	60	400
"1"	2	2	80	500
"2"	5	5	100	600
"3"	10	10	125	800
"4"	15	15	150	1000
"5"	20	20	175	1200
"6"	30	25	200	1500
"7"	50	30	225	2000
"8"	100	40	250	3000
"9"	Unlimited	50	300	Unlimited

The services that are included in a licence may be encoded into four digits, digits no 2 to no 5, of the covert

code. These digits are called the service code. The licence may apply to one kind of service tickets only, to a group of related service tickets that are used by one service, or to a group of selected services. The service code can be chosen to represent a particular name of service, such as "basic directory service", "basic mail service", "basic calendar service", in any desired manner but it will indicate what types of tickets are included and how many licence service tickets are included for each type of service.

The digit no 6, the most significant digit, may be used to specify a particular product line. In the examples shown in the drawings the covert codes all commence with the number 2, indicating they relate to the same product line.

Any number of licences may be installed in the server profile of any server. The activation key is verified, and the covert code calculated from the serial number and the activation key at license installation time. The mapping of covert code to service ticket is, preferably, not stored in the directory, rather it is recalculated by a licensing subsystem every time it starts up. All tickets of the same type are indistinguishable. The licensing subsystems do not keep track of individual tickets issued.

Any number of identical tickets may be obtained at once by a service from the corresponding licensing subsystem, providing of course that they are available. Tickets can be returned if they are not used.

The licensing subsystem does not force services to obtain tickets rather it is the service's responsibility to offer services only to legal users and without obtaining a respective ticket, a service which requires a licence will not function.

Session tickets are associated with client-server sessions.

Unless a service wants to allow unlimited usage, it should obtain a session ticket whenever a session starts. Determining when each session starts and ends is the responsibility of the service. Session tickets may not be applicable to all services. It is important that session tickets are returned when the sessions end, otherwise they will be unusable, at least until the licencing subsystem is resynchronised. This is achieved at server start up, when there are no sessions in existence, by setting the used session ticket count to zero.

When a user is given the right to use a service, the associated user ticket should be obtained first. Because in a currently preferred embodiment, users are created and user rights given by the directory service, the licenses that include user tickets should be installed into the directory server. The directory service is the only service that requests user tickets and it is responsible for maintaining consistency of the used ticket counts. It periodically counts all users in the directory and their user rights and sets the number of tickets in use as appropriate.

Some kinds of tickets are "consumable" e.g. for sending mail messages, and these will not be returned unless, for example, the message is cancelled.

Clearly if an originally purchased licence becomes inadequate, due for example to an increased number of users, then supplemental licences can be purchased which when installed will increase the number of available tickets for a service. Additional functionality can of course also be purchased subsequently, in order to add new features to a system, and the appropriate software and licence installed in an appropriate server.

It is considered that with the above description of the licence management system and method proposed by the

invention, a software developer will have difficulty producing the corresponding code for licence management for a particular software product written in a particular language, and hence no further description is considered necessary in this respect.

CLAIMS

1. A software licence management method for use with a computer system including at least one server, the method being such that before a service can offer functionality to a user, the said service shall verify that the user has a licence for said service, and wherein the computer system further includes a licensing subsystem with which are associated service tickets corresponding to existing licences, the method including the steps of the said service requesting a respective service ticket from the licensing subsystem prior to offering functionality to the user, and the licensing subsystem issuing a said service ticket to the said service, if one is available, thereby verifying the licence exists and allowing the said service to offer functionality.

2. A method as claimed in Claim 1, including the step of installing licence information comprising a licence serial number and a licence activation key into the computer system, the activation key containing encoded details of the licensed services, and wherein the computer system calculates, from the serial number and the activation key, information including the types of service tickets associated with a particular licence, the numbers of service tickets, and the duration of the licence.

3. A method as claimed in Claim 2 wherein the licensing subsystem maintains a log of the numbers of the maximum available and issued service tickets.

4. A method as claimed in Claim 2 or Claim 3, wherein a covert code is calculated by the computer system from the serial number and activation key and wherein mapping of the covert code to service tickets is calculated by

the licencing subsystem each time it is started.

5. A method as claimed in any one of the preceding claims wherein the computer system comprises a plurality of computers connected in a network and wherein a said server comprises a directory server, providing a directory service and including a respective licencing subsystem, together with a directory database and a ticket database, wherein stored in the directory database are directory server profile details, licence details and user profile details, and wherein the ticket database includes details of service tickets available in accordance with the respective licence details and issued, and wherein adding a user to the computer system includes the steps of starting the directory server licensing subsystem, the directory server licensing subsystem fetching the directory server profile with licence details from the directory database and updating the ticket database, the requesting of a user service ticket by the directory service from the licensing subsystem, the checking of ticket availability in the ticket database by the licensing subsystem, the issuing of a ticket by the licensing subsystem to the directory service, and the adding to the directory database of the new user's profile by the directory service.

6. A method as claimed in any one of Claims 1 to 4 wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another said server comprises a directory server providing a directory service and with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and licence details, and wherein the ticket

database includes details of service tickets available in accordance with the licence details and issued, and wherein obtaining a use ticket for the application service includes the steps of starting the application server licensing subsystem, the subsystem fetching the application server profile and licence details from the directory database and updating the ticket database accordingly, starting the application service without providing funtionality, the requesting by the application service of a user service ticket from the licensing subsystem, the checking of ticket availability in the ticket database by the licensing subsystem, and the issuing of a service ticket to the application service by the licensing subsystem, the application service then providing its functionality to a user.

7. A method as claimed in any one of Claims 1 to 4 and for independent licence sharing, wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another said server comprises a directory server providing a directory service and including a respective licensing subsystem with a respective ticket base and with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and shareable licence details, the number of service tickets being unlimited, wherein the directory server ticket database includes details of service tickets available in accordance with the shareable licence details and issued, and wherein the application server ticket database includes details of service tickets issued, and wherein obtaining a service ticket for the application service includes the steps of the directory server licensing system fetching the server profile from the

directory database, analysing the shareable licence details and updating the corresponding ticket types and ticket limits in the directory server ticket database, the application server licensing subsystem fetching the application server and the directory server profiles and shareable licence details from the directory database and analysing them and updating the corresponding ticket types in the application server ticket database, starting the application service without providing functionality, the requesting by the application service of a service ticket from the application server licensing system, the granting of a service ticket, and the issuing of the service ticket to the application service by the application server licensing system, the application service then providing its functionality to a user.

8. A method as claimed in any one of Claims 1 to 4 and for licence sharing with site licensing, wherein the computer system comprises a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, another said server comprises a site licensing server providing a site licensing service and including a respective licensing subsystem with a respective ticket database, and a further said server comprises a directory server providing a directory service and having a directory database, wherein stored in the directory database are directory server profile details, site licensing server profile details, application server profile details and licence details, wherein the site licensing subsystem ticket database includes details of service tickets available in accordance with the licence details and issued, and wherein obtaining a service ticket for the application service when the application server has no

respective licence includes the steps of the licensing subsystems fetching their corresponding server profiles from the directory database, analysing the installed licence details and the site licensing server updating the respective ticket database, starting the application service without providing functionality, the requesting by the application service of a service ticket from the site licensing service, the requesting of a service ticket and a ticket-forwarding ticket by the site licensing service from its licensing subsystem, the checking of ticket availability and the issuing of the service and ticket-forwarding tickets to the site licensing service, the forwarding of the service ticket to the application server licensing subsystem, and the issuing of the service ticket to the application service, the application service then providing its functionality to a user.

9. A computer system including at least one server and a software licence management system, the management system being such that before a service can offer functionality to a user, the service shall verify that the user has a licence for said service, the management system including a licencing subsystem with which are associated service tickets corresponding to existing licences, and the management system being such that a said service ticket is issued to a service, if one is available, upon request by the service, thereby verifying existence of a licence and allowing the said service to offer functionality.

10. A computer system as claimed in Claim 9, wherein the management system includes means for calculating from an input licence serial number and input licence activation key, information including the types of service tickets associated with a particular licence, the numbers of service tickets and the duration of the licence, said

information being encoded in the activation key.

11. A computer system as claimed in Claim 10, and including a log in which are stored the numbers of the maximum available and issued service tickets.

12. A computer system as claimed in Claim 9 or Claim 10, and wherein the calculating means include means for calculating a covert code from the serial number and activation key, and the licensing subsystem including means for mapping the covert code into service tickets each time the licensing subsystem is started.

13. A computer system as claimed in any one of Claims 9 to 12 and comprising a plurality of computers connected in a network, wherein a said server comprises a directory server, providing a directory service and including a respective licensing subsystem, together with a directory database and a ticket database, wherein stored in the directory database are directory server profile details, licence details and user profile details, and wherein the ticket database includes details of service tickets available in accordance with respective licence details and issued.

14. A computer system as claimed in any one of Claims 9 to 12 and comprising a plurality of computers connected in a network, wherein a said server comprises an application server providing an application service and including a respective licensing subsystem with a respective ticket database, and another server comprises a directory server providing a directory service with a respective directory database, wherein stored in the directory database are directory server profile details, application server profile details and licence details, and wherein the ticket database includes details of service tickets available in accordance with the licence

details and issued.

15. A computer system as claimed in Claim 14 and for independent licence sharing, wherein the directory server includes a respective directory licensing subsystem and a respective ticket database, shareable licence details, for which the number of service tickets available is unlimited, being stored in the directory database, the directory ticket database including details of service tickets available in accordance with the shareable licence details and issued, and the application server ticket database including details of service tickets issued.

16. A computer system as claimed in Claim 14 and for licence sharing with site licensing, and including another said server comprising a site licensing server providing a site licensing service and including a respective licensing subsystem with a respective ticket database, the directory database also including site licensing server profile details, and wherein the site licensing ticket database includes details of service tickets available in accordance with the licence detailed and issued.

17. A software licence management method substantially as herein described with reference to an as illustrated in Figure 1, Figure 2, Figure 3, or Figure 4, of the accompanying drawings.

18. A computer system including at least one server and a software licence management system substantially as herein described with reference to and as illustrated in Figure 1, or Figure 2, or Figure 3, or Figure 4 of the accompanying drawings.

The Patent Office

29

Application No:	GB 9617596.3	**Examiner:**	Mike Davis
Claims searched:	1-18	**Date of search:**	26 September 1996

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.O): G4A (AAP)

Int Cl (Ed.6): G06F

Other:

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	GB 2236604 A	(SUN MICROSYSTEMS)	1,9 at least
X	EP 0332304 A2	(DIGITAL EQUIPMENT)	•
X	WO 93/11480 A1	(INTERGRAPH)	•
X	US 5375206	(HUNTER ET AL)	•
X	US 4924378	(HERSHEY ET AL)	•

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :		(11) International Publication Number:
G06F 1/00	A1	WO 92/20022
		(43) International Publication Date: 12 November 1992 (12.11.92)

(21) International Application Number: PCT/US92/03812

(22) International Filing Date: 6 May 1992 (06.05.92)

(30) Priority data:

697,652	8 May 1991 (08.05.91)	US
723,456	28 June 1991 (28.06.91)	US
722,840	28 June 1991 (28.06.91)	US
723,457	28 June 1991 (28.06.91)	US

(71) Applicant: DIGITAL EQUIPMENT CORPORATION [US/US]; 146 Main Street, Maynard, MA 01754 (US).

(72) Inventor: WYMAN, Robert, Mark; 410 Second Avenue, South No. 108, Kirkland, WA 98033 (US).

(74) Agents: NATH, Ram, B. et al.; c/o Joyce D. Lange, Digital Equipment Corporation, 111 Powdermill Road, Maynard, MA 10754 (US).

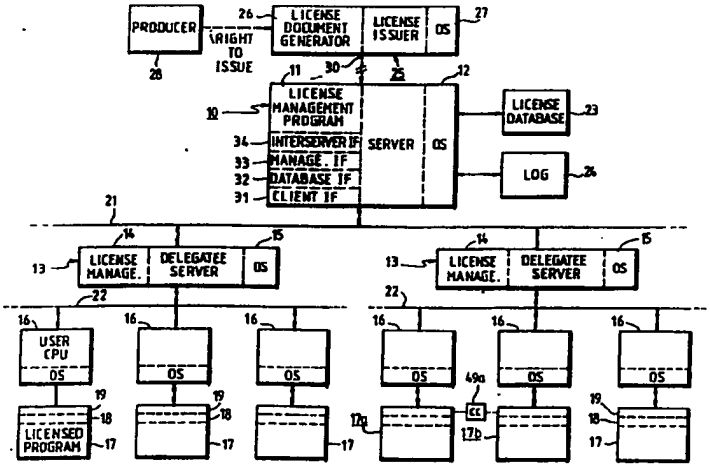
(81) Designated States: AT, AT (European patent), AU, BB, BE (European patent), BF (OAPI patent), BG, BJ (OAPI patent), BR, CA, CF (OAPI patent), CG (OAPI patent), CH, CH (European patent), CI (OAPI patent), CM (OAPI patent), CS, DE, DE (European patent), DK, DK (European patent), ES, ES (European patent), FI, FR (European patent), GA (OAPI patent), GB, GB (European patent), GN (OAPI patent), GR (European patent), HU, IT (European patent), JP, KP, KR, LK, LU, LU (European patent), MC (European patent), MG, ML (OAPI patent), MR (OAPI patent), MW, NL, NL (European patent), NO, PL, RO, RU, SD, SE, SE (European patent), SN (OAPI patent), TD (OAPI patent), TG (OAPI patent).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: MANAGEMENT INTERFACE AND FORMAT FOR LICENSE MANAGEMENT SYSTEM



(57) Abstract

A distributed computer system employs a license management system to account for software product usage. A management policy having a variety of alternative styles and contexts is provided. Each licensed product upon start-up makes a call to a license server to check on whether usage is permitted, and the license server checks a database of the licenses, called product use authorizations, that it administers. If the particular use requested is permitted, a grant is returned to the requesting user node. The product use authorization is structured to define a license management policy allowing a variety of license alternatives by values called "style", "context", "duration" and "usage requirements determination method". The license administration may be delegated by the license server to a subsection of the organization, by creating another license management facility duplicating the main facility. The license server must receive a license document (a product use authorization) from an issuer of licenses, where a license document generator is provided. A mechanism is provided for one user node to make a call to use a software product located on another user node; this is referred to as a "calling card", by which a user node obtains permission to make a procedure call to use a program on another node. A management interface allows a license manager at a server to modify the license documents in the database maintained by the server, within the restraints imposed by the license, to make delegations, assignments, etc. The license documents are maintained in a standard format referred to as a license document interchange format so the management system is portable and can be used by all adhering software vendors. A feature of the database management is the use of a filter function.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	ML	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NL	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		
ES	Spain	MG	Madagascar		

MANAGEMENT INTERFACE AND FORMAT FOR LICENSE MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

This invention relates to methods of operation of computer systems, and more particularly to a method and system for managing the licensing of software executed on computer systems.

In U.S. Patent 4,937,863, issued to Robert, Chase and Schafer and assigned to Digital Equipment Corporation, the assignee of this invention, a Software Licensing Management System is disclosed in which usage of licensed software may be monitored in a computer system to determine if a use is within the scope of a license. The system maintains a database of licenses for software products,

delivering the license document may be in the form of a network, or may be a phone line using modems, or may include physical delivery by disks or CD ROMs, for example. Likewise, the method of delivery of the software products being licensed, i.e., the applications programs 17 to be executed on the CPUs 16, is not material to the license management facility of the invention; the products are delivered by some appropriate means, e.g., the communications link 30 and the networks 21 and 22, by CD ROMs or disks physically distributed, etc.

Although shown in Figure 1 as operating on a distributed system, in the simplest case the license management facility of the invention may be operated on a single CPU. The license management program 11 and the applications program 17 may be executing on the same CPU, in which case the license document would be stored in a database 23 as before, on this CPU, and the calls from the unit 18 to the license server would be local instead of RPCs. As in the distributed system, however, the licensed product would still not have access to the license document, but instead could only make inquires to the server program, even if all are executing on the same CPU.

In operation of the distributed system of Figure 1, the producer 28 gives the issuer 25 authority to grant licenses on its behalf (the producer and issuer can be a single entity or multiple entities). The license document generator program 26, under control of a user (a person), generates a license (usually the result of negotiation between the user of program 26 and a user of the server 10). This license is called a product use authorization, and it is transmitted by the link 30 to the server 10. The license management program in the server 10 stores the product use authorization in the database 23, and, if delegation is an authorized option, may distribute parts of the authorized use to the delegatee servers 13,

where it is likewise stored in a database. Thereafter, administration of the license is only in response to inquiries from user nodes 16. When execution of a program 17 begins, the unit 18 is invoked to check on the availability of a license for this particular node. The unit 18 sends (as by an RPC) a request to the license management program 14 (or 11 if there is no delegatee), where the product use authorization stored in database 23 is checked to see if use is authorized. If so, a return is sent to the user node 16, granting permission to continue. When the program 17 has finished executing, the unit 18 again is invoked to signal to the license management program, again by an RPC, that the authorization is released, so the license management program can take appropriate action, e.g., log the use in log 24, etc.

To implement these operations, the license management program 11 or 14 contains several functions, including a client interface 31, a database interface 32, a management interface 33, and an interserver interface 34 for communicating with the delegatees 13 (if any). The client interface 31, as described below, handles the requests received from the user nodes 16, and returns resulting from these requests. The database interface 32 handles the storing and retrieval of license information in the database 23, and logging license usage activity to log 24, and retrieval of this data. The management interface 33 handles the tasks of receiving the product use authorizations from the issuer 25 and maintaining the database 23 via the database interface 32. The interserver interface 34 handles the task of communicating with the delegatee servers 13, including transmitting the assigned parts of the product use authorizations, or communicating with other license servers that may be separately executing the license management function; for example, calls for validating calling cards may be made to another such server.

If there are no delegatees or no other license servers, then of course the interserver interface 34 has no function, and is idle.

The license document or "product use authorization" forming the basis for the license management activity of the program 11 on the server 10 may be illustrated as a data structure containing the information set forth in Figure 2; in actual practice the product use authorization is preferably a more abstract data arrangement, not in such a rigidly structured format as illustrated. For example, the product use authorization as well as similar documents stored in the database 23, or passed between components of the system of Figure 1, may be of the so-called tag-length-value data format, where the data structure begins with an identifying tag (e.g., PUA or product use authorization) followed by a field giving the length, followed by the value itself (the content). One type of data treatment using this tag-length-value format is an international standard referred to as ASN.1 or Abstract Syntax Notation. In any event, the document 35 illustrated in Figure 2 is merely for discussing the various items of data, rather than representing the way the information is stored. Some of the fields shown here exist at some times and not others, and some are optional; the product use authorization may also include additional fields not shown or discussed here. Also it should be noted that copies of parts of this type of document are made for the delegatees, so this representation of Figure 2 is a composite of several documents used in the system of Figure 1. The document 35 includes fields 36 identifying the software product by product name, producer, version numbers, release date, etc. The issuer 25 is identified in field 37, and the licensee (usually the owner of the license server 10) identified in field 38. The essential terms of the license grant are then defined in fields 40-46. The start date and end date are specified in fields 40; these store the exact time (date, hour, minute, second, etc.) when the license becomes valid and

when it ends, so licenses may be granted to start at some future time and to end at a particular time. Note that the previous practice has been to specify only the ending date, rather than also a start date as employed here. Each of the nodes, including issuer 25, servers 10 and 13, and user nodes 16, maintain a time value by a local clock referenced to a standard, so inherent in the license management facility is the maintaining of a time standard to compare with the start and end date information in the fields 40. The units granted are specified in field 41; the units are an arbitrary quantitative measure of program usage. In a delegatee server 13, the units field 41 will have some subset of the units field in the original product use authorization. As units are granted to users 16 or delegated, the remaining units available for grant are indicated in a subfield 42 in the copy of the document used by the server. The management policy occupies fields 43-46, and includes style, context, duration and LURDM (license use requirements determination method), as will be explained. The style field 43 specifies whether the licensed units are controlled by an "allocative" style or "consumptive" style, or some other "private" algorithm, where styles are ways used to account for the consumption or allocation of the units. The context field 44 specifies the location and environment in which product use or license management occurs, i.e., a CPU or an individual user or a network, etc. Duration field 45 indicates whether the license granted to a user is by assignment, by transaction, or immediate. The LURDM field 46 indicates the license use requirements determination method, in some cases using a license use requirements table (LURT) seen as field 47, as will be described.

Additional fields 48-54 in the product use authorization 35 of Figure 2 define features such as delegation authorization, calling authorization, overdraft

authorization, combination authorization, token, signature, checksum, etc. These will be described in the following paragraphs.

If the delegation field 48 is true, a license server 10 may distribute license units to multiple servers 13. A time limit may be imposed, i.e., units can be delegated to other hardware systems until they time out. Delegation allows an administrator to distribute units to improve response time and increase the resilience of the system. For example, the communication network 21 may include a satellite link to a remote facility where the local server 13 has a number of clients or users 16, in which case the calls to the server 13 would be completed much quicker than would be the case if calls had to be made to the server 10. Also, delegation may be used as a method of allocating licensed units within a budget for administrative purposes. Usually the delegation authorization is a feature that is priced by the issuer, i.e., a license granting 1000 units with delegation authorization is priced higher than without this authorization.

The field 49 contains a calling authorization and/or a caller authorization. If the caller authorization in field 49 is true, the product is permitted to receive calls from other named products requesting use of the product, and if conditions are met (identified caller is authorized) the server can grant a calling card, as described below. If the calling authorization is true, the product can make calls to other products. If neither is true, then the product can neither make or receive calls using the calling card feature. Referring to Figure 1, if product 17a wishes to make a remote procedure call to a feature of product 17b running on a different user node 16, it makes a call to its server 13 including a request for a calling card, and, if permitted, the return to product 17a includes a calling card 49a. The product 17a then makes a call to product 17b in the usual manner of

RPCs, sending along the calling card 49a, which the product 17b then verifies by a call to its server 13 before executing the called procedure and issuing its return to product 17a. The feature of calling cards is important for distributed applications. For example, if a product is able to execute faster in a distributed system by assigning tasks to other CPUs, then the issue is presented of which license policy is needed, i.e., does every node executing a part of the task have to be licensed and consume or receive allocation of a unit, or just the one managing the task? This is resolved for most applications by use of this calling card concept. The product use authorization for such a product has the calling authorization field 49 enabled, so calling cards can be issued. This feature is typically separately priced.

The combination authorization field 50 of Figure 2 determines whether or not license requests from a user node 16 can be satisfied by combining units from multiple product use authorizations. It may be advantageous to purchase licenses with different policy values, and use units from certain product use authorizations only for overflow or the like. Or, for other reasons, it may be advantageous to "borrow" and "lend" units among delegated servers or user nodes. This function is permitted or denied by the content of field 50.

The overdraft field 51 determines whether or not a requested allocation from a user node 16 will be nevertheless granted, even though the units available field 42 is zero or too small to permit the requested use. Overdrafts can be unlimited, or a specific overdraft pool can be set up by a server 10, for a customer's internal administrative purposes. That is, the overdraft value may be unlimited in the original license, but limited or zero for internally distributed copies of the license. Thus, the product use authorization sent by the issuer 25 to

the customer may have overdrafts permitted by the field 51, but the customer may deny overdraft permission for its own budgeting purposes. In any event, if overdraft is permitted, additional fees have to be paid to the issuer at some accounting period, when the logged usage from log 24 indicates the available units have been exceeded. If overdraft is denied, then the units 18 of the user nodes making request allocations are structured to inform the products 17 that a license grant is not available. The intent is not to prevent the application program from running; the license server merely informs the application whether or not the license manager determines that it is authorized to run. The application can itself be structured to shut itself down if not authorized to run, or it can be structured to shut down certain functions (e.g., ability to save files, ability to print, etc.), or it can be structured to continue in a fully functional manner. The purpose of the license management facility is not that of enforcement, nor that of "copy protection", but instead is merely that of license management.

An optional token field 52 is available in the product use authorization 35 of Figure 2. This field can contain comments or other information desired by the issuer or user. For example, a telephone support number may be included in the token field, then when the product 17 shows its "help screen" the number is inserted. This number would be part of the argument, i.e., data transmitted to the user node 16, when the server 10 makes a return following a request allocation message from the user. This field may also be used to store information used in a "private" style, where the information from this field returned to the user node is employed by the application program 17 or the stub 19 to determine if the application can be activated.

The signature field 53 in the product use authorization 35 is a part of a validation mechanism which provides important features. This field contains a digital signature encoded to reflect the data in the license itself, as well as other encoding methods not known to customers, so it cannot be duplicated unless the encoding algorithm is known. In a preferred embodiment, a so-called "public/private key" system of encoding is used for the signature field 53. The encoding algorithm used to generate the signature 53 is known to the issuer 25, using a private key, and anyone knowing the public key can decode the signature to determine if it is valid but cannot determine the encoding algorithm so it cannot produce a forged signature. So, if the server 10 knows the public key which is unique to the issuer 25, it can determine if a license document 35 is genuine, but it cannot itself generate license documents. However, if the server possesses a valid license document that gives it the right to delegate, then it will be assigned its own private key (different from all other issuers or servers) and its delegatees 13 will be able to determine if a valid delegated license is delivered to them as they will be given the public key for the servers 13. The field 53 will thus contain both the original signature from the issuer 25 and the license server's signature when delivered to a delegatee 13. The decoding algorithm using a public key for any signatures is thus used by the license server 10 or delegatee 13 to make sure a product use authorization 35 is authentic before it is stored in the database 23. Related to the digital signature 53 is a checksum field 54, which merely encodes a value related by some known algorithm to the data in the product use authorization 35 itself. This field may be used merely to check for corruption of the data as it is stored, recalled, and transmitted within the system. That is, the checksum is used for data validation rather than security.

Two concepts central to the license management system implemented using the license document or product use authorization 35 of Figure 2 are the "license units", specified in field 41 or 42 and the "context", specified in field 44. License units are an abstract numerical measure of product use allowed by the license. When a product 17 (or a function or feature of a product) makes a license-checking request, the license management program 11 on server 10 computes how many license units are required to authorize this particular use of the product, and this is the license units requirement, in some cases using the LURDM field 46. A "context" is a set of tagged values which define the location and environment in which product use or license management occurs. Context values may be specified in field 44 of the product use authorization 35 of Figure 2 to restrict the environments in which the license may be managed and in which product use may occur. A context template may also be specified in the field 44 to indicate which parts of the complete context of product use (sub-contexts) are significant in differentiating product uses for the purposes of unit allocation; when this is specified, it allows separate product uses to share license units in a controlled way.

The two general types of policies specified in field 43 are allocative and consumptive. An allocative policy grants to the holder a specific number of license units (field 41) and specifies the policy which must be used to account for the allocation of these units. A software product 17 which is being managed by an allocative license will require verification that the appropriate number of license units have been allocated to it prior to performing services to the user. Typically, this allocation of units occurs either at the time of activation of the product 17 or at the time that product use is enabled on a particular platform (user CPU 16). The units typically remain allocated to the product 17 throughout the period that the product is running or is enabled to run. Upon termination of

processing or disabling, the allocated units are deallocated and made available for allocation to other instances of the software product 17 (other users 16 activating the product). In general, as long as any license units remain unallocated in field 42, the holder of the license is contractually authorized to increase his utilization of the licensed product. The usage does not deplete the license, however, as the units are returned to the units-available field 42 after a user is finished, and can be granted again to another user.

A consumptive unit based license, indicated in policy field 43, grants to the holder a specific number of initial license units (from field 42) and specifies the policy used to account for the consumption of those units. A software product 17 which is being managed by a consumptive license will cause an appropriate number of license units to be consumed to reflect the services provided by the product. Once consumed, units cannot be reused. Thus, the number of units available for future use declines upon every use of the licensed software product 17. This may also be referred to as a "metered" policy, being conceptually similar to measured consumption of electricity, water, etc. When the number of available units in field 42 reaches zero, the license may require that further use of the product is prohibited, or, the agreement may permit continued decrementing of the number of available units; the result is the accumulation of a negative number of available units in the field 42. It is anticipated that most consumptive unit based licenses will consider negative units to represent an obligation of the license holder to pay the license issuer 25. The transaction log 24 maintains an audit trail for providing a record of the units used in a consumptive license.

Referring to Figure 3, the major elements of the management policy are set forth in a table, where the possible entries for the fields 43, 44, 45 and 46 are

listed. For the style entry 43, the possibilities are allocative and consumptive as just described, plus a category called "private" which represents a style of management undefined at present but instead to be created especially for a given product, using its own unique algorithm. It is expected that most licenses may be administered using the named alternatives of Figure 3, but to allow for future expansion to include alternatives not presently envisioned, or to permit special circumstances for unique software, the "private" choices are included, which merely mean that the product 17 will generate its own conditions of use. It is important to note that, except for the "private" alternative, the license management is totally in control of the license management program 11 on the license server 10 (or delegatee 13), rather than at the product 17. All the product 17 does, via the unit 18, is to make the request inquiry to the server 10 via the client interface 31, and report when finished.

The context field 44 specifies those components (sub-contexts) of the execution-context name which should be used in determining if unit allocations are required. License data is always used or allocated within, or for the benefit of, some named licensing context, and context can include "platform contexts" and "application contexts". Platform contexts are such things as a specific network, an execution domain, a login domain, a node, a process ID or a process family, a user name, a product name, an operating system, a specific hardware platform, as listed in Figure 3. Applications contexts are information supplied from the application (the product 17), such as may be used in a "private" method of determining license availability. The context name can use several of these, in which case the context name is constructed by concatenating the values of all subcontexts into a single context name, e.g., a VAX 3100 platform using VMS operating system.

The duration field 45 defines the duration of an allocation of license units to a specific context or the duration of the period which defines a valid consumptive use. For durations of type "Assignment," the specification of a reassignment constraint is also provided for, as discussed below. There are three types of duration, these being "transaction," "assignment" and "immediate" as seen in Figure 3.

The transaction duration type, when specified for an allocative policy, indicates that license units should be allocated to the specified context upon receipt of a license request and that those units should be deallocated and returned to the pool of available units upon receipt of a corresponding license release from a user node 16. Abnormal termination of the process or context having made the original license request will be semantically equivalent to a license release. On the other hand, when specified for a consumptive policy, this duration type indicates that license units should be allocated to the specified context upon receipt of a license request and permanently removed from the available units pool (field 42) upon receipt of a license release which reflects successful completion of the transaction. Upon receipt of a license release which carries an error status or upon abnormal termination of the processor context having made the original license request, the allocated units will be deallocated and returned to the pool of available units (field 42).

The assignment duration type in Figure 3 (field 45 of Figure 2) imposes the constraint that the required units must have been previously assigned to a specific context. The sub-contexts which must be specified in the assignment are those given in the context-template. A "reassignment constraint" may be imposed, and this is a limitation on how soon a reassignment can be made. For example, a

reassignment constraint of 30-days would require that units assigned to a specific context could not be reassigned more often than every 30-days; this would prevent skirting the intent of the license by merely reassigning units whenever a user of another context made a request allocation call for the product. Related to this assignment constraint, a "reallocation limit" may also be imposed, to state the minimum duration of an allocation; where there is a context template of process, the intent is to count the number of uses of the software product at a given time, but where software runs in batch rather than interactive mode it may run very quickly on a powerful machine, so a very few concurrent uses may permit almost unlimited usage - by imposing a reallocation constraint of some time period, this manner of skirting the intent of the license may be constrained.

The immediate duration type (field 45 of Figure 2) is used to indicate that the allocation or consumption of an appropriate number of license units from the pool of available units (field 42) should be performed immediately upon receipt of a license request. Receipt of license release or abnormal terminations will then have no impact on the license management system. When specified as the duration for an allocative policy, the effect will be simply to check if an appropriate number of license units are available at the time of a license request. When specified as the duration for a consumptive policy, the effect will be to deduct the appropriate number of license units from the available pool at the time of a license request, and, thereafter, abnormal termination, such as a fault at the user CPU 16 or failure of the network link, will not reinstate the units.

The LURDM or license unit requirement determination method, field 46, has the alternatives seen in Figure 3 and stores information used in calculating the number of units that should be allocated or consumed in response to a license

request. If this field specifies a table lookup kind, this means license unit requirements are to be determined by lookup in the LURT (field 47) which is associated with the current license. If a constant kind is specified, this indicates that the license units requirements are constant for all contexts on which the licensed product or product feature may run. A private LURDM specifies that the license unit requirements are to be determined by the licensed product 17, not by the license management facility 11. The license unit requirements tables (LURTs) provide a means by which issuers of licenses can store information describing the relation between context (or row selector) and unit requirements. The license units requirements determination method (LURDM) must specify "table lookup" for the LURT to be used, and if so a row selector must be specified, where a valid row selector is any subcontext, e.g., platform ID, user name, time of day, etc. An example of an LURT fragment is shown in Figure 4, illustrating the license unit requirements table mechanism. In this example, the row selector is "platform-ID" so the platform-ID value determines which row is used. The issuer of this LURT of Figure 4 has established three unit requirement tiers for use in determining the unit requirements for that issuer's products. The reason for the tiers is not mandated by the license management system, but the issuer 25 (actually the user of the program 26) would probably be establishing three pricing tiers, each reflecting a different perspective on the relative utility of different platforms in supporting the use of various classes of product 17. The first column in Figure 4, Column A, specifies the use requirements for a class of products whose utility is highly sensitive to the characteristics of the specific platform on which they are run. This can be seen by observing that the unit requirements are different for every row in Column A. Products which use the second column (Column B) appear to have a utility which is more related to the class of platform on which they run. This is indicated by the fact that all the PC

platforms share a single value which is different from that assigned to the VAX platform. The final column (Column C) is for use with a class of products which is only supported on the VAX platform. Figure 4 is of course merely an example, and the actual LURT created by the license document generator 26 and stored in the license database 23 (as field 47 of the product use authorization 35) can be of any content of this general format, as desired by the license issuer.

Instead of always selecting the rows in LURT tables according to the platform ID of the execution platform, in order to handle the breadth of business practices that need to be supported by the license management facility, the LURT mechanism is extended by providing a "row selector" attribute in the LURT class structure. No default is provided although it is expected that the normal value for the row selector attribute will be "platform ID."

In the system of patent 4,937,863, a concept similar to that of the LURT of Figure 4 was provided, with rows selected by the platform ID and columns selected by some arbitrary means, typically according to product type. The system of this invention allows flexibility in the selection of both LURT row and column while continuing to provide backwards compatibility for licenses defined within the constraints of patent 4,937,863.

Some examples will illustrate potential uses for the row selector attribute. A customer may only want to pay for the use of a product during one or two months of the year; the product may be FORTRAN and the reason for this request may be that the company has a fairly stable set of FORTRAN subroutines that are given regular "annual maintenance" only during the months of May and June. To handle this customer's needs, the FORTRAN product would generate

an application subcontext which would contain a value representing the month of the year. Then, a LURT table would be defined with twelve rows, one for each month of the year. In some column, probably column A, a negative one (-1) would be placed in each month except for May and June. These two months would contain some positive number. The product use authorization would then have a LURDM field specifying a LURT for use to determine the units requirement, and would name this custom LURT table. The effect would be that the PUA could only be used during the months of May and June since negative one is interpreted by license managers to mean "use not authorized." This mechanism could also be used to do "time of day" charging. Perhaps charging fewer units per use at night than during the day. Also, if a subcontext was used that contained a year value, a type of license would be provided that varied in its unit requirements as time passed. For instance, it might start by costing 10-units per use in 1991 but then cost one unit less every year as time passed, eventually getting to the point where the unit requirement was zero.

Another example is font names. A specific customer may purchase a license giving it the right to concurrent use of 100-units of a large font collection; some of the fonts may cost more to use than others. For instance, Times Roman might cost 10-units per use while New Century Schoolbook costs 20-units per use. The problem is, of course, making sure that charges are properly made. The solution is to build a LURT table with a specified application subcontext as its row selector. A row is then created for each font in the collection and in Column A of the LURT, the number of units required to pay for use of the font would be specified. The print server would then specify the name of a font as the value of the application subcontext whenever it does an *lm_request_allocation()* call. This will allow charges to be varied according to font name.

A further example is memory size. Some products are more or less valuable depending on the size of memory available to support them. A software vendor wishing to determine unit requirements based on memory size will be able to do so by building LURT tables with rows for each reasonable increment of memory (probably 1-megabyte increments). Their applications would then sense memory size (using some mechanism not part of the license management facility) and pass a rounded memory size value to the license manager in a private context.

Other examples are environment and operating system. Some products may be valued differently depending on whether they are being run in an interactive mode or in batch. This can be accomplished by building LURT rows for each of the standard platform subcontexts that specify environment. Regarding operating system, it has been considered desirable by many to have a single product use authorization permit the use of a product on any number of operating systems, this conflicts with some vendors policies who do not want to have to create a single price for a product that applies to all operating systems. Thus, if an operating system independent license were offered for a C compiler, the price would be the same on MS-DOS, VMS, and/or UNIX. Clearly, it can be argued that the value of many products is, in part, dependent on the operating system that supports them. By using a row selector of operating system (one of the standard platform subcontexts), license designers could, in fact, require different numbers of units for each operating system. However, it might be more desirable to base the row selection on a private application subcontext that normally had the same value as the operating system subcontext. The reason for this is that the license designer might want to provide a default value for operating system names that were unknown at the time the LURT rows were defined. If this is the case, the product would contain a list of known operating systems and

pass the subcontext value of "Unknown" when appropriate. The LURT row for "Unknown" would either contain a negative one (-1) to indicate that this operating system was unsupported or it would contain some default unit requirement.

Another example is variable pricing within a group. One of the problems with a "group" license is that there is only one unit requirements field on the PUA for a group. Thus, all members of the group share a single unit requirement. However, in those cases were all members of the group can be appropriately licensed with a constant unit requirement yet it is desired to charge different amounts for the use of each group member, a LURT can be built that has rows defined for each group member. The row selector for such a group would be the standard platform subcontext "product name."

Many different types of license can be created using different combinations of contexts, duration and policy from the table of Figure 3. As examples, the following paragraphs show some traditional licensing styles which can be implemented using the appropriate values of the product use authorization fields 43-46.

A "system license" as it is traditionally designated is a license which allows unlimited use of a product on a single hardware system. The correct number of units must be allocated to the processor in advance and then an unlimited product use is available to users of the system. The product use authorization would have in the context field 44 a context template for a node name, the duration field would be "assignment" and the policy style field 43 would be "allocative".

A "concurrent use" license is one that limits the number of simultaneous uses of a licensed product. Concurrent use license units are only allocated when the product is being used and each simultaneous user of the licensed product requires their own units. In this case the context template, field 44, is a process ID, the duration field is "transaction" and the policy style 43 is "allocative".

A "personal use" license is one that limits the number of named users of a licensed product. This style of licensing guarantees the members of a list of users access to a product. Associated with a personal use type of product use authorization there is a list of registered users. The administrator is able to assign these users as required up to the limit imposed by the product use authorization; the number of units assigned to each user is indicated by the LURDM. It may be a constant or it may vary as specified in a LURT. The context template is "user name", the duration is "assignment", and the policy is "allocative".

A "site license" is one that limits the use of a licensed product to a physical site. Here the product use authorization contains for the context template either "network name" or "domain name", the duration is "assignment" and the policy style field 43 is "allocative".

Generally, a license to use a software product is priced according to how much benefit can be gained from using the product, which is related to the capacity of the machine it will run on. A license for unlimited use on a large platform such as a mainframe, where there could be thousands of potential users at terminals, would be priced at a high level. Here the style would be "allocative", the context template = "node", the duration = "assignment" and the LURDM may be "Column A" - the units, however, would be large, e.g., 1000. At the other end

of the scale would be a license for use on a single personal computer, where the field values would be the same as for the mainframe except the units would be "1". If a customer wanted to make the product available on the mainframe but yet limit the cost, he could perhaps get a license that would allow only five users at any given time to use the product; here the fields in the product use authorization would be: units = 5; style = allocative; context template = process; duration = transaction; LURDM = constant, 1-unit. This would still be priced fairly high since a large number of users may actually use the product if a session of use was short. A lower price would probably be available for a personal use license where only five named persons could use the product, these being identified only in the license server 10, not named by the license issuer 25. Here the fields in the product use authorization are: units = 5; style = allocative; context template = user name; duration = transaction; LURDM = constant, 1-unit.

An additional feature that may be provided for in the product use authorization 35 is license combination. Where there are multiple authorizations for a product, license checking requests sent by user nodes 16 may be satisfied by combining units from multiple authorizations. Individual product use authorizations may prohibit combined use. Thus, a licensee may have a license to use a product 17 on an allocative basis for a certain number of units and on a consumptive basis for another number of units (this may be attractive from pricing standpoint); there might not be enough units available for a particular context from one of these licenses, so some units may be "borrowed" from the other license (product use authorization), in which case a combination is made.

The interface between the program executing on the client or user 16 and the license server 10 or its delegatees 13 includes basically three procedure calls:

a request allocation, a release allocation and a query allocation. Figure 5 illustrates in flow chart form some of the events occurring in this client interface. The request allocation is the basic license checking function, a procedure call invoked when a software product 17 is being instantiated, functioning to request an allocation of license units, with the return being a grant or refusal to grant. Note that a product may use request allocation calls at a number of points in executing a program, rather than only upon start-up; for example, a request allocation may be sent when making use of some particular feature such a special graphics package or the like. The release allocation call is invoked when the user no longer needs the allocation, e.g., the task is finished, and this return is often merely an acknowledge; if the style is consumptive, the caller has the opportunity via the release allocation call to influence the number of units consumed, e.g., decrease the number due to some event. The query allocation call is invoked by the user to obtain information about an existing allocation, or to obtain a calling card, as will be described.

The request allocation, referred to as *lm_request_allocation()*, is a request that license units be allocated to the current context. This function returns a grant or denial status that can be used by the application programmer to decide whether to permit use of the product or product feature. The status is based on the existence of an appropriate product use authorization and any license management policies which may be associated with that product use authorization. License units will be allocated or consumed, if available, according to the policy statement found on the appropriate product use authorization. The product would normally call this function before use of a licensed product or product feature. The function will not cause the product's execution to be terminated should the request fail. The decision of what to do in case of failure to obtain allocation of license

units is up to the programmer. The arguments in a request allocation call are the product name, producer name, version, release date, and request extension. The product name, producer name, version and release date are the name of the software product, name of producer, version number and release date for specifically identifying the product which the user is requesting an allocation be made. The request extension argument is an object describing extended attributes of the request, such as units required, LURT column, private context, and comment. The results sent back to the calling node are a return code, indicating whether the function succeeded and, if not, why not, and a grant handle, returned if the function completes successfully, giving an identifying handle for this grant so it can be referred to in a subsequent release allocation call or query allocation call, for example.

The release allocation, referred to as *lm_release_allocation()*, is an indication from a user to the license manager to release or consume units previously allocated. This function releases an allocation grant made in response to a prior call to request allocation. Upon release, the license management style 38 determines whether the units should be returned to the pool of available units or consumed. If the caller had specified a request extension on the earlier call to request allocation which contained a units-required-attribute, and the number of units requested at that time are not the number of units that should be consumed for the completed operation, the caller should state with the units-consumed argument how many units should be consumed. The arguments of the release allocation are: grant handle, units consumed, and comment. The grant handle identifies the allocation grant created by a previous call to request allocation. The units-consumed argument identifies the number of units which should be consumed if the license policy is consumptive; this argument should only be used

in combination with an earlier call to request allocation which specified a units requirement in a request extension. Omission of this argument indicates that the number of units to be consumed is the same as the number allocated previously. The comment argument is a comment which will be written to the log file 24 if release units are from a consumptive style license or if logging is enabled. The result is a return code indicating if the function succeeded, and, if not, why not.

The query allocation, or *lm_query_allocation()*, is used by licensed products which have received allocations by a previous request allocation call. The query is to obtain information from the server 10 or delegatee server 13 about the nature of the grant that has been made to the user and the license data used in making the grant, or to obtain a calling card (i.e., a request that a calling card be issued). Typically, the item read by this query function is the token field 52 which contains arbitrary information encoded by the license issuer and which may be interpreted as required by the stub 19 for the licensed product software 17, usually when a "private" allocation style or context is being employed. The arguments in this procedure call are the grant handle, and the subject. The grant handle identifies the allocation grant created by a previous call to request allocation. The subject argument is either "product use authorization" or "calling card request"; if the former then the result will contain a public copy of the product use authorization. If this argument is a calling card request and a calling card which matches the previous constraints specified in that request can be made available, the result will contain a calling card. If the subject argument is omitted, the result will contain an instance of the allocation. The results of the query allocation call are (1) a return code, indicating whether the function succeeded, and, if not, why not, and (2) a result, which is either an allocation, a product use authorization or a calling card, depending on type and presence of the subject argument.

Referring to Figure 5, the flow chart shows the actions at the client in its interface with the server. When the software product 17 is to be invoked, the unit 18 is first executed as indicated by the block 60, and the first action is to make a request allocation call via the stub 19, indicated by the block 61. The client waits for a return, indicated by the loop 62, and when a return is received it is checked to see if it is a grant, at decision block 63. If not, the error code in the return is checked at block 64, and if a return code indicates a retry is possible, block 65, control passes back to the beginning, but if no retry is to be made then execution is terminated. If the policy is to allow use of the product 17 without a license grant, this function is separately accounted for. If the decision point 63 indicates a grant was made, the grant handle is stored, block 66, for later reference. The program 17 is then entered for the main activities intended by the user. During this execution of product 17, or before or after, a query allocation call can be made, block 67, though this is optional and in most cases not needed. When execution of the program 17 is completed, the grant handle is retrieved, block 68, and a release allocation call is made, block 69. A loop 70 indicates waiting for the return from the server, and when the return received it is checked for an error code as before, and a retry may be appropriate. If the release is successfully acknowledged, the program exits.

Referring to Figure 6, the actions of the server 10 or delegatee server 13 in executing the license management program 11 or 14, for the client interface, are illustrated in flow diagram form. A loop is shown where the server program is checking for receipt of a request, release or query call from its clients. The call would be a remote procedure call as discussed above, and would be a message communicated by a network, for example. This loop shows the decision blocks 71, 72 and 73. If a release allocation call is received, a list of products for which

authorizations are stored is scanned, block 74, and compared to the product identity given in the argument of the received call, block 75. If there is no match, an error code is returned to the client, block 76, and control goes back to the initial loop. If the product is found, the authorization is retrieved from the database 23, block 77 (there may be more than one authorization for a given product, in which case all would be retrieved, but only one will be referred to here) and all of the information is matched and the calculations made depending upon the management policy of Figures 3 and 4, indicated by the decision block 78. If a grant can be made, it is returned as indicated at block 79, or if not an error code is returned, block 80. If a release allocation call is received, indicated by a positive at the decision block 72, the grant handle in the argument is checked for validity at block 81. If no match is found, an error code is returned, block 82, and control passes back to the initial loop. If the handle is valid, the authorization for this product is retrieved from the database 23 at block 83, and updated as indicated by the block 84. For example, if the license management style is allocative, the units are returned to the available pool. Or, in some cases, no update is needed. The authorization is stored again in the database, block 85, and a return made to the client, block 86, before control passes back to the initial loop. If the decision block 73 indicates that a query allocation call is received, again the grant handle is checked at block 87, and an error code returned at block 88 if not valid. If the grant handle matches, the authorization is retrieved from the database 23, at block 89, and a return is made to the client giving the requested information in the argument, block 90.

The basic allocation algorithm used in the embodiment of the license management system herein described, and implemented in the method of Figures 5 and 6, is very simple and can handle a very large proportion of known license

unit allocation problems. However, it should be recognized that a more elaborate and expanded algorithm could be incorporated. Additions could be made in efforts to extend the allocation algorithm so that it would have specific support for optimizing unit allocation in a wider variety of situations. Particularly, sources of non-optimal allocations occurring when using the basic allocation algorithm are those that arise from combination and reservation handling.

The first step is formation of full context. The client stub 19 is responsible for collecting all specified platform and application subcontexts from the execution environment of the product 17 and forwarding these collected subcontexts to the license management server 13 or 10. The collection of subcontexts is referred to as the "full context" for a particular license unit allocation request.

The next step is retrieval of the context template. When the license manager receives an *lm_request_allocation()*, it will look in its list of available product use authorizations (PUA) to determine if any of them conform to the product identifier provided in the *lm_request_allocation()* call. The product identifier is composed of: product name, producer, version, release date. If any match is found, the license manager will extract from the matching PUA the context template. This template is composed of a list of subcontexts that are relevant to the process of determining unit requirements. Thus, a context template may indicate that the node-ID subcontext of a specific full context is of interest for the purposes of unit allocation. The context template would not specify any specific value for the node-ID; rather, it simply says that node-ID should be used in making the allocation computation.

The next step is masking the full context. Having retrieved the context template, the license manager will then construct an "allocation context" by filtering the full context to remove all subcontexts which are not listed in the context template. This allocation context is the context to be used in determining allocation requirements.

Then follows the step of determining if the request is new. The license manager maintains for each product use authorization a dynamic table which includes the allocation contexts of all outstanding allocations for that PUA (i.e., allocations that have been granted but have not yet been released). Associated with each entry in this table is some bookkeeping information which records the number of units allocated, the full context, etc. To determine if a recent *lm_request_allocation()* requires an allocation of units to be made, the license manager compares the new allocation context with all those allocation contexts in the table of outstanding allocations and determines if an allocation has already been made to the allocation context. If the new allocation context does not already exist in the table, an attempt will be made to allocate the appropriate number of units depending on the values contained in the LURDM structure of the PUA and any LURTs that might be required. If an allocation context similar to that specified in the new allocation request does exist in the table, the license manager will verify that the number of units previously allocated are equal to or greater than the number of units which would need to be allocated to satisfy the new allocation request. If so, the license manager will return a grant handle to the application which indicates that the allocation has been made (i.e., it is a "shared allocation" - the allocated units are shared between two requests.) If not, the license manager will attempt to allocate a number of units equal to the

difference between the number previously allocated and the number of units required.

The step of releasing allocations (Fig. 6, blocks 84-85) occurs when the license manager receives an *lm_release_allocation()* call; it will remove the record in its dynamic allocation table that corresponds to the allocation to be released. Having done this, the license manager will then determine if the allocation to be removed is being shared by any other allocation context. If so, the units associated with the allocation being released will not be released. They will remain allocated to the remaining allocation contexts. Some of the units might be released if the license manager determines that the number of allocated units exceeds the number needed to satisfy the outstanding allocation contexts. If this is the case, the license manager will "trim" the number of allocated units to an appropriate level.

In summary, the two things that make this algorithm work are (1) the basic rule that no more than one allocation will be made to any single allocation context, and (2) the use of the context template to make otherwise dissimilar full contexts appear to be similar for the purposes of allocation.

The license designer's task, when defining basic policy, is then to determine which contexts should appear to be the same to the license manager. If the license designer decides that all contexts on a single node should look the same (context template = node-ID), then any requests that come from that node will all share allocations. On the other hand, a decision that all contexts should be unique (i.e., context template = process-ID) will mean that allocations are never shared.

and stores a unit value indicating the number of licensing units for each product. When a user wishes to use a licensed product, a message is sent to the central license management facility requesting a license grant. In response to this message, the facility accesses the database to see if a license exists for this product, and, if so, whether units may be allocated to the user, depending upon the user's characteristics, such as the configuration of the platform (CPU) which will execute the software product. If the license management facility determines that a license can be granted, it sends a message to the user giving permission to proceed with activation of the product. If not, the message denies permission.

While the concepts disclosed in the patent 4,937,863 are widely applicable, and indeed are employed in the present invention, there are additional functions and alternatives that are needed in some applications. For example, the license management system should allow for simultaneous use of a wide variety of different licensing alternatives, instead of being rigidly structured to permit only one or only a few. When negotiating licenses with users, vendors should have available a wide variety of terms and conditions, even though a given vendor may decide to narrow the selection down to a small number. For example, a software product may be licensed to a single individual for use on a single CPU, or to an organization for use by anyone on a network, or for use by any users at terminals in a cluster, or only for calls from another specific licensed product, or any of a large number of other alternatives. A vendor may have a large number of products, some sold under one type of license and some under others, or a product may be a composite of a number of features from one or more vendors having different license policies and prices; it would be preferable to use the same license management system for all such products.

Distributed computing systems present additional licensing issues. A distributed system includes a number of processor nodes tied together in a network of servers and clients. Each node is a processor which may execute programs locally, and may also execute programs or features (subparts of programs) via the network. A program executing on one node may make remote procedure calls to procedures or programs on other nodes. In this case, some provision need be made for defining a license permitting a program to be executed in a distributed manner rather than separately on a single CPU, short of granting a license for execution on all nodes of a network.

In a large organization such as a company or government agency having various departments and divisions, geographically dispersed, a software license policy is difficult to administer and enforce, and also likely to be more costly, if individual licenses are negotiated, granted and administered by the units of the organization. A preferred arrangement would be to obtain a single license from the software producer, and then split this license into locally-administered parts by delegation. The delays caused by network communication can thus be minimized, as well as budgetary constraints imposed on the divisions or departments. Aside from this issue of delegation, the license management facility may best be operated on a network, where the licensing of products run on all nodes of the network may be centrally administered. A network is not necessary for use of the features of the invention however, since the license management can be implemented on a single platform.

Software products are increasingly fragmented into specific functions, and separate distribution of the functions can be unduly expensive. For example, a spreadsheet program may have separate modules for advanced color graphics, for

accessing a database, for printing or displaying an expanded list of fonts, etc. Customers of the basic spreadsheet product may want some, none or all of these added features. Yet, it would be advantageous to distribute the entire combination as one package, then allow the customer to license the features separately, in various combinations, or under differing terms. The customer may have an entire department of the company needing to use the spreadsheet every day, but only a few people who need to use the graphics a few days a month. It is advantageous, therefore, to provide alternatives for varied licensing of parts or features of software packages, rather than a fixed policy for the whole package.

Another example of distribution of products in their entirety, but licensing in parts, would be that of delivering CD ROMs to a customer containing all of the software that is available for a system, then licensing only those parts the customer needs or wishes to pay fees for rights to use. Of course, the product need not be merely applications programs, operating systems, or traditional executable code, but instead could also include static objects such as printer fonts, for example, or graphics images, or even music or other sound effects.

As will be explained below, calling and caller authorizations are provided in the system according to one feature of the invention, in order to provide technological support for a number of business practices and solve technical problems which require the use of what is called "transitive licensing." By "transitive licensing" is meant that the right to use one product or feature implies a right to use one or more other products or features. Transitive licenses are similar to group licenses in that both types of license consist of a single instrument providing rights of use for a plurality of products. However, transitive licenses differ from group licenses in that they restrict the granted rights by specifying that

the licensed products can only be used together and by further specifying one or more permitted inter-product calling/caller relationships. Some examples may help to clarify the use and nature of a transitive license: the examples to be explained are (1) two products sold together, (2) a give-away that results from narrow choices of licensing alternatives, (3) a client licensing method in a client/server environment, (4) impact of modular design, and (5) the impact of distributed design.

A software vendor might have two products for sale: the first a mail system, and the second a LEXIS™-like content-based text retrieval system. Each of these products might be valued at $500 if purchased separately. Some customers would be satisfied by purchasing the rights to use only one of these products. others might find that they can justify use of both. In order to increase the likelihood that customers will, in fact, purchase both products, it would not be surprising if the software vendor offered his potential customers a volume discount, offering the two products for a combined price of $800. The customers who took advantage of this combined offer would find that they had received two products, each of which could be exploited to its fullest capabilities independently from the other. Thus, these customers would be able to use the content based retrieval system to store and retrieve non-mail documents. However, from time to time, the vendor may discover that particularly heavy users of mail wish to be able to use the content based retrieval system only to augment the filing capabilities provided by the standard mail offering. It is likely that many of these potential customers would feel that $800 is simply too much to pay for an extended mail capability. The vendor might then consider offering these customers a license that grants mail users the right to use the content-based retrieval system only when they are using mail and prohibits the use of content

based retrieval with any other application that might be available on the customers system. This type of license is referred to below a "transitive license," and it might sell for $600.

Another example is a relational database product (such as that referred to as Rdb™) designed for use on a particular operating system, e.g., VMS. This relational database product has two components: (1) A user interface used in developing new databases, and (2) a "run-time" system which supports the use of previously developed databases. The developers of the database product might spend quite a bit of effort trying to get other products made by the vendor of the database product to use it as a database instead of having those other products build their own product-specific databases. Unfortunately, the other product designers may complain that the cost of a run-time license for the database product, when added to the cost of licenses for their products, would inevitably make their products uncompetitive. Thus, some mechanism would be needed that would allow one or another of the vendor's products to use the run-time system for the relational database product in a "private" manner while not giving unlicensed access to products of other vendors. No such mechanism existed, prior to this invention; thus, the vendor might be forced to sell the right to use its run-time system for the database product with its proprietary operating system license. Clearly, this combined license would make it possible for the vendor's products to use its database product without increasing their prices; however, it also would make it possible for any customers and third-parties to use the database product without paying additional license fees. However, had the system of the invention been available, the vendor could have granted transitive licenses for the run-time component of its database product to all the vendor's products. Essentially, these licenses would have said that the database run-time could be used without an

additional license fee if and only if it was used in conjunction with some other of the vendor's products. Any customer wishing to build a new relational database application or use a third-party application that relied on the vendor's database product would have had to pay the vendor for its database run-time license.

A proposed client/server licensing method provides yet another example of a problem which could be solved by transitive licensing. Typically, a client is only used by one user at a time, while a server can support an arbitrary number of clients depending on the level of client activity and the capacity of the machine which is supporting the server. While traditionally, server/client applications have been licensed according to the number of clients that a server could potentially support, this may not be the most appropriate method for licensing when the alternatives afforded by the invention are considered. The business model for the proposed client/server method requires that each client be individually licensed and no explicit licensing of servers is required to support properly licensed clients. Such a licensing scheme makes it possible to charge customers only for the specific number of clients they purchase. Additionally, it means that a single client can make use of more than one server without increasing the total cost of the system. The solution to this transitive licensing problem would be to provide a mechanism that would allow the clients to obtain license unit allocations and then pass a "proof" of that allocation to any servers they may wish to use. Servers would then support any clients whose proofs could be verified to be valid. On the other hand, if a client that had not received a proof of allocation attempted to use a server, the server would obtain a license allocation for that client session prior to performing any services. Such a solution has not been heretofore available.

As the complexity and size of the software systems provided to customers increases, it is found that the actual solution provided to customers is no longer a single product. Rather, customers are more often now offered solutions which are built up by integrating an increasing number of components or products, each of which can often stand alone or can be part of a large number of other solutions. In fact, a product strategy may rely almost exclusively on the vendor's engineering and selling a broad range of specialized components that can only be fully exploited when combined together with other components into a larger system. Such components include the relational database runtime system mentioned above, mail transport mechanisms, hyperinformation databases, document format conversion services, time services, etc. Because these components are not sold on their own merits, but rather on their ability to contribute to some larger system, it is unlikely that any one customer will be receiving the full abstract economic value of any one of the components once integrated into a system. Similarly, it can be observed that the value of any component once integrated into a larger system varies greatly from system to system. Thus, it may be found that a mail transport mechanism contributes a large part of a system whose primary focus is mail, however, it will contribute proportionally less of the value of a system that provides a broader office automation capability. As a result of these observations, the job of the business analyst who is attempting to find the "correct" market price for each component standing on its own, is more complex. In reality, the price or value of the component can only be determined when considering the contribution of that component to the full system or solution in which it is integrated. Attempting to sell the components at prices based on their abstract, independent values will simply result in overpriced systems.

Transitive license styles are particularly suited to dealing with pricing of modular components, since component prices can be clearly defined in relation to the other components or systems which they support. Thus, a vendor can charge a price of $100 for the right to use a mail transport system in conjunction with one product, yet charge $200 for the use of the same mail transport system when used by another product.

In addition to the "business" reasons for wanting to support transitive licensing, there is also a very good technical reason that arises from the growing tendency of developers to build "distributed products" as well as the drive toward application designs that exploit either tightly or loosely coupled multiprocessor systems; the availability and growing use of remote procedure calls has contributed to this tendency. This technical problem can be seen to arise when considering a product which has a number of components, each of which may run in a different process space and potentially on a different computer system. Thus, there might be a mail system whose user interface runs on one machine, its "file cabinet" is supported by a second machine and its mail transport system runs on yet a third machine. The simple question which arises is: "Which of the three components should check for licenses?" Clearly it must be ensured that no single component can be used if a valid license is not present. Thus, the answer to the question will probably be that all three components should check for licenses. However, the question is then presented: "Where are the licenses to be located?". This can become more complex.

Increasingly, the distributed systems being built are being designed so that it is difficult to predict on which precise machine any particular component will run. Ideally, networks are supposed to optimize the placement of functions

automatically so that the machine with the most available resource is always the one that services any particular request. This dynamic method of configuring the distribution of function servers on the network makes it very difficult for a system or network manager to predict which machines will run any particular function and thus very difficult for him to decide on which machines software licenses should be loaded.

Even if a system manager could predict which machines would be running the various application components and thus where the license units should be loaded, the situation would still be less than ideal. The problem arises from the fact that each of the components of the application would be independently making requests for license unit allocations. This behavior will result in a difficult problem for anyone trying to decide how many license units are required to support any one product. Given the mail example, the problem wouldn't exist if it were assumed that all three components (i.e., user interface, file cabinet, and transport system) were required by the design of the mail system to be in use simultaneously. If this were the case, it could be simply assumed that supporting a single activation of the mail system would require three units. However, in a real mail system, it will be inevitably discovered that many users will only be using just the user-interface and file-cabinet components of the system at one time. Thus, there will be some unused units available which could be used to authorize additional users. This situation might not be what is desired by the software vendor.

The problem of providing license support to multi-component products which are dynamically configured could be solved by viewing each of the product components as a distinct licensable product and by treating the problem as one

of transitive licensing, but a mechanism for accomplishing this has not been available. Essentially, a single license document would be created that stated that if any one of the components had successfully obtained a license to run, it could use this grant to give it the right to exploit the other components. Thus, in the example above, the user might start the mail system by invoking its user interface. This user interface code would then query the license management facility for a license allocation and once it has received that allocation, it would pass a proof of allocation to the other mail components that it uses. Each of the other components would request that the license management system validate that the "proof" is valid prior to performing any service; however, none of the other components would actually require specific allocations to be made to them. In this way, the complexity of licensing and managing networks of distributed applications can be significantly reduced.

SUMMARY OF THE INVENTION

In accordance with one embodiment of the invention, a license management system is used to account for software product usage in a computer system. The system employs a license management method which establishes a management policy having a variety of simultaneously-available alternative styles and contexts. A license server administers the license, and each licensed product upon start-up makes a call to the license server to check on whether usage is permitted, in a manner similar to that of patent 4,937,863. The license server maintains a store of the licenses, called product use authorizations, that it administers. Upon receiving a call from a user, the license server checks the product use authorization to determine if the particular use requested is

permitted, and, if so, returns a grant to the requesting user node. The license server maintains a database of product use authorizations for the licensed products, and accesses this database for updating and when a request is received from a user. While this license management system is perhaps of most utility on a distributed computer system using a local area network, it is also operable in a stand-alone or cluster type of system. In a distributed system, a license server executes on a server node and the products for which licenses are administered are on client nodes. However, the license management functions and the licensed products may be executing on the same processor in some embodiments.

The product use authorization is structured to define a license management policy allowing a variety of license alternatives by components called "style", "context", "duration" and "usage requirements determination method". The style may be allocative or consumptive. An allocative style means the units of the license may be allocated temporarily to a user when a request is received, then returned to the pool when the user is finished, so the units may be reused when another user makes a request. A consumptive style means the units are deducted from an available pool when a user node makes a valid request, and "consumed", not to be returned for reuse. The context value defines the context in which the use is to be allowed, such as on a particular network, by a particular type of CPU, by a particular user name, by a particular process, etc. The duration value (used in conjunction with the style component) concerns the time when the license units are to be deducted from the available pool of units, whether at the time of request, after a use is completed, etc. A usage requirements determination method may be specified to define or provide information concerning the number of license units charged in response to a license request from a user node; for example, some CPU platforms may be charged a larger number of license units

than others. A table may be maintained of usage requirements, and the determination method may specify how to access the table, for example. The important point is that the user node (thus the software product) can only make a request, identifying itself by user, platform, process, etc., and the license management facility calculates whether or not the license can be granted (that is, units are available for allocation), without the user node having access to any of the license data or calculation. There is a central facility, the license server, storing the license documents, and, upon request, telling the licensed products whether they can operate under the license terms.

An important feature of one embodiment is that the license administration may be delegated to a subsection of the organization, by creating another license management facility duplicating the main facility. For example, some of the units granted in the product use authorization may be delegated to another server, where the user nodes serviced by this server make requests and receive grants.

The license management facility cannot create a license itself, but instead must receive a license document (a product use authorization) from an issuer of licenses. As part of the overall license management system of the invention, a license document generator is provided which creates the product use authorizations under authority of the owner of the software, as negotiated with customers. Thus, there are three distinct rights in the overall license management facility of the invention: (1) the right to issue licenses, (2) the right to manage licenses, and (3) the right to use the licensed products. Each one of these uses the license document only in prescribed ways. The license issuer can generate a license document. The license manager (or license server as referred to herein) can grant products the right to use under the license, and can delegate parts of the

licensed units for management by another server, as defined by the license document; the way of granting rights to products is by responding to certain defined calls from the products. And, the licensed products can make certain calls to the license server to obtain grants of rights based upon the license document, inquire, or report, but ordinarily cannot access the document itself.

As explained above, transitive licensing is an important feature of one embodiment. This is the provision of a mechanism for one user node to get permission to use another software product located on another user node; this is referred to as a calling authorization and a caller authorization, using a "calling card," and these are examples of the optional features which must be specifically permitted by the product use authorization. A user node must obtain permission to make a procedure call to use a program on another node; this permission is obtained by a request to the license server as before, and the permission takes the form of a calling card. When a calling card is received by a second node (i.e., when the procedure call is made), a request is made by the second node to the license server to verify (via the product use authorization) that the calling card is valid, and a grant sent to the user node if allowed. In this manner, all nodes may have use of a program by remote calls, but only one consumes license units.

Another important feature of one embodiment is a management interface which allows a license manager to modify the license policy components of a license document maintained by at a license server in its database. Usually the license manager can only make modifications that restrict the license policy components to be more restrictive than originally granted. Of course, the management interface is used to make delegations and assignments, if these are authorized.

The license document interchange format is an important feature, in that it allows the license management system to be used with a wide variety of software products from different vendors, so long as all follow the defined format. The format uses data structures that are defined by international standards.

An important function is the filter function, used in the management interface and also in the client interface to select among elements in the data structures.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof, will be best understood by reference to the detailed description of specific embodiments which follows, when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a diagram in block form of a distributed computer system which may be used to implement the license management operations according to one embodiment of the invention;

Figure 2 is a diagram of the content of a license document or "product use authorization" generated by the license document generator and stored by the license server in the system of Figure 1;

Figure 3 is a diagram of the alternatives for license style, context and duration making up the license management policy implemented in the system of Figure 1, according to one embodiment of the invention;

Figure 4 is a diagram of an example of a fragment of a license use requirements table (LURT) used in the system of Figure 1, according to one embodiment of the invention;

Figure 5 is a logic flow chart of a program executed by a user node (client), in the system of Figure 1, according to one embodiment of the invention;

Figure 6 is a logic flow chart of a program executed by a license server, in the system of Figure 1, according to one embodiment of the invention; and

Figure 7 is a diagram of the calls and returns made in an example of use of calling cards in the system of Figure 1.

Figure 8 is a diagram of an LDIF document identifier, according to an standard format;

Figure 9 is a syntax diagram of an LDIF document;

Figure 10 is a diagram of an LDIF document structure;

Figures 11, 13, 15, 17, 18, 19, 21-28 and 31-43 are syntax diagrams for elements of various ones of the LDIF data structures;

Figure 16 is a diagram of a license data structure;

Figures 12, 14 and 20 are examples of descriptions of data elements using a standard notation;

Figures 29 and 30 are examples of context templates used in the license management system;

Figures 44 and 45 are tables of attributes specific to filter and filter item type; and

Figure 46 is notation in a standard format for an example of a filter.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Referring to Figure 1, a license management facility according to one example embodiment of the invention is centered around a license server 10, which typically includes a CPU located in the customer's main office and executing a license management program 11 as will be described, under an operating system 12. The license server 10 communicates with a number of delegatees 13 which likewise include CPUs in departments or divisions of the company or organization, each also executing a license management program 14 under an operating system 15. The license management program 14 is the same as the program 11 executing on the main server 10; the only difference in the functions of server 10 and servers 13 is that the latter have a delegated subset of the license units granted to the server 10, as will be described. The CPUs 13 are in turn servers for a number of

users 16, which are CPU nodes where the licensed programs 17 are actually executed. The programs 17 executing on the user CPUs 16 are applications programs (or operating systems, etc.) which have added to them units 18 and 19, according to the invention, allowing them to make inquiry to the their server 13 (or 10) before executing and to report back after executing, using a client stub 19 in the manner of remote procedure calls, in one embodiment. A user node 16 may have many different programs 17 that may be executed, and the various user nodes 16 would usually each have a set of programs 17 different from the other user nodes, all of which would be administered by the license management program 14 or 11. The terms "program" and "licensed product" are used in reference to the element 17, but it is understood that the products being administered may be segments of programs, or functions or features called by another program, or even merely data (such as printer fonts), as well as complete stand-alone applications programs. The license server 10 communicates with the delegatee servers 13 by a network 21, as is usual in large organizations, and the delegatee servers 13 each communicate with their user nodes 16 by networks 22; these networks may be of the Ethernet, token ring, FDDI types or the like, or alternatively, the user nodes 16 may be merely a cluster of terminals on a multiuser system with the delegatee being a host CPU. The particular hardware construction of the user nodes, server nodes, communication networks, etc., and the operating systems 12 or 15, are of no concern regarding the utility of the features of the invention, the only important point being that the user CPUs 16 of the software products 17 in question can communicate readily and quickly with their respective server nodes 13 or 10. In one embodiment, remote procedure calls (RPCs) are used as the communication medium for the interfaces between components of the system, handling the inquiries and grants as will be described.

A remote procedure call is similar to a local procedure call but is made to a procedure located on a remote node, by way of a communications network.

The function of the unit 19 is that of a client stub, in a remote procedure call sense. The calls to the license server 10 are made through this stub 19, and returns are received by the stub 19 and passed on to the program 17. The stub 19 is responsible for obtaining the network addresses of other nodes on the network, such as the server 10. Also, the stub 19 is responsible for determining the context (as defined below) for passing on to the server 10. The unit 18 functions to execute a "private" type of license availability determination if this is used, rather than this task being done by the application program 17, but if the ordinary method of determination is employed (using the license server) as is usually the case, the unit 18 is merely code that starts the execution and passes calls and returns back and forth between the program 17 and the unit 19.

The license server 10, using the license management program 11, maintains a license data file 23 comprising a number of license documents or licenses (product use authorizations), and also maintains a log 24 which is a record of the usage activity of all of the user CPUs 16 of each of the licensed programs. The delegatee servers 13 would maintain similar license databases and logs. The license server 10 has no authority to originate a license, but instead must receive a license from a license issuer 25. The issuer 25 is again a CPU executing a license document generator program 26 under an operating system 27. The license issuer 25 may be under control of the producer 28 of the programs or software products being licensed, or may be controlled by a distributor who has received the authority to grant licenses from the producer or owner 28. The communications link 30 between the license issuer 25 and the license server 10 for

This mechanism permits the system of the invention to dispose of the cumbersome, explicit support of license types having different scope such as the cluster licenses, node licenses, and process licenses found in prior license management systems including that of patent 4,937,863. Instead of defining a limited set of scopes (cluster, node, etc.), the system of this invention provides a general mechanism which allows an effectively unlimited range of allocation scopes to be defined.

Transitive licensing, as referred to above, is supported by the system of the invention by (1) calling authorizations, which are statements made in field 49 of the product use authorization 35 for one product (the "caller") to permit that product to call another product (the "callee"), and, (2) caller authorizations, which are statements made in field 49 of the product use authorization for one product (the "callee") to permit it to be called by another product (the "caller").

If calling or caller authorizations are to be exploited by products, then whenever one product calls another product, it must pass the callee a calling card 49a. This calling card 49a is an encoding of an identification of the caller as well as a statement by the license management system that a license unit allocation has been made to the caller which is passing the calling card. This calling card is then passed by the callee to the license management system for validation and, if the either the product use authorization of the caller carries an appropriate calling authorization or the product use authorization of the callee carries an appropriate caller authorization, the use of the callee by the caller will be authorized without requiring any additional license unit allocations.

Referring to Figure 7, the intercomponent interactions that occur when either calling or caller authorizations are being used are illustrated. This figure shows a license management server 10, a caller product 17a named "Product-1" and a callee product 17b named "Product-2". When Product-1 starts to run, it will make an *lm_request_allocation()* call to the license management server 10 to obtain a grant handle for an allocation of some number of units of the Product-1 license. Either immediately, or at some later time, but always prior to making a call to Product-2, Product-1 will call *lm_query_allocation()*, passing the grant handle received earlier and specifying that it wants a calling card for the product named "Product-2." If the field 49 of the product use authorization 35 used to satisfy the grant represented by the grant handle carries a calling authorization in field 49 naming "Product-2," the license manager will create a calling card 49a which includes the statement that a calling authorization exists and pass this calling card back to Product-1. If the calling authorization does not exist, the calling card passed to Product-1 will contain a statement to that effect.

Once Product-1 has successfully obtained a calling card 49a from the license manager, it will then make a call to Product-2, passing the calling card along with any other initialization parameters that would normally be used when starting Product-2. Product-2 will then pass that calling card to the license manager as part of its *lm_request_allocation()* call and the license manager will determine if the calling card is valid. Note that calling cards become invalid once the process which received the calling card makes an *lm_release_allocation()* call or terminates abnormally. If the calling card is valid, and it indicates that a calling authorization is present, the license manager will verify this statement and if found to be true, will return a grant handle to Product-2. If, on the other hand, the calling card carries an indication that no calling authorization is present, the

license manager will attempt to find a product use authorization for Product-2 that contains a caller authorization naming Product-1 as an authorized caller. If the caller authorization is found, a grant handle will be passed back to Product-2. If not, the license manager will ignore the calling card and proceed with the normal *lm_request_allocation()* logic.

The requirement to be passing calling cards between products requires that both the caller and the callee be "aware" of the fact that calling and caller authorizations may be used. This is one of the few examples of a requirement for a product 17 to become actively involved in the licensing problem when using the licensing management system of the invention. However, since the use of calling/caller authorizations if a fairly "sophisticated" and powerful feature, it is considered acceptable to impose this burden on application coders.

MANAGEMENT INTERFACE

Referring to Figure 1, the license management program 11 executing on a server 10 includes a license management interface 33 which functions to allow a user at a console for the server 10 CPU or at a remote terminal to implement certain necessary operations. The management interface 33 is essentially the tools or mechanisms available to the license manager at the licensee's site to (a) load the various licenses received from issuers 25 into the database 23 and make them available for request allocation calls from the users, (b) remove the licenses from the machine when expired, (c) to make delegations if permitted, (d) to make assignments, (e) to make reservations, etc. Whatever the license manager is allowed to do to modify the license for his special circumstances (within the

original grant, of course), he does it by the mechanism of the management interface 33. Some licenses are not modified at all, but merely loaded. In a multiple machine environment, as on a network, there is considerable modification, as it is necessary to make sure the correct number of units are distributed onto the correct machines, the right people have access, other people don't have access, etc. Thus, in a network environment, there is extensive use of the management interface 33.

In reference to the terminology used in describing the management interface, as well as the license management system in general, it is helpful to note that the documentation conventions, data declarations, macro declarations, etc., for the object management used in one embodiment of the invention are according to the standards set forth in *OSI Object Management API Specification, Version 2.0*, X.400 API Association and X/Open Company Limited, 24 August 1990, a published document.

The specific operations available to the management interface 33 are to allow a manager to open and close a management session, register (load) objects in the license database 23, obtain a list of objects in the license database 23, and control a cursor (a cursor is a movable pointer to a member of a list of items). Once an object in the license database 23 is identified with the cursor, certain changes may be made in the object by a write function. For example, certain fields of a license document of Figure 2 or an LURT of Figure 4 may be changed in only specified ways as will be explained.

The operation of opening a session goes by the name of *lm_open_session()* and is used to establish a license management service session between a

management client and the service. Opening a session also creates a workspace to contain objects returned as a result of functions invoked within the session. Object management Objects can be created and manipulated within this workspace. Objects created within this workspace, and only such objects, may be used as Object arguments to the other license management service management functions used during the session established by a call to this function. More than one session may exist simultaneously.

The arguments that go with a *lm_open_session()* call are (a) the binding handle, which is binding information that defines one possible binding (a client-server relationship), and (b) a comment which will be inserted in the log file 24 if logging is enabled. The results from a *lm_open_session()* call are (a) a return code indicating whether the function succeeded, and, if not, why not, (b) a session, which is an established license management session between the management client and the license management service, and (c) a workspace that will contain all objects returned as a result of functions invoked in the session.

The close session call is referred to by *lm_close_session()* and functions to terminate the lm session. This function terminates the license service management session and makes the argument unavailable for use with other interface functions. The arguments that go with a *lm_close_session()* call are (a) the session which identifies the established lm session between the management client and the license management service, and (b) a comment which will be inserted in the log file if logging is enabled. The result of the call is a return code indicating whether the function succeeded, and, if not, why not.

The list function returns a set of selected objects in the license database 23, and uses the name *lm_list_licenses()*. This function is used to search the license database 23 and return a cursor which represents the first of one or more objects which match the specified filter. The specified filter will be applied to each object in the license database 23; all objects for which the filter evaluates true will be included in the object list accessible by the set_cursor function. The arguments that go with *lm_list_licenses()* are (a) session which identifies an established session between the management client and the license management service, and (b) a filter which is an object used to select license database 23 objects; license database objects will only be included in the object list headed by the cursor if they satisfy the filter - the constant no-filter may be used as the value of this argument if all license data objects are to be included in the object list. The results of the *lm_list_licenses()* call are (a) a return code indicating whether the function succeeded, and, if not, why not, and (b) a license list upon successful completion of this call containing a cursor which represents the first of one or more objects in the current license database 23 for which the specified filter evaluates true.

The register function is to register objects in the license database 23, and uses the name *lm_register()*. This function is used to register (i.e., load or create) new objects, or modify existing objects, in the license database 23; the objects which may be registered include only those which are subclasses of the license data class or history objects. The arguments are (a) session, which identifies an established session between the management client and the license management service, (b) license data object which is to be registered; if this argument is omitted, the comment argument is a required argument and a history object containing the comment will be registered in the license database 23, and (c)

comment, which will be inserted in the log file if logging is enabled. The result is a return code indicating whether the function succeeded, and, if not, why not. The errors possible when it does not succeed include data-expired, duplicate-object, no-such-session, memory-insufficient, network-error, etc., indicated by this return code.

The set cursor function establishes a new cursor, and is called by *lm_set_cursor()*. The arguments are (a) session, which identifies an established session between the management client and the license management service, (b) forward, which is a boolean value indicating if the direction in which the cursor is to be moved is forward or reverse, (c) filter which is used to eliminate cursors from the search for the next cursor that are not wanted; a new cursor will only be set if it satisfies the filter - the constant no-filter may be used as the value of this argument if any cursor is to be considered as the target cursor, and (d) the cursor which is to be used as the starting point in searching for the new cursor. The results are (a) a return code indicating whether the function succeeded, and, if not, why not, and (b) next-cursor, which is the requested cursor. The error codes in the return code may be end-of-list, not-a-cursor, etc.

After a session is opened, and an object such as a product use authorization or a LURT has been identified by the cursor, using the functions explained above, the management interface 33 is able to execute certain object management interface functions such as write or copy. By this mechanism, the management interface can modify certain limited attributes. None of these attributes can be modified in such a way that they reduce constraints established by corresponding attributes in the license data objects. The more important attributes which can be modified by the management interface 33 using this mechanism are:

(a) assignment: an assignment of some or all of the units granted on the associated product use authorization;

(b) reservation: a reservation of some or all of the units granted on the associated product use authorization;

(c) delegation: a delegation of the right to manage some or all of the units granted on the associated product use authorization, or if the associated license data is not a product use authorization, the delegation is of the right to use that license data;

(d) backup delegation: a statement of the right to manage some or all or the units granted on the associated product use authorization; this right is only active at times when the delegating server is not available;

(e) allocation: an allocation of units to a specific context;

(f) allocation period: the minimum duration of a single allocation - all allocated units cannot be allocated to a new context until a time period equal to the allocation period has passed since the units were last allocated;

(g) termination date: a date which is to override the value specified as the end date of the product use authorization 40 - this date must be earlier than specified;

(h) delegation permitted: an override of the delegation permitted flag of the associated license data;

(i) overdraft: the current overdraft level;

(j) overdraft logging: an override of the overdraft logging attribute of the associated product use authorization;

(k) comment: a comment created by the licensee;

(l) extended info: information not defined by the architecture which may be of use in managing the license data.

It will be noted that an assignment and a reservation are identical, the only difference being that a reservation is something optional, while an assignment is something that is required. If the duration is Assignment in the policy declaration of Figure 3, the license manager must assign some or all of the units before units can be allocated. Reservations, on the other hand, are made by the license manager using the management interface, regardless of the policy.

Thus, there are certain attributes that can be changed by a license administrator using the management interface at the server 10, but none of these can result in obtaining more extensive rights to use than granted by the product use authorization. In each case, the license administrator can limit the rights which will be allocated to users in some way that may be appropriate for the administrator for control purposes.

LICENSE DOCUMENT INTERCHANGE FORMAT

The major structural components of an ASN.1 encoded document which conforms to the specifications for the license management system discussed above will be described. The object identifier that is assigned to this data syntax, according to one embodiment, is that specified in ASN.1 as seen in Figure 8. The International Standards Organization or ISO, as it is referred to, defines how bit patterns are chosen to uniquely identify an object type, so the bit pattern set forth in Figure 8 would preceed each document used in the license management system so the document could be identified as being a document conforming to the prescribed License Document Interchange Format.

A document encoded according to this format is represented by a value of a complex data type called "license document interchange format document" of LDIFDocument, in this embodiment. A value of this data type represents a single document. This self-describing data structure is of the syntax defined in the international standard ASN.1 referred to above. The X/Open standard referred to above defines the conventions that must be used in employing this syntax, while the syntax itself is described in an OSI (Open Systems Interconnect, a standard administered by ISO) document identified as X.409 (referenced in the X/Open document identified herein).

The LDIFDocument data type consists of an ordered sequence of three elements: the document descriptor, the document header, and the document itself. Each of these elements are in turn composed of other elements. The overall structure of the LDIFDocument data type will be described, and the nature of the document descriptor and document header types. Then, the document content elements will be described in detail, as well as the various component data types used in the definition of the descriptor, the header and the content.

The LDIFDocument represents a single license document, with the syntax being shown in Figure 9 and the high-level structure of an LDIF document in graphical form being seen in Figure 10. The DocumentDescriptor of Figure 9 is a description of the document encoding, the DocumentHeader contains parameters and processing instructions that apply to the document as a whole, and the DocumentContent is the content of the document, all as explained below.

Referring to Figure 9, what this says is that an LDIFDocument is composed of (::= means "is composed of") a number of elements, the first thing in an

LDIFDocument is a bit pattern (tag) according to an international standard, indicating a certain type of document follows, which is indicated here to be "private" or vendor selected, the number 16373 in this case. Following the bit pattern which functions as a "starting delimiter" it is "implicit" that a "sequence" of elements must follow, where a sequence is distinguished from a set. A sequence is one or more of the elements to follow, whereas a set is exactly one of the elements to be listed. Implicit means that any file identified as LDIFDocument must have a sequence data type, rather than some other type. In the case of Figure 9, the sequence is document-descriptor, document header and document content; the document-content is mandatory, whereas the first two are optional. If an element in the sequence begins with a "0" it is a document-descriptor, "1" means a document-header, and "2" means it is a document-content. Again, it is implicit that the data following is of the format DocumentDescriptor, etc., in each case, and these are defined in Figure 11, Figure 13 and Figure 15.

Each file is in the tag-length-value format mentioned above, and also each element of a file containing multiple elements is of the tag-length-value format. The data stream could be examined beginning at any point, and its content determined by first looking for a tag, which will tell what data structure this is, then a length field will say how long it is, then the content will appear. These structures are nested within one another; a document containing several product-use-authorizations would be an LDIFDocument of the format of Figure 9, with a number of DocumentContent elements of Figure 15 following, with the length given for the LDIFDocument spanning the several PUAs, and the length given for each PUA being for the one PUA.

In Figure 11, the elements major-version and minor-version are seen to be "implicit integer". This means that because the element is of the type major-version, etc.. it must be an integer. Various other implicit types are given in other syntax diagrams, such as character-string, boolean, etc.

In Figure 15, the license body is identified as being of the type "choice" meaning it can be one of PUA, LURT, GroupDefinition, KeyRegistration, etc. Thus, knowing this is a license-body does not mean the data type of the object is known; it is a bit further where the kind of a license-body becomes known. The defintion of a license body is not implicit, but instead is a chioce type.

The contents of the various data elements will now be described in detail with reference to Figures 11-43. Using these detailed descriptions, the exact format of each of the elements used in the LDIF can be interpreted.

The license document descriptor or DocumentDescriptor consists of an ordered sequence of four elements which specify the version level of the LDIF encoding and identify the software that encoded the document, with the syntax being shown in Figure 11. An example of the way a product called PAKGEN V1.0 is expressed in the DocumentDescriptor encoding is shown in Figure 12. The fields in the DocumentDescriptor syntax are major-version, minor-version, encoder-identifier and encoder-name. The major-version field is the primary indicator of compatibility between LDIF processors and the encoding of the present document; this major-version field is updated if changes are made to the system encoding that are not backward compatible. The minor-version field is the revision number of the system encoding. The encoder-identifier field is a registered facility mnemonic representing the software that encoded the document;

the encoder-identifier can be an acronym or abbreviation for the encoder name - this identifier is constant across versions of the encoder. The encoder-identifier should be used as a prefix to Named Value Tags in Named Value Lists to identify the encoder of the named value. The encoder-name field is the name of the product that encoded the document; the encoder-name string must contain the version number of the product.

The document header or DocumentHeader contains data that pertains to the document as a whole, describing the document to processors that receive it; the syntax is shown in Figure 13. An example of a document header is shown in Figure 14, using the hypothetical product PAKGEN V1.0 of Figure 12. The private-header-data contains the global information about the document that is not currently standardized; all interpretations of this information are subject only to private agreements between parties concerned, so a processor which does not understand private header data may ignore that data. The Title field is the user-visible name of the document. The Author field is the name of the person or persons responsible for the information content of the document. The Version field is the character string used to distinguish this version of the document from all other versions. The Date filed is the date associated with this document. Note that the nature and significance of the Title, Author, Version, and Date fields can vary between processing systems.

The content of an LDIF document is represented by a value of a complex data type called DocumentContent. An element of this type contains one or more LicenseData content element using a syntax as shown in Figure 15. There are no restrictions on the number, ordering or context of LicenseData elements. The structure of a LicenseData element is represented in Figure 16. No restrictions

are made on the number, ordering, or context of LicenseData elements. The license-data-header field of Figure 16 specifies that data, common to all types of license data, which describes the parties to the licensing agreement, the term of the agreement, and any constraints that may have been placed on the management of the license data encoded in the license body. The license-body is an element that contains one content element, including: product use authorizations, license unit requirements tables, group definitions, key registrations, and various forms of delegations. The Management-Info is an element that contains information concerning the current state of the license data; this element is not encoded by Issuers.

The license data header, called LicenseDataHeader, is represented as a syntax diagram in Figure 17. The license-id field provides a potentially unique identification of the encoded license data, so issuers of license data can generate unique license-ids to distinguish each issuance of license data; however, the architecture does not require this to be the case, since the only architectural restriction is that no two objects in any single license management domain may have the same value for license-id. The licensee field identifies the party who has received the rights reflected in the license data; there are at least two parties involved in all transfers of license data, first, the issuer of the license data, and second, the licensee or recipient of that data - it is anticipated that individual licensees will specify to those issuing them licenses what the licensee fields on their license data should contain. the term field identifies the term during which the license data may be used; the validity of license data can be limited by issuers to specific time ranges with given starting and ending dates, which are carried in the term element - attempts to use license data or products described by that data either before the start date or after the end date will result in conforming license

managers denying access to the license. Management-constraints identifies constraints placed on the right to manage the associated license data; these constraints can include (a) limiting the set of contexts permitted to manage the data, (b) limiting the set of platforms which may benefit from that management, and (c) limiting the right to backup and delegate the managed data. The signature provides the digital signature used by the issuer to sign the license data and identifies the algorithm used in encoding the signature. Issuer-comment is a comment provided by the issuer and associated with the license data.

The IssuerComment is of an informational nature and does not impact the process of authorizing product or feature use. This field is not included in the fields used to generate the signature for a license, thus, even if specified by an issuer, the IssuerComment can be omitted from a license without invalidating the license. If specified, the IssuerComment should be stored in the appropriate license data base with the associated license data. The IssuerComment can be retrieved by products which use the system and may be of particular utility to products in the "Software Asset Management" domain which are intended to extend or augment the administrative or accounting facilities or basic system components. Some examples of potential uses for this field are order information, additional terms and conditions, and support information. For order information, some issuers may wish to include with their loadable license data some indication of the purchase order or orders which caused the license data to be issued; licensees may find it useful to include this data in their license databases to assist in the license management process. For additional terms and conditions, the system will never provide automatic means for the management of all possible license terms and conditions, and so some issuers may wish to include summaries of non-system managed terms and conditions in the comment as a reminder. For

support information, the IssuerComment could be used to record the phone numbers or addresses of the responsible individuals within the issuing organization who should be contacted if there are problems with the data as issued.

A product use authorization as previously discussed in reference to Figure 2 is used to express the issuance of a right to use some product, product feature, or members of some product group. As such, it records the identity of the product for which use is authorized and specifies the means that will be used by the license manager to ensure that the licensee's actual use conforms to the terms and conditions of the license. Figure 18 illustrates a syntax diagram for a ProductUseAuthorization. Product-id identifies the name of the producer of the product or product feature of which usage rights are being granted as well as the name of that product; in addition, issuers of product use authorizations may specify a range of versions and/or releases whose use is controlled by the specific product use authorization. Units-granted - Contains the number of units of product use which are granted by the license. Management-policy defines the policy which is to be used in managing the granted software usage rights; this definition specifies the Style, Context-Template, Duration, and License Unit Requirements Determination Method which must be used. The calling-authorizations and caller-authorizations are as explained above in reference to calling cards. The execution-constraints field identifies constraints placed on the characteristics of execution contexts which may be authorized to benefit from the units granted by this Product Use Authorization. The product-token filed contains product specific data not interpreted in any way by any processors conformant with the architecture; software product producers 28 use this array to augment the capabilities of conformant license managers.

Some anticipated uses of the token field include language support, detailed feature authorizations, and product support number. For language support, a token could be constructed which contains a list of local language interface versions whose use is authorized; thus, if a product were available in English, German, French and Spanish, a token could be constructed listing only English and German as the authorized languages. For detailed feature authorizations, some license issuers will wish to have very fine control over the use of features in a complex product; however, they may not wish to issue a large number of individual Product Use Authorizations to "turn on" each feature, so these vendors could construct tokens which contain lists of the features authorized or whose use is denied. For product support number, some issuers may wish to include on the product use authorization, and thus make available to the running product, some information concerning the support procedures for the product; for example, an issuer might include the telephone number of the support center or a support contract number, and the product could be designed to retrieve this data from the license manager and display it as part of Help dialogues.

The LURT's or license use requirements tables of Figure 4 provide a means by which issuers of licenses, whose LURDM is dependent on the type of platform on which the product is run, can store information describing the relationship between the platform type and unit requirements. A syntax diagram for a LURT is shown in Figure 19. In Figure 20, an example of how the LURT of Figure 4 might be encoded is illustrated. Lurt-name specifies the name by which the LURT is to be known to conforming license managers. The rows field models a list of multicolumn lurt rows. Platform-id identifies the platform for which this LurtRow provides license unit requirements. The lurt-columns field provides a list of one or more lurt column values; the first value provided is

assigned to column-1 of the lurt-row, the second value provided is assigned to column-, etc. A lurt column value of -1 indicates that use of the product or feature is not authorized, while a lurt column value of 0 or greater indicates the number of units that must be allocated in order to authorize product use on the platform described by this lurt-row. All unspecified columns (e.g., columns whose number is greater than the number of column values provided in the lurt columns element) will be considered to contain the value -1.

In reference to Figure 19, to use the row-selector feature mentioned above, the platform-ID element would be replaced with *row-selector* which would be implicit of Context. Also, in Figure 34 described below, in the lurdm-kind element, *row-selector* would be included if the row-select feature is to be used.

As discussed above, Figure 4 provides an example of a hypothetical LURT, illustrating the LURT mechanism, where the issuer of this LURT table has established three unit requirement tiers for use in determining the unit requirements for that issuer's products. Figure 20 provides an example of how the LURT presented in Figure 4 might be encoded.

A group definition is used to define and name a license group. Once so defined, the name of this group can be used on product use authorizations in the same manner as a product name. Since a single product use authorization specifies the management policy for all members of the group, the members of that group must be compatible in their licensing styles, i.e., a personal use type product can not be mixed with a concurrent use product in the same group. Figure 21 shows a group definition syntax diagram. Group-name is the name which must appear on Product Use Authorizations for this group. Group-version

specifies the current version of this group; the requirements for matching between the version information on a product use authorization and that on a specified group definition are the same as those rules which require matching between produce use authorizations and the Release Date data provided by products. Group-members lists those products or features which are components of the named group.

A key registration is used by a producer 28 or issuer 25 who have been registered as authorized license issuers and provided with an appropriate public and private key pair. The key registration identifies the public key which is to be used by conforming license managers 10 in evaluating signatures 53 created by the named issuer 25 or producer 28. A key registration syntax diagram is shown in Figure 22. Key-owner-name provides the name which must be used in either of, or both, of the Producer and Issuer fields of license data generated by the issuer; the key-owner-name must be identical to that specified in the Issuer field of the header record. Key-algorithm identifies the registered algorithm that is to be used when producing digital signatures with this key. Key-value identifies the public key.

An issuer delegation is typically issued by a producer 28 and authorizes the named issuer 25 to issue licenses for products produced by the producer. An issuer delegation syntax diagram is shown in Figure 23. Delegated-issuer-name identifies the name which must appear in the Issuer field of any Product Use Authorization generated using the License Issuer Delegation. Delegated-product-id identifies the products whose licenses the named issuer is authorized to issue. Delegated-units-granted, if specified, indicates that the use of this IssuerDelegation is to be managed in the style of a consumptive license; the value of this attribute

gives the number of units for which license documents may be generated (i.e., if granted 1000 units by a Producer, an Issuer can only issue 1000 units.) Template-authorization provides a "template" Product Use Authorization whose attribute values must be included on any Product Use Authorization generated using this IssuerDelegation; in the case of attributes which have a scalar value (i.e., Version, Release Date, etc.), the Issuer may issue licenses with more restrictive values than those specified on the Template Authorization. Sub-license-permitted indicates whether the Issuer identified on this IssuerDelegation may issue an IssuerDelegation for the delegated-product-id.

A license delegation, as shown in a syntax diagram of Figure 24, is used to delegate the right to manage license data. Such delegations are created by the licensee (by the license manager), if authorized by the issuer 28. A backup delegation, also shown in Figure 24, is used by one license management facility to authorize another to manage the delegated rights in the case that the delegating license manager is not running. The delegated-units field specifies the number of units whose management is being delegated; this may only be specified when a product use authorization is being delegated. Delegation-distribution-control defines the mechanisms by which the distribution and refreshing of the delegation will be accomplished. Delegatee-execution-constraints identifies any constraints which are placed on the execution-context of the Delegatee; these constraints are applied in addition to those which are a part of the delegated License Data. Assignment-list identifies any assignments of the delegated units that must be respected by the delegatee. Delegated-data stores a copy of the LicenseData received from the issuer that is the subject of the delegation; the delegated data is not provided when the LicenseDelegation element is included in a DelegationList.

The management information or ManagementInfo element records information concerning the current state of the LicenseData with which it is associated. A syntax diagram of the ManagementInfo element is shown in Figure 25. The assignments field identifies a list of one or more assignments which may be outstanding for the units on the associated product use authorization. Reservations identifies a list of one or more reservations which may be outstanding for the units on the associated product use authorization. Delegations identifies a list of all outstanding delegations. Backup-delegations identifies all outstanding backup delegations. the allocations field provides detailed information about outstanding allocations which involve units from the associated product use authorization. Registration-date is the date on which the LicenseData was registered in the license database. Registrar is the context which caused the LicenseData to be registered. Local-comment is a comment field. Termination-date means a license defined date after which the license data may not be used; this date must be earlier than the end-date specified in the license data's term record. The extended-info field allows additional information concerning the state of the LicenseData and its handling by the license manager that is not standardized.

The defined types of elements will now be described. These defined type are:

Allocation	ManagementPolicy
Assignment	Member
Context	NamedValue
DistributionControl	NamedValueList
ExecutionConstraints	ProductID
IntervalTime	Signature

LicenseID	Term
LUDRM	Version
ManagementConstraints	

The allocation element records the information concerning a single unit allocation, and is shown in a syntax diagram in Figure 26. Allocation-context specifies the context to which the allocation was made. The allocation-lur field specifies the license unit requirement which applies to the allocation-context; this license unit requirement is calculated without consideration of any allocation sharing which may be possible. The allocation-group-id field identifies the "allocation-group" for the current allocation, in which an unshared allocation will always have an allocation group id of 0; allocations which utilize shared units will have an allocation group id which is shared by all other allocations sharing the same units.

The assignment element is shown in syntax diagram in Figure 27. Assigned-units identifies the number of units which are assigned. Assignment-term identifies the start and end of the assignment period. Assignee identifies the context to which the assignment is made.

The context element is shown in syntax diagram in Figure 28. The SubContext-type field identifies the type of subcontext, and this type can be either standard or private; if standard, the type value will be taken from the standard-subcontext-type enumeration: (a) network-subcontext means the subcontext value identifies a network; (b) execution-domain-subcontext means the subcontext value is the name of the management domain within which the caller is executing; (d) login-domain-subcontext means the subcontext value is the name of the

management domain within which the user of the caller was originally authenticated or "logged in"; (d) node-subcontext means the subcontext value is the name of a node; (e) process-family-subcontext means the subcontext value is an implementation specific identifier for a group of related processes; (f) process-ID-subcontext means the subcontext value is an implementation specific process identifier; (g) user-name-subcontext means the subcontext value is a user name; (h) product-name-subcontext means the subcontext value is the same as the product name found on the Product Use Authorization; (i) operating-system-subcontext means the subcontext value is a character string representation of the name of the operating system; (j) platform-ID-subcontext means the subcontext value is an identifier that describes the hardware platform supporting the context. The subcontext-value field is the value of the subcontext.

As discussed above, license data is always used or allocated within, or for the benefit of, some named licensing context. This context name is constructed by concatenating the values of all subcontexts into a single context name. A Context Template specifies those components of the context name which should be used in calculating license unit requirements. The management system determines the need to perform a unit allocation each time license units are requested. The full context on whose behalf the allocation should be made is obtained for each requested authorization. The system will mask the full context to exclude all sub-contexts not specified in the context template and then determine if the resulting context already has units allocated to it. If not, units will be allocated according to the specification of the LURDM, otherwise, the units previously allocated will be shared by the new context. Thus, if a given product authorization contains a context specification of NODE + USER_NAME, each context which requests license unit allocations and which has a unique pair

of NODE + USER_NAME subcontext values will require an explicit grant of license units to be made. On the other hand, any contexts which share the same pair of NODE and USER_NAME subcontext values will be able to "share" a single allocation of license units. The requirement for specific allocations of units and the ability to share units is exhibited in Figure 29 which attempts to provide a "snapshot" of the units allocated for the product FOOBAR V4.1 at a particular instance. It is seen from the figure that although presented with five unique full contexts, only four of them are unique when looking only at those portions of each context which are described by the Context Template (ie: NODE + USER_NAME). A unit allocation must be made for each of the four instances of unique contexts, when masked by the Context Template. The fifth context can share allocated units with another context. Thus, the total requirement to support product use as described in this example would be 40-units (ie: four allocations of ten units each). Significant changes in the unit requirements can be achieved by making small modifications to the Context Template. Figure 30 shows the same contexts as in Figure 29 but a Context_Template of NODE. The total unit requirement for this example would be three units (three allocations of ten units each) rather than the forty units required in the previous example.

The distribution control element defines the mechanism that will be used for distributing the subject delegation and records some status information concerning the distribution of that delegation. A syntax diagram of the distribution control element is shown in Figure 31. Distribution-method identifies the means by which the delegation will be distributed, and the alternatives are refresh-distribution, initial=distribution-only, and manual-distribution. Refresh-distribution means the license manager shall be responsible for the initial distribution of the delegation and for ensuring that refresh delegations are

properly distributed. Initial-distribution-only means the license manager shall be responsible for the initial distribution of the delegation, however, distribution of refresh delegations will be made by some other means. Manual-distribution means the distribution of the delegation will be under the control of some other mechanism (perhaps a license asset manager). Current-start-date is the time that the last successful initial or refresh delegation distribution was performed. Current-end-date identifies the last date on which the most recent delegation distribution was performed. Refresh-interval identifies the period of time between attempts to refresh the delegation; the refresh-interval may not be longer than the maximum-delegation-period and should normally be less than that in order to ensure that refresh delegations are distributed prior to the expiration of the previous delegations that they are replacing. Retry-interval identifies the amount of time to wait for an unsuccessful distribution attempt to try again. Maximum-retry-count identifies the maximum number of times that an unsuccessful distribution attempt may be retried. Retries-attempted records the number of unsuccessful retry attempts which have been made since the last successful initial or refresh delegation distribution was performed.

The execution constraints elements place limits on the environments and contexts which may receive allocations. A syntax diagram of the execution constraints element is shown in Figure 32. Operating-system contains a list of zero or more operating systems on which the use of the subject license is authorized; if no operating systems are specified, it is assumed that license use is authorized on all operating systems. Execution-context specifies a list of zero or more full or partial context names which identify the contexts within which products described by the license data may be executed; if no context names are specified, the licensed products may be executed in any context controlled by the licensee.

Environment-list identifies those environments within which the licensed product may be used.

The interval time element is defined by the syntax IntervalTime ::= UTCTime.

The license ID element uniquely identifies the license data it is associated with, and is described by the syntax diagram of Figure 33. Here issuer uniquely identifies the issuer of the license data as well as the name space within which the LicenseID Number is maintained. While the issuer name will typically be the same as the name of the issuer's company or personal name, this is not a requirement. For instance: The issuer name for Digital Equipment Corporation is "DEC," an abbreviation of the corporate name. Valid contents of the Issuer field are maintained in the an Issuer Registry. The serial-number provides a unique identification or serial number for the license data. The amendment field is an integer which is incremented each time license data is amended by its issuer, with the first version of any license data carries the amendment number 0; an amendment can only be applied to license data if that license data has identical Issuer and Number values and an amendment number less than the number of the amendment to be applied.

The license units requirements determination method or LURDM element is shown in syntax diagram in Figure 34. The combination-permitted field indicates whether conforming license managers are permitted to combine together into a common pool the units from different product use authorizations if those produce use authorizations have the same product record value; for example, if combination is permitted and a single license manager discovers in its database

two 500-unit authorizations for the use of DEC Cobol, the license manager would be permitted to combine these two authorizations into a logical grant of 1000 units. The overdraft-limit modifies the behavior of a conforming license management facility in those cases where it is found that there are zero or fewer license units available for use at the time of a request for the allocation or consumption of additional license units. Operation of overdraft is different depending upon whether allocative, or consumptive style is being used. In using with allocative style, an allocation is granted even though the remaining units are zero or less, up to the overdraft-limit. In using with consumptive style, the license is authorized to accumulate a negative balance of license units, up to the overdraft-limit. Overdraft-logging-required indicates whether all license grants which are the result of overdraft use must cause a log record to be generated. When the allocation-size field is non-zero, then all unit allocations and delegations must be made in sizes which are whole number multiples of the allocation-size value. Lurdm-kind identifies the method by which license unit requirements will be calculated once the requirement for an allocation has been discovered, the permitted alternatives being (a) LURT which specifies that license unit requirements are to be determined by lookup in the LURT which is associated with the current license, (b) Constant which specifies that license unit requirements are constant for all platforms on which the licensed product or product feature may run, and (c) Private-LURDM which specifies that license unit requirements are to be determined by the licensed product, not by the license management facility. The named-lurt-id specifies the name of the LURT table to be used in determining license unit requirements if the LURDM-kind is specified as LURT; if the LURDM-kind is specified as LURT and no table is explicitly named, the name of the table to be used is constructed from the issuer name on the product use authorization. Lurdm-value specifies the LURT column to be

used when LURDM-kind = LURT; however, when LURDM-kind = Constant, the Lurdm-value field contains the precise number of units to be allocated or consumed. Default-unit-requirement specifies the unit requirement value to be used when the appropriate LURT does not have a row corresponding to the appropriate platform ID; when specified on a product use authorization with Style = Allocative, the context template will change to Process + Product_Specific and the Duration will change to Transaction in cases of unrecognized Platform ID's.

The management constraints element is shown in a syntax diagram in Figure 35. The management-context field specifies a list of zero or more partial context names which identify the specific contexts within which the license data may be managed. If no management contexts are specified, the license data may be managed within any context controlled by the licensee. The contexts used in specifying Management Context Constraints may only contain the Network, Domain, and Node subcontexts. Specifying a list of management contexts does not effect whether or not the license data can be used within other contexts. For example, unless otherwise restricted, license data with a specified management context can be remotely accessed from or delegated to other nodes in a network. The management-scope field defines the maximum permitted size of the license management domain within which the license data may be managed or distributed, these being single-platform, management-domain, or entire-network. Single-platform constrains the license management domain for the subject license data to be no larger than a single platform. Management-domain constrains the license management domain for the subject license data to be no larger than a single management domain. Entire-network constrains the license management domain for the subject license data to be no larger than a single wide area network; that

network which contains the platform on which the license units were initially loaded. Although technology may not exist to detect the interorganizational boundaries of a wide area network (i.e., what is on the Internet as opposed to being on a company's own network), the assumption is that interorganization and internetwork sharing of licenses will normally be considered a violation of license terms and conditions. The backup-permitted field indicates if the Issuer has authorized the use of backup delegations for this data. Delegation-permitted indicates if the Issuer has authorized the licensee to delegate this data. Maximum-delegation-period identifies the longest interval during which a delegation may be valid; by default, delegations have a life of 72-hours.

The major elements of the management policy specification are shown in Figure 3, as previously discussed. A syntax diagram for the management policy element is shown in Figure 36. For the Style field, three fundamental styles of license management policy are supported, allocative, consumptive, and private-style, as explained above. Only one of these styles may be assigned to any single product use authorization. The Context-template specifies those components (sub-contexts) of the execution-context name which should be used in determining if unit allocations are required. The Duration defines the duration of an allocation of license units to a specific context or the duration of the period which defines a valid consumptive use. For durations of type "Assignment," the specification of a Reassignment Constraint is also provided for. Three types of Duration_Kind are supported, these being Transaction, Assignment and Immediate, as explained above. The lur-determination-method stores information used in calculating the number of units that should be allocated or consumed in response to a license request. The allocation-sharing-limit identifies the largest number of execution contexts that may share an allocation made under this management policy; an

allocation-sharing-limit of 0 indicates that the number of execution contexts that may share an allocation is unlimited. The reassignment-constraint specifies a minimum duration of assignment; although there is normally no constraint placed on how frequently granted units may be reassigned, an issuer may constrain reassignment by specifying this minimum duration of an assignment, in which case reassignment of assigned units will not be supported until the amount of time specified in the Reassignment Constraint has passed. If an assignment of some particular set of units has been delegated and the delegation period for that delegation has not terminated, cancellation of the delegation must be performed prior to reassignment.

The member element identifies a specific licensed product which may be part of a calling authorization or group definition, and is shown in syntax diagram in Figure 37. Member-product identifies the product which is a member. Member-signature is constructed from the product and token fields of the called member structure as well as the product and issuer fields of the calling product. Member-token provides the data which should be used as the product token for this member.

Named values are data elements with a character string tag that identifies the data element, and have a syntax as shown in Figure 38, which also shows the syntax for ValueData and named value list. A named value list models a list of named values, with an example being shown in Figure 39. In Figure 38, Value-Name uniquely identifies the value; no standard value names are defined, and the period character can be used as a part of the value name to form a hierarchical tag registry at the discretion of the issuer. Value-data is the data that has been named; data types are selected from the possible Value Data types, seen in the

Figure. Value-boolean means the named data is a boolean value. Value-integer means the named data is an integer value. Value-text means the named data is a StringList value. Value-general means the named data is a stream of bytes in any format. Value-list means the named data is a list of named data values.

The product ID explicitly identifies the product which is the subject of the license data with which it is associated, with the syntax for ProductID being shown in Figure 40. The version and release date fields provide a mechanism for defining which specific instances of the licensed product are described in the associated license data. The Producer field is a registered name which identifies the producer of the licensed feature; in the case of Group Names, the Producer is always also the Issuer of the group. The Product-name identifies a licensed software feature. The First-version identifies the earliest version of the product whose use is authorized. The Last-version identifies the latest version of the product whose use is authorized. The First-release-date identifies the earliest release of the product whose use is authorized. The Last-release-date identifies the latest release of the product whose use is authorized. Conforming license managers are required to interpret the contents of these fields in the most restrictive way possible. Thus, if a license is issued with Last-version = 3.0 and a Last-release-Date of 1-Jan-1991, then the use of version 2.0 of the licensed product would be unauthorized if it had a release date of 2-Jan-1991. If either a First-version or First-release-date is specified without a matching Last-version or Last-release-date, use of the produce is authorized for all versions or release dates following that specified. Similarly, if either a last-version or Last-release-date is specified without a matching First-version or First-release-date, use of the produce is assumed to be authorized for all versions or release dates prior to that specified. Issuers should typically only specify one of either First-version or First-release-

date. This is the case since it is anticipated that these fields will typically refer to events which occurred prior to the moment of license data issuance. Thus, it should normally be possible for the issuer to state unambiguously with only one of these two fields which is the oldest implementation of the product that is to be authorized. The architecture does permit, however, both fields to be used in a single product authorization.

The signature element is used to establish the integrity and authorship of the license data with which it is associated. A syntax diagram for the signature element is shown in Figure 41. The Signature-algorithm field identifies the registered algorithm that was used to produce the digital signature. Signature-parameters are the values of the algorithm's parameters that are to be used; the need for and syntax of parameters is determined by each individual algorithm. Signature-value is an enciphered summary of the information to which the signature is appended; the summary is produced by means of a one-way hash function, while the enciphering is carried out using the secret key of the signer (Issuer).

The term element defines an interval during which the license data is valid, and is shown in syntax diagram form in Figure 42. The fields are start-date and end-date. Start-date identifies the first date of the term; if not specified, the license data is considered valid on any date prior to the end-date. End-date identifies the last date of the term; if not specified, the license data is considered valid on any date after the Start-date. While the Start-date is always either omitted or specified as an absolute date, the End-date can be either absolute or relative. If the End-date is specified as a relative or "interval" date and the Start-date has been omitted, the date of license registration will be used as the effective

start date in computing the valid term of the license data. It should be noted that the system does not specify the mechanism by which system dates are maintained by platforms supporting system components. Instead, the system always accepts that system time returned to it as correct. Thus, the reliability of the management of license data which specifies terms is dependent on the time management function of the underlying platform.

The version element identifies a four-part version of the licensed software product or feature. A syntax diagram of the version element is shown in Figure 43. The schematics of each of the four parts is not detailed, but it is required that producers who wish to permit version ranges to be specified on product use authorizations ensure that the collating significance of the four parts is maintained. When comparing versions, Part-1 is considered first, then Part-2, then Part-3, and finally, Part-4. Part-1 identifies a major modification to the versioned object. Part-2 identifies a modification to the versioned object which is less significant than a modification which would cause a change in the Part-1 value. Part-3 identifies a modification to the versioned object which is less significant than a modification which would cause a change in the Part-2 value. Part-4 identifies a modification to the versioned object which is less significant than a modification which would cause a change in the Part-3 value.

FILTERS

An important feature is the use of filters in the license management program 11, including the client interface 31 and the management interface 33. A filter is used is select items in the license database 23, for example. Various

selection mechanisms are used in picking out or doing lookups in database technology; filters are one of them. The filter engine used in the license management system 11 of Figure 1 is generally of a known construction, with the exception of the select filter item type as will be described, which allows a complex rather than a flat data format to be selected from. The feature that is of importance to this embodiment is the way of specifying items as an input to the filter function , rather than the filter function itself. Thus, there is described below a template for specifying input to the filter engine. This is as if a form were used as the input, with blanks on the form; by filing in certain blanks these would be the items selected on, the blanks not filled in would be "don't care".

An instance of the class *filter* is a basis for selecting or rejecting an object on the basis of information in that object. At any point in time, a filter has a value relative to every object - this value is false, true or undefined. The object is selected if and only if the filter's value is true. This concrete class has the attributes of its superclass - *Object* - and the specific attributes listed in the table of Figure 44.

A filter is a collection of simpler filters and elementary filter-items together with a Boolean operation. The filter value is undefined if and only if all the component filters and filter-items are undefined. Otherwise, the filter has a Boolean value with respect to any object, which can be determined by evaluating each of the nested components and combining their values using Boolean operation (components whose value is undefined or ignored). The attributes specific to *filter* as shown in Figure 44 are (a) *filter items* which are a collection of assertions, each relating to just one attribute of an object, (b) *filters* which are a

collection of simple filters, and (c) *filter type* which is the filter's type, of one of the following values: And, Or, Not.

An instance of the class *filter item* is a component of a *filter*. It is an assertion about the existence or values of a single attribute of a license data object or one or its subobjects. This concrete class has the attributes of its superclass - *object* - and the specific attributes listed in the table of Figure 45.

The value of a filter item is undefined if: (a) the Attribute Types are unknown, or (b) the syntax of the Match Value does not conform to the attribute syntax defined for the attribute type, or (c) a required Attribute is not provided. The attributes specific to *filter item* as shown in Figure 45 are (a) *filter item type* which identifies the type of filter item and thereby the nature of the filter, and its value must be one of

equality	less
inequality	present
greater or equal	select
less or equal	request candidates
greater	simulate request

(b) *attribute type* which identifies the type of that attribute whose value or presence is to be tested; the value of All Attributes may be specified, (c) *match value* which is the value which is to be matched against the value of the attribute, (d) *filter* which identifies the filter to be used in evaluating a selected subobject of the current object; the filter is ignored if the *filter item type* is not *select* or if the specified attribute type is not present in the object, and upon evaluation of the *filter* the value of *filter item* will be set to that of the *filter*, (e) *initial substring*, if present, this is the substring to compare against the initial portion of the value of

the specified attribute type, (f) *substring*, if present, this is the substring(s) to compare against all substrings of the value of the specified attribute type, (g) *final substring*, if present, this is the substring to compare against the final portion of the value of the specified attribute type, and (h) *license request*, if present, this is license request against which the appropriate license data objects should be evaluated; this attribute may only be specified if the value of the filter item type is either Request Candidates or Simulate Request.

An instance of enumeration syntax *Filter Type* identifies the type of a filter. Its value is chosen from one of the following: (a) *And* means the filter is the logical conjunction of its components; the filter is true unless any of the nested filters or filter items is false, or if there are no nested components, the filter is true; (b) *Or* means the filter is the logical disjunction of its components; the filter is false unless any of the nested filters or filter items is true, or, if there are no nested components, the filter is false; (c) *Not* means the result of the filter is reversed; there must be exactly one nested filter or filter item, and the filter is true if the enclosed filter or filter item is false, and is false if the enclosed filter or filter item is true.

An instance of enumeration syntax *Filter Item Type* identifies the type of a filter item. Its value is chosen from one of the following: (a) *Equality* which means the filter item is true if the object contains at least one attribute of the specified type whose value is equal to that specified by Match Value (according to the equality matching rule in force), and false otherwise; (b) *Inequality* which means the filter item is true if the object contains at least one attribute of the specified type whose value is not equal to that specified by Match Value (according to the equality matching rule in force), and false otherwise; (c) *Greater*

or Equal which means the filter item is true if the object contains at least one attribute of the specified type whose value is equal to or greater than the value specified by Match Value (according to the matching rule in force), and false otherwise; (d) *Less or Equal* which means the filter item is true if the object contains at least one attribute of the specified type whose value is equal or less than the value specified by Match Value (according to the matching rule in force), and false otherwise; (e) *Greater* which means the filter item is true if the object contains at least one attribute of the specified type whose value is greater than the value specified by Match Value (according to the matching rule in force), and false otherwise; (f) *Less* which means the filter is true if the object contains at least one attribute of the specified type, whose value is less than the value specified by Match Value (according to the matching rule in force),and false otherwise; (g) *Present* which means the filter item is true if the object contains at least one attribute of the specified type, and false otherwise; (h) *Select* which means the filter item is true if the object contains at least one attribute of the specified type which has an object syntax and when the Filter is evaluated against the attributes of that object the Filter is true, and false otherwise; (i) *Request Candidates* which means the filter item is true if the object against which it is evaluated is one which could be used to provide some or all of the units requested by the specified License Request; the evaluation is made independently of any outstanding allocations or preallocations; and (j) *Simulate Request* which means the filter item is true if the object against which it is evaluated is one which would be used to provide some or all of the units requested by the specified License Request.

The Request Candidates and Simulate Request filter item types are of special use in testing and prototyping of systems by a license manager at a

licensee's site. For example, the license manager can simulate the effect of potential assignments, the effect of a population of certain types on a network, etc.

As an example, Figure 46 shows how a filter may be constructed to identify "All Product Use Authorizations issued by Digital for the Product 'Amazing Graphics System' which contains a calling authorization for Digital's 'Amazing Database' Product". This example is in the international standard format referred to as X.409 as mentioned above.

Filters can also used in a request allocation, being specified in a request extension as explained above. That is, a filter is one of the optional items in a request extension. For example, if a user wanted to use a version of WordPerfect with French language extension, and there were version with and without on the network, his request allocation would have a request extension that specified a filter for "French" in the token field. In this manner, a product can describe itself more richly. The filter in the request extension can be a Required filter or a Preferred filter, meaning the feature such as "French" is either absolutely necessary, or merely the preferred.

While this invention has been described with reference to specific embodiments, this description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the true scope of the invention.

WHAT IS CLAIMED IS:

1. A method of managing use of licensed software items, said software items separately executable on a computer system or accessible by said computer system, the computer system including a processor and one or more nodes, comprising the steps of:

maintaining by said processor a store of license authorizations for said software items; each license authorization including an indication of license management policy for a software item, said indication having a plurality of sets of policy components, said sets of policy components granting alternatives of specified restrictive rights to selectively access and execute said software items in said system; said indication of license management policy being in the format of an encoded document of a data type consisting of an ordered sequence of elements;

accessing said store by said processor to modify in said store one or more of said specified restrictive rights of said policy components of an identified license authorization;

accessing said store by said processor using a filter to obtain information from said license authorization for a selected software item, in response to a request from a node, and

comparing an identification of said node and said software item with said information, to produce and send to said node a grant or refusal of said request.

2. A method according to claim 1 including the step of receiving said license authorizations , for storing in said store,

from a license grantor external to said processor, and wherein said step of accessing said store to modify in said store one or more of said specified restrictive rights employs management functions executable on said processor but not on said nodes or said license grantor to identify a license authorization in said store.

3. A method according to claim 1 wherein said indication is in the format of an encoded document of a data type consisting of an ordered sequence of three elements, the three elements including a document descriptor, a document header and the document content.

4. A method according to claim 1 wherein said filter specifies one or more of said attributes and a Boolean operator for each selected attribute.

5. A method according to claim 2 wherein said step of accessing said store to modify one or more of said policy components is to allow grant of rights to use which are more restrictive than said specified restrictive rights.

6. A method according to claim 2 including the steps of:

sending a request by a user of one of said software items to obtain permission to use said software item; said request identifying the user and said software item;

accessing said store to obtain information from said license authorization for said software item, in response to said request, and comparing said identification of said user and said software item with said information, to produce a grant or refusal of said request for sending to said user.

7. A method according to claim 6 wherein said store is maintained by a license server, and said request is sent to said server and wherein said request is in the form of a remote procedure call, and said grant or refusal sent to said user is a return of said procedure call.

8. A method according to claim 7 wherein said license authorization is a data arrangement specified as a product use authorization, and said product use authorization is received by said server from an issuer, and wherein said server and said users are nodes on a computer network.

9. A method according to claim 2 wherein said policy components include a termination date, and said management functions can modify said termination date to an earlier termination date and wherein said policy components include a right of delegation of a right to grant said requests to another server, and said management functions can modify said right of delegation to remove said right of delegation.

10. A method according to claim 2 including storing in association with said license authorization a number of management attributes, and said management functions being able to modify said management attributes.

11. A method according to claim 10 wherein said management attributes include a reservation of units of license use granted by said license authorization so that said units will not be granted to a user in response to said request, and wherein said management attributes include an allocation of units of license use to a specific context.

12. A method according to claim 10 wherein said management attributes include an allocation period which is the minimum duration of an allocation of units, and wherein said management attributes include permission to enable a backup delegation of the right to grant said requests.

13. A system for managing use of licensed software products, comprising: means for maintaining a store of license documents, one for each said product; each license document including an indication of license policy having plurality of sets of policy components granting specified restrictive rights to use said software products, said policy components in each set providing alternatives;

a management interface for accessing said store to modify

selected ones of said components of an identified license authorization.

14. A system according to claim 13 including:

means for sending a request from a user of one of said products to obtain permission to use said product; said request identifying the user and said product;

means for accessing said store to obtain information from said license document for said product, in response to said request, and for comparing said identification of said user and said product with said information, and with constraints imposed by said policy components, to produce a grant or refusal of said request and send said grant or refusal to said user.

15. A system according to claim 13 wherein said management interface can modify said selected ones of said components to allow grant of rights to use which are more restrictive than said specified restrictive rights and wherein said means for maintaining, and said means for accessing and sending to said user are all located at a server on a distributed network, and said means for sending a request is located at a user node on said network.

16. A system according to claim 14 wherein said request is in the form of a remote procedure call, and said grant or refusal sent to said user is a return of said procedure call, and wherein said

license document is a data arrangement specified as a product use authorization, and said product use authorization is received by said server from a license issuer.

17. A system according to claim 13 wherein said policy components include a termination date, and said management functions can modify said termination date to an earlier termination date, and wherein said policy components include a right of delegation of a right to grant said requests to another server, and said management functions can modify said right of delegation to remove said right of delegation.

18. A system according to claim 15 including means for storing in association with said license authorization a number of management attributes, wherein said management functions are able to modify said management attributes and wherein said management attributes include a reservation of units of license use granted by said license authorization so that said units will not be granted to a user in response to said request.

19. A system according to claim 18 wherein said management attributes include an allocation of units of license use to a specific context.

20. A system according to claim 18 wherein said management attributes include an allocation period which is the minimum

duration of an allocation of units, and include permission to enable a backup delegation of the right to grant said requests.

21. A method according to claim 3 wherein said document descriptor includes an encoding method version number, and encoder-identifier and an encoder-name, and wherein said document-header includes a title, an author, a version and a date for the software item.

22. A method according to claim 3 wherein said document content includes at least one of the following:

a product-use-authorization;

a license-use-requirements-table;

a group-definition;

a key-registration;

a delegation.

23. A method according to claim 3 wherein said document-content includes a license-data-header, and said license-data-header describes the parties to the license document, the term of the agreement and constraints that may have been placed on management of the license data.

24. A method according to claim 3 wherein said document-content includes management-info, where the management-info may include at least one of the following:

an assignment;

a reservation;

a delegation;

a backup delegation;

an allocation;

a registration date;

a registrar;

a comment;

a termination-date.

25. A method according to claim 3 wherein:

said document descriptor includes an encoding method version and a date for the software item;

said document content may include at least one of the following: a product-use-authorization, a license-use-requirements-table, a group-defination, a key-registration, and a delegation;

said document-content selectively includes a license-data-header, and said license-data-header describes the parties to the license document, the term of the agreement and constraints that may have been placed on management of the license data;

said document-content may have been placed on management of the license data;

said document-content selectively includes management-info, where the management-info may include at least one of the following: an assignment, a reservation, a delegation, a backup delegation, an allocation, a registration date, a registrar, and a

comment.

26. A method according to claim 3 wherein said store is maintained by a license server, and said request is sent to said server, and wherein said server and said users are nodes on a computer network.

27. A method according to claim 3 wherein said request is in the form of a remote procedure call, and said grant or refusal sent to said user is a return of said procedure call, and wherein said license authorization is received by said server from an issuer.

28. A method according to claim 3 including the steps of: sending a request by a user of one of said software items to obtain permission to use said software item; said request identifying the user and said software item;

sending said grant or refusal to said user.

29. Apparatus for managing use of licensed software items, comprising:

means for maintaining a store of license authorizations for said software items; each license authorization including an indication of license management policy for a software item, said indication being in the format of an encoded document of a data type consisting of an ordered sequence of three elements, the three elements including a document descriptor, a document header and the

document content;

means for sending a request by a user of one of said software items to obtain permission to use said software item; said request identifying the user and said software item;

means for accessing said store to obtain information from said license authorization for said software item, in response to said request, and comparing said identification of said user and said software item with said information, to produce a grant or refusal of said request;

means for sending said grant or refusal to said user.

30. Apparatus according to claim 29 wherein said document descriptor includes an encoding method version number, and an encoder-identifier and an encoder-name, and wherein said document-header includes a title, an author, a version and a date for the software item.

31. Apparatus according to claim 29 wherein said document content includes at least one of the following:

a product-use-authorization;

a license-use-requirements-table;

a group-definition;

a key-registration;

a delegation.

32. Apparatus according to claim 29 wherein said document-content includes a license-data-header, and said license-data-header describes the parties to the license document, the term of the agreement and constraints that may have been placed on management of the license data.

33. Apparatus according to claim 29 wherein said document-content includes management-info, where the management-info may include at least one of the following:

an assignment;

a reservation;

a delegation;

a backup delegation;

an allocation;

a registration date;

a registrar;

a comment;

a termination-date.

34. Apparatus according to claim 29 wherein:

said document descriptor includes an encoding method version number, and encoder-identifier and an encoder-name;

said document-header includes a title, an author, a version and a date for the software item;

said document content may include at least one of the following: a product-use-authorization, a license-use-requirements-

table, a group-definition, a key-registration, and a delegation;

said document-content may include a license-data-header, and said license-data-header describes the parties to the license document, the term of the agreement and constraints that may have been placed on management of the license data;

said document-content may include management-info, where the management-info may include at least one of the following: an assignment, a reservation, a delegation, a backup delegation, an allocation, a registration date, a registrar, and a comment.

35. Apparatus according to claim 29 wherein said store is maintained by a license server, and said request is sent to said server, and wherein said request is in the form of a remote procedure call, and said grant or refusal sent to said user is a return of said procedure call.

36. Apparatus according to claim 29 wherein said license authorization is received by said server from an issuer, and wherein said server and said users are nodes on a computer network.

37. A method of storing license documents by a server for a license management system, comprising the steps of:

maintaining a store of license documents for software items; each license document including an indication of license management policy for a software item, said indication being in the format of an encoded document of a data type consisting of an ordered

sequence of three elements, the three elements including a document descriptor, a document header and the document content;

accessing said store to obtain information from a selected one of said license documents for a software item, in response to a request, and referencing said indication of license management policy, to produce a grant or refusal of said request.

38. A method according to claim 37 wherein said document descriptor includes an encoding method version number, an encoder-identifier and an encoder-name, and wherein said document-header includes a title, an author, a version and a date for the software item.

39. A method according to claim 37 wherein said document content includes at least one of the following:

a product-use-authorization;

a license-use-requirements-table;

a group-definition;

a key-registration;

a delegation.

40. A method according to claim 4 wherein said step of selecting by a filter may select on one or more of the attributes: issuer, producer, product name, product use authorization, calling authorization, and wherein said store is maintained by a license server, and said request is sent to said server.

41. A method according to claim 4 wherein said request is in the form of a remote procedure call, and said grant or refusal sent

to said user is a return of said procedure call.

42. A method according to claim 40 wherein said license authorization is a data arrangement specified as a product use authorization, and said product use authorization is received by said server from an issuer, and wherein said server and said users are nodes on a computer network.

43. Apparatus for managing use of licensed software items, comprising:

means for maintaining a store of license authorizations for said software items; each license authorization including an indication of license management policy for a software item, said indication being an encoded document containing a number of attributes defining said license policy;

filter means for selecting from said store, said filter means specifying one or more of said attributes and a Boolean operator for each selected attribute;

means for sending a request by a user of one of said software items to obtain permission to use said software item; said request identifying the user and said software item;

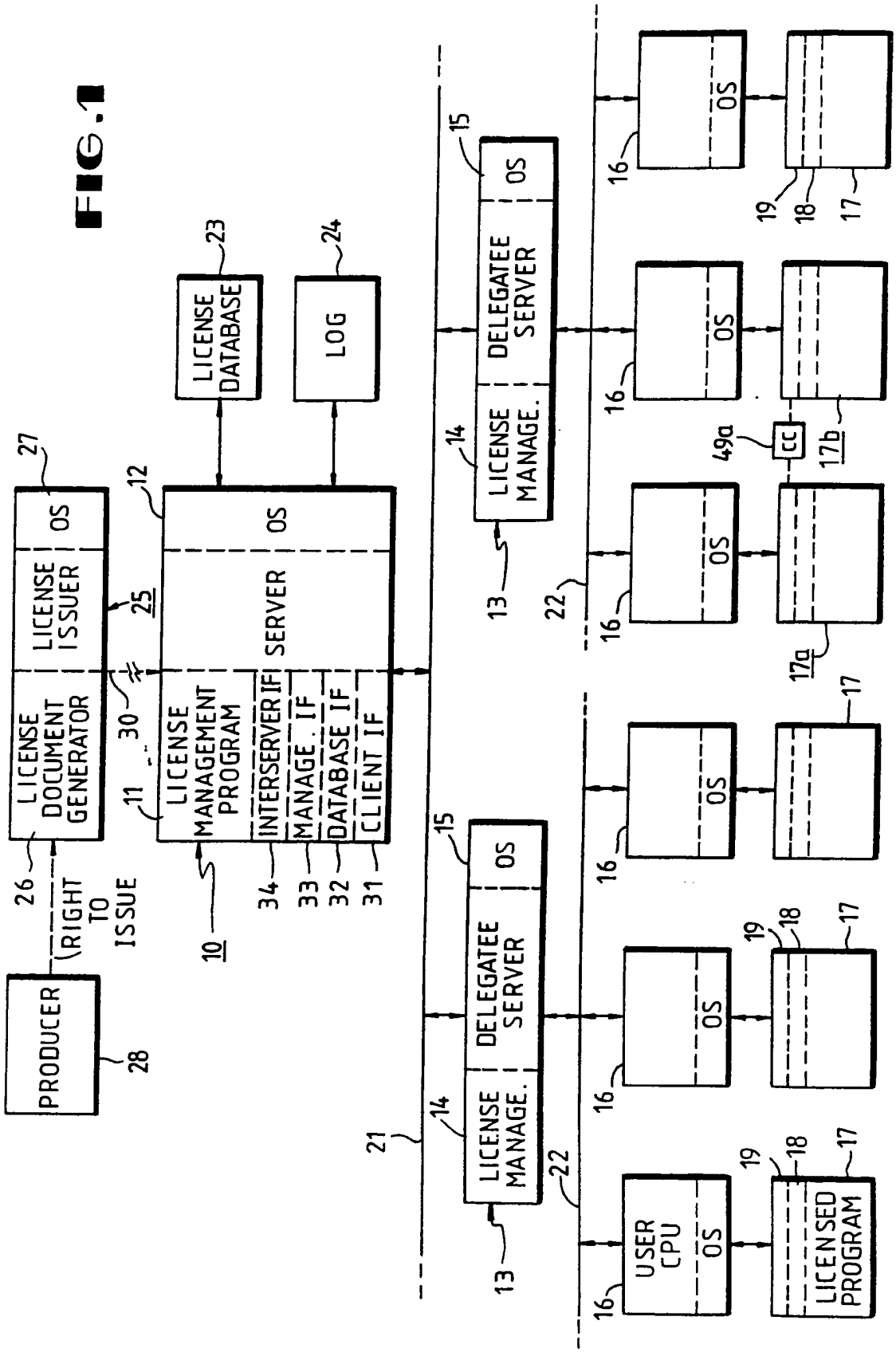
means for accessing said store to obtain information from said license authorization for said software item, in response to said request, and comparing said identification of said user and said software item with said information, to produce a grant or refusal of said request; and

means for sending said grant or refusal to said user.

44. Apparatus according to claim 43 wherein said filter means may select on one or more of the attributes: issuer, producer, product name, product use authorization, calling authorization, and wherein said store is maintained by a license server, and said request is sent to said server, and wherein said request is in the form of a remote procedure call, and said grant or refusal sent to said user is a return of said procedure call.

45. Apparatus according to claim 43 wherein said license authorization is a data arrangement specified as a product use authorization, and said product use authorization is received by said server from an issuer, wherein said server and said users are nodes on a computer network.

FIG.1



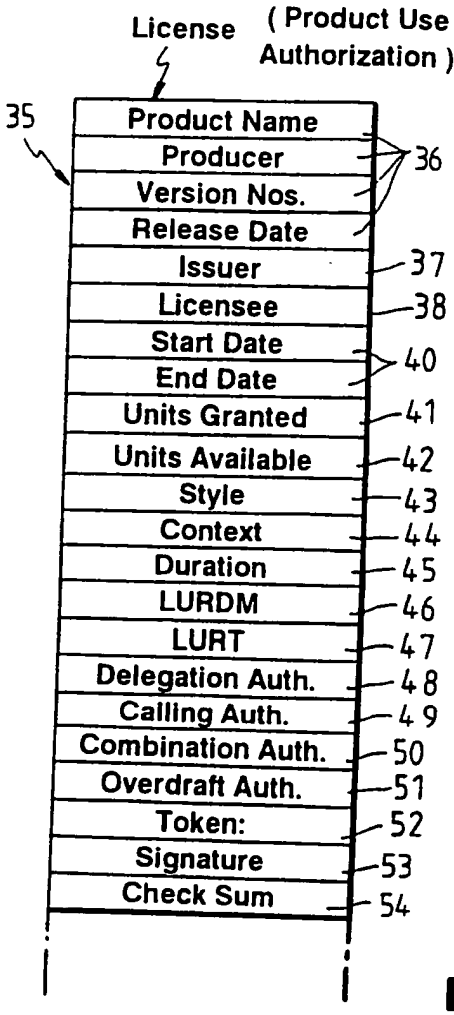


FIG.2

License Unit Requirements Table

Row Selector	Columns		
Platform ID	A	B	C
PC-0	10	230	-1
PC-1	12	230	-1
VAX 6210	158	300	150

FIG.4

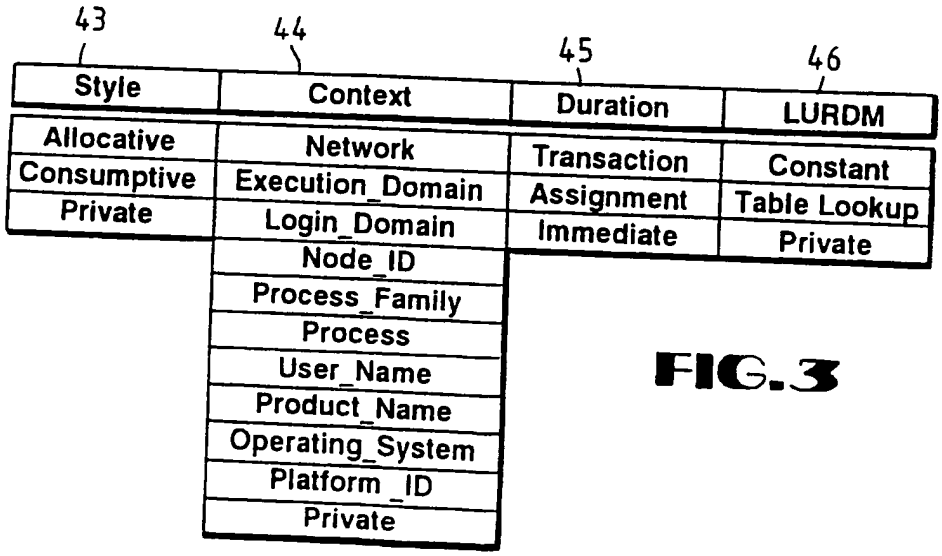


FIG.3

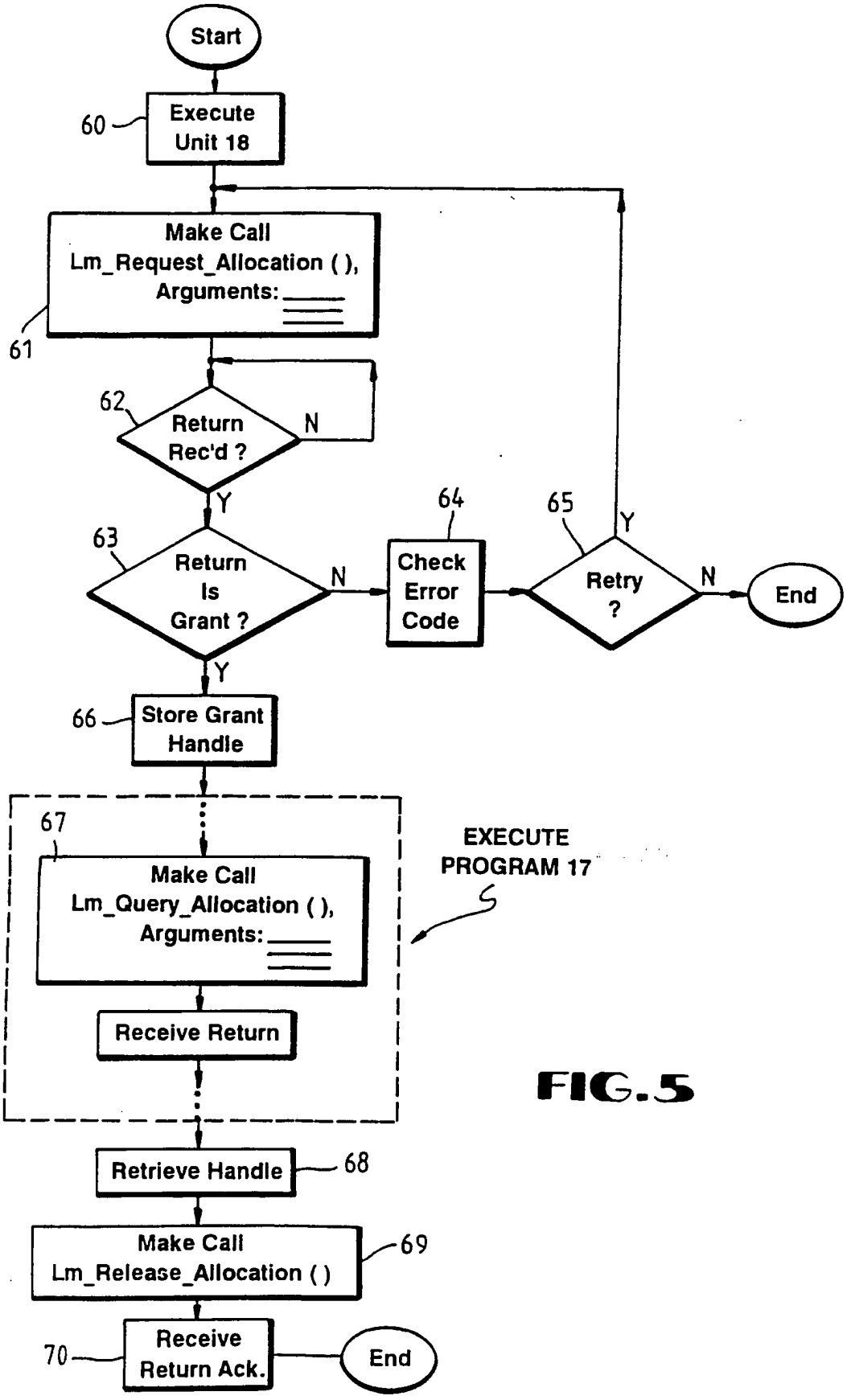


FIG.5

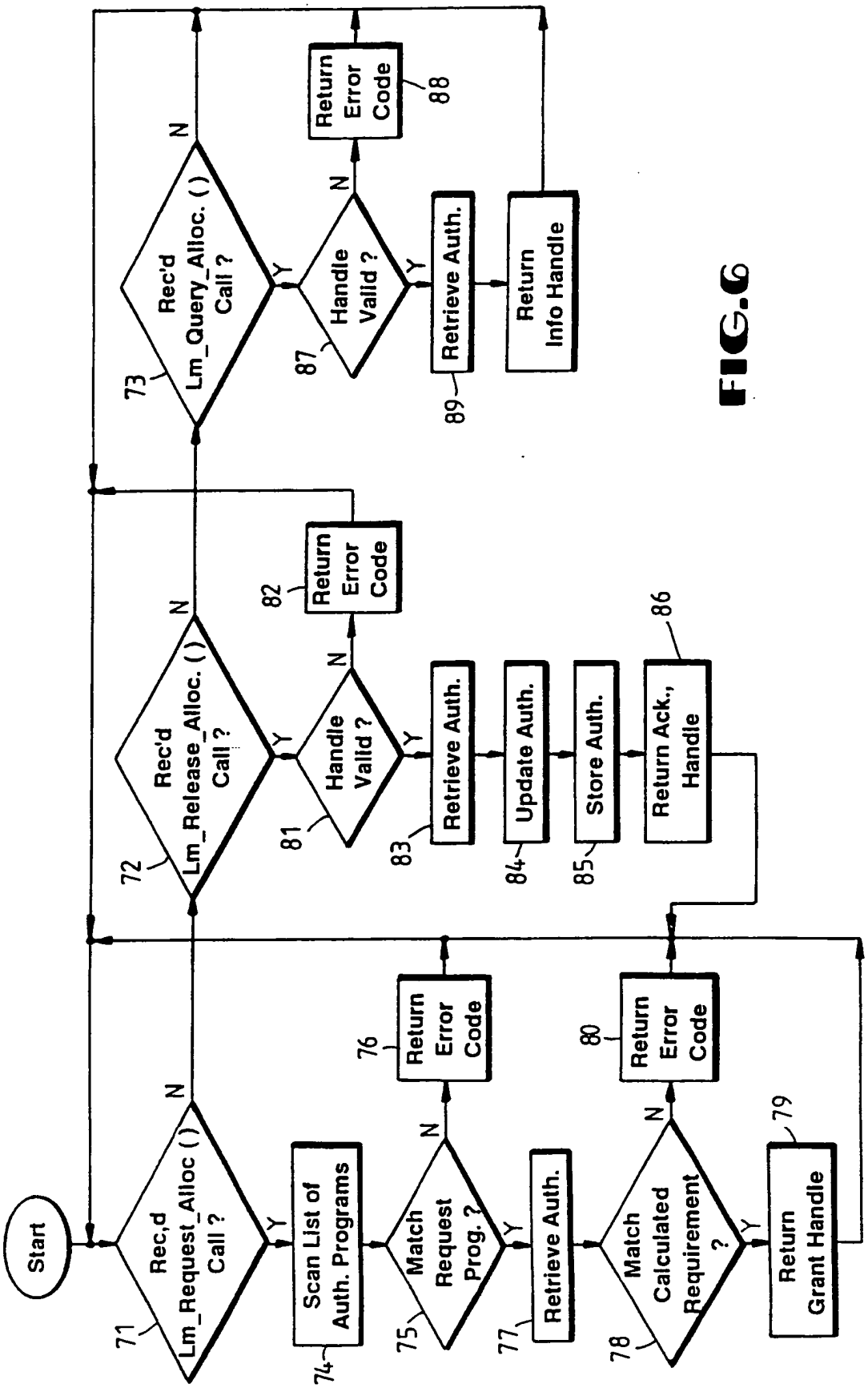


FIG.6

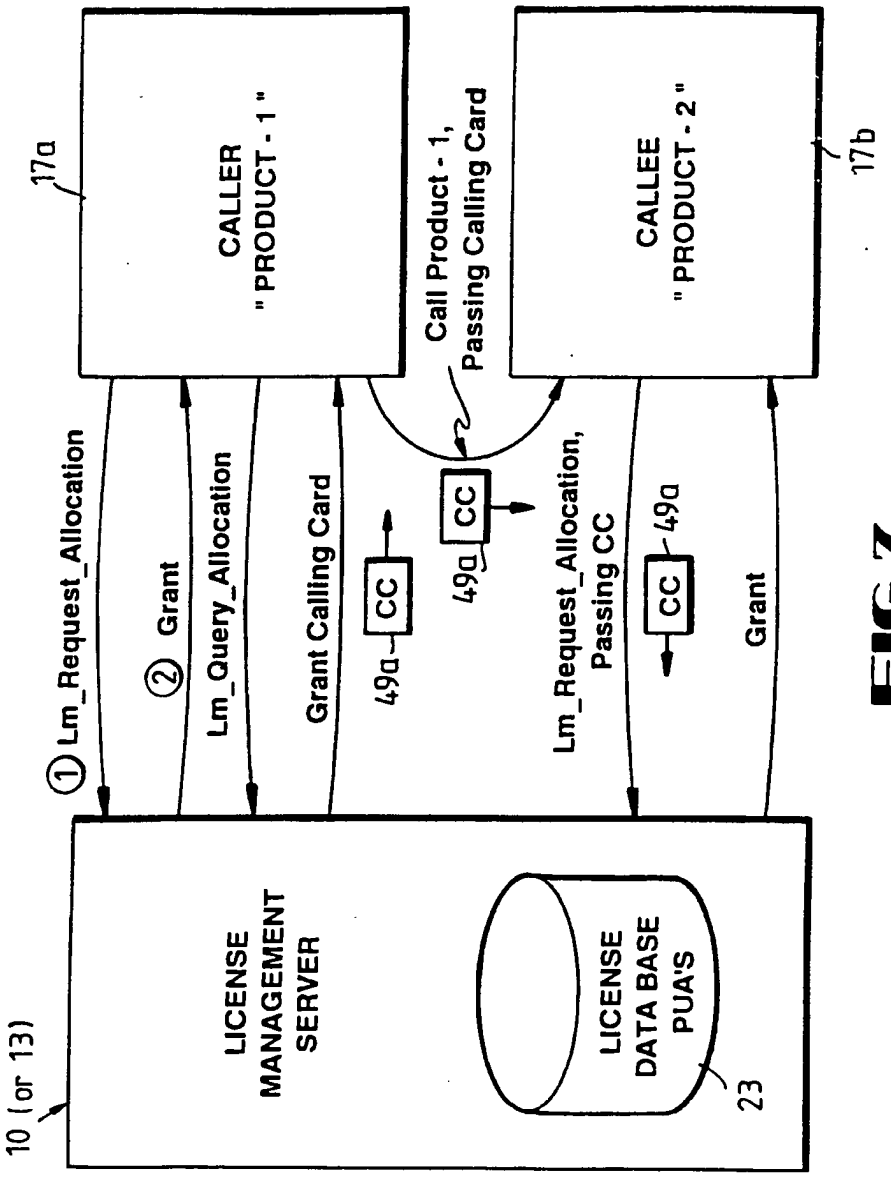


FIG.7

```
Object Identifier Value ::= {
                    iso(1)
                    identified-organization(3)
                    icd-ecma(12)
                    member-company(2)
                    dec(1011)
                    data-syntaxes(1)
                    cda(3)
                    ldif(17)
                }
Object Identifier Encoding ::= {
                    0x6, 0x8, 0x2B, 0xC, 0x2,
                    0x87, 0x73, 0x1, 0x3, 0x11
                }
```

FIG. 8 LDIF Object Identifier

```
LDIFDocument        ::=    [PRIVATE 16373] IMPLICIT SEQUENCE {
  document-descriptor          [0] IMPLICIT DocumentDescriptor OPTIONAL,
  document-header              [1] IMPLICIT DocumentHeader OPTIONAL,
  document-content             [2] IMPLICIT DocumentContent
                               }
```

FIG. 9 LDIF Document Syntax Diagram

FIG. 10 LDIF Document Structure

```
DocumentDescriptor      ::=     SEQUENCE {
    major-version                   [0] IMPLICIT INTEGER OPTIONAL,
    minor-version                   [1] IMPLICIT INTEGER OPTIONAL,
    encoder-identifier              [2] IMPLICIT Character-String OPTIONAL,
    encoder-name                    [3] IMPLICIT Character-String OPTIONAL
                            }
```

FIG. 11 Document Descriptor Syntax Diagram

```
Pakgen DocumentDescriptor  ::= {
        major-version 1,
        minor-version 0,
        encoder-identifier "PAKGEN",
        encoder-name {Character-String "PAK Generator V1.0"}
        }
```

FIG. 12 Document Descriptor Example

```
DocumentHeader      ::=     SEQUENCE {
  private-header-data           [0] IMPLICIT NamedValueList OPTIONAL,
  title                         [1] IMPLICIT Character-String OPTIONAL,
  author                        [2] IMPLICIT Character-String OPTIONAL,
  version                       [3] IMPLICIT Character-String OPTIONAL,
  date                          [4] IMPLICIT UTCTime OPTIONAL
                        }
```

FIG. 13 Document Header Syntax Diagram

```
example-header document-header ::= {
    title {Character-String "PAKGEN Licenses with Associated LURT data"}
    author {Character-String "Tom Jones, Foobar, Inc. License Department"}
    version {Character-String "V0.1"}
    date "19880102110O-0500"
    }
```

FIG. 14 Document Header Example

```
Document Content        ::= SEQUENCE OF LicenseData

LicenseData             ::= SEQUENCE {
  license-data-header       [0] IMPLICIT LicenseDataHeader,
  license-body              [1] CHOICE {
      product-use-authorization     [0] IMPLICIT ProductUseAuthorization,
      license-units-requirements-table  [1] IMPLICIT LURT,
      group-definition              [2] IMPLICIT GroupDefinition,
      key-registration              [3] IMPLICIT KeyRegistration,
      issuer-delegation             [4] IMPLICIT IssuerDelegation,
      license-delegation            [5] IMPLICIT LicenseDelegation,
      backup-delegation             [6] IMPLICIT BackupDelegation
                                },
  management-info           [2] IMPLICIT ManagementInfo OPTIONAL
                            }
```

FIG. 15 Document Content Syntax Diagram

FIG. 16 License Data Structure

```
LicenseDataHeader      ::= SEQUENCE {
   license-id                 [0] IMPLICIT LicenseID,
   licensee                   [1] IMPLICIT Character-String OPTIONAL,
   term                       [2] IMPLICIT Term OPTIONAL,
   management-constraints     [4] IMPLICIT ManagementConstraints OPTIONAL,
   signature                  [5] IMPLICIT Signature,
   issuer-comment             [6] IMPLICIT NamedValueList OPTIONAL
```

FIG. 17 License Data Header Syntax Diagram

```
ProductUseAuthorization    ::= SEQUENCE {
    product-id                     [0] IMPLICIT ProductID,
    units-granted                  [1] IMPLICIT INTEGER,
    management-policy              [2] IMPLICIT ManagementPolicy,
    calling-authorizations         [3] IMPLICIT SEQUENCE OF Member OPTIONAL,
    caller-authorizations          [4] IMPLICIT SEQUENCE OF Member OPTIONAL,
    execution-constraints          [5] IMPLICIT ExecutionConstraints OPTIONAL,
    product-token                  [6] IMPLICIT NamedValueList OPTIONAL
                               }
```

FIG. 18 Product Use Authorization Syntax Diagram


```
LURT                       ::= SEQUENCE {
    lurt-name                      [0] IMPLICIT Character-String,
    rows                           [1] IMPLICIT RowList
                               }
RowList                    ::= SEQUENCE OF LurtRow

LurtRow                    ::= SEQUENCE {
    platform-id                    [0] IMPLICIT Character-String,
    lurt-columns                   [1] IMPLICIT SEQUENCE OF INTEGER
                               }
```

FIG. 19 License Unit Requirement Table Syntax Diagram

```
Example LURT ::= {
    lurt-name { Character-String "Example LURT"}
    rows {
        LurtRow {
            {Character-String "PC-0"}
            {{10} {230} {-1}}
        LurtRow {
            {Character-String "PC-1"}
            {{12} {230} {-1}}
        LurtRow {
            {Character-String "VAX 6210"}
            {{158} {300} {150}}
        }
    }
}
```

FIG. 20 Example Encoding of LURT

```
Group Definition        ::= SEQUENCE {
    group-name              [0] IMPLICIT Character-String,
    group-version           [1] IMPLICIT Version,
    group-release-date      [2] IMPLICIT UTCTime,
    group-members           [3] IMPLICIT SEQUENCE OF Member
                        }
```

FIG. 21 Group Definition Syntax Diagram

```
KeyRegistration         ::= SEQUENCE {
    key-owner-name          [0] IMPLICIT Character-String,
    key-algorithm           [1] IMPLICIT Character-String,
    key-value               [2] IMPLICIT OCTET STRING
                        }
```

FIG. 22 Key Registration Syntax Diagram

```
IssuerDelegation        ::= SEQUENCE {
  delegated-issuer-name       [0] IMPLICIT Character-String,
  delegated-product-id        [1] IMPLICIT SEQUENCE OF Member,
  delegated-units-granted     [2] IMPLICIT INTEGER OPTIONAL,
  template-authorization      [3] IMPLICIT ProductUseAuthorization OPTIONAL,
  sub-license-permitted       [4] IMPLICIT BOOLEAN DEFAULT FALSE
                          }
```

FIG. 23 Issuer Delegation Syntax Diagram

```
LicenseDelegation       ::= SEQUENCE {
  delegated-units             [0] IMPLICIT INTEGER OPTIONAL
  delegated-distribution-control  [1] IMPLICIT DistributionControl,
  delegatee-execution-constraints [2] IMPLICIT ExecutionConstraints OPTIONAL,
  assignment-list             [3] IMPLICIT AssignmentList OPTIONAL,
  delegated-data              [4] IMPLICIT LicenseData OPTIONAL
                          }
```

FIG. 24 License Delegation & Backup Delegation Syntax Diagrams

```
ManagementInfo ::= SEQUENCE {
    assignments          [0] IMPLICIT AssignmentList OPTIONAL,
    reservations         [1] IMPLICIT AssignmentList OPTIONAL,
    delegations          [2] IMPLICIT DelegationList OPTIONAL,
    backup-delegations   [3] IMPLICIT DelegationList OPTIONAL,
    allocations          [4] IMPLICIT AllocationList OPTIONAL,
    registration-date    [5] IMPLICIT UTCTime,
    registrar            [6] IMPLICIT Context,
    local-comment        [7] IMPLICIT NamedValueList OPTIONAL,
    termination-date     [8] IMPLICIT UTCTime OPTIONAL,
    extended-info        [9] IMPLICIT NamedValueList OPTIONAL
}
```

FIG. 25 ManagementInfo Syntax Diagram

```
AllocationList      ::= SEQUENCE OF Allocation

Allocation          ::= SEQUENCE {
    allocation-context      [0] IMPLICIT Context,
    allocation-lur          [1] IMPLICIT INTEGER,
    allocation-group-id     [2] IMPLICIT INTEGER OPTIONAL
                        }
```

FIG. 26 Allocation Syntax Diagram

```
AssignmentList      ::= SEQUENCE OF Assignment

Assignment          ::= SEQUENCE {
    assigned-units          [0] IMPLICIT INTEGER,
    assignment-term         [1] IMPLICIT Term,
    assignee                [2] IMPLICIT Context
                        }
```

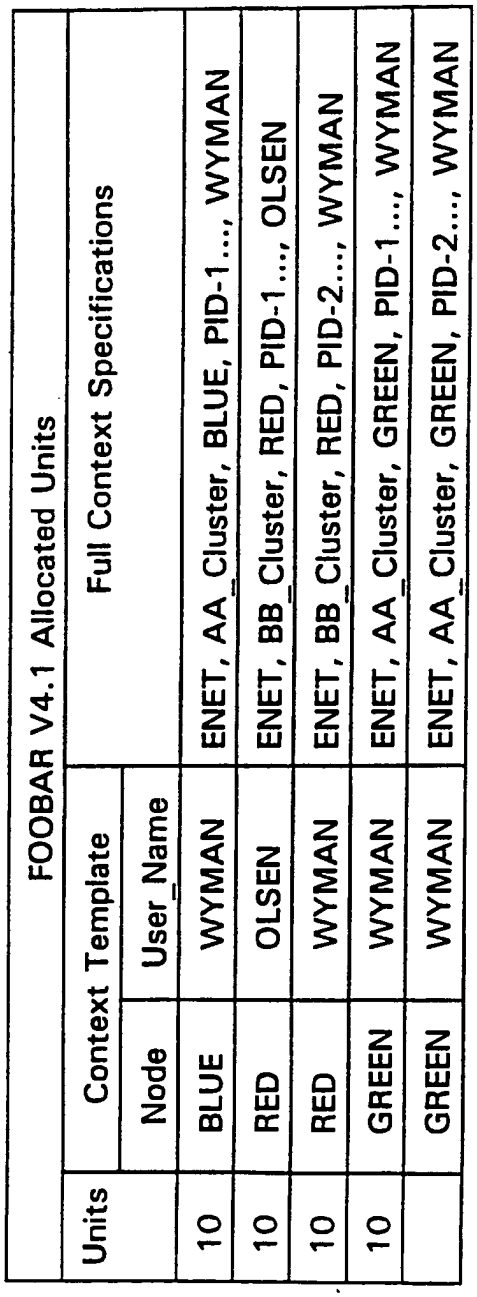
FIG. 27 Assignment Syntax Diagram

```
ContextList          ::= SEQUENCE OF Context

Context              ::= SEQUENCE OF Subcontext

SubContext           ::= SEQUENCE {
    sub-context-type          [0] SubContextType,
    subcontext-value          [1] ValueData
                         }
SubContextType       ::= CHOICE {
    standard-subcontext-type      [0] IMPLICIT INTEGER {
        network-subcontext(1),
        execution-domain-subcontext(2),
        login-domain-subcontext(3),
        node-subcontext(4),
        process-family-subcontext(5),
        process-id-subcontext(6),
        user-name-subcontext(7),
        product-name-subcontext(8),
        operating-system-subcontext(9),
        platform-id-subcontext(10)
                                  }
    private-subcontext            [1] IMPLICIT INTEGER {first(0),last(255)}
                         }
```

FIG. 28 Context Syntax Diagram



FOOBAR V4.1 Allocated Units			
Units	Context Template		Full Context Specifications
	Node	User_Name	
10	BLUE	WYMAN	ENET, AA_Cluster, BLUE, PID-1..., WYMAN
10	RED	OLSEN	ENET, BB_Cluster, RED, PID-1..., OLSEN
10	RED	WYMAN	ENET, BB_Cluster, RED, PID-2..., WYMAN
10	GREEN	WYMAN	ENET, AA_Cluster, GREEN, PID-1..., WYMAN
	GREEN	WYMAN	ENET, AA_Cluster, GREEN, PID-2..., WYMAN

FIG. 29 Only unique contexts require explicit unit allocations.

FOOBAR V4.1 Allocated Units		
Units	Context Template Node	Full Context Specifications
10	BLUE	ENET, AA_Cluster, BLUE, PID-1..., WYMAN
10	RED	ENET, BB_Cluster, RED, PID-1..., OLSEN
	RED	ENET, BB_Cluster, RED, PID-2..., WYMAN
10	GREEN	ENET, AA_Cluster, GREEN, PID-1..., WYMAN
	GREEN	ENET, AA_Cluster, GREEN, PID-2..., WYMAN

FIG. 30 Modification of Context_Template impacts units requirements.