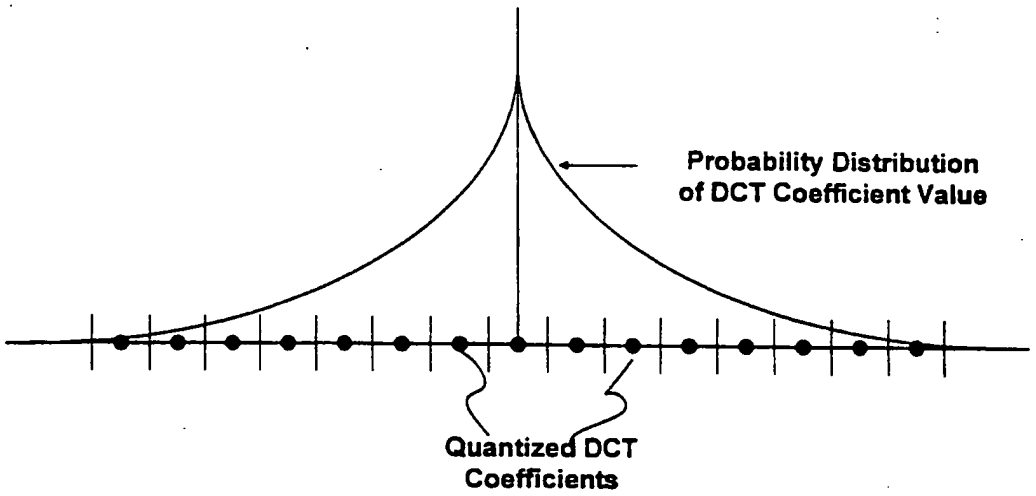


FIG. 2A

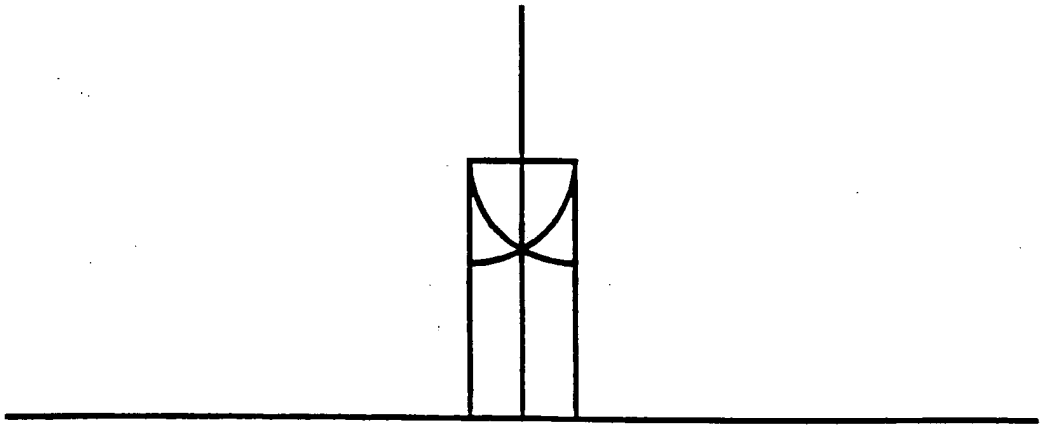


FIG. 2B

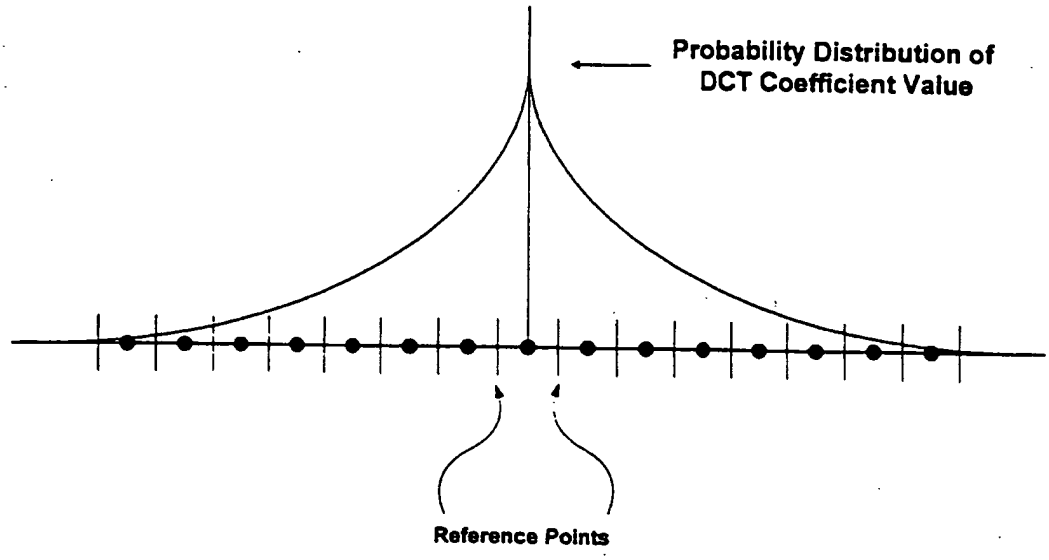


FIG. 3A

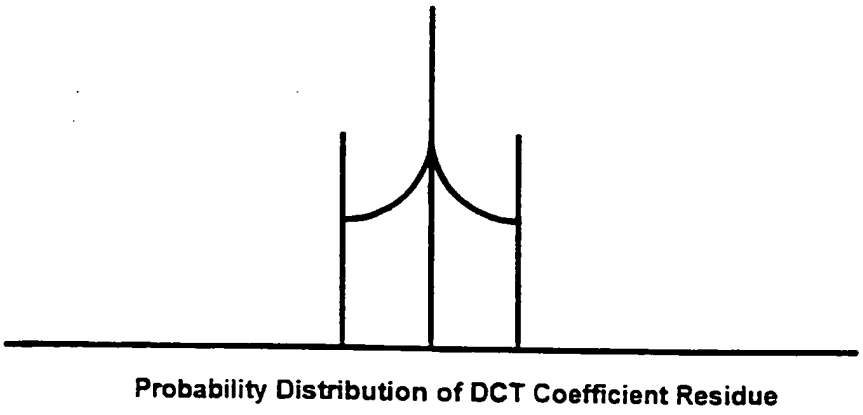


FIG. 3B

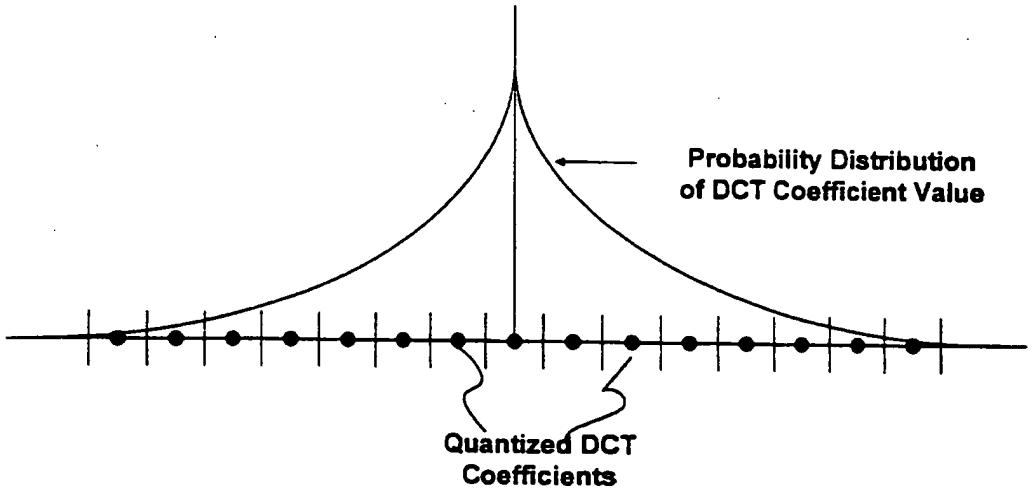


FIG. 3C

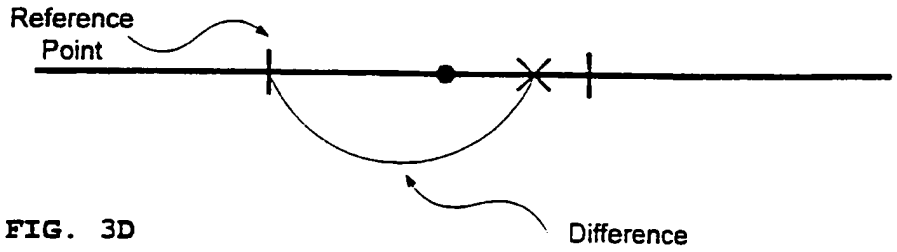


FIG. 3D

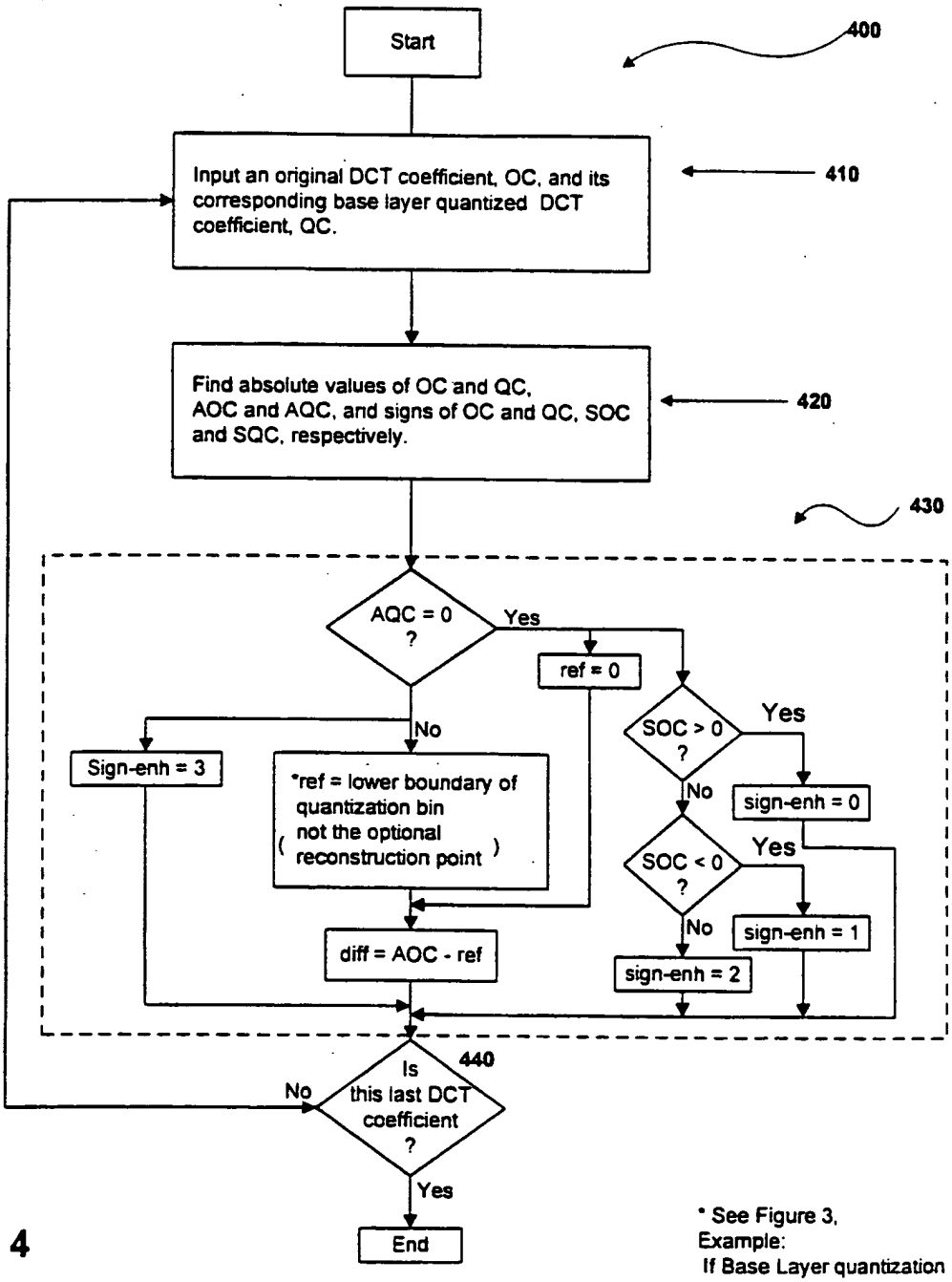


FIG. 4

* See Figure 3,
Example:
If Base Layer quantization is
AQC = AOC / (2*Q)
lower boundary is AQC * (2*Q)
optimal point is AQC*(2*Q) + Q

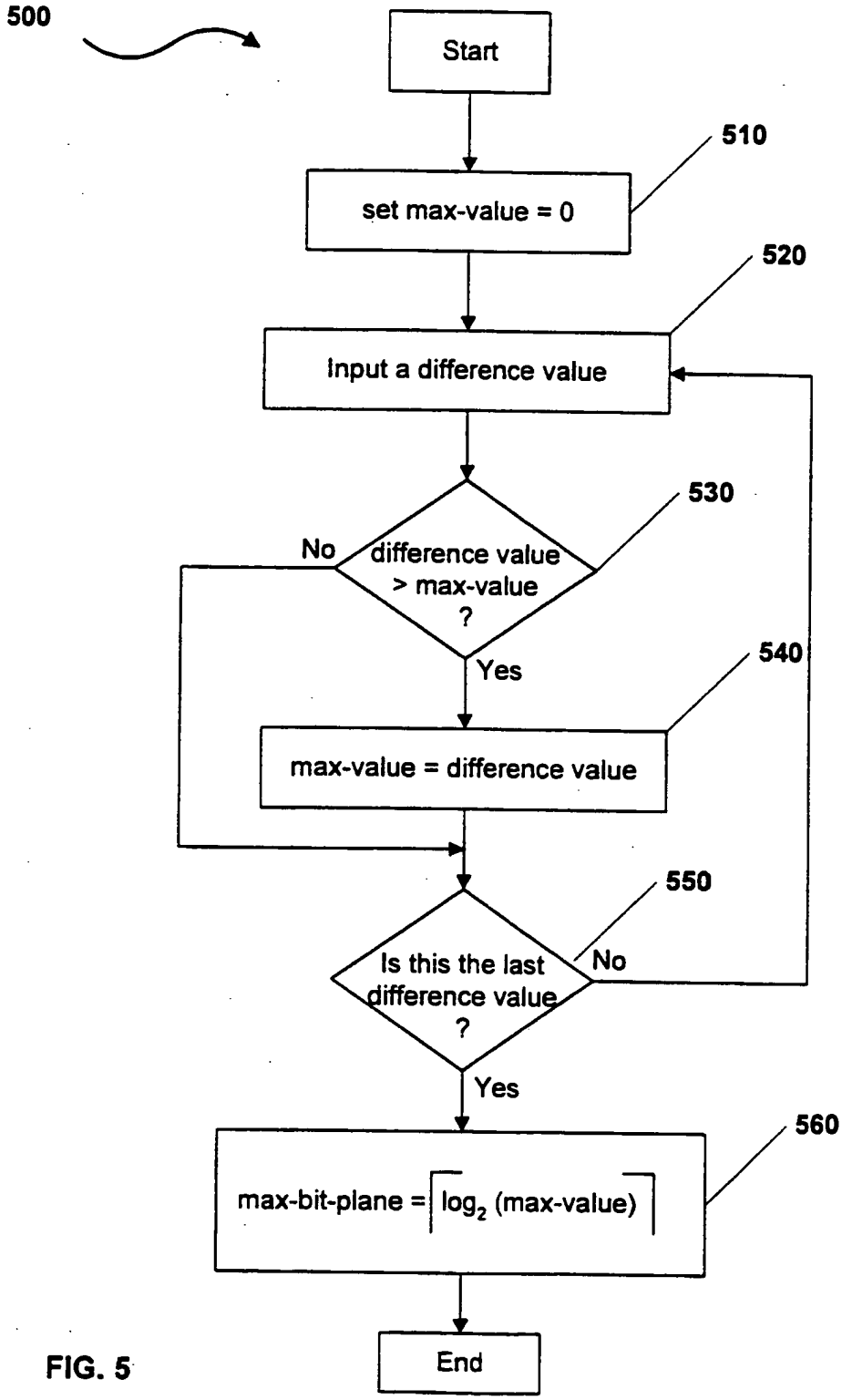

500
Start
510
set max-value = 0
520
Input a difference value
530
difference value > max-value ?
No
Yes
540
max-value = difference value
550
Is this the last difference value ?
No
Yes
560
max-bit-plane = $\lceil \log_2 (\text{max-value}) \rceil$
End

FIG. 5

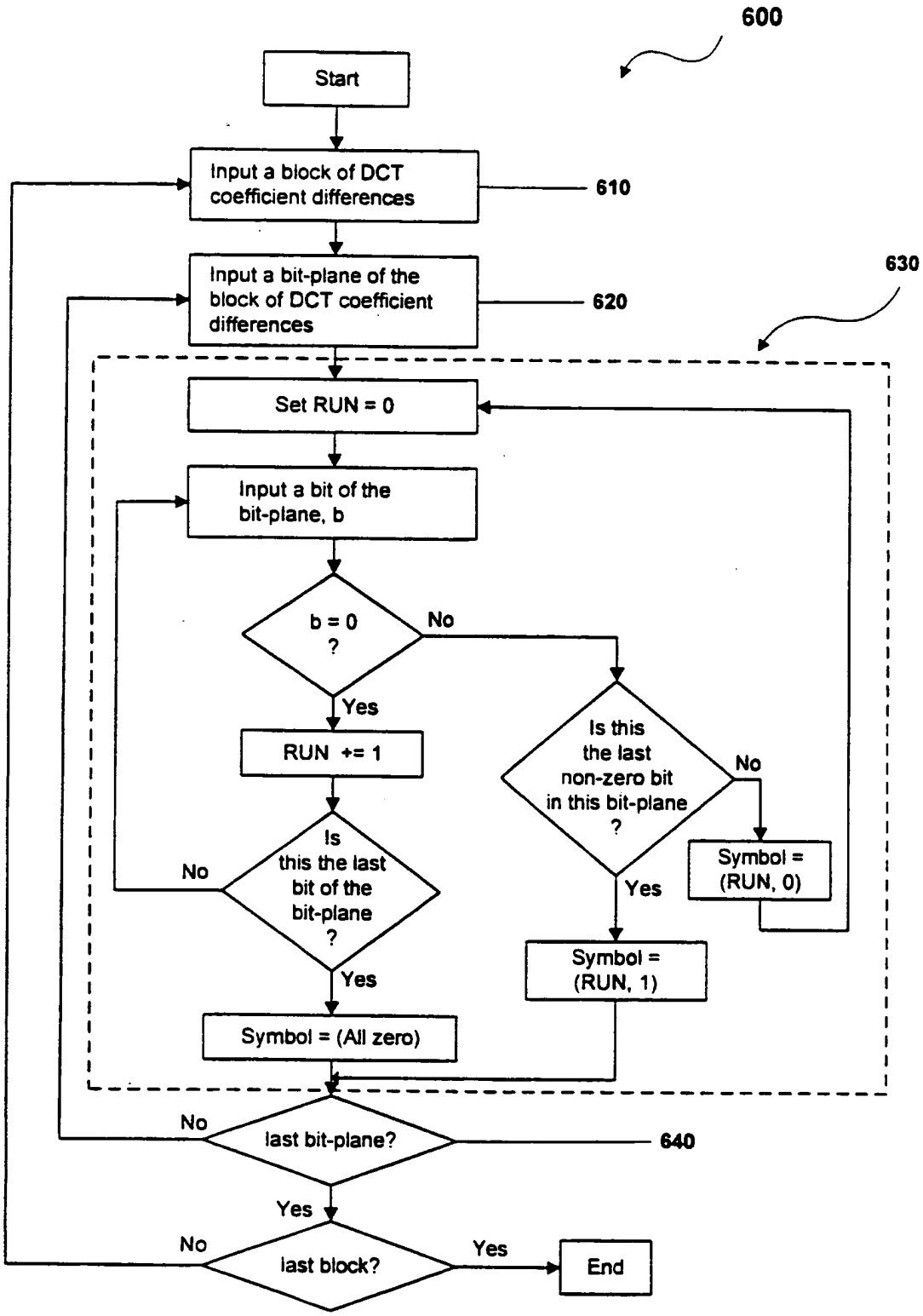

600
Start
Input a block of DCT coefficient differences
610
Input a bit-plane of the block of DCT coefficient differences
620
630
Set RUN = 0
Input a bit of the bit-plane, b
b = 0 ?
No
Yes
RUN += 1
Is this the last non-zero bit in this bit-plane ?
No
Symbol = (RUN, 0)
Yes
Is this the last bit of the bit-plane ?
No
Yes
Symbol = (RUN, 1)
Symbol = (All zero)
No
last bit-plane?
640
Yes
No
last block?
Yes
End

FIG. 6

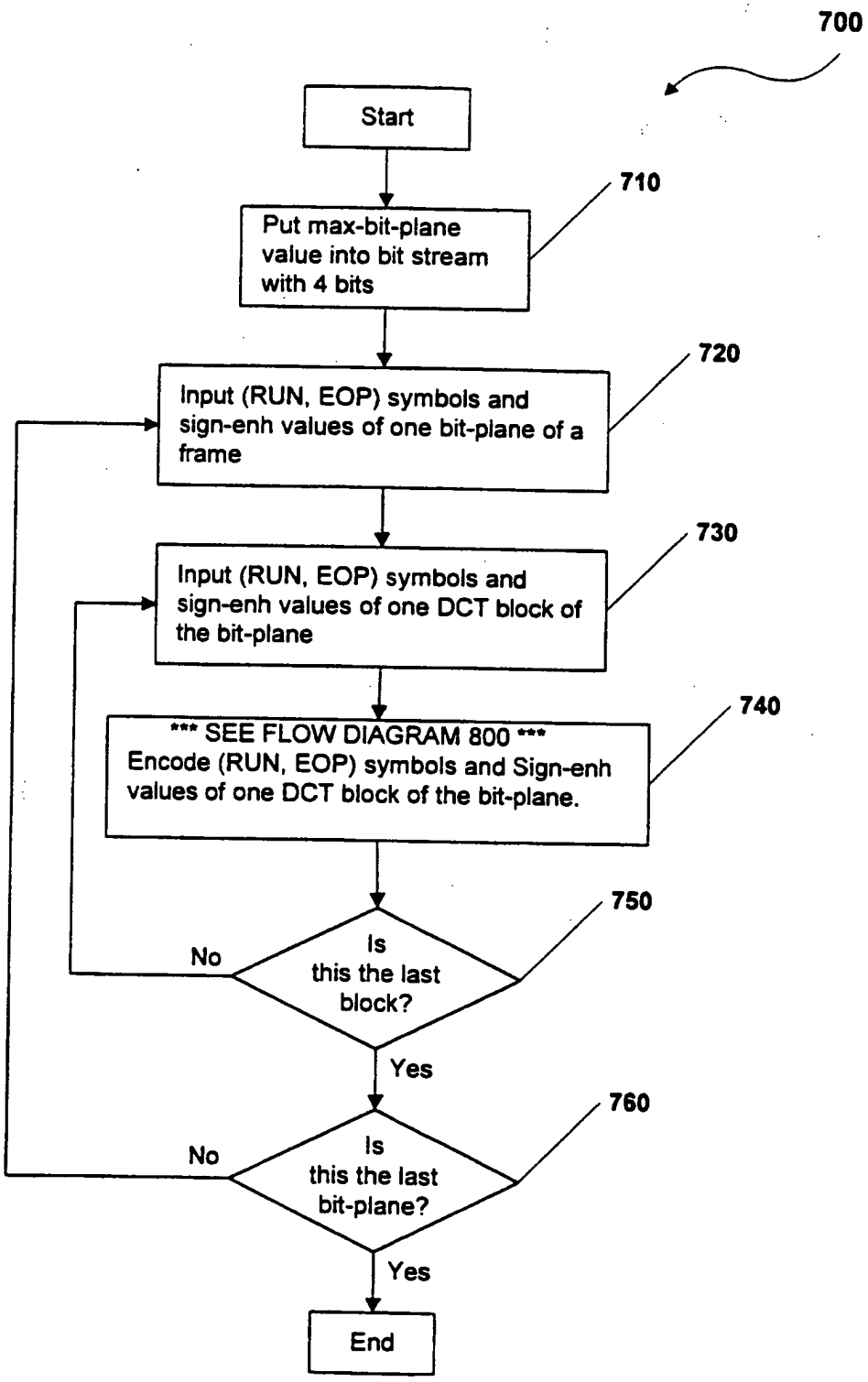

700
Start
710
Put max-bit-plane value into bit stream with 4 bits
720
Input (RUN, EOP) symbols and sign-enh values of one bit-plane of a frame
730
Input (RUN, EOP) symbols and sign-enh values of one DCT block of the bit-plane
740
*** SEE FLOW DIAGRAM 800 ***
Encode (RUN, EOP) symbols and Sign-enh values of one DCT block of the bit-plane.
750
Is this the last block?
No
Yes
760
Is this the last bit-plane?
No
Yes
End

FIG. 7

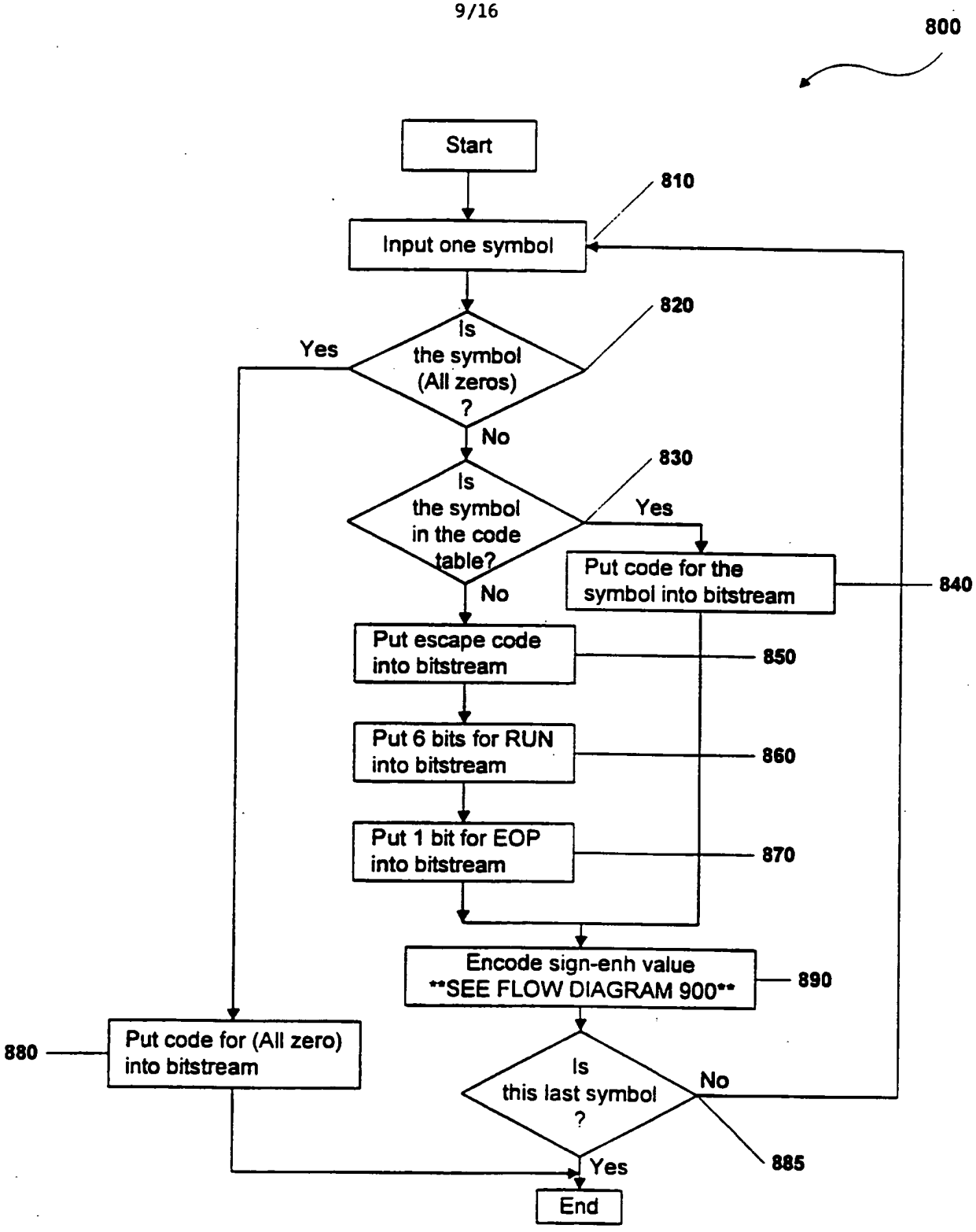

800
Start
810
Input one symbol
820
Is the symbol (All zeros) ?
Yes
No
830
Is the symbol in the code table?
Yes
No
Put code for the symbol into bitstream
840
Put escape code into bitstream
850
Put 6 bits for RUN into bitstream
860
Put 1 bit for EOP into bitstream
870
Encode sign-enh value **SEE FLOW DIAGRAM 900**
890
Put code for (All zero) into bitstream
880
Is this last symbol ?
No
885
Yes
End

FIG. 8

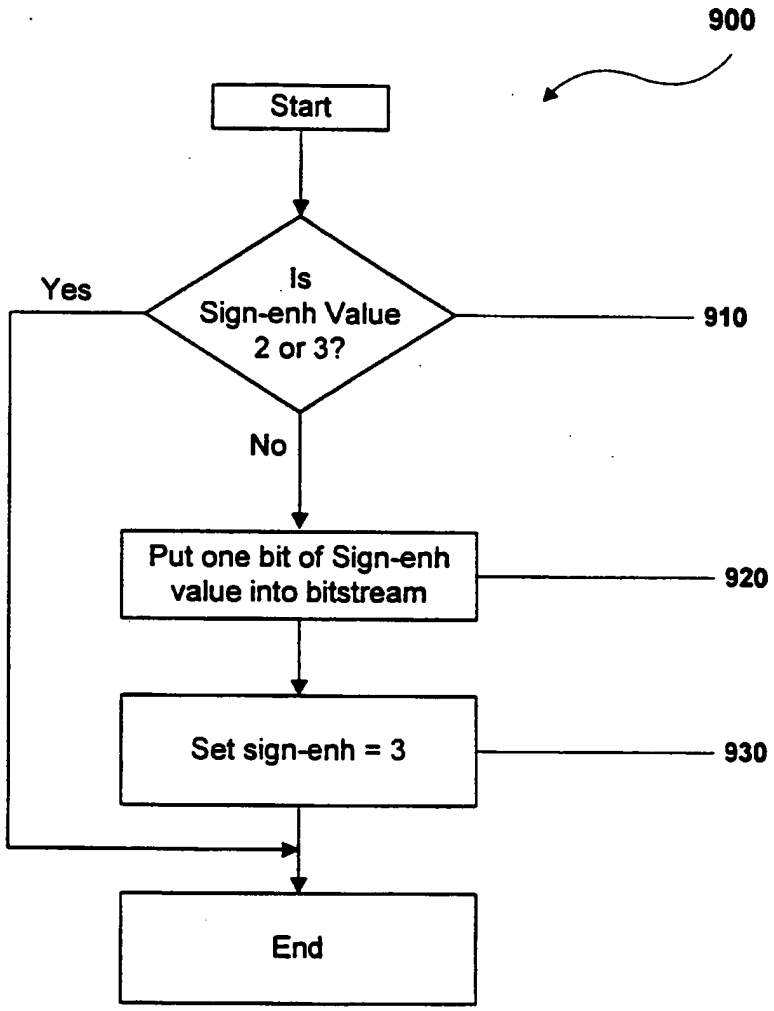

900
Start
Is
Sign-enh Value
2 or 3?
Yes
910
No
Put one bit of Sign-enh
value into bitstream
920
Set sign-enh = 3
930
End

FIG. 9

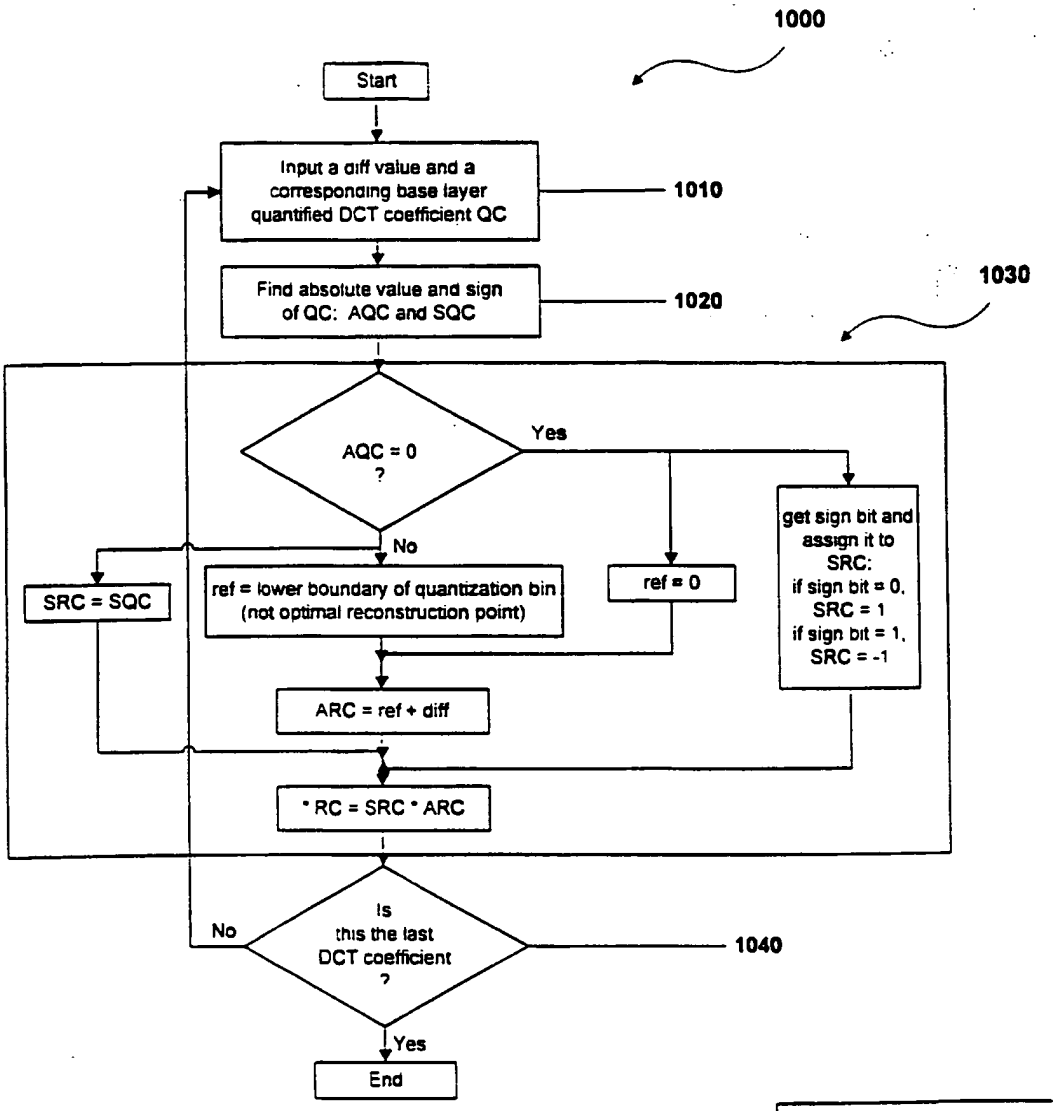

1000
Start
Input a diff value and a corresponding base layer quantified DCT coefficient QC
1010
Find absolute value and sign of QC: AQC and SQC
1020
1030
AQC = 0 ?
Yes
No
SRC = SQC
ref = lower boundary of quantization bin (not optimal reconstruction point)
ref = 0
get sign bit and assign it to SRC:
if sign bit = 0, SRC = 1
if sign bit = 1, SRC = -1
ARC = ref + diff
* RC = SRC * ARC
Is this the last DCT coefficient ?
1040
No
Yes
End
RC is the constructed DCT coefficient.
SRC is the sign of RC and
ARC is the absolute value of RC.

FIG. 10

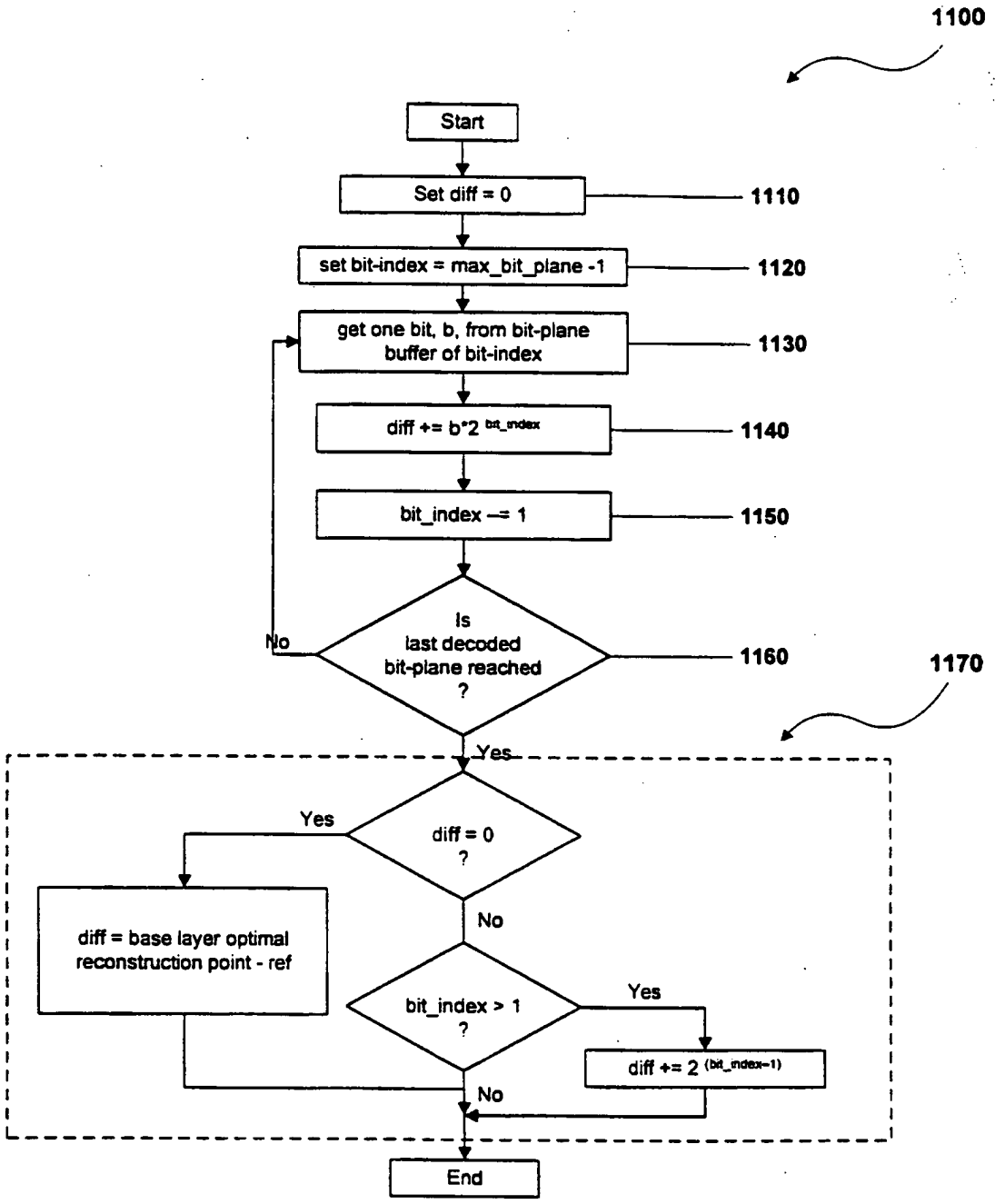

1100
Start
Set diff = 0
1110
set bit-index = max_bit_plane -1
1120
get one bit, b, from bit-plane buffer of bit-index
1130
diff += b*2 bit_index
1140
bit_index -= 1
1150
Is last decoded bit-plane reached ?
1160
No
Yes
1170
diff = 0 ?
Yes
diff = base layer optimal reconstruction point - ref
No
bit_index > 1 ?
Yes
diff += 2 (bit_index-1)
No
End

FIG. 11

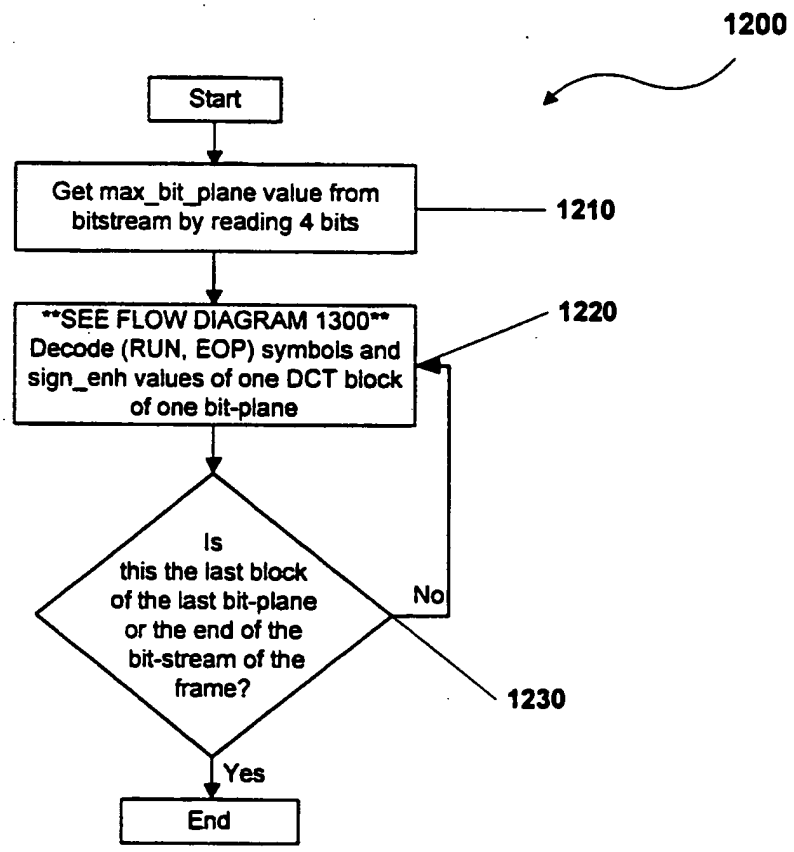

1200
Start
Get max_bit_plane value from bitstream by reading 4 bits
1210
SEE FLOW DIAGRAM 1300 Decode (RUN, EOP) symbols and sign_enh values of one DCT block of one bit-plane
1220
Is this the last block of the last bit-plane or the end of the bit-stream of the frame?
No
1230
Yes
End

FIG. 12

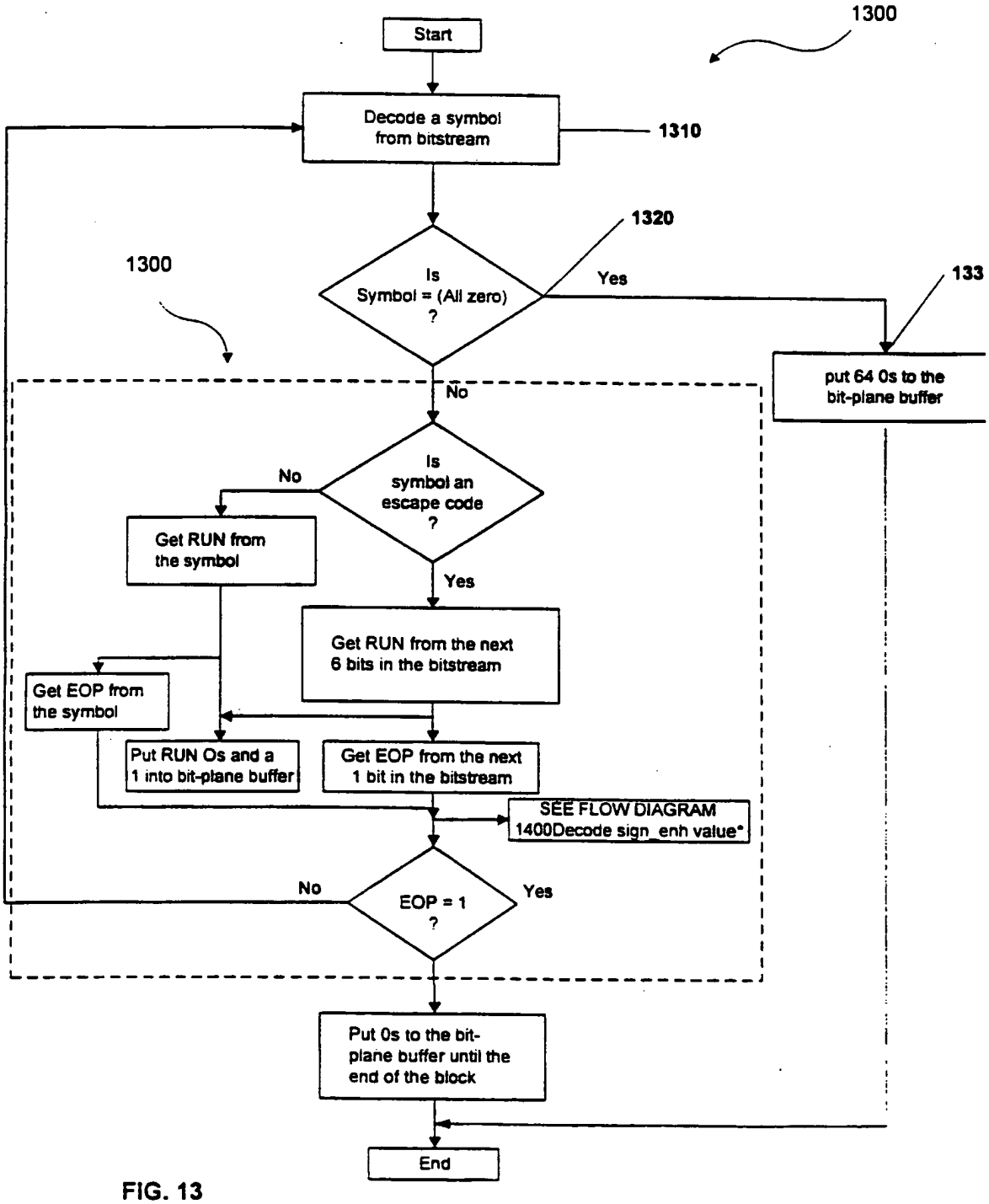

1300
Start
Decode a symbol from bitstream
1310
1320
1300
Is Symbol = (All zero) ?
Yes
133
put 64 0s to the bit-plane buffer
No
Is symbol an escape code ?
No
Get RUN from the symbol
Yes
Get RUN from the next 6 bits in the bitstream
Get EOP from the symbol
Put RUN Os and a 1 into bit-plane buffer
Get EOP from the next 1 bit in the bitstream
SEE FLOW DIAGRAM 1400Decode sign_enh value*
No
EOP = 1 ?
Yes
Put 0s to the bit-plane buffer until the end of the block
End

FIG. 13

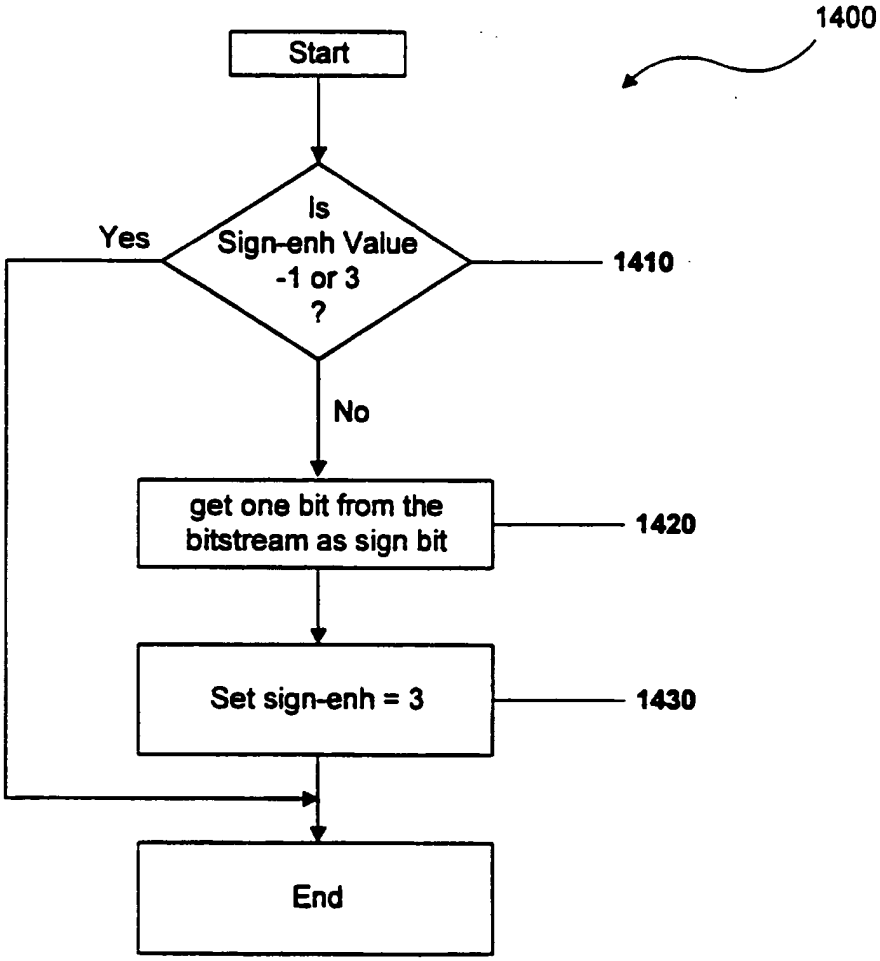

1400
Start
Is Sign-enh Value -1 or 3 ?
Yes
1410
No
get one bit from the bitstream as sign bit
1420
Set sign-enh = 3
1430
End

FIG. 14

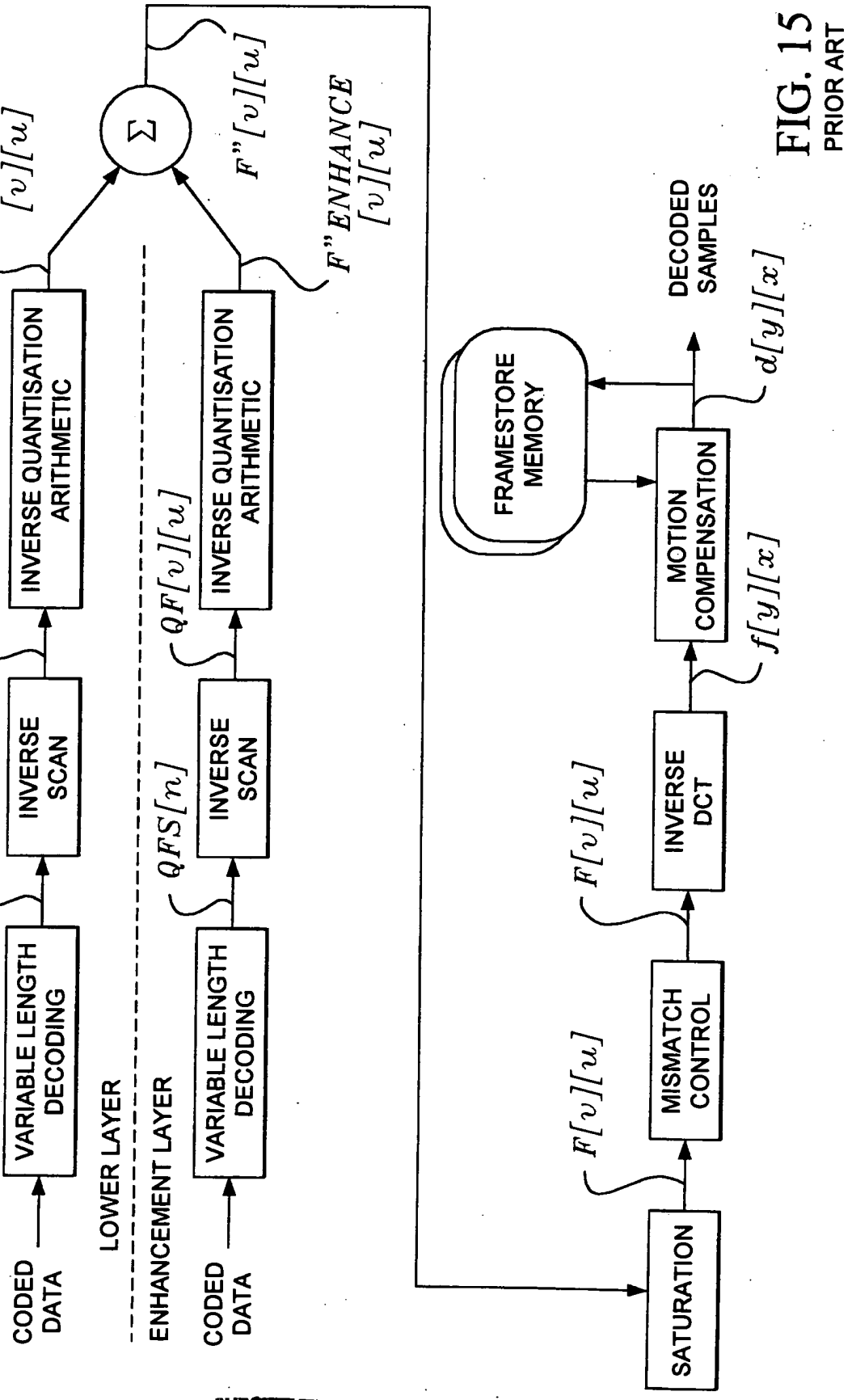


FIG. 15
PRIOR ART

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 9/00

A2

(11) International Publication Number: **WO 00/59152**

(43) International Publication Date: 5 October 2000 (05.10.00)

(21) International Application Number: PCT/US00/04983

(22) International Filing Date: 25 February 2000 (25.02.00)

(30) Priority Data:

60/126,614	27 March 1999 (27.03.99)	US
09/290,363	12 April 1999 (12.04.99)	US
09/482,928	13 January 2000 (13.01.00)	US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmond, WA 98052 (US).

(72) Inventors: BLINN, Arnold, N.; 9401 NE 27th Street, Bellevue, WA 98004 (US). JONES, Thomas, C.; 23617 NE 6th Street, Redmond, WA 98053–3618 (US).

(74) Agents: ROCCI, Steven, J. et al.; Woodcock Washburn Kurtz Mackiewicz & Norris LLP, 46th floor, One Liberty Place, Philadelphia, PA 19103 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

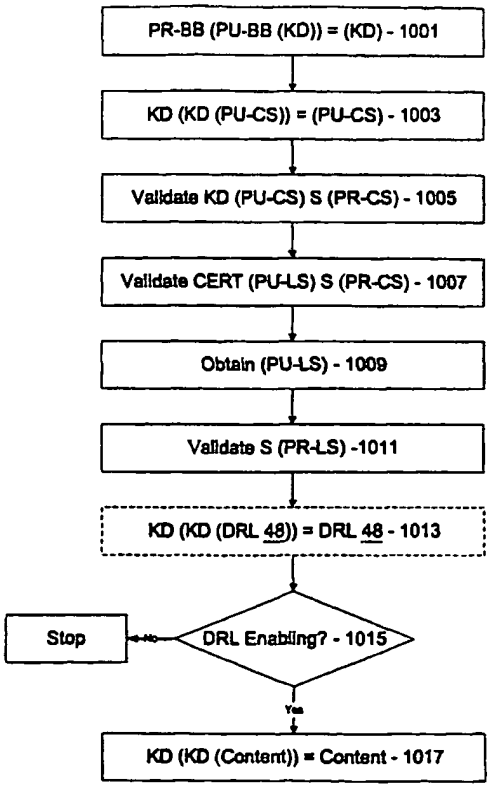
Published

Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD FOR INTERDEPENDENTLY VALIDATING A DIGITAL CONTENT PACKAGE AND A CORRESPONDING DIGITAL LICENSE

(57) Abstract

A method is disclosed for a device to interdependently validate a digital content package having a piece of digital content in an encrypted form, and a corresponding digital license for rendering the digital content. A first key is derived from a source available to the device, and a first digital signature is obtained from the digital content package. The first key is applied to the first digital signature to validate the first digital signature and the digital content package. A second key is derived based on the first digital signature, and a second digital signature is obtained from the license. The second key is applied to the second digital signature to validate the second digital signature and the license.



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.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHOD FOR INTERDEPENDENTLY VALIDATING A DIGITAL CONTENT PACKAGE AND A CORRESPONDING DIGITAL LICENSE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. Patent Application No. 09/290,363, filed April 12, 1999 and entitled "ENFORCEMENT ARCHITECTURE AND METHOD FOR DIGITAL RIGHTS MANAGEMENT", and claims the benefit of U.S. Provisional Application No. 60/21,614, filed March 27, 1999 and entitled "ENFORCEMENT ARCHITECTURE AND METHOD FOR DIGITAL RIGHTS MANAGEMENT", both of which are hereby incorporated by reference.

TECHNICAL FIELD

The present invention relates to an architecture for enforcing rights in digital content. More specifically, the present invention relates to such an enforcement architecture that allows access to encrypted digital content only in accordance with parameters specified by license rights acquired by a user of the digital content.

BACKGROUND OF THE INVENTION

Digital rights management and enforcement is highly desirable in connection with digital content such as digital audio, digital video, digital text, digital data, digital multimedia, etc., where such digital content is to be distributed to users. Typical modes of distribution include tangible devices such as a magnetic (floppy) disk, a magnetic tape, an optical (compact) disk (CD), etc., and intangible media such as an electronic bulletin board, an electronic network, the Internet, etc. Upon being received by the user, such user renders or 'plays' the digital content with the aid of an appropriate rendering device such as a media player on a personal computer or the like.

Typically, a content owner or rights-owner, such as an author, a publisher, a broadcaster, etc. (hereinafter "content owner"), wishes to distribute such digital content to a user or recipient in exchange for a license fee or some other consideration. Such content owner, given the choice, would likely wish to restrict what

the user can do with such distributed digital content. For example, the content owner would like to restrict the user from copying and re-distributing such content to a second user, at least in a manner that denies the content owner a license fee from such second user.

In addition, the content owner may wish to provide the user with the flexibility to purchase different types of use licenses at different license fees, while at the same time holding the user to the terms of whatever type of license is in fact purchased. For example, the content owner may wish to allow distributed digital content to be played only a limited number of times, only for a certain total time, only on a certain type of machine, only on a certain type of media player, only by a certain type of user, etc.

However, after distribution has occurred, such content owner has very little if any control over the digital content. This is especially problematic in view of the fact that practically every new or recent personal computer includes the software and hardware necessary to make an exact digital copy of such digital content, and to download such exact digital copy to a write-able magnetic or optical disk, or to send such exact digital copy over a network such as the Internet to any destination.

Of course, as part of the legitimate transaction where the license fee was obtained, the content owner may require the user of the digital content to promise not to re-distribute such digital content. However, such a promise is easily made and easily broken. A content owner may attempt to prevent such re-distribution through any of several known security devices, usually involving encryption and decryption. However, there is likely very little that prevents a mildly determined user from decrypting encrypted digital content, saving such digital content in an un-encrypted form, and then re-distributing same.

A need exists, then, for providing an enforcement architecture and method that allows the controlled rendering or playing of arbitrary forms of digital content, where such control is flexible and definable by the content owner of such digital content. A need also exists for providing a controlled rendering environment

on a computing device such as a personal computer, where the rendering environment includes at least a portion of such enforcement architecture. Such controlled rendering environment allows that the digital content will only be rendered as specified by the content owner, even though the digital content is to be rendered on a computing device which is not under the control of the content owner.

Further, a need exists for a trusted component running on the computing device, where the trusted component enforces the rights of the content owner on such computing device in connection with a piece of digital content, even against attempts by the user of such computing device to access such digital content in ways not permitted by the content owner. As but one example, such a trusted software component prevents a user of the computing device from making a copy of such digital content, except as otherwise allowed for by the content owner thereof.

SUMMARY OF THE INVENTION

The aforementioned needs are satisfied at least in part by an enforcement architecture and method for digital rights management, where the architecture and method enforce rights in protected (secure) digital content available on a medium such as the Internet, an optical disk, etc. For purposes of making content available, the architecture includes a content server from which the digital content is accessible over the Internet or the like in an encrypted form. The content server may also supply the encrypted digital content for recording on an optical disk or the like, wherein the encrypted digital content may be distributed on the optical disk itself. At the content server, the digital content is encrypted using an encryption key, and public / private key techniques are employed to bind the digital content with a digital license at the user's computing device or client machine.

When a user attempts to render the digital content on a computing device, the rendering application invokes a Digital Rights Management (DRM) system on such user's computing device. If the user is attempting to render the digital content for the first time, the DRM system either directs the user to a license server to obtain a license to render such digital content in the manner sought, or transparently obtains

such license from such license server without any action necessary on the part of the user. The license includes:

- a decryption key (KD) that decrypts the encrypted digital content;
- a description of the rights (play, copy, etc.) conferred by the license and related conditions (begin date, expiration date, number of plays, etc.), where such description is in a digitally readable form; and
- a digital signature that ensures the integrity of the license.

The user cannot decrypt and render the encrypted digital content without obtaining such a license from the license server. The obtained license is stored in a license store in the user's computing device.

Importantly, the license server only issues a license to a DRM system that is 'trusted' (i.e., that can authenticate itself). To implement 'trust', the DRM system is equipped with a 'black box' that performs decryption and encryption functions for such DRM system. The black box includes a public / private key pair, a version number and a unique signature, all as provided by an approved certifying authority. The public key is made available to the license server for purposes of encrypting portions of the issued license, thereby binding such license to such black box. The private key is available to the black box only, and not to the user or anyone else, for purposes of decrypting information encrypted with the corresponding public key. The DRM system is initially provided with a black box with a public / private key pair, and the user is prompted to download from a black box server an updated secure black box when the user first requests a license. The black box server provides the updated black box, along with a unique public/private key pair. Such updated black box is written in unique executable code that will run only on the user's computing device, and is re-updated on a regular basis. When a user requests a license, the client machine sends the black box public key, version number, and signature to the license server, and such license server issues a license only if the version number is current and the signature is valid. A license request also includes an identification of the digital content for which a license is requested and a key ID that identifies the

decryption key associated with the requested digital content. The license server uses the black box public key to encrypt the decryption key, and the decryption key to encrypt the license terms, then downloads the encrypted decryption key and encrypted license terms to the user's computing device along with a license signature.

Once the downloaded license has been stored in the DRM system license store, the user can render the digital content according to the rights conferred by the license and specified in the license terms. When a request is made to render the digital content, the black box is caused to decrypt the decryption key and license terms, and a DRM system license evaluator evaluates such license terms. The black box decrypts the encrypted digital content only if the license evaluation results in a decision that the requestor is allowed to play such content. The decrypted content is provided to the rendering application for rendering.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing summary, as well as the following detailed description of the embodiments of the present invention, will be better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there are shown in the drawings embodiments which are presently preferred. As should be understood, however, the invention is not limited to the precise arrangements and instrumentalities shown. In the drawings:

Fig. 1 is a block diagram showing an enforcement architecture in accordance with one embodiment of the present invention;

Fig. 2 is a block diagram of the authoring tool of the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 3 is a block diagram of a digital content package having digital content for use in connection with the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 4 is a block diagram of the user's computing device of Fig. 1 in accordance with one embodiment of the present invention;

Figs. 5A and 5B are flow diagrams showing the steps performed in connection with the Digital Rights Management (DRM) system of the computing device of Fig. 4 to render content in accordance with one embodiment of the present invention;

Fig. 6 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to determine whether any valid, enabling licenses are present in accordance with one embodiment of the present invention;

Fig. 7 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to obtain a license in accordance with one embodiment of the present invention;

Fig. 8 is a block diagram of a digital license for use in connection with the architecture of Fig. 1 in accordance with one embodiment of the present invention;

Fig. 9 is a flow diagram showing the steps performed in connection with the DRM system of Fig. 4 to obtain a new black box in accordance with one embodiment of the present invention;

Fig. 10 is a flow diagram showing the key transaction steps performed in connection with the DRM system of Fig. 4 to validate a license and a piece of digital content and render the content in accordance with one embodiment of the present invention;

Fig. 11 is a block diagram showing the license evaluator of Fig. 4 along with a Digital Rights License (DRL) of a license and a language engine for interpreting the DRL in accordance with one embodiment of the present invention; and

Fig. 12 is a block diagram representing a general purpose computer system in which aspects of the present invention and/or portions thereof may be incorporated.

Detailed Description of the Invention

Referring to the drawings in details, wherein like numerals are used to indicate like elements throughout, there is shown in Fig. 1 an enforcement architecture 10 in accordance with one embodiment of the present invention. Overall, the enforcement architecture 10 allows an owner of digital content 12 to specify license rules that must be satisfied before such digital content 12 is allowed to be rendered on a user's computing device 14. Such license rules are embodied within a digital license 16 that the user / user's computing device 14 (hereinafter, such terms are interchangeable unless circumstances require otherwise) must obtain from the content owner or an agent thereof. The digital content 12 is distributed in an encrypted form, and may be distributed freely and widely. Preferably, the decrypting key (KD) for decrypting the digital content 12 is included with the license 16.

COMPUTER ENVIRONMENT

Fig. 12 and the following discussion are intended to provide a brief general description of a suitable computing environment in which the present invention and/or portions thereof may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, it should be appreciated that the invention and/or portions thereof may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

As shown in Fig. 12, an exemplary general purpose computing system

includes a conventional personal computer 120 or the like, including a processing unit 121, a system memory 122, and a system bus 18 that couples various system components including the system memory to the processing unit 121. The system bus 18 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read-only memory (ROM) 19 and random access memory (RAM) 20. A basic input/output system 21 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 120, such as during start-up, is stored in ROM 19.

The personal computer 120 may further include a hard disk drive 22 for reading from and writing to a hard disk (not shown), a magnetic disk drive 128 for reading from or writing to a removable magnetic disk 129, and an optical disk drive 25 for reading from or writing to a removable optical disk 131 such as a CD-ROM or other optical media. The hard disk drive 22, magnetic disk drive 128, and optical disk drive 25 are connected to the system bus 18 by a hard disk drive interface 27, a magnetic disk drive interface 28, and an optical drive interface 29, respectively. The drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 129, and a removable optical disk 131, it should be appreciated that other types of computer readable media which can store data that is accessible by a computer may also be used in the exemplary operating environment. Such other types of media include a magnetic cassette, a flash memory card, a digital video disk, a Bernoulli cartridge, a random access memory (RAM), a read-only memory (ROM), and the like.

A number of program modules may be stored on the hard disk, magnetic disk 129, optical disk 131, ROM 19 or RAM 20, including an operating system 30, one or more application programs 136, other program modules 137 and

program data 138. A user may enter commands and information into the personal computer 120 through input devices such as a keyboard 35 and pointing device 142. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner, or the like. These and other input devices are often connected to the processing unit 121 through a serial port interface 41 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or universal serial bus (USB). A monitor 42 or other type of display device is also connected to the system bus 18 via an interface, such as a video adapter 148. In addition to the monitor 42, a personal computer typically includes other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 12 also includes a host adapter 50, a Small Computer System Interface (SCSI) bus 156, and an external storage device 162 connected to the SCSI bus 156.

The personal computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 149. The remote computer 149 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 120, although only a memory storage device 150 has been illustrated in Fig. 12. The logical connections depicted in Fig. 12 include a local area network (LAN) 46 and a wide area network (WAN) 47. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the personal computer 120 is connected to the LAN 46 through a network interface or adapter 48. When used in a WAN networking environment, the personal computer 120 typically includes a modem 49 or other means for establishing communications over the wide area network 47, such as the Internet. The modem 49, which may be internal or external, is connected to the system bus 18 via the serial port interface 41. In a networked environment, program modules depicted relative to the personal computer 120, or portions thereof, may be stored in the remote memory storage device. It will be

appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

ARCHITECTURE

Referring again to Fig. 1, in one embodiment of the present invention, the architecture 10 includes an authoring tool 18, a content-key database 20, a content server 22, a license server 24, and a black box server 26, as well as the aforementioned user's computing device 14.

ARCHITECTURE - Authoring Tool 18

The authoring tool 18 is employed by a content owner to package a piece of digital content 12 into a form that is amenable for use in connection with the architecture 10 of the present invention. In particular, the content owner provides the authoring tool 18 with the digital content 12, instructions and/or rules that are to accompany the digital content 12, and instructions and/or rules as to how the digital content 12 is to be packaged. The authoring tool 18 then produces a digital content package 12p having the digital content 12 encrypted according to an encryption / decryption key, and the instructions and/or rules that accompany the digital content 12.

In one embodiment of the present invention, the authoring tool 18 is instructed to serially produce several different digital content 12 packages 12p, each having the same digital content 12 encrypted according to a different encryption / decryption key. As should be understood, having several different packages 12p with the same digital content 12 may be useful for tracking the distribution of such packages 12p / content 12 (hereinafter simply "digital content 12", unless circumstances require otherwise). Such distribution tracking is not ordinarily necessary, but may be used by an investigative authority in cases where the digital content 12 has been illegally sold or broadcast.

In one embodiment of the present invention, the encryption / decryption key that encrypts the digital content 12 is a symmetric key, in that the encryption key is also the decryption key (KD). As will be discussed below in more detail, such decryption key (KD) is delivered to a user's computing device 14 in a hidden form as

part of a license 16 for such digital content 12. Preferably, each piece of digital content 12 is provided with a content ID (or each package 12p is provided with a package ID), each decryption key (KD) has a key ID, and the authoring tool 18 causes the decryption key (KD), key ID, and content ID (or package ID) for each piece of digital content 12 (or each package 12p) to be stored in the content-key database 20. In addition, license data regarding the types of licenses 16 to be issued for the digital content 12 and the terms and conditions for each type of license 16 may be stored in the content-key database 20, or else in another database (not shown). Preferably, the license data can be modified by the content owner at a later time as circumstances and market conditions may require.

In use, the authoring tool 18 is supplied with information including, among other things:

- the digital content 12 to be packaged;
- the type and parameters of watermarking and/or fingerprinting to be employed, if any;
- the type and parameters of data compression to be employed, if any;
- the type and parameters of encryption to be employed;
- the type and parameters of serialization to be employed, if any; and
- the instructions and/or rules that are to accompany the digital content 12.

As is known, a watermark is a hidden, computer-readable signal that is added to the digital content 12 as an identifier. A fingerprint is a watermark that is different for each instance. As should be understood, an instance is a version of the digital content 12 that is unique. Multiple copies of any instance may be made, and any copy is of a particular instance. When a specific instance of digital content 12 is illegally sold or broadcast, an investigative authority can perhaps identify suspects according to the watermark / fingerprint added to such digital content 12.

Data compression may be performed according to any appropriate compression algorithm without departing from the spirit and scope of the present

invention. For example, the .mp3 or .wav compression algorithm may be employed. Of course, the digital content 12 may already be in a compressed state, in which case no additional compression is necessary.

The instructions and/or rules that are to accompany the digital content 12 may include practically any appropriate instructions, rules, or other information without departing from the spirit and scope of the present invention. As will be discussed below, such accompanying instructions / rules / information are primarily employed by the user and the user's computing device 14 to obtain a license 16 to render the digital content 12. Accordingly, such accompanying instructions / rules / information may include an appropriately formatted license acquisition script or the like, as will be described in more detail below. In addition, or in the alternative, such accompanying instructions / rules / information may include 'preview' information designed to provide a user with a preview of the digital content 12.

With the supplied information, the authoring tool 18 then produces one or more packages 12p corresponding to the digital content 12. Each package 12p may then be stored on the content server 22 for distribution to the world.

In one embodiment of the present invention, and referring now to Fig. 2, the authoring tool 18 is a dynamic authoring tool 18 that receives input parameters which can be specified and operated on. Accordingly, such authoring tool 18 can rapidly produce multiple variations of package 12p for multiple pieces of digital content 12. Preferably, the input parameters are embodied in the form of a dictionary 28, as shown, where the dictionary 28 includes such parameters as:

- the name of the input file 29a having the digital content 12;
- the type of encoding that is to take place
- the encryption / decryption key (KD) to be employed,
- the accompanying instructions / rules / information ('header information') to be packaged with the digital content 12 in the package 12p.
- the type of muxing that is to occur: and

- the name of the output file 29b to which the package 12p based on the digital content 12 is to be written.

As should be understood, such dictionary 28 is easily and quickly modifiable by an operator of the authoring tool 18 (human or machine), and therefore the type of authoring performed by the authoring tool 18 is likewise easily and quickly modifiable in a dynamic manner. In one embodiment of the present invention, the authoring tool 18 includes an operator interface (not shown) displayable on a computer screen to a human operator. Accordingly, such operator may modify the dictionary 28 by way of the interface, and further may be appropriately aided and/or restricted in modifying the dictionary 28 by way of the interface.

In the authoring tool 18, and as seen in Fig. 2, a source filter 18a receives the name of the input file 29a having the digital content 12 from the dictionary 28, and retrieves such digital content 12 from such input file and places the digital content 12 into a memory 29c such as a RAM or the like. An encoding filter 18b then performs encoding on the digital content 12 in the memory 29c to transfer the file from the input format to the output format according to the type of encoding specified in the dictionary 28 (i.e., .wav to .asp, .mp3 to .asp, etc.), and places the encoded digital content 12 in the memory 29c. As shown, the digital content 12 to be packaged (music, e.g.) is received in a compressed format such as the .wav or .mp3 format, and is transformed into a format such as the .asp (active streaming protocol) format. Of course, other input and output formats may be employed without departing from the spirit and scope of the present invention.

Thereafter, an encryption filter 18c encrypts the encoded digital content 12 in the memory 29c according to the encryption / decryption key (KD) specified in the dictionary 28, and places the encrypted digital content 12 in the memory 29c. A header filter 18d then adds the header information specified in the dictionary 28 to the encrypted digital content 12 in the memory 29c.

As should be understood, depending on the situation, the package 12p may include multiple streams of temporally aligned digital content 12 (one stream

being shown in Fig. 2), where such multiple streams are multiplexed (i.e., 'muxed'). Accordingly, a mux filter 18e performs muxing on the header information and encrypted digital content 12 in the memory 29c according to the type of muxing specified in the dictionary 28, and places the result in the memory 29c. A file writer filter 18f then retrieves the result from the memory 29c and writes such result to the output file 29b specified in the dictionary 28 as the package 12p.

It should be noted that in certain circumstances, the type of encoding to be performed will not normally change. Since the type of muxing typically is based on the type of encoding, it is likewise the case that the type of muxing will not normally change, either. If this is in fact the case, the dictionary 28 need not include parameters on the type of encoding and/or the type of muxing. Instead, it is only necessary that the type of encoding be 'hardwired' into the encoding filter and/or that the type of muxing be 'hardwired' into the mux filter. Of course, as circumstance require, the authoring tool 18 may not include all of the aforementioned filters, or may include other filters, and any included filter may be hardwired or may perform its function according to parameters specified in the dictionary 28, all without departing from the spirit and scope of the present invention.

Preferably, the authoring tool 18 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure.

ARCHITECTURE - Content Server 22

Referring again to Fig. 1, in one embodiment of the present invention, the content server 22 distributes or otherwise makes available for retrieval the packages 12p produced by the authoring tool 18. Such packages 12p may be distributed as requested by the content server 22 by way of any appropriate distribution channel without departing from the spirit and scope of the present invention. For example, such distribution channel may be the Internet or another network, an electronic bulletin

board, electronic mail, or the like. In addition, the content server 22 may be employed to copy the packages 12p onto magnetic or optical disks or other storage devices, and such storage devices may then be distributed.

It will be appreciated that the content server 22 distributes packages 12p without regard to any trust or security issues. As discussed below, such issues are dealt with in connection with the license server 24 and the relationship between such license server 24 and the user's computing device 14. In one embodiment of the present invention, the content server 22 freely releases and distributes packages 12p having digital content 12 to any distributee requesting same. However, the content server 22 may also release and distribute such packages 12p in a restricted manner without departing from the spirit and scope of the present invention. For example, the content server 22 may first require payment of a pre-determined distribution fee prior to distribution, or may require that a distributee identify itself, or may indeed make a determination of whether distribution is to occur based on an identification of the distributee.

In addition, the content server 22 may be employed to perform inventory management by controlling the authoring tool 18 to generate a number of different packages 12p in advance to meet an anticipated demand. For example, the server could generate 100 packages 12p based on the same digital content 12, and serve each package 12p 10 times. As supplies of packages 12p dwindle to 20, for example, the content server 22 may then direct the authoring tool 18 to generate 80 additional packages 12p, again for example.

Preferably, the content server 22 in the architecture 10 has a unique public / private key pair (PU-CS, PR-CS) that is employed as part of the process of evaluating a license 16 and obtaining a decryption key (KD) for decrypting corresponding digital content 12, as will be explained in more detail below. As is known, a public / private key pair is an asymmetric key, in that what is encrypted in one of the keys in the key pair can only be decrypted by the other of the keys in the key pair. In a public / private key pair encryption system, the public key may be made

known to the world, but the private key should always be held in confidence by the owner of such private key. Accordingly, if the content server 22 encrypts data with its private key (PR-CS), it can send the encrypted data out into the world with its public key (PU-CS) for decryption purposes. Correspondingly, if an external device wants to send data to the content server 22 so that only such content server 22 can decrypt such data, such external device must first obtain the public key of the content server 22 (PU-CS) and then must encrypt the data with such public key. Accordingly, the content server 22 (and only the content server 22) can then employ its private key (PR-CS) to decrypt such encrypted data.

As with the authoring tool 18, the content server 22 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention, the authoring tool 18 and the content server 22 may reside on a single computer, processor, or other computing machine, each in a separate work space. It should be recognized, moreover, that the content server 22 may in certain circumstances include the authoring tool 18 and/or perform the functions of the authoring tool 18, as discussed above.

Structure of Digital Content Package 12p

Referring now to Fig. 3, in one embodiment of the present invention, the digital content package 12p as distributed by the content server 22 includes:

- the digital content 12 encrypted with the encryption / decryption key (KD), as was discussed above (i.e., (KD(CONTENT)));
- the content ID (or package ID) of such digital content 12 (or package 12p);
- the key ID of the decryption key (KD):
- license acquisition information, preferably in an un-encrypted form; and

- the key KD encrypting the content server 22 public key (PU-CS), signed by the content server 22 private key (PR-CS) (i.e., (KD (PU-CS) S (PR-CS))).

With regard to (KD (PU-CS) S (PR-CS)), it is to be understood that such item is to be used in connection with validating the digital content 12 and/or package 12p, as will be explained below. Unlike a certificate with a digital signature (see below), the key (PU-CS) is not necessary to get at (KD (PU-CS)). Instead, the key (PU-CS) is obtained merely by applying the decryption key (KD). Once so obtained, such key (PU-CS) may be employed to test the validity of the signature (S (PR-CS)).

It should also be understood that for such package 12p to be constructed by the authoring tool 18, such authoring tool 18 must already possess the license acquisition information and (KD (PU-CS) S (PR-CS)), presumably as header information supplied by the dictionary 28. Moreover, the authoring tool 18 and the content server 22 must presumably interact to construct (KD (PU-CS) S (PR-CS)). Such interaction may for example include the steps of:

- the content server 22 sending (PU-CS) to the authoring tool 18;
- the authoring tool 18 encrypting (PU-CS) with (KD) to produce (KD (PU-CS));
- the authoring tool 18 sending (KD (PU-CS)) to the content server 22;
- the content server 22 signing (KD (PU-CS)) with (PR-CS) to produce (KD (PU-CS) S (PR-CS)); and
- the content server 22 sending (KD (PU-CS) S (PR-CS)) to the authoring tool 18.

ARCHITECTURE - License Server 24

Referring again to Fig. 1, in one embodiment of the present invention, the license server 24 performs the functions of receiving a request for a license 16 from a user's computing device 14 in connection with a piece of digital content 12,

determining whether the user's computing device 14 can be trusted to honor an issued license 16, negotiating such a license 16, constructing such license 16, and transmitting such license 16 to the user's computing device 14. Preferably, such transmitted license 16 includes the decryption key (KD) for decrypting the digital content 12. Such license server 24 and such functions will be explained in more detail below. Preferably, and like the content server 22, the license server 24 in the architecture 10 has a unique public / private key pair (PU-LS, PR-LS) that is employed as part of the process of evaluating a license 16 and obtaining a decryption key (KD) for decrypting corresponding digital content 12, as will be explained in more detail below.

As with the authoring tool 18 and the content server 22, the license server 24 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention the authoring tool 18 and/or the content server 22 may reside on a single computer, processor, or other computing machine together with the license server 24, each in a separate work space.

In one embodiment of the present invention, prior to issuance of a license 16, the license server 24 and the content server 22 enter into an agency agreement or the like, wherein the license server 24 in effect agrees to be the licensing authority for at least a portion of the digital content 12 distributed by the content server 22. As should be understood, one content server 22 may enter into an agency agreement or the like with several license servers 24, and/or one license server 24 may enter into an agency agreement or the like with several content servers 22, all without departing from the spirit and scope of the present invention.

Preferably, the license server 24 can show to the world that it does in fact have the authority to issue a license 16 for digital content 12 distributed by the content server 22. To do so, it is preferable that the license server 24 send to the content server 22 the license server 24 public key (PU-LS), and that the content server

22 then send to the license server 24 a digital certificate containing PU-LS as the contents signed by the content server 22 private key (CERT (PU-LS) S (PR-CS)). As should be understood, the contents (PU-LS) in such certificate can only be accessed with the content server 22 public key (PU-CS). As should also be understood, in general, a digital signature of underlying data is an encrypted form of such data, and will not match such data when decrypted if such data has been adulterated or otherwise modified.

As a licensing authority in connection with a piece of digital content 12, and as part of the licensing function, the license server 24 must have access to the decryption key (KD) for such digital content 12. Accordingly, it is preferable that license server 24 have access to the content-key database 20 that has the decryption key (KD), key ID, and content ID (or package ID) for such digital content 12 (or package 12p).

ARCHITECTURE - Black Box Server 26

Still referring to Fig. 1, in one embodiment of the present invention, the black box server 26 performs the functions of installing and/or upgrading a new black box 30 in a user's computing device 14. As will be explained in more detail below, the black box 30 performs encryption and decryption functions for the user's computing device 14. As will also be explained in more detail below, the black box 30 is intended to be secure and protected from attack. Such security and protection is provided, at least in part, by upgrading the black box 30 to a new version as necessary by way of the black box server 26, as will be explained in more detail below.

As with the authoring tool 18, the content server 22, and the license server 24, the black box server 26 is implemented on an appropriate computer, processor, or other computing machine by way of appropriate software. The structure and operation of such machine and such software should be apparent based on the disclosure herein and therefore do not require any detailed discussion in the present disclosure. Moreover, in one embodiment of the present invention the license server 24, the authoring tool 18, and/or the content server 22 may reside on a single computer.

processor, or other computing machine together with the black box server 26, each in a separate work space. Note, though, that for security purposes, it may be wise to have the black box server 26 on a separate machine.

ARCHITECTURE - User's Computing Device 14

Referring now to Fig. 4, in one embodiment of the present invention, the user's computing device 14 is a personal computer or the like, having elements including a keyboard, a mouse, a screen, a processor, RAM, ROM, a hard drive, a floppy drive, a CD player, and/or the like. However, the user's computing device 14 may also be a dedicated viewing device such as a television or monitor, a dedicated audio device such as a stereo or other music player, a dedicated printer, or the like, among other things, all without departing from the spirit and scope of the present invention.

The content owner for a piece of digital content 12 must trust that the user's computing device 14 will abide by the rules specified by such content owner, i.e. that the digital content 12 will not be rendered unless the user obtains a license 16 that permits the rendering in the manner sought. Preferably, then, the user's computing device 14 must provide a trusted component or mechanism 32 that can satisfy to the content owner that such computing device 14 will not render the digital content 12 except according to the license rules embodied in the license 16 associated with the digital content 12 and obtained by the user.

Here, the trusted mechanism 32 is a Digital Rights Management (DRM) system 32 that is enabled when a user requests that a piece of digital content 12 be rendered, that determines whether the user has a license 16 to render the digital content 12 in the manner sought, that effectuates obtaining such a license 16 if necessary, that determines whether the user has the right to play the digital content 12 according to the license 16, and that decrypts the digital content 12 for rendering purposes if in fact the user has such right according to such license 16. The contents and function of the DRM system 32 on the user's computing device 14 and in connection with the architecture 10 are described below.

DRM SYSTEM 32

The DRM system 32 performs four main functions with the architecture 10 disclosed herein: (1) content acquisition, (2) license acquisition, (3) content rendering, and (4) black box 30 installation / update. Preferably, any of the functions can be performed at any time, although it is recognized that some of the functions already require that digital content 12 be acquired.

DRM SYSTEM 32 - Content Acquisition

Acquisition of digital content 12 by a user and/or the user's computing device 14 is typically a relatively straight-forward matter and generally involves placing a file having encrypted digital content 12 on the user's computing device 14. Of course, to work with the architecture 10 and the DRM system 32 disclosed herein, it is necessary that the encrypted digital content 12 be in a form that is amenable to such architecture 10 and DRM system 32, such as the digital package 12p as will be described below.

As should be understood, the digital content 12 may be obtained in any manner from a content server 22, either directly or indirectly, without departing from the spirit and scope of the present invention. For example, such digital content 12 may be downloaded from a network such as the Internet, located on an obtained optical or magnetic disk or the like, received as part of an E-mail message or the like, or downloaded from an electronic bulletin board or the like.

Such digital content 12, once obtained, is preferably stored in a manner such that the obtained digital content 12 is accessible by a rendering application 34 (to be described below) running on the computing device 14, and by the DRM system 32. For example, the digital content 12 may be placed as a file on a hard drive (not shown) of the user's computing device 14, or on a network server (not shown) accessible to the computing device 14. In the case where the digital content 12 is obtained on an optical or magnetic disk or the like, it may only be necessary that such disk be present in an appropriate drive (not shown) coupled to the user's computing device 14.

In the present invention, it is not envisioned that any special tools are

necessary to acquire digital content 12, either from the content server 22 as a direct distribution source or from some intermediary as an indirect distribution source. That is, it is preferable that digital content 12 be as easily acquired as any other data file. However, the DRM system 32 and/or the rendering application 34 may include an interface (not shown) designed to assist the user in obtaining digital content 12 . For example, the interface may include a web browser especially designed to search for digital content 12, links to pre-defined Internet web sites that are known to be sources of digital content 12, and the like.

DRM SYSTEM 32 - Content Rendering, Part 1

Referring now to Fig. 5A, in one embodiment of the present invention, assuming the encrypted digital content 12 has been distributed to and received by a user and placed by the user on the computing device 14 in the form of a stored file, the user will attempt to render the digital content 12 by executing some variation on a render command (step 501). For example, such render command may be embodied as a request to 'play' or 'open' the digital content 12. In some computing environments, such as for example the "MICROSOFT WINDOWS" operating system, distributed by MICROSOFT Corporation of Redmond, Washington, such play or open command may be as simple as 'clicking' on an icon representative of the digital content 12. Of course, other embodiments of such render command may be employed without departing from the spirit and scope of the present invention. In general, such render command may be considered to be executed whenever a user directs that a file having digital content 12 be opened, run, executed, and/or the like.

Importantly, and in addition, such render command may be embodied as a request to copy the digital content 12 to another form, such as to a printed form, a visual form, an audio form, etc. As should be understood, the same digital content 12 may be rendered in one form, such as on a computer screen, and then in another form, such as a printed document. In the present invention, each type of rendering is performed only if the user has the right to do so, as will be explained below.

In one embodiment of the present invention, the digital content 12 is in

the form of a digital file having a file name ending with an extension, and the computing device 14 can determine based on such extension to start a particular kind of rendering application 34. For example, if the file name extension indicates that the digital content 12 is a text file, the rendering application 34 is some form of word processor such as the "MICROSOFT WORD", distributed by MICROSOFT Corporation of Redmond, Washington. Likewise, if the file name extension indicates that the digital content 12 is an audio, video, and/or multimedia file, the rendering application 34 is some form of multimedia player, such as "MICROSOFT MEDIA PLAYER", also distributed by MICROSOFT Corporation of Redmond, Washington.

Of course, other methods of determining a rendering application may be employed without departing from the spirit and scope of the present invention. As but one example, the digital content 12 may contain meta-data in an un-encrypted form (i.e., the aforementioned header information), where the meta-data includes information on the type of rendering application 34 necessary to render such digital content 12.

Preferably, such rendering application 34 examines the digital content 12 associated with the file name and determines whether such digital content 12 is encrypted in a rights-protected form (steps 503, 505). If not protected, the digital content 12 may be rendered without further ado (step 507). If protected, the rendering application 34 determines from the encrypted digital content 12 that the DRM system 32 is necessary to play such digital content 12. Accordingly, such rendering application 34 directs the user's computing device 14 to run the DRM system 32 thereon (step 509). Such rendering application 34 then calls such DRM system 32 to decrypt the digital content 12 (step 511). As will be discussed in more detail below, the DRM system 32 in fact decrypts the digital content 12 only if the user has a valid license 16 for such digital content 12 and the right to play the digital content 12 according to the license rules in the valid license 16. Preferably, once the DRM system 32 has been called by the rendering application 34, such DRM system 32 assumes control from the rendering application 34, at least for purposes of determining whether

the user has a right to play such digital content 12 (step 513).

DRM System 32 Components

In one embodiment of the present invention, and referring again to Fig. 4, the DRM system 32 includes a license evaluator 36, the black box 30, a license store 38, and a state store 40.

DRM System 32 Components - License Evaluator 36

The license evaluator 36 locates one or more licenses 16 that correspond to the requested digital content 12, determines whether such licenses 16 are valid, reviews the license rules in such valid licenses 16, and determines based on the reviewed license rules whether the requesting user has the right to render the requested digital content 12 in the manner sought, among other things. As should be understood, the license evaluator 36 is a trusted component in the DRM system 32. In the present disclosure, to be 'trusted' means that the license server 24 (or any other trusting element) is satisfied that the trusted element will carry out the wishes of the owner of the digital content 12 according to the rights description in the license 16, and that a user cannot easily alter such trusted element for any purpose, nefarious or otherwise.

The license evaluator 36 has to be trusted in order to ensure that such license evaluator 36 will in fact evaluate a license 16 properly, and to ensure that such license evaluator 36 has not been adulterated or otherwise modified by a user for the purpose of bypassing actual evaluation of a license 16. Accordingly, the license evaluator 36 is run in a protected or shrouded environment such that the user is denied access to such license evaluator 36. Other protective measures may of course be employed in connection with the license evaluator 36 without departing from the spirit and scope of the present invention.

DRM System 32 Components - Black Box 30

Primarily, and as was discussed above, the black box 30 performs encryption and decryption functions in the DRM system 32. In particular, the black box 30 works in conjunction with the license evaluator 36 to decrypt and encrypt certain information as part of the license evaluation function. In addition, once the

license evaluator 36 determines that a user does in fact have the right to render the requested digital content 12 in the manner sought, the black box 30 is provided with a decryption key (KD) for such digital content 12, and performs the function of decrypting such digital content 12 based on such decryption key (KD).

The black box 30 is also a trusted component in the DRM system 32. In particular, the license server 24 must trust that the black box 30 will perform the decryption function only in accordance with the license rules in the license 16, and also trust that such black box 30 will not operate should it become adulterated or otherwise modified by a user for the nefarious purpose of bypassing actual evaluation of a license 16. Accordingly, the black box 30 is also run in a protected or shrouded environment such that the user is denied access to such black box 30. Again, other protective measures may be employed in connection with the black box 30 without departing from the spirit and scope of the present invention. Preferably, and like the content server 22 and license server 24, the black box 30 in the DRM system 32 has a unique public / private key pair (PU-BB, PR-BB) that is employed as part of the process of evaluating the license 16 and obtaining a decryption key (KD) for decrypting the digital content 12, as will be described in more detail below.

DRM System 32 Components - License Store 38

The license store 38 stores licenses 16 received by the DRM system 32 for corresponding digital content 12. The license store 38 itself need not be trusted since the license store 38 merely stores licenses 16, each of which already has trust components built thereinto, as will be described below. In one embodiment of the present invention, the license store 38 is merely a sub-directory of a drive such as a hard disk drive or a network drive. However, the license store 38 may be embodied in any other form without departing from the spirit and scope of the present invention, so long as such license store 38 performs the function of storing licenses 16 in a location relatively convenient to the DRM system 32.

DRM System 32 Components - State Store 40

The state store 40 performs the function of maintaining state

information corresponding to licenses 16 presently or formerly in the license store 38. Such state information is created by the DRM system 32 and stored in the state store 40 as necessary. For example, if a particular license 16 only allows a pre-determined number of renderings of a piece of corresponding digital content 12, the state store 40 maintains state information on how many renderings have in fact taken place in connection with such license 16. The state store 40 continues to maintain state information on licenses 16 that are no longer in the license store 38 to avoid the situation where it would otherwise be advantageous to delete a license 16 from the license store 38 and then obtain an identical license 16 in an attempt to delete the corresponding state information from the state store 40.

The state store 40 also has to be trusted in order to ensure that the information stored therein is not reset to a state more favorable to a user. Accordingly, the state store 40 is likewise run in a protected or shrouded environment such that the user is denied access to such state store 40. Once again, other protective measures may of course be employed in connection with the state store 40 without departing from the spirit and scope of the present invention. For example, the state store 40 may be stored by the DRM system 32 on the computing device 14 in an encrypted form.

DRM SYSTEM 32 - Content Rendering, Part 2

Referring again to Fig. 5A, and again discussing content rendering in one embodiment of the present invention, once the DRM system 32 has assumed control from the calling rendering application 34, such DRM system 32 then begins the process of determining whether the user has a right to render the requested digital content 12 in the manner sought. In particular, the DRM system 32 either locates a valid, enabling license 16 in the license store (steps 515, 517) or attempts to acquire a valid, enabling license 16 from the license server 24 (i.e. performs the license acquisition function as discussed below and as shown in Fig. 7).

As a first step, and referring now to Fig. 6, the license evaluator 36 of such DRM system 32 checks the license store 38 for the presence of one or more received licenses 16 that correspond to the digital content 12 (step 601). Typically, the

license 16 is in the form of a digital file, as will be discussed below, although it will be recognized that the license 16 may also be in other forms without departing from the spirit and scope of the present invention. Typically, the user will receive the digital content 12 without such license 16, although it will likewise be recognized that the digital content 12 may be received with a corresponding license 16 without departing from the spirit and scope of the present invention.

As was discussed above in connection with Fig. 3, each piece of digital content 12 is in a package 12p with a content ID (or package ID) identifying such digital content 12 (or package 12p), and a key ID identifying the decryption key (KD) that will decrypt the encrypted digital content 12. Preferably, the content ID (or package ID) and the key ID are in an un-encrypted form. Accordingly, and in particular, based on the content ID of the digital content 12, the license evaluator 36 looks for any license 16 in the license store 38 that contains an identification of applicability to such content ID. Note that multiple such licenses 16 may be found, especially if the owner of the digital content 12 has specified several different kinds of licenses 16 for such digital content 12, and the user has obtained multiple ones of such licenses 16. If in fact the license evaluator 36 does not find in the license store 38 any license 16 corresponding to the requested digital content 12, the DRM system 32 may then perform the function of license acquisition (step 519 of Fig. 5), to be described below.

Assume now that the DRM system 32 has been requested to render a piece of digital content 12, and one or more licenses 16 corresponding thereto are present in the license store 38. In one embodiment of the present invention, then, the license evaluator 36 of the DRM system 32 proceeds to determine for each such license 16 whether such license 16 itself is valid (steps 603 and 605 of Fig. 6). Preferably, and in particular, each license 16 includes a digital signature 26 based on the content 28 of the license 16. As should be understood, the digital signature 26 will not match the license 16 if the content 28 has been adulterated or otherwise modified. Thus, the license evaluator 36 can determine based on the digital signature 26 whether the

content 28 is in the form that it was received from the license server 24 (i.e., is valid). If no valid license 16 is found in the license store 38, the DRM system 32 may then perform the license acquisition function described below to obtain such a valid license 16.

Assuming that one or more valid licenses 16 are found, for each valid license 16, the license evaluator 36 of the DRM system 32 next determines whether such valid license 16 gives the user the right to render the corresponding digital content 12 in the manner desired (i.e., is enabling) (steps 607 and 609). In particular, the license evaluator 36 determines whether the requesting user has the right to play the requested digital content 12 based on the rights description in each license 16 and based on what the user is attempting to do with the digital content 12. For example, such rights description may allow the user to render the digital content 12 into a sound, but not into a decrypted digital copy.

As should be understood, the rights description in each license 16 specifies whether the user has rights to play the digital content 12 based on any of several factors, including who the user is, where the user is located, what type of computing device 14 the user is using, what rendering application 34 is calling the DRM system 32, the date, the time, etc. In addition, the rights description may limit the license 16 to a pre-determined number of plays, or pre-determined play time, for example. In such case, the DRM system 32 must refer to any state information with regard to the license 16, (i.e., how many times the digital content 12 has been rendered, the total amount of time the digital content 12 has been rendered, etc.), where such state information is stored in the state store 40 of the DRM system 32 on the user's computing device 14.

Accordingly, the license evaluator 36 of the DRM system 32 reviews the rights description of each valid license 16 to determine whether such valid license 16 confers the rights sought to the user. In doing so, the license evaluator 36 may have to refer to other data local to the user's computing device 14 to perform a determination of whether the user has the rights sought. As seen in Fig. 4, such data

may include an identification 42 of the user's computing device (machine) 14 and particular aspects thereof, an identification 44 of the user and particular aspects thereof, an identification of the rendering application 34 and particular aspects thereof, a system clock 46, and the like. If no valid license 16 is found that provides the user with the right to render the digital content 12 in the manner sought, the DRM system 32 may then perform the license acquisition function described below to obtain such a license 16, if in fact such a license 16 is obtainable.

Of course, in some instances the user cannot obtain the right to render the digital content 12 in the manner requested, because the content owner of such digital content 12 has in effect directed that such right not be granted. For example, the content owner of such digital content 12 may have directed that no license 16 be granted to allow a user to print a text document, or to copy a multimedia presentation into an un-encrypted form. In one embodiment of the present invention, the digital content 12 includes data on what rights are available upon purchase of a license 16, and types of licenses 16 available. However, it will be recognized that the content owner of a piece of digital content 12 may at any time change the rights currently available for such digital content 12 by changing the licenses 16 available for such digital content 12.

DRM SYSTEM 32 - License Acquisition

Referring now to Fig. 7, if in fact the license evaluator 36 does not find in the license store 38 any valid, enabling license 16 corresponding to the requested digital content 12, the DRM system 32 may then perform the function of license acquisition. As shown in Fig. 3, each piece of digital content 12 is packaged with information in an un-encrypted form regarding how to obtain a license 16 for rendering such digital content 12 (i.e., license acquisition information).

In one embodiment of the present invention, such license acquisition information may include (among other things) types of licenses 16 available, and one or more Internet web sites or other site information at which one or more appropriate license servers 24 may be accessed, where each such license server 24 is in fact capable

of issuing a license 16 corresponding to the digital content 12. Of course, the license 16 may be obtained in other manners without departing from the spirit and scope of the present invention. For example, the license 16 may be obtained from a license server 24 at an electronic bulletin board, or even in person or via regular mail in the form of a file on a magnetic or optical disk or the like.

Assuming that the location for obtaining a license 16 is in fact a license server 24 on a network, the license evaluator 36 then establishes a network connection to such license server 24 based on the web site or other site information, and then sends a request for a license 16 from such connected license server 24 (steps 701, 703). In particular, once the DRM system 32 has contacted the license server 24, such DRM system 32 transmits appropriate license request information 36 to such license server 24. In one embodiment of the present invention, such license 16 request information 36 may include:

- the public key of the black box 30 of the DRM system 32 (PU-BB);
- the version number of the black box 30 of the DRM system 32;
- a certificate with a digital signature from a certifying authority certifying the black box 30 (where the certificate may in fact include the aforementioned public key and version number of the black box 30);
- the content ID (or package ID) that identifies the digital content 12 (or package 12p);
- the key ID that identifies the decryption key (KD) for decrypting the digital content 12;
- the type of license 16 requested (if in fact multiple types are available);
- the type of rendering application 34 that requested rendering of the digital content 12;

and/or the like, among other things. Of course, greater or lessor amounts of license 16 request information 36 may be transmitted to the license server 24 by the DRM system

32 without departing from the spirit and scope of the present invention. For example, information on the type of rendering application 34 may not be necessary, while additional information about the user and/or the user's computing device 14 may be necessary.

Once the license server 24 has received the license 16 request information 36 from the DRM system 32, the license server 24 may then perform several checks for trust / authentication and for other purposes. In one embodiment of the present invention, such license server 24 checks the certificate with the digital signature of the certifying authority to determine whether such has been adulterated or otherwise modified (steps 705, 707). If so, the license server 24 refuses to grant any license 16 based on the request information 36. The license server 24 may also maintain a list of known 'bad' users and/or user's computing devices 14, and may refuse to grant any license 16 based on a request from any such bad user and/or bad user's computing device 14 on the list. Such 'bad' list may be compiled in any appropriate manner without departing from the spirit and scope of the present invention.

Based on the received request and the information associated therewith, and particularly based on the content ID (or package ID) in the license request information, the license server 24 can interrogate the content-key database 20 (Fig. 1) and locate a record corresponding to the digital content 12 (or package 12p) that is the basis of the request. As was discussed above, such record contains the decryption key (KD), key ID, and content ID for such digital content 12. In addition, such record may contain license data regarding the types of licenses 16 to be issued for the digital content 12 and the terms and conditions for each type of license 16. Alternatively, such record may include a pointer, link, or reference to a location having such additional information.

As mentioned above, multiple types of licenses 16 may be available. For example, for a relatively small license fee, a license 16 allowing a limited number of renderings may be available. For a relatively greater license fee, a license 16

allowing unlimited renderings until an expiration date may be available. For a still greater license fee, a license 16 allowing unlimited renderings without any expiration date may be available. Practically any type of license 16 having any kind of license terms may be devised and issued by the license server 24 without departing from the spirit and scope of the present invention.

In one embodiment of the present invention, the request for a license 16 is accomplished with the aid of a web page or the like as transmitted from the license server 24 to the user's computing device 14. Preferably, such web page includes information on all types of licenses 16 available from the license server 24 for the digital content 12 that is the basis of the license 16 request.

In one embodiment of the present invention, prior to issuing a license 16, the license server 24 checks the version number of the black box 30 to determine whether such black box 30 is relatively current (steps 709, 711). As should be understood, the black box 30 is intended to be secure and protected from attacks from a user with nefarious purposes (i.e., to improperly render digital content 12 without a license 16, or outside the terms of a corresponding license 16). However, it is to be recognized that no system and no software device is in fact totally secure from such an attack.

As should also be understood, if the black box 30 is relatively current, i.e., has been obtained or updated relatively recently, it is less likely that such black box 30 has been successfully attacked by such a nefarious user. Preferably, and as a matter of trust, if the license server 24 receives a license request with request information 36 including a black box 30 version number that is not relatively current, such license server 24 refuses to issue the requested license 16 until the corresponding black box 30 is upgraded to a current version, as will be described below. Put simply, the license server 24 will not trust such black box 30 unless such black box 30 is relatively current.

In the context of the black box 30 of the present invention, the term 'current' or 'relatively current' may have any appropriate meaning without departing

from the spirit and scope of the present invention. consistent with the function of providing trust in the black box 30 based on the age or use thereof. For example, 'current' may be defined according to age (i.e., less than one month old). As an alternative example, 'current' may be defined based on a number of times that the black box 30 has decrypted digital content 12 (i.e., less than 200 instances of decryption). Moreover, 'current' may be based on policy as set by each license server 24, where one license server 24 may define 'current' differently from another license server 24, and a license server 24 may further define 'current' differently depending on the digital content 12 for which a license 16 is requested. or depending on the type of license 16 requested, among other things.

Assuming that the license server 24 is satisfied from the version number of a black box 30 or other indicia thereof that such black box 30 is current, the license server 24 then proceeds to negotiate terms and conditions for the license 16 with the user (step 713). Alternatively, the license server 24 negotiates the license 16 with the user, then satisfies itself from the version number of the black box 30 that such black box 30 is current (i.e., performs step 713, then step 711). Of course, the amount of negotiation varies depending on the type of license 16 to be issued, and other factors. For example, if the license server 24 is merely issuing a paid-up unlimited use license 16, very little need be negotiated. On the other hand, if the license 16 is to be based on such items as varying values. sliding scales, break points, and other details, such items and details may need to be worked out between the license server 24 and the user before the license 16 can be issued.

As should be understood, depending on the circumstances, the license negotiation may require that the user provide further information to the license server 24 (for example, information on the user. the user's computing device 14, etc.). Importantly, the license negotiation may also require that the user and the license server 24 determine a mutually acceptable payment instrument (a credit account, a debit account, a mailed check, etc.) and/or payment method (paid-up immediately, spread over a period of time, etc.), among other things.

Once all the terms of the license 16 have been negotiated and agreed to by both the license server 24 and user (step 715). a digital license 16 is generated by the license server 24 (step 719), where such generated license 16 is based at least in part on the license request, the black box 30 public key (PU-BB), and the decryption key (KD) for the digital content 12 that is the basis of the request as obtained from the content-key database 20. In one embodiment of the present invention, and as seen in Fig. 8, the generated license 16 includes:

- the content ID of the digital content 12 to which the license 16 applies;
- a Digital Rights License (DRL) 48 (i.e., the rights description or actual terms and conditions of the license 16 written in a predetermined form that the license evaluator 36 can interrogate), perhaps encrypted with the decryption key (KD) (i.e., KD (DRL));
- the decryption key (KD) for the digital content 12 encrypted with the black box 30 public key (PU-BB) as receive in the license request (i.e.,(PU-BB (KD));
- a digital signature from the license server 24 (without any attached certificate) based on (KD (DRL)) and (PU-BB (KD)) and encrypted with the license server 24 private key (i.e., (S (PR-LS))); and
- the certificate that the license server 24 obtained previously from the content server 22, such certificate indicating that the license server 24 has the authority from the content server 22 to issue the license 16 (i.e., (CERT (PU-LS) S (PR-CS))).

As should be understood, the aforementioned elements and perhaps others are packaged into a digital file or some other appropriate form. As should also be understood, if the DRL 48 or (PU-BB (KD)) in the license 16 should become adulterated or otherwise modified, the digital signature (S (PR-LS)) in the license 16 will not match and therefore will not validate such license 16. For this reason, the DRL 48 need not necessarily be in an encrypted form (i.e., (KD(DRL)) as mentioned

above), although such encrypted form may in some instances be desirable and therefore may be employed without departing from the spirit and scope of the present invention.

Once the digital license 16 has been prepared. such license 16 is then issued to the requestor (i.e., the DRM system 32 on the user's computing device 14) (step 719 of Fig. 7). Preferably, the license 16 is transmitted over the same path through which the request therefor was made (i.e., the Internet or another network), although another path may be employed without departing from the spirit and scope of the present invention. Upon receipt, the requesting DRM system 32 preferably automatically places the received digital license 16 in the license store 38 (step 721).

It is to be understood that a user's computing device 14 may on occasion malfunction, and licenses 16 stored in the license store 38 of the DRM system 32 on such user's computing device 14 may become irretrievably lost. Accordingly, it is preferable that the license server 24 maintain a database 50 of issued licenses 16 (Fig. 1), and that such license server 24 provide a user with a copy or re-issue (hereinafter 're-issue') of an issued license 16 if the user is in fact entitled to such re-issue. In the aforementioned case where licenses 16 are irretrievably lost, it is also likely the case that state information stored in the state store 40 and corresponding to such licenses 16 is also lost. Such lost state information should be taken into account when re-issuing a license 16. For example. a fixed number of renderings license 16 might legitimately be re-issued in a pro-rated form after a relatively short period of time, and not re-issued at all after a relatively longer period of time.

DRM SYSTEM 32 - Installation/Upgrade of Black Box 30

As was discussed above, as part of the function of acquiring a license 16, the license server 24 may deny a request for a license 16 from a user if the user's computing device 14 has a DRM system 32 with a black box 30 that is not relatively current, i.e., has a relatively old version number. In such case. it is preferable that the black box 30 of such DRM system 32 be upgraded so that the license acquisition function can then proceed. Of course, the black box 30 may be upgraded at other times

without departing from the spirit and scope of the present invention.

Preferably, as part of the process of installing the DRM system 32 on a user's computing device 14, a non-unique 'lite' version of a black box 30 is provided. Such 'lite' black box 30 is then upgraded to a unique regular version prior to rendering a piece of digital content 12. As should be understood, if each black box 30 in each DRM system 32 is unique, a security breach into one black box 30 cannot easily be replicated with any other black box 30.

Referring now to Fig. 9, the DRM system 32 obtains the unique black box 30 by requesting same from a black box server 26 or the like (as was discussed above and as shown in Fig. 1) (step 901). Typically, such request is made by way of the Internet, although other means of access may be employed without departing from the spirit and scope of the present invention. For example, the connection to a black box server 26 may be a direct connection, either locally or remotely. An upgrade from one unique non-lite black box 30 to another unique non-lite black box 30 may also be requested by the DRM system 32 at any time, such as for example a time when a license server 24 deems the black box 30 not current, as was discussed above.

Thereafter, the black box server 26 generates a new unique black box 30 (step 903). As seen in Fig. 3, each new black box 30 is provided with a version number and a certificate with a digital signature from a certifying authority. As was discussed above in connection with the license acquisition function, the version number of the black box 30 indicates the relative age and/or use thereof. The certificate with the digital signature from the certifying authority, also discussed above in connection with the license acquisition function, is a proffer or vouching mechanism from the certifying authority that a license server 24 should trust the black box 30. Of course, the license server 24 must trust the certifying authority to issue such a certificate for a black box 30 that is in fact trustworthy. It may be the case, in fact, that the license server 24 does not trust a particular certifying authority, and refuses to honor any certificate issued by such certifying authority. Trust may not occur, for example, if a particular certifying authority is found to be engaging in a pattern of

improperly issuing certificates.

Preferably, and as was discussed above, the black box server 26 includes a new unique public / private key pair (PU-BB, PR-BB) with the newly generated unique black box 30 (step 903 of Fig. 9). Preferably, the private key for the black box 30 (PR-BB) is accessible only to such black box 30, and is hidden from and inaccessible by the remainder of the world, including the computing device 14 having the DRM system 32 with such black box 30, and the user thereof.

Most any hiding scheme may be employed without departing from the spirit and scope of the present invention, so long as such hiding scheme in fact performs the function of hiding the private key (PR-BB) from the world. As but one example, the private key (PR-BB) may be split into several sub-components, and each sub-component may be encrypted uniquely and stored in a different location. In such a situation, it is preferable that such sub-components are never assembled in full to produce the entire private key (PR-BB).

In one embodiment of the present invention, such private key (PR-BB) is encrypted according to code-based encryption techniques. In particular, in such embodiment, the actual software code of the black box 30 (or other software code) is employed as encrypting key(s). Accordingly, if the code of the black box 30 (or the other software code) becomes adulterated or otherwise modified, for example by a user with nefarious purposes, such private key (PR-BB) cannot be decrypted.

Although each new black box 30 is delivered with a new public / private key pair (PU-BB, PR-BB), such new black box 30 is also preferably given access to old public / private key pairs from old black boxes 30 previously delivered to the DRM system 32 on the user's computing device 14 (step 905). Accordingly, the upgraded black box 30 can still employ the old key pairs to access older digital content 12 and older corresponding licenses 16 that were generated according to such old key pairs, as will be discussed in more detail below.

Preferably, the upgraded black box 30 delivered by the black box server 26 is tightly tied to or associated with the user's computing device 14. Accordingly,

the upgraded black box 30 cannot be operably transferred among multiple computing devices 14 for nefarious purposes or otherwise. In one embodiment of the present invention, as part of the request for the black box 30 (step 901) the DRM system 32 provides hardware information unique to such DRM system 32 and/or unique to the user's computing device 14 to the black box server 26, and the black box server 26 generates a black box 30 for the DRM system 32 based in part on such provided hardware information. Such generated upgraded black box 30 is then delivered to and installed in the DRM system 32 on the user's computing device 14 (steps 907, 909). If the upgraded black box 30 is then somehow transferred to another computing device 14, the transferred black box 30 recognizes that it is not intended for such other computing device 14, and does not allow any requested rendering to proceed on such other computing device 14.

Once the new black box 30 is installed in the DRM system 32, such DRM system 32 can proceed with a license acquisition function or with any other function.

DRM SYSTEM 32 - Content Rendering, Part 3

Referring now to Fig. 5B, and assuming, now, that the license evaluator 36 has found at least one valid license 16 and that at least one of such valid licenses 16 provides the user with the rights necessary to render the corresponding digital content 12 in the manner sought (i.e., is enabling), the license evaluator 36 then selects one of such licenses 16 for further use (step 519). Specifically, to render the requested digital content 12, the license evaluator 36 and the black box 30 in combination obtain the decryption key (KD) from such license 16, and the black box 30 employs such decryption key (KD) to decrypt the digital content 12. In one embodiment of the present invention, and as was discussed above, the decryption key (KD) as obtained from the license 16 is encrypted with the black box 30 public key (PU-BB(KD)), and the black box 30 decrypts such encrypted decryption key with its private key (PR-BB) to produce the decryption key (KD) (steps 521, 523). However, other methods of obtaining the decryption key (KD) for the digital content 12 may be employed without

departing from the spirit and scope of the present invention.

Once the black box 30 has the decryption key (KD) for the digital content 12 and permission from the license evaluator 36 to render the digital content 12, control may be returned to the rendering application 34 (steps 525, 527). In one embodiment of the present invention, the rendering application 34 then calls the DRM system 32 / black box 30 and directs at least a portion of the encrypted digital content 12 to the black box 30 for decryption according to the decryption key (KD) (step 529). The black box 30 decrypts the digital content 12 based upon the decryption key (KD) for the digital content 12, and then the black box 30 returns the decrypted digital content 12 to the rendering application 34 for actual rendering (steps 533, 535). The rendering application 34 may either send a portion of the encrypted digital content 12 or the entire digital content 12 to the black box 30 for decryption based on the decryption key (KD) for such digital content 12 without departing from the spirit and scope of the present invention.

Preferably, when the rendering application 34 sends digital content 12 to the black box 30 for decryption, the black box 30 and/or the DRM system 32 authenticates such rendering application 34 to ensure that it is in fact the same rendering application 34 that initially requested the DRM system 32 to run (step 531). Otherwise, the potential exists that rendering approval may be obtained improperly by basing the rendering request on one type of rendering application 34 and in fact rendering with another type of rendering application 34. Assuming the authentication is successful and the digital content 12 is decrypted by the black box 30, the rendering application 34 may then render the decrypted digital content 12 (steps 533, 535).

Sequence of Key Transactions

Referring now to Fig. 10, in one embodiment of the present invention, a sequence of key transactions is performed to obtain the decryption key (KD) and evaluate a license 16 for a requested piece of digital content 12 (i.e., to perform steps 515-523 of Figs. 5A and 5B). Mainly, in such sequence, the DRM system 32 obtains the decryption key (KD) from the license 16, uses information obtained from the

license 16 and the digital content 12 to authenticate or ensure the validity of both, and then determines whether the license 16 in fact provides the right to render the digital content 12 in the manner sought. If so, the digital content 12 may be rendered.

Bearing in mind that each license 16 for the digital content 12, as seen in Fig. 8, includes:

- the content ID of the digital content 12 to which the license 16 applies;
- the Digital Rights License (DRL) 48, perhaps encrypted with the decryption key (KD) (i.e., KD (DRL));
- the decryption key (KD) for the digital content 12 encrypted with the black box 30 public key (PU-BB) (i.e.,(PU-BB (KD));
- the digital signature from the license server 24 based on (KD (DRL)) and (PU-BB (KD)) and encrypted with the license server 24 private key (i.e., (S (PR-LS))); and
- the certificate that the license server 24 obtained previously from the content server 22 (i.e., (CERT (PU-LS) S (PR-CS))),

and also bearing in mind that the package 12p having the digital content 12, as seen in Fig. 3, includes:

- the content ID of such digital content 12;
- the digital content 12 encrypted by KD (i.e., (KD(CONTENT)));
- a license acquisition script that is not encrypted; and
- the key KD encrypting the content server 22 public key (PU-CS), signed by the content server 22 private key (PR-CS) (i.e., (KD (PU-CS) S (PR-CS))),

in one embodiment of the present invention, the specific sequence of key transactions that are performed with regard to a specific one of the licenses 16 for the digital content 12 is as follows:

1. Based on (PU-BB (KD)) from the license 16, the black box 30 of the DRM system 32 on the user's computing device 14 applies its private key (PR-

BB) to obtain (KD) (step 1001). (PR-BB (PU-BB (KD)) = (KD)). Note, importantly, that the black box 30 could then proceed to employ KD to decrypt the digital content 12 without any further ado. However, and also importantly, the license server 24 trusts the black box 30 not to do so. Such trust was established at the time such license server 24 issued the license 16 based on the certificate from the certifying authority vouching for the trustworthiness of such black box 30. Accordingly, despite the black box 30 obtaining the decryption key (KD) as an initial step rather than a final step, the DRM system 32 continues to perform all license 16 validation and evaluation functions, as described below.

2. Based on (KD (PU-CS) S (PR-CS)) from the digital content 12, the black box 30 applies the newly obtained decryption key (KD) to obtain (PU-CS) (step 1003). (KD (KD (PU-CS)) = (PU-CS)). Additionally, the black box 30 can apply (PU-CS) as against the signature (S (PR-CS)) to satisfy itself that such signature and such digital content 12 / package 12p is valid (step 1005). If not valid, the process is halted and access to the digital content 12 is denied.

3. Based on (CERT (PU-LS) S (PR-CS)) from the license 16, the black box 30 applies the newly obtained content server 22 public key (PU-CS) to satisfy itself that the certificate is valid (step 1007), signifying that the license server 24 that issued the license 16 had the authority from the content server 22 to do so, and then examines the certificate contents to obtain (PU-LS) (step 1009). If not valid, the process is halted and access to the digital content 12 based on the license 16 is denied.

4. Based on (S (PR-LS)) from the license 16, the black box 30 applies the newly obtained license server 24 public key (PU-LS) to satisfy itself that the license 16 is valid (step 1011). If not valid, the process is halted and access to the digital content 12 based on the license 16 is denied.

5. Assuming all validation steps are successful, and that the DRL 48 in the license 16 is in fact encrypted with the decryption key (KD), the license evaluator 36 then applies the already-obtained decryption key (KD) to (KD(DRL)) as obtained from the license 16 to obtain the license terms from the license 16 (i.e., the

DRL 48) (step 1013). Of course, if the DRL 48 in the license 16 is not in fact encrypted with the decryption key (KD), step 1013 may be omitted. The license evaluator 36 then evaluates / interrogates the DRL 48 and determines whether the user's computing device 14 has the right based on the DRL 48 in the license 16 to render the corresponding digital content 12 in the manner sought (i.e., whether the DRL 48 is enabling) (step 1015). If the license evaluator 36 determines that such right does not exist, the process is halted and access to the digital content 12 based on the license 16 is denied.

6. Finally, assuming evaluation of the license 16 results in a positive determination that the user's computing device 14 has the right based on the DRL 48 terms to render the corresponding digital content 12 in the manner sought, the license evaluator 36 informs the black box 30 that such black box 30 can render the corresponding digital content 12 according to the decryption key (KD). The black box 30 thereafter applies the decryption key (KD) to decrypt the digital content 12 from the package 12p (i.e., (KD(KD(CONTENT)) = (CONTENT)) (step 1017).

It is important to note that the above-specified series of steps represents an alternating or 'ping-ponging' between the license 16 and the digital content 12. Such ping-ponging ensures that the digital content 12 is tightly bound to the license 16, in that the validation and evaluation process can only occur if both the digital content 12 and license 16 are present in a properly issued and valid form. In addition, since the same decryption key (KD) is needed to get the content server 22 public key (PU-CS) from the license 16 and the digital content 12 from the package 12p in a decrypted form (and perhaps the license terms (DRL 48) from the license 16 in a decrypted form), such items are also tightly bound. Signature validation also ensures that the digital content 12 and the license 16 are in the same form as issued from the content server 22 and the license server 24, respectively. Accordingly, it is difficult if not impossible to decrypt the digital content 12 by bypassing the license server 24, and also difficult if not impossible to alter and then decrypt the digital content 12 or the license 16.

In one embodiment of the present invention, signature verification, and especially signature verification of the license 16, is alternately performed as follows. Rather than having a signature encrypted by the private key of the license server 16 (PR-LS), as is seen in Fig. 8, each license 16 has a signature encrypted by a private root key (PR-R) (not shown), where the black box 30 of each DRM system 32 includes a public root key (PU-R) (also not shown) corresponding to the private root key (PR-R). The private root key (PR-R) is known only to a root entity, and a license server 24 can only issue licenses 16 if such license server 24 has arranged with the root entity to issue licenses 16.

In particular, in such embodiment:

1. the license server 24 provides its public key (PU-LS) to the root entity;
2. the root entity returns the license server public key (PU-LS) to such license server 24 encrypted with the private root key (PR-R) (i.e., (CERT (PU-LS) S (PR-R))); and
3. the license server 24 then issues a license 16 with a signature encrypted with the license server private key (S (PR-LS)), and also attaches to the license the certificate from the root entity (CERT (PU-LS) S (PR-R)).

For a DRM system 18 to validate such issued license 16, then, the DRM system 18:

1. applies the public root key (PU-R) to the attached certificate (CERT (PU-LS) S (PR-R)) to obtain the license server public key (PU-LS); and
2. applies the obtained license server public key (PU-LS) to the signature of the license 16 (S (PR-LS).

Importantly, it should be recognized that just as the root entity gave the license server 24 permission to issue licenses 16 by providing the certificate (CERT (PU-LS) S (PR-R)) to such license server 24, such license server 24 can provide a

similar certificate to a second license server 24 (i.e., (CERT (PU-LS2) S (PR-LS1)), thereby allowing the second license server to also issue licenses 16. As should now be evident, a license 16 issued by the second license server would include a first certificate (CERT (PU-LS1) S (PR-R)) and a second certificate (CERT (PU-LS2) S (PR-LS1)). Likewise, such license 16 is validated by following the chain through the first and second certificates. Of course, additional links in the chain may be added and traversed.

One advantage of the aforementioned signature verification process is that the root entity may periodically change the private root key (PR-R), thereby likewise periodically requiring each license server 24 to obtain a new certificate (CERT (PU-LS) S (PR-R)). Importantly, as a requirement for obtaining such new certificate, each license server may be required to upgrade itself. As with the black box 30, if a license server 24 is relatively current, i.e., has been upgraded relatively recently, it is less likely that license server 24 has been successfully attacked. Accordingly, as a matter of trust, each license server 24 is preferably required to be upgraded periodically via an appropriate upgrade trigger mechanism such as the signature verification process. Of course, other upgrade mechanisms may be employed without departing from the spirit and scope of the present invention.

Of course, if the private root key (PR-R) is changed, then the public root key (PU-R) in each DRM system 18 must also be changed. Such change may for example take place during a normal black box 30 upgrade, or in fact may require that a black box 30 upgrade take place. Although a changed public root key (PU-R) may potentially interfere with signature validation for an older license 16 issued based on an older private root key (PR-R), such interference may be minimized by requiring that an upgraded black box 30 remember all old public root keys (PU-R). Alternatively, such interference may be minimized by requiring signature verification for a license 16 only once, for example the first time such license 16 is evaluated by the license evaluator 36 of a DRM system 18. In such case, state information on whether signature verification has taken place should be compiled, and such state information

should be stored in the state store 40 of the DRM system 18.

Digital Rights License 48

In the present invention, the license evaluator 36 evaluates a Digital Rights License (DRL) 48 as the rights description or terms of a license 16 to determine if such DRL 48 allows rendering of a corresponding piece of digital content 12 in the manner sought. In one embodiment of the present invention, the DRL 48 may be written by a licensor (i.e., the content owner) in any DRL language.

As should be understood, there are a multitude of ways to specify a DRL 48. Accordingly, a high degree of flexibility must be allowed for in any DRL language. However, it is impractical to specify all aspects of a DRL 48 in a particular license language, and it is highly unlikely that the author of such a language can appreciate all possible licensing aspects that a particular digital licensor may desire. Moreover, a highly sophisticated license language may be unnecessary and even a hindrance for a licensor providing a relatively simple DRL 48. Nevertheless, a licensor should not be unnecessarily restricted in how to specify a DRL 48. At the same time, the license evaluator 36 should always be able to get answers from a DRL 48 regarding a number of specific license questions.

In the present invention, and referring now to Fig. 11, a DRL 48 can be specified in any license language, but includes a language identifier or tag 54. The license evaluator 36 evaluating the license 16, then, performs the preliminary step of reviewing the language tag 54 to identify such language, and then selects an appropriate license language engine 52 for accessing the license 16 in such identified language. As should be understood, such license language engine 52 must be present and accessible to the license evaluator 36. If not present, the language tag 54 and/or the DRL 48 preferably includes a location 56 (typically a web site) for obtaining such language engine 52.

Typically, the language engine 52 is in the form of an executable file or set of files that reside in a memory of the user's computing device 14, such as a hard drive. The language engine 52 assists the license evaluator 36 to directly interrogate

the DRL 48, the license evaluator 36 interrogates the DRL 48 indirectly via the language engine 48 acting as an intermediary, or the like. When executed, the language engine 52 runs in a work space in a memory of the user's computing device 14, such as RAM. However, any other form of language engine 52 may be employed without departing from the spirit and scope of the present invention.

Preferably, any language engine 52 and any DRL language supports at least a number of specific license questions that the license evaluator 36 expects to be answered by any DRL 48, as will be discussed below. Accordingly, the license evaluator 36 is not tied to any particular DRL language; a DRL 48 may be written in any appropriate DRL language; and a DRL 48 specified in a new license language can be employed by an existing license evaluator 36 by having such license evaluator 36 obtain a corresponding new language engine 52.

DRL Languages

Two examples of DRL languages, as embodied in respective DRLs 48, are provided below. The first, 'simple' DRL 48 is written in a DRL language that specifies license attributes, while the second 'script' DRL 48 is written in a DRL language that can perform functions according to the script specified in the DRL 48. While written in a DRL language, the meaning of each line of code should be apparent based on the linguistics thereof and/or on the attribute description chart that follows:

Simple DRL 48:

```
<LICENSE>
    <DATA>
        <NAME>Beastie Boy's Play</NAME>
        <ID>39384</ID>
        <DESCRIPTION>Play the song 3 times</DESCRIPTION>
        <TERMS></TERMS>
        <VALIDITY>
            <NOTBEFORE>19980102 23:20:14Z</NOTBEFORE>
            <NOTAFTER>19980102 23:20:14Z</NOTAFTER>
        </VALIDITY>
        <ISSUEDDATE>19980102 23:20:14Z</ISSUEDDATE>
        <LICENSORSITE>http://www.foo.com</LICENSORSITE>
```

```
<CONTENT>
        <NAME>Beastie Boy's</NAME>
        <ID>392</ID>
        <KEYID>39292</KEYID>
        <TYPE>MS Encrypted ASF 2.0</TTYPE>
</CONTENT>
<OWNER>
        <ID>939KDKD393KD</ID>
        <NAME>Universal</NAME>
        <PUBLICKEY></PUBLICKEY>
</OWNER>
<LICENSEE>
        <NAME>Arnold</NAME>
        <ID>939KDKD393KD</ID>
        <PUBLICKEY></PUBLICKEY>
</LICENSEE>
<PRINCIPAL TYPE='AND'>
        <PRINCIPAL TYPE='OR'>
                <PRINCIPAL>
                        <TYPE>x86Computer</TYPE>
                        <ID>3939292939d9e939</ID>
                        <NAME>Personal Computer</NAME>
                        <AUTHTYPE>Intel Authenticated Boot PC
                        SHA-1 DSA512</AUTHTYPE>
                        <AUTHDATA>29293939</AUTHDATA>
                </PRINCIPAL>
                <PRINCIPAL>
                        <TYPE>Application</TYPE>
                        <ID>2939495939292</ID>
                        <NAME>Window's Media Player</NAME>
                        <AUTHTYPE>Authenticode SHA-
                        1</AUTHTYPE>
                        <AUTHDATA>93939</AUTHDATA>
                </PRINCIPAL>
        </PRINCIPAL>
        <PRINCIPAL>
                <TYPE>Person</TYPE>
                <ID>39299482010</ID>
                <NAME>Arnold Blinn</NAME>
                <AUTHTYPE>Authenticate user</AUTHTYPE>
                <AUTHDATA>\\redmond\arnoldb</AUTHDATA>
        </PRINCIPAL>
</PRINCIPAL>
```

```
                <DRLTYPE>Simple</DRLTYPE>  [the language tag 54]
                <DRLDATA>
                        <START>19980102 23:20:14Z</START>
                        <END>19980102 23:20:14Z</END>
                        <COUNT>3</COUNT>
                        <ACTION>PLAY</ACTION>
                </DRLDATA>
                <ENABLINGBITS>aaaabbbbccccdddd</ENABLINGBITS>
        </DATA>
        <SIGNATURE>
        <SIGNERNAME>Universal</SIGNERNAME>
                <SIGNERID>9382ABK3939DKD</SIGNERID>
                <HASHALGORITHMID>MD5</HASHALGORITHMID>
                <SIGNALGORITHMID>RSA 128</SIGNALGORITHMID>
                <SIGNATURE>xxxyyyxxxyyyxxxyyy</SIGNATURE>
                <SIGNERPUBLICKEY></SIGNERPUBLICKEY>
                <CONTENTSIGNEDSIGNERPUBLICKEY></CONTENTSIGNEDSI
                GNERPUBLICKEY>
        </SIGNATURE>
</LICENSE>
```

Script DRL 48:

```
<LICENSE>
        <DATA>
                <NAME>Beastie Boy's Play</NAME>
                <ID>39384</ID>
                <DESCRIPTION>Play the song unlimited</DESCRIPTION>
                <TERMS></TERMS>
                <VALIDITY>
                        <NOTBEFORE>19980102 23:20:14Z</NOTBEFORE>
                        <NOTAFTER>19980102 23:20:14Z</NOTAFTER>
                </VALIDITY>
                <ISSUEDDATE>19980102 23:20:14Z</ISSUEDDATE>
                <LICENSORSITE>http://www.foo.com</LICENSORSITE>
                <CONTENT>
                        <NAME>Beastie Boy's</NAME
                        <ID>392</ID>
                        <KEYID>39292</KEYID>
                        <TYPE>MS Encrypted ASF 2.0</TTYPE>
                </CONTENT>
                <OWNER>
                        <ID>939KDKD393KD</ID>
```

```
                <NAME>Universal</NAME>
                <PUBLICKEY></PUBLICKEY>
            </OWNER>
            <LICENSEE>
                <NAME>Arnold</NAME>
                <ID>939KDKD393KD</ID>
                <PUBLICKEY></PUBLICKEY>
            </LICENSEE>
            <DRLTYPE>Script</DRLTYPE>    [the language tag 54]
            <DRLDATA>
                function on_enable(action. args) as boolean
                    result = False
                    if action = "PLAY" then
                        result = True
                    end if
                    on_action = False
                end function
                . . .
            </DRLDATA>
        </DATA>
        <SIGNATURE>
            <SIGNERNAME>Universal</SIGNERNAME>
            <SIGNERID>9382</SIGNERID>
            <SIGNERPUBLICKEY></SIGNERPUBLICKEY>
            <HASHID>MD5</HASHID>
            <SIGNID>RSA 128</SIGNID>
            <SIGNATURE>xxxyyyxxxyyyxxxyyy</SIGNATURE>
            <CONTENTSIGNEDSIGNERPUBLICKEY></CONTENTSIGNEDSI
            GNERPUBLICKEY>
        </SIGNATURE>
</LICENSE>
```

In the two DRLs 48 specified above, the attributes listed have the following descriptions and data types:

Attribute	Description	Data Type
Id	ID of the license	GUID
Name	Name of the license	String
Content Id	ID of the content	GUID
Content Key Id	ID for the encryption key of the content	GUID
Content Name	Name of the content	String
Content Type	Type of the content	String

Owner Id	ID of the owner of the content	GUID
Owner Name	Name of the owner of the content	String
Owner Public Key	Public key for owner of content. This is a base-64 encoded public key for the owner of the content.	String
Licensee Id	Id of the person getting license. It may be null.	GUID
Licensee Name	Name of the person getting license. It may be null.	String
Licensee Public Key	Public key of the licensee. This is the base-64 encoded public key of the licensee. It may be null.	String
Description	Simple human readable description of the license	String
Terms	Legal terms of the license. This may be a pointer to a web page containing legal prose.	String
Validity Not After	Validity period of license expiration	Date
Validity Not Before	Validity period of license start	Date
Issued Date	Date the license was issued	Date
DRL Type	Type of the DRL. Example include "SIMPLE" or "SCRIPT"	String
DRL Data	Data specific to the DRL	String
Enabling Bits	These are the bits that enable access to the actual content. The interpretation of these bits is up to the application, but typically this will be the private key for decryption of the content. This data will be base-64 encoded. Note that these bits are encrypted using the public key of the individual machine.	String
Signer Id	ID of person signing license	GUID
Signer Name	Name of person signing license	String
Signer Public Key	Public key for person signing license. This is the base-64 encode public key for the signer.	String
Content Signed Signer Public Key	Public key for person signing the license that has been signed by the content server private key. The public key to verify this signature will be encrypted in the content. This is base-64 encoded.	String

Hash Alg Id	Algorithm used to generate hash. This is a string, such as "MD5".	String
Signature Alg Id	Algorithm used to generate signature. This is a string, such as "RSA 128".	String
Signature	Signature of the data. This is base-64 encoded data.	String

Methods

As was discussed above, it is preferable that any language engine 52 and any DRL language support at least a number of specific license questions that the digital license evaluator 36 expects to be answered by any DRL 48. Recognizing such supported questions may include any questions without departing from the spirit and scope of the present invention, and consistent with the terminology employed in the two DRL 48 examples above, in one embodiment of the present invention, such supported questions or 'methods' include 'access methods', 'DRL methods', and 'enabling use methods', as follows:

Access Methods

Access methods are used to query a DRL 48 for top-level attributes.

VARIANT QueryAttribute (BSTR key)

Valid keys include License.Name, License.Id, Content.Name, Content.Id, Content.Type, Owner.Name, Owner.Id, Owner.PublicKey, Licensee.Name, Licensee.Id, Licensee.PublicKey, Description, and Terms, each returning a BSTR variant; and Issued, Validity.Start and Validity.End, each returning a Date Variant.

DRL Methods

The implementation of the following DRL methods varies from DRL 48 to DRL 48. Many of the DRL methods contain a variant parameter labeled 'data' which is intended for communicating more advanced information with a DRL 48. It

is present largely for future expandability.

Boolean IsActivated(Variant data)

This method returns a Boolean indicating whether the DRL 48 / license 16 is activated. An example of an activated license 16 is a limited operation license 16 that upon first play is active for only 48 hours.

Activate(Variant data)

This method is used to activate a license 16. Once a license 16 is activated, it cannot be deactivated.

Variant QueryDRL(Variant data)

This method is used to communicate with a more advanced DRL 48. It is largely about future expandability of the DRL 48 feature set.

Variant GetExpires(BSTR action, Variant data)

This method returns the expiration date of a license 16 with regard to the passed-in action. If the return value is NULL, the license 16 is assumed to never expire or does not yet have an expiration date because it hasn't been activated. or the like.

Variant GetCount(BSTR action, Variant data)

This method returns the number of operations of the passed-in action that are left. If NULL is returned, the operation can be performed an unlimited number of times.

Boolean IsEnabled(BSTR action, Variant data)

This method indicates whether the license 16 supports the requested action at the present time.

Boolean IsSunk(BSTR action, Variant data)

This method indicates whether the license 16 has been paid for. A license 16 that is paid for up front would return TRUE, while a license 16 that is not paid for up front, such as a license 16 that collects payments as it is used, would return FALSE.

Enabling Use Methods.

These methods are employed to enable a license 16 for use in decrypting content.

Boolean Validate (BSTR key)

This method is used to validate a license 16. The passed-in key is the black box 30 public key (PU-BB) encrypted by the decryption key (KD) for the corresponding digital content 12 (i.e.,(KD(PU-BB))) for use in validation of the signature of the license 16. A return value of TRUE indicates that the license 16 is valid. A return value of FALSE indicates invalid.

int OpenLicense 16(BSTR action, BSTR key. Variant data)

This method is used to get ready to access the decrypted enabling bits. The passed-in key is (KD(PU-BB)) as described above. A return value of 0 indicates success. Other return values can be defined.

BSTR GetDecryptedEnablingBits (BSTR action. Variant data)

Variant GetDecryptedEnablingBitsAsBinary (BSTR action. Variant Data)

These methods are used to access the enabling bits in decrypted form. If this is not successful for any of a number of reasons, a null string or null variant is returned.

void CloseLicense 16 (BSTR action, Variant data)

This method is used to unlock access to the enabling bits for performing the passed-in action. If this is not successful for any of a number of reasons. a null string is returned.

Heuristics

As was discussed above, if multiple licenses 16 are present for the same piece of digital content 12, one of the licenses 16 must be chosen for further use. Using the above methods, the following heuristics could be implemented to make such choice. In particular, to perform an action (say "PLAY") on a piece of digital content 12, the following steps could be performed:

1. Get all licenses 16 that apply to the particular piece of digital content 12.

2. Eliminate each license 16 that does not enable the action by calling the IsEnabled function on such license 16.

3. Eliminate each license 16 that is not active by calling IsActivated on such license 16.

4. Eliminate each license 16 that is not paid for up front by calling IsSunk on such license 16.

5. If any license 16 is left, use it. Use an unlimited-number-of-plays license 16 before using a limited-number-of-plays license 16, especially if the unlimited-number-of-plays license 16 has an expiration date. At any time, the user should be allowed to select a specific license 16 that has already been acquired, even if the choice is not cost-effective. Accordingly, the user can select a license 16 based on criteria that are perhaps not apparent to the DRM system 32.

6. If there are no licenses 16 left, return status so indicating. The user would then be given the option of:

using a license 16 that is not paid for up front, if available;

activating a license 16, if available; and/or

performing license acquisition from a license server 24.

CONCLUSION

The programming necessary to effectuate the processes performed in connection with the present invention is relatively straight-forward and should be

apparent to the relevant programming public. Accordingly, such programming is not attached hereto. Any particular programming, then, may be employed to effectuate the present invention without departing from the spirit and scope thereof.

In the foregoing description, it can be seen that the present invention comprises a new and useful enforcement architecture 10 that allows the controlled rendering or playing of arbitrary forms of digital content 12, where such control is flexible and definable by the content owner of such digital content 12. Also, the present invention comprises a new useful controlled rendering environment that renders digital content 12 only as specified by the content owner, even though the digital content 12 is to be rendered on a computing device 14 which is not under the control of the content owner. Further, the present invention comprises a trusted component that enforces the rights of the content owner on such computing device 14 in connection with a piece of digital content 12, even against attempts by the user of such computing device 14 to access such digital content 12 in ways not permitted by the content owner.

It should be appreciated that changes could be made to the embodiments described above without departing from the inventive concepts thereof. It should be understood, therefore, that this invention is not limited to the particular embodiments disclosed, but it is intended to cover modifications within the spirit and scope of the present invention as defined by the appended claims.

CLAIMS

1. A method for a device to interdependently validate:

a digital content package having a piece of digital content in an encrypted form; and

a corresponding digital license for rendering the digital content,

the method comprising:

deriving a first key from a source available to the device;

obtaining a first digital signature from the digital content package;

applying the first key to the first digital signature to validate the first digital signature and the digital content package;

deriving a second key based on the first digital signature;

obtaining a second digital signature from the license; and

applying the second key to the second digital signature to validate the second digital signature and the license.

2. The method of claim 1 wherein deriving the first key comprises:

obtaining a first encrypted key from the license;

applying a key available to the device to the first encrypted key to decrypt the first encrypted key;

obtaining a second encrypted key from the digital content; and

applying the decrypted first encrypted key to the second encrypted key to produce the first key.

3. The method of claim 2 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first encrypted key is the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))).

4. The method of claim 2 wherein the device has a public key (PU-D) and a private key (PR-D), and wherein the key available to the device is (PR-D).

5. The method of claim 2 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the second encrypted key is the content provider public key (PU-C) encrypted with the decryption key (KD) (i.e., KD (PU-C)).

6. The method of claim 2 wherein the second encrypted key is the basis for the first digital signature.

7. The method of claim 1 wherein deriving the second key comprises:

obtaining a signed certificate from the license. the signed certificate having contents therein; and

applying the first key to the signature of the signed certificate to produce the contents of the certificate and also to validate the signature.

8. The method of claim 7 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the contents of the certificate is (PU-L).

9. The method of claim 8 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))).

10. The method of claim 8 wherein the digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the root source private key (PR-R) (i.e., (CERT (PU-L) S (PR-R))).

11. The method of claim 1 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the first key is (PU-C).

12. The method of claim 11 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and

is signed by the content provider private key (PR-C) (i.e.. (KD (PU-C) S (PR-C))).

13. The method of claim 12 wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

14. The method of claim 13 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))). and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

15 The method of claim 14 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

16. The method of claim 14 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

17. The method of claim 11 wherein the encrypted digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the first digital signature is a signed certificate containing the content provider public key (PU-C) and signed by the root source private key (PR-R) (i.e., (CERT (PU-C) S (PR-R))).

18. The method of claim 1 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L). and wherein the second key is (PU-L).

19. The method of claim 18 wherein the second digital signature is a digital signature encrypted with the license provider private key (i.e., (S (PR-L))).

20. The method of claim 19 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), wherein the

license has a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))), and wherein deriving (PU-L) comprises:

deriving (PU-C) from a source available to the device;

obtaining (CERT (PU-L) S (PR-C)) from the license; and

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), to produce (PU-L) and also to validate the content provider.

21. The method of claim 20 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))), and wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

22. The method of claim 21 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))). and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

23. The method of claim 22 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

24. The method of claim 22 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

25. A method for a device to interdependently validate a piece of digital content and a corresponding digital license for rendering the digital content. the digital content being encrypted, the encrypted digital content being decryptable according to a decryption key (KD) and being packaged in a digital content package. the digital content package being provided by a content provider having a public key (PU-C) and a private key (PR-C), the digital license being provided by a license provider having

a public key (PU-L) and a private key (PR-L). the device having a public key (PU-D) and a private key (PR-D), the digital content package comprising:

the encrypted digital content; and

the content provider public key (PU-C) encrypted with the decryption key (KD) and signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C)));

the digital license comprising:

the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD)));

a digital signature from the license provider (without any attached certificate) based on (KD (DRL)) and (PU-D (KD)) and encrypted with the license provider private key (i.e., (S (PR-L))); and

a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C)));

the method comprising:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD);

obtaining (KD (PU-C) S (PR-C)) from the digital content package;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C);

applying (PU-C) to (S (PR-C)) to validate (KD (PU-C) S (PR-C)), thereby validating the digital content package;

obtaining (CERT (PU-L) S (PR-C)) from the license;

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), thereby validating the content provider, and also to obtain (PU-L);

obtaining (S (PR-L)) from the license; and

applying (PU-L) to (S (PR-L)), thereby validating the license.

26. The method of claim 25 wherein the digital content package further comprises a content / package ID identifying one of the digital content and the digital content package, and wherein the license further comprises the content / package ID of the corresponding digital content / digital content package, the method further comprising ensuring that the content / package ID of the license in fact corresponds to the content / package ID of the digital content / digital content package.

27. The method of claim 25 wherein the license further comprises a license rights description (DRL) specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising;

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

28. The method of claim 27 wherein the license rights description is encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD (DRL)) to obtain the license terms and conditions.

29. A computer-readable medium having computer-executable instructions for performing a method for a device to interdependently validate:

a digital content package having a piece of digital content in an encrypted form; and

a corresponding digital license for rendering the digital content, the method comprising:

deriving a first key from a source available to the device;

obtaining a first digital signature from the digital content package;

applying the first key to the first digital signature to validate the first digital signature and the digital content package;

deriving a second key based on the first digital signature;

obtaining a second digital signature from the license; and

applying the second key to the second digital signature to validate the second digital signature and the license.

30. The method of claim 28 wherein deriving the first key comprises:

obtaining a first encrypted key from the license;

applying a key available to the device to the first encrypted key to

decrypt the first encrypted key;

obtaining a second encrypted key from the digital content; and

applying the decrypted first encrypted key to the second encrypted key to produce the first key.

31. The method of claim 30 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first encrypted key is the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))).

32. The method of claim 30 wherein the device has a public key (PU-D) and a private key (PR-D), and wherein the key available to the device is (PR-D).

33. The method of claim 30 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the second encrypted key is the content provider public key (PU-C) encrypted with the decryption key (KD) (i.e., KD (PU-C)).

34. The method of claim 30 wherein the second encrypted key is the basis for the first digital signature.

35. The method of claim 29 wherein deriving the second key comprises:

obtaining a signed certificate from the license, the signed certificate having contents therein; and

applying the first key to the signature of the signed certificate to produce the contents of the certificate and also to validate the signature.

36. The method of claim 35 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the contents of the certificate is (PU-L).

37. The method of claim 36 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))).

38. The method of claim 36 wherein the digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the signed certificate is a certificate containing the license provider public key (PU-L) and signed by the root source private key (PR-R) (i.e., (CERT (PU-L) S (PR-R))).

39. The method of claim 29 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), and wherein

the first key is (PU-C).

40. The method of claim 39 wherein the encrypted digital content is decryptable according to a decryption key (KD), and wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))).

41. The method of claim 40 wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device;

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

42. The method of claim 41 wherein the device has a public key (PU-D) and a private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))). and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

43. The method of claim 42 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL))

to obtain the license terms and conditions.

44. The method of claim 42 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

45. The method of claim 39 wherein the encrypted digital content package is provided by a content provider authorized by a root source to provide the package, wherein the root source has a public key (PU-R) and a private key (PR-R) and wherein the first digital signature is a signed certificate containing the content provider public key (PU-C) and signed by the root source private key (PR-R) (i.e., (CERT (PU-C) S (PR-R))).

46. The method of claim 29 wherein the digital license is provided by a license provider having a public key (PU-L) and a private key (PR-L), and wherein the second key is (PU-L).

47. The method of claim 46 wherein the second digital signature is a digital signature encrypted with the license provider private key (i.e., (S (PR-L))).

48. The method of claim 47 wherein the digital content package is provided by a content provider having a public key (PU-C) and a private key (PR-C), wherein the license has a certificate containing the license provider public key (PU-L) and signed by the content provider private key (PR-C) (i.e., (CERT (PU-L) S (PR-C))), and wherein deriving (PU-L) comprises:

deriving (PU-C) from a source available to the device;

obtaining (CERT (PU-L) S (PR-C)) from the license; and

applying (PU-C) to (CERT (PU-L) S (PR-C)) to validate (CERT (PU-L) S (PR-C)), to produce (PU-L) and also to validate the content provider.

49. The method of claim 48 wherein the encrypted digital content is decryptable according to a decryption key (KD), wherein the first digital signature is based on the content provider public key (PU-C) encrypted with the decryption key (KD) and is signed by the content provider private key (PR-C) (i.e., (KD (PU-C) S (PR-C))), and wherein deriving (PU-C) comprises:

deriving (KD) from a source available to the device:

applying (KD) to (KD (PU-C) S (PR-C)) to produce (PU-C).

50. The method of claim 49 wherein the device has a public key (PU-D) and a

private key (PR-D), wherein the license has the decryption key (KD) encrypted with the device public key (PU-D) (i.e.,(PU-D (KD))), and wherein deriving (KD) comprises:

obtaining (PU-D (KD)) from the license;

applying (PR-D) to (PU-D (KD)) to produce (KD).

51. The method of claim 50 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the license rights description being encrypted with the decryption key (KD) (i.e., (KD (DRL))), the method further comprising applying (KD) to (KD(DRL)) to obtain the license terms and conditions.

52. The method of claim 50 wherein the license has a license rights description specifying terms and conditions that must be satisfied before the digital content may be rendered, the method further comprising:

evaluating the license terms and conditions to determine whether the digital content is permitted to be rendered in the manner sought;

if so, applying (KD) to the encrypted digital content to decrypt such encrypted digital content; and

rendering the decrypted digital content.

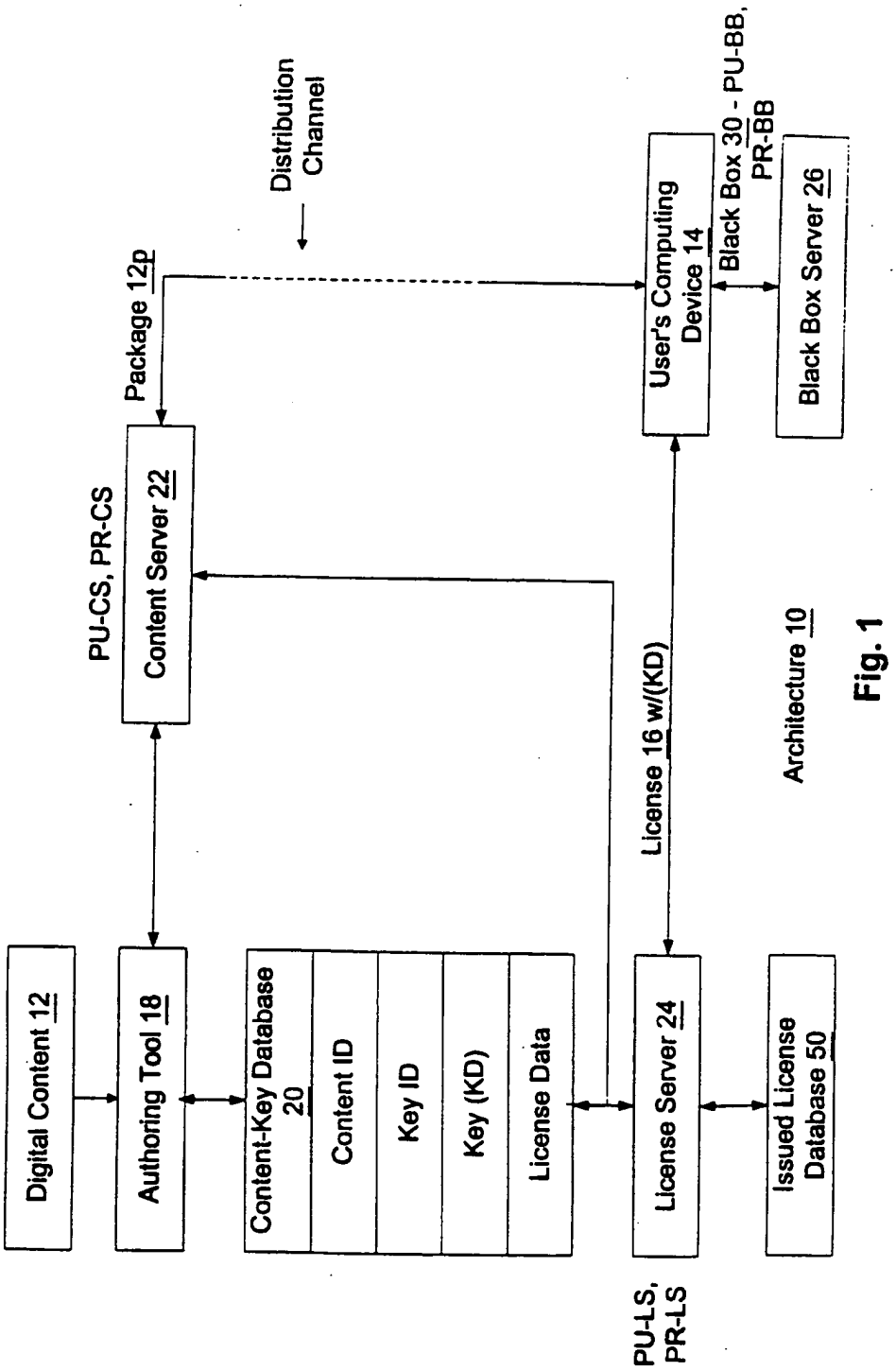

Distribution Channel
Package 12p
User's Computing Device 14
Black Box 30 - PU-BB, PR-BB
Black Box Server 26
PU-CS, PR-CS
Content Server 22
License 16 w/(KD)
Architecture 10
Digital Content 12
Authoring Tool 18
Content-Key Database 20
Content ID
Key ID
Key (KD)
License Data
License Server 24
Issued License Database 50
PU-LS, PR-LS

Fig. 1

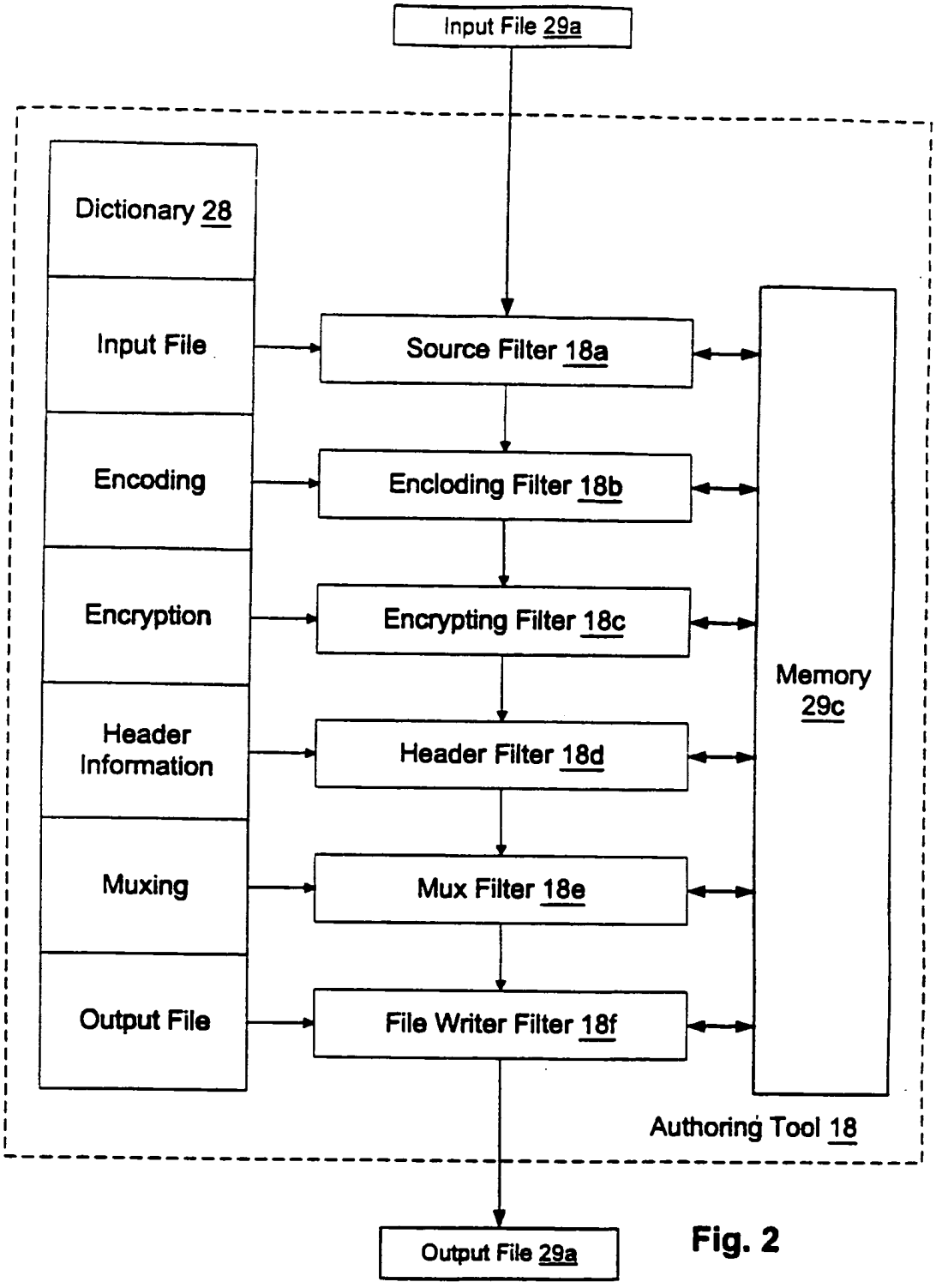

Input File 29a
Dictionary 28
Input File
Encoding
Encryption
Header Information
Muxing
Output File
Source Filter 18a
Encloding Filter 18b
Encrypting Filter 18c
Header Filter 18d
Mux Filter 18e
File Writer Filter 18f
Memory 29c
Authoring Tool 18
Output File 29a

Fig. 2

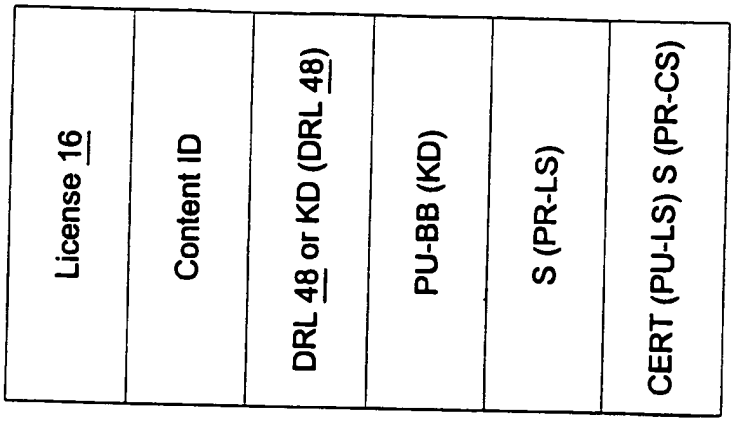

License 16
Content ID
DRL 48 or KD (DRL 48)
PU-BB (KD)
S (PR-LS)
CERT (PU-LS) S (PR-CS)

Fig. 8

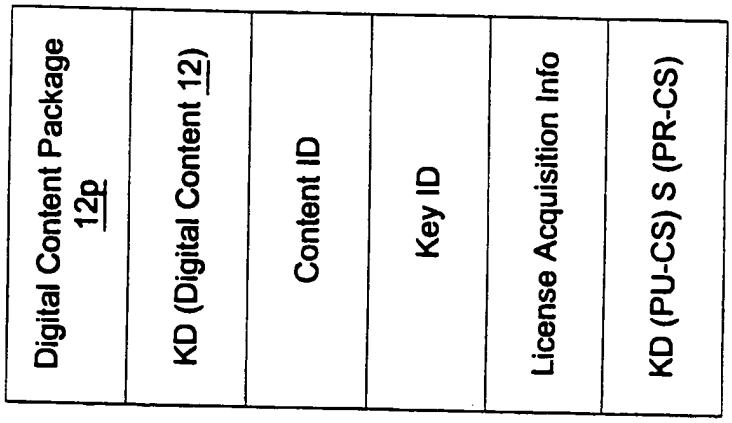

Digital Content Package 12p
KD (Digital Content 12)
Content ID
Key ID
License Acquisition Info
KD (PU-CS) S (PR-CS)

Fig. 3

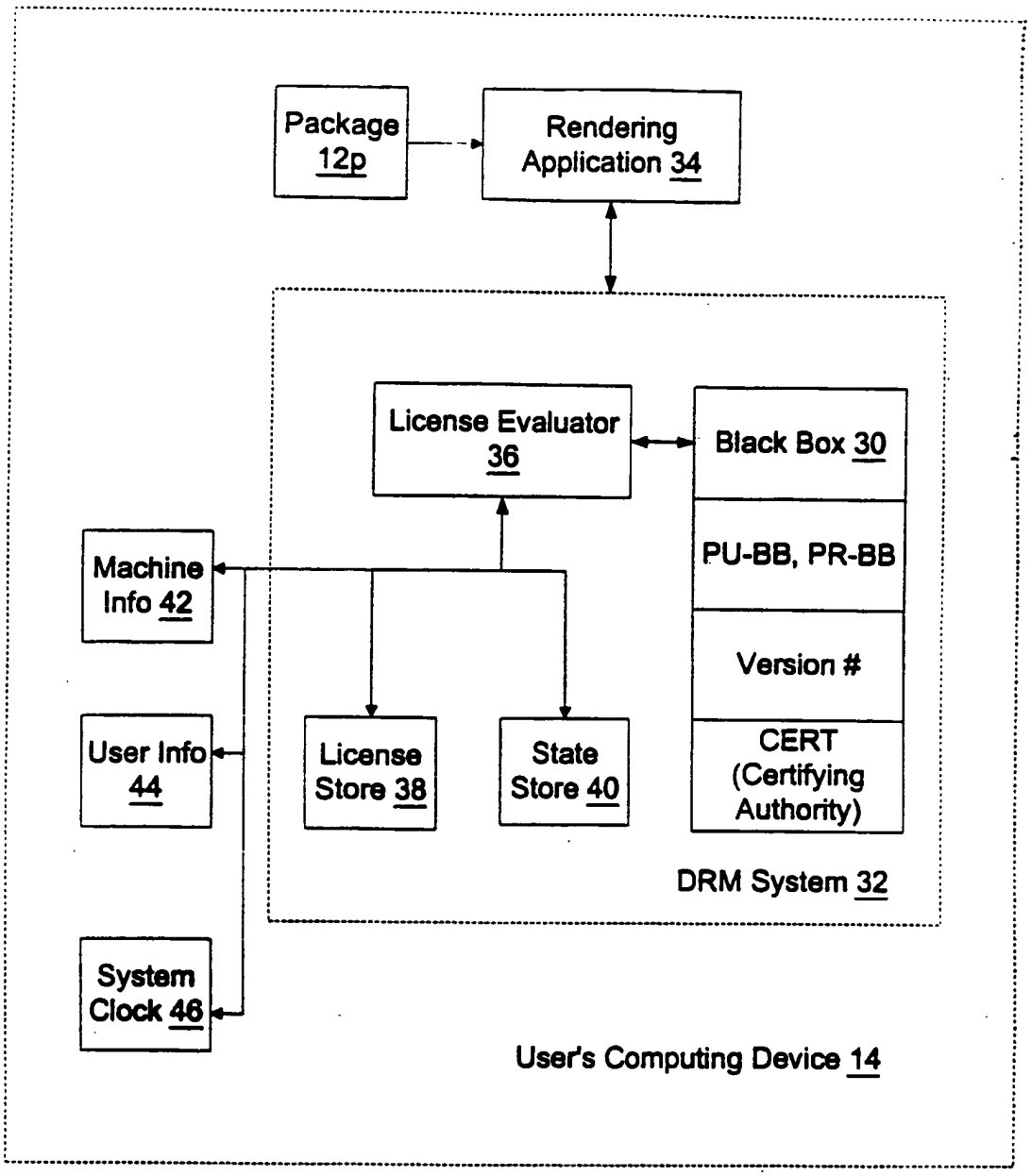

Package 12p
Rendering Application 34
License Evaluator 36
Black Box 30
PU-BB, PR-BB
Version #
CERT (Certifying Authority)
Machine Info 42
User Info 44
License Store 38
State Store 40
DRM System 32
System Clock 46
User's Computing Device 14

Fig. 4

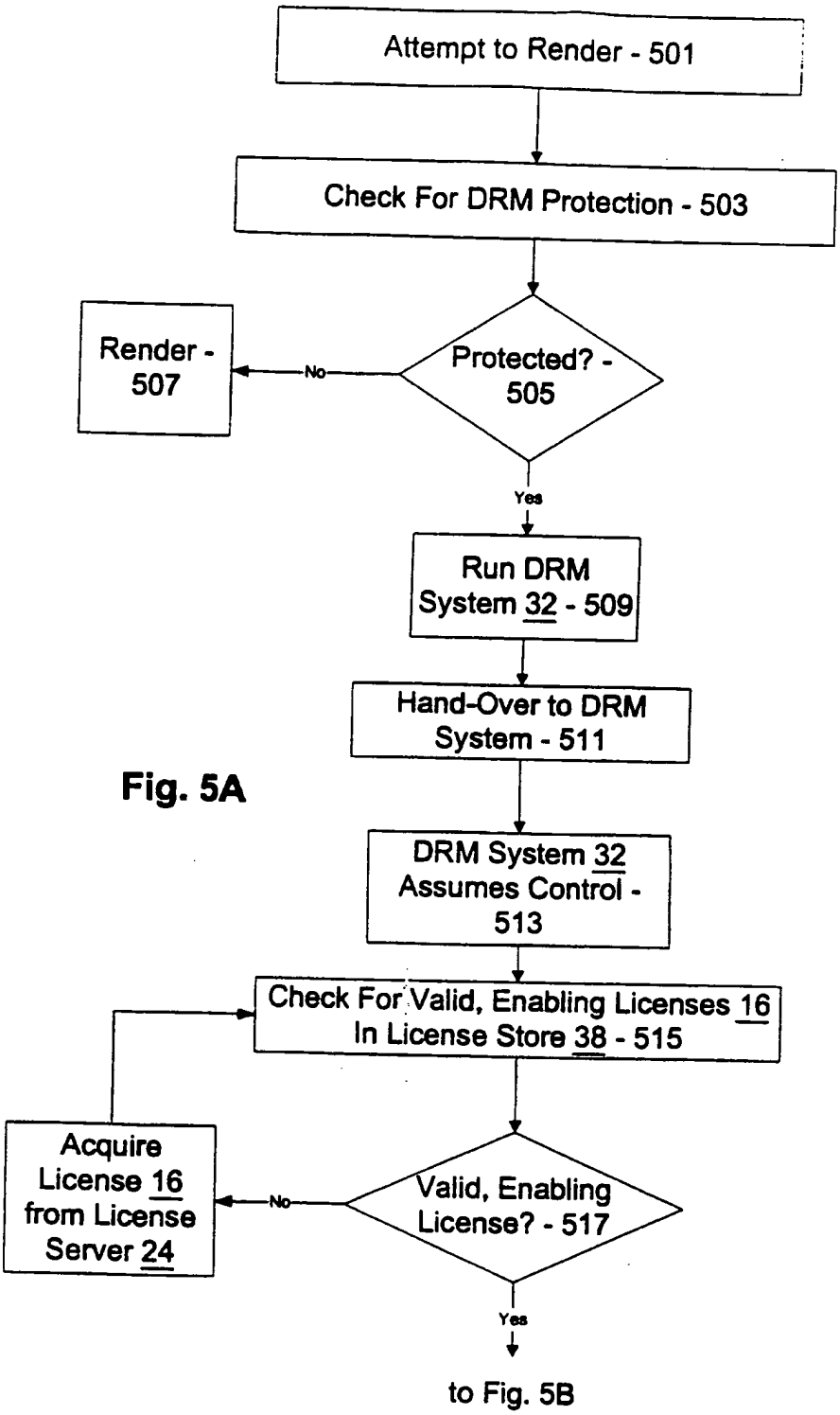

Attempt to Render - 501
Check For DRM Protection - 503
Protected? - 505
No
Render - 507
Yes
Run DRM System 32 - 509
Hand-Over to DRM System - 511
DRM System 32 Assumes Control - 513
Check For Valid, Enabling Licenses 16 In License Store 38 - 515
Valid, Enabling License? - 517
No
Acquire License 16 from License Server 24
Yes
to Fig. 5B

Fig. 5A

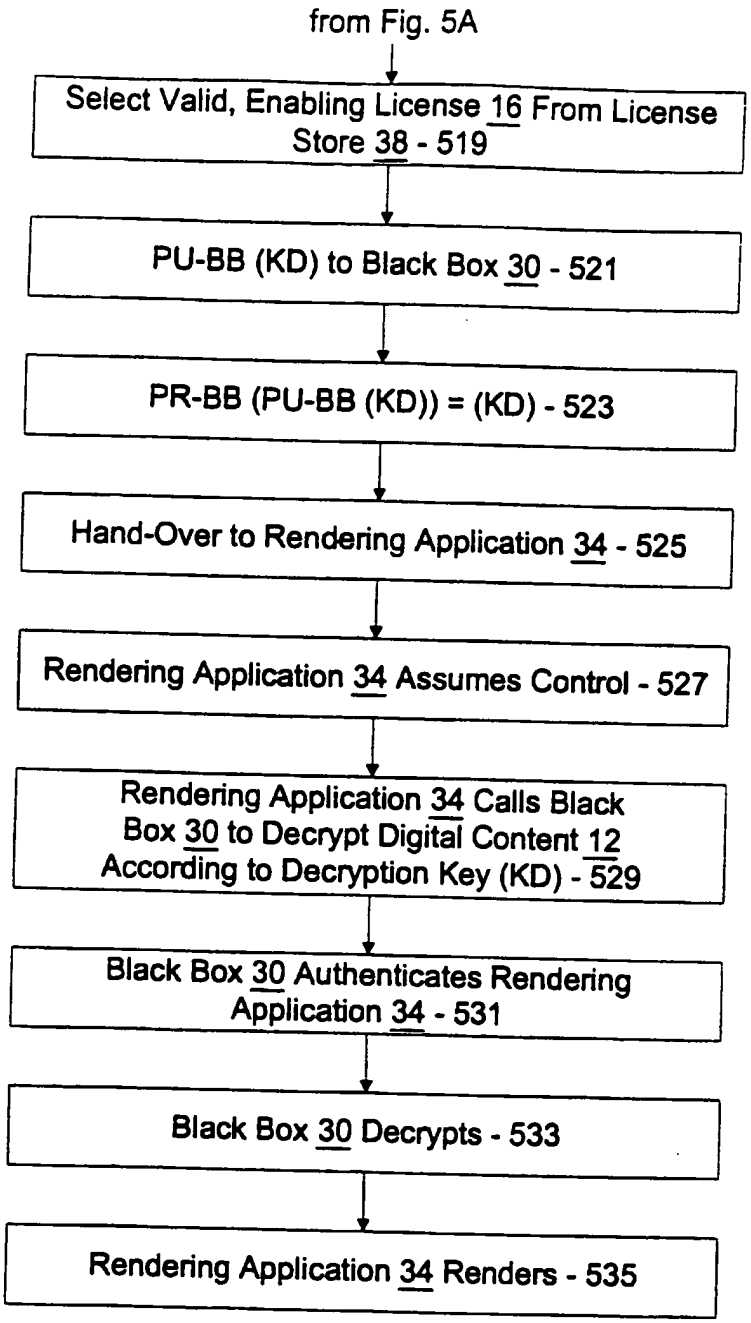

from Fig. 5A
Select Valid, Enabling License 16 From License Store 38 - 519
PU-BB (KD) to Black Box 30 - 521
PR-BB (PU-BB (KD)) = (KD) - 523
Hand-Over to Rendering Application 34 - 525
Rendering Application 34 Assumes Control - 527
Rendering Application 34 Calls Black Box 30 to Decrypt Digital Content 12 According to Decryption Key (KD) - 529
Black Box 30 Authenticates Rendering Application 34 - 531
Black Box 30 Decrypts - 533
Rendering Application 34 Renders - 535

Fig. 5B

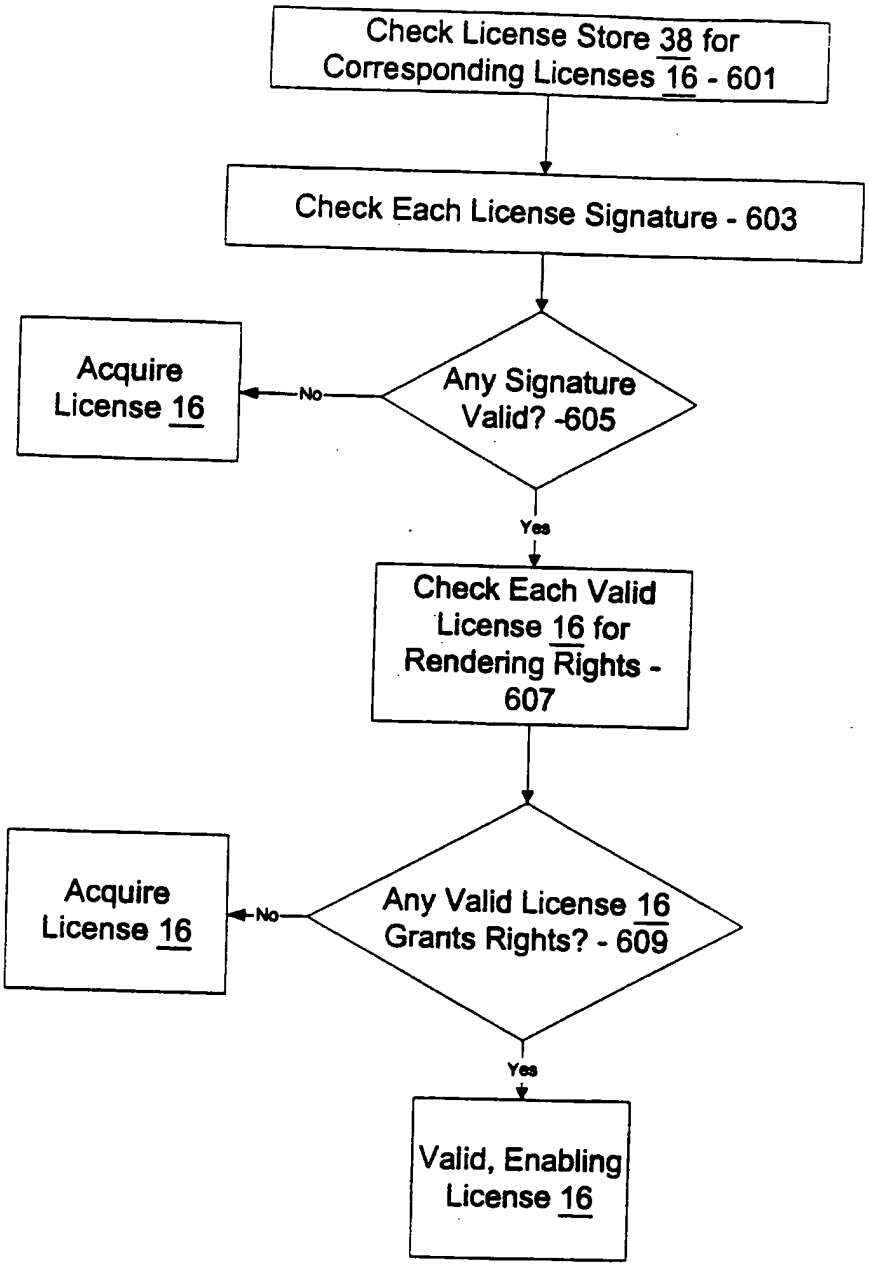

Check License Store 38 for Corresponding Licenses 16 - 601
Check Each License Signature - 603
Any Signature Valid? -605
No
Acquire License 16
Yes
Check Each Valid License 16 for Rendering Rights - 607
Any Valid License 16 Grants Rights? - 609
No
Acquire License 16
Yes
Valid, Enabling License 16

Fig. 6

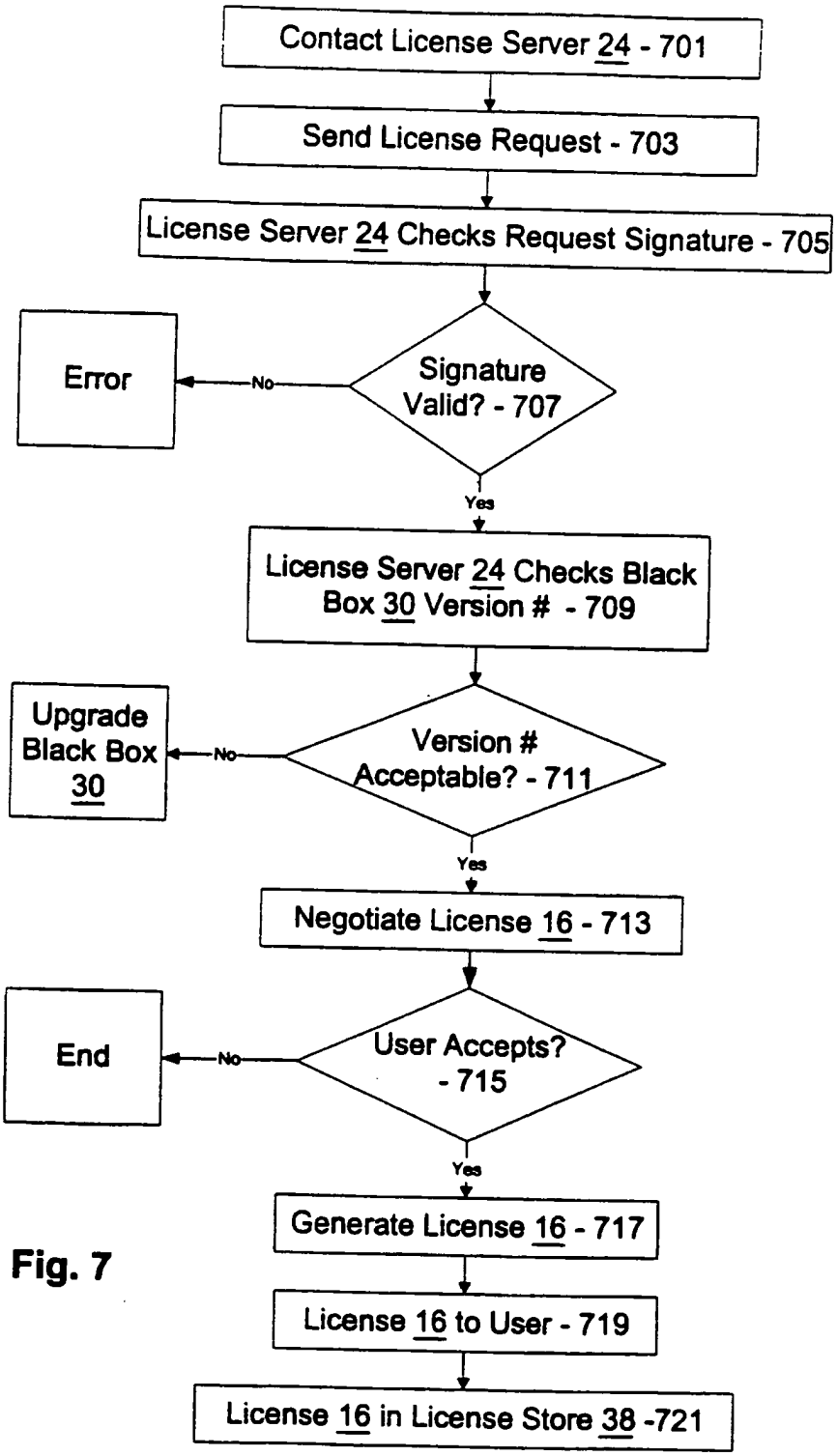

Contact License Server 24 - 701
Send License Request - 703
License Server 24 Checks Request Signature - 705
Signature Valid? - 707
No
Error
Yes
License Server 24 Checks Black Box 30 Version # - 709
Version # Acceptable? - 711
No
Upgrade Black Box 30
Yes
Negotiate License 16 - 713
User Accepts? - 715
No
End
Yes
Generate License 16 - 717
License 16 to User - 719
License 16 in License Store 38 -721

Fig. 7

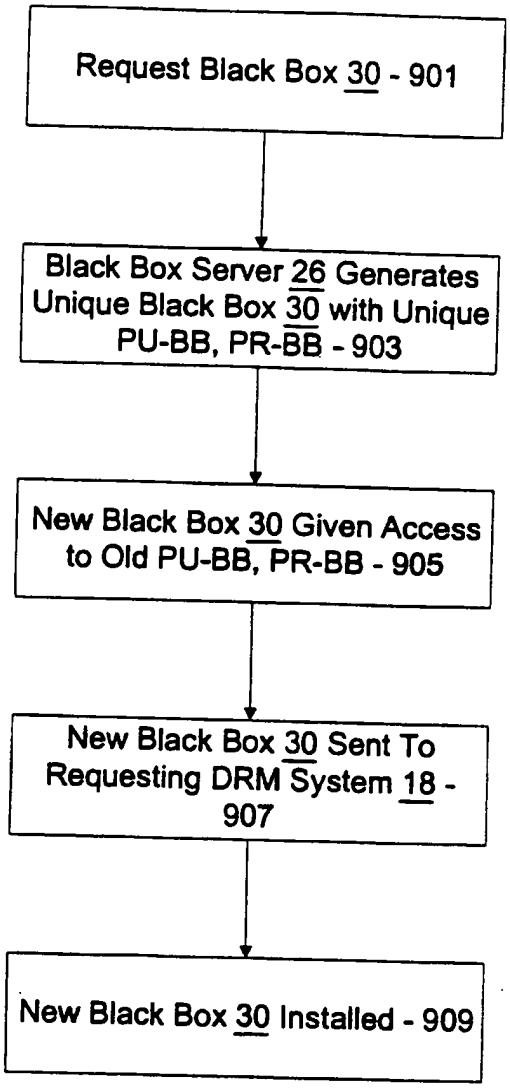

Request Black Box 30 - 901
Black Box Server 26 Generates Unique Black Box 30 with Unique PU-BB, PR-BB - 903
New Black Box 30 Given Access to Old PU-BB, PR-BB - 905
New Black Box 30 Sent To Requesting DRM System 18 - 907
New Black Box 30 Installed - 909

Fig. 9

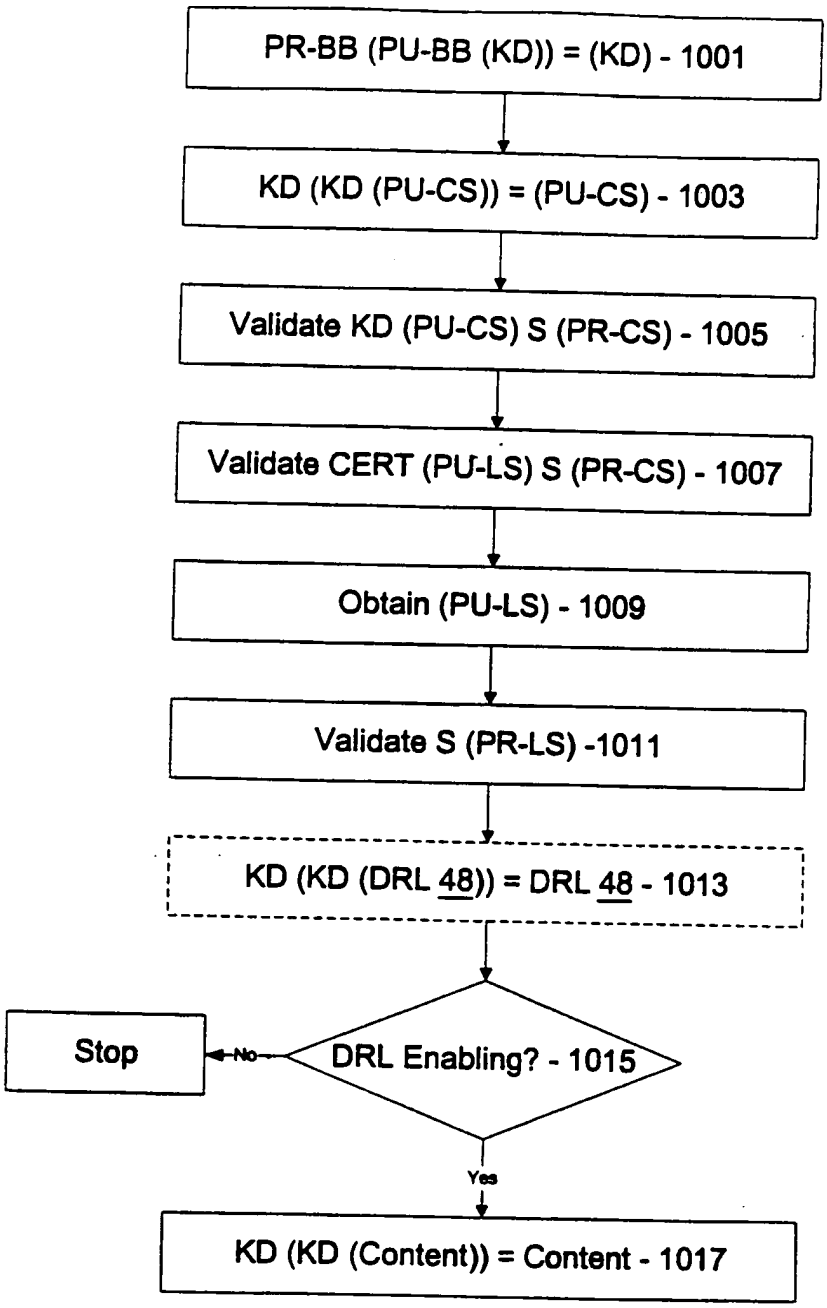

PR-BB (PU-BB (KD)) = (KD) - 1001
KD (KD (PU-CS)) = (PU-CS) - 1003
Validate KD (PU-CS) S (PR-CS) - 1005
Validate CERT (PU-LS) S (PR-CS) - 1007
Obtain (PU-LS) - 1009
Validate S (PR-LS) -1011
KD (KD (DRL 48)) = DRL 48 - 1013
Stop
No
DRL Enabling? - 1015
Yes
KD (KD (Content)) = Content - 1017

Fig. 10

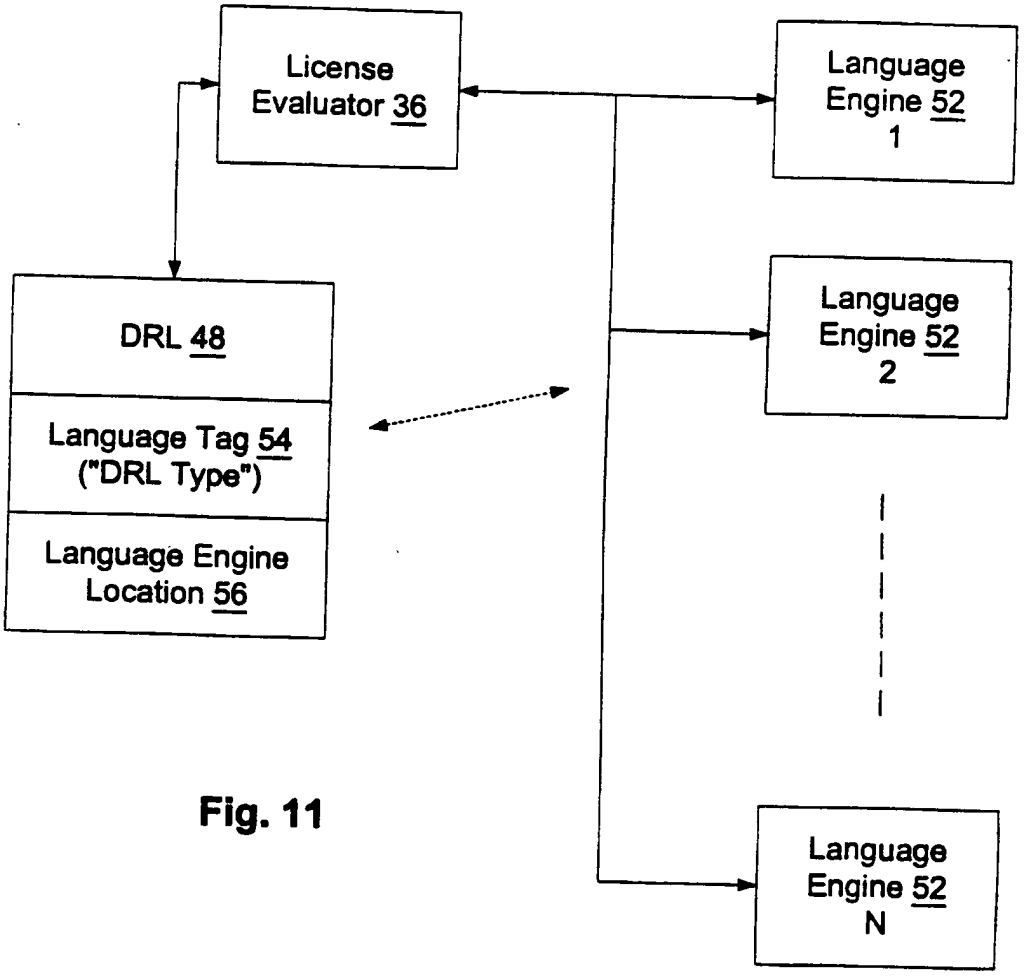

License Evaluator 36
Language Engine 52 1
Language Engine 52 2
Language Engine 52 N
DRL 48
Language Tag 54 ("DRL Type")
Language Engine Location 56

Fig. 11

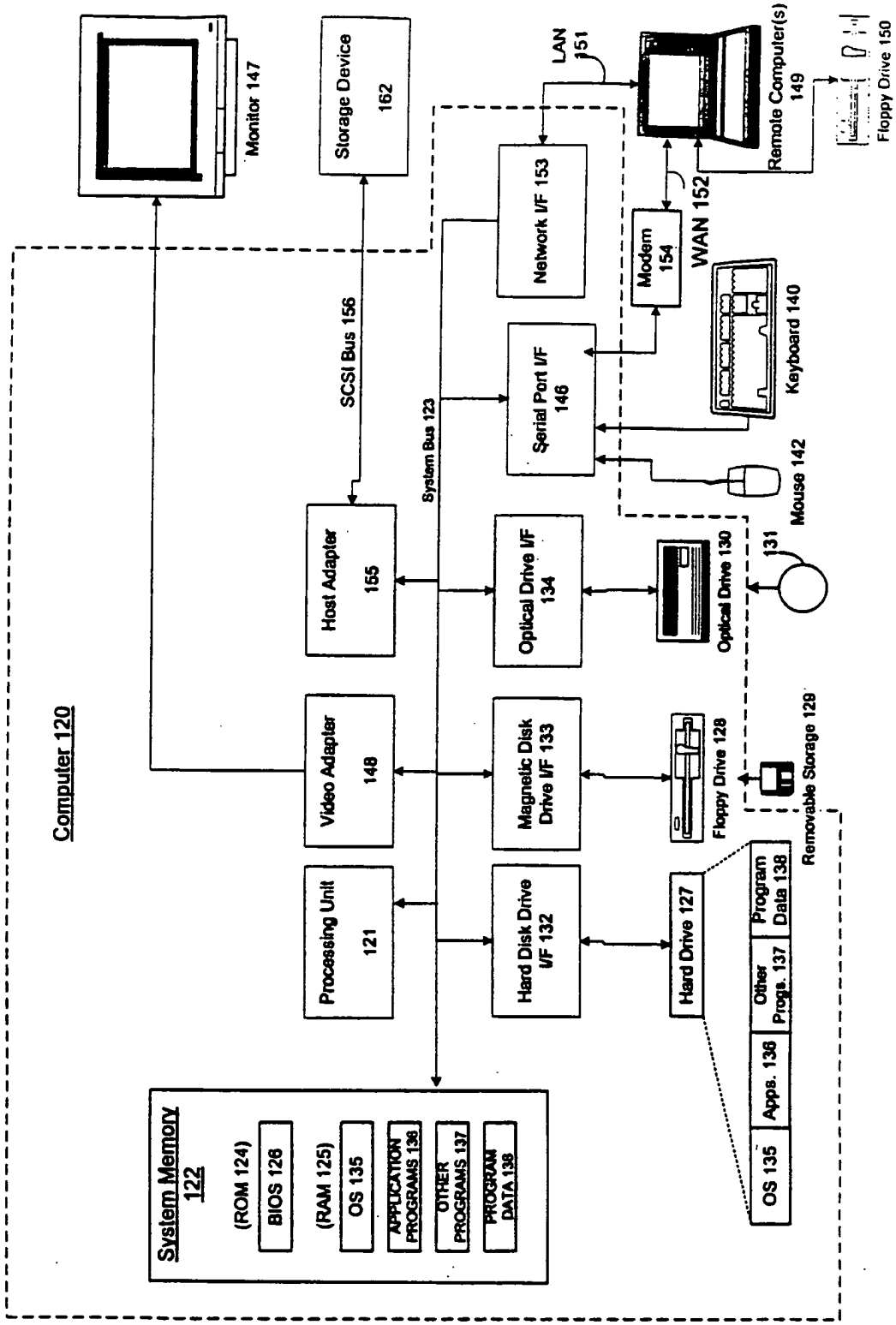

Computer 120
System Memory 122
(ROM 124)
BIOS 126
(RAM 125)
OS 135
APPLICATION PROGRAMS 136
OTHER PROGRAMS 137
PROGRAM DATA 138
Processing Unit 121
Video Adapter 148
Host Adapter 155
Hard Disk Drive I/F 132
Magnetic Disk Drive I/F 133
Optical Drive I/F 134
Serial Port I/F 146
Network I/F 153
Modem 154
WAN 152
LAN 151
SCSI Bus 156
System Bus 123
Monitor 147
Storage Device 162
Remote Computer(s) 149
Floppy Drive 150
Keyboard 140
Mouse 142
131
Optical Drive 130
Floppy Drive 128
Removable Storage 128
Hard Drive 127
OS 135
Apps. 136
Other Progs. 137
Program Data 138

Fig. 12

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72118 A1

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US00/10213

(22) International Filing Date: 13 April 2000 (13.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/316,717 21 May 1999 (21.05.1999) US

(71) Applicant: COMPAQ COMPUTERS INC. [US/US]; 10435 N. Tautau Avenue, Loc 200-16, Cupertino, CA 95014-3548 (US).

(72) Inventors: GLASSMAN, Steven, C.; 615 Palo Alto Avenue, Mountain View, CA 94041 (US). MANASSE, Mark, S.; 1270 Monterey Boulevard, San Francisco, CA 94127 (US).

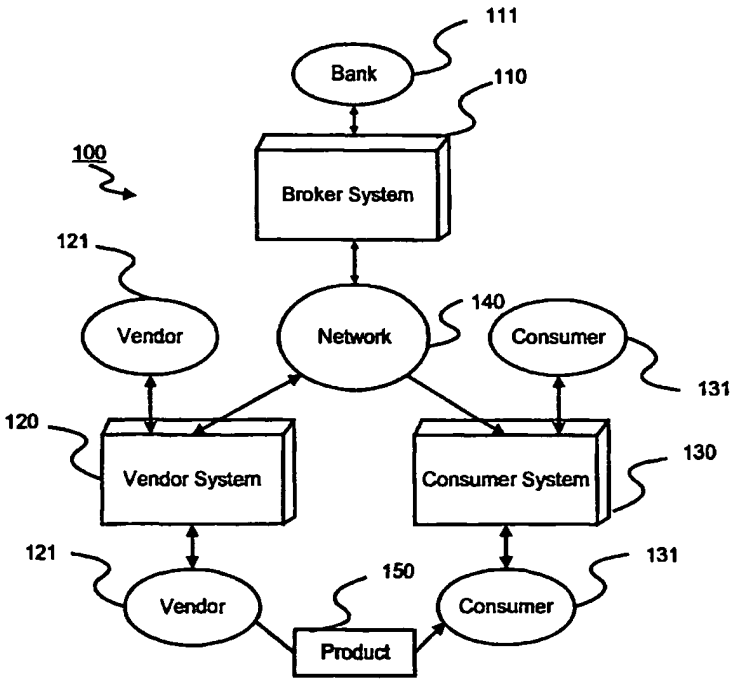
(74) Agents: GRANATELLI, Lawrence; Fenwick & West LLP, Two Palo Alto Square, Palo Alto, CA 94306 et al. (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK



(57) Abstract: An electronic commerce system and method enforces a license agreement for content on an open network (140) by restricting the number of consumers (131) that can concurrently access the content. A consumer (131) initially acquires vendor scrip, either from a broker or the vendor (121) itself. The consumer (131) presents the vendor scrip to the vendor (121) along with a request to access the content. In response, the vendor (121) gathers information about the consumer (131) to determine whether the consumer (131) belongs to the class allowed to access the content. The information may be gathered from the scrip or from other sources. If the consumer (131) belongs to the class, then the vendor (121) determines if a license to access the content is available. Generally, a license is available if the number of other consumers (131) having licenses to access the content is less than the maximum specified in the license agreement. If no licenses are available, the vendor (121) provides the consumer (131) with an estimate of when a license will be available. If a license is available, the vendor (121) directs the consumer (131) to obtain license scrip which allows the consumer (131) to access the content. The license scrip expires after a relatively brief period of time. When the consumer (131) uses the license scrip to access the content, the vendor (121) provides the consumer (131) with new license scrip having a later expiration time.

WO 00/72118 A1

MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND SYSTEM FOR ENFORCING LICENSES ON AN OPEN NETWORK

BACKGROUND

FIELD OF THE INVENTION

This invention relates generally to an electronic commerce system and more particularly to a commerce system supporting restricted use of a resource, and even more particularly to a commerce system supporting N-user license agreements.

BACKGROUND OF THE INVENTION

It is common for a library, corporation, or other organization to purchase content that will be made available to members of the organization. Often, the content is subject to a license restriction limiting distribution of the content. For example, a corporation may license or purchase a magazine and then distribute the magazine to interested employees. Typically, the corporation is restricted by the licensing agreement or copyright law from photocopying the magazine. Accordingly, the corporation must either obtain multiple copies of the magazine or circulate the single copy through the organization.

Similarly, the content licensed or purchased by the organization may be in electronic form. For example, the corporation may license a CD-ROM holding an electronic version of the magazine. While the CD-ROM can be loaded onto a server accessible to employees of the corporation via a computer network, the content may be restricted by an N-user license that forbids the corporation from allowing more than N users to simultaneously access the CD-ROM. To implement the restriction, software executing on the server tracks the number of people currently accessing the CD-ROM and blocks usage that exceeds the scope of the license.

In existing systems, the license control is performed by a combination of a specialized lock server and a client program. The lock server validates users' requests for access to the content and maintains the status of active users. The client program interacts with the lock server to acquire a lock and to provide access to the content.

There are many existing implementations of lock servers. However, they all are subject to one or more of the following undesirable restrictions:

each content source has its own, separate, and proprietary lock server;

the user's system already has the content (protected from direct access) and the client program gets the lock to access the content;

acquiring a lock is a complicated action; and/or

the set of valid users is limited.

For these reasons, existing lock servers are undesirable on an open network.

A lock server providing an N-user license on an open network should also support the following requirements:

an unrestricted set of potential users;

no single administrative domain covers all users;

the users do not need to have a separate user application for each source of content;

access to the content can be easily restricted; and

the content exists on the server and not with the user.

Accordingly, there is a need for a way to provide restricted access to electronic content that works with a wide variety of possible access schemes. Preferably, the solution will allow enforcement of an N-user license for content located on an open network like the Internet.

SUMMARY OF THE INVENTION

The above needs are met by a method and system for electronic commerce that uses special scrip - called "license scrip" - to provide temporary licenses to consumers accessing content. Scrip is primarily used as a form of electronic currency, however it can be more generally considered as a one-time token representing a general value. When scrip is used as an electronic currency, its value is monetary. When scrip is used as a temporary license, its value is the permission to access specific content. This permission may be unlimited or it may be for only a relatively brief period of time, say a few minutes to a few hours.

Accessing content with license scrip is very much like buying regular content with monetary scrip. Instead of having a price specified in monetary terms. Each page of content has a price (which may be zero) given in terms of license scrip. A consumer obtains license scrip from the vendor, preferably exchanging regular vendor scrip for the license scrip.

The vendor uses the license scrip to enforce an N-user license agreement - granting up to N people simultaneous access to the content. The vendor tracks the number and identity of consumers currently having licenses to access the content (i.e., consumers currently possessing valid license scrip).

A consumer initially lacks the license scrip needed to access the content. Upon receiving an access request from the consumer, the vendor determines whether a license is available. If a license is not available, the vendor tells the consumer to try again later and, optionally, provides the consumer with an estimate of when a license will be available.

If a license is available, then the vendor directs the consumer to obtain license scrip. Normally, the consumer obtains license scrip by requesting it from the vendor, but the consumer may get the license by any acceptable means. After receiving a license scrip

request. the vendor verifies that the consumer belongs to a class entitled to have a license. For example, if licenses are available to residents of only a certain state, the vendor ensures that the consumer resides in the state before granting the consumer a license.

If a license is available, then the vendor provides the consumer with the license scrip and remembers the granted license. The license scrip is preferably set to expire after a brief time period, but the duration of the license may vary depending upon business or legal concerns. To access content covered by the license, the consumer provides the license scrip when requesting content from the vendor. Each time the consumer accesses the content, the vendor returns replacement license scrip having the same or a later expiration time. Accordingly, the consumer can access the content as long as their license remains valid. When the consumer has not accessed the content for a while, the license scrip expires and the consumer can no longer access the content without obtaining new license scrip.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a top-level block diagram illustrating a computerized system for conducting electronic commerce;

FIGURE 2 is a block diagram illustrating a computer system used in the system of FIG. 1;

FIGURE 3 is a flow diagram illustrating the operations of the system of FIG. 1;

FIGURE 4 is a block diagram illustrating the data fields of a piece of scrip used in the system of FIG. 1;

FIGURE 5 is a diagram illustrating transactions between a consumer and a vendor utilizing license scrip to enforce an N-user license agreement according to the present invention; and

FIGURE 6 is a flow chart illustrating steps for determining whether to grant a license to a consumer.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of the present invention restricts access to electronic content through the use of an electronic commerce system. Accordingly, it is useful to describe the electronic commerce system before detailing how the system is utilized according to the present invention.

FIG. 1 shows a computerized system 100 for conducting electronic commerce. The system 100 includes a broker system 110, a vendor system 120, and a consumer system 130 interconnected by a communications network 140.

For clarity, the system 100 depicted in FIG. 1 shows only single broker, vendor, and consumer systems. In actual practice, any number of broker, vendor, and consumer systems can be interconnected by the network 140. The network 140 can be public or private, such as, for example, the Internet, an organization's intranet, a switched telephone system, a satellite linked network, or another form of network. The broker 111 using the broker system 110 can be a bank, a credit provider, an Internet service provider, a telephone company, or any institution the consumer trusts to sell electronic currency called "scrip."

The vendor system 120 is operated by a vendor 121. The vendor 121 provides products and/or content 150 of any type to consumers and, in one embodiment, provides content which is available by subscription. Each subscription page (i.e., page of data that is available for "purchase") has a price of zero but requires a special type of scrip, called "subscription scrip," before it can be accessed. Since the price of a page is zero, the consumer 131 can "purchase" an unlimited number of pages once the consumer 131 has the

proper subscription scrip 330. The subscription expires when the subscription scrip 330 expires.

A consumer 131 can use the consumer computer system 130 to electronically acquire the products or content 150 of the vendor 121. As used herein, "consumer" refers to an organization such as a library or corporation, a member of the organization, such as a librarian or an employee, or an individual, such as a person visiting a library or a home computer user. Of course, actions attributed to the organization are usually performed by a member of the organization.

A computer system 200 suitable for use as the broker, vendor, and consumer systems is shown in FIG. 2. The computer system 200 includes a central processing unit (CPU) 210, a memory 220, and an input/output interface 230 connected to each other by a communications bus 240. The CPU 210, at the direction of users 250, e.g. brokers, vendors, and/or consumers, executes software programs, or modules, for manipulating data. The programs and data can be stored in the memory 220 as a database (DB) 221. The DB 221 storing programs and data on the consumer computer system 130 is referred to as a "wallet." In a preferred embodiment of the present invention described herein, many of the operations attributed to the consumer are, in fact, performed automatically by the wallet 221.

The memory 220 can include volatile semiconductor memory as well as persistent storage media, such as disks. The I/O interface 230 is for communicating data with the network 140, the users 250, and other computer system peripheral equipment, such as printers, tapes, etc.

The computer system 200 is scaled in size to function as the broker, vendor, or consumer systems. For example, when scaled as the consumer computer system 130, the computer system 200 can be a small personal computer (PC), fixed or portable. The

configurations of the computer system 200 suitable for use by the broker 111 and the vendor 121 may include multiple processors and large database equipped with "fail-safe" features. The fail-safe features ensure that the database 221 is securely maintained for long periods of time.

FIG. 3 shows an operation of the electronic commerce system 100. The consumer 131 uses currency to purchase electronic broker scrip 320 generated by the broker 111. Here, purchasing means that upon a validation of the authenticity of the consumer 131 and the consumer's currency 310, the broker system 110 generates signals, in the form of data records. The signals are communicated, via the network 140, to the consumer system 130 for storage in the wallet 221 of the memory 220 of the consumer system 130.

The scrip is stamped by the generator of the scrip to carry information that is verifiable by the originator, and any other system that has an explicit agreement with the originator. In addition, each scrip is uniquely identifiable and valid at only a single recipient. After a single use, the recipient of the scrip can invalidate it, meaning that the signals of the data record are no longer accepted for processing by the recipient computer system.

In one embodiment, the consumer 131 exchanges the broker scrip 320 with the broker 111 for vendor scrip 330. To complete this transaction, the broker system 110 executes licensed software programs which generate scrip 330 for consumers as needed. Alternatively, the broker 111, in a similar transaction 303, exchanges currency 310 for bulk vendor scrip 330 which is then sold to consumers.

In another embodiment, the consumer 131 exchanges currency with the vendor 121 for regular vendor. In this latter embodiment, there is no need for a broker 111. In addition, the vendor scrip may be free, meaning that the consumer 131 does not need to exchange currency for the scrip.

The consumer 131, in a transaction 304, provides the scrip 330 to the vendor 121. The vendor 121 checks the stamp of the scrip 330 to verify its authenticity, and also checks to make sure the value of the scrip covers the requested content and has not expired. Approval of the transaction results in the delivery of the desired content 150 to the consumer 131. The vendor 121 can also return 304 modified scrip 330 to the consumer 131 as change.

FIG. 4 is a block diagram illustrating the data fields of a single piece of scrip 400. The scrip 400 is logically separated into seven data fields. The Vendor field 410 identifies the vendor for the scrip 400. The Value field 412 gives the value of the scrip 400. The scrip ID field 414 is the unique identifier of the scrip. The Customer ID field 416 is used by the broker 111 and vendor 121 to verify that the consumer has the right to spend the scrip. The Expires field 418 gives the expiration time for the scrip 400. The Props field 420 holds consumer properties, such as the consumer's age, state of residence, employer, etc. Finally, the Stamp field 422 holds a digital stamp and is used to detect tampering with the scrip 400.

The present invention uses "license" scrip, which can be thought of as special purpose scrip having a short period of validity. A consumer with license scrip has a license to view the content covered by the license until the scrip expires.

FIG. 5 is a diagram illustrating transactions between a consumer 510 and a vendor 512 utilizing license scrip to enforce an N-user license agreement according to the present invention. In the transactions of FIG. 5, the vendor 512, for example, can be a library located at a state university. Assume the library purchases a four user license for a CD-ROM and makes the CD-ROM available to other terminals in the library via a local area network and residents of the state via the Internet. To conform with the license, the library must ensure that no more than four consumers are simultaneously accessing the CD-ROM. In this

example, the library is the vendor 512 and the people who can access the CD-ROM, either in the library or elsewhere, are the consumers 510.

In another example, a newspaper publisher operates a web site. Assume that a corporation purchases a 20 user license allowing up to 20 people from the corporation to simultaneously access content on the web site. To police its license, the publisher tracks the users of its web site and block users who are not licensed or who have exceeded the scope of the applicable license. Accordingly, the newspaper publisher is the vendor 512 and the corporation and its employees are the consumers 510.

Although neither the illustrated transactions nor the above examples directly utilize a broker, there may be circumstances where it is desirable to use a broker 111 to perform one or more of the transactions described below. Those of ordinary skill in the art will understand that certain transactions attributed to the consumer or the vendor can be performed instead by a broker 111. For example, the library and/or newspaper may issue vendor and license scrip directly or rely on a third-party broker for this task.

Turning to FIG. 5, the consumer 510 initially requests 520 content from the vendor 512 without valid license scrip. In response, the vendor 512 checks to determine whether there is an available license (i.e., whether an additional consumer is allowed to view the content under the license). Preferably, the vendor 512 maintains a data structure associated with the licensed content that can be quickly scanned to determine whether a license is available. In one embodiment, this data structure is a simple N-entry array, with each entry holding fields for the expiration time and Customer ID of the consumer 510 having the license. As licenses are granted, the vendor 512 fills in the array until no more entries are available.

If no licenses are available, then the vendor 512 instructs 522 the consumer 510 to try again later. In one embodiment, the vendor 512 scans the data structure to determine when the first license may become available and provides the consumer 510 with that time as a suggestion of when to try to access the content again. If a license is available, then the vendor 512 instructs the consumer 510 to go and obtain license scrip.

In response, the consumer 510 attempts 524 to obtain license scrip from the vendor 512. The vendor 512 determines whether the consumer 510 is entitled to a license (i.e., entitled to view the content). FIG. 6 is a flow chart 600 illustrating steps for determining whether to grant license scrip to the consumer 510. When the vendor 512 receives the request from the consumer 510, the vender retrieves 610 information about the consumer. The vendor 514 may retrieve this information by asking the consumer 510 to provide it, from the scrip used to request the license scrip, from a "cookie" on the consumer's computer system, or from a table of information shared by the vendor 512 and the consumer 510 or a broker 111. Additionally, the wallet 221 on the consumer's computer system 130 may be configured to automatically provide information about the consumer 510 when requested by a vendor 512. Depending on the needs of the vendor 512 and the license agreement for the content, the information that may be gathered in this manner includes whether the consumer 510 is a member of an organization, the state of residence of the consumer, the consumer's age, or any other information that is relevant to determining whether to provide access to the consumer 510.

The vendor 512 uses this information to determine 612 whether the consumer belongs to a class that has access to the content held by the vendor 512. If the consumer does not belong to a class having access, for example, if the consumer is not a state resident, then the

vendor denies 614 access to the consumer 510. Preferably, the vendor 512 directs the consumer 510 to a web page explaining why access was denied.

If the consumer 510 belongs to a class having access, the vendor 512 scans the data structure identifying the current licensees of the content and determines 616 whether an additional license is available. Since there may be a delay between the time the consumer 510 is told to buy license scrip and when the wallet 221 tries to buy the scrip, it is possible that the available license may have been acquired by another consumer during that time. If no licenses are available, then the consumer 510 is told to try again later and optionally given a time when a license may be available.

If a license is available, then the vendor 512 grants 618 the license to the consumer 510. The vendor 512 provides 526 the consumer with license scrip that allows the consumer 510 to access the content. The license scrip preferably has a relatively short validity period, say a few minutes to an hour, and allows the consumer 510 full access to the licensed material for the duration of the scrip. The choice of expiration time for the scrip is a business or legal decision. Since the intention of the license scrip is to hold onto one license slot while the consumer 510 is actively using the content, the duration of the license should cover the time that the consumer 510 is expected to be active. In another embodiment, the duration of the scrip is determined, at least in part, by the type of content accessed by the consumer 510. In addition, the vendor 512 preferably records data about the granted license, including the Customer ID of the consumer 510 and the expiration time of the license in the appropriate data structure.

Each time the consumer 510 wishes to access 528 content held by the vendor 512, the consumer provides the license scrip to the vendor. If the scrip is expired or otherwise invalid, then the consumer's request for access is treated as a request without scrip as illustrated by

transaction 520. If the scrip is valid, then the vendor 512 allows the consumer 510 to access the content. In addition, the vendor 512 provides 530 the consumer 510 with replacement license scrip having an updated expiration time. Typically, the updated expiration time is later than the old expiration time, although it can be the same or earlier. In one embodiment, the vendor 512 grants the consumer 510 less additional time each time the vendor issues new license scrip to ensure that the consumer's license eventually expires and other consumers may eventually access the content. The vendor 512 also updates its data structure to reflect the new expiration date of the consumer's license.

Periodically, the vendor 514 preferably scans the data structure to determine whether any licenses have expired. If so, the entry is purged from the data structure, thereby freeing up a license for another consumer 510. Accordingly, the present invention uses license scrip to enforce an N-user license agreement.

It should be understood that FIG. 5 illustrates only one possible set of transactions. FIG. 3. in combination with FIG. 5, provides insight into other possible transactions. For example, a corporation could purchase an N-user license agreement from a broker 111 to access content on a vendor's system 120. The broker 111 can verify that the corporation is entitled to a license and then issue the license scrip from a special scrip series corresponding to the number of users covered by the license. The vendor 121 knows from the scrip series to restrict access from consumers using that license scrip.

Having described a preferred embodiment of the invention, it will now become apparent to those skilled in the art that other embodiments incorporating its concepts may be provided. It is felt therefore, that this invention should not be limited to the disclosed invention, but should be limited only by the spirit and scope of the appended claims.

CLAIMS

We claim:

1. A method of restricting simultaneous access to content, comprising the steps of:

receiving a request to access the content from a consumer;

determining whether the consumer is entitled to access the content; and

responsive to a positive determination, providing the consumer with license scrip allowing access to the content.

2. The method of claim 1, wherein the request to access the content is accompanied by license scrip having an expiration time and wherein the providing step provides the consumer with additional license scrip having an updated expiration time.

3. The method of claim 1, wherein the license scrip has an expiration time and further comprising the steps of:

receiving a second request to access the content from the consumer, the second request including the license scrip; and

responsive to the second request, providing the consumer with replacement license scrip having an updated expiration time.

4. The method of claim 1, wherein the step of determining whether the consumer is entitled to access the content comprises the steps of:

determining whether the consumer belongs to a class having access to the content; and

determining whether a license to access the content is available.

5. The method of claim 4, wherein the step of determining whether the consumer belongs to a class having access to the content comprises the step of:

determining information about the consumer from scrip utilized to request access to the content.

6. The method of claim 4, wherein the step of determining whether a license to access the content is available comprises the steps of:

determining a number of consumers that have licenses to access the content; and

determining a number of allowed licenses;

wherein a license to access the content is available if the number of consumers that have licenses to access the content is less than the number of allowed licenses.

7. The method of claim 4, further comprising the step of:

responsive to a determination that no licenses to access the content are available, providing the consumer with an estimate of when a license will be available.

8. A computer program product having computer-readable instructions embodied thereon for restricting access to content stored on a computer system, the computer-readable instructions comprising instructions for:

receiving a request to access the content stored on the computer system, the request accompanied by scrip;

determining whether the scrip authorizes access to the content;

responsive to a determination that the scrip does not authorize access to the content, determining whether scrip authorizing access to the content is available; and

responsive to a determination that scrip authorizing access to the content is available, providing the scrip.

9. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that the scrip authorizes access to the content, providing replacement scrip having an updated expiration time.

10. The computer program product of claim 8, wherein the instructions for determining whether the scrip authorizes access to the content further comprise computer instructions for:

determining a type of the scrip accompanying the request; and

responsive to a determination that accompanying scrip is license scrip, determining whether the license scrip has expired, wherein unexpired license scrip authorizes access to the content.

11. The computer program product of claim 8, wherein the instructions for determining whether scrip authorizing access to the content is available comprise instructions for:

determining a maximum number of requesters that can be authorized to access the content;

determining whether a current number of requesters authorized to access the content is less than the maximum number of requesters; and

responsive to a determination that the current number of requesters authorized to access the content is less than the maximum number of requesters, determining that scrip authorizing access to the content is available.

12. The computer program product of claim 8, further comprising instructions for:

responsive to a determination that scrip authorizing access to the content is not available, calculating an estimate of when the scrip authorizing access will be available.

13. A computer system for limiting a number of users that can access content stored on a server associated with the computer system, the computer system comprising:

a module for receiving a request from a user to access the content stored on the server;

a module for determining the number of users currently having rights to access the content; and

a module for providing the user with license scrip if the number of users currently having rights to access the content is less than a number of users allowed to access the content, the license scrip granting the user the right to access the content.

14. The system of claim 13, wherein the module for determining the number of users currently having access rights to content comprises:

a module for scanning a data structure stored in a memory of the computer system, the data structure having one or more entries indicating the number of users having access rights to the content.

15. The computer system of claim 14, wherein the data structure indicates when users' rights to access the content expire, further comprising:

a module for purging the entries of users whose right to access the content has expired.

16. The system of claim 13, wherein only a privileged class can access the content, further comprising:

a module for determining whether the user is a member of the privileged class.

17. The system of claim 13, wherein the license scrip grants the user the right to access the content until an expiration time.

18. The system of claim 17, further comprising:

a module for receiving a second request from the user to access the content stored on the server accompanied by the license scrip; and

a module for providing the user with replacement license scrip having a later expiration time.

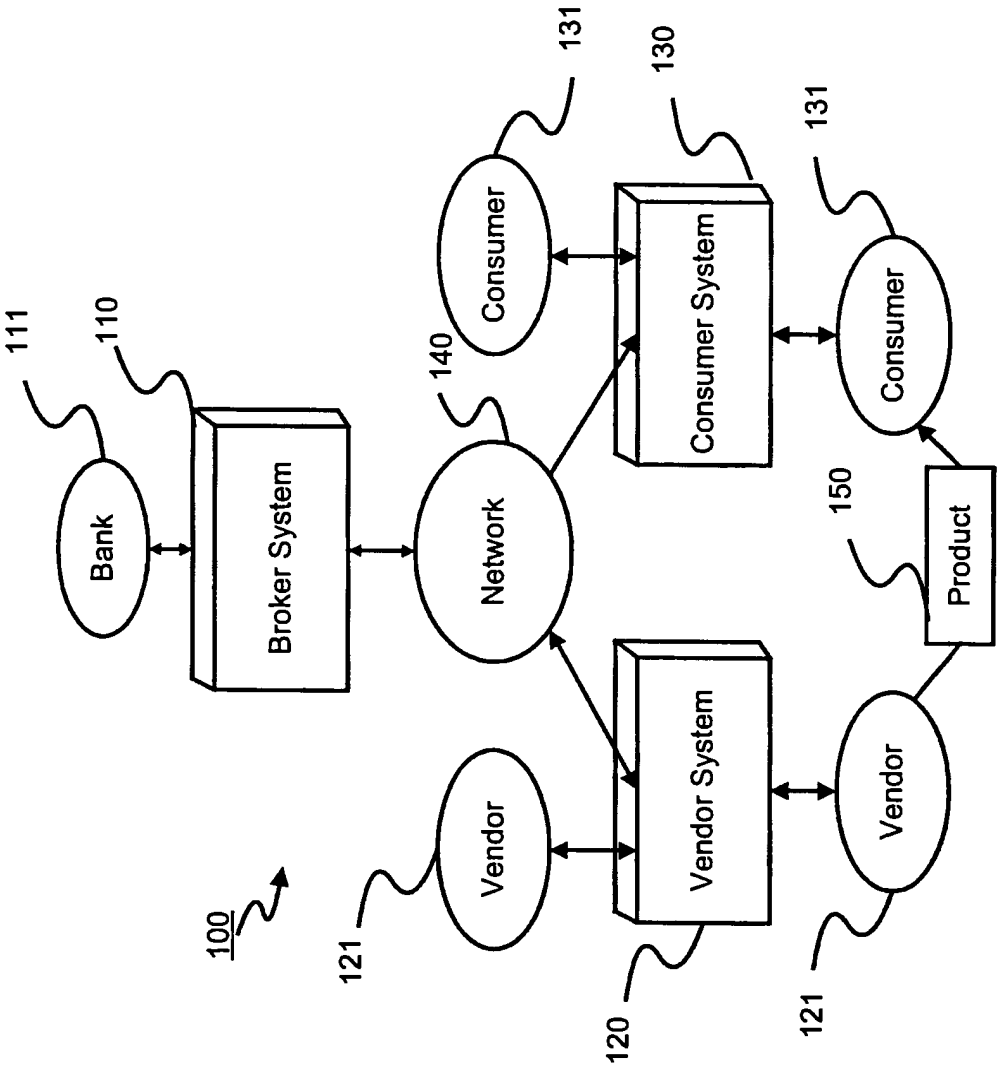


FIG. 1

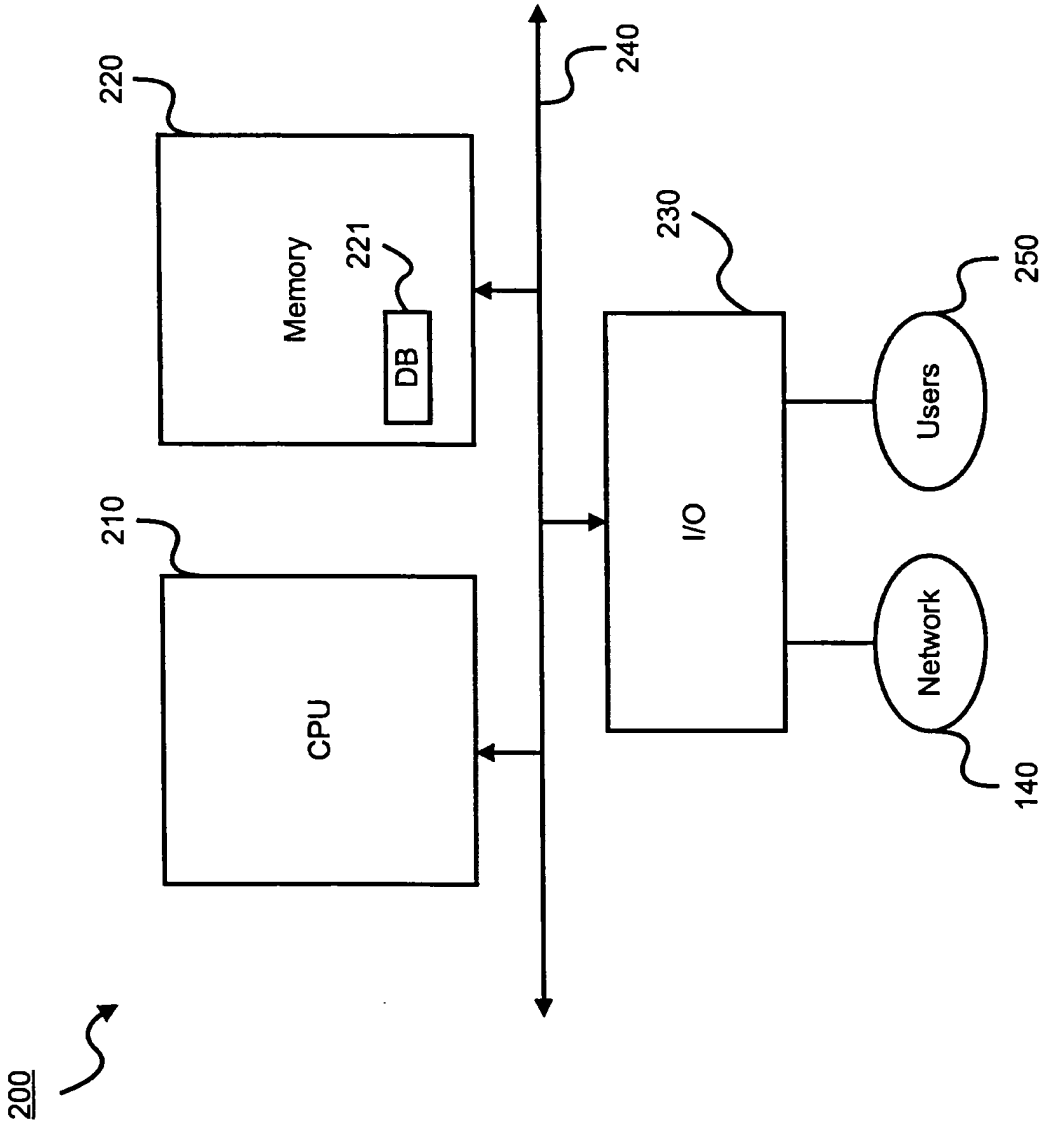


FIG. 2

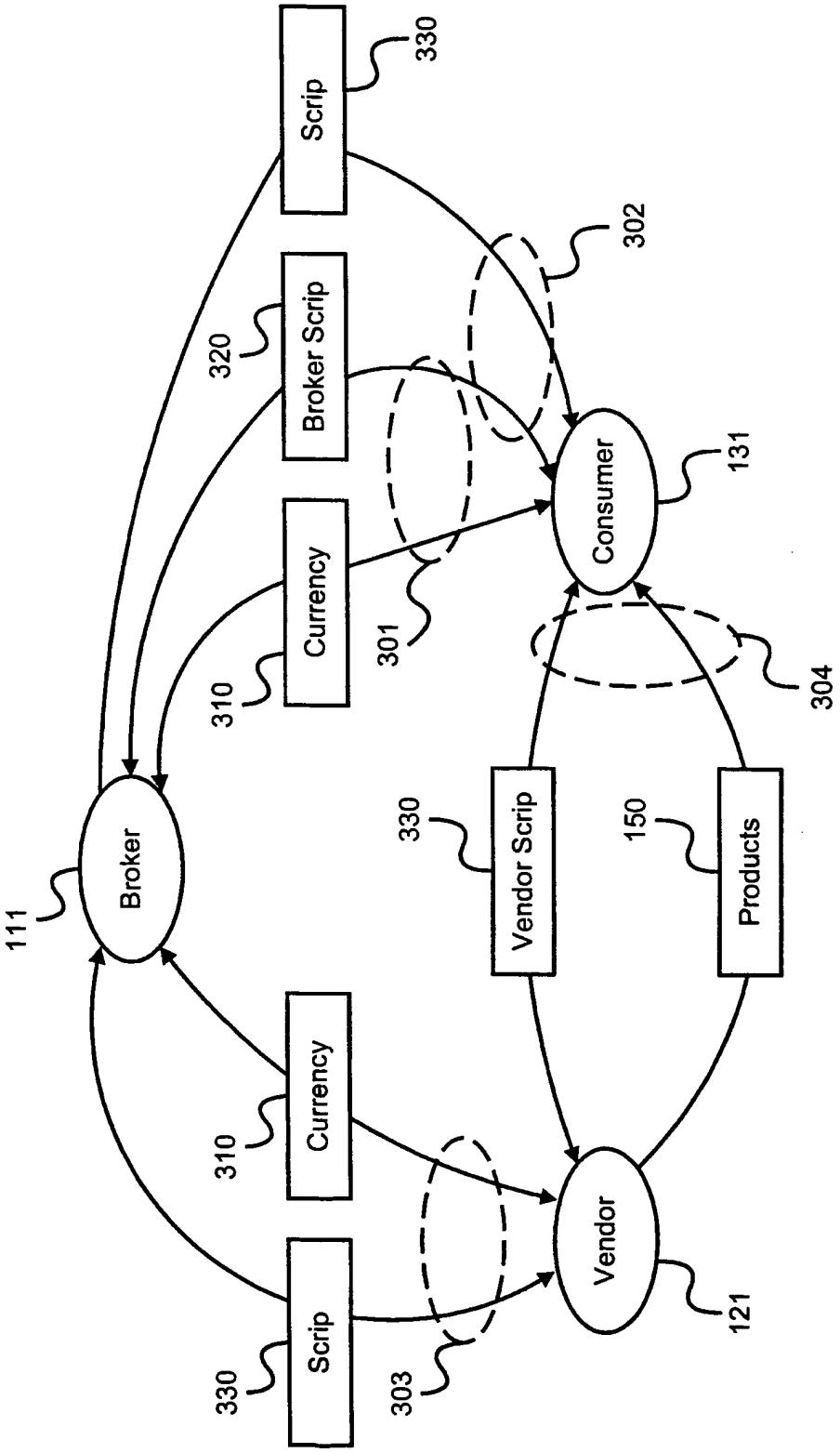


FIG. 3

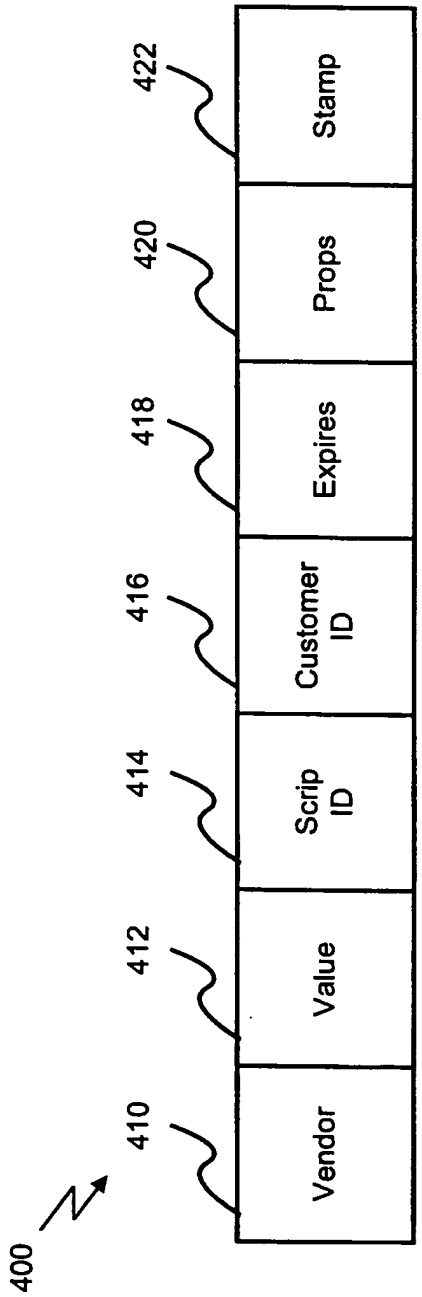


FIG. 4

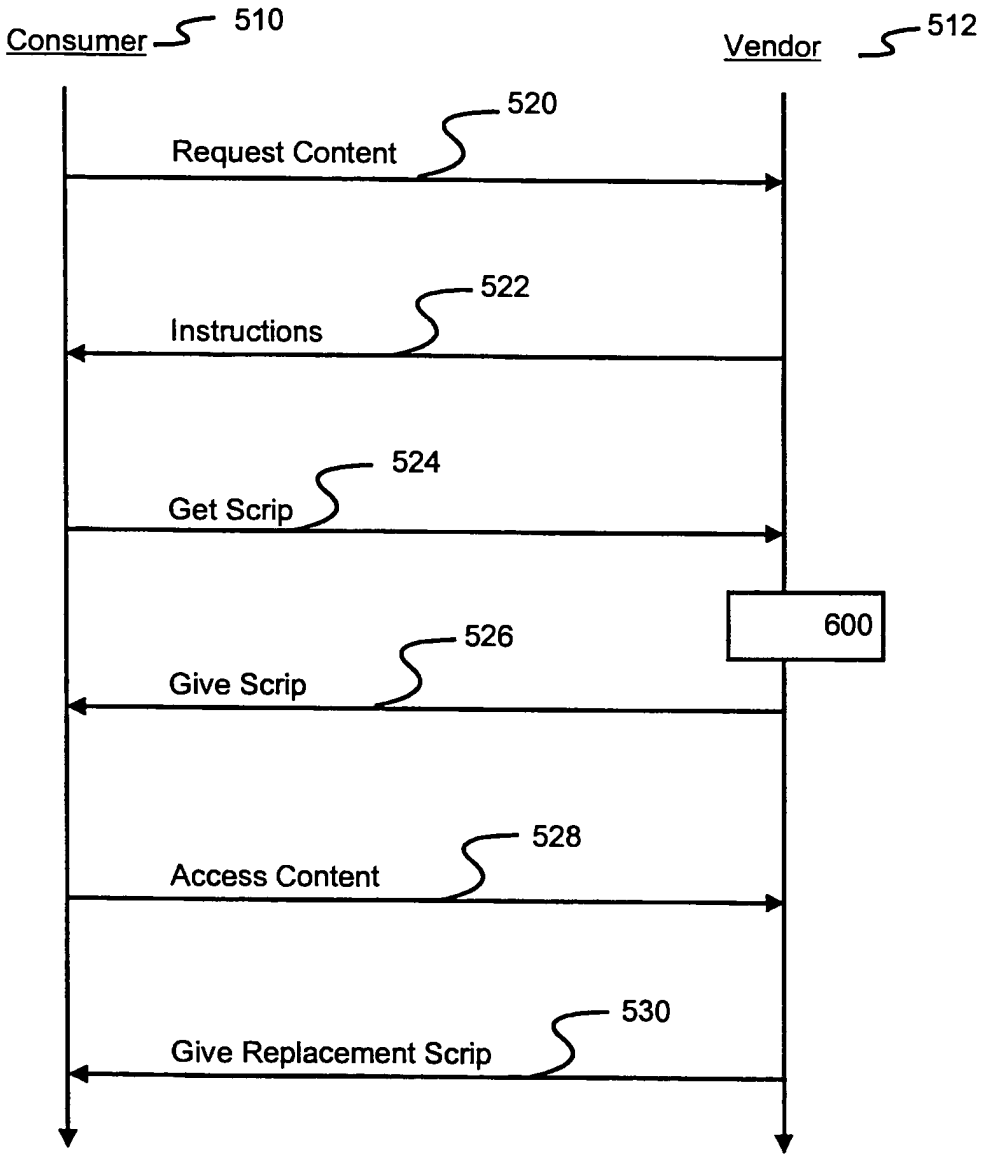
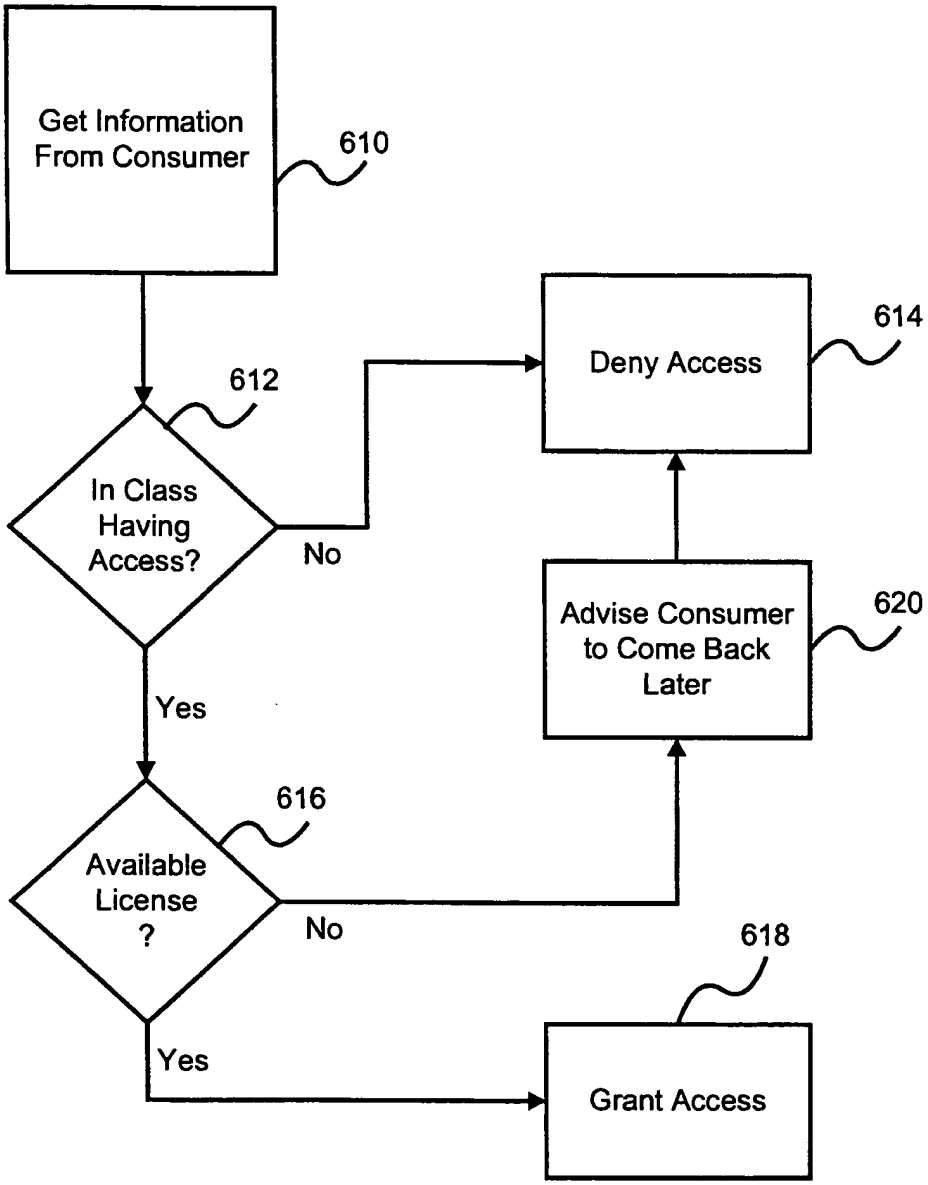


FIG. 5



INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/US 00/10213

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

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)
IPC 7 G06F G07F

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 practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 58306 A (OYLER SCOTT ;GUTHRIE JOHN (US); TECHWAVE INC (US); KRISHNAN GANAPA) 23 December 1998 (1998-12-23)	1,4-6,8, 10,11, 13,14, 16,17
Y		18
A		2,3,7,9, 12,15
	abstract page 6, line 4 -page 8, line 10 page 10, line 8 -page 16, line 17 page 28, line 7 -page 30, line 23 page 39, line 2 - line 11 figures 1-4	

—

-/—

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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
2 August 2000	09/08/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Jacobs, P

2

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern nal Application No
PCT/US 00/10213

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 97 03423 A (DIGITAL EQUIPMENT CORP) 30 January 1997 (1997-01-30)	1-4,8-10
Y		18
A		5,13,16, 17
	page 4, line 15 -page 11, line 33 figures 1-5	
X	US 5 905 860 A (BRINGHURST ADAM L ET AL) 18 May 1999 (1999-05-18)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract column 2, line 40 -column 16, line 3	
X	GB 2 316 503 A (ICL PERSONAL SYSTEMS OY) 25 February 1998 (1998-02-25)	1,4-6, 13,14, 16,17
A		2,3, 7-12,15, 18
	abstract page 6 -page 21 figures 1-4 claim 1	

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

...rmation on patent family members

Interna...al Application No
PCT/US 00/10213

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9858306 A	23-12-1998	US 6073124 A	06-06-2000
		AU 8150598 A	04-01-1999
WO 9703423 A	30-01-1997	US 5802497 A	01-09-1998
		BR 9606450 A	30-09-1997
		EP 0796480 A	24-09-1997
		IL 117195 A	20-06-1999
		JP 2984731 B	29-11-1999
		JP 9510814 T	28-10-1997
US 5905860 A	18-05-1999	US 5758069 A	26-05-1998
GB 2316503 A	25-02-1998	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 December 2000 (07.12.2000)

PCT

(10) International Publication Number
WO 00/73922 A2

(51) International Patent Classification[7]: G06F 17/00

(21) International Application Number: PCT/US00/11078

(22) International Filing Date: 25 April 2000 (25.04.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/323,635 1 June 1999 (01.06.1999) US

(71) Applicant: ENTERA, INC. [US/US]; 40971 Encyclopedia Circle, Fremont, CA 94538 (US).

(72) Inventor: SCHARBER, John, M.; 1616 Placer Circle, Livermore, CA 94550 (US).

(74) Agents: FAHMI, Tarek, N. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 00/73922 A2

(54) Title: CONTENT DELIVERY SYSTEM

(57) Abstract: Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables. Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

CONTENT DELIVERY SYSTEM
1

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the transmission and storage of digital information across a network. More particularly, the invention relates to an improved system and method for caching and/or delivering various types of digital content using a plurality of network protocols.

Description of the Related Art

The World Wide Web (hereinafter "the Web") is a network paradigm which links documents known as "Web pages" locally or remotely across multiple network nodes (i.e., Web servers). A single Web page may have links (a.k.a., "hyperlinks") which point to numerous other Web pages. When a user points and clicks on a link using a cursor control device such as a mouse, the user can jump from the initial page to another page, regardless of where the Web pages are actually located. For example, the initial Web page might be stored on a Web server in New York and the second page (accessed via the hyperlink in the first page) might be stored on a Web server in California.

The underlying principles of the Web were developed 1989 at the European Center for Nuclear Research (CERN) in Geneva. By 1994 there were approximately 500 Web servers on the Internet. Today there are more than a million, with new sites starting up at an extraordinary rate. In sum, the Web has become the center of Internet activity and is the primary reason for the explosive growth of the Internet over the past several years.

In addition to providing a simple point-and-click interface to vast amounts of information on the Internet, the Web is quickly turning into a content delivery system. Well known Internet browsers such as Netscape Navigator™ and Microsoft Internet Explorer™ frequently provide plug-in software which allow additional features to be incorporated into the browser program. These include, for example, support for audio and video streaming, telephony, and videoconferencing.

The unparalleled increase in Web usage combined with the incorporation of high bandwidth applications (i.e., audio and video) into browser programs has created serious

performance/bandwidth problems for most Internet Service Providers (hereinafter "ISPs"). Moreover, the network traffic resulting from non-Web-based Internet services such as Internet News (commonly known as "Usenet" News) has increased on the same scale as the increase in Web traffic, thereby further adding to the bandwidth problems experienced by most ISPs.

These issues will be described in more detail with respect to **Figure 1** which illustrates an ISP 100 with a link 160 to a larger network 150 (e.g., the Internet) through which a plurality of clients 130, 120 can access a plurality of Web servers 140-144 and/or News servers 146-148. Maintaining a link 160 to the Internet 150 with enough bandwidth to handle the continually increasing traffic requirements of its clients 120, 130 represents a significant cost for ISP. At the same time, ISP 110 must absorb this cost in order to provide an adequate user experience for its clients 120, 130.

One system which is currently implemented to reduce network traffic across link 160 is a proxy server 210 with a Web cache 220, illustrated in **Figure 2**. When client 120 initially clicks on a hyperlink and requests a Web page (shown as address "www.isp.com/page.html") stored on Web server 144, client 120 will use proxy server 210 as a "proxy agent." This means that proxy server 210 will make the request for the Web page on behalf of client 120 as shown. Once the page has been retrieved and forwarded to client 120, proxy server will store a copy of the Web page locally in Web cache 220. Thus, when client 120 or another client – e.g., client 130 – makes a subsequent request for the same Web page, proxy server 210 will immediately transfer the Web page from its Web cache 210 to client 130. As a result, the speed with which client 130 receives the requested page is substantially increased, and at the same time, no additional bandwidth is consumed across Internet link 160.

While the foregoing proxy server configuration alleviates some of the network traffic across Internet link 160, several problems remain. One problem is that prior Web cache configurations do not have sufficient intelligence to deal with certain types of Web pages (or other Web-based information). For example, numerous Web pages and associated content can only be viewed by a client who pays a subscriber fee. As such, only those clients which provide proper authentication should be permitted to download the information. Today, proxy servers such as proxy server 210 will simply not cache a Web document which requires authentication.

In addition, Web caches do not address the increasing bandwidth problem associated with non-Web based Internet information. In particular, little has been done to alleviate the increasing bandwidth problems created by Usenet news streams. In fact, ISPs today must set aside a substantial amount of bandwidth to provide a continual Usenet news feed to its clients. Moreover, no mechanism is currently available for caching other data transmissions such as the streaming of digital audio and video. The term "streaming" implies a one-way transmission from a server to a client which provides for uninterrupted sound or video. When receiving a streaming transmission, the client will buffer a few seconds of audio or video information before it starts sending the information to a pair of speakers and/or a monitor, thus compensating for momentary delays in packet delivery across the network.

Accordingly, what is needed is a content delivery system which will reduce the bandwidth requirements for ISP 110 while still providing clients 120, 130 with an adequate user experience. What is also needed is a system which will work seamlessly with different types of Web-based and non-Web-based information and which can be implemented on currently available hardware and software platforms. What is also needed is an intelligent content delivery system which is capable of caching all types of Web-based information, including information which requires the authentication of a client before it can be accessed. What is also needed is a content delivery system which is easily adaptable so that it can be easily reconfigured to handle the caching of new Internet information and protocols. Finally, what is needed is a data replication system which runs on a distributed database engine, thereby incorporating well known distributed database procedures for maintaining cache coherency.

SUMMARY OF THE INVENTION

Disclosed is a network content delivery system configured to: select a first content routing technique for processing a first set of network content; and select a second content routing technique for processing a second set of network content, wherein the first and second content routing techniques are selected based on one or more content routing variables.

Also disclosed is a content delivery system comprising: a network node for storing network content; a first transmission medium communicatively coupled to the network node for transmitting a first set of network content to the network node; and a second transmission medium communicatively coupled to the network node for transmitting a second set of

network content to the network node, wherein the first and second sets of network content are selected based on one or more routing variables.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which:

FIG. 1 illustrates generally a network over which an ISP and a plurality of servers communicate.

FIG. 2 illustrates an ISP implementing a proxy server Web cache.

FIG. 3 illustrates one embodiment of the underlying architecture of an Internet content delivery system node.

FIG. 4 illustrates a plurality of Internet content delivery system nodes communicating across a network.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

One embodiment of the present system is a computer comprising a processor and a memory with which software implementing the functionality of the internet content delivery system described herein is executed. Such a computer system stores and communicates (internally or with other computer systems over a network) code and data using machine readable media, such as magnetic disks, random access memory, read only memory, carrier waves, signals, etc. In addition, while one embodiment is described in which the parts of the present invention are implemented in software, alternative embodiments can implement one or more of these parts using any combination of software, firmware and/or hardware.

The underlying architecture of one embodiment of the present internet content delivery system (hereinafter "ICDS") is illustrated in **Figure 3**. A single ICDS node 300 is shown including a cache 330, an ICDS application programming interface (hereinafter "API") 360 which includes a distributed database engine 361, and a plurality of software modules 310-326

which interface with the ICDS API 360. ICDS node 300 may communicate over a network 340 (e.g., the Internet) over communication link 370 and may also interface with a plurality of clients 350-351 and/or other ICDS nodes (e.g., through link 380).

As is known in the art, an API such as ICDS API 360 is comprised of a plurality of subroutines which can be invoked by application software (i.e., software written to operate in conjunction with the particular API). Thus, in **Figure 3** each of the plurality of software modules 310-326 may be uniquely tailored to meet the specific needs of a particular ISP. The modules interface with API 360 by making calls to the API's set of predefined subroutines. Another significant feature of ICDS API 360 is that it is platform-independent. Accordingly, it can be implemented on numerous hardware platforms including those that are Intel-based, Macintosh-based and Sun Microsystems-based.

In one embodiment, a portion of API 360's subroutines and a set of prefabricated modules can be marketed together as a Software Development Kit (hereinafter "SDK"). This will allow ISPs, corporations and/or end-users to customize the type of internet content delivery/caching which they require. In addition, because modules 310-326 may be dynamically linked, they may be loaded and unloaded without having to reboot the hardware platform on which cache 330 is executed.

I. Distributed Content Processing

As illustrated, ICDS node 300 includes a plurality of network protocol modules 310-319 which interface with API 360. These modules provide caching support on ICDS node 300 for numerous different Internet protocols including, but not limited to, Web protocols such as the Hypertext Transfer Protocol (hereinafter "HTTP") 310, Usenet news protocols such as the Network News Transport Protocol (hereinafter "NNRP") 312, directory access protocols such as the Lightweight Directory Access Protocol (hereinafter "LDAP") 314, data streaming protocols such as the Real Time Streaming Protocol (hereinafter "RTSP") 316, and protocols used to perform Wide Area Load Balancing (hereinafter "WALB") 318. Because the underlying architecture of the present ICDS system includes an open API, new protocol modules (e.g., module 319) can be seamlessly added to the system as needed.

One embodiment of the ICDS system includes a plurality of standardized service definitions through which individual service modules 320-326 may be configured to interface

with the ICDS API 360. These service modules provide the underlying functionality of ICDS node 300 and may include a data services module 320, an access services module 322, a transaction services module 323, a commercial services module 324, a directory services module 325, and a resource services module 326. The functionality of each of these modules will be described in more detail below.

In one embodiment of the ICDS system, the ICDS API includes a distributed relational database engine 361. As a result, a plurality of ICDS nodes 410-440 can be distributed across ISP 400's internal network and still maintain a coherent, up-to-date storage of Internet content. For example, if a particular data object is updated at two nodes simultaneously, the underlying distributed database system may be configured to resolve any conflicts between the two modifications using a predefined set of distributed database algorithms. Accordingly, the present system provides built in caching support for dynamically changing Internet content (e.g., Web pages which are modified on a regular basis). Such a result was not attainable with the same level of efficiency in prior art caching systems such as proxy server 210 of **Figure 2** (which are executed on, e.g., standard flat file systems such as UNIX or NFS file servers).

Data Services

Data services modules such as module 320 running on each ICDS node 410-440 provide support for data replication and distribution across ISP 400's internal network 480. This includes caching support for any data protocol included in the set of protocol modules 310-319 shown in **Figure 3** as well as for any future protocol which may be added as a module to the ICDS API 360. Because the ICDS API 360 provides a set of standardized service definitions for data services module 320, an ISP using a plurality of ICDS nodes 300 as illustrated in **Figure 4** can replicate data across its network without an extensive knowledge of distributed database technology. In other words, the ISP can configure its plurality of nodes by invoking the standardized service definitions associated with data services module 320 and leave the distributed database functionality to the distributed database engine 361.

Generally, three different types of data replication may be implemented by the present system: dynamic replication, database replication (or "actual" replication), and index replication. Using dynamic replication, if client 472 requests content from internal ICDS server 460 or from a server across network 490, the content will be delivered to client 472 and replicated in ICDS node 430. If client 473 (or any other client) subsequently requests the

same content, it will be transmitted directly from ICDS node 430 rather than from its original source (i.e., a second request to server 460 or a server across network 490 will not be required). Accordingly, bandwidth across ISP 400's internal network and across Internet link 405 is conserved.

The dynamic replication mechanism just described works well for replicating static content but not for replicating dynamically changing content. For example, if the replicated content is a magazine article then caching a copy locally works well because it is static information – i.e., there is no chance that the local copy will become stale (out of date). However, if the replicated content is a Web page which contains continually changing information such as a page containing stock market quotes, then dynamic replication may not be appropriate. No built in mechanism is available for proxy cache server 210 to store an up-to-date copy of the information locally.

The present ICDS system, however, may use database replication to maintain up-to-date content at each ICDS node 410-440. Because the present system includes a distributed database engine 361, when a particular piece of content is changed at one node (e.g., ICDS server 460) a store procedure may be defined to update all copies of the information across the network. This may be in the form of a relational database query. Thus, the present system may be configured to use dynamic replication for static content but to use database replication for time-sensitive, dynamically changing content.

The third type of database replication is known as index replication. Using index replication a master index of content is replicated at one or more ICDS nodes 410-440 across the network 480. Once again, this implementation is simplified by the fact that the underlying ICDS node engine is a distributed database engine. Certain types of information distributions are particularly suitable for using index replication. For example, news overview information (i.e., the list of news articles in a particular newsgroup) is particularly suited to index replication. Instead of replicating each individual article, only the news overview information needs to be replicated at various nodes 410-440 across the network 480. When a client 473 wants to view a particular article, only then will the article be retrieved and cached locally (e.g., on ICDS node 430).

ICDS node 430 is capable of caching and delivering various types of Internet data using any of the foregoing replication techniques. While prior art proxy servers such as proxy server 210 may only be used for caching Web pages, ICDS node 430 is capable of caching various other types of internet content (e.g., news content) as a result of the protocol modules 310-319 interfacing with ICDS API 360. Moreover, as stated above, ICDS node 430 (in conjunction with nodes 410, 420 and 440) may be configured to cache dynamic as well as static Web-based content using various distributed database algorithms.

One specific example of a data service provided by one embodiment of the present system is Wide Area Load Balancing (hereinafter "WALB") using layer 7 switching as described in the co-pending U.S. Patent Application entitled "WIDE AREA LOAD BALANCING" (Serial No. _____), which is assigned to the assignee of the present application and which is incorporated herein by reference. The present system may also perform dynamic protocol selection, dynamic query resolution, and/or heuristic adaptation for replicating content across a network as set forth in the co-pending U.S. Patent Application entitled Dynamic Protocol Selection and "QUERY RESOLUTION FOR CACHE SERVERS" (Serial No. ____), which is assigned to the assignee of the present application and which is incorporated herein by reference. Finally, the present system also may include network news (e.g., Usenet news) services set forth in the co-pending U.S. Patent Applications entitled "HYBRID NEWS SERVER" (Serial No. ___), and "SELF-MODERATED VIRTUAL COMMUNITIES" (Serial No. ___), each assigned to the assignee of the present application and each incorporated herein by reference.

Access Services

As stated above, prior art proxy server cache systems such as proxy server 210 are only capable of caching static, publicly available Web pages. A substantial amount of Web-based and non-Web-based content, however, requires some level of authentication before a user will be permitted to download it. Thus, client 472 (in Figure 4) may pay a service fee to obtain access to content on a particular web site (e.g., from server 460 or from another server over network 490). As a result, when he attempts to access content on the site he will initially be prompted to enter a user name and password. Once the user transmits this information to the Web server, he will then be permitted to download Web server content as per his service agreement.

A problem that arises, however, is that prior art cache systems such as proxy server 210 are not permitted to cache the requested content. This is because proxy server 210 has no way of authenticating subsequent users who may attempt to download the content. Thus, documents which require authentication are simply uncacheable using current network cache systems.

The present ICDS system, however, includes user authentication support embedded in access services module 322. Thus, when client 473, for example, attempts to access a Web page or other information which requires authentication, ICDS node will determine whether the requested content is stored locally. If it is, then ICDS node 430 may communicate with the authentication server (e.g., server 460 or any server that is capable of authenticating client 473's request) to determine whether client 473 should be granted access to the content. This may be accomplished using standardized authentication service definitions embedded in access services module 322. Using these definitions, ICDS node 430 will not only know what authentication server to use, it will also know what authentication *protocol* to use when it communicates to the authentication server. As a result of providing local access services module 322 for authentication, network information which requires authentication can now be cached locally in ICDS node 430, thereby conserving additional bandwidth across network link 405 and/or ISP network 480.

One particular embodiment of the present system replicates Remote Authentication Dial In User Service (hereinafter "RADIUS") information across network 480. RADIUS is an application-level protocol used by numerous ISP's to provide user authentication and profile services. This is achieved by setting up a central RADIUS server with a database of users, which provides both authentication services (i.e., verification of user name and password) and profile services detailing the type of service provided to the user (for example, SLIP, PPP, telnet, rlogin).

Users connect to one or more Network Access Servers (hereinafter "NASs") which operate as a RADIUS clients and communicate with the central RADIUS server. The NAS client passes the necessary user information to the central RADIUS server, and then acts on the response which is returned. RADIUS servers receive user connection requests, authenticate users, and then return all configuration information necessary for the client to deliver service to the user.

One problem associated with the RADIUS protocol is that it does not provide any built in facilities for replication of RADIUS information. Accordingly, on large ISP's such as America Online ("AOL"), which may have tens of millions of users, RADIUS servers are hard hit, potentially handling thousands of logon requests a minute. This may create severe performance/bandwidth problems during high traffic periods. In response, some ISP's have taken a brute-force approach to distributing RADIUS information by simply copying the information to additional servers across the network without any built in mechanism to keep the RADIUS data coherent and up-to-date.

One embodiment of the present ICDS system provides an efficient, dynamic mechanism for distributing RADIUS information. Specifically, a RADIUS module is configured to interface with ICDS API 360 in this embodiment (similar to the way in which protocol modules 310-319 interface with the ICDS API 360). RADIUS information can them be seamlessly distributed across the system using distributed database engine 361. For example, the RADIUS module in conjunction with access services module 322 on ICDS node 430 may maintain radius information for local users. [Exactly how will this work? I assume that access services module will be used but there will be a separate RADIUS protocol module to support the protocol??] Thus, when client 472 first logs in to the system, ICDS node 430 may communicate with a second ICDS node (e.g., central ICDS server 460) which contains the necessary RADIUS authentication and user profile information. Client 472 will input a user name and password and will then be permitted access to the network as per his service agreement with ISP 400.

Unlike previous RADIUS systems, however, ICDS node 430 in the present embodiment may locally cache client 472's RADIUS information so that the next time client 472 attempts to login to the network, the information will be readily available (i.e., no access to a second ICDS node will be necessary). ICDS node 430 may be configured to save client 472's RADIUS information locally for a predetermined period of time. For example, the information may be deleted if client 472 has not logged in to local ICDS node 430 for over a month.

Thus, if client 472 represents a user who frequently travels across the country and logs in to ISP 400's network 480 from various different ICDS nodes, the present system provides a quick, effective mechanism for dynamically replicating client 472's user information into

those geographical locations from which he most commonly accesses ISP 400. This reduces the load which would otherwise be borne by a central RADIUS server and also improves client 472's user experience significantly (i.e., by providing him with a quick login).

Database replication can also be used to update RADIUS information distributed across multiple ICDS nodes 410-440. This may be done using known store procedures defined in relational database 361. For example, if client 472 cancels his service agreement with ISP 400, he should not be able to continually log in to local ICDS node 430 using the RADIUS information which has been cached locally. Thus, under the present ICDS system, ISP 400 may simply issue a relational database query such as [let's add another update query here using database terminology as an example] to immediately update ICDS node 430's radius information.

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout the foregoing description, specific embodiments of the ICDS system were described using the RADIUS protocol in order to provide a thorough understanding of the operation of the ICDS system. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. For example, the ICDS system may also distribute authentication and user profile information in the form of the Lightweight Directory Access Protocol ("LDAP"). In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention.

Access services module 322 may also provide local encryption/decryption and watermarking of internet content. Audio or video content delivery systems, for example, commonly use encryption of content to protect the rights of the underlying copyright holder. When a user requests a particular piece of content some delivery systems encrypt the content using a unique client encryption key. Only a client who possesses the encryption key (presumably the client who paid for the content) will be permitted to play the content back. Other systems provide for the "watermarking" of content (rather than encrypting) so that the rightful owner of the content may be identified. This simply entails embedding a unique "tag"

which identifies the source of the content and/or the owner of the content (i.e., the one who paid for it).

Prior art caching systems such as proxy server 210 are not capable of dealing with encrypted or watermarked content because the encryption/watermarking functionality was not provided locally (i.e., proxy server 210 was not "smart" enough). In one embodiment of the present ICDS system, however, access services module 322 of ICDS nodes 410-440 includes a local encryption module and/or a local watermarking module. For example, if client 473 requests specific content such as a copyrighted music content stored on a music server (e.g., ICDS server 460), the initial request for the content will be made from ICDS node 430 on behalf of client 473. ICDS node 430 will retrieve the requested content and cache it locally. If the requested content requires encryption, ICDS node 430 will use its local encryption module to encrypt the requested content using a unique user encryption key for client 473.

If a second client – e.g., client 472 – requests the same content, the copy stored locally on ICDS module 430 can be used. ICDS module 430 will simply encrypt the content using a *different* encryption key for user 472. Thus, frequently requested multimedia content (which, as is known in the art, can occupy a substantial amount of storage space) may be cached locally at ICDS node 430 notwithstanding the fact that the content requires both user authentication and encryption.

The same functionality may be provided for watermarking of content. ICDS node will use a watermarking module (which may comprise a component of access services module 322) to individually watermark multimedia information requested by individual clients, thereby protecting the rights of the copyright holder of the underlying multimedia content. This information can then be regularly communicated back to a central database repository.

As is known in the art, multimedia files can be extremely large and, accordingly, take substantially more time to communicate across a network than do, for example, generic Web pages. As such, the ability to locally cache multimedia files significantly reduces traffic across network 480, and also significantly improves the user experience for local users when downloading multimedia information.

Transaction Services

In addition to replicating data services and access services information across a network, the present ICDS system also provides for the replication of transaction services. Transaction services includes maintaining information on client payments for use of ISP 400's services as well as information relating to the client's online access profile (i.e., recording of the times when the user is online).

When a client logs in to an ISP today, the client's online information is maintained on a single central server. The central server maintains records of when and for how long the client logged in to the network and may also include information about what the client did while he was online. As was the case with maintaining a central RADIUS server, maintaining a central transaction server for all users of a large ISP is inefficient and cumbersome. The present system solves the performance and bandwidth problems associated with such a configuration by storing transaction information locally via transaction module 323 and algorithms build around distributed database engine 361.

Thus, if client 472 only logs on to ISP 400's network 480 via ICDS node 430, all of his transaction and billing information will be stored locally. The information may then be communicated across network 480 to a central billing server at predetermined periods of time (e.g., once a month). [We didn't go into great detail on transaction services and the rest; please add information as you feel appropriate]

Commercial Services

Commercial services module 324 provides a significantly improved local caching capability for add rotation and accounting. An add rotation system operating in conjunction with a typical proxy cache server will now be described with respect to **Figure 2.** When client 120 downloads a web page from Web server 142 the Web page may contain an ad tag or an add tag may automatically be inserted. The add tag will identify add server 170 and will indicate that an add should be inserted into the requested Web page from add server 170. Add server will then identify a specific add to insert into the downloaded Web page from add content server. The Web page plus the inserted add will then be forwarded to proxy server 210 and on to client 120.

Add server 170 will keep an accounting of how many different users have downloaded Web pages with adds inserted as described above. However, one problem with accounting on this system is that proxy server 210 requests Web pages *on behalf of* its clients. Accordingly, once the requested Web pages has been cached locally in Web cache 220, add server will only receive requests from proxy server 210 for any subsequent requests for the Web page. This will result in an inaccurate accounting of how many unique clients actually requested the Web page (and how many adds were viewed by unique users).

Because one embodiment of the present system provides built in caching support for ad rotation services, an accurate accounting of the number of hits that a particular ad receives may be maintained. More specifically, one embodiment of the present ICDS system solves this problem by providing a commercial services module that monitors and records how often individual clients request Web pages containing particular adds from add content server 171. This information than then be communicated to a central server (e.g., ICDS server 460) at predetermined intervals for generating add rotation usage reports.

Directory Services

Directory services provide the ability to cache locally a directory of information across network 480 or 490. That is, the question here is not whether the particular information is available but where exactly over networks 480 or 490 it is located. It should be noted that there may be some overlap between the directory services concept and the index replication concept described above with respect to data services. [I'm still not 100% sure what this is – please elaborate with an example]

II. Content Routing

The term "content routing" refers to the ability to select among various techniques/protocols for maintaining a coherent set of content across a network. The selection of a particular technique may be based on several routing variables including, but not limited to, the type of content involved (i.e., FTP, HTTP . . . etc), the size of the content involved (i.e., small files such as HTTP vs. large files such as audio/video streaming), the location of the content on network 480 and/or network 490, the importance of a particular piece of content (i.e., how important it is that the content be kept up-to-date across the entire network), the particular user requesting the content and the terms of his subscription agreement (i.e., some users may be willing to pay more to be insured that they receive only the most up-to-date

content without having to wait), the frequency with which the content is accessed (e.g., 5%-10% of content on the Internet represents 90% of all the *requested* content), and the underlying costs and bandwidth constraints associated with maintaining up-to-date, coherent content across a particular network (e.g., network 480).

Three content routing techniques which may be selected (based on one or more of the foregoing variables) to maintain coherent content across the plurality of nodes illustrated in **Figure 4** are content revalidation, content notification, and content synchronization.

Content Revalidation

When content validation is selected, the original content source will be checked only when the content is requested locally. For example, client 473 may request an installation program for a new Web browser (e.g., the latest version of Microsoft's™ Internet Explorer™). The file may then be transmitted form ICDS server 460 to client 473 and a copy of the file cached locally on ICDS node 430. Consequently, if client 472 requests the same program, for example, two weeks later, ICDS node 430 may be configured to check ICDS server 460 to ensure that it contains the most recent copy of the file before passing it on to client 472 (i.e., ICDS node 430 "revalidates" the copy it has locally).

ICDS node 430 may also be configured to revalidate a piece of content only if has been stored locally for a predetermined amount of time (e.g., 1 week). The particular length of time selected may be based on one or more of the variables discussed above. Moreover, in one embodiment, the age/revision of a particular piece of content is determined based on tags (e.g., HTML metatags) inserted in the particular content/file.

Revalidation may work more efficiently with certain types of content than with others. For example, revalidation may be an appropriate mechanism for maintaining up-to-date copies of larger files which do not change very frequently (i.e., such as the program installation files described above). However, revalidation may not work as efficiently for caching smaller and/or continually changing files (e.g., small HTML files) because the step of revalidating may be just as time consuming as making a direct request to ICDS server 460 for the file itself. If the file in question is relatively small and/or is changing on a minute-by-minute basis (e.g., an HTML file containing stock quotes) then one or more other content routing techniques may be more appropriate.

Of course, other routing variables may influence the decision on which technique to use, including the issue of how strong the data transmission connection is between ICDS node 430 and ICDS server 460 (i.e., how reliable it is, how much bandwidth is available . . . etc) and the necessity that the underlying information cached locally (at ICDS node 430) be accurate. The important thing to remember is that ICDS node 430 – because of its underlying open API architecture – may be configured based on the unique preferences of a particular client.

Content Notification

Content notification is a mechanism wherein the central repository for a particular piece of content maintains a list of nodes, or "subscribers," which cache a copy of the content locally. For example, in **Figure 4**, a plurality of agents may run on ICDS server 460 which maintain a list of content subscribers (e.g., ICDS node 430, ICDS node 420 . . . etc) for specific types of content (e.g., HTML, data streaming files, FTP files . . . etc). In one embodiment of the system, a different agent may be executed for each protocol supported by ICDS server 460 and/or ICDS nodes 410-440.

When a particular piece of content is modified on ICDS server 460, a notification of the modification may be sent to all subscriber nodes (i.e., nodes which subscribe to that particular content). Upon receiving the notification, the subscriber node – e.g., ICDS node 430 – may then invalidate the copy of the content which it is storing locally. Accordingly, the next time the content is requested by a client (e.g., client 472), ICDS node 430 will retrieve the up-to-date copy of the content from ICDS server 460. The new copy will then be maintained locally on ICDS node 430 until ICDS node 430 receives a second notification from an agent running on ICDS server 460 indicating that a new copy exists.

Alternatively, each time content is modified on ICDS server 460 the modified content may be sent to all subscriber nodes along with the notification. In this manner a local, up-to-date copy of the content is always ensured. In one embodiment of the system, notification and/or transmittal of the updated content by the various system agents is done after a predetermined period of time has elapsed (e.g., update twice a day). The time period may be selected based on the importance of having an up-to-date copy across all nodes on the network 480, 490.

As was the case with content revalidation, the different varieties of content notification may work more efficiently in some situations than in others. Accordingly, content notification may be selected as a protocol (or not selected) based on one or more of the routing variables recited at the beginning of this section (i.e., the "content routing" section). For example, content notification may be an appropriate technique for content which is frequently requested at the various nodes across networks 480 and 490 (e.g., for the 5-10% of the content which is requested 90% of the time), but may be a less practical technique for larger amount of content which is requested infrequently. As another example, large files which change frequently may not be well suited for content notification (i.e., particularly the type of content notification where the actual file is sent to all subscribers along with the notification) due to bandwidth constraints across networks 480 and/or 490 (i.e., the continuous transmission of large, frequently changing files may create too much additional network traffic).

Content Synchronization

Content synchronization is a technique for maintaining an exact copy of a particular type of content on all nodes on which it is stored. Using content synchronization, as soon as a particular piece of content is modified at, for example, ICDS node 430, it will immediately be updated at all other nodes across networks 480 and/or 490. If the same piece of data was concurrently modified at one of its other storage locations (e.g., ICDS node 410) then the changes may be backed off in order to maintain data coherency. Alternatively, an attempt may be made to reconcile the two separate modifications if it is possible to do so (using, e.g., various data coherency techniques).

Once again, as with content notification and content revalidation, content synchronization is more suitable for some situations than it is for others. For example, content synchronization is particularly useful for information which can be modified from several different network nodes (by contrast, the typical content notification paradigm assumes that the content is modified at one central node). Moreover, content synchronization may be useful for maintaining content across a network which it is particularly important to keep current. For example, if network 480 is an automatic teller machine (hereinafter "ATM") network, then when a user withdraws cash from a first node (e.g., ICDS node 440), his account will be instantly updated on all nodes (e.g., ICDS node 410, 420, 460, and 430) to reflect the withdrawal. Accordingly, the user would not be able to go to a different node in a different part of the country and withdraw more than what he actually has in his account.

As another example, a user's account status on a network (i.e., whether he is a current subscriber and/or what his network privileges are) may be maintained using content synchronization. If, for example, a user of network 480 were arrested for breaking the law over network 480 (e.g., distributing child pornography), it would be important to disable his user account on all network nodes on which this information might be cached. Accordingly, using content synchronization, once his account was disabled at one node on network 480 this change would automatically be reflected across all nodes on the network.

As previously stated, the choice of which content routing technique to use for a particular type of content may be based on any of the variables set forth above. In one embodiment, the frequency with which content is accessed across the networks 480 and/or 490 may be an important factor in deciding which protocol to use. For example, the top 1% accessed content may be selected for content synchronization, the top 2%-10% accessed content may be selected for content notification, and the remaining content across networks 480, 490 may be selected for content revaildation.

III. Content Delivery Medium Selection

In addition to the content routing flexibility provided by the content delivery system as set forth above, one embodiment of the system allows content delivery nodes such as ICDS node 430 to select from a plurality of different transmission media. For example, ICDS node 430 may receive content from ICDS server 460 via a plurality of communication media, including, but not limited to, satellite transmission, wireless RF transmission, and terrestrial transmission (e.g., fiber).

Moreover, as with the selection of a particular content routing technique, the selection of a particular transmission medium may be based on any of the variables set forth above (see, e.g., routing variables listed under counter routing heading; page 24, line 18 through page 25, line 9). Moreover, the choice of a particular transmission medium may be dynamically adjustable based on performance of that medium. For example, ICDS node 430 may be configured to receive all of its content over terrestrial network 480 as long as network 480 is transmitting content at or above a threshold bandwidth. When transmissions over network 480 dip below the threshold bandwidth, ICDS node 480 may then begin receiving certain content via satellite broadcast or wireless communication.

In addition, a transmission medium may be selected for transmitting specific content based on how frequently that content is accessed. For example, the top 10% frequently accessed content may be continually pushed out to ICDS node 430 via satellite broadcast while the remaining content may be retrieved by (i.e., "pulled" to) ICDS node 430 over network 480 upon request by clients (e.g., client 473). Accordingly, those employing ICDS nodes such as node 430 can run a cost-benefit analysis to determine the most cost effective way to implement their system by taking in to consideration, for example, the needs of their users, the importance of the content involved and the expense of maintaining multiple transmission connections into ICDS node 430 (e.g., the cost associated with maintaining an ongoing satellite connection).

In one embodiment of the system, tags (e.g., HTML metatags) may be inserted into particular types of content to identify a specific transmission path/medium for delivering that content to ICDS node 480. The tags in this embodiment may identify to various nodes (and/or routers) across networks 480 and/or 490 how the particular content should be routed across the networks (e.g., from node 410 to node 420 via terrestrial network 480; from node 420 to node 430 via wireless transmission).

One of ordinary skill in the art will readily recognize from the preceding discussion that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the invention. Throughout this detailed description, numerous specific details are set forth such as specific network protocols (i.e., RADIUS) and networks (i.e., the Internet) in order to provide a thorough understanding of the present invention. It will be appreciated by one having ordinary skill in the art, however, that the present invention may be practiced without such specific details. In other instances, well known software and hardware configurations/techniques have not been described in detail in order to avoid obscuring the subject matter of the present invention. The invention should, therefore, be measured in terms of the claims which follow.

CLAIMS

What is claimed is:

1. A network content delivery system configured to:

select a first content routing technique for processing a first set of network content; and

select a second content routing technique for processing a second set of network content, wherein said first and second content routing techniques are selected based on one or more content routing variables.

2. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content revalidation technique.

3. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content notification technique.

4. The network content delivery system as claimed in Claim 1 wherein one of said selected content routing techniques is a content synchronization technique.

5. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is accessed by users.

6. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the size of said network content.

7. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the frequency with which said network content is modified.

8. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is the type of network content (e.g., HTML, Usenet News).

9. The network content delivery system as claimed in Claim 1 wherein one of said content routing variables is identity of the user requesting said network content.

10. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the size of said network content.

11. The network content delivery system as claimed in Claim 2 wherein said content revalidation technique is selected based on the frequency with which said network content is accessed.

12. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the size of said network content.

13. The network content delivery system as claimed in Claim 3 wherein said content notification technique is selected based on the frequency with which said network content is accessed.

14. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the size of said network content.

15. The network content delivery system as claimed in Claim 4 wherein said content synchronization technique is selected based on the frequency with which said network content is accessed.

16. The network content delivery system as claimed in Claim 1 including the additional step of selecting a first transmission medium for a first group of network content based on one or more of said content routing variables.

17. The network content delivery system as claimed in Claim 16 including the additional step of selecting a second transmission medium for a second group of network content based on one or more of said content routing variables.

18. The network content delivery system as claimed in Claim 1 including an application programming interface for interfacing with a plurality of network protocol and service modules.

19. A content delivery system comprising:

a network node for storing network content;

a first transmission medium communicatively coupled to said network node for transmitting a first set of network content to said network node; and

a second transmission medium communicatively coupled to said network node for transmitting a second set of network content to said network node,

wherein said first and second sets of network content are selected based on one or more routing variables.

20. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a satellite transmission.

21. The content delivery system as claimed in Claim 19 wherein said first transmission medium is a wireless radio frequency transmission.

22. The content delivery system as claimed in Claim 19 wherein said first transmission medium is terrestrial-based transmission.

23. The content delivery system as claimed in Claim 19 wherein said network node monitors transmission bandwidth of said first transmission medium and reallocates content

from said first set to said second set if said first transmission medium drops below a predetermined threshold value.

24. The content delivery system as claimed in Claim 23 wherein said first transmission medium is terrestrial and said second transmission medium is non-terrestrial.

25. The content delivery system as claimed in Claim 19 wherein content is included in said first set based on the frequency with which said content is accessed.

26. The network content delivery system as claimed in Claim 19 including an application programming interface for interfacing with a plurality of network protocol and service modules.

27. An article of manufacture including a sequence of instructions stored on a computer-readable media which, when executed by a network node, cause the network node to perform the acts of:

establishing a plurality of groups of network content to be cached on said network node based on one or more content routing variables;

selecting a first content routing technique for maintaining data coherency in a first group of said plurality; and

selecting a second content routing technique for maintaining data coherency in a second group of said plurality.

28. The article of manufacture as claimed in claim 27 wherein said first content routing technique is content revalidation.

29. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content notification.

30. The article of manufacture as claimed in Claim 28 wherein said second content routing technique is content synchronization.

31. The article of manufacture as claimed in Claim 28 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

32. The article of manufacture as claimed in Claim 29 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

33. The article of manufacture as claimed in Claim 30 wherein said content routing variable used to select said content for said first group is the frequency with which said content is accessed.

34. A network node comprising:

an application programming interface ("API"), said API including a distributed relational database engine;

a plurality of protocol modules for interfacing with said API, said protocol modules configured to allow said system to communicate over a network using a plurality of network protocols;

a cache memory for caching data communicated to said cache memory using said plurality of protocol modules; and

a data services module for maintaining coherency between said data stored in said cache memory and data stored at other nodes across said network.

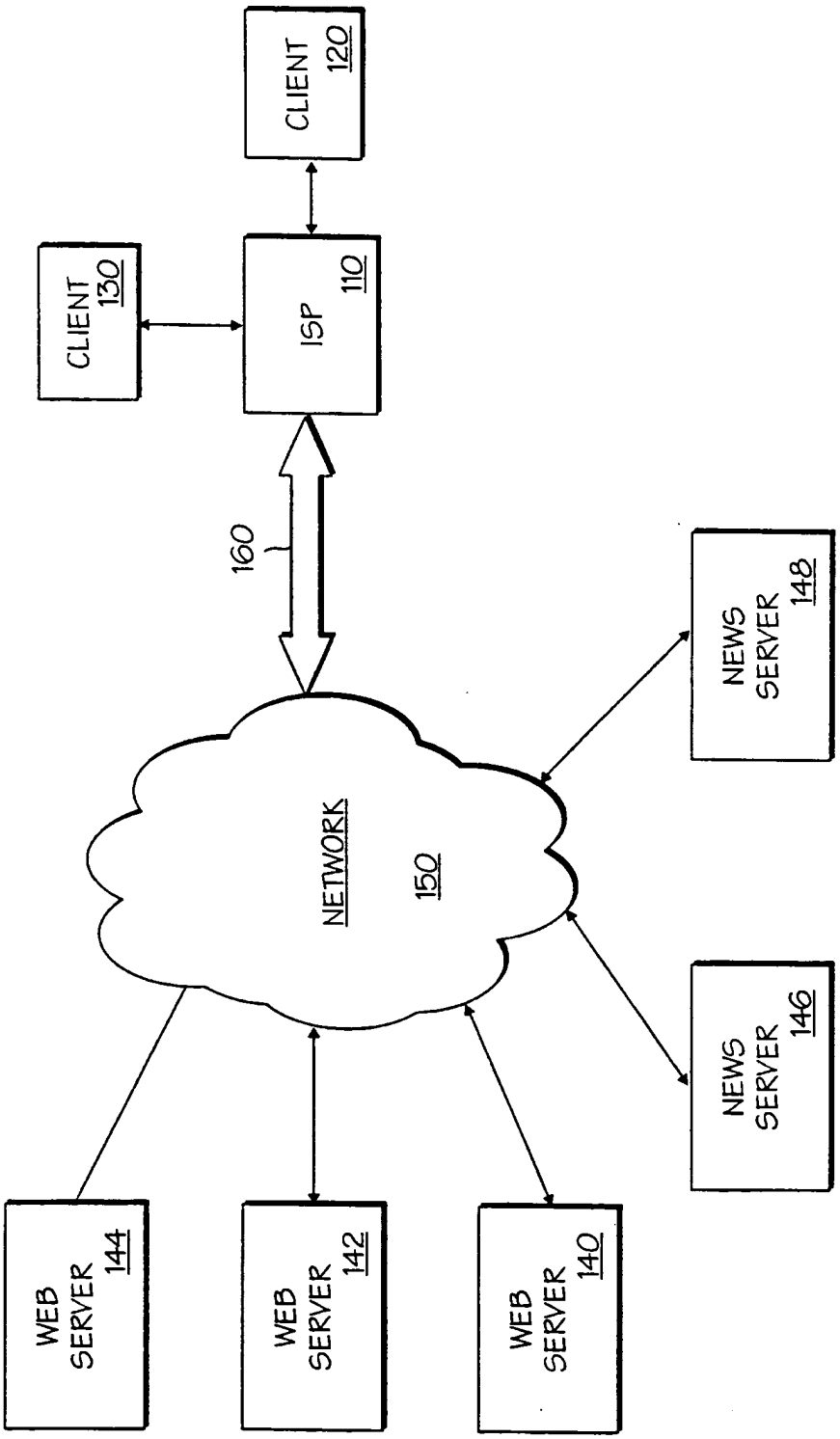

CLIENT 120
CLIENT 130
ISP 110
160
NETWORK 150
NEWS SERVER 148
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 1

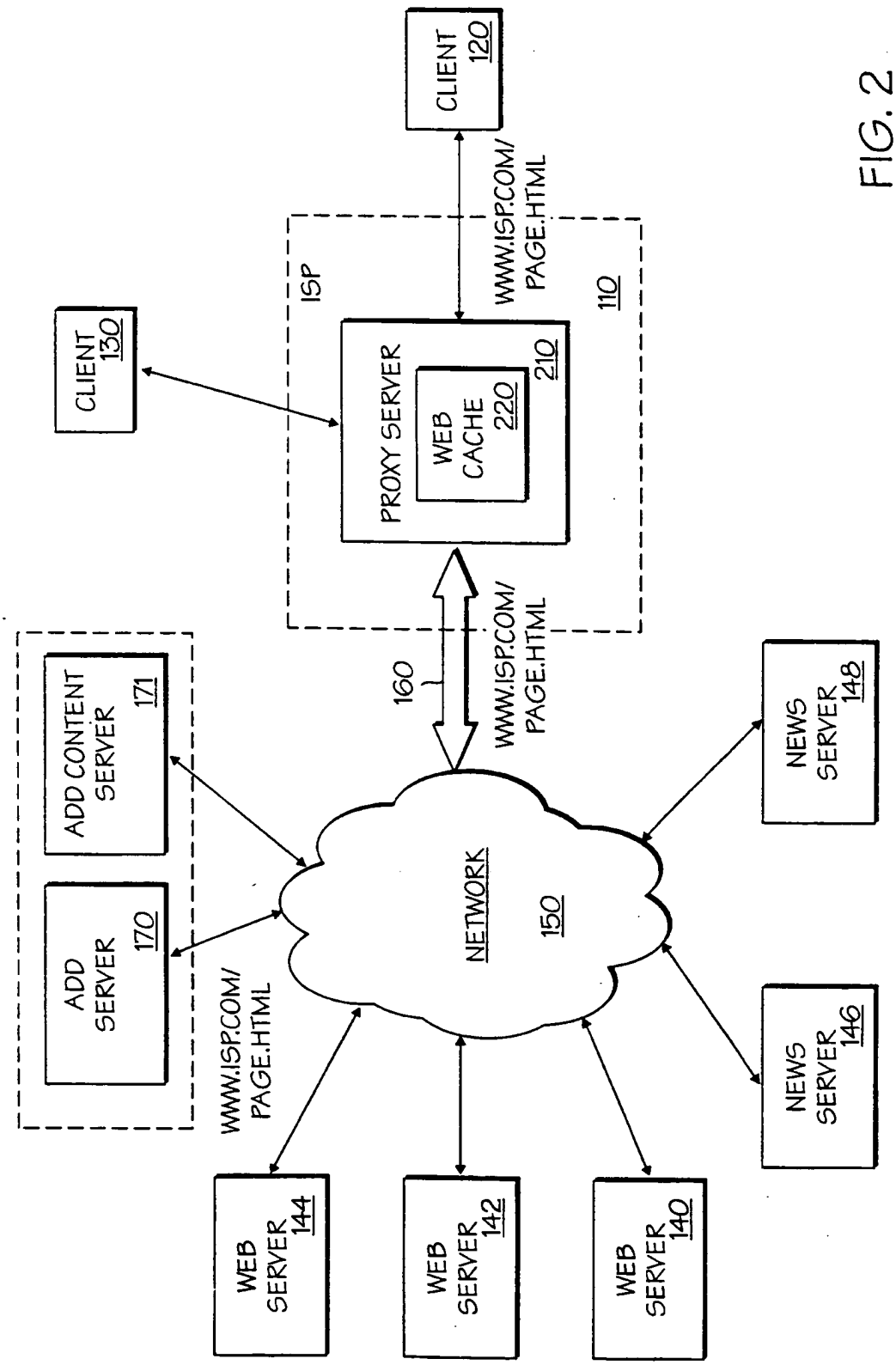

CLIENT 120
WWW.ISP.COM/ PAGE.HTML
ISP
110
CLIENT 130
PROXY SERVER 210
WEB CACHE 220
160
WWW.ISP.COM/ PAGE.HTML
ADD CONTENT SERVER 171
ADD SERVER 170
WWW.ISP.COM/ PAGE.HTML
NETWORK 150
NEWS SERVER 148
NEWS SERVER 146
WEB SERVER 144
WEB SERVER 142
WEB SERVER 140

FIG. 2

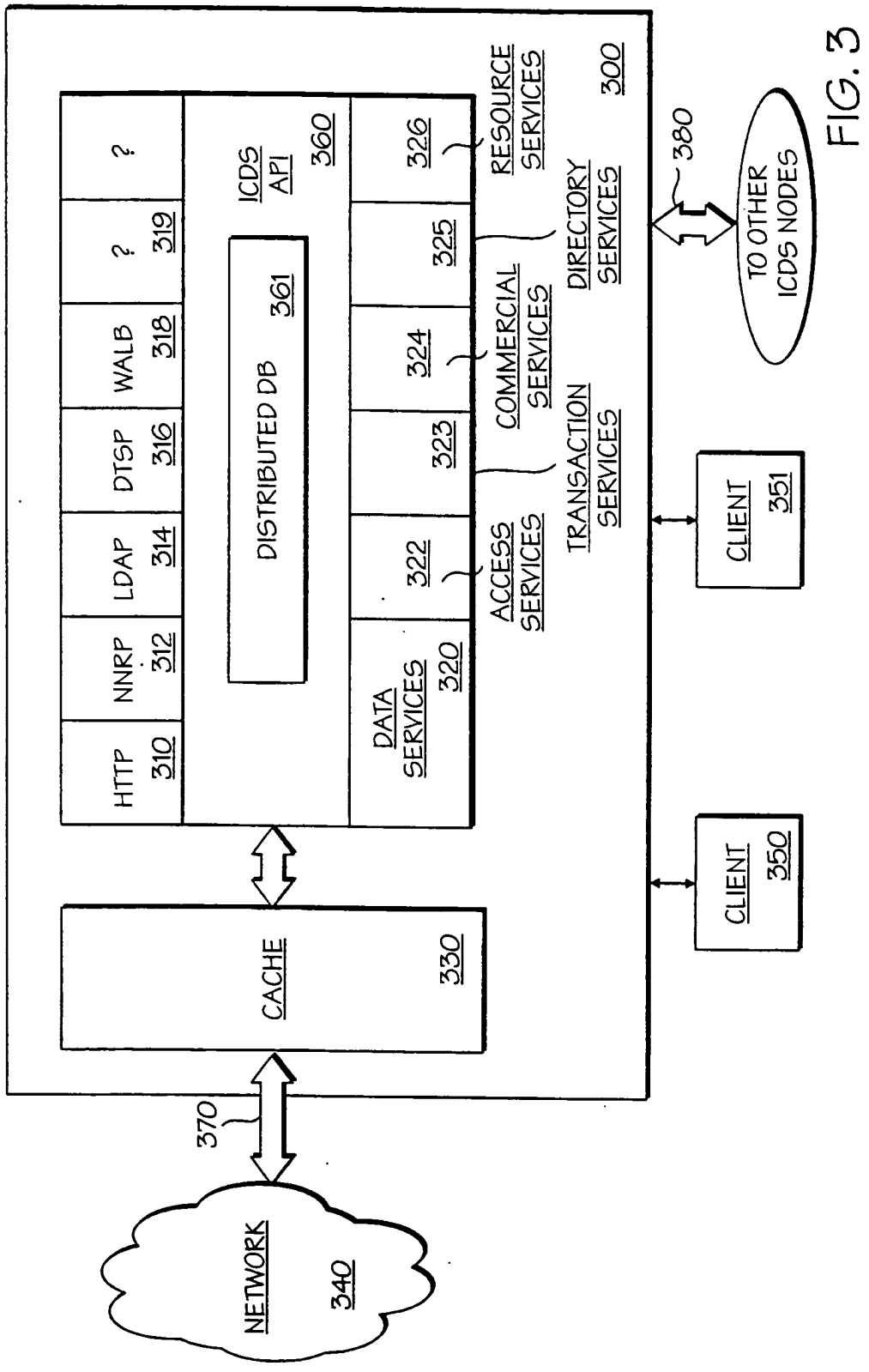

HTTP 310
NNRP 312
LDAP 314
DTSP 316
WALB 318
? 319
?
ICDS API 360
DISTRIBUTED DB 361
DATA SERVICES 320
ACCESS SERVICES 322
TRANSACTION SERVICES 323
COMMERCIAL SERVICES 324
DIRECTORY SERVICES 325
RESOURCE SERVICES 326
300
CACHE 330
CLIENT 350
CLIENT 351
380
TO OTHER ICDS NODES
370
NETWORK 340

FIG. 3

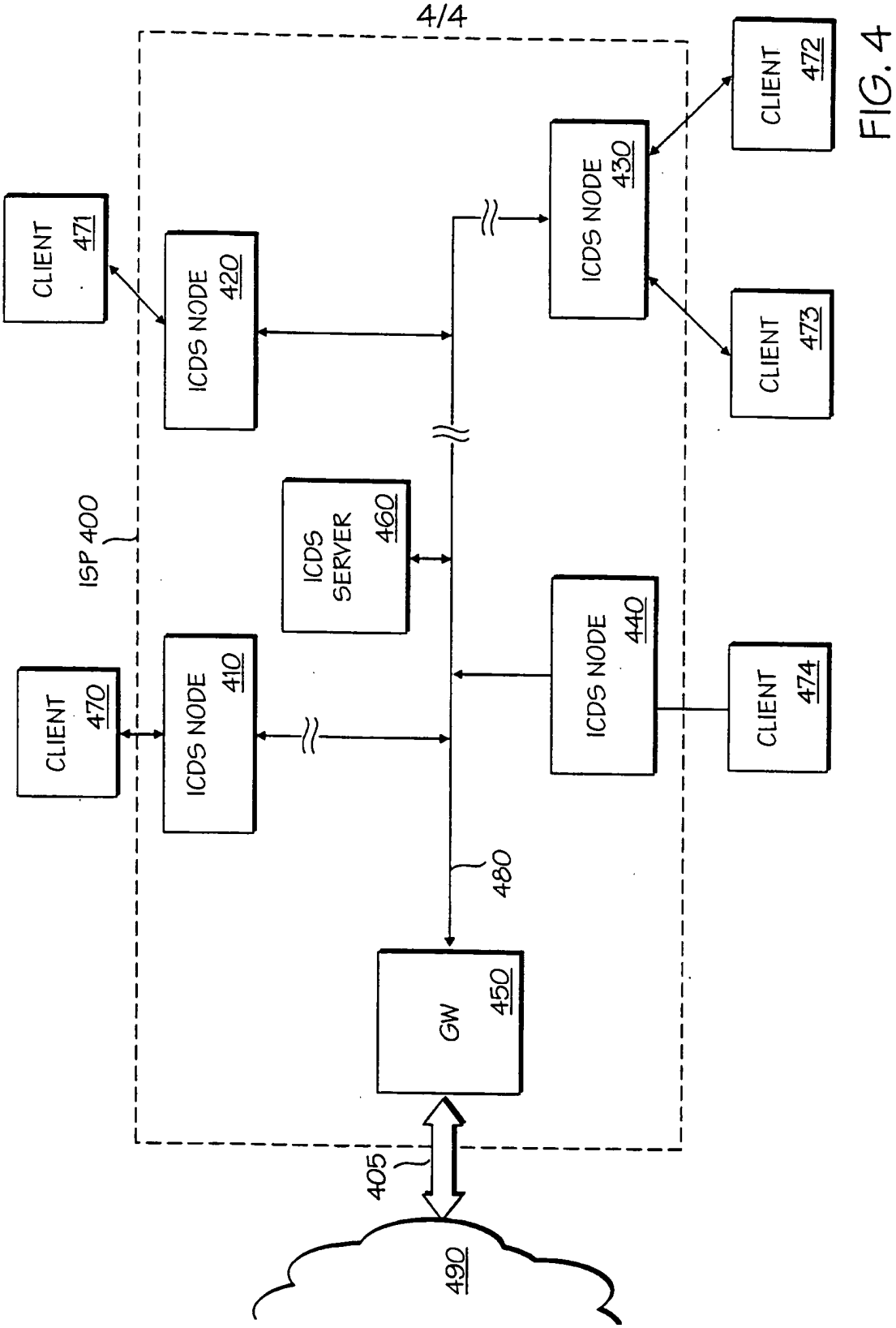

ISP 400
CLIENT 470
ICDS NODE 410
CLIENT 471
ICDS NODE 420
ICDS SERVER 460
ICDS NODE 430
CLIENT 472
CLIENT 473
ICDS NODE 440
CLIENT 474
480
GW 450
405
490

FIG. 4

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/08	A1	(11) International Publication Number: WO 99/34553
		(43) International Publication Date: 8 July 1999 (08.07.99)

(21) International Application Number: PCT/US98/27531

(22) International Filing Date: 30 December 1998 (30.12.98)

(30) Priority Data:
09/001,463 31 December 1997 (31.12.97) US

(71) Applicant: V-ONE CORPORATION [US/US]; Suite 300, 20250 Century Boulevard, Germantown, MD 20874 (US).

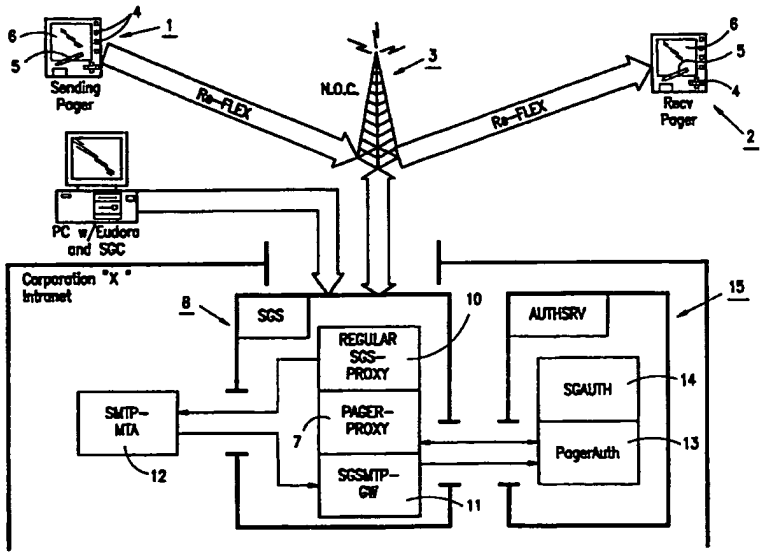
(72) Inventors: WRIGHT, Steven, R.; Apartment 21, 12010 Waterside View Drive, Reston, VA 20194 (US). BROOK, Christopher, T.; 7308 Pomander Lane, Chevy Chase, MD 20815 (US).

(74) Agents: URCIA, Benjamin, E. et al.; Bacon & Thomas, PLLC, 4th floor, 625 Slaters Lane, Alexandria, VA 22314 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

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: KEY ENCRYPTION SYSTEM AND METHOD, PAGER UNIT, AND PAGER PROXY FOR A TWO-WAY ALPHANUMERIC PAGER NETWORK



(57) Abstract

A method and system allows encryption services to be added to an existing wireless two-way alphanumeric pager (4) network by providing a pager proxy (7) which is arranged to receive an encrypted message from a sending pager (1) and re-packages it for retransmission to the destination pager (2). The sending pager encrypts the message using a session key, and encrypts the session key so that it can only be recovered by a secret key of the pager proxy. Authentication (13) of the sending pager and proxy server is provided by encryption of the session keys together with identifying data, and authentication of the message is provided by a message authentication code generated by computing a message authentication code based on the session key, identifying data, and the message.

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.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

KEY ENCRYPTION SYSTEM AND METHOD, PAGER UNIT, AND PAGER PROXY FOR A TWO-WAY ALPHANUMERIC PAGER NETWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a system and method of encrypting messages for transmission and/or receipt by a pager, and in particular to a system and method for which uses a standard two-way wireless pager protocol to send encrypted messages over an existing paging infrastructure. The invention also relates to a pager unit capable of sending and receiving encrypted alphanumeric messages over a wireless pager network, and to a pager proxy server which provides key management functions for enabling transmission of encrypted alphanumeric messages over the wireless pager network.

2. Description of Related Art

Paging systems capable of transmitting simple alphanumeric messages and displaying the messages on a miniature two-way pager are becoming increasingly popular. Such two-way paging systems enable messages like "Meet me at the gym at 6:00" or "I love you" to be both transmitted and received by equipment that is smaller, less complex, and less intrusive than a wireless telephone. The messages are transmitted as packets containing source and destination address data formatted for transmission over

the response channel of a wireless paging network, using a protocol that allows users to respond to messages directly from their pager units without having to use a telephone.

Two-way pagers are currently offered by Motorola and Wireless Access, with national paging services being provided by MTEL, which uses Motorola's Re-FLEX™ paging protocol. The Re-FLEX™ paging protocol allows users to respond to messages using a selection of pre-programmed responses or by formatting a free-form text reply, and in addition includes a TCP/IP protocol stack that allows the user to initiate messages to subscribers on wired networks, including e-mail and fax machine addresses.

The present invention concerns a method and system for encrypting and authenticating messages transmitted over the existing pager system, using the Re-FLEX™ protocol, or over other yet-to-be-implemented paging systems in the U.S. and elsewhere which may or may not use the Re-FLEX™ protocol. Unlike previously proposed arrangements, which either rely on complex encoding schemes and sophisticated hardware at the sending and destination ends of a transmission, over transfer of keys and authentication of keys using a telephone rather than the wireless network, the present invention offers the advantages of (i) providing authenticable key encryption of messages at the source of the transmission and key decryption at the destination, with protection of the communication by keys that are unique to each pager, rather than shared, and yet with no need for a key exchange between the sending and destination pagers, (ii) using existing two-way pager designs and paging system infrastructure, and (iii) providing the encryption capabilities without adding to carrier overhead. The addition of full key encryption and authentication capabilities to an existing pager system without adding to carrier overhead or capital costs distinguishes the system and method of the invention from all previously proposed pager encryption schemes.

An example of a previously proposed pager encryption scheme is described in

U.S. Patent Nos. 5,452,356 and 5,481,255, assigned to Data Critical Corp. Although the term "encryption" is used in these patents, the patents are directed primarily to a data compression and encoding protocol for enabling transmission of large volumes of data over a wireless pager network using modified transmitting and receiving hardware, including separate computers at each end of the transmission. The only discussion of encryption in these patents is a cursory reference to "encryption" as an added security layer provided by utilizing a "commercially available algorithm" (see, *e.g.*, col. 11, lines 15-32 of U.S. Patent No. 5,452,356) during encoding of the files by a computer connected to the pager. Because all encryption and decryption in the Data Critical patents is disclosed as being carried out by software on computers connected directly to the sending and receiving pagers, the only possible ways that true key encryption could be provided for would be to use encryption keys corresponding to decryption keys common to all possible recipients of the message, to use unique keys for each potential recipient but to store the corresponding encryption keys in the sender's computer, or to exchange keys prior to a transmission. While these alternatives might be reasonable in the context of, for example, a medical paging system in which all transmissions are between doctors or trusted medical personnel, none of them are suitable for use in connection with a paging system designed to transmit simple text messages using miniature handheld paging units and which is open to the general public, both because of the hardware intensive nature of the encoding scheme and the problem of key management.

In addition to the wireless pager protocol described in the Data Critical patents the prior art includes a number of patents describing authentication or encryption schemes that are used in connection with wireless paging, but are carried out over a telephone line. The systems described in these patents are more suited to traditional one-way paging environments than with two-way protocols, even though one of the patents issued only recently, and none disclose systems that can be practically applied to the current two-way paging networks.

U.S. Patent No. 5,668,876, for example, discloses a modified pager which provides authentication of a caller. The modified pager calculates a unique response code based on a transmitted challenge code, an input personal identification number, and an internal key. The resulting response code is converted into DTMF tones and transmitted by telephone to a central computer which authenticates the caller. This system does not provide for encryption of messages, or authentication by the receiving party of communications forwarded by the central computer, and yet requires a challenge response form of authentication which requires simultaneous two-way communications, which is currently neither possible nor required by existing two-way wireless pager protocols.

U.S. Patent No. 5,285,496 describes a paging system with two options: the first is to send and receive encrypted messages using private key encryption by transmitting a clear text message over a private communications line to a local client of the pager network where the message is encrypted using a private key, and broadcast over a pager network, and the second is to send the message in clear text by telephone directly to the central control system of the pager network, where the message is encrypted. However, neither of the two options provides for encryption of the original pager transmission, which must be sent in clear text form over a telephone line, and which, in the case where a local client computer is used, provides no way to maintain centralized control. In addition, for the local client computer option, in which the address is encrypted together with the message, the destination pager must decrypt every message sent over the system in order to determine whether a message is addressed to it, which is only possible in pager networks with a very limited number of participants.

In the system described in U.S. Patent No. 5,638,450, on the other hand, reception by a pager of encrypted messages over a radio frequency pager network is made possible by having the pager transmit an encryption key via DTMF tones over a telephone line to a central office, the central office then encrypting the messages before forwarding them

to the recipient. This system does not permit outgoing messages to be encrypted, and provides no way of key encrypting messages between two pagers on the network, and again is not applicable in the context of the present invention.

It will be appreciated that none of the above patents, representing the known pager message protection proposals, describes a system that enables true key encryption and authentication capabilities to be added to a conventional two-way wireless alphanumeric paging system of the type with which the present invention is concerned, using existing pager protocols and equipment, and in which any individual can send a simply alphanumeric message by keying the message into a miniature two-way pager (or choosing from a menu of pre-stored messages), entering a destination address, and pressing a send button, the message then being retrievable by the intended recipient by a simple keystroke on the recipient's pager, with the message being encrypted by a key unique to the sending pager and decrypted by a key unique to the destination pager. In contrast, the present invention not only provides these capabilities, but adds further levels of security by using strong secret or private key based encryption algorithms, with multi-tier authentication of a transmitted packet, while permitting central registration and billing for encryption services and recovery of messages by legal authorities without adding to carrier overhead or increasing the costs of the paging service for users who do not require encryption.

All of the above advantages of the system and method of the invention are made possible through the use of a proxy server to intercept an encrypted message and repackage it for delivery to the intended recipient in a form that the intended recipient is capable of reading, thus eliminating the need for shared keys or key exchange between the sender and ultimate recipient of the message or complex, hardware-intensive encoding schemes, and allowing encrypted messages to be transmitted using existing two-way alphanumeric pager protocols. Because the invention involves key encryption and not encoding of the message, and requires knowledge by the sending and receiving

units of only one or two keys (for example, a private key unique to the pager and a server's public key), encryption being simpler to implement than encoding since it merely involves performing arithmetically combining the message with the key, the present invention can be used with existing pager hardware and protocols, and by avoiding the need for challenge/response authentication, the present invention can be used with existing channels and therefore with the existing pager infrastructure. None of the previously proposed systems and methods has these capabilities.

Not only does the use of a proxy server relieve the sending and receiving pagers of key management functions, but the manner in which the invention utilizes strong encryption capabilities, by separately encrypting the session key, further minimizes the processing resources required by the sending and receiving pagers. Essentially, encryption of the message itself can be carried out with a relatively short session key to minimize usage of the processor, while the relatively short session key can be protected by a strong encryption algorithm. Because the session key is not re-used, key integrity can easily be maintained.

SUMMARY OF THE INVENTION

It is accordingly a first objective of the invention to provide a system of adding full key encryption services to a pager network, allowing key encrypted alphanumeric messages to be sent by any pager unit registered with the encryption service provider to any other registered pager unit via the network, as well as to e-mail addresses, fax machines and other destinations capable of receiving text messages.

It is a second objective of the invention to provide a method of adding full key encryption services to a pager network, allowing key encrypted messages to be sent by any pager unit registered with the encryption service provider to any other registered pager unit via the network, as well as e-mail addresses, fax machines and other

destinations capable of receiving text messages.

It is a third objective of the invention to provide a system which allows encryption of alphanumeric messages by a paging unit for wireless transmission over a paging network in a manner which is transparent to the person sending the message, and which allows decryption and display of the messages by a receiving pager in a manner which is transparent to the person receiving the message.

It is a fourth objective of the invention to provide a method which allows encryption of messages by a paging unit for wireless transmission over a paging network in a manner which is transparent to the person sending the message, and which allows decryption and display of the messages by a receiving pager in a manner which is transparent to the person receiving the message.

It is a fifth objective of the invention to provide a system and method of adding encryption capabilities with centralized key management and unique secret keys for each user, without modification of existing pager network infrastructure or paging transmission protocols.

It is a sixth objective of the invention to provide a system and method of encrypting text messages capable of being transmitted over a pager network, which can be provided as an add-on or option to the services provided by the pager network, and which can be centrally managed using a proxy server connected to the network to provide the encryption services to subscribers who select the encryption option.

It is a seventh objective of the invention to provide a system and method of authenticating messages transmitted in encrypted form over a pager network, without the need for an authentication channel or challenge/response protocol.

It is an eighth objective of the invention to providing a standard alphanumeric pager unit with the capability of encrypting, decrypting, and authenticating messages transmitted using a two-way alphanumeric pager protocol, with minimal or no hardware modification.

It is a ninth objective of the invention to provide a proxy server arrangement which can be connected to the network operations center of a pager network in order to manage transmission of key encrypted messages over the network.

These objectives are achieved, in accordance with the principles of a preferred embodiment of the invention, by using a pager proxy server to carry out decryption of a message encrypted by a session key and received from the sending pager, and to have the pager proxy generate a new session key for re-encryption of the message transmitted to the receiving pager, with the original session key being encrypted at least by a secret key shared by the sending pager and the pager proxy server or by a public key corresponding to a private key of the pager proxy server, and the new session key being encrypted either by a secret key shared by the pager proxy server and the destination pager or a public key corresponding to a private key held by the destination pager, thereby freeing the sending and destination pagers from having to store more than one secret key or of having to carry out a direct exchange of keys, and allowing each pager on the network to be provided with a unique key.

In accordance with the principles of an especially preferred embodiment of the invention, in order to encrypt a message, the sending pager must have hard-coded into memory a unique identification number and a secret key associated with the identification number. When a user is ready to send an encrypted message, he or she begins by entering the message to be sent, after which the user is prompted for an access code to gain access to the encrypted shared key, the encrypted shared is decrypted, and a session key is generated. The message that was entered by the user is then encrypted with the

session key, and the session key is encrypted with the public key of the pager proxy server, or a shared secret key of the sending pager, and appended to the encrypted message for transmission via the network operations center to the pager proxy server.

Pager messages are formatted in accordance with standard pager protocols to include a destination header, which is generally the address or telephone number of the receiving pager, and with an additional space in the header to indicate that the message is encrypted, as will be explained in more detail below. When the network operations center receives a message that is in encrypted form, it forwards it to the encryption service center, which must at least include a pager proxy server, using an appropriate protocol, examples of which include but are not limited to TME-X and TNPP. In the illustrated embodiment, the pager proxy server is included in a gateway server in order to enable the system to package e-mail messages for transmission in encrypted form to pagers on the pager network, or to package pager messages according to an e-mail protocol for transmission over a wired network such as the Internet to an e-mail address, but it will be understood by those skilled in the art that the pager proxy may be operated as a separate unit.

In the illustrated embodiment of the invention, the pager proxy server has the role of verifying the authenticity of the message sent by the sending pager, decrypting the data with its private key or alternatively with a secret key shared with the sending pager to obtain the session key that was generated by the sending pager, and decrypting the message with the session key generated by the sending pager. Once this is accomplished, the server generates a new session key to encrypt the message with, and then encrypts the session key with a secret key shared with the destination pager or with a public key corresponding to the private key of the destination pager, or alternatively with a secret key shared with the destination pager, the two entities being appended together and sent to the recipient pager. The destination pager, after receiving the encrypted message, alerts the user and, when the user is ready to read the encrypted page, prompts him or her

for the access code to begin decryption of the appropriate shared secret key or private key, which is then used to decrypt the session key used to decrypt the message.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic diagram showing the principal elements of a pager encryption system constructed according to the principles of a preferred embodiment of the invention.

Fig. 2 is a schematic illustration summarizing the operation of the two-way pager for sending an encrypted message over a wireless network in accordance with the principles of a preferred embodiment of the invention.

Fig. 3 is a functional block diagram of a module used by a two-way pager to encrypt a message and package it for wireless transmission over a pager network to a network operations center.

Fig. 4 is a functional block diagram of a module used by a pager proxy server to authenticate the sender of an encrypted message, authenticate the message, and extract information from the message which can be used to re-package the message for transmission a destination address.

Fig. 5 is a functional block diagram of a module used by the pager proxy server to repackage a message and send it to the network operations center for transmission for re-transmission over the wireless pager network to a destination pager.

Fig. 6 is a functional block diagram showing the principal elements of a module used by a destination pager to decrypt and display a message received in encrypted form from the network operations center over the wireless paging network.

Fig. 7 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 3.

Fig. 8 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 4.

Fig. 9 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 5.

Fig. 10 is a flowchart of a preferred process corresponding to the functional block diagram of Fig. 6.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As illustrated in Fig. 1, the system of the preferred embodiment of the invention allows encrypted communications between a sending pager and a receiving pager via a two-way wireless paging system such as M-TEL's system, using two-way alphanumeric pagers such as, but not limited to, the Motorola and Wireless Access pagers. The basic elements of the system are a sending pager 1, a receiving pager 2 which may be identical to the sending pager, and a network operations center (NOC) 3 which provides basic message forwarding and subscription management services for all communications carried by the system.

As is conventional, the sending and receiving or destination pagers (or pager units) 1 and 2 include function and data entry keys 4, and/or a stylus 5 or other data entry device, for allowing a user to input and send alphanumeric messages, and an LCD or other device 6 which allows received alphanumeric messages to be displayed. The pagers can also provide other functions such an alarm function to alert the user that a message has been received, and includes a microprocessor and circuitry capable of formatting an

input message and transmitting it to the network operations center according to an appropriate protocols, including but not limited to the ReFLEX™ protocol. The sending and receiving or destination pagers also include a memory for storing a unique user identification number (UID) that identifies a particular pager for addressing purposes, and other information such as a password that can be used to prevent unauthorized users from accessing the transmission or message display functions of the pager, as well as an addressing mode (AM) generator that is used in the pager protocol to indicate the type of addressing used by the paging system, and a timer that can be used to generate a message number.

In order to be used with the system and method of the illustrated embodiment of the invention, the pager memory must also have stored therein at least a private key of the pager unit, a corresponding public key of the pager unit, and a public key corresponding to a private key of the server, for encrypting either the message itself or a session key used to encrypt the message, and software capable of running on the included processor for performing an encryption algorithm and a decryption algorithm. In addition, according to the preferred embodiment of the invention illustrated in Figs. 2-10, the pager must be capable of generating a session key for each message to be transmitted, storing a private key unique to the pager which is used to authenticate the pager, and computing a message authentication code which is used to authenticate the message being transmitted or received.

It will be appreciated by those skilled in the art, however, that whenever a public key or private key is required, a shared secret key could be substituted using an appropriate algorithm, and that while the use of session keys is highly advantageous, the session key could also be eliminated in favor of public-private key encryption. In addition, while the illustrated system provides both encryption and decryption capabilities in at least two pagers, so that each pager can send or receive messages, the system and method of the invention could also be applied to systems in which some or

all of the pagers have reception capabilities only, *i.e.*, in which some or all of the pagers are designed to allow the pagers to receive encrypted messages originating from e-mail addresses and/or two-way pagers, but not to originate messages. Conceivably, the system and method of the invention could even be applied to systems in which at least some of the pagers are capable of sending encrypted messages, but not receiving and decrypting them, although such a system would seem to make little commercial sense. In any case, it will be appreciated that the system and method illustrated in Figs. 2-10 are intended as being illustrative in nature only, and should not be interpreted as being limitative of the scope of the invention.

As indicated above, the number of keys required of a pager to encrypt and decrypt messages is at most two, so that the key storage requirements are minimal. The encryption algorithms themselves simply involve a series of mathematical steps, and are well within the capabilities of the microprocessors used in the conventional pagers, as are message authentication code generating techniques such as CRC or SHA1. The session key used in the preferred embodiment to encrypt the message itself consists, in a practical implementation, of just sixteen characters (128 bits), and thus encryption of the alphanumeric message using RC4 or a similar stream cipher or other algorithm which makes use of a shared secret key can be accomplished without a large amount of processing resources, while strong overall protection of the transmission is still provided because the more processor intensive encryption algorithms are reserved for encryption of the relatively small session key rather than the alphanumeric message itself. Of course, the session key is not limited to a particular bit size, and it is possible for example to use 256 bit session keys, or longer or shorter session keys as desired.

In the preferred embodiment, encryption of the session key is carried out by RSA (1024 bits) but other stronger private key algorithms such as ECC PK1 (~2500 bits) can also be used, as well as shared secret key-based encryption methods such as RC4. The public-private key encryption algorithms are preferred not only because of the strong

encryption provided, but also because the permit authentication of the sender, as explained below, but legal or other considerations may also affect the choice of encryption algorithm, and thus the system of the invention is designed to permit the use of different mutually exclusive encryption algorithms by the sending and destination pagers.

The sending pager 1 illustrated in Fig. 1 transmits messages to the network operations center 3 in the form of a packet that includes a clear text applications header that tells the center to forward the text to the pager proxy server 7, which is conveniently though not essentially included in a gateway 8 capable of network communications as well as the pager encryption and decryption functions required by the present invention. Forwarding of the packet to the pager proxy or gateway server preferably involves use of a network data transfer protocol such as TME-X, although the manner in which the packet is forwarded to the proxy will depend on the wireless protocol used by the pager network and the capabilities of the network operations center. TME-X is a preferred transfer protocol for use with Re-FLEX encoded packets because of the presence of a TCP/IP stack in the standard format packets that allows the Re-FLEX™ protocol to communicate directly with computer networks.

The gateway 8 may include a general purpose proxy server 10 such as the one described in U.S. Patent No. 5,602,918, entitled "Application Level Security System And Method," and also in U.S. Patent Application Ser. No. 08/917,341, filed August 26, 1997, entitled "Multi-Access Virtual Private Network," both of which are incorporated herein by reference. The two patent documents describe a system currently available from V-One Corporation of Germantown, Maryland under the name SmartGate™ (SG in the figures) which is especially suitable for use with the pager proxy of the present invention, although the pager proxy server of the invention could also be used with other gateway servers, or without any network connection capabilities.

As illustrated, gateway 8 also includes a dedicated e-mail server or gateway 11, and e-mail protocol message transfer agent (MTA) 12 for transferring messages from the gateway server 10 to the e-mail gateway. Both the e-mail gateway 11 and pager proxy 7 may be physically incorporated in the gateway server or provided on independent or separate computers, and are connected to a pager authentication module 13 which may be physically incorporated into a general purpose gateway authentication module 14 of a separate authentication server 15, combined with the gateway server, or may be provided as an independent unit.

Computers on the network with capabilities of communicating with the general purpose proxy server are represented in Fig. 1 by computer 16, and include gateway client software that permits the computer to establish a secured communications path to the gateway server, as well as an e-mail program which packages messages in an appropriate format such as that provided by the SMTP protocol for transmission over the secured communications path established by the gateway client software. An example of an e-mail program is "Eudora™," although the use of standard protocols such as SMTP and Re-FLEX™ allows any e-mail program to communicate with the gateway and thence with the pager network, so that the system of the invention is not limited to use in connection with any particular e-mail program, the conventional pager network already being equipped to handle e-mail transmissions to or from the wireless network. The invention may be considered to apply equally to pager-to-pager communications, pager-to-email communications, and email-to-pager communications. In addition, it is possible that the invention could be adapted to communications originating from a fax machine, in which case the clear packet transmitted by the fax machine over a telephone line would be processed by a facsimile proxy for packaging and encryption by the pager proxy, and messages addressed to the fax machine would be decrypted by the pager proxy and forwarded to the facsimile proxy for transmission as clear text over a telephone line, the principles of the invention still being applicable to the encryption and decryption of the messages by the pager proxy and sending or receiving pagers.

Turning to the specific embodiment illustrated in Figs. 2-10, the system and method of the invention take the form of modifications to the header of the transmission packet sent by the sending pager 1 and/or the pager proxy 7. Essentially in order to send messages over the paging system, the sending pager and pager proxy, (or pager proxy alone in the case of a message originating from computer 16 or a source of clear text messages such as a facsimile machine) generates a header which includes the information necessary to enable processing by the recipient of the packet, and in the case of the pager proxy, for forwarding of a repackaged packet to a destination address. The header should at least include the session key encrypted message, the encrypted session key, a sender identification number, and a destination header or address, but because the header format will vary if a protocol other than Re-FLEX™ is used, it should be appreciated that the other information contained in the illustrated header, and the position of the information, can be varied without departing from the scope of the invention, and the invention is intended to encompass headers formatted for other alphanumeric wireless paging protocols, as well as for encryption algorithms and authentication protocols other than the specific algorithms and protocols indicated.

Fig. 2 illustrates the format of the preferred header, which is divided into three fields. It is to be understood that while the illustration refers to the communication between the sending pager and the pager proxy, the same header will be used for the communication between the pager proxy and the destination pager, with appropriate substitutions of addresses and keys as explained in more detail below. As shown in Fig. 2, the first field is a clear text field that contains the encryption method indicator EM, pager addressing mode (AM), and user identification number (UID) (sometimes referred to as a PIN, but not to be confused with the password entered by the user to access pager functions), while the second field contains the encrypted session key (SESKey1) and various data referred to as "header data" (HdrData) including the destination header or address (DH) and a message authentication code (MAC), the information in the second field being encrypted by the unique private key of the sending pager (pv.sender) in order

to authenticate the sender, and by a public key corresponding to a private key held by the server (pb.server) in order to protect the contents of this field. The third field contains the message encrypted by the session key.

The various fields illustrated in Fig. 2 may be formatted in any convenient manner permitted or required by the protocol used to package the data in the fields for transmission, but in the illustrated example most or all of the data in at least fields one and two can conveniently be in hexadecimal format. Whenever the drawings illustrate a hexadecimal number, the number ## will be preceded by a "0x" to form 0x##.

The encryption method indicator EM indicates which of the possible encryption methods handled by the server is being used to encrypt the session key and other information in field 2, so that the session key can be recovered and used to decrypt the encrypted message in field 3. As indicated above, possible encryption methods include the RC4 secret key encryption method, which requires the parties to the communication to have a shared secret key that is used for both encryption or decryption, and the RSA public key encryption method, which is the method illustrated in Fig. 2. The indicator itself is simply a number assigned to the encryption method. While any given pager will generally have only a single encryption method stored in memory, it is possible for the pager proxy to be arranged to handle multiple different methods and thus need to have an indication of the type of encryption method, to accommodate different pager systems or legal requirements, particularly if international pager traffic is involved.

The addressing mode (AM) indicates the type of address involved. For example, in the U.S., pager addressing modes are assigned one application header, while e-mail addressing modes are assigned another application header. This indicator may not be necessary in all protocols since the destination header may be unique to a specific type of address, but is included in field 1 as part of the Re-FLEX™ protocol.

The user identification number (UID) included in clear text in field 1 and in encrypted form in field 2, is the unique address assigned to the pager, and is used to indicate the source of the message so as to enable the pager proxy to retrieve the appropriate public decryption key (pb.sender), and for use in authentication of the sender and for display by a receiving pager. Preferably, this number is hard-coded into memory so that it cannot easily be altered, and in current U.S. paging systems is in the form of a ten digit number.

The header data (HdrData) of the second field includes an application header (AH), which included in a field having variable length and string value, the address mode and destination header (AM/DH), the user identification number (UID), which is the same as the one included in field 1, and a message number (MSGNO) and message authentication code (MAC). In addition, e-mail address protocols require a byte indicative of address length to be added where the address mode indicates an e-mail address.

For purposes of the present invention, the message number can be any arbitrary number, although the use of a time-related reference, as allowed by the Re-FLEX protocol, is useful for account tracking or billing purposes, and in addition can be used to ensure that received message is not a recording of a message sent earlier and intercepted by an unauthorized party. For example, the message number has previously been defined as the number of seconds since January 1, 1970.

The message authentication code is a checksum used to verify that the recovered message is identical to the original message, and may be computed using an error correction code function such the cyclic recovery code (CRC) function, with CRCs being used in the illustrated embodiment or, alternatively, by computing a hash or one-way combination of the header data with the message and the session key, using an algorithm such as SHA1. By combining the message with other data to obtain the message

authentication code in a way that can only recreated if the data used to recreate the code is the same as the data originally used to generate the code, the code can be used to authenticate the message, *i.e.*, to verify that the message has not been altered since the time when the code was first generated, as will be described in more detail below. It will be appreciated that the exact form of the message authentication code is not a part of the present invention, and that any message authentication code may be used so long as it can be used to authenticate the message in the manner described below.

The three blocks above the header data in Fig. 2 indicate the source of the data for the various fields. The manner in which the data is combined to form the fields is described in more detail in connection with Figs. 3-10, but the sources of the data may be summarized as (i) information entered by the user, which consists of the message (MSG) and the recipient address which forms the destination header, (ii) information stored in memory, including private and public keys of the pager, a public key of the pager proxy server, an access code which is to be compared with an access code input by the user, the encryption method indicator (EM), the user identification number (UID), and the application header, and (iii) information generated at runtime, *i.e.*, during assembly of the packet header, including the session key (SESKey), the message number (MSGNO), the addressing mode (AM), and the message authentication code (MAC).

The details of the manner in which the data shown in Fig. 2 is assembled by sending pager 1 to form the header shown in Fig. 2 is illustrated in the functional block diagram of Fig. 3, as well as the flowchart of Fig. 7. As illustrated in Fig. 3, the pager 1 includes a user input 20 connected to keys 4 or stylus 5, which supplies the destination header (DH) to a functional block 21 which assembles the header data (HdrData), and to a functional block 22 which computes the message authentication code (MAC). In addition, the user input 20 supplies the message to functional block 28, the output of which is field 3 of the header.

Pager 1 also includes a memory 24 which stores the encryption method (EM), the application header (AH), the user identification number (UID) and the encryption method identifier (EM), which are supplied directly to functional block 23 for inclusion in field 1, the user identification number and application header being also supplied to functional block 21 for inclusion in the header data, which in turn is supplied to functional block 22 for inclusion in the message authentication code. The address mode (AM), which is associated with the destination header (DH) in the header data is generated by an address mode generator 25 which can be in the form of a look-up table, device that reads a particular identifying bit in the destination header, or other device, and the message number can be generated by a counter, timer, or other device 26 depending on the nature of the message number. Finally, the session key (SESKey1) for this embodiment of the invention is an eight character string generated by a random or pseudorandom number generator 27, which supplies the session key to functional block 28 for use in encrypting the message (MSG), to functional block 22 for inclusion in the message authentication code, and to functional block 29 for encryption together with the header data by the private key of the sender. The output of functional block 29 is supplied to functional block 30 for encryption by the public key of the server, the output of block 30 serving as field 2 of the header for the packet transmitted by the sending pager.

It will be appreciated by those skilled in the art that any of the functional blocks and data or number generators illustrated in Fig. 3, or in Figs 4-6, may be implemented either by hardware or software, and that while distinguishable by function, the functions may be carried out using common subroutines, hardware, or software.

Turning to Fig. 4, the pager proxy 7 includes a database of public keys corresponding to the unique public keys of pagers registered with the encryption service provider that operates the proxy server. The database is accessed by functional block 31 according to the clear text user identification number (UID) present in the header of a packet forwarded to the pager proxy by the network operations center. Field 2 is

decrypted by functional block 32 using the private key of the server (pv.server) and by functional block 33 using the public key of the sender (pb.sender) to recover the session key, and the user identification number (UID) recovered from field 2 is compared by functional block 34 with the user identification number of field 1 to verify the authenticity of field 2 and recover the session key (SESKey1). A functional block 35 then uses the session key to decrypt the message (MSG).

The message recovered by the pager proxy is authenticated in functional block 37, by comparing the message authentication code recovered from field 2 with the output of a functional block 36 that computes the message authentication code based on the destination header (DH), application header (AH), user identification number (UID), message number (MSGNO), and session key (SESKey1) recovered from field 2, and the message recovered from field 3. The message, session key, and header data (HdrData) are then made available by functional block 38 to an encryption or repackaging module, illustrated in Fig. 5, for repackaging in a way that will enable decryption by a destination pager.

As shown in Fig. 5, the application header (AH) and message number (MSGNO) received from functional block 38 is provided to functional blocks 41 and 42 for inclusion in the header data and message authentication code, while the address mode (AM) and encryption method (EM) obtained from field 1 of the packet received from the sender is passed to functional block 43 or regenerated for inclusion as clear text in the packet header. In order to permit decryption and authentication of the repackaged header by the receiving pager, however, the destination header (DH) and user identification number (UID) are swapped, so that the original destination header is supplied by the pager proxy to functional blocks 41, 42, and 43 as the user identification number (UID), and the original user identification number are supplied to functional blocks 41 and 42 as the destination header (DH). Functional block 42 generates a message authentication code based on the new destination header (DH), application header (AH), user

identification number (UID), message number (MSGNO), while a new session key (SESKey2) is generated by functional block 44 in the same manner as functional block 27 shown in Fig. 3, and the resulting message authentication code (MAC) together with the new session key and header data from functional block 41 are encrypted by functional block 45 using the private key of the server (pv.server) before being sealed by functional block 46 using the public key of the destination pager (pb.recipient) and included in the header as field 2. Functional block 47 receives the message and new session key and re-encrypts the message using the new session key and an algorithm such as RC4 to generate field 3, fields 1-3 being assembled into a packet 50 for transmission to the destination pager 2 via the network operations center 3.

Again, those skilled in the art will appreciate that all of the functional blocks illustrated as being present in the proxy server and/or proxy authentication module may be implemented as software, hardware, or a combination of software and hardware, and may be varied depending on the encryption method and requirements of the pager protocol.

In addition, those skilled in the art will appreciate that the illustrated embodiment could be modified by eliminating the session key and instead using public key encryption of the message. Alternatively, instead of having the pager proxy perform any decryption of the message, the original session key could simply be re-encrypted by the pager proxy using at least the public key of the destination pager as described above, or a secret key shared with the destination pager, in which the encrypted message would simply be forwarded to the destination pager unit with the session key re-encrypted so that it can be recovered by the destination pager. While neither of these options is currently preferred because elimination of the session key leaves transmissions vulnerable to recording, and elimination of message decryption by the pager proxy makes message authentication more difficult, they should nevertheless be considered to be within the scope of the invention.

Turning to Fig. 6, the destination pager 2 includes functional blocks mirroring those of the server for decrypting messages and authenticating packets received from the pager proxy 7 via the network operations center 3. These include functional block 51 for retrieving the server public key (pb.server) from memory, functional blocks 52 and 53 for decrypting the field 2 using the recipient private key (pv.recipient) and the server public key, functional block 54 for comparing the user identification number recovered from field 2 with the user identification number in field 1, functional block 56 for decrypting the message (MSG) using the session key (SESKey2) recovered from field 2, and functional blocks 57 and 58 for generating a message authentication code and comparing it with the message authentication code recovered from field 2. It will be noted that functional block 57 may also be used to generate a message authentication code for an outgoing message, avoiding duplication of the hardware or software which performs this function.

Finally, destination pager 2 includes a functional block 59 for displaying the message (MSG) and destination header (DH) corresponding to the user identification number of the sending pager, and for alerting the user as necessary that a message has been received. The display is identical to that used for an unencrypted message, and thus the decryption operation is entirely transparent to the user.

The method steps that implement the functions illustrated in Figs. 3-6 are as follows:

First, as shown in Fig. 7, upon input of a message and destination address by the user of a pager (step 100), which may follow the input and verification of a password (not shown), a message number, address mode, and session key are generated (step 110) and the encryption method identifier, application header, user identification number, server public key, and sender private key are retrieved from memory (step 120). The encryption method identifier, address mode, and user identification number are included in field 1 (step 130), a message authentication code based on the destination header, application

header, user identification number, message number, message, and session key is computed (step 140), and the application header, user identification number, destination header, message number, message authentication code, and session key are encrypted by the private key of the sending pager (step 150) and then by the public key of the pager proxy (step 160) to obtain field 2 of the packet header. Finally, the message is encrypted by the session key (step 170) to obtain field 3, and the packet header is transmitted via the network operations center to the pager proxy (step 180).

Upon receipt by the pager proxy, as shown in Fig. 8, the public key of the sending pager is retrieved based on the user identification number in field 1 (step 200), and field 2 of the packet is decrypted by the private key of the server (step 210) and then by the public key of the sending pager (step 220) based on the encryption method identified by the identifier in field 1. Authentication of the sender is provided by comparing the user identification number recovered from field 2 with the user identification number in field 1 (step 230), the message included in field 3 is decrypted using the session key recovered from field 2 (step 240), and authentication of the message is provided by generating a message authentication code based on the destination header, application header, user identification number, message number, and session key recovered from field 2 together with the decrypted message (step 250), and by then comparing the computed message authentication code with the message authentication code recovered from field 2 (step 260).

As illustrated in Fig. 9, after authenticating the information contained in field 2, the proxy server generates a new session key (step 300), encrypts the message using the new session key (step 310), assigns the original user identification as the new destination header and the original destination header as the new user identification number, computes a new message authentication code (step 330), encrypts the address header, message number, new user identification number, new destination header, new session key, and new message authentication code using the private key of the server (step 340),

encrypts the result of step 340 using the public key of the destination pager (step 350), and assembles the header and packet for RF transmission to the destination pager via the network operations center (step 360).

As illustrated in Fig. 10, upon receipt by the destination pager, as shown in Fig. 8, the public key of the pager proxy server is retrieved based on the user identification number in field 1 (step 400), and field 2 of the packet is decrypted by the private key of the destination pager (step 410) and then by the public key of the pager proxy server (step 420) based on the encryption method identified by the identifier in field 1. Authentication of the sender is provided by comparing the user identification number recovered from field 2 with the user identification number in field 1 (step 430), the message included in field 3 is decrypted using the session key recovered from field 2 (step 440), and authentication of the message is provided by computing a message authentication code based on the destination header, application header, user identification number, message number, and session key recovered from field 2 together with the decrypted message (step 450), and by then comparing the computed message authentication code with the message authentication code recovered from field 2 (step 460). Finally, after authentication of the user identification number and message, the user is alerted that a message has been received and the decrypted message and information contained in the destination header are displayed at the request of the user (step 470).

Having thus described a preferred embodiment of the invention in sufficient detail to enable those skilled in the art to practice the invention, it is nevertheless anticipated that numerous variations and modifications of the invention will occur to those skilled in the art, and it is intended that all such variations and modifications be included within the scope of the invention. For example, although the preferred embodiment of the invention has the pager proxy re-package the message by first decrypting it, and then re-encrypting it using a new session key, it is also within the scope of the invention to have the pager proxy decrypt only the session key and re-encrypt the same session key using

the public key or shared secret key of the destination pager. Accordingly, it is intended that the above description not be taken as limiting, but rather that it be defined solely by the appended claims.

I claim:

1. A system for adding encryption services to an existing pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, at least two of said pager units comprising:

means for inputting an alphanumeric message and a destination address;

means for including the alphanumeric message in a packet for transmission to the destination address by wireless transmission via the network operations center;

means for receiving an alphanumeric message from the network operations center; and

means for displaying the alphanumeric message received from the network operations center,

wherein the system for adding encryption services comprises:

means in at least one of said pager units for encrypting a message and transmitting the encrypted message via the network operations center to another of said pager units;

means in said another one of said pager units for decrypting and displaying the encrypted message; and

a pager proxy server including means for receiving a packet containing the encrypted message that has been sent to the network operations center, decrypting at least a portion of the packet, and re-encrypting said portion of the packet for delivery to said another of said pager units via said network operations center.

2. A system as claimed in claim 1, wherein said means for encrypting the message comprises means for encrypting the message by a secret key.

3. A system as claimed in claim 2, wherein said secret key is a first session key generated by a sending pager unit, said sending pager unit further comprising means for encrypting said first session key by a public key corresponding to a private key held by the pager proxy server so that the session key can be recovered only by the paging proxy server.

4. A system as claimed in claim 3, wherein said sending pager unit further comprises means for encrypting at least the first session key by a private key of the sending pager unit, and wherein said pager proxy server includes means for retrieving a public key corresponding to the private key of the sending pager unit for use as a first level authentication of the sending pager unit.

5. A system as claimed in claim 4, further comprising means for appending a unique user identification number of the sending pager unit to the header in clear text form, said user identification number being hard-coded into the sending pager unit.

6. A system as claimed in claim 5, wherein said means for encrypting at least the session key by a private key of the sending pager unit also encrypts the user identification number of the sending pager unit, and said paging proxy server includes means for decrypting the encrypted user identification number together with the first session key and comparing it with the clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and first session key.

7. A system as claimed in claim 4, wherein the sending pager unit further comprises means for generating a first message authentication code based on various header data and the message and encrypting the various information together with the session key and the first message authentication code using the private key of the sending pager unit, and wherein the pager proxy server further comprises means for decrypting the various header

data, first message authentication code, and session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

8. A system as claimed in claim 7, wherein said message authentication code is an error correction code function.

9. A system as claimed in claim 7, wherein said various header data includes at least a user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

10. A system as claimed in claim 9, wherein said various header data further includes a message number and application header.

11. A system as claimed in claim 4, wherein the sending pager further comprises means for adding an encryption method identifier in clear text to the packet header.

12. A system as claimed in claim 4, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

13. A system as claimed in claim 4, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

14. A system as claimed in claim 2, wherein said sending pager unit further comprises means for generating an address mode and appending the address mode in clear text to the packet header.

15. A system as claimed in claim 14, wherein said address mode indicates an address type selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network gateway server and includes means for re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

16. A system as claimed in claim 15, further comprising means for receiving e-mail packets from said computer network gateway server, and re-packaging said e-mail packets for transmission to the destination pager unit via said network operation center, and means for repackaging packets received from the network operations center for forwarding to an e-mail server.

17. A system as claimed in claim 1, wherein said means included in the pager proxy server for decrypting at least a portion of the packet includes means for decrypting, using a secret key, a portion of the packet containing a first session key used by a sending pager unit to encrypt said portion of the packet.

18. A system as claimed in claim 17, wherein said pager proxy server further includes means for decrypting said message using said first session key, means for generating a second session key, and means for re-encrypting the message using the second session key.

19. A system as claimed in claim 18, wherein said means for re-encrypting said portion of the packet includes means for encrypting the second session key by a secret key.

20. A system as claimed in claim 19, wherein said means for encrypting said portion of the packet by a secret key includes means for re-encrypting the second session key by a public key corresponding to a private key of a destination pager unit.

21. A system as claimed in claim 20, wherein said means for encrypting said portion of the packet by a secret key further includes means for, before re-encrypting the second session key by the public key corresponding to a private key of the destination pager, encrypting the second session key and various additional data by a private key of the pager proxy server.

22. A system as claimed in claim 21, wherein said additional data includes a second user identification number, said second user identification number corresponding to a first destination header included in said decrypted portion of the packet received from the sending pager unit, and wherein said destination paging unit includes means for comparing said second user identification number encrypted with said second session key to a clear text version of the second user identification number received from the pager proxy server in order to authenticate the pager proxy server.

23. A system as claimed in claim 22, wherein said additional data includes a second destination header corresponding to the first user identification number, and wherein said second pager unit includes means for displaying information included in said second destination header in order to indicate an address of the sending pager unit.

24. A system as claimed in claim 22, wherein said additional data includes a second destination header corresponding to the first user identification number, a message number recovered from said decrypted portion of the packet received from the sending pager unit, and an application number.

25. A system as claimed in claim 22, wherein said pager proxy server further comprises means for generating a message authentication code based on said message, said second session key, and said additional data, and said destination pager unit includes means for recovering said additional data and computing a message authentication code based on the additional data, said second session key, and said message in order to authenticate said message.

26. An encryption method according to which encryption services may be added to an existing two-way wireless pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, comprising the steps of:

causing one of said pager units to perform the steps of encrypting a message, including the encrypted message in a wireless transmission packet, and transmitting the encrypted message from said one of said pager units to a pager proxy server via the network operations center;

causing the pager proxy server to perform the steps of receiving the encrypted message and repackaging it for transmission to another of said pager units via the network operations center; and

causing said another of said pager units to perform the steps of decrypting and displaying the encrypted message.

27. A method as claimed in claim 26, wherein the step of encrypting the message comprises the step of encrypting the message by a secret key corresponding to a secret key of the pager proxy server so that the session key can only be recovered by the paging proxy server.

28. A method as claimed in claim 26, wherein said secret key is a first session key generated by a sending pager unit, and wherein said sending pager unit further performs the step of encrypting said first session key by a public key corresponding to a private key held by the pager proxy server.

29. A method as claimed in claim 27, wherein said sending pager unit further performs the step of encrypting at least the first session key by a private key of the sending pager unit, and wherein said pager proxy server performs the step of retrieving a public key corresponding to the private key of the sending pager unit for use as a first

level authentication of the sending pager unit.

30. A method as claimed in claim 29, further comprising of the step of appending a unique user identification number of the sending pager unit to the header of the transmission to the paging proxy server in clear text form, said user identification number being hard-coded into the sending pager unit.

31. A method as claimed in claim 30, wherein said step of encrypting at least the session key by a private key of the sending pager unit includes the step of encrypting the user identification number of the sending pager unit, and said paging proxy server further performs the steps of decrypting the encrypted user identification number together with the first session key and comparing it with the clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and first session key.

32. A method as claimed in claim 29, wherein the sending pager unit further performs the step of computing a first message authentication code based on various header data and the message and encrypting the various information together with the session key and the first message authentication code using the private key of the sending pager unit, and wherein the pager proxy server further performs the steps of decrypting the various header data, first message authentication code, and session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

33. A method as claimed in claim 32, wherein said message authentication code is an error correction code function.

34. A method as claimed in claim 32, wherein said various header data includes at least the user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

35. A method as claimed in claim 34, wherein said various header data further includes a message number and application header.

36. A method as claimed in claim 34, wherein the sending pager further performs the step of adding an encryption method identifier in clear text to the packet header.

37. A method as claimed in claim 29, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

38. A method as claimed in claim 27, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

39. A method as claimed in claim 37, wherein said sending pager unit further performs the step of generating an address mode and appending the address mode in clear text to the packet header.

40. A method as claimed in claim 39, wherein said address mode indicates an address type selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network gateway server and further performs the step of re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

41. A method as claimed in claim 40, further performs the steps of receiving e-mail packets from said computer network gateway server, and re-packaging said e-mail

packets for transmission to the destination pager unit via said network operation center.

42. A method as claimed in claim 26, wherein said step of repackaging the encrypted message for transmission includes the step of causing the pager proxy server to encrypt, using a secret key, a portion of the packet containing a first session key used by a sending pager unit to encrypt said portion of the packet.

43. A method as claimed in claim 42, wherein said pager proxy server further performs the steps of decrypting said message using said first session key, generating a second session key, and re-encrypting the message using the second session key.

44. A method as claimed in claim 43, wherein said pager proxy server further performs the step of encrypting the second session key by a secret key.

45. A method as claimed in claim 44, wherein said step of encrypting said portion of the packet by a secret key includes the step of re-encrypting the second session key by a public key corresponding to a private key of a destination pager unit.

46. A method as claimed in claim 45, wherein said step of encrypting said portion of the packet by a secret key further includes the step of, before re-encrypting the second session key by the public key corresponding to a private key of the destination pager, encrypting the second session key and various additional data by a private key of the pager proxy server.

47. A method as claimed in claim 46, wherein said additional data includes a second user identification number, said second user identification number corresponding to a first destination header included in said decrypted portion of the packet received from the sending pager unit, and wherein said destination paging unit perform the step of comparing said second user identification number encrypted with said second session key

to a clear text version of the second user identification number received from the pager proxy server in order to authenticate the pager proxy server.

48. A method as claimed in claim 47, wherein said additional data includes a second destination header corresponding to the first user identification number, and wherein said second pager unit performs the step of displaying information included in said second destination header in order to indicate an address of the sending pager unit.

49. A method as claimed in claim 47, wherein said additional data includes a second destination header corresponding to the first user identification number, a message number recovered from said decrypted portion of the packet received from the sending pager unit, and an application number.

50. A method as claimed in claim 47, wherein said pager proxy server further performs the step of computing a message authentication code based on said message, said second session key, and said additional data, and said destination pager unit further performs the step of recovering said additional data and computing a message authentication code based on the additional data, said second session key, and said message in order to authenticate said message.

51. A two-way alphanumeric pager unit, comprising:

means for inputting a message and a destination address;

means for generating a session key;

means for encrypting the message using the session key;

means for protecting the session key so that it can only be recovered by a pager proxy server;

means for transmitting the message via a wireless pager network to the pager proxy server;

means for receiving an encrypted message transmitted via the wireless pager network from the pager proxy server;

means for decrypting an encrypted session key appended to the message;

means for decrypting the encrypted message transmitted from the pager proxy server using the decrypted session key; and

means for displaying the message.

52. A pager unit as claimed in claim 51, wherein said means for protecting the session key comprises means for encrypting the session key by a secret key.

53. A pager unit as claimed in claim 52, wherein said secret key is a first session key generated by the pager unit, said sending pager unit further comprising means for encrypting said first session key by a public key corresponding to a private key held by the pager proxy server.

54. A pager unit as claimed in claim 53, further comprising means for appending a unique user identification number of the pager unit to the header in clear text form, said user identification number being hard-coded into the pager unit.

55. A pager unit as claimed in claim 54, wherein said means for encrypting at least the session key by a secret key also encrypts the user identification number of the sending pager unit, said encrypted user identification number being compared by the pager proxy server with a clear text version of the user identification number transmitted with a packet header in order to authenticate the pager unit.

56. A pager unit as claimed in claim 55, wherein the pager unit further comprises means for computing a message authentication code based on various header data and the message, and means for encrypting the various information together with the session key and the message authentication code using a private key of the sending pager unit in order to provide a means for authentication by the pager proxy of the message.

57. A pager unit as claimed in claim 56, wherein said message authentication code is an error correction code function.

58. A pager unit as claimed in claim 57, wherein said various header data includes at least the user identification number of the pager unit and a destination header corresponding to the input address of a destination pager.

59. A pager unit as claimed in claim 58, wherein said various header data further includes a message number and application header.

60. A pager unit as claimed in claim 52, wherein the pager unit further comprises means for adding an encryption method identifier in clear text to a packet header.

61. A pager unit as claimed in claim 60, wherein an encryption algorithm used to encrypt the first session key is a public-private key encryption algorithm.

62. A pager unit as claimed in claim 60, wherein said secret key is a first session key generated by a sending pager unit and said first session key is encrypted by a stream cipher that uses a shared secret key.

63. A pager unit as claimed in claim 62, wherein said pager unit further comprises means for generating an address mode and appending the address mode in clear text to the packet header.

64. A pager unit as claimed in claim 62, wherein said address mode is selected from the group consisting of pager address types and e-mail address types, and wherein the pager proxy server is connected to a computer network server and includes means for re-packaging said message in an e-mail packet and transmitting the e-mail packet via said computer network server to an e-mail address.

65. A pager proxy server, comprising:

means for receiving a message encrypted by a session key, the session key being encrypted and appended to the encrypted message, from a network operations center of a pager network;

means for recovering the session key using a secret key of the server;

means for authenticating the sender of the message; and

means for re-transmitting the message encrypted by a session key in a manner which enables decryption of the message only by a holder of a second secret key.

66. A server as claimed in claim 65, wherein said means for re-transmitting the message comprises means for decrypting the message using the first session key, re-encrypting the message using a second session key, and encrypting the second session key.

67. A server as claimed in claim 66, wherein said first secret key is a private key held by the pager proxy server.

68. A server as claimed in claim 67, further comprising means for retrieving a public key corresponding to a private key of a sending pager unit for use as a first level authentication of the sending pager unit.

69. A server as claimed in claim 68, further comprising means for decrypting the a user identification number of the sending pager unit together with the session key and comparing it with a clear text user identification number in order to authenticate the contents of the field containing the encrypted user identification number and session key.

70. A server as claimed in claim 69, further comprising means for decrypting various header data, a first message authentication code, and a session key using a public key corresponding to the private key of the sending pager unit, decrypting the message using

the session key, generating a second message authentication code based on the message and various header data, and comparing the first message authentication code with the second message authentication code in order to authenticate the message.

71. A server as claimed in claim 70, wherein said message authentication code is an error correction code function.

72. A server as claimed in claim 70, wherein said various header data includes at least the user identification number of the sending pager and a destination header corresponding to the input address of the destination pager.

73. A server as claimed in claim 72, wherein said various header data further includes a message number and application header.

74. A server as claimed in claim 73, wherein said encryption method is a public-private key encryption algorithm.

75. A server as claimed in claim 73, wherein said encryption method is RC4 secret key encryption.

76. A server as claimed in claim 72, further comprising means for receiving e-mail packets from said computer network server, and re-packaging said e-mail packets for transmission to the destination pager unit via said network operation center.

77. A system for adding encryption services to an existing pager network, the pager network including a network operations center which provides a means for receiving an alphanumeric message from any of a plurality of handheld pager units and forwarding the alphanumeric message to another of the plurality of handheld pager units, at least one of said pager units comprising:

means for inputting an alphanumeric message and a destination address;

means for including the alphanumeric message in a packet for transmission to the destination address by wireless transmission via the network operations center;

means for receiving an alphanumeric message from the network operations center; and

means for displaying the alphanumeric message received from the network operations center,

wherein the system for adding encryption services comprises:

means in at least one of said pager units for decrypting and displaying an encrypted message; and

a pager proxy server including means for receiving a packet containing the encrypted message, decrypting at least a portion of the packet, and re-encrypting said portion of the packet for delivery to said at least one of said pager units via said network operations center.

78. An alphanumeric pager unit, comprising:

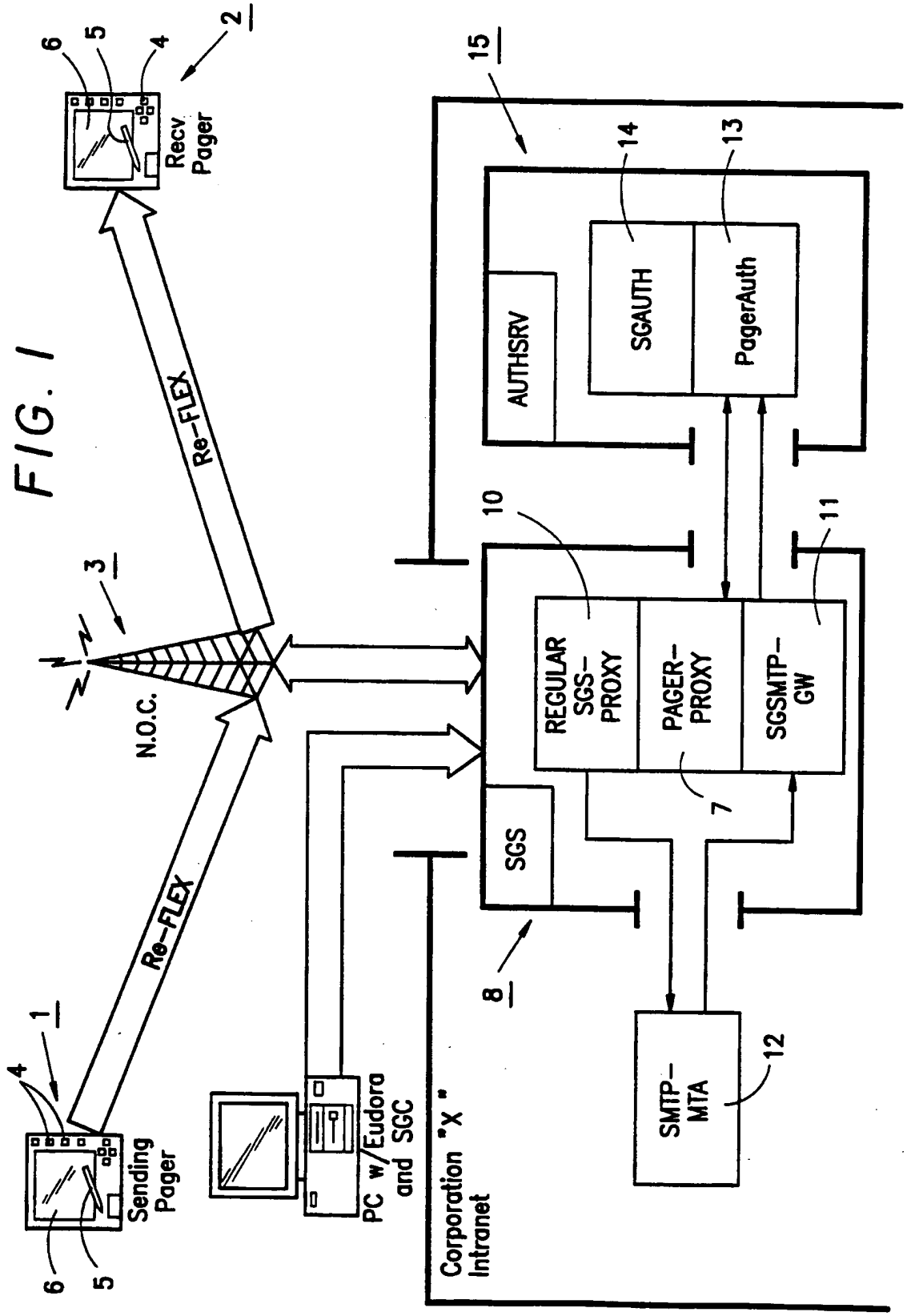
means for receiving an encrypted message transmitted via a wireless pager network from a pager proxy server;

means for decrypting an encrypted session key appended to the message;

means for decrypting the encrypted message transmitted from the pager proxy server using the decrypted session key; and

means for displaying the message.

FIG. 1



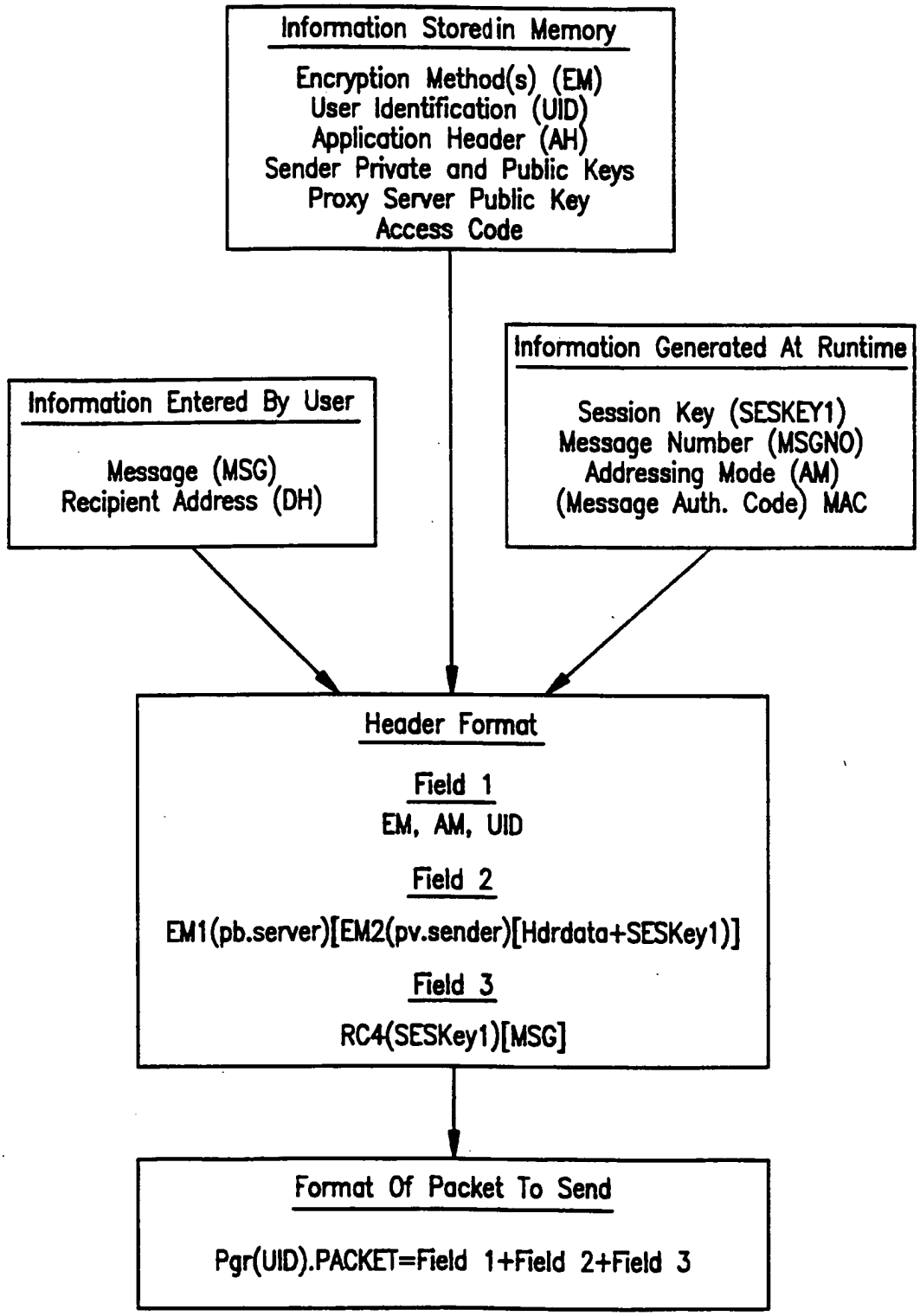
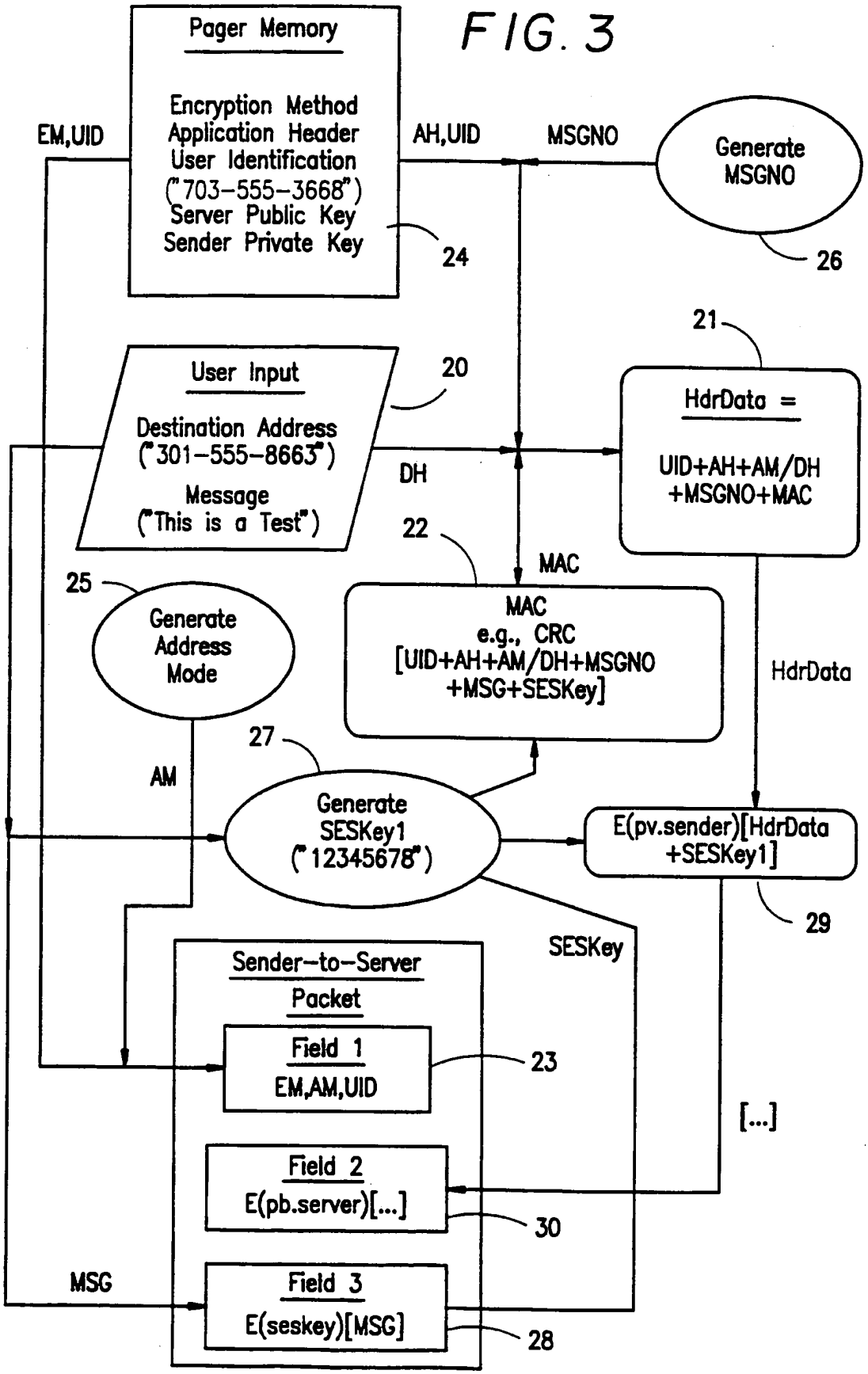


FIG. 2

FIG. 3



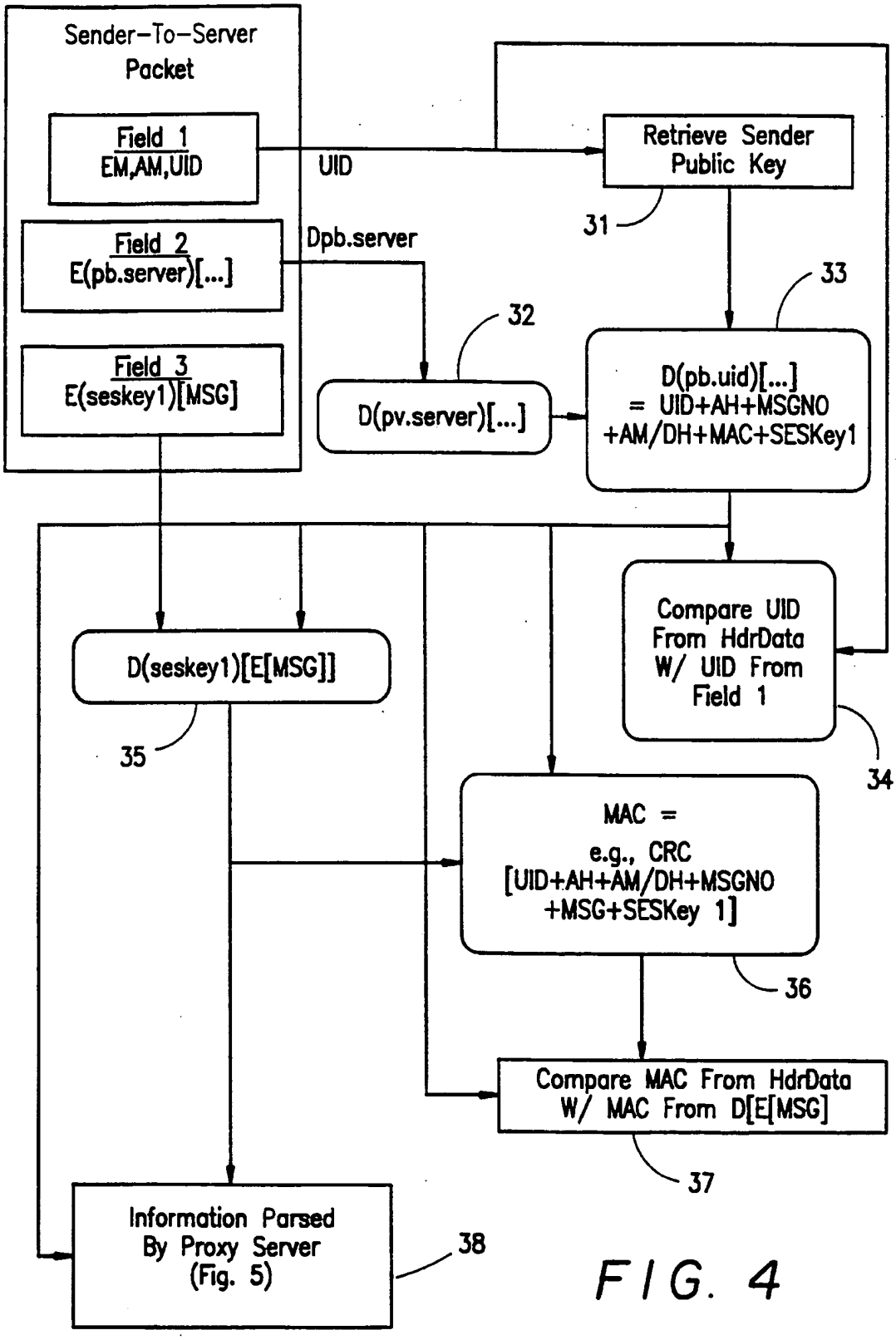


FIG. 4

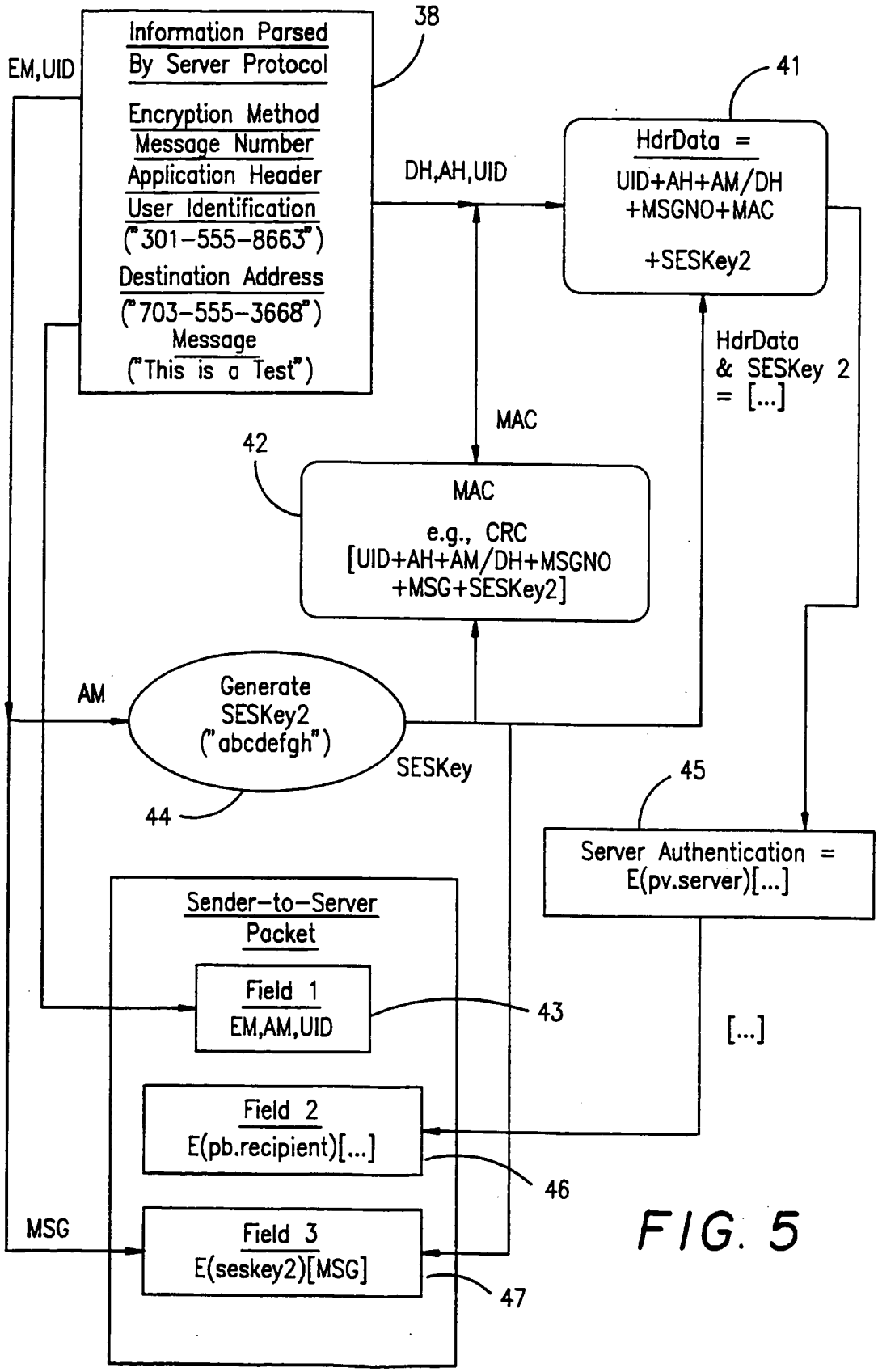


FIG. 5

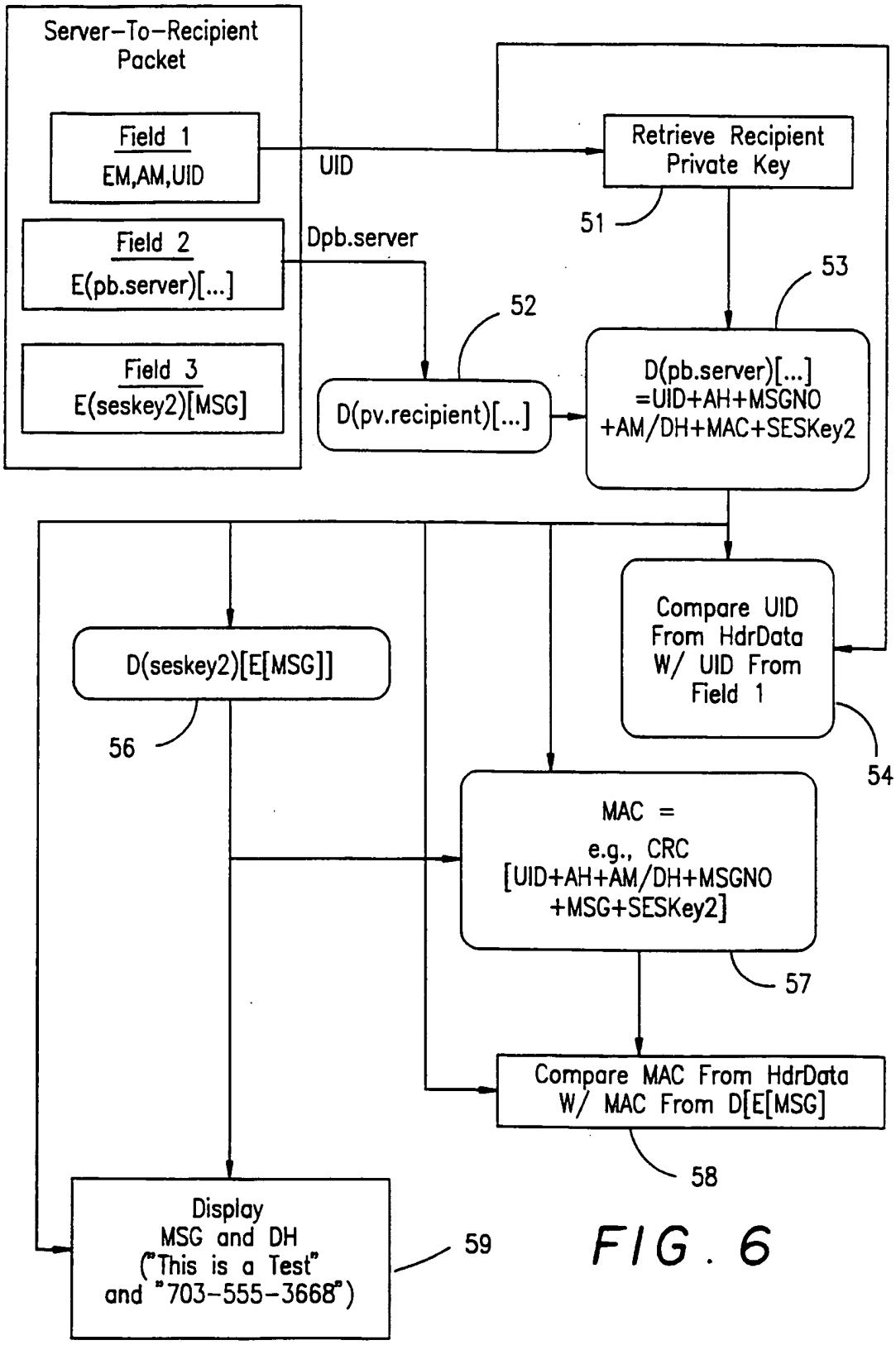


FIG. 6

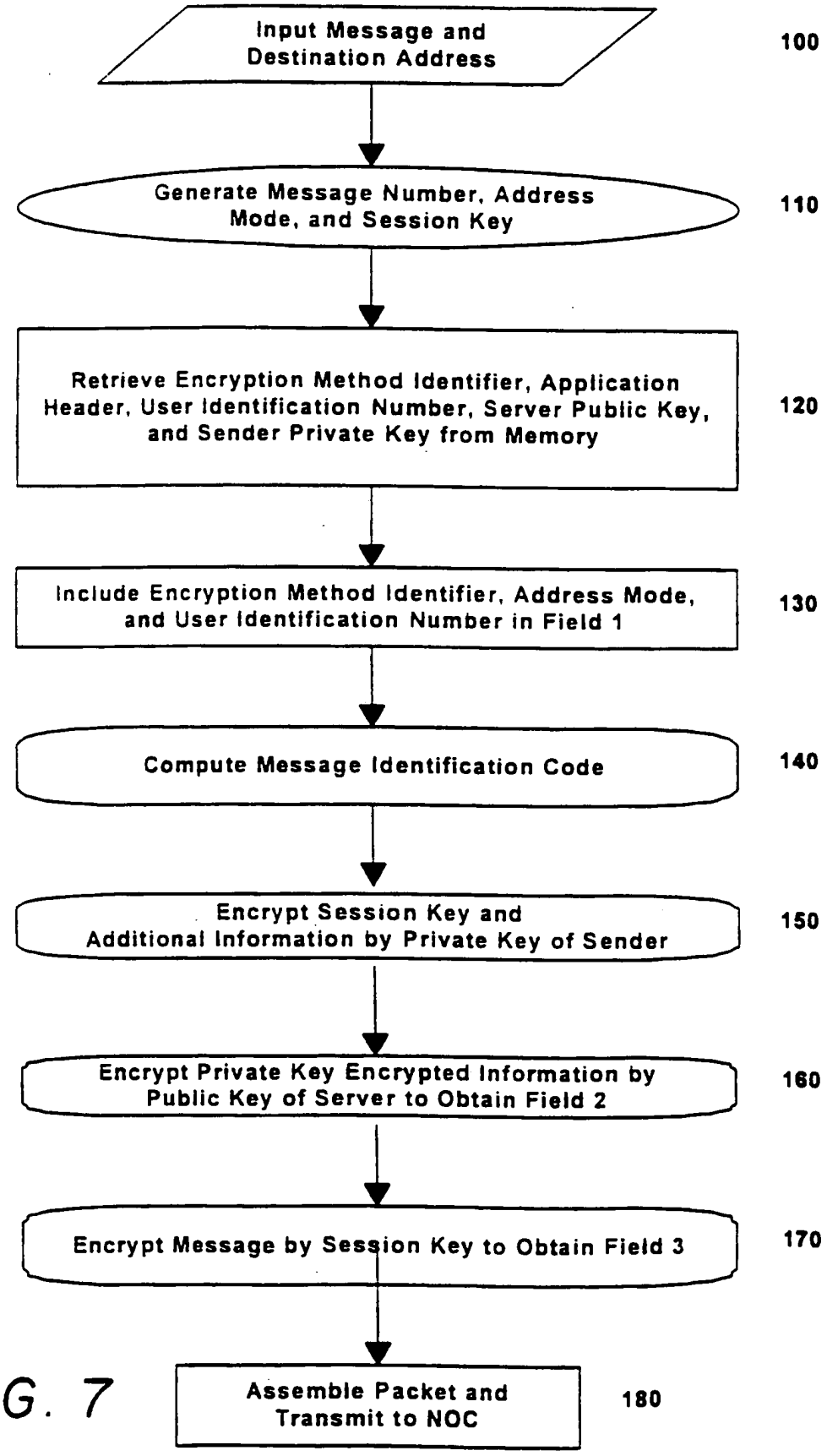
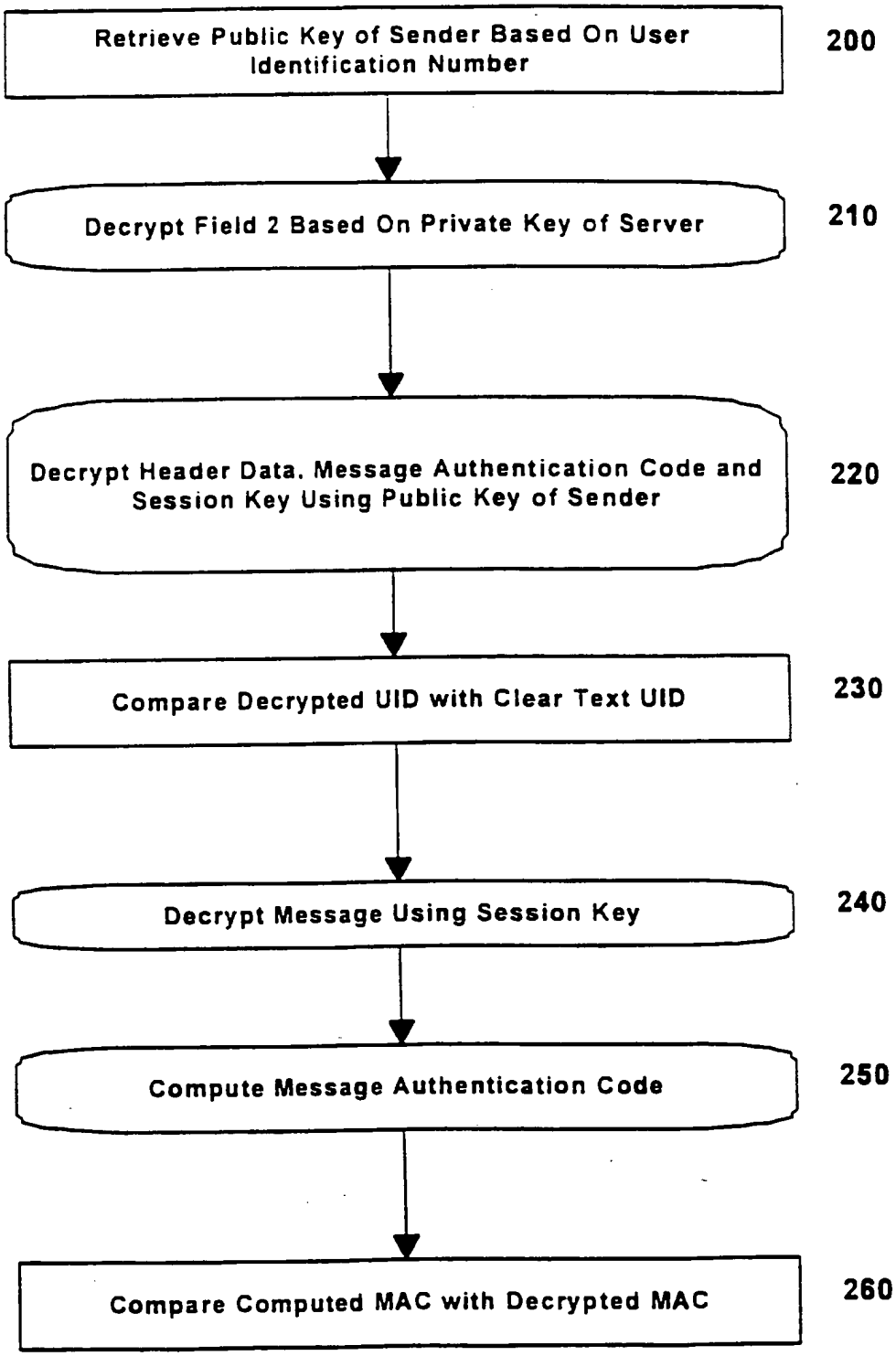


FIG. 7

FIG. 8



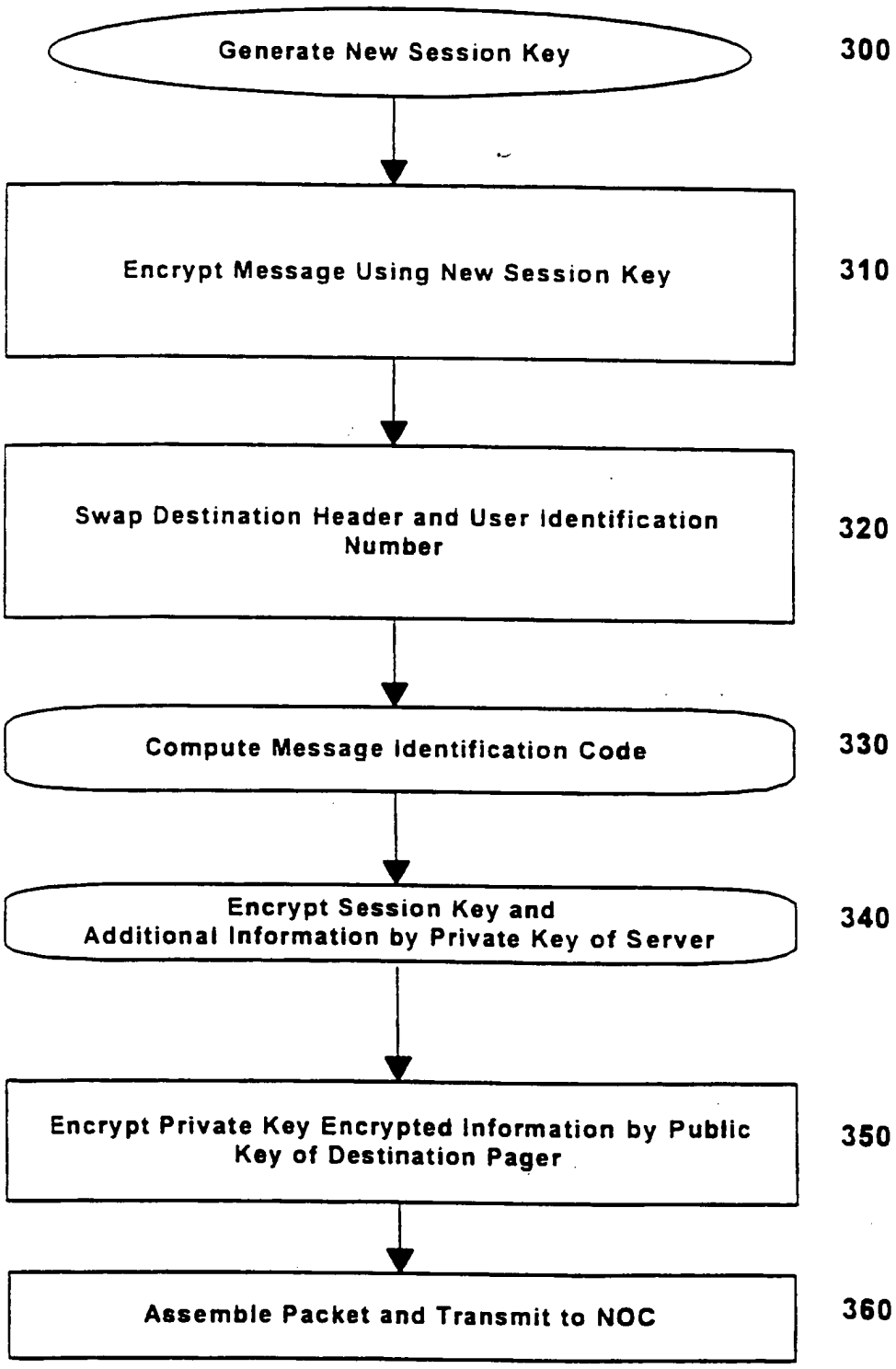
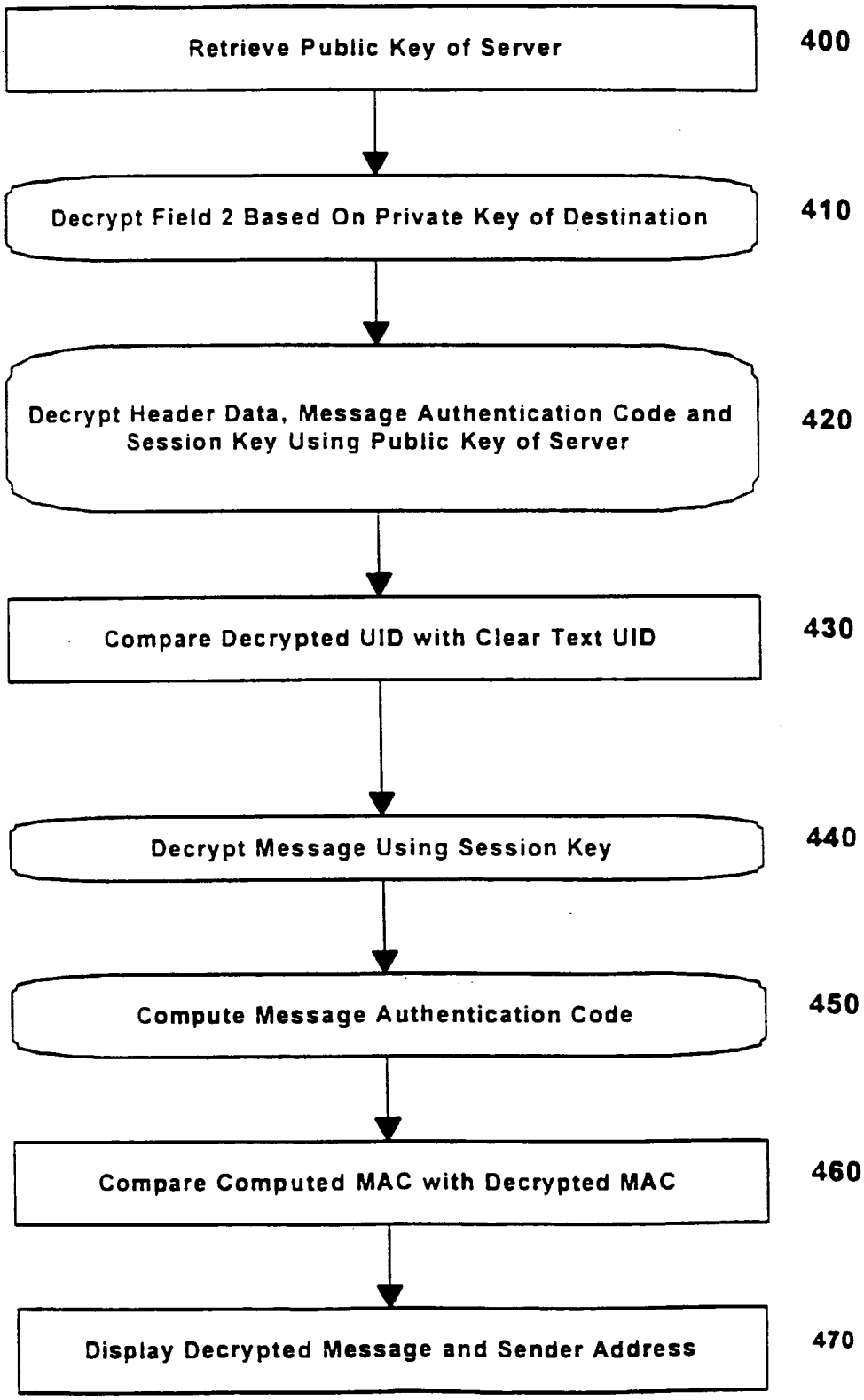


FIG. 9

FIG. 10



INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :H04L 9/08
US CL :380/21
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. : 380/21,44,45,49

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)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,285,496 A (FRANK et al) 08 February 1994 (08.02.94), column 2, lines 28-44, column 4, lines 12-68, column 6, lines 11-49.	1-31,36-56, 58,60-68,74-78
Y	US 5,604,801 A (DOLAN et al) 18 February 1997 (18.02.97), abstract, column 3, lines 2-38, 50-60, column 4, lines 19-24, 40-55.	1-31,36-56 58,60-68, 74-78
A	US 5,602,918 A (CHEN et al) 11 February 1997 (11.02.97), abstract, column 2, lines 36-41,57-60, column 4, lines 43-63.	3-4,6,17-18,27-28,31-32,40,42,53,65,68-70,77
A	US 5,452,356 A (ALBERT et al) 19 September 1995 (19.09.95), column 1, lines 60-68, column 2, lines 1-42, column 11, lines 15-55.	1-78

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance	"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
"E"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search 17 FEBRUARY 1999	Date of mailing of the international search report 06 MAY 1999
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 GAIL HAYES Telephone No. (703) 305-9711

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,495,533 A (LINEHAN et al) 27 February 1996 (27.02.96), column 9, lines 42-58, column 10, lines 22-32.	7,9,35,59,69-70,72-73

Form PCT/ISA/210 (continuation of second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/27531

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS
search terms: cypher, cipher, encode,encrypt, decrypt, key, keys, pager, wireless, proxy server, authenticate, authentication, transmission, transmitting, key management, public key, two-way communication, re-encrypt, messages, data, information

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L — A2

(11) International Publication Number: **WO 97/41661**

(43) International Publication Date: 6 November 1997 (06.11.97)

(21) International Application Number: PCT/US97/06161

(22) International Filing Date: 16 April 1997 (16.04.97)

(30) Priority Data:
08/639,457 29 April 1996 (29.04.96) US

(71) Applicant: MOTOROLA INC. [US/US]; 1303 East Algonquin Road, Schaumburg, IL 60196 (US).

(72) Inventor: DORENBOS, David; 241 N. Larch, Elmhurst, IL 60126 (US).

(74) Agents: LUKASIK, Susan, L. et al.; Motorola Inc., Intellectual Property Dept., 1303 East Algonquin Road, Schaumburg, IL 60196 (US).

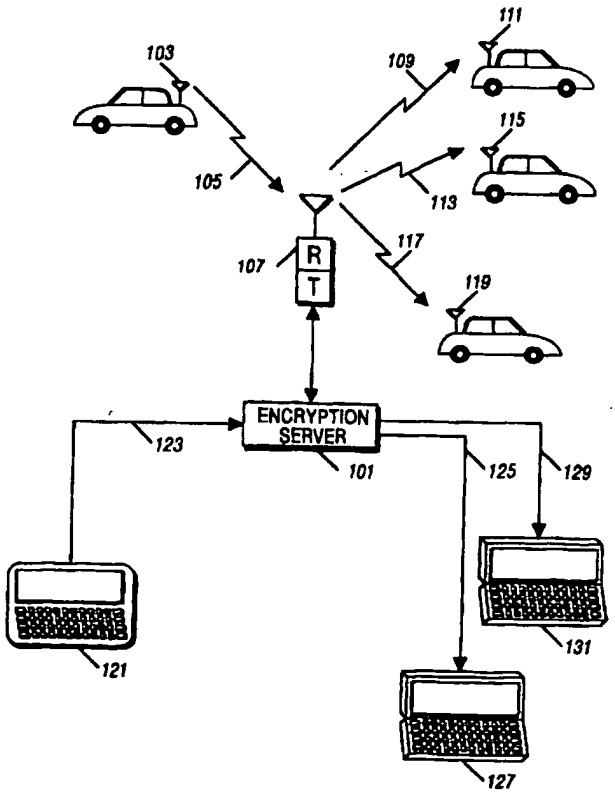
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: USE OF AN ENCRYPTION SERVER FOR ENCRYPTING MESSAGES

(57) Abstract

An encryption server receives a first encrypted message (105) and decrypts (403) the encrypted message using a first key, yielding a decrypted message comprising a second encrypted message (105A), an identification of a sender of the first encrypted message, and an identification of a first recipient. The second encrypted message, the identification of the sender, and the identification of the first recipient are determined (405) from the decrypted message. The second encrypted message and the identification of the sender are encrypted (409) with a second key, yielding a third encrypted message (109). The third encrypted message (109) is transmitted to the first recipient.



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.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

USE OF AN ENCRYPTION SERVER FOR ENCRYPTING MESSAGES

Field of the Invention

This invention relates to communication systems, including but not limited to encrypted communication systems.

Background of the Invention

Encrypted voice and data communication systems are well known. Many of these systems provide secure communications between two or more users by sharing one or more pieces of information between the users, which permits only those users knowing that information to properly decrypt the message. This information is known as the encryption key, or key for short. Encryption keys may be private keys, where a single key is utilized for encryption and decryption, or public keys, where multiple keys are utilized for encryption and decryption.

Methods of encrypting using public-key encryption are well known in the art. Typically, a public-key encryption is a method of encryption by which a single message is encrypted using a sender's private key and then a recipient's public key. The recipient then decrypts the message using the recipient's private key and then the sender's public key. Typically, public keys are 512 bits long, although some public keys have as few as 256 bits. Some encryption experts recommend using 1024-bit keys. Because the computational power required to break a key increases exponentially with the length of the key, longer keys provide more security. In addition, because two keys are needed to decrypt a message, two longer keys are more difficult to decrypt if neither key is known.

Today, secure communication systems are used to transmit data in an encrypted fashion. If a user wishes to send the same message to five different recipients, the user must encrypt the message five different times, each time using the public key of a different recipient for the message. The user then transmits the five messages to the five recipients. Such a process, however, is troublesome when the user wishes to transmit to, for example, 100 or more recipients. In this instance, the user must encrypt each message individually 100 or more times, one for each recipient. If the user has a portable communication device, such as a laptop computer, the user's battery may run out of power before encryption and transmission of each message has occurred. In addition, the encryption and transmission process can consume a lot of time and processing power for the portable device, rendering the portable device unavailable for other activities by the user during the encryption and transmission time period. Thus, such transmissions would be impractical for portable users.

Accordingly, there is a need for a method of transmitting encrypted data messages to multiple users without resulting in a time or power barrier to the user's communication device.

Brief Description of the Drawings

FIG. 1 is a block diagram of a communication system having an encryption server in accordance with the invention.

FIG. 2 is a block diagram of an encryption server in accordance with the invention.

FIG. 3 is a flowchart showing a method of transmission of a digital data message to an encryption server in accordance with the invention.

FIG. 4 is a flowchart showing a method of transmission of an encrypted message by an encryption server in accordance with the invention.

Description of a Preferred Embodiment

The following describes an apparatus for and method of using an encryption server for encrypting messages. Messages are encrypted twice, once with the sender's private key and then with an encryption server's public key before transmission of the messages to the encryption server. The encryption server decrypts received messages with the encryption server's private key, yielding an encrypted message, a user identification (ID), and one or more recipient IDs. The encryption server encrypts the encrypted message and the user ID individually with each of the recipient's public keys and transmits the resultant message(s) to the appropriate recipient. Each recipient decrypts the messages using the recipient's private key and the sender's public key. A secure communication system is thereby provided, wherein portable communication devices are neither tied up nor drained of power because the device's user wishes to send a single encrypted message to multiple recipients.

A method of using an encryption server for encrypting messages comprises the steps of, at a communication unit operated by a user generating a digital data message. The digital data message is encrypted using a first key, yielding a first encrypted message. An identification of the user and an identification of a first recipient are appended to the first encrypted message, yielding an appended first encrypted message. The appended first encrypted message is encrypted using a second key, yielding a second encrypted message. The second encrypted message is transmitted to an encryption server. At the encryption server, the second encrypted message is received. The second encrypted message is decrypted using a third key, yielding the appended first encrypted message. The first encrypted message, the identification of the user, and the identification of the first recipient are determined from the appended first encrypted message. The first encrypted message and the identification of the user are encrypted with a fourth key, yielding a third encrypted message. The third encrypted message is transmitted to the first recipient. In the preferred embodiment, the first key is a private key associated with the user, the second key is a public key associated with the encryption server, the third key is a private key associated with the encryption server, and the fourth key

is a public key associated with the first recipient. Alternatively, the second key and the third key may be identical. The transmitting steps may be performed over wireless communication resources, such as radio frequency communication resources, or wireline communication resources, such as standard telephone lines or fiber optic cable.

In addition, the step of appending may further comprise the step of appending an identification of a second recipient to the first encrypted message, thereby yielding the appended first encrypted message. In this case, the method further comprises the steps of encrypting, by the encryption server, the first encrypted message and the identification of the user with a fifth key, yielding a fourth encrypted message, and transmitting the fourth encrypted message to the second recipient. In the preferred embodiment, the fifth key is a public key associated with the second recipient. Alternatively, the step of appending may comprise the step of appending three or more identifications of recipients to the first encrypted message, thereby yielding the appended first encrypted message.

A block diagram of a communication system having an encryption server is shown in FIG. 1. An encryption server 101 is shown at the center of FIG. 1. Further details of the encryption server 101 are shown in FIG. 2 described below. A user of a first communication unit 103 utilizes the first communication unit 103 to generate an digital data message that is encrypted in two stages in the preferred embodiment. In the first stage, the digital data message is encrypted using a first key, which is the user's private key in the preferred embodiment. The result of this encryption is a first-stage encrypted message. (In an alternate embodiment, the digital data message is not encrypted using the first key.) The user's identification (ID) and one or more recipient IDs are appended to the first-stage encrypted message, yielding an appended message. The appended message is encrypted using a second key, yielding a second-stage encrypted message 105. In the preferred embodiment, the second key is the public key associated with the encryption server 101. The communication unit transmits the second-stage encrypted message 105 to the encryption server via a wireless communication link to a wireless communication device 107, such as a radio frequency (RF) base station, repeater, or radio, or infrared

communication device. The second-stage encrypted message 105 is conveyed by the wireless communication device 107 to the encryption server 101.

The encryption server 101 decrypts the second-stage encrypted message 105 using an appropriate key. In the preferred embodiment, the appropriate key is the encryption server's private key. The encryption server 101 then determines the user's ID from the decrypted message and also determines the IDs of all recipients that the user indicated as intended targets of the first-stage encrypted message. The encryption server 101 then encrypts the user's ID along with the first-stage encrypted message by encrypting with the public key of the first recipient. The resultant message 109 is transmitted to the first recipient, who utilizes communication unit 111. The encryption server then encrypts the first-stage encrypted message along with the user's ID by encrypting with the public key of the second recipient and transmitting the resultant encrypted message 113 to the second recipient, who utilizes communication unit 115. This process continues until the encryption server reaches the last recipient ID on the user's list, and encrypts the first-stage encrypted message along with the user's ID by encrypting with the public key of the last recipient and transmitting the resultant encrypted message 117 to the last recipient, who utilizes communication unit 119.

The encryption server 101 may also receive user requests for encryption from wireline communication devices 121 via wireline channels. As with the wireless transmission, the encryption server decrypts the received message 123 using the private key of the encryption server, then encrypts the resultant message individually for each different recipient using the appropriate recipient's individual public key. These recipients may be wireline devices 127 and 131, which receive the messages 125 and 129 via wireline communication channels.

The above examples describe RF to RF transmission and wireline to wireline transmission of encrypted messages. Nevertheless, the method of the present invention is equally successful if a wireline device 121 requests transmission to wireless communication units 111, 115, and 119. Similarly,

a wireless communication unit 103 may request transmission from the encryption server 101 to wireline communication devices 127 and 131. In addition, the recipients may be a combination of both wireless and wireline communication units 111, 115, 119, 127, and 131, regardless of whether the sender uses a wireless communication unit 103 or a wireline communication device 121.

Upon receipt of the encrypted message from the encryption server, each recipient decrypts the message with the recipient's own private key, and after determining the user's ID, decrypts the resultant message with the user's public key, thereby yielding the original digital data message. The user is also referred to as the sender of the (second-stage) encrypted message 105.

A block diagram of an encryption server 101, including its input signals 105 and output signals 109, 113, 125, and 117, is shown in FIG. 2. In the preferred embodiment, the encryption server 101 is a Sun SparcServer2000 in a multiprocessor configuration, available from Sun Microsystems. The encryption server 101 comprises one or more processors 201, such as microprocessors or digital signal processors, as are well known in the art. The processors 201 have access to encryption and decryption algorithm(s) 203, a public key data base 205, and memory 211. The encryption/decryption algorithms 203 include public key algorithms, private algorithms, and other algorithms as may be used in the art. The public key data base 205 includes a list of IDs, as used by senders (users) and recipients, and the public keys associated with each of these IDs. The memory 211 includes programming and other data as is necessary to provide functionality as described herein for the encryption server 101. A receive block for wireline and wireless communications 207 and a transmit block for wireline and wireless communications 209 are also connected to the processors 201. The receive block for wireline and wireless communications 207 performs appropriate demodulation techniques on received messages 105 and 123. The transmit block for wireline and wireless communications 209 performs appropriate modulation techniques on messages 109, 113, 124, and 117 to be transmitted. In addition, the encryption server 101 may be equipped with hardware

and/or software to provide the encryption server 101 with over-the-air-rekeying capabilities.

As shown in FIG. 2, a user message 105 comprises a second-stage encrypted (encrypted using the encryption server's public key) message comprising the digital data message 105A, first-stage encrypted with the user's (sender's) private key, in addition to the user ID and a number of recipient IDs. Alternatively, the user message 105 may comprise an unencrypted digital data message 105A, the user ID, and one or more recipient IDs. The user message 105 is input to the receive wireline/wireless block 207, the output of which is input to the processor(s) 201. The processor(s) 201 utilize(s) the encryption/decryption algorithm(s) 203 and the public key data base 205 to decrypt the message 105 using the private key of the encryption server. The processor(s) 201 then determine(s) the first-stage encrypted message 105A, the user ID, and the first recipient ID from the decrypted message. The processor(s) 201 then determine(s) the first recipient's public key from public key data base 205, and the encrypt the first-stage encrypted message 105A and the user ID by using the encryption/decryption algorithms 203 and the first recipient's public key. The processor(s) 201 then append(s) the first recipient ID, thereby yielding a message 109 that is sent to the transmit wireline/wireless block 209 for transmitting to the first recipient's communication unit 111, as shown in FIG. 1. A similar process is performed on the first-stage encrypted message (or unencrypted digital data message) 105A and the user ID for each of the recipients listed in the user's message 105.

In an alternate embodiment, the encryption server 101 may be physically distributed as one or more encryption servers. In this embodiment, the encryption server 101 encrypts the message using a second set of private and public keys associated with a second server. The message so encrypted is transmitted to the second encryption server. The second server decrypts the message and then encrypts the message using the public key(s) of the recipient(s). When traffic is heavy, the encryption server 101 may optimize its efficiency by determining the computation required to transmit directly to each recipient or transmit the request to one or more distributed servers. This process is transparent to the user.

The flowchart of FIG. 3 shows a method for use by a communication unit in transmitting a digital data message to an encryption server 101. At step 301, a digital data message is generated. If at step 303 the digital data message is not to be encrypted, the process continues with step 307. If at step 303 the digital data message is to be encrypted, the process continues with step 305, where the digital data message is encrypted using the private-key of the user who wishes to communicate the message. At step 307, it is determined if the IDs of the user and/or recipient(s) are to be encrypted. If the IDs are to be encrypted, the process continues with step 309, where the user ID and recipient ID(s) are appended to the encrypted message from step 305 or the unencrypted message from step 301 if no encryption took place. At step 311, the message from step 309, including the appended IDs, is encrypted using the public key of the encryption server 101. The process continues with step 317, where the encrypted message is transmitted to the encryption server 101. If at step 307 the IDs are not to be encrypted, the process continues with step 313, where the encrypted message of step 305 (or the unencrypted message from step 301 if no encryption took place) is encrypted with the public key of the encryption server 101. At step 315, the user ID and recipient ID(s) are appended to the encrypted message of step 313, and the process continues with step 317.

In an alternative embodiment, i.e., when the digital data message is not to be encrypted at step 303 of FIG. 3, the sender or user may decrypt the digital data message and, if desired, the recipient IDs only once, using the encryption server's public key. The encryption server then decrypts the message using the encryption server's private key, and encrypts the message individually for each of the recipients with the recipient's public key. The recipient then decrypts the message using only the recipient's private key. This method requires the user to locally store only one public key, the key of the encryption server. With this method, a single symmetrical key may be used to encrypt and decrypt the messages between the user and the encryption server 101, and one or more keys may be used to encrypt the messages between the encryption server and the recipient. Nevertheless, for better security, the encryption server 101 engaged in this embodiment should be a physically secured, e.g., locked away with limited

access, because unencrypted information is present inside the encryption server 101. An advantage of such a system includes enabling law enforcement officials the ability to read the decrypted message as available in the encryption server 101.

The flowchart of FIG. 4 shows the method performed by the encryption server 101 in accordance with the present invention. At step 401, the encryption server receives the encrypted message transmitted by the communication unit 103. At step 403, the encryption server decrypts the message received at step 401 with the private key of the encryption server 101. At step 405, the encryption server determines the user ID, the recipient ID(s), and the encrypted (generated at step 305 of FIG. 3) or unencrypted (generated at step 301 of FIG. 3) data message. In an alternate embodiment, the encryption server 101 may be equipped with the appropriate keys to decrypt the digital data message 105A (when the message 105A is encrypted) so that law enforcement agencies may have full access to all information transmitted in the system.

At step 407, it is determined if the IDs (i.e., the user ID and/or recipient ID(s)) are to be encrypted before transmission. If the IDs are to be encrypted, the process continues with step 409, where the encryption server encrypts the encrypted data message along with the user ID, and the recipient's ID if desired, with the recipient's public key. At step 411, the encryption server transmits the encrypted message to the recipient whose public key was used at step 409. If at step 413 there are more recipients identified by the user to which the encryption server has not yet encrypted and transmitted the message, the process continues with step 407. If there are no more recipients at step 413, the process ends. If at step 407, the IDs are not to be encrypted, the process continues with step 415, where the encrypted data message is encrypted with the recipient's public key, and the user ID and the recipient's ID are appended to that encrypted message without further encryption, and the process continues with step 411.

Optionally, all messages may be encrypted at one time, and then transmitted in succession at one time, rather than encrypting a first message with one public key, then transmitting the encrypted first message

right away, then encrypting a second message using another public key, and transmitting the encrypted second message immediately, and so forth.

The above text and associated drawings describe a method using public-key encryption. Private-key encryption, where the same key is used to encrypt and decrypt a message, may also be used. For example, the key used to encrypt the message send to the encryption server may be the same or identical key used to decrypt the encrypted message at the encryption server. In addition, the encryption method employed by the user to encrypt the original digital data message 105A may also be private-key encryption, rather than public-key encryption. In addition, a different encryption algorithm may be utilized for the user's first stage of encryption than for the user's second stage of encryption, the result of which is transmitted to the encryption server.

In the above manner, the encryption server encrypts the user's data message individually for each different recipient using that particular recipient's public key. The encryption server has more computing resources available to it than an individual communication unit, and can encrypt and transmit a message multiple times to many different users in a more efficient manner than can an individual communication unit. Individual communication units need not store all possible recipient's public keys, but instead need store only the encryption server's public key. Encryption of the recipient's ID(s) helps to secure the identity of the recipient(s) and eliminates a source of information for traffic analysis by undesired readers/interceptors of such information.

What is claimed is:

Claims

1. A method comprising the steps of:

at a communication unit operated by a user:

generating a digital data message;

encrypting the digital data message using a first key, yielding a first encrypted message;

appending an identification of the user and an identification of a first recipient to the first encrypted message, yielding an appended first encrypted message;

encrypting the appended first encrypted message using a second key, yielding a second encrypted message;

transmitting the second encrypted message to the encryption server, wherein the encryption server is not the first recipient.

2. The method of claim 1, wherein the first key is a private key associated with the user and wherein the second key is a public key associated with the encryption server.

3. The method of claim 1, wherein the step of appending further comprises the step of appending an identification of a second recipient to the first encrypted message, thereby yielding the appended first encrypted message.

4. A method comprising the steps of:

at an encryption server:

receiving a first encrypted message;

decrypting the encrypted message using a first key, yielding a decrypted message comprising a second encrypted message, an identification of a sender of the first encrypted message, and an identification of a first recipient;

determining the second encrypted message, the identification of the sender, and the identification of the first recipient from the decrypted message;

encrypting the second encrypted message and the identification of the sender with a second key, yielding a third encrypted message;

transmitting the third encrypted message to the first recipient.

5. The method of claim 4, wherein the first key is a private key associated with the encryption server and wherein the second key is a public key associated with the first recipient.

6. The method of claim 4, further comprising, when a second identification of a second recipient is part of the decrypted message, the steps of encrypting, by the encryption server, the second encrypted message and the identification of the sender with a third key, yielding a fourth encrypted message, and transmitting the fourth encrypted message to the second recipient.

7. A method comprising the steps of:

at a communication unit operated by a user:

generating a digital data message;

encrypting the digital data message using a first key, yielding a first encrypted message;

encrypting the first encrypted message using a second key, yielding a second encrypted message;

appending an identification of the user and an identification of a first recipient to the second encrypted message, yielding an appended second encrypted message;

transmitting the appended second encrypted message to the encryption server;

at the encryption server:

receiving the appended second encrypted message;

determining the second encrypted message, the identification of the user, and the identification of the first recipient from the appended second encrypted message;

decrypting the second encrypted message using a third key, yielding the first encrypted message;

encrypting the first encrypted message with a fourth key, yielding a third encrypted message;

transmitting the third encrypted message to the first recipient.

8. The method of claim 7, wherein the step of appending further comprises the step of appending an identification of a second recipient to the second encrypted message, thereby yielding the appended second encrypted message.

9. The method of claim 7, wherein the first key is a private key associated with the user, wherein the second key is a public key associated with the encryption server, wherein the third key is a private key associated with the encryption server, and wherein the fourth key is a public key associated with the first recipient.

10. The method of claim 7, wherein the identification of the user is encrypted using the second key before the step of appending.

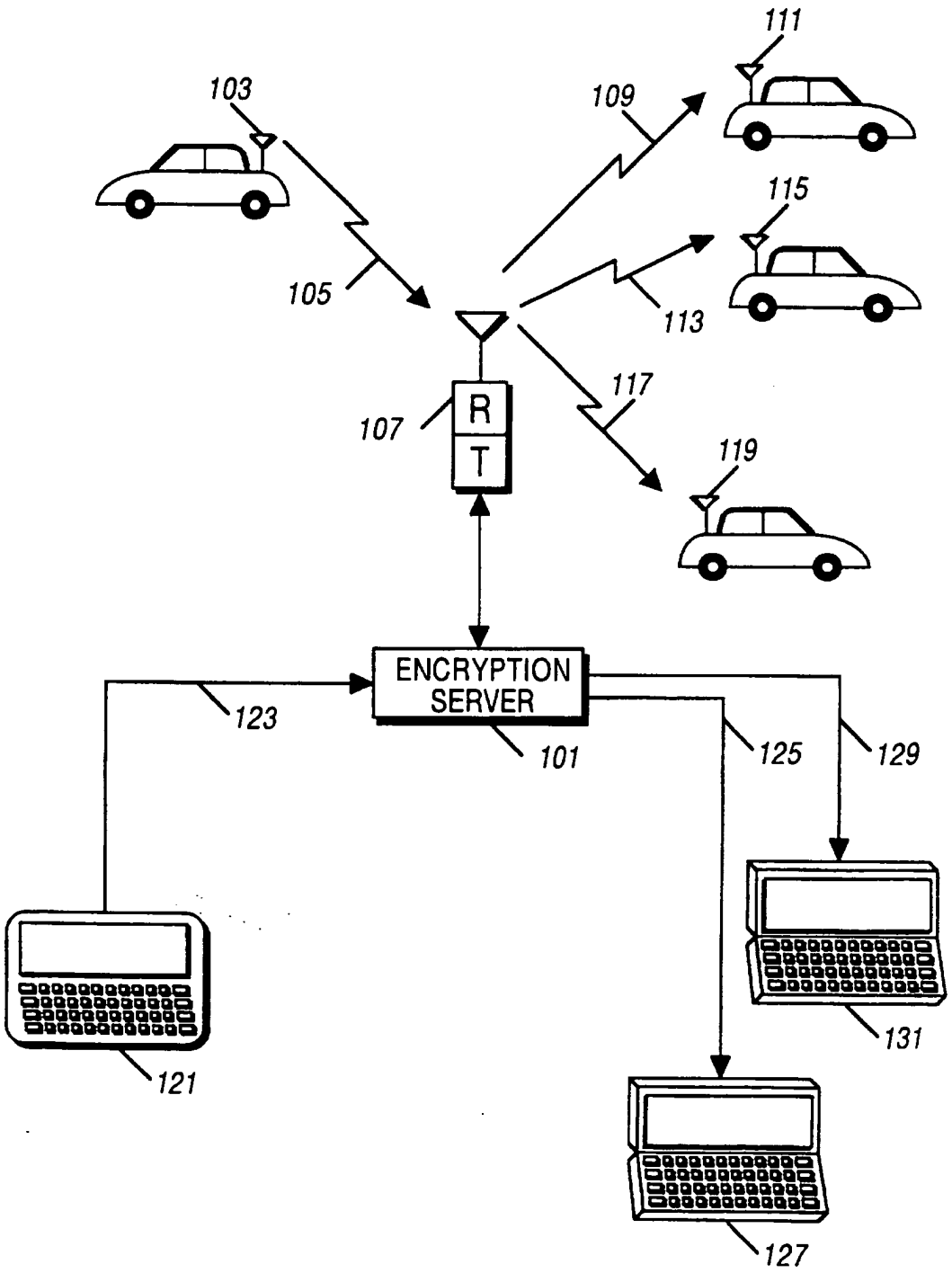

111
103
109
115
105
113
R
T
107
117
119
ENCRYPTION SERVER
123
101
125
129
121
131
127

FIG. 1

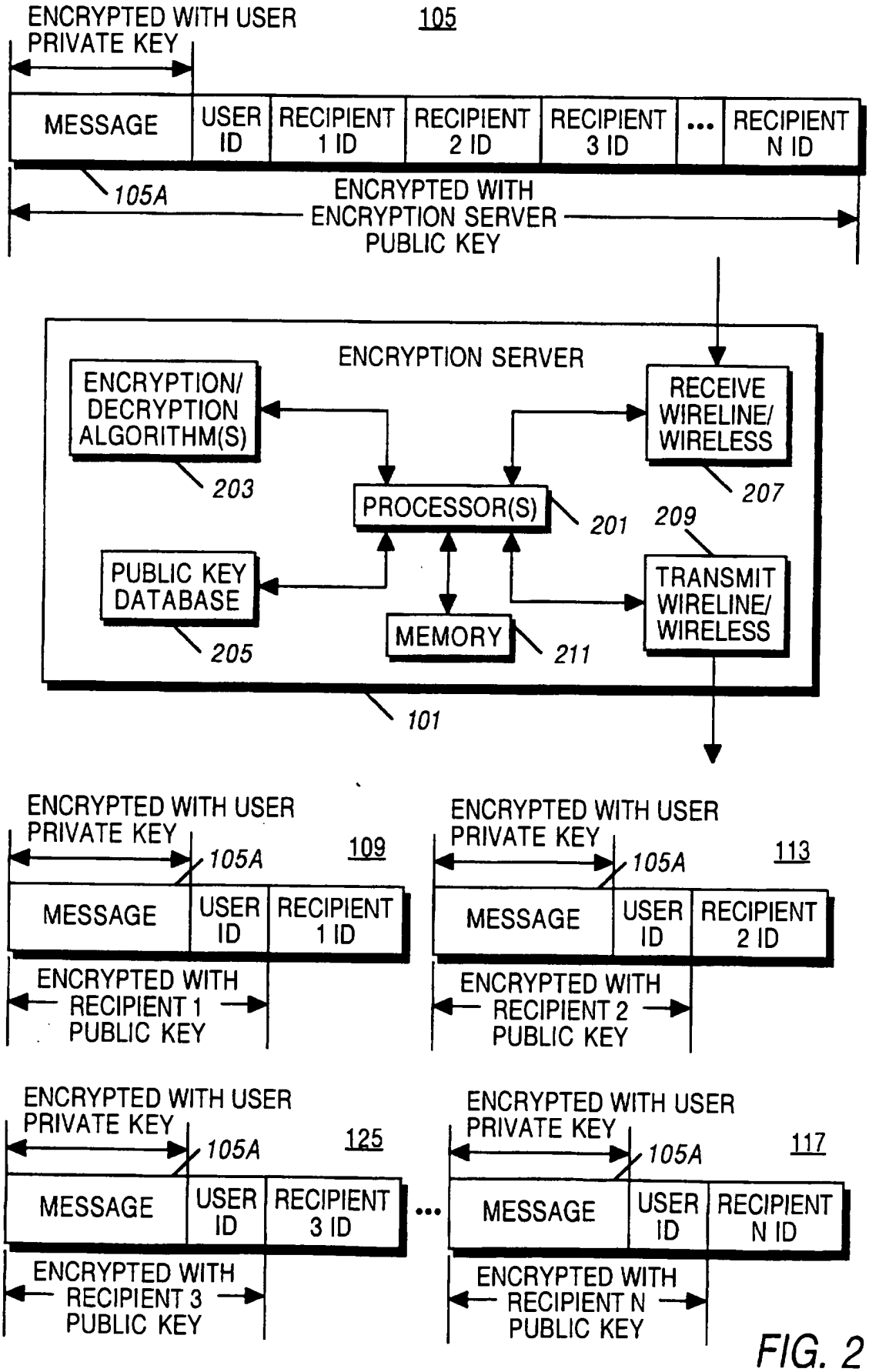

ENCRYPTED WITH USER PRIVATE KEY
105
MESSAGE
USER ID
RECIPIENT 1 ID
RECIPIENT 2 ID
RECIPIENT 3 ID
...
RECIPIENT N ID
105A
ENCRYPTED WITH ENCRYPTION SERVER PUBLIC KEY
ENCRYPTION SERVER
ENCRYPTION/ DECRYPTION ALGORITHM(S)
203
RECEIVE WIRELINE/ WIRELESS
207
PROCESSOR(S)
201
209
PUBLIC KEY DATABASE
205
MEMORY
211
TRANSMIT WIRELINE/ WIRELESS
101
ENCRYPTED WITH USER PRIVATE KEY
105A
109
MESSAGE
USER ID
RECIPIENT 1 ID
ENCRYPTED WITH RECIPIENT 1 PUBLIC KEY
ENCRYPTED WITH USER PRIVATE KEY
105A
113
MESSAGE
USER ID
RECIPIENT 2 ID
ENCRYPTED WITH RECIPIENT 2 PUBLIC KEY
ENCRYPTED WITH USER PRIVATE KEY
105A
125
MESSAGE
USER ID
RECIPIENT 3 ID
ENCRYPTED WITH RECIPIENT 3 PUBLIC KEY
...
ENCRYPTED WITH USER PRIVATE KEY
105A
117
MESSAGE
USER ID
RECIPIENT N ID
ENCRYPTED WITH RECIPIENT N PUBLIC KEY

FIG. 2

FIG. 3

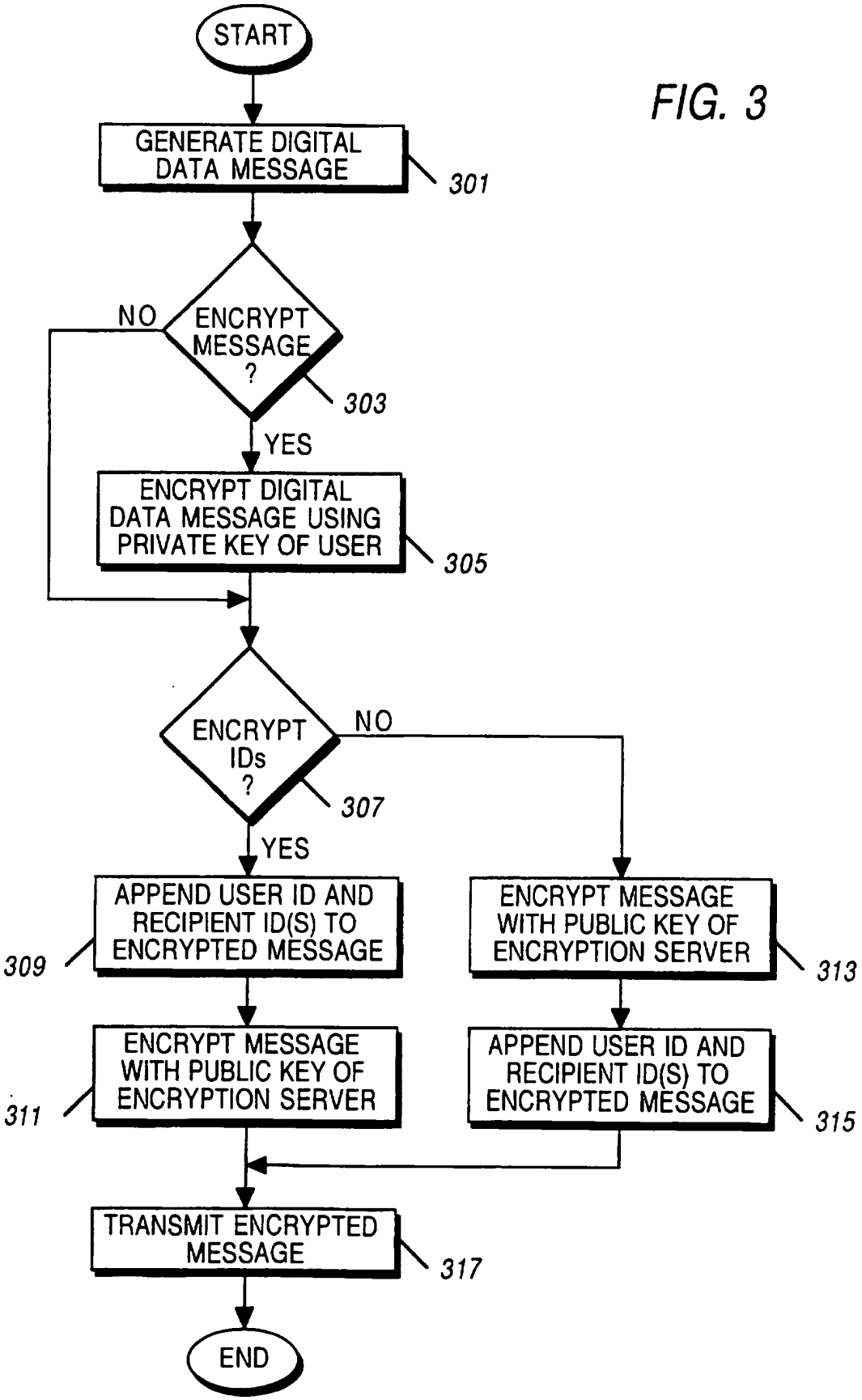
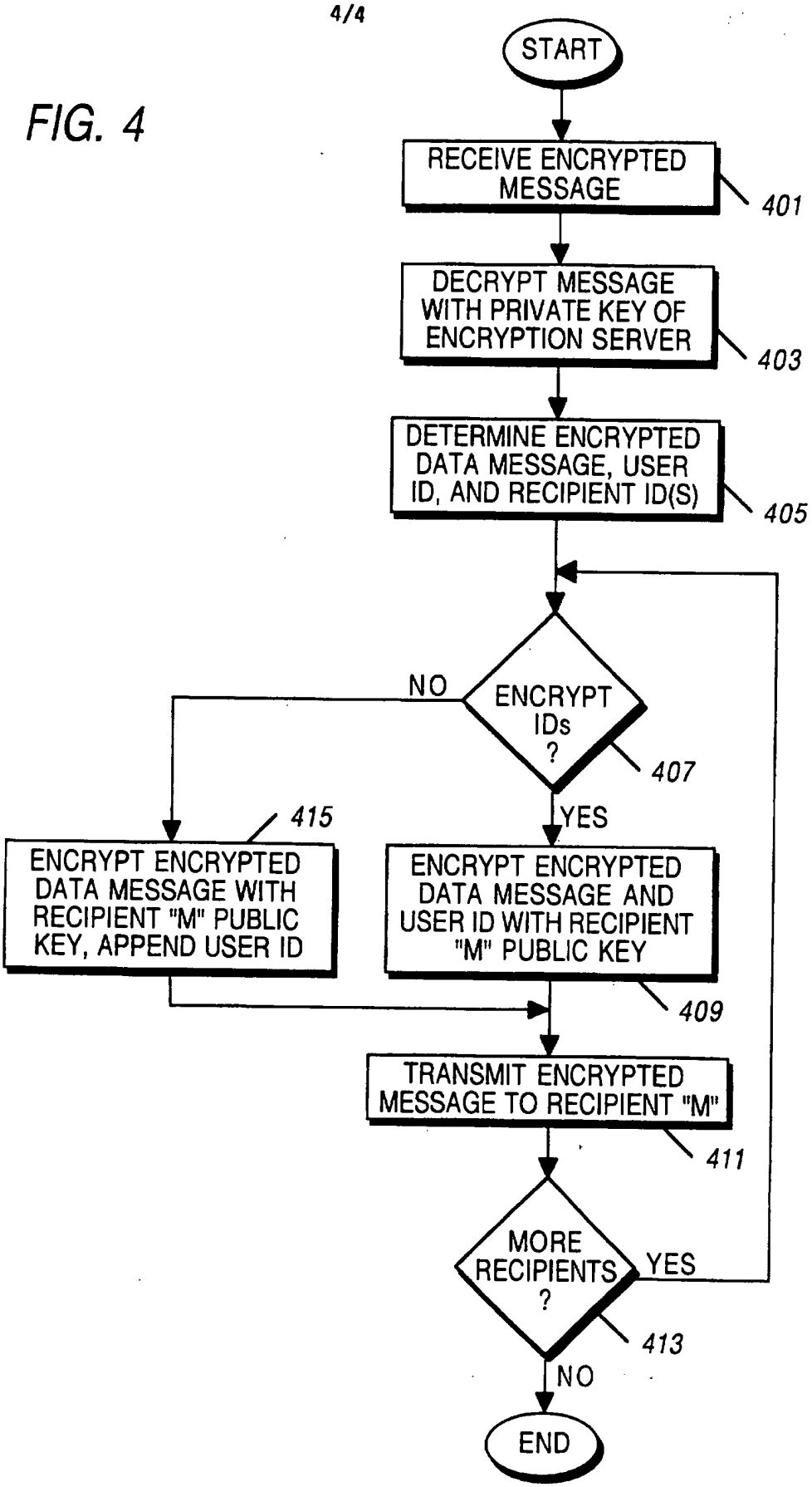

START
GENERATE DIGITAL DATA MESSAGE
301
ENCRYPT MESSAGE ?
303
NO
YES
ENCRYPT DIGITAL DATA MESSAGE USING PRIVATE KEY OF USER
305
ENCRYPT IDs ?
307
NO
YES
APPEND USER ID AND RECIPIENT ID(S) TO ENCRYPTED MESSAGE
309
ENCRYPT MESSAGE WITH PUBLIC KEY OF ENCRYPTION SERVER
311
ENCRYPT MESSAGE WITH PUBLIC KEY OF ENCRYPTION SERVER
313
APPEND USER ID AND RECIPIENT ID(S) TO ENCRYPTED MESSAGE
315
TRANSMIT ENCRYPTED MESSAGE
317
END

FIG. 4


START
RECEIVE ENCRYPTED MESSAGE
401
DECRYPT MESSAGE WITH PRIVATE KEY OF ENCRYPTION SERVER
403
DETERMINE ENCRYPTED DATA MESSAGE, USER ID, AND RECIPIENT ID(S)
405
ENCRYPT IDs ?
407
NO
YES
ENCRYPT ENCRYPTED DATA MESSAGE WITH RECIPIENT "M" PUBLIC KEY, APPEND USER ID
415
ENCRYPT ENCRYPTED DATA MESSAGE AND USER ID WITH RECIPIENT "M" PUBLIC KEY
409
TRANSMIT ENCRYPTED MESSAGE TO RECIPIENT "M"
411
MORE RECIPIENTS ?
413
YES
NO
END

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04N 5/913

A1

(11) International Publication Number: **WO 97/37492**

(43) International Publication Date: 9 October 1997 (09.10.97)

(21) International Application Number: PCT/US97/05257

(22) International Filing Date: 31 March 1997 (31.03.97)

(30) Priority Data:
60/014,684 1 April 1996 (01.04.96) US

(71) Applicant *(for all designated States except US)*: MACROVISION CORPORATION [US/US]; 1341 Orleans Drive, Sunnyvale, CA 94089 (US).

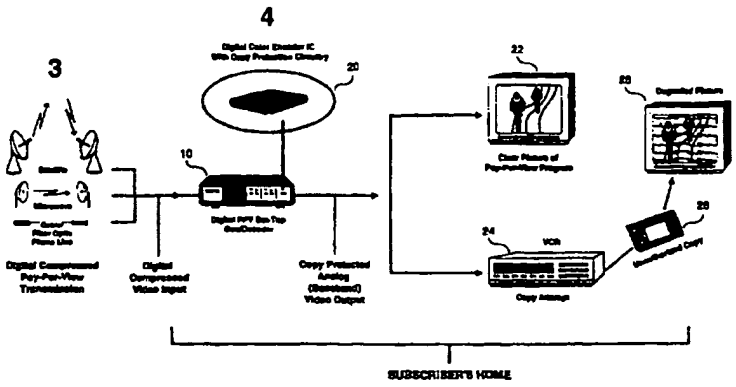
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: WONFOR, Peter, J. [US/US]; 962 Malaga, El Granada, CA 94089 (US). NELSON, Derek [US/US]; 3250 A. Glendale Avenue, Menlo Park, CA 94025 (US).

(74) Agent: BRILL, Gerow, D.; Macrovision Corporation, 1341 Orleans Drive, Sunnyvale, CA 94089 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

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: A METHOD FOR CONTROLLING COPY PROTECTION IN DIGITAL VIDEO NETWORKS



(57) Abstract

A method and system of providing copy protection of video analog and digital signals and the like, wherein the signals are transmitted via a digital delivery network, and may comprise, for example, pay-per-view (PPV) program materials protected by copyrights of respective program rights holders. The right holders authorize video service providers (3) to apply copy protection to the program material. The copy protection process is supplied to the rights holders or the service providers (3) by a copy protection process licensor. The video service providers (3) supply suitable copy protection control software via respective control and billing (tracking) centers to generate commands which activate, control and reconfigure the copy protection process being applied to the programs being transmitted. A set-top box (10) is provided to each consumer and contains a copy protection circuit which is adapted to apply selected anticopy waveforms to the video signal corresponding to the program material in response to the commands from the service providers (3). Usage data pertinent to each consumer is returned by the set-top box (10) to the service providers (3), which then report the copy protection usage to the respective rights holders and process licensor.

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.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

A METHOD FOR CONTROLLING COPY PROTECTION IN DIGITAL VIDEO NETWORKS

BACKGROUND OF THE INVENTION

Field of the Invention

This disclosure is directed to a method of controlling copy protection in digital video networks where it is desired to copy protect an analog or digital video output signal associated with a digital video network.

Background of the Invention

Various well known copy protection schemes for video signals include that disclosed in U.S. Patent No. 4,631,603, John O. Ryan, December 23, 1986 and assigned to Macrovision Corporation, incorporated by reference, directed to modifying an analog video signal to inhibit making of acceptable video recordings therefrom. This discloses adding a plurality of pulse pairs to the otherwise unused lines of a video signal vertical blanking interval, each pulse pair being a negative-going pulse followed closely by a positive-going pulse. The effect is to confuse AGC (automatic gain control circuitry) of a VCR (video cassette recorder) recording such a signal, so that the recorded signal is unviewable due to the presence of an excessively dark picture when the recorded signal is played back.

Another analog video protection scheme is disclosed in U.S. Patent No. 4,914,694 issued April 3, 1990, to Leonard, and assigned to Eidak Corp., incorporated by reference. The Eidak system (see Abstract) increases or decreases the length of each video field from the standard length, either by changing the time duration of the respective horizontal line intervals in each field while keeping a constant, standard number of lines per frame, or by changing the number of horizontal line intervals which constitute a frame while maintaining the standard duration of each line interval.

These video protection systems modify the video signal to be recorded (for instance on tape) or to be broadcast (for instance protected pay-per-view television programs) to make copying by ordinary VCRs difficult or impossible. When a video tape on which is recorded the copy protected video signal is played back for viewing using a VCR, the copy protection process is essentially transparent, i.e., it does not interfere with viewing. However, any attempt made to copy the video signal from the tape using a second VCR to record the output of the first (playback) VCR yields a picture degraded to some extent, depending on the efficacy of the particular copy protection system. These present video copy protection systems protect only analog video signals, which are the type of video signals broadcast and recorded using current consumer video technology.

Some digital and hybrid solutions to the copy protection problem were solved by US Patent 5,315,448, issued May 24, 1994, issued to Ryan and assigned to Macrovision Corporation, incorporated by reference. This patent is directed to copy protection for use with digital signal recording where it is desired to copy protect both an analog and digital signal associated with a digital VCR, and any signal material where the original source material is not copy protectable.

A fundamental revolution is under way that will dramatically affect the delivery of home entertainment. Consumers will soon have hundreds of viewing options from which to choose because of advances in digital compression technologies and the associated reduction in costs accompanying each advance. Because of the increased number of channels more channels will be allocated for pay-per-view (PPV). The increased number of PPV channels will mean video service providers (VSP), also known as PPV providers or system operators, can provide a greater number of movies and more start times, ultimately changing the way many consumers purchase and view movies in their homes. Already, market research experts are predicting that the pay-per-view business will rival today's videocassette rental and sell-through business within 3-5 years.

Even with such a positive outlook for the future of PPV, the full benefits to the consumer of PPV programming may be delayed unless new digital video networks can protect PPV program copyrights. Rights owners are concerned that when digital programming is delivered to the home any digital set-top box will be able to produce a commercial quality video when recorded by a consumer VCR.

SUMMARY OF THE INVENTION

In this new world of direct-to-home video programming, video service providers will be called upon to protect PPV programming against unauthorized copying. They will be obligated to develop and manage the headend (cable) or uplink (satellite) systems which monitor, control, track, and report the application of copy protection on each pay-per-view video program. To this end, the present invention provides copy protection management framework which meets these needs while complementing the more technically detailed copy protection management strategy for video service providers. This framework serves to integrate all components of copy protection delivery in a digital network, and is designed to fit the diverse needs of DBS, Telco, and Cable operators while meeting the requirements of rights owners for a robust and secure environment in which to deliver copy protected PPV programming.

The value of PPV copy protection is maximized when the appropriate control and tracking systems are in place at the video service provider's control and billing centers. These control and tracking systems are best specified during the design phase of the digital signal material delivery system. At a minimum, the following system components are required:

- Copy protection-capable set-top boxes
- Capability to deliver programmable copy protection configuration
- Capability to deliver real time on/off/mode command
- Transaction/billing reporting systems/programs

A control and tracking system in accordance with the invention, for providing copy protection for a typical digital delivery system can be best understood through a short case study which begins when a consumer, that is a subscriber, receives a new set-top box. Each set-top box includes a copy protection capable digital-to-analog encoder chip. When the set-top box is initially powered on, the encoder chip is remotely programmed via a video service provider with the desired copy protection configuration. Thus the video service provider's system management software (SMS), also termed hereinafter as system control software (SCS), has the ability to store and track the designated configuration. The configuration information

applies to all copy protected programming and is updated only when a video service provider is informed of a change in the process or when a set-top box is initialized.

The copy protection status or option of each program is contained in the video service provider's system control software database. There are several potential copy protection status options. For example, a first option is for copy protection which allows for viewing only at a PPV transaction fee. A second option is for copy protection which allows for taping at a higher transaction fee. A third option is for non-protected program material for which no copy protection is required (for example, broadcast television).

When the consumer selects a viewing choice via an electronic program guide, a correct menu of options is displayed. Once a PPV program is selected by the consumer, the correct copy protection status is applied as determined by the consumer's chosen option and scheduling software of the system control software database. Either the headend/uplink facility's control software or software at the set-top box can determine and send the appropriate on/off/mode command to the copy protection capable digital-to-analog chip of previous mention.

The headend/uplink software communicates the on/off/mode command to the set-top box to correctly set the copy protection for a particular program. The system scheduling software has the capability to prevent copy protection from being applied to any type of program other than PPV programming since copy protection is licensed only for use on PPV programming. After a PPV program is viewed by a consumer, the set-top box is able to communicate to a billing subsystem of the system control software all relevant transaction data. From this data the billing subsystem is able to add this information to copy protection activity reports. These reports contain information such as the number of purchases, retail price, and copy protection usage fees owed to a licensor.

The copy protection process is applied to the analog video signal just prior to its exiting the consumer's set-top box. The application of the copy protection process is controlled and managed by system control/access software of the system control software that resides in the video service provider's operations control and billing center.

All set-top boxes in the network need to contain copy protection circuitry. If a set-top box does not have copy protection capability then the video service provider

is able to identify those set-top boxes and deny them copy protected PPV programming.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram depicting a summary of the functions of the present invention.

Fig. 2 is a block diagram depicting a typical digital set top box/decoder of the present invention.

Fig. 3 is a block diagram illustrating an example of the circuitry and architecture of the set-top box of Fig. 2 in further detail.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The basic copy protection which is controlled and tracked in accordance with the present invention, is the subject of numerous patents and co-pending applications. The PPV copy protection process works by exploiting the differences between the way television (TV) sets and VCRs respond to video signals. The two components of the anticopy process are known as the automatic gain control (AGC) and Colorstripe™ processes. The purpose of these two separate components or processes is to modify the video signal in a manner which has no effect on a TV set but which inhibits a recording VCR from making a watchable copy.

The combination of the AGC based anticopy process and the Colorstripe™ technology developed specifically for PPV applications results in an overall effectiveness rating of more than 95%. This means that over 95% of unauthorized copies will be either unwatchable or have substantially reduced entertainment value.

Security is also a major factor in the operational effectiveness of PPV copy protection. Security is a measure of the difficulty in bypassing or defeating the anticopy process. Ideally the system is completely undefeatable, but as a practical matter the copy protection system needs to be secure enough to thwart attempted breaches by typical consumers, including reasonably sophisticated consumers. The security system is successful if the vast majority of consumers are prevented from taping PPV programs in the home.

Both video service providers (VSPs), that is, PPV providers, and rights owners benefit when current movie programming is offered to consumers at the same time or shortly after these movies are available on videocassette. Subscribers benefit as well since this scenario provides them with more choices and added convenience.

As digital PPV programming generates increasing revenue for rights owners and becomes a viable viewing option to prerecorded videocassettes, video service providers will be called upon to copy protect PPV programming so that the videocassette rental and videocassette sell-through businesses are not compromised. Rights owners also will require video service providers to monitor, control, track, and report the application of copy protection on each video program for billing purposes.

Copy protection has emerged as a key element in the delivery of PPV programming via digital signal delivery networks. The aggregate system implications of copy protection are very manageable, but only when designed as a part of the overall digital delivery system architecture.

The description of the present invention is intended to apply to systems where one or more video service providers are, or will be in the future, connected to a pay-per-view (PPV) service. The PPV service can be either a video-on-demand (VOD) format, or a near video-on-demand (NVOD) format and digital delivery network, and where set-top boxes (STBs) from multiple manufacturers may be connected to the network. It is assumed that one class of technology will be deployed initially [such as Direct Broadcast Satellite (DBS), Multi-point Microwave Distribution System (MMDS), telephone line or Hybrid-Fiber Coax (HFC)] to be followed by another class of technology at some future date. Although a different technology may arise, it is intended that the invention is applicable to use with multiple platforms and technologies.

Fig. 1 illustrates a control and tracking method and system for enabling and controlling the application of copy protection of video signals and the like via digital video networks. Station 1 represents the issuance of instructions to video service providers by program rights holders who hold the copyrights, for the application by the providers of copy protection to the programs which are protected by per-per-view (PPV) or pay-to-tape (PTT) requirements.

Station 2 depicts a control and billing center of the licensed video service providers who supply copy protection control software for the respective protected programs being broadcast, to generate the commands required to activate, control and reconfigure the copy protection process for each specific PPV/PTT program offering. Although a single provider is depicted, it is understood that station 2 represents any plurality of video service providers each with their respective proprietary control and tracking (billing) software, in accordance with the present invention.

Station 3 represents the procedure of transmitting the particular copy protection command codes of the respective providers, for the PPV/PTT program offerings, via the typical broadcasting networks. Such transmissions may be made by satellite, by microwave, by phone line or by cable transmission systems as depicted.

Station 4 represents the subscriber's home, or other receiving facility, and includes a set-top box 10 for each of a multitude of subscribers. Each set-top box contains copy protection circuitry including a digital color encoder integrated chip (IC), which is adapted to apply selected anticopy waveforms to the analog or digital video signal which is supplied therefrom to a television set or monitor. The receiving facility is further described in Fig. 2.

Station 5 represents the procedure whereby data identifying each PPV or PTT transaction, including copy protection usage, is sent by the set-top box 10 back through the transmission networks of station 3, generally to the respective video service provider's control and billing (tracking) center. The center includes billing procedures which are a subset of the system control software and which process the return transaction data to provide for billing the subscriber for the PPV or PTT transaction usage.

Station 6 represents the procedure whereby each of the licensed video service providers report the copy protection usage to the program rights holder, whereby the provider pays the copy protection fees to the rights holder, i.e., the licensor.

Fig. 2 illustrates in further detail the subscriber's facility, station 4 of Fig. 1, receiving the digital, and usually compressed, pay-per-view transmissions from the broadcasting networks depicted as station 3 of Fig. 1. The compressed digital video

signal, or the like, is supplied to the respective set-top box 10 of a multitude of set-top boxes, wherein each box includes conventional circuits for converting and decoding the digital compressed video signal to an analog (baseband) video signal. The set-top box 10 also includes a digital color encoder IC 20 of previous mention which contains copy protection circuitry for applying the selected copy protection waveforms to the analog (or digital) video signal, namely, the programs which are being protected. In this example, the copy protected analog baseband video is supplied by the set-top box to a TV set 22 where the pay-per-view protected program clearly is displayed for viewing if the subscriber is authorized to view the program. If the subscriber is not authorized for a particular PPV protected program, the corresponding picture is modified so as to be un-viewable.

In the event a subscriber records the PPV protected program via a VCR 24 to obtain a taped copy 26 without authorization, the unauthorized copy will be degraded to the degree that it is un-watchable, as depicted by a TV set 28. However, if the subscriber subscribes to a pay-to-tape transaction and to the required higher PTT transaction fee, then the copy is authorized and the resulting taped copy would readily be watchable.

Referring to Fig. 3, there is illustrated in further detail an architecture of the set-top box(es) 10 of Figs. 1, 2. Upon power up of the set-top box 10 the configuration bits stored in flash memory 48 are read and written into the appropriate CP control registers 52 in the NTSC/PAL encoder 20. When the compressed digital video signal, including the copy protection control commands of previous and following discussion, are supplied by the delivery network of previous mention (satellite, HFC, MMDS, phone line) to a demodulator circuit 32, as depicted by an input lead 30. The demodulated video/audio and control signals are supplied to a demultiplexer circuit 34 where the video/audio signals are separated into respective channels and supplied to an MPEG-2 decoder and digital decompression circuit 36. The copy protection control commands are supplied from the demultiplexer 34 to a conditional access system module 38. The commands are supplied to a microprocessor in a CPU 40. The CPU processes information located in memory that is associated with the Electronic Program Guide (EPG) 46 or runs the copy protection application software 44 residing in memory 42 to deliver the activation command to the NTSC/PAL encoder 20. The EPG may also have data

which is used to determine if copy protection should be activated. There are additional methods that may be employed to activate copy protection.

In response to the control commands, the CPU 40 supplies control signals to the NTSC/PAL encoder IC 20 of previous mention, Fig. 2. The encoder IC 20 includes copy protection control registers 50, 52 for receiving the mode bits and configuration control bits respectively, of previous and following discussion. The configuration bits 52 determine the form of the copy protection (i.e., where the Pseudo Sync and AGC pulses will be located or positions of the colorstripe lines etc.) The on/off/mode byte 50 determines which components of the copy protection process will be activated. See table 1 below. The encoder IC 20 also receives decompressed video from the MPEG-2 decoder and digital decompression circuit 36. Encoder IC 20 outputs a RF signal, a composite video signal and/or an S-video signal via video leads 54. The decompressed audio signal is supplied from the circuit 36 to an audio processing circuit 56 which, in turn, outputs left and right channel stereo signals and/or an AC-3 signal on audio leads 58.

In accordance with the invention, the set-top box needs to satisfy certain requirements to insure that the copy protection process is correctly generated, controlled and tracked. Control and tracking of the copy protection process usage takes place at the VSP's control and billing center, station 2 of Fig. 1. This in turn requires that certain capabilities exist which involve the set-top box, the system control and the billing systems and programs in order to satisfy these requirements.

There follows a description of the requirements which ensure that the copy protection process or technique is correctly activated and controlled and its usage tracked. It is expected that if non-compliant set-top box hardware is attached to the digital delivery network, that each licensed service provider will be able to identify such hardware as non-compliant and will withhold copy protected programs from the respective subscriber.

Implementation of these control requirements over the network (i.e. control of the anticopy process from the program origination control and billing center) requires knowledge of the set-top box control system and process, the application program interfaces (API) present at the box and the dialog between it and the integrated circuit (IC) which incorporates the copy protection apparatus.

Copy protection control software (CPCS) is a software module or set of software modules that reside in the service provider's system control software (SCS). It provides a system operator (that is, the service provider) with an interface to manage the necessary attributes of the pay-per-view copy protection in accordance with the present invention.

For security reasons there needs to be the capability to control access to the CPCS from the system control software. This restriction is designed to limit access to the CPCS for control of the copy protection process. The operating system supporting the SCS is generally the first level of security. Every employee is required to enter a login account and password. Without these an employee is denied access. The employee's account specifies the respective privileges. A system administrator of the service provider is responsible for the assignment of the employee's privileges.

Thus, every executable file residing on the host which is capable of modifying the operational status of the copy protection process has permissions restricted to authorized personnel. Without the proper permissions, the personnel are unable to run the executable software.

The CPCS is the portion of the video service provider's software control where the decision to apply the options of pay-per-view and pay-to-tape are applied on a program-by-program basis.

There is access control to the CPCS either through password control or the assignment/denial of privileges through software. If password control is the selected method then once the correct initial password is entered, CPCS forces the selection of a new password for future access to CPCS. In this way the service provider can limit access to CPCS to those employees who carry the authority to modify the copy protection database. The password is valid for a reasonable amount of time before it expires and selection of a new password is required.

Additionally there is an access control to a subsystem within the CPCS that allows the modification of selected bits which define the configuration control and mode, and thus determine the characteristics, of the copy protection process. Any unauthorized changes to these bits can result in severe playability and effectiveness problems. In order to maximize the security of the system the video service provider needs to have a short list of personnel who are authorized to change these bits.

A mode control group controls access to the mode bits. This group has the ability to change the contents of the mode byte(s) which is sent with each PPV program to activate or deactivate the copy protection process. The membership of this group is controlled by the system administrator. The number of the service provider's personnel allowed in this group is kept to a minimum.

Similarly, a configuration control group controls access to the configuration bits. This group has the ability to change the contents of the configuration bits which define the copy protection process. These are the bits that are sent periodically to every set-top box to assure that all boxes are using the correct version of the process. The number of the service provider's personnel allowed in this group also is kept to a minimum.

Each password described below should be at least eight (8) alpha-numeric characters in length. The system administrator is responsible for defining and distributing the current password to the authorized personnel. Each password described below should have a life of no more than four months before the system administrator changes the password.

Password access to the software that applies or removes the copy protection process on a program-by-program basis is designed to query mode or configuration control group authorized personnel for an authorization password to ensure that they are a member. If the authorized personnel correctly enter the password they will be allowed to apply or remove the copy protection for a particular PPV or series of PPV events. Conversely, if authorized personnel fail to enter the password they must be denied access to that portion of the database. It is the system administrator's responsibility to ensure that only authorized personnel know the password for either the mode or configuration control. An authorized personnel will be given three attempts to login before a message is generated for the system administrator that an unauthorized request to modify the application or remove the copy protection has been made.

Alternative proposals for accessing CPCS and controlling access to the mode and configuration of the copy protection process may be developed by one skilled in the art.

The CPCS will perform the following functions: Copy protection on/off and mode control; copy protection validation; functionally unlocking copy protection

capability in a set-top box; and copy protection process configuration reprogramming.

The copy protection process which is incorporated in the set-top box is controlled by the CPCS at the licensed video service provider's control and billing central location. The need to invoke copy protection on an individual program forms part of a descriptor for each program. A default for copy protection within the descriptor needs to be turned off (i.e., no copy protection).

Steps need to be taken to prevent copy protection being applied to non-PPV program channels, since copy protection can be licensed only for PPV programming. If the system control software automatically verifies that a program is designated for PPV use, this requirement may be automated. Similarly, access to CPCS may be automatically denied for non-PPV programming. If such an automatic verification is not made, a warning notice is generated when CPCS is accessed to change the copy protection status of a program. This notice needs to be displayed until a specific keyboard entry is made to acknowledge the warning.

In the case of MPEG signals, the MPEG copyright header bits on their own are not sufficient to activate copy protection in the set-top box. The following reasons are the basis for not allowing the MPEG header bits to be used as the sole control of the copy protection process. An application routine is required in order to (a) differentiate between digital-to-digital and digital-to-analog copy protection conditions, (b) provide sufficient control capacity to set the copy protection operating mode, and (c) facilitate access to the copy protection system only by licensed video service providers.

It is preferred that the anticopy process on/off control is achieved by setting all the individual parameter on/off and mode control bits rather than a master on/off control. This requires that the N0 (N-zero) bits in the control bit listing be set as required. Depending on the individual system, this will require the control of from 5 to 8 bits.

The delivery of the mode byte to the set-top box to activate or deactivate the copy protection process may be accomplished in several ways. Each method has its positive aspects as well as its negative aspects. When selecting a mechanism to control the copy protection technology, a service provider selects one of the following means or may develop an entirely new means.

One method may be for the mode byte to be delivered via the conditional access system via the entitlement control message (ECM). Another method might be to include the mode byte in a private data field in the MPEG transport data stream.

Another method may deliver the mode byte in a user defined section of the electronic program guide (EPG) that is not identified in released documentation as controlling copy protection. This method also requires some additional security to keep the memory location of the mode byte from being accessed for unauthorized changes and the setting of a return flag that indicates the actual status of the mode byte when transmitted to the NTSC encoder.

Another method may be a combination of the conditional access ECM and EPG. The transport of the mode byte in the EPG could be combined with two bits within the ECM. To activate the copy protection technology then it would be an or operation between the ECM bits and the EPG bits. If either is set, the copy protection technology, both ECM and EPG would have to indicate that deactivation is necessary.

When a copy protected PPV program is viewed, part of the information that will need to be tracked will be the actual setting of the mode byte. In this way both the copy protection process and the service provider will have a means to discover if copy protection has been circumvented in the set-top box. The return flag may be a simple bit set to 'true' to indicate that the copy protection process was correctly activated and 'false' if it was incorrectly activated. It is required that the mode byte be sent to the NTSC encoder on a periodic basis. The frequency of the transmission is on the order of once every minute.

Setting the operating mode of the copy protection process requires independent activation of the three component parts of the copy protection process (pulses within the vertical blanking interval, pulses at end of field, colorburst phase modification) and up to 5 additional mode set parameters using N0 bits as indicated above.

Access to copy protection at the set-top box by the video service provider needs to be restricted to authorized providers. This should not to be confused with access to the CPCS as defined earlier. It follows that each system operator or video service provider is required to procure the means (i.e., keys/codes, etc.) to activate

the copy protection system control software on a program-by-program basis. When a service provider obtains the means to activate copy protection, the provider will gain access to the copy protection process at the set-top box. The copy protection process (i.e. on/off/mode or reprogramming commands) at the set-top box needs to have controlled access such that only authorized providers can issue valid commands to the box. The set-top box needs to reject commands for the copy protection process from unauthorized video service providers.

Set-top boxes such as depicted in Figs. 1, 2, may be shipped by the manufacturer with the copy protection capability installed, but functionally locked. This means that the set-top box will not respond to any copy protection control codes. However, the set-top box will be unlocked (i.e. enabled) by a message initiated via the CPCS or SCS and sent through the system by a licensed video service provider. This message may be sent as part of the log-on routine when a subscriber accesses a provider. This message need only be acted upon once by the set-top box during the lifetime of the box. Only authorized video service providers are provided with the unlocking message data.

The copy protection unlock message consists of at least 8 bytes. The set-top boxes are manufactured with an appropriate unlock message code. This code is provided by the set-top box manufacturer only to a copy protection licensor, who in turn provides the code to licensed video service providers. The copy protection unlock message is different for each set-top box manufacturer, but is the same for all boxes made by that manufacturer.

Alternative proposals on the methodology to enable the copy protection process in the set-top box will be apparent to those skilled in the art.

To ensure that over the life of the set-top box the copy protection process provides the maximum effectiveness with VCRs and compatibility with TV sets, the copy protection system needs to be upgradeable on a system-wide basis by means of commands initiated by the CPCS. This will result in new process configuration data being transmitted. In response, the set-top box processes the data to reconfigure the adjustable parameters of the copy protection process. The set-top box may be placed in a "diagnostics" mode for this feature implementation, or the configuration data may be sent and acted on by the box on a routine basis as part of the program description data or log-on routine.

However, it is recommended that the entitlement control message (ECM) be used. The ECM is embedded in the conditional access system.

In one version, configuration data of 108 bits is provided to accommodate the reconfiguration data, however, 108 bits does not fall on a byte boundary. Therefore, it is recommended that 112 be sent with a pad 0. The data is presented to the service provider in the form of hexadecimal numbers for entry into the CPCS. The 112 bits thus are entered as a string of 28 hexadecimal numbers.

In another version, configuration data of 132 bits is provided to accommodate the reconfiguration data, however, 132 bits does not fall on a byte boundary. Thus, it is recommended that 136 be sent with a pad 0. The data is presented to the provider in the form of hexadecimal numbers for entry into the CPCS. The 136 bits thus are entered as a string of 34 hexadecimal numbers.

It is possible to verify the current configuration stored by the CPCS by accessing the current contents of the configuration bits presented as the correct number hexadecimal characters. An alpha-numeric password of at least 8 bytes is required to gain access to change the programming data within CPCS. This password is separate from the password which allows access to CPCS. The service provider has the option of receiving the 'C' source code of an executable file to which to pass parameters.

The following warning notice is presented on the screen of the operational control and billing center of a provider after entering the correct password:

WARNING

Changing this copy protection configuration data without the written authorization carries the serious risk of problems with the performance of the copy protection system and degraded picture quality.

This warning notice is displayed until a specific keyboard entry is made to acknowledge the warning.

By way of example only, Table 1 illustrates a mode control bit listing which defines the corresponding bit pattern or command, which provides the routine on/off

and mode selection functions when transmitted to the set-top boxes via the delivery networks. The configuration control bit listing is generally equivalent to that of the mode control, though relatively longer since it controls considerably more control and reprogramming functions.

TABLE 1
Mode Control Bit Listing
Routine On/Off and Mode Selection

N0	**On/off and mode control; 8 bits**		
N0[7]	Reserved		CPC0[3]
N0[6]	Pay-to-tape allowed/prohibited	(Allowed=1, Default=0)	CPC0[2]
N0[5]	VBI pulses On/Off (VBIP)	(ON=1)	CPC0[1]
N0[4]	End of Field Back Porch Pulses on/off (EOFP)	(ON=1)	CPC0[0]
N0[3]	Colorstripe process On/Off (CSP)	(ON=1)	CPC1[3]
N0[2]	AGC pulse normal (amplitude cycling)/static mode select (AGCY)	(Cycling=Default=1)	CPC1[2]
N0[1]	H-sync amplitude reduction On/Off (HAMP)	(ON=1)	CPC1[1]
N0[0]	V-sync amplitude reduction On/Off (VAMP)	(ON=1)	CPC1[0]

The pay-per-view transaction information is collected by each video service provider for each subscriber so that monthly copy protection activity reports required for royalty payments and other fees may be generated. The reports include information regarding the number of subscribers accessing each copy protected program, with subtotals of the copy protection status or options selected by respective subscribers. The reports further include information sorted by PPV title, PPV program supplier, copy protection activation status requested by the subscriber, and by set-top box model code. The reports are provided by the report generating software of previous mention at the video service provider centers.

The activity report includes a manufacturer and model type descriptor code in the transaction acknowledgment between the set-top box and the control and billing system when a PPV purchase transaction is reported to the provider.

The CPCS and the set-top box are capable of applying and reporting anticopy usage according to the following conditions. The overall system allows the subscriber's copy protection to be turned off at the box only as permitted by the PPV program rights holder.

(a) PPV program rights holder permits viewing only:

The pay-to-tape mode is prohibited (off). All STBs output copy protected waveform only. I.e., the copy protection waveform unconditionally appears on the set-top box analog video output signal.

This is reported to the billing system as a "pay-per-view" copy protected transaction.

(b) PPV program rights holder permits viewing and recording:
The pay-to-tape mode bit is set for pay-to-tape permitted (on). Under this option, when the subscriber selects the "pay-to-tape" option, the copy protection process is turned "off" in the STB to allow the PPV program to be recorded (taped) for a higher transaction fee than for "viewing only." I.e., the copy protection waveform will not be present on the STB analog video output signal.

This is reported to the billing system as a "pay-to-tape" copy protected transaction.

The following Table 2 provides a summary of the control options and includes additional information.

TABLE 2
Pay-per-view and Pay-to-tape Control Options for Pay-per-view Programs

Program Descriptor of PPV Program	Consumer Request (Pay-per-view or Pay-to-tape)	Result
Copy protection NOT required	N/A	ACP off
Copy protection REQUIRED Taping NOT permitted	Pay-per-view	ACP will be ON. Pay-per-view transaction cost incurred by consumer.
Copy protection REQUIRED Taping NOT permitted	Pay-to-tape	Requested option not available. ACP will be ON. Pay-per-view transaction cost incurred by consumer.
Copy protection REQUIRED Taping permitted (at higher transaction cost)	Pay-per-view	ACP will be turned ON by STB control system. Pay-per-view transaction cost incurred by consumer.
Copy protection REQUIRED Taping permitted (at higher transaction cost)	Pay-to-tape	ACP will be turned OFF by STB control system. Pay-to-tape transaction cost incurred by consumer.

It is to be understood that various terms employed in the description herein are interchangeable. For example, a "video service provider" also is known as a pay-per-view (PPV) provider or a system operator, and the "system management software" preferably is referred to as the system control software. Likewise, the "control and billing centers" of the PPV providers represented by station 2 (and generally station 5) also may be referred to as operations control/tracking centers, program origination/termination centers, headend (cable)/uplink (satellite) control centers, etc. A licensed PPV provider facility supplies the necessary control instructions to associated software and/or circuitry in a set-top box to allow a respective subscriber access to program material to which he or she is entitled, and also receives at designated times of the week, month, etc., the usage data

automatically returned by the set-top box. A billing and license fees software subset of the system control software then enables each PPV provider to bill the subscribers and to report and pay the attendant licensing fees to the rights holders, etc.

Accordingly, the above description of the invention is illustrative and not limiting. Further modifications will be apparent to one of ordinary skill in the art in light of this disclosure. For example, although the invention is described herein relative to a video signal, and primarily an analog video signal, it is to be understood that the invention concepts may be applied to other signals with properties equivalent to a video signal where copy protection is desired. Likewise, the invention is applicable to the copy protection of digital as well as analog signal materials, such as those disclosed in the U.S. Patent No. 5,315,448 of previous mention. Further, although a specific example of a code word is disclosed herein for enabling the copy protection process via the set-top box, other combinations and numbers of bits may be employed. In addition, a selected portion of the control software for effecting the copy protection process may reside in the set-top box in the form of an insertable "smart" card, wherein for example the smart card contains the data concerning the subscriber's options and privileges.

Thus, the scope of the invention is defined by the following claims and their equivalents.

What is claimed is:

1. A method of providing copy protection of signal material transmitted via digital delivery networks, to prevent unauthorized viewing or copying of the signal material, comprising the steps of:

supplying copy protection controls indicative of desired copy protection for the signal material;

transmitting commands derived from and in response to the copy protection controls which activate the copy protection for the signal material; and

applying anticopy waveforms to the signal material in response to the commands to prevent the unauthorized viewing or copying of the signal material.

2. The method of claim 1 wherein the step of supplying includes:

establishing selected requirements for activating and controlling a process which enables said copy protection and which reports the corresponding usage thereof; and

providing copy protection control software in response to the selected requirements, which software provides said copy protection controls to activate and control the copy protection process and the usage reports.

3. The method of claim 2 wherein the step of establishing includes:

establishing requirements which differentiate between digital-to-digital and digital-to-analog copy protection conditions, which determine a copy protection process operating mode and configuration, and which ensure that there is only authorized access to the copy protection process.

4. The method of claim 2 wherein the step of providing includes:

generating the commands in the form of a bit pattern in response to the copy protection control software; and

said commands including a first bit pattern which enables real time on/off/mode control, and a second bit pattern which determines a programmable copy protection configuration.

5. The method of claim 4 including the step of:

receiving the transmitted first and second bit patterns to activate the copy protection and to control and reconfigure the copy protection process respectively in response thereto; and wherein the anticopy waveforms are applied to the signal material to provide the copy protection.

6. The method of claim 2 including the step of:

limiting access to the steps of establishing and providing to prevent unauthorized access to the application of the copy protection process or to the copy protection control software which activates and controls the process.

7. The method of claim 2 wherein the step of applying includes:

storing the copy protection controls in memory at a service provider receiving facility; and

storing control data in memory at a signal material receiving facility, which stored control data is responsive to the commands to activate, control and reconfigure the stored copy protection process.

8. The method of claim 2 including the step of:

collecting periodic copy protection activity information including copy protection activation status such pay-per-view and pay-to-tape number of signal material events watched.

9. The method of Claim 8 including the steps of generating reports which include the number of accessing receiving facilities, the rights holder of the signal material events, the number of total events watched, and corresponding billing information.

10. The method of claim 2 wherein the step of applying includes:

modifying a selected synchronizing signal in a corresponding blanking interval of a television line in response to said commands to degrade a subsequent

decoding of the synchronizing signal in the event that a recording is made of the corresponding signal material.

11. The method of claim 2 wherein the signal material is a video analog or digital signal.

12. Apparatus for controlling copy protection of proprietary signal material transmitted via digital delivery networks, wherein a service provider enables a copy protection process which prevents unauthorized copying of the signal material by consumers, the apparatus comprising:

a control/billing center for supplying copy protection control signals as directed by the service provider;

means for transmitting selected commands in response to the copy protection control signals to selectively control the copy protection process; and

means located with each consumer for applying the copy protection process to the signal material in response to the transmitted selected commands to prevent or allow viewing or copying of the signal material.

13. The apparatus of claim 12 wherein the copy protection control signals of the service provider include:

a mode command for activating the box means; and

a configuration bit pattern for determining the copy protection process's operating configuration.

14. The apparatus of claim 13 wherein the copy protection control signals include an access password for identifying that a service provider's authorized personnel have access to and control of the copy protection process.

15. The apparatus of claim 13 wherein the box means includes a set-top box having encoder means containing a copy protection circuit adapted to add anticopy signals to the signal material in response to the command signals.

16. The apparatus of claim 15 wherein the set-top box includes:

memory means for storing the copy protection configuration and/or copy protection mode; and said encoder means including means for receiving the mode command and the configuration bit pattern and for controlling the activation and configuration of the stored copy protection process in response to the command and bit pattern.

17. The apparatus of claim 15 wherein the set-top box includes software for returning usage data back to the service provider's control/billing center, said usage data being used by the service provider to bill the consumers and to provide a report of the usage and corresponding license fees.

18. The apparatus of claim 13 wherein the signal material is a pay-per-view or pay-to-tape video analog or digital signal.

19. The apparatus of claim 12 wherein the control/billing center includes:

instructional information establishing requirements for activating and controlling the copy protection process and for reporting the copy protection activity; and

wherein the service provider supplies copy protection control software commensurate with said requirements, and said copy protection control signals in response to the control software.

20. A method of providing copy protection of signal material transmitted via a digital delivery network, wherein a service provider enables a copy protection process via a set-top box located at a consumer's facility, comprising the steps of:

supplying selected control bit patterns from the service provider to the consumer's facility via the digital delivery network;

storing a copy protection configuration in the set-top box;

receiving the control bit pattern in said set-top box; and

applying the copy protection process to the transmitted signal material in response to the control bit pattern each time a selection of the material is made at the consumer's facility to prevent or allow the selected signal material to be copied.

21. The method of claim 20 wherein the step of supplying includes:

developing copy protection control software which describes selected control signals for applying the copy protection process to the signal material and for returning to the service provider usage data indicative of the signal material selected at the consumer's facility;

generating said selected control bit patterns in response to the copy protection control software; and

transmitting said selected control bit patterns to the set-top box of the consumer's facility when the consumer joins the delivery network and thereafter on a prescribed routine basis.

22. The method of claim 21 including the steps of:

storing in the set-top box copy protection application software which activates and controls the copy protection process; and

enabling the stored application software in response to the transmitted control bit pattern to selectively activate and/or modify the configuration of the copy protection process.

23. The method of claim 22 including the steps of:

modifying the configuration control bit pattern commensurate with a desired change in the copy protection process; and

transmitting the modified configuration control bit pattern to the set-top-box to effect the change in the copy protection process.

24. The method of claim 21 including the steps of:

storing consumer information in the set-top box which is indicative of viewing and/or copying options desired at the consumer's facility; and

comparing the control bit pattern to the stored consumer's information in the set-top box when a selection of the signal material is made to determine if the consumer is authorized to view only and/or to copy the material.

25. The method of claim 20 wherein:

the signal material is a pay-per-view (PPV) or pay-to-tape (PTT) signal; and

the step of supplying includes establishing selected requirements for activating and controlling the PPV and PTT copy protection process and for reporting the corresponding usage activity of the process to the service provider; and

providing copy protection control software in response to the selected requirements, which software provides said control bit pattern to activate, control and modify the PPV and PTT copy protection process.

26. The method of claim 25 including the step of:

providing limited access to the steps of establishing and providing to prevent unauthorized access to the control of the copy protection process or to the copy protection control software.

27. The method of claim 25 wherein the signal material is a pay-per-view or pay-to-tape video analog or digital signal.

28. The method of claim 27 wherein the step of applying includes:

modifying a selected synchronizing signal in a corresponding blanking interval of a television line in response to said control bit pattern to degrade any subsequent decoding of the synchronizing signal when an unauthorized attempt is made to view or copy the pay-per-view signal.

29. A method of providing copy protection of signal material transmitted via a digital delivery network, wherein a service provider enables a copy protection process via set-top boxes located at consumers' facilities, comprising the steps of:

establishing selected requirements for activating, controlling and modifying a copy protection process for the signal material and for reporting the corresponding usage thereof;

providing copy protection control software in response to the selected requirements;

generating via the control software, mode and configuration control bit patterns which enable real time on/off mode control and programmable copy protection process configuration control respectively;

transmitting the mode control and configuration control code words to the set-top boxes;

selectively applying the copy protection process to the transmitted signal material in response to the transmitted mode bit pattern each time a selection of the signal material is made via the set-top boxes to prevent or allow the selected signal material to be viewed or copied.

30. The method of claim 29 including the steps of:

storing the application software in the set-top boxes; receiving and writing the mode bit pattern in the set-top boxes; and

wherein the stored application software responds to the transmitted mode bit pattern to activate, control and modify the copy protection process as defined by the configuration control bit pattern.

31. The method of claim 30 wherein the set-top box is functionally locked including: downloading via the service provider a selected bit pattern or software adapted to functionally unlock the set top box.

32. The method of claim 30 wherein the set-top box is functionally locked including activating at the service provider's facility selected software adapted to functionally unlock the set-top box

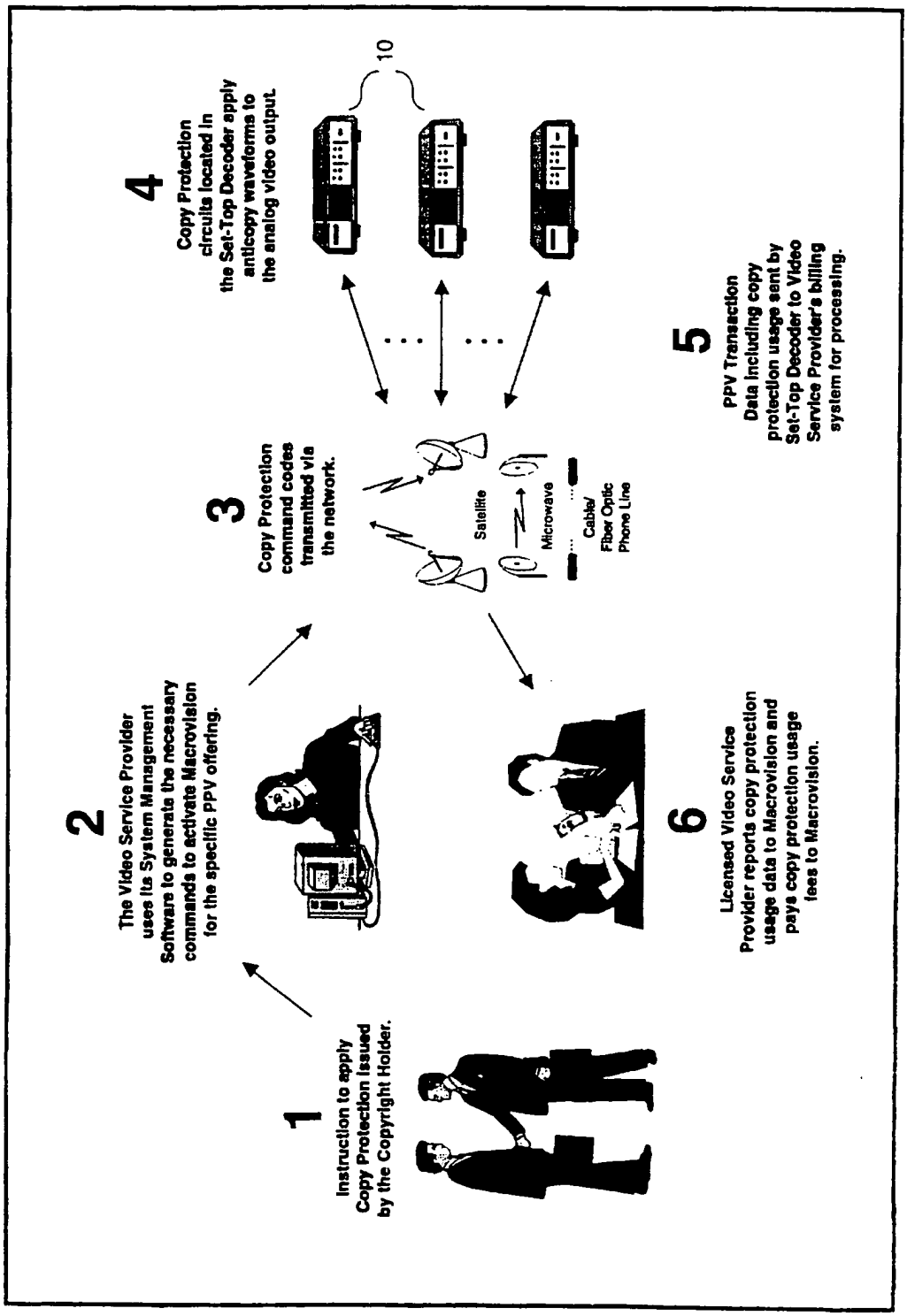

1
Instruction to apply Copy Protection issued by the Copyright Holder.
2
The Video Service Provider uses its System Management Software to generate the necessary commands to activate Macrovision for the specific PPV offering.
3
Copy Protection command codes transmitted via the network.
Satellite
Microwave
Cable/ Fiber Optic Phone Line
4
Copy Protection circuits located in the Set-Top Decoder apply anticopy waveforms to the analog video output.
10
5
PPV Transaction Data including copy protection usage sent by Set-Top Decoder to Video Service Provider's billing system for processing.
6
Licensed Video Service Provider reports copy protection usage data to Macrovision and pays copy protection usage fees to Macrovision.

FIG. 1

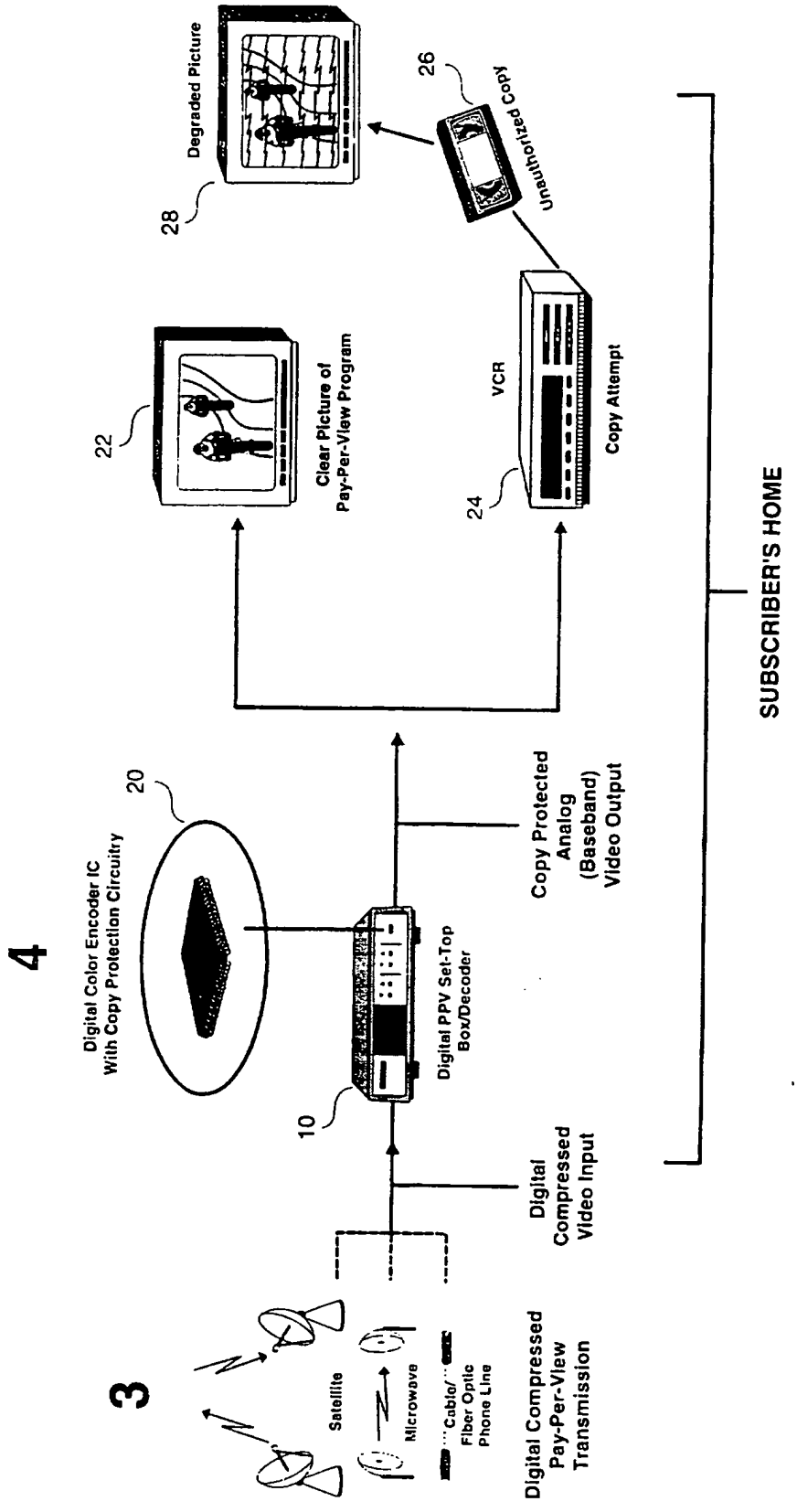

Degraded Picture
26
Unauthorized Copy
28
22
Clear Picture of Pay-Per-View Program
VCR
24
Copy Attempt
SUBSCRIBER'S HOME
20
Digital Color Encoder IC With Copy Protection Circuitry
Copy Protected Analog (Baseband) Video Output
10
Digital PPV Set-Top Box/Decoder
Digital Compressed Video Input
4
3
Satellite
Microwave
Cable/ Fiber Optic Phone Line
Digital Compressed Pay-Per-View Transmission

FIG. 2

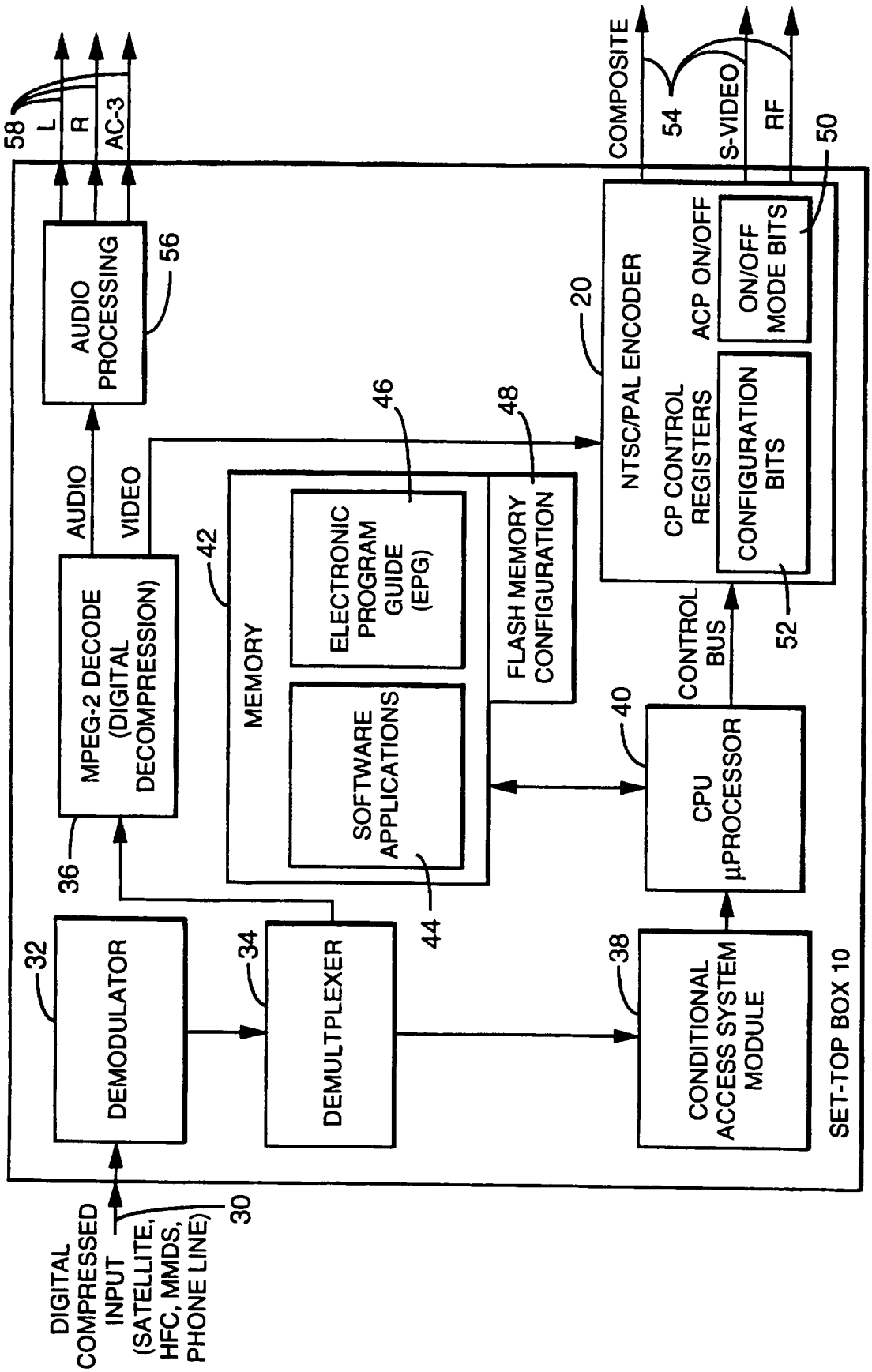

DIGITAL COMPRESSED INPUT (SATELLITE, HFC, MMDS, PHONE LINE)
30
SET-TOP BOX 10
DEMODULATOR
32
DEMULTPLEXER
34
MPEG-2 DECODE (DIGITAL DECOMPRESSION)
36
AUDIO
VIDEO
AUDIO PROCESSING
56
L
R
AC-3
58
MEMORY
42
ELECTRONIC PROGRAM GUIDE (EPG)
46
SOFTWARE APPLICATIONS
44
FLASH MEMORY CONFIGURATION
48
CONDITIONAL ACCESS SYSTEM MODULE
38
CPU µPROCESSOR
40
CONTROL BUS
NTSC/PAL ENCODER
20
CP CONTROL REGISTERS
CONFIGURATION BITS
52
ACP ON/OFF
ON/OFF MODE BITS
50
COMPOSITE
54
S-VIDEO
RF

FIG. 3

INTERNATIONAL SEARCH REPORT

Int. .onal Application No
PCT/US 97/05257

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04N5/913

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)
IPC 6 H04N

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 691 787 A (SONY CORPORATION) 10 January 1996 see the whole document ---	1,2,5, 11,12, 15,18, 20,21, 27,29
A	US 5 315 448 A (RYAN) 24 May 1994 cited in the application see the whole document -----	1,4, 10-12, 18,20, 21,27-29

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in 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 document but 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

1

Date of the actual completion of the international search	Date of mailing of the international search report
13 August 1997	22.08.97
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Authorized officer Verleye, J

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/US 97/05257

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 691787 A	10-01-96	CN 1115150 A	17-01-96
		JP 8077706 A	22-03-96
US 5315448 A	24-05-94	AU 677999 B	15-05-97
		AU 6359394 A	11-10-94
		BR 9406002 A	02-01-96
		CA 2158021 A	29-09-94
		CN 1122177 A	08-05-96
		EP 0689751 A	03-01-96
		HU 73989 A	28-10-96
		JP 8507912 T	20-08-96
		PL 310623 A	27-12-95
		WO 9422266 A	29-09-94

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/08, G07C 9/00 — A1

(11) International Publication Number: **WO 97/25800**

(43) International Publication Date: 17 July 1997 (17.07.97)

(21) International Application Number: PCT/CA96/00847

(22) International Filing Date: 17 December 1996 (17.12.96)

(30) Priority Data:
08/584,375 8 January 1996 (08.01.96) US

(71) Applicant: MYTEC TECHNOLOGIES INC. [CA/CA]; Suite 430, 10 Gateway Boulevard, Don Mills, Ontario M3C 3A1 (CA).

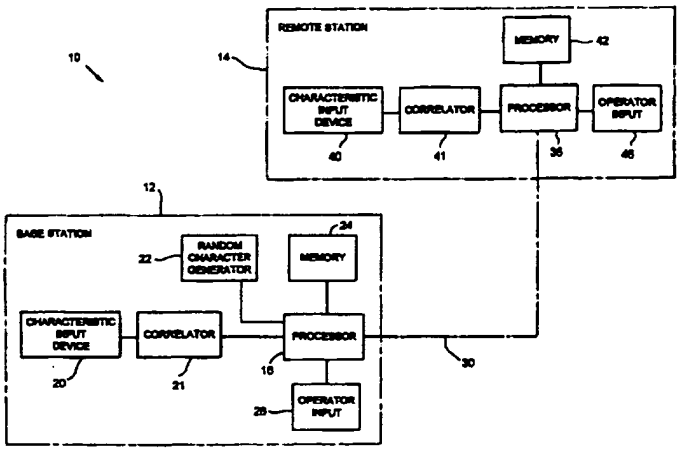
(72) Inventors: TOMKO, George, J.; Mytec Technologies Inc., Suite 430, 10 Gateway Boulevard, Don Mills, Ontario M3C 3A1 (CA). STOIANOV, Alexei; Mytec Technologies Inc., Suite 430, 10 Gateway Boulevard, Don Mills, Ontario M3C 3A1 (CA).

(74) Agent: FAGGETTER, Ronald, D.; Fetherstonhaugh & Co., Suite 2300, 439 University Avenue, P.O. Box 39, Station P, Toronto, Ontario M5S 2S6 (CA).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD FOR SECURE DATA TRANSMISSION BETWEEN REMOTE STATIONS



(57) Abstract

A method for permitting the secure handling of data between two remote stations firstly involves the generation of an encrypted decryption key which is based on a fingerprint information signal from a user of a first station, a fingerprint information signal from a user of a second station, and a key representing function derived from a random key. The encrypted decryption key is of the type with the property that when it is written to a spatial light modulator (SLM) of an optical correlator, the output of the correlator is similar when input with either one of the fingerprint information signals. The encrypted key is then stored at both stations. Thereafter a message encrypted with the key may be decrypted at either station by retrieving the encrypted key, writing the encrypted key to a filter of an optical correlator, inputting one of the fingerprint information signals to the correlator in order to allow recovery of the decryption key, and applying the decryption key to the encrypted message.

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.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

METHOD FOR SECURE DATA TRANSMISSION BETWEEN REMOTE STATIONS

Background of the Invention

1. Field of the Invention

The present invention provides a method for permitting the secure passing of data between two remote stations.

2. Background of the Invention

While use of the internet has increased rapidly, concerns for the privacy and security of data transferred over the internet have remained. The present invention seeks to provide a method for permitting the secure handling of data between remote stations, such as remote computers hooked to the internet.

Summary of the Invention

In accordance with the present invention, there is provided a method for permitting the secure passing of data between two remote stations, comprising the steps of: obtaining from a user of a first of two remote stations, a first characteristic information signal; obtaining from a user of a second of two remote stations, a second characteristic information signal; generating a sequence of random characters to obtain a random key; obtaining a key function which represents said key; obtaining a Fourier transform of said key representing function; obtaining at least one encrypted version of said key based on said Fourier transform of said key representing function, and a least one of said first characteristic information signal and said second characteristic information signal such that said key may be recovered by writing said at least one encrypted version of said encrypted key to a spatial light modulator (SLM) of an optic correlator and inputting either one of said first characteristic information signal and said second characteristic information signal

to said optic correlator; storing said at least one encrypted version of said key at each of said first station and said second station, whereby thereafter any message encrypted in such a way that it may be decrypted by said key may be decrypted at either of said two remote stations by retrieving said stored encrypted key, writing said at least one encrypted version of said encrypted key to a spatial light modulator (SLM) of an optic correlator and inputting either one of said first characteristic information signal and said second characteristic information signal to said optic correlator.

In accordance with another aspect of the present invention, there is provided a method for the secure handling of data between two remote stations, comprising the steps of: at a base station, encrypting a message such that said message may be decrypted by a decryption key; passing said message to a remote station; at said remote station, obtaining from a user of said remote station a remote station user optical characteristic information signal; retrieving from storage an encrypted version of said decryption key, said encrypted decryption key having the property that when it is written to a SLM of an optical correlator, the output of said correlator is similar when input with either one of said remote station user characteristic information signal or a base station user optical characteristic information signal; writing a remote station optical correlator with said encrypted decryption key; inputting said remote station correlator with a Fourier transform of said remote station user optical characteristic information signal; regenerating said decryption key from an output of said remote station correlator; and decrypting said message with said decryption key.

Brief Description of the Drawings

Figure 1 is a schematic view of a system for use in the secure handing of data between two remote stations made in accordance with this invention,

figure 2 is a schematic detail of a portion of figure 1, and

figure 2A is a schematic representation of an alternative embodiment for a portion of figure 2.

Detailed Description of the Preferred Embodiments

Turning to figure 1, a system indicated generally at 10 for permitting the secure passing of data between two remote stations, comprises a base station indicated generally at 12 and a remote station indicated generally at 14. The base station comprises a processor 16 linked to a correlator 21, a random character generator 22, a memory 24, and an operator input device 26. The correlator 21 is optically linked to a characteristic input device 20. The processor 16 of the base station 12 is connected for two-way communication with a processor 36 of remote station 14 on line 30. The processor 36 of the remote station is linked to a correlator 41, a memory 42, and an operator input device 46. The correlator 41 is optically linked to a characteristic input device 40.

The characteristic input device 20 and correlator 21 of base station 12 are detailed in figure 2. Turning to figure 2, input device 20 comprises a source of coherent light 222 and input prism 224 with an optical output 225 to correlator 21. The correlator 21 comprises a Fourier transform lens 228, a full-complex spatial light modulator (SLM) 230, an inverse Fourier transform lens 232, a CCD camera 234 with an A\D convertor 236 outputting to processor 16 on line 237. The processor outputs to the input of SLM 230 on line 260. The characteristic input device 40 and correlator 41 of remote station 14 may be identically constructed.

System 10 is used, firstly, to develop an encrypted version of a message decryption key at the base station which may be transmitted to the remote station without concern for privacy and, subsequently, to encrypt messages at either of the stations for transmission to other of the stations where they may be decrypted.

(i) Developing an encrypted decryption key

Assuming the user of base station 12 wishes to communicate in a secure fashion with the user of remote station 14, the user of the base station first agrees upon a temporary secret key with the user of the remote station. This secret key can, for example,

be based on a Diffie-Hellman key derivation, an exponential key derivation scheme or public key system. The user of the remote station then utilizes input device 40 to develop an information signal impressed with characteristics peculiar to the remote station user. With the input device 40 and correlator 41 configured as shown in figure 2, the remote station user activates the light source of the input device and causes the processor 36 to make the SLM of the correlator transparent so that the correlator is effectively bypassed. Next the remote station user places his finger on the input prism creating an optical signal impressed with characteristics of the fingerprint of the user. This optical characteristic signal is imaged at the camera. This characteristic information signal is then digitized and passed to the processor 36. The previously agreed upon secret key is used to encode the digitized fingerprint and this encrypted fingerprint may then be passed to the base station 12 on line 30.

At the base station 12, referencing figure 2, the base station user may activate light source 222 and cause processor 16 to make SLM 230 transparent. The base station user may then place his fingerprint 226 on the input prism so that a fingerprint (characteristic) information signal is imaged at the camera 234. The digitized version of this signal is then passed to processor 16. Returning to figure 1, the processor decrypts the fingerprint information signal from the remote station utilizing the previously agreed upon method to generate a temporary secret key, which may either be derived by processor 16 and stored in memory 24 or input directly from the operator input 26. Next the processor 16 numerically determines spatial Fourier transforms of the remote station fingerprint information signal and the base station fingerprint information signal.

The processor now prompts random character generator 22 to generate a sequence of random characters which will comprise a decryption key. The processor 16 then develops a key function which represents the key. For example, the key representing function could be developed by applying each character of the decryption key as a coefficient to a set of normalized orthogonal basis functions, preferably, delta-shaped functions. The processor then numerically calculates a Fourier transform of the key representing function.

Next, the processor obtains an encrypted version of the decryption key. In the first embodiment of the invention, this step includes developing a composite filter based on the remote station fingerprint information signal, the base station fingerprint information signal, and the key representing function. This composite filter has the property that when it is written to the SLM, the output of the correlator is similar when input with either the remote station fingerprint information signal or the base station fingerprint information signal. Preferably, this output is a set of narrow peaks, the positions of which correspond to the maxima of the delta-shaped basis functions. Methods of obtaining a composite filter with these properties are known to those skilled in the art and described in, for example, an article entitled "Tutorial Survey of Composite Filter Designs for Optical Correlators" by B.V.K. Vijaya Kumar, Applied Optics, Volume 31, No. 23, pages 4773 to 4801. Briefly, the composite filter may be constructed as a linear combination of the complex conjugate Fourier transforms of the remote station fingerprint information signal and the base station fingerprint information signal multiplied by the Fourier transform of the key representing function. The coefficients of the linear combination are determined from a set of equations derived in accordance with certain criteria.

To illustrate the process of composite filter development, let us consider a case of two fingerprints, $f_1(x)$ and $f_2(x)$, where $f_1(x)$ and $f_2(x)$ are the base and the remote station fingerprint information signals, respectively (we use a one-dimensional spatial coordinate system for simplicity). The Fourier transforms of these signals are $F_1(q)$ and $F_2(q)$ respectively, where q is a coordinate in a Fourier domain.

The key representing function may be written as

$$k(x) = \sum_{n=1}^{N} k_n \, \delta(x - x_n) \ ,$$

where $\delta()$ is a delta-function; x_n are the coordinates of the narrow peaks and N is the number of the peaks; k_n are numerical coefficients. The Fourier transform of the key representing function is

$$K(q) = \sum_{n=1}^{N} k_n \exp(- iqx_n)$$

The composite filter, H(q), may be presented in the form

$$H(q) = K(q) (C_1 F_1^*(q) + C_2 F_2^*(q)) ,$$

where coefficients C_1 , C_2 should be determined; " * " means complex conjugation. If this filter is put on a SLM and the SLM is illuminated with the signal $f_1(x)$, we will get a correlation function, $B_1(x)$, at the output of the correlator, and a correlation function $B_2(x)$ for the signal $f_2(x)$. For the correlation functions we have:

$$B_1(x) = (1/2\pi)C_1 \sum_{n=1}^{N} k_n \int F_1(q)F_1^*(q) \exp(iq(x-x_n)) dq +$$

$$(1/2\pi)C_2 \sum_{n=1}^{N} k_n \int F_1(q)F_2^*(q) \exp(iq(x-x_n)) dq ,$$

$$B_2(x) = (1/2\pi)C_1 \sum_{n=1}^{N} k_n \int F_2(q)F_1^*(q) \exp(iq(x-x_n)) dq +$$

$$(1/2\pi)C_2 \sum_{n=1}^{N} k_n \int F_2(q)F_2^*(q) \exp(iq(x-x_n)) dq$$

Substituting $x = x_n$, $n = 1, 2, \ldots N$ into the equations and setting, for example, the sums $\Sigma B_1(x_n)$, $\Sigma B_2(x_n)$ equal to certain values, we can obtain as many algebraic equations as necessary to find the unknown variables C_1 , C_2 , k_n and to develop the composite filter. To make sure that the number of the equations equals the number of the unknown coefficients, one can use different criteria. For example, a sum (or a sum of squares, or a product, etc.) of the heights of the output narrow peaks is set equal to a certain value. In another embodiment, the height of each peak is set equal to a certain value, but in this case both users (i.e. at the base station and at the remote station) record a few fingerprint information signals, that is, the number of the signals equals or exceeds the number of the peaks in the key representing function.

In the second embodiment of the invention, the step of obtaining an encrypted version of the decryption key includes dividing the Fourier transform of the key representing function by the Fourier transform of the base station fingerprint information signal to obtain a first filter, and dividing the Fourier transform of the key representing function by the Fourier transform of the remote station fingerprint information signal to obtain a second filter. A concatenation of the two filters can now be stored and this yields the encrypted version of the decryption key for both base and remote station fingerprint information signal.

The encrypted version of the decryption key may be stored in memory 24. Also, because the decryption key is encrypted, it may be passed to the remote station on line 30 and will remain secure even if intercepted. The remote station stores the received encrypted decryption key in its memory 42.

In a third embodiment, the decryption key generated by the base station is encrypted by the temporary secret key and transmitted to the remote station over line 30. Each station may then develop a key representing function using the techniques aforedescribed. Then each station develops a filter based on the developed key representing function and the characteristic information signal of that station, again using techniques as aforedescribed. A number of alternative approaches for generating both key representing functions and filters are described in U.S. patent application No. 08/508,978 filed July 28, 1995 and PCT/CA95/00509 filed Sept. 6, 1995, the disclosures of which are incorporated herein by reference.

(ii) Sending messages

Once an encrypted version of the decryption key is present at both the base and remote stations, encrypted messages may be sent from either station to the other and decrypted by the recipient station. For example, if the base station user wished to send an encrypted message to the remote station, he could obtain the decryption key by applying his fingerprint to the characteristic input device 20 and prompting processor 16 to write

SLM 230 with the encrypted decryption key. This will return the key representing function at camera 234 from which the key can be extracted by the processor. The base station user may then input a message by way of operator input 26 which message may be encrypted with the decryption key and the encrypted message sent on line 30 to the remote station.

In the second embodiment of the invention, the processor 16 writes to the SLM each of the previously concatenated two filters of the encrypted decryption key either in sequence or simultaneously. If the fingerprint is the same as was used at the base station during developing the encrypted decryption key, the camera 234 will register a set of narrow peaks in the case of the first filter and a random pattern in the case of the second filter. The positions of the peaks correspond to the maxima of the delta-shaped basis functions and, thus, determine the decryption key.

At the remote station, the remote user may prompt processor 36 to retrieve the encrypted decryption key from memory and write same to the filter of correlator 41. Next this user may input his fingerprint to characteristic input device 40. This will cause the correlator to return the key representing function to the processor 36 so that the processor may determine the key from this function. The decryption key may then be used to decrypt the incoming message.

In a similar fashion, the remote station user could encrypt a message by obtaining the decryption key in the manner aforedescribed and inputting a message to be encrypted at operator input 46. The encrypted message could then be decrypted by the base station in the same fashion as the remote station decrypts messages passed in the other direction.

The only difference between the base station and the remote station is the presence of random character generator 22 at the base station. The roles of these stations may be easily reversed by including a random character generator at the remote station.

As described, the subject invention is suitable for use in secure communications between two computers where the decryption key is released only by applying the fingerprint of the proper user to an input device. Of course, the characteristic input device may be modified to accept other body parts of a user so that a different biometric, such as a vein structure, or an iris pattern of a user is input.

Where the base station user is an entity such as a corporation or other organization, it may not be desirable to have access controlled by a biometric of a single individual. Figure 2a illustrates an alternative characteristic input device 300 which may be used in such instance. Turning to figure 2a, input device 300 comprises a SLM 324 held in place by holder 318 in the light path of coherent light source 222. Processor 16 writes a corporation's proprietary characteristic information (PCI) on the SLM 324 which impresses the light beam with selected characteristics such that a characteristic information signal is generated. When not in use, the PCI would be stored in a secure location in the corporation.

If the base station is sufficiently secure, it may be preferred to store an unencrypted version of the decryption key in memory 24. In such instance, correlator 21 becomes unnecessary and may be replaced with an imaging lens, CCD camera, and A/D convertor. The only use made of the base station characteristic input device would then be during generation of the encrypted decryption key.

System 10 has been described in conjunction with a decryption key which is a symmetric private key. Alternatively, the decryption key could be the private key for public key encrypted messages.

Certain parts of the subject invention have been described as using Fourier Transforms which are an expansion on a set of complex exponential orthogonal basis functions. Alternatively, other orthogonal expansions on a set of basis function can also be used such as Walsh and wavelet functions.

Other modifications will be apparent to those skilled in the art and, therefore, the invention is defined in the claims.

WHAT IS CLAIMED IS:

1. A method for permitting the secure passing of data between two remote stations, comprising the steps of:

- obtaining from a user of a first of two remote stations, a first characteristic information signal;
- obtaining from a user of a second of two remote stations, a second characteristic information signal;
- generating a sequence of random characters to obtain a random key;
- obtaining a key function which represents said key;
- obtaining a Fourier transform of said key representing function;
- obtaining at least one encrypted version of said key based on said Fourier transform of said key representing function, and a least one of said first characteristic information signal and said second characteristic information signal such that said key may be recovered by writing said at least one encrypted version of said encrypted key to a spatial light modulator (SLM) of an optic correlator and inputting either one of said first characteristic information signal and said second characteristic information signal to said optic correlator;
- storing said at least one encrypted version of said key at each of said first station and said second station, whereby thereafter any message encrypted in such a way that it may be decrypted by said key may be decrypted at either of said two remote stations by retrieving said stored encrypted key, writing said at least one encrypted version of said encrypted key to a spatial light modulator (SLM) of an optic correlator and inputting either one of said first characteristic information signal and said second characteristic information signal to said optic correlator.

2. The method of claim 1 wherein the step of obtaining a first characteristic information signal comprises obtaining an optical beam modulated with a biometric image of a first body part of said user of said first station, registering said optical beam in a two-dimensional plane and digitizing said registered optical beam.

3. The method of claim 2 wherein the step of obtaining a second characteristic information signal comprises obtaining an optical beam modulated with a biometric image of a second body part of said user of said second station, registering said optical beam in a two-dimensional plane and digitizing said registered optical beam.

4. The method of claim 3 wherein the step of obtaining said key representing function comprises obtaining normalized orthogonal basis functions and, for each basis function, applying a character of said key as a co-efficient.

5. The method of claim 4 wherein said first characteristic information signal is obtained at said first station and including the steps of:
- encrypting said digitized registered optical beam modulated with a biometric of a first body part with a pre-selected key to obtain an encrypted first biometric signal;
- sending said encrypted first biometric signal to said second station;
- utilizing said pre-selected key at said second station to decrypt said encrypted biometric of said first body part; and
- obtaining said encrypted key at said second station.

6. The method of claim 4 wherein said key representing function is obtained at said first station and including the steps of:
- encrypting said key representing function with a pre-selected key to obtain an encrypted key representing function;
- sending said encrypted key representing function to said second station;
- utilizing said pre-selected key at said second station to decrypt said encrypted key representing function; and
- obtaining said encrypted key at said second station.

7. A method for the secure handling of data between two remote stations, comprising the steps of:
- at a base station, encrypting a message such that said message may be decrypted by a decryption key;

- passing said message to a remote station;
- at said remote station,
- obtaining from a user of said remote station a remote station user optical characteristic information signal;
- retrieving from storage an encrypted version of said decryption key, said encrypted decryption key having the property that when it is written to a SLM of an optical correlator, the output of said correlator is similar when input with either one of said remote station user characteristic information signal or a base station user optical characteristic information signal;
- writing a remote station optical correlator with said encrypted decryption key;
- inputting said remote station correlator with a Fourier transform of said remote station user optical characteristic information signal;
- regenerating said decryption key from an output of said remote station correlator; and
- decrypting said message with said decryption key.

8. The method of claim 7 wherein the step of encrypting a message at said base station comprises encrypting said message utilizing said decryption key.

9. The method of claim 8 wherein the step of encrypting a message at said base station comprises the steps of:
- obtaining from a base station user said base station optical characteristic information signal, such that said base station optical characteristic signal is impressed with characteristics of a body part of said base station user;
- retrieving from storage said encrypted version of said decryption key;
- writing a base station optical correlator with said encrypted decryption key;
- inputting said base station correlator with said base station user optical characteristic information signal;
- regenerating said decryption key from an output of said base station correlator; and
- encrypting said message with said regenerated decryption key.

10. The method of claim 4 wherein said step of obtaining at least one encrypted version of said key is based on both said first characteristic information and said second characteristic information signal.

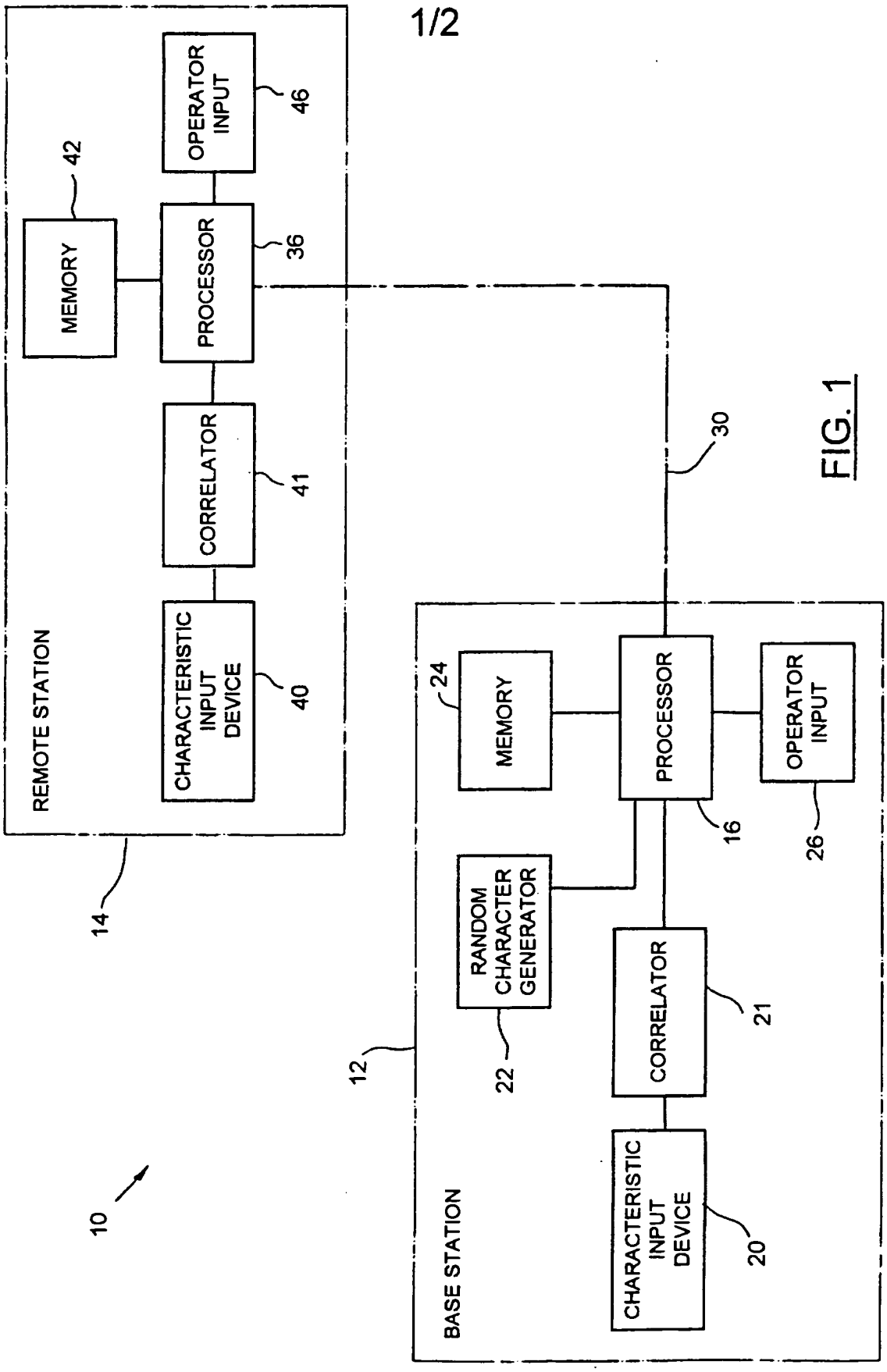


FIG. 1

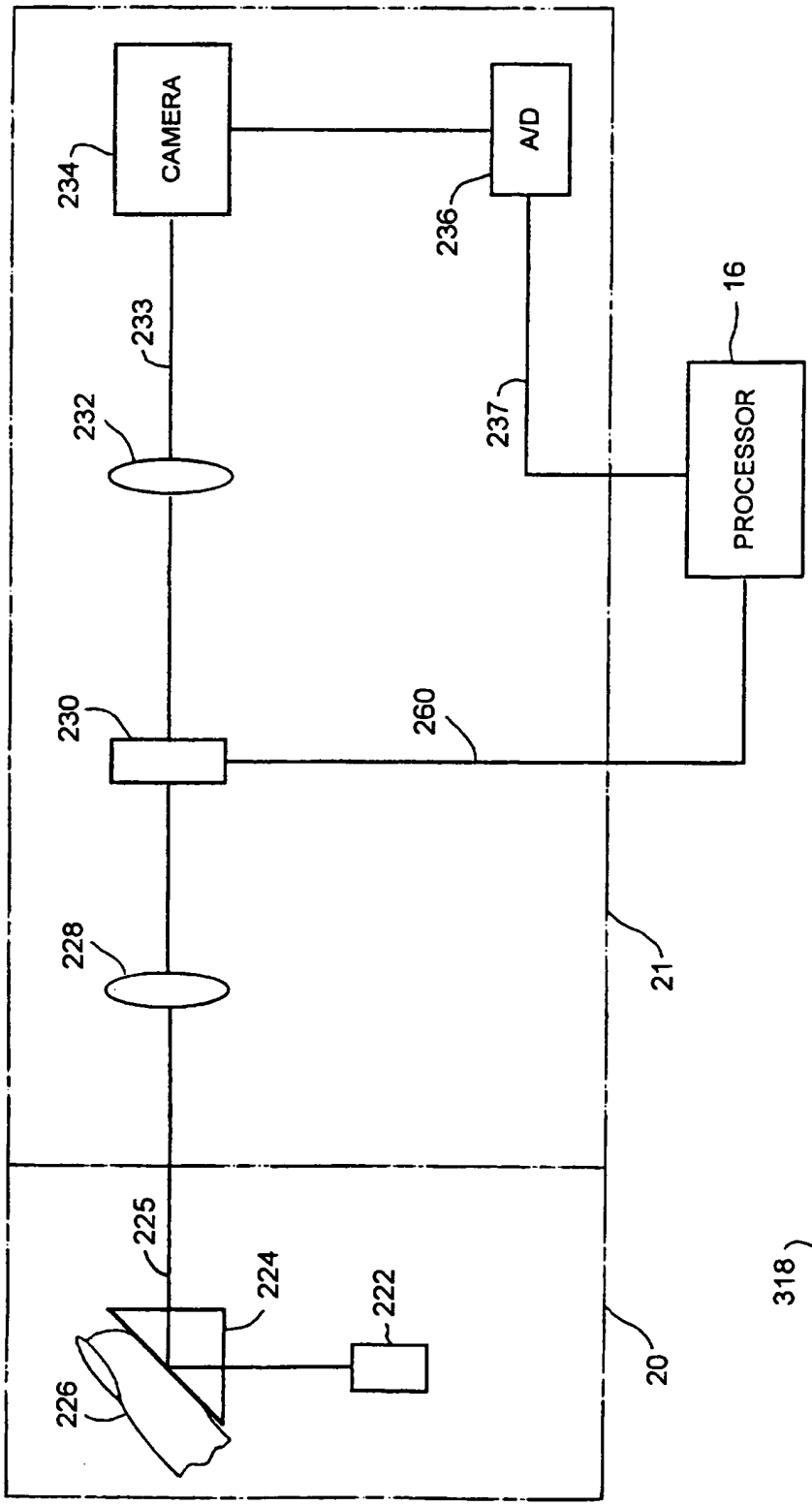


FIG. 2

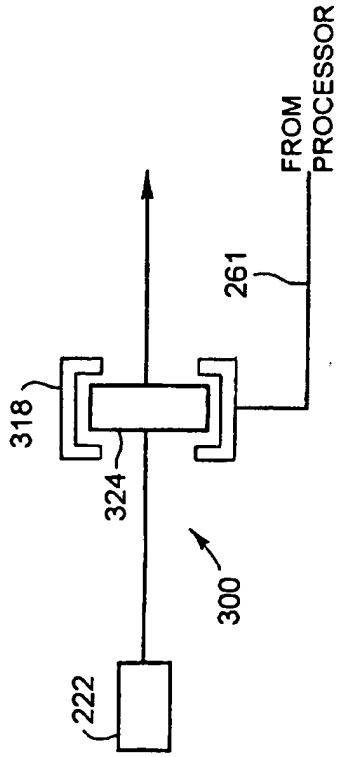


FIG. 2A

INTERNATIONAL SEARCH REPORT

Interna[illegible]l Application No
PCT/[illegible]A 96/00847

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L9/08 G07C9/00

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)
IPC 6 H04L G07C

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4 532 508 A (RUELL) 30 July 1985 see column 1, line 57 - column 2, line 2 see column 2, line 18 - line 30 see column 3, line 44 - column 4, line 41 ---	1,2,7
A	ADVANCES IN CRYPTOLOGY, PROCEEDINGS OF CRYPTO 82, SANTA BARBARA, CA, USA, 23-25 AUG. 1982, ISBN 0-306-41366-3, 1983, NEW YORK, NY, USA, PLENUM, USA, pages 219-229, XP002029301 MUELLER-SCHLOER C ET AL: "Cryptographic protection of personal data cards" see page 226, line 2 - page 228, last line ---	1,5
A	US 5 095 194 A (BARBANELL) 10 March 1992 see column 3, line 43 - column 5, line 24 see column 6, line 28 - line 59 see column 7, line 35 - column 8, line 47 --- -/--	1,7

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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
11 April 1997	28. 04. 97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+ 31-70) 340-3016	Holper, G

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	DE 42 43 908 A (GAO) 30 June 1994 see column 2, line 32 - line 48 see column 3, line 30 - line 51 see column 4, line 18 - column 5, line 17 ---	1,7
A	US 5 050 220 A (MARSH ET AL.) 17 September 1991 see column 5, line 18 - column 6, line 24 -----	7

1

INTERNATIONAL SEARCH REPORT

.rmation on patent family members

Interna al Application No
PCT/CA 96/00847

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 4532508 A	30-07-85	EP 0121222 A	10-10-84
US 5095194 A	10-03-92	NONE	
DE 4243908 A	30-06-94	NONE	
US 5050220 A	17-09-91	NONE	

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04N 7/16, 7/167

A1

(11) International Publication Number: **WO 98/43426**

(43) International Publication Date: 1 October 1998 (01.10.98)

(21) International Application Number: PCT/EP97/02108

(22) International Filing Date: 25 April 1997 (25.04.97)

(30) Priority Data:
97400650.4 21 March 1997 (21.03.97) EP
(34) Countries for which the regional or international application was filed: FR et al.

(71) Applicant *(for all designated States except US)*: CANAL+ SOCIETE ANONYME [FR/FR]; 85/89, quai André Citroën, F–75711 Paris Cedex 15 (FR).

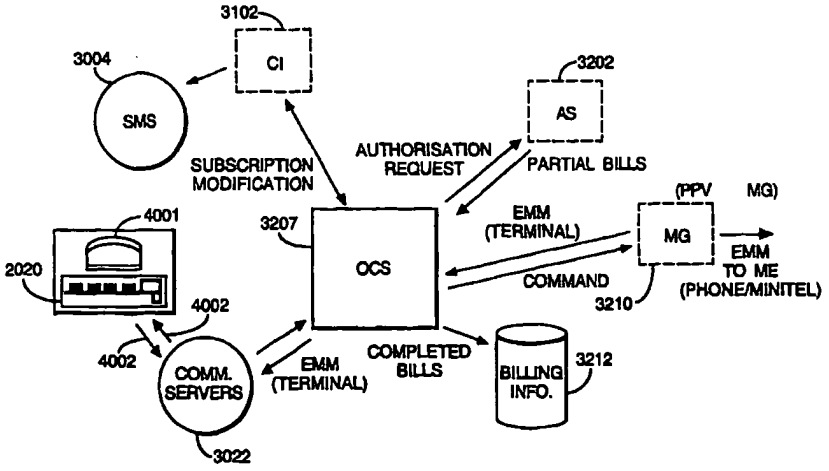
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BAYASSI, Mulham [FR/FR]; 30, rue de Chambéry, F–75015 Paris (FR). DE LA TULLAYE, Pierre [FR/FR]; 7, allée Marcel Jouhandeau, F–92500 Rueil Malmaison (FR). JEZEQUEL, Jean–François [FR/FR]; 35, rue du Commandant Kieffer, F–95240 Cormeille en Parisis (FR).

(74) Agent: COZENS, Paul, Dennis; Mathys & Squire, 100 Grays Inn Road, London WC1X 8AL (GB).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: BROADCAST AND RECEPTION SYSTEM, AND CONDITIONAL ACCESS SYSTEM THEREFOR



(57) Abstract

A digital satellite television system has a plurality of set–top–boxes associated with a plurality of end users' television receivers, a modem and a decoder housed in each STB, a Subscriber Authorization System (SAS) incorporating or having associated therewith a plurality of communication servers, means included in the SAS for generating Electronic Managements Messages (EMM), a back channel interconnecting each of the STBs individually with the SAS, means included in the SAS and each STB so that the necessary information required to inject a relevant EMM into the system is supplied directly to the relevant communication server included in or associated with the SAS to authorise the release of the said EMM and/or means to connect the modem to the back channel and means whereby an EMM is transmissible to the decoder directly from a relevant communication server included in or associated with the SAS. Further important features are also disclosed.

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.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

BROADCAST AND RECEPTION SYSTEM, AND CONDITIONAL ACCESS SYSTEM THEREFOR

The present invention relates to a broadcast and reception system, in particular to a mass-market digital interactive satellite television system, and to a conditional access system therefor.

In particular, but not exclusively, the invention relates to a mass-market broadcast system having some or all of the following preferred features:-

- It is an information broadcast system, preferably a radio and/or television broadcast system
- It is a satellite system (although it could be applicable to cable or terrestrial transmission)
- It is a digital system, preferably using the MPEG, more preferably the MPEG-2, compression system for data/signal transmission
- It affords the possibility of interactivity.

More particularly the present invention relates to so-called pay television (or radio) where a user/viewer selects a programme/film/game to be viewed for which payment is to be made, this being referred to as a pay-per-view (PPV) or in the case of data to be downloaded a so-called pay-per-file (PPF).

With such known PPV or PPF systems a significant amount of time is required to be spent by the user/viewer in order to carry out the actions necessary to actually access the product being selected.

For example, in one known system the sequence of steps which have to be carried out are as follows:-

I) The user telephones a so-called Subscriber Management System (SMS) which in this known system includes a number of human operators which answer the subscriber's call and to whom the subscriber communicates the necessary information concerning the selected product and concerning the financial status of the subscriber

to a so-called Subscriber Authorization System (SAS) which has included in it or associated with it a plurality of communications servers.

ii) The operator at the SMS then has to check the financial status of the user before authorising the connection from the communications servers to the user's television set so that the product can be delivered and viewed by the user.

In another known system the human operator is replaced by an automatic voice server so that when the user telephones the SMS he/she hears a voice activated recording to which the user conveys the same information as I) above.

This second arrangement reduces the delay inherent in the first described arrangement which can be more easily overloaded when large numbers of users are wishing to order a product at the same time.

However, even with this second arrangement the user is involved in inputting significant information in the form of lengthy serial numbers which operation provides plenty of scope for error as well as being time consuming.

The third known arrangement involves the user making use of existing screen based systems such as MINITEL in France and PRESTEL in the United Kingdom, which systems replace the voice activated server referred to above in connection with the second arrangement. The MINITEL and PRESTEL systems themselves incorporate a modem at the consumer end.

In all these known arrangements the user is involved in the expenditure of significant time and effort in inputting all the information necessary to enable the system to in effect authorize the transmission of the chosen product to the user's television set.

In the case of a satellite television system there is a further delay involved in the user actually receiving the product selected.

In PPV and PPF systems the key element in controlling the user's access to products are so-called Entitlement Management Messages (EMM) which have to be injected into the system in order to give the user product access. More particularly the EMMs are the mechanism by which the encrypted data representative of a product is decrypted for a particular individual user.

In known satellite television systems the EMMs are transmitted to the user's televisions via the satellite link at regular intervals in the MPEG-2 data stream. Thus in the case of a particular user's EMM there can be a significant delay of perhaps several minutes before the user's next EMM transmission arrives at that user's television set.

This transmission delay is in addition to the delay referred to earlier which is inherent in the user having to manually input certain data into the system. The cumulative effect of these two delays is that it may take perhaps typically five minutes for a user to be able to gain access to the selected product.

The present invention is concerned with overcoming this problem.

In a first aspect, the present invention provides a conditional access system comprising:

means for generating a plurality of (preferably conditional access) messages; and

means for receiving the messages, said receiving means being adapted to communicate with said generating means via a communications server connected directly to said generating means.

Preferably, the message is an entitlement message for transmission (for example by broadcast) to the receiving means, said generating means being adapted to generate entitlement messages in response to data received from said receiving means.

The generating means may be arranged to transmit a message as a packet of digital

data to said receiving means either via said communications server or via a satellite transponder.

The receiving means may be connectable to said communications server via a modem and telephone link.

In a related aspect, the present invention provides a conditional access system for affording conditional access to subscribers, comprising:

a subscriber management system;

a subscriber authorization system coupled to the subscriber management system; and

a communications server; said server being connected directly to the subscriber authorization system.

The system may further comprise a receiver/decoder for the subscriber, the receiver/decoder being connectable to said communications server, and hence to said subscriber authorization system, via a modem and telephone link.

In a second aspect, the present invention provides a broadcast and reception system including a conditional access system as described above.

In a third aspect, the present invention provides a broadcast and reception system comprising:

means for generating a plurality of entitlement messages relating to broadcast programs;

means for receiving said messages from said generating means; and

means for connecting the receiving means to the generating means to receive said messages, said connecting means being capable of effecting a dedicated connection between the receiving means and the generating means.

The dedicated connection would typically be a hard-wired connection and/or a modemmed connection, with the possibility of the connection been made via a cellular

telephone system. In other words, the dedicated connection is capable of forming a channel of communication (from point to point). This is in contrast to broadcasting of information through the air or ambient medium. The connecting means would typically be a modem at the receiving means.

Hence, in a closely related aspect, the present invention provides a broadcast and reception system comprising:

means for generating a plurality of entitlement messages relating to broadcast programs;

means for receiving said messages from said generating means via a modem; and

means for connecting said modem to said generating means and said receiving means.

The above features can afford the advantage of providing the user the necessary viewing authorization (via the EMM) more quickly than has hitherto been possible, partly because, since the SAS typically uses a smaller amount of computer code than the SMS, the SAS can operate more efficiently (and in real time), partly because the SAS can itself, directly, generate the requisite EMM, and partly because the EMM can be passed to the user or subscriber via a dedicated (typically modemmed) link.

Preferably, the generating means is connected to said modem via a communications server which is preferably included in or associated with said generating means.

The receiving means may be further adapted to receive said entitlement messages via a satellite transponder.

The receiving means may be a receiver/decoder comprising means for receiving a compressed MPEG-type signal, means for decoding the received signal to provide a television signal and means for supplying the television signal to a television.

Preferably, the receiving means is adapted to communicate with said generating means

via said modem and connecting means. The receiving means may comprise means for reading a smartcard insertable thereinto by an end user, the smartcard having stored therein data to initiate automatically the transmission of a message from said receiving means to said generating means upon insertion of the smartcard by the end user.

In addition, the system may further comprise a voice link to enable the end user of the broadcast and reception system to communicate with the generating means.

It will be understood from the above that the present invention provides two arrangements by which the time it takes for an end user to access a desired product is reduced. Preferably both arrangements are employed to achieve the maximum time saving but either arrangement can be used individually.

According to a further aspect of the present invention, there is provided a broadcast and reception system, comprising, at the broadcast end:

a broadcast system including means for broadcasting a callback request;

and at the reception end:

a receiver including means for calling back the broadcast system in response to the callback request.

By providing that the broadcast system can request the receiver to call it back, the possibility is afforded of the broadcast system obtaining information from the receiver about the state of the receiver.

Preferably, the means for calling back the broadcast system includes a modem connectable to a telephone system. By using a modemmed back channel, a simple way of putting the invention into effect can be provided.

Preferably also, the means for calling back the broadcast system is arranged to transfer to the broadcast system information concerning the receiver. This information might include the number of remaining tokens, the number of pre-booked sessions, and so

on.

Preferably, the broadcast system includes means for storing the information, so that it can be processed at a later time, as desired.

Preferably, the broadcast means is arranged to broadcast a callback request which includes a command that the callback be made at a given time, and the means for calling back the broadcast system is arranged to respond to said command. By arranging for the callback to be later than the actual request, greater flexibility can be imparted to the system.

The broadcasting means may be arranged to broadcast as the callback request one or more Entitlement Messages for broadcast.

Preferably, the broadcast system includes means for generating a check message (such as a random number) and passing this to the receiver, the receiver includes means for encrypting the check message and passing this to the broadcast system, and the broadcast system further includes means for decrypting the check message received from the receiver and comparing this with the original check message. In this way it can be checked whether the receiver is genuine.

Any of the above features may be combined together in any appropriate combination. They may also be provided, as appropriate, in method aspects.

Preferred features of the present invention will now be described, purely by way of example, with reference to the accompanying drawings, in which:-

Figure 1 shows the overall architecture of a digital television system according to the preferred embodiment of the present invention;

Figure 2 shows the architecture of a conditional access system of the digital television system;

Figure 3 shows the structure of an Entitlement Management Message used in the conditional access system;

Figure 4 is a schematic diagram of the hardware of a Subscriber Authorisation System (SAS) according to a preferred embodiment of the present invention;

Figure 5 is a schematic diagram of the architecture of the SAS;

Figure 6 is a schematic diagram of a Subscriber Technical Management server forming part of the SAS;

Figure 7 is a flow diagram of the procedure for automatic renewal of subscriptions as implemented by the SAS;

Figure 8 is a schematic diagram of a group subscription bitmap used in the automatic renewal procedure;

Figure 9 shows the structure of an EMM used in the automatic renewal procedure;

Figure 10 shows in detail the structure of the EMM;

Figure 11 is a schematic diagram of an order centralized server when used to receive commands directly through communications servers;

Figure 12 illustrates diagrammatically a part of Figure 2 showing one embodiment of the present invention;

Figure 13 is a schematic diagram of the order centralized server when used to receive commands from the subscriber authorization system to request a callback;

Figure 14 is a schematic diagram of the communications servers;

Figure 15 shows the manner in which EMM emission cycle rate is varied according to the timing of a PPV event;

Figure 16 is a schematic diagram of a Message Emitter used to emit EMMs;

Figure 17 is a schematic diagram showing the manner of storage of EMMs within the Message Emitter;

Figure 18 is a schematic diagram of a smartcard;

Figure 19 is a schematic diagram of an arrangement of zones in the memory of the smartcard; and

Figure 20 is a schematic diagram of a PPV event description.

An overview of a digital television broadcast and reception system 1000 according to the present invention is shown in Figure 1. The invention includes a mostly conventional digital television system 2000 which uses the known MPEG-2 compression system to transmit compressed digital signals. In more detail, MPEG-2 compressor 2002 in a broadcast centre receives a digital signal stream (typically a stream of video signals). The compressor 2002 is connected to a multiplexer and scrambler 2004 by linkage 2006. The multiplexer 2004 receives a plurality of further input signals, assembles one or more transport streams and transmits compressed digital signals to a transmitter 2008 of the broadcast centre via linkage 2010, which can of course take a wide variety of forms including telecom links. The transmitter 2008 transmits electromagnetic signals via uplink 2012 towards a satellite transponder 2014, where they are electronically processed and broadcast via notional downlink 2016 to earth receiver 2018, conventionally in the form of a dish owned or rented by the end user. The signals received by receiver 2018 are transmitted to an integrated receiver/decoder 2020 owned or rented by the end user and connected to the end user's television set 2022. The receiver/decoder 2020 decodes the compressed MPEG-2 signal into a television signal for the television set 2022.

A conditional access system 3000 is connected to the multiplexer 2004 and the receiver/decoder 2020, and is located partly in the broadcast centre and partly in the decoder. It enables the end user to access digital television broadcasts from one or more broadcast suppliers. A smartcard, capable of decrypting messages relating to commercial offers (that is, one or several television programmes sold by the broadcast supplier), can be inserted into the receiver/decoder 2020. Using the decoder 2020 and smartcard, the end user may purchase events in either a subscription mode or a pay-per-view mode.

An interactive system 4000, also connected to the multiplexer 2004 and the receiver/decoder 2020 and again located partly in the broadcast centre and partly in the decoder, enables the end user to interact with various applications via a modemmed back channel 4002.

The conditional access system 3000 is now described in more detail.

With reference to Figure 2, in overview the conditional access system 3000 includes a Subscriber Authorization System (SAS) 3002. The SAS 3002 is connected to one or more Subscriber Management Systems (SMS) 3004, one SMS for each broadcast supplier, by a respective TCP-IP linkage 3006 (although other types of linkage could alternatively be used). Alternatively, one SMS could be shared between two broadcast suppliers, or one supplier could use two SMSs, and so on.

First encrypting units in the form of ciphering units 3008 utilising "mother" smartcards 3010 are connected to the SAS by linkage 3012. Second encrypting units again in the form of ciphering units 3014 utilising mother smartcards 3016 are connected to the multiplexer 2004 by linkage 3018. The receiver/decoder 2020 receives a "daughter" smartcard 3020. It is connected directly to the SAS 3002 by Communications Servers 3022 via the modemmed back channel 4002. The SAS sends amongst other things subscription rights to the daughter smartcard on request.

The smartcards contain the secrets of one or more commercial operators. The

"mother" smartcard encrypts different kinds of messages and the "daughter" smartcards decrypt the messages, if they have the rights to do so.

The first and second ciphering units 3008 and 3014 comprise a rack, an electronic VME card with software stored on an EEPROM, up to 20 electronic cards and one smartcard 3010 and 3016 respectively, for each electronic card, one (card 3016) for encrypting the ECMs and one (card 3010) for encrypting the EMMs.

The operation of the conditional access system 3000 of the digital television system will now be described in more detail with reference to the various components of the television system 2000 and the conditional access system 3000.

Multiplexer and Scrambler

With reference to Figures 1 and 2, in the broadcast centre, the digital video signal is first compressed (or bit rate reduced), using the MPEG-2 compressor 2002. This compressed signal is then transmitted to the multiplexer and scrambler 2004 via the linkage 2006 in order to be multiplexed with other data, such as other compressed data.

The scrambler generates a control word used in the scrambling process and included in the MPEG-2 stream in the multiplexer 2004. The control word is generated internally and enables the end user's integrated receiver/decoder 2020 to descramble the programme.

Access criteria, indicating how the programme is commercialised, are also added to the MPEG-2 stream. The programme may be commercialised in either one of a number of "subscription" modes and/or one of a number of "Pay Per View" (PPV) modes or events. In the subscription mode, the end user subscribes to one or more commercial offers, or "bouquets", thus getting the rights to watch every channel inside those bouquets. In the preferred embodiment, up to 960 commercial offers may be selected from a bouquet of channels. In the Pay Per View mode, the end user is provided with the capability to purchase events as he wishes. This can be achieved

by either pre-booking the event in advance ("pre-book mode"), or by purchasing the event as soon as it is broadcast ("impulse mode"). In the preferred embodiment, all users are subscribers, whether or not they watch in subscription or PPV mode, but of course PPV viewers need not necessarily be subscribers.

Both the control word and the access criteria are used to build an Entitlement Control Message (ECM); this is a message sent in relation with one scrambled program; the message contains a control word (which allows for the descrambling of the program) and the access criteria of the broadcast program. The access criteria and control word are transmitted to the second encrypting unit 3014 via the linkage 3018. In this unit, an ECM is generated, encrypted and transmitted on to the multiplexer and scrambler 2004.

Each service broadcast by a broadcast supplier in a data stream comprises a number of distinct components; for example a television programme includes a video component, an audio component, a sub-title component and so on. Each of these components of a service is individually scrambled and encrypted for subsequent broadcast to the transponder 2014. In respect of each scrambled component of the service, a separate ECM is required.

Programme Transmission

The multiplexer 2004 receives electrical signals comprising encrypted EMMs from the SAS 3002, encrypted ECMs from the second encrypting unit 3014 and compressed programmes from the compressor 2002. The multiplexer 2004 scrambles the programmes and transmits the scrambled programmes, the encrypted EMMs and the encrypted ECMs as electric signals to a transmitter 2008 of the broadcast centre via linkage 2010. The transmitter 2008 transmits electromagnetic signals towards the satellite transponder 2014 via uplink 2012.

Programme Reception

The satellite transponder 2014 receives and processes the electromagnetic signals transmitted by the transmitter 2008 and transmits the signals on to the earth receiver

2018, conventionally in the form of a dish owned or rented by the end user, via downlink 2016. The signals received by receiver 2018 are transmitted to the integrated receiver/decoder 2020 owned or rented by the end user and connected to the end user's television set 2022. The receiver/decoder 2020 demultiplexes the signals to obtain scrambled programmes with encrypted EMMs and encrypted ECMs.

If the programme is not scrambled, that is, no ECM has been transmitted with the MPEG-2 stream, the receiver/decoder 2020 decompresses the data and transforms the signal into a video signal for transmission to television set 2022.

If the programme is scrambled, the receiver/decoder 2020 extracts the corresponding ECM from the MPEG-2 stream and passes the ECM to the "daughter" smartcard 3020 of the end user. This slots into a housing in the receiver/decoder 2020. The daughter smartcard 3020 controls whether the end user has the right to decrypt the ECM and to access the programme. If not, a negative status is passed to the receiver/decoder 2020 to indicate that the programme cannot be descrambled. If the end user does have the rights, the ECM is decrypted and the control word extracted. The decoder 2020 can then descramble the programme using this control word. The MPEG-2 stream is decompressed and translated into a video signal for onward transmission to television set 2022.

Subscriber Management System (SMS)

A Subscriber Management System (SMS) 3004 includes a database 3024 which manages, amongst others, all of the end user files, commercial offers (such as tariffs and promotions), subscriptions, PPV details, and data regarding end user consumption and authorization. The SMS may be physically remote from the SAS.

Each SMS 3004 transmits messages to the SAS 3002 via respective linkage 3006 which imply modifications to or creations of Entitlement Management Messages (EMMs) to be transmitted to end users.

The SMS 3004 also transmits messages to the SAS 3002 which imply no

modifications or creations of EMMs but imply only a change in an end user's state (relating to the authorization granted to the end user when ordering products or to the amount that the end user will be charged).

As described later, the SAS 3002 sends messages (typically requesting information such as call-back information or billing information) to the SMS 3004, so that it will be apparent that communication between the two is two-way.

Entitlement Management Messages (EMMs)

The EMM is a message dedicated to an individual end user (subscriber), or a group of end users, only (in contrast with an ECM, which is dedicated to one scrambled programme only or a set of scrambled programmes if part of the same commercial offer). Each group may contain a given number of end users. This organisation as a group aims at optimising the bandwidth; that is, access to one group can permit the reaching of a great number of end users.

Various specific types of EMM are used in putting the present invention into practice. Individual EMMs are dedicated to individual subscribers, and are typically used in the provision of Pay Per View services; these contain the group identifier and the position of the subscriber in that group. So-called "Group" subscription EMMs are dedicated to groups of, say, 256 individual users, and are typically used in the administration of some subscription services. This EMM has a group identifier and a subscribers' group bitmap. Audience EMMs are dedicated to entire audiences, and might for example be used by a particular operator to provide certain free services. An "audience" is the totality of subscribers having smartcards which bear the same Operator Identifier (OPI). Finally, a "unique" EMM is addressed to the unique identifier of the smartcard.

The structure of a typical EMM is now described with reference to Figure 3. Basically, the EMM, which is implemented as a series of digital data bits, comprises a header 3060, the EMM proper 3062, and a signature 3064. The header 3060 in turn comprises a type identifier 3066 to identify whether the type is individual, group,

audience or some other type, a length identifier 3068 which gives the length of the EMM, an optional address 3070 for the EMM, an operator identifier 3072 and a key identifier 3074. The EMM proper 3062 of course varies greatly according to its type. Finally, the signature 3064, which is typically of 8 bytes long, provides a number of checks against corruption of the remaining data in the EMM.

Subscriber Authorization System (SAS)

The messages generated by the SMS 3004 are passed via linkage 3006 to the Subscriber Authorization System (SAS) 3002, which in turn generates messages acknowledging receipt of the messages generated by the SMS 3004 and passes these acknowledgements to the SMS 3004.

As shown in Figure 4, at the hardware level the SAS comprises in known fashion a mainframe computer 3050 (in the preferred embodiment a DEC machine) connected to one or more keyboards 3052 for data and command input, one or more Visual Display Units (VDUs) 3054 for display of output information and data storage means 3056. Some redundancy in hardware may be provided.

At the software level the SAS runs, in the preferred embodiment on a standard open VMS operating system, a suite of software whose architecture is now described in overview with reference to Figure 5; it will be understood that the software could alteratively be implemented in hardware.

In overview the SAS comprises a Subscription Chain area 3100 to give rights for subscription mode and to renew the rights automatically each month, a Pay Per View Chain area 3200 to give rights for PPV events, and an EMM Injector 3300 for passing EMMs created by the Subscription and PPV chain areas to the multiplexer and scrambler 2004, and hence to feed the MPEG stream with EMMs. If other rights are to be granted, such as Pay Per File (PPF) rights in the case of downloading computer software to a user's Personal Computer, other similar areas are also provided.

One function of the SAS 3002 is to manage the access rights to television

programmes, available as commercial offers in subscription mode or sold as PPV events according to different modes of commercialisation (pre-book mode, impulse mode). The SAS 3002, according to those rights and to information received from the SMS 3004, generates EMMs for the subscriber.

The Subscription Chain area 3100 comprises a Command Interface (CI) 3102, a Subscriber Technical Management (STM) server 3104, a Message Generator (MG) 3106, and the Ciphering Unit 3008.

The PPV Chain area 3200 comprises an Authorisation Server (AS) 3202, a relational database 3204 for storing relevant details of the end users, a local blacklist database 3205, Database Servers 3206 for the database, an Order Centralized Server (OCS) 3207, a Server for Programme Broadcaster (SPB) 3208, a Message Generator (MG) 3210 whose function is basically the same as that for the Subscription Chain area and is hence not described further in any detail, and the Ciphering Unit 3008.

The EMM Injector 3300 comprises a plurality of Message Emitters (MEs) 3302, 3304, 3306 and 3308 and Software Multiplexers (SMUXs) 3310 and 3312. In the preferred embodiment, there are two MEs, 3302 and 3304 for the Message Generator 3106, with the other two MEs 3306 and 3308 for the Message Generator 3210. MEs 3302 and 3306 are connected to the SMUX 3310 whilst MEs 3304 and 3308 are connected to the SMUX 3312.

Each of the three main components of the SAS (the Subscription Chain area, the PPV Chain area and the EMM Injector) are now considered in more detail.

Subscription Chain Area

Considering first the Subscription Chain area 3100, the Command Interface 3102 is primarily for despatching messages from the SMS 3004 to the STM server 3104, as well as to the OCS 3206, and from the OCS to the SMS. The Command Interface takes as input from the SMS either direct commands or batch files containing commands. It performs syntactic analysis on the messages coming from the STM

server, and is able to emit accurate messages when an error occurs in a message (parameter out of range, missing parameter, and so on). It traces incoming commands in textual form in a trace file 3110 and also in binary form in a replay file 3112 in order to be able to replay a series of commands. Traces can be disabled and the size of files limited.

Detailed discussion of the STM server 3104 is now provided with particular reference to Figure 6. The STM server is effectively the main engine of the Subscription Chain area, and has the purpose of managing free rights, the creation of new subscribers and the renewal of existing subscribers. As shown in the figure, commands are passed on to the Message Generator 3106, albeit in a different format from that in which the commands are passed to the STM server. For each command, the STM server is arranged to send an acknowledgement message to the CI only when the relevant command has been successfully processed and sent to the MG.

The STM server includes a subscriber database 3120, in which all the relevant parameters of the subscribers are stored (smartcard number, commercial offers, state, group and position in the group, and so on). The database performs semantic checks of the commands sent by the CI 3102 against the content of the database, and updates the database when the commands are valid.

The STM server further manages a First In First Out (FIFO) buffer 3122 between the STM server and the MG, as well as a backup disk FIFO 3124. The purpose of the FIFOs is to average the flow of commands from the CI if the MG is not able to respond for a while for any reason. They can also ensure that in the case of a crash of the STM server or MG no command will be lost, since the STM server is arranged to empty (that is, send to the MG) its FIFOs when restarted. The FIFOs are implemented as files.

The STM server includes at its core an automatic renewal server 3126 which automatically generates renewals, and, if required by the operators, free rights. In this context, the generation of renewals may be thought of as including the generation of

rights for the first time, although it will be understood that the generation of new rights is initiated at the SMS. As will become apparent, the two can be treated by roughly the same commands and EMMs.

Having the STM separate from the SAS, and the automatic renewal server within the SAS rather than (in known systems) in the SMS 3004, is a particularly important feature, since it can significantly reduce the number of commands which need to be passed from the SMS to the SAS (bearing in mind that the SMS and SAS may be in different locations and operated by different operators). In fact, the two main commands required from the SMS are merely commands that a new subscription should be started and that an existing subscription should be stopped (for example in the case of non-payment). By minimising command exchange between the SMS and SAS, the possibility of failure of command transfer in the linkage 3006 between the two is reduced; also, the design of the SMS does not need to take into account the features of the conditional access system 3000 generally.

Automatic renewal proceeds in the fashion indicated in the flow diagram of Figure 7. In order to reduce bandwidth, and given that a very high percentage of all renewals are standard, renewal proceeds in groups of subscribers; in the preferred embodiments there are 256 individual subscribers per group. The flow diagram begins with the start step 3130, and proceeds to step 3132 where a monthly activation of the renewal function is made (although of course it will be appreciated that other frequencies are also possible). With a monthly frequency, rights are given to the end user for the current month and all of the following month, at which point they expire if not renewed.

In step 3134 the subscriber database 3120 is accessed in respect of each group and each individual within that group to determine whether rights for the particular individual are to be renewed.

In step 3136, a group subscription bitmap is set up according to the contents of the subscriber database, as shown in Figure 8. The bitmap comprises a group identifier

(in this case Group 1 – "G1") 3138 and 256 individual subscriber zones 3140. The individual bits in the bitmap are set to 1 or zero according to whether or not the particular subscriber is to have his rights renewed. A typical set of binary data is shown in the figure.

In step 3142 the appropriate commands, including the group subscription bitmap, are passed to the Message Generator 3106. In step 3143 the Message Generator sets an obsolescence date to indicate to the smartcard the date beyond which the particular subscription EMM is not valid; typically this date is set as the end of the next month.

In step 3144 the Message Generator generates from the commands appropriate group subscription EMMs and asks the Ciphering Unit 3008 to cipher the EMMs, the ciphered EMMs being then passed to the EMM Injector 3300, which, in step 3146, injects the EMMs into the MPEG–2 data stream.

Step 3148 indicates that the above described procedure is repeated for each and every group. The process is finally brought to an end at stop step 3150.

The flow diagram described above with reference to Figure 7 relates in fact specifically to the renewal of subscriptions. The STM also manages in a similar way free audience rights and new subscribers.

In the case of free audience rights, available for specific television programmes or groups of such programmes, these are made available by the STM issuing a command to the Message Generator to generate appropriate audience EMMs (for a whole audience) with an obsolescence date a given number of days (or weeks) hence. The MG computes the precise obsolescence date based on the STM command.

In the case of new subscribers, these are dealt with in two stages. Firstly, on purchase the smartcard in the receiver/decoder 2020 (if desired by the operator) affords the subscriber free rights for a given period (typically a few days). This is achieved by generating a bitmap for the subscriber which includes the relevant obsolescence date.

The subscriber then passes his completed paperwork to the operator managing the subscriber (at the SMS). Once the paperwork has been processed, the SMS supplies to the SAS a start command for that particular subscriber. On receipt by the SAS of the start command, the STM commands the MG to assign a unique address to the new subscriber (with a particular group number and position within the group) and to generate a special, so-called "commercial offer" subscription EMM (as opposed to the more usual "group" subscription EMM used for renewals) to provide the particular subscriber with rights until the end of the next month. From this point renewal of the subscriber can occur automatically as described above. By this two stage process it is possible to grant new subscribers rights until the SMS issues a stop command.

It is to be noted that the commercial offer subscription EMM is used for new subscribers and for reactivation of existing subscribers. The group subscription EMM is used for renewal and suspension purposes.

With reference to Figure 9, a typical subscription EMM proper (that is, ignoring the header and signature) generated by the above procedure comprises the following main portions, namely typically a 256 bit subscription (or subscribers' group) bitmap 3152, 128 bits of management ciphering keys 3154 for the ciphering of the EMM, 64 bits of each exploitation ciphering key 3156 to enable the smartcard 3020 to decipher a control word to provide access to broadcast programmes, and 16 bits of obsolescence date 3158 to indicate the date beyond which the smartcard will ignore the EMM. In fact in the preferred embodiment three exploitation keys are provided, one set for the present month, one set for the next month, and one for resume purposes in the event of system failure.

In more detail, the group subscription EMM proper has all of the above components, except the management ciphering keys 3154. The commercial offer subscription EMM proper (which is for an individual subscriber) includes instead of the full subscribers' group bitmap 3152 the group ID followed by the position in the group, and then management ciphering keys 3154 and three exploitation keys 3156, followed by the relevant obsolescence date 3158.

The Message Generator 3106 serves to transform commands issued by the STM server 3104 into EMMs for passing to the Message Emitter 3302. With reference to Figure 5, firstly, the MG produces the EMMs proper and passes them to the Ciphering Unit 3008 for ciphering with respect to the management and exploitation keys. The CU completes the signature 3064 on the EMM (see Figure 3) and passes the EMM back to the MG, where the header 3060 is added. The EMMs which are passed to the Message Emitter are thus complete EMMs. The Message Generator also determines the broadcast start and stop time and the rate of emission of the EMMs, and passes these as appropriate directions along with the EMMs to the Message Emitter. The MG only generates a given EMM once; it is the ME which performs its cyclic transmission.

Again with reference to Figure 5, the Message Generator includes its own EMM database 3160 which, for the lifetime of the relevant EMM, stores it. It is erased once its emission duration has expired. The database is used to ensure consistency between the MG and ME, so that for example when an end user is suspended the ME will not continue to send renewals. In this regard the MG computes the relevant operations and sends them to the ME.

On generation of an EMM, the MG assigns a unique identifier to the EMM. When the MG passes the EMM to the ME, it also passes the EMM ID. This enables identification of a particular EMM at both the MG and the ME.

Also concerning the Subscription Chain area, the Message Generator includes two FIFOs 3162 and 3164, one for each of the relevant Message Emitters 3302 and 3304 in the EMM Injector 3300, for storing the ciphered EMMs. Since the Subscription Chain area and EMM Injector may be a significant distance apart, the use of FIFOs can allow full continuity in EMM transmission even if the links 3166 and 3168 between the two fail. Similar FIFO's are provided in the Pay Per View Chain area.

One particular feature of the Message Generator in particular and the conditional access system in general concerns the way that it reduces the length of the EMM

proper 3062 by mixing parameter length and identifier to save space. This is now described with reference to Figure 10 which illustrates an exemplary EMM (in fact a PPV EMM, which is the simplest EMM). The reduction in length occurs in the Pid (Packet or "Parameter" identifier) 3170. This comprises two portions, the actual ID 3172, and the length parameter for the packet 3174 (necessary in order that the start of the next packet can be identified). The whole Pid is expressed in just one byte of information, 4 bits being reserved for the ID, and four for the length. Because 4 bits is not sufficient to define the length in true binary fashion, a different correspondence between the bits and the actual length is used, this correspondence being represented in a look-up table, stored in storage area 3178 in the Message Generator (see Figure 5). The correspondence is typically as follows:-

0000	=	0
0001	=	1
0010	=	2
0011	=	3
0100	=	4
0101	=	5
0110	=	6
0111	=	7
1000	=	8
1001	=	9
1010	=	10
1011	=	11
1100	=	12
1101	=	16
1110	=	24
1111	=	32

It will be seen that the length parameter is not directly proportional to the actual length of the packet; the relationship is in part more quadratic rather than linear. This allows for a greater range of packet length.

Pay Per View Chain Area

Concerning the Pay Per View Chain area 3200, with reference to Figure 5 in more detail the Authorisation Server 3202 has as its client the Order Centralized Server 3207, which requests information about each subscriber which connects to the Communications Servers 3022 to purchase a PPV product.

If the subscriber is known from the AS 3202, a set of transactions takes place. If the subscriber is authorized for the order, the AS creates a bill and sends it to the OCS. Otherwise, it signals to the OCS that the order is not authorized.

It is only at the end of this set of transactions that the AS updates the end users database 3204 via the database servers (DBAS) 3206, if at least one transaction was authorized; this optimizes the number of database accesses.

The criteria according to which the AS authorizes purchase are stored in the database, accessed through DBAS processes. In one embodiment, the database is the same as the database accessed by the STM.

Depending on consumer profile, the authorization may be denied (PPV_Forbidden,Casino_Forbidden ...). These kind of criteria are updated by STM 3104, on behalf of the SMS 3004.

Other parameters are checked, such as limits allowed for purchase (either by credit card, automatic payment, or number of authorized token purchases per day).

In case of payment with a credit card, the number of the card is checked against a local blacklist stored in the local blacklist database 3205.

When all the verifications are successful, the AS:-

1. Generates a bill and sends it to the OCS, which completes this bill and stores it in a file, this file being later sent to the SMS for processing (customer actual billing); and

2. Updates the database, mainly to set new purchase limits.

This check-and-generate-bill-if-OK mechanism applies for each command a subscriber may request during a single connection (it is possible to order e.g. 5 movies in a single session).

It is to be noted that the AS has a reduced amount of information concerning the subscriber, by comparison with that held by the SMS. For example, the AS does not hold the name or address of the subscriber. On the other hand, the AS does hold the smartcard number of the subscriber, the subscriber's consumer category (so that different offers can be made to different subscribers), and various flags which state whether, for example, the subscriber may purchase on credit, or he is suspended or his smartcard has been stolen. Use of a reduced amount of information can help to reduce the amount of time taken to authorize a particular subscriber request.

The main purpose of the DBASs 3206 is to increase database performance seen from the AS, by paralleling the accesses (so actually it does not make much sense to define a configuration with only one DBAS). An AS parameter determines how many DBASes should connect. A given DBAS may be connected to only one AS.

The OCS 2307 mainly deals with PPV commands. It operates in several modes.

Firstly, it operates to process commands issued by the SMS, such as product refreshment (for instance, if the bill is already stored by the SMS, no bill is generated by the OCS), update of the wallet in the smartcard 3020, and session cancellation/update.

The various steps in the procedure are:-

1. Identifying the relevant subscriber (using the AS 3202);
2. If valid, generate adequate commands to the Message Generator, in order to send an appropriate EMM. Commands may be:

Product commands,

Update of the wallet,

Session erasure.

Note that these operations do not imply creation of billing information, since billing is already known from the SMS. These operations are assimilated to "free products" purchase.

Secondly, the OCS deals with commands received from the subscribers through the Communications Servers 3022. These may be received either via a modem connected to the receiver/decoder 2020, or by voice activation via the telephone 4001, or by key activation via a MINITEL, PRESTEL or like system where available.

Thirdly, the OCS deals with callback requests issued by the SMS. These last two modes of operation are now discussed in more detail.

In the second type of mode described above it was stated that the OCS deals with commands received directly from the end user (subscriber) through the Communications Servers 3022. These include product orders (such as for a particular PPV event), a subscription modification requested by the subscriber, and a reset of a parental code (a parental code being a code by which parents may restrict the right of access to certain programmes or classes of programmes).

The way in which these commands are dealt with is now described in more detail with reference to Figure 11.

Product orders by a subscriber involve the following steps:

1. Identifying through the AS the caller who is making a call through the CS 3022 ordering a particular product;
2. Checking the caller's request validity, again using the AS (where the order is placed using the receiver/decoder 2020, this is achieved by verifying the smartcard 3020 details);
3. Ascertain the price of the purchase;
4. Check that the price does not exceed the caller's credit limit etc;
5. Receiving a partial bill from the AS;

6. Filling additional fields in the bill to form a completed bill;
7. Adding the completed bill to a billing information storage file 3212 for later processing; and
8. Sending corresponding command(s) to the PPV Message Generator 3210 to generate the relevant EMM(s).

The EMM(s) is sent either on the modem line 4002 if the consumer placed the product order using the receiver/decoder 2020 (more details of this are described later), or else it is broadcast. The one exception to this is where there is some failure of the modem connection (in the case where the consumer places the order using the receiver/decoder); in this event the EMM is broadcast over the air.

A subscription modification requested by a subscriber involves:

1. Identifying the caller (using the AS);
2. Sending information to the Command Interface; the CI in turn forwards this information to the SMS; and
3. Via the CI, the OCS then receives an answer from the SMS (in terms of the cost of the modification, if the modification is possible).

If modification was requested using the receiver/decoder, the OCS generates a confirmation to the SMS. Otherwise, for example in the case of phone or Minitel, the subscriber is prompted for confirmation and this answer sent to the SMS via the OCS and the CI.

Reset of a parental code involves:

1. Identifying the caller (using AS); and
2. Sending a command to the MG to generate an appropriate EMM bearing an appropriate reset password.

In the case of reset of parental code, the command to reset the code is for security reasons not permitted to originate from the receiver/decoder. Only the SMS, telephone and MINITEL or like can originate such a command. Hence in this

particular case the EMM(s) are broadcast only on air, never on the telephone line.

It will be understood from the above examples of different modes of operation of the OCS that the user can have direct access to the SAS, and in particular the OCS and AS, in that the Communications Servers are directly connected to the SAS, and in particular the OCS. This important feature is concerned with reducing the time for the user to communicate his command to the SAS.

This feature is illustrated further with reference to Figure 12, from which it can be seen that the end user's Set-Top-Box, and in particular its receiver/decoder 2020, has the capability of communicating directly with the Communications Servers 3022 associated with the SAS 3002. Instead of the connection from the end user to the Communications Servers 3022 of the SAS 3002 being through the SMS 3004 the connection is directly to the SAS 3002.

In fact, as directly mentioned two direct connections are provided.

The first direct connection is by a voice link via a telephone 4001 and appropriate telephone line (and/or by MINITEL or like connection where available) where the end users still have to input a series of voice commands or code numbers but time is saved compared with the communication being via the SMS 3004.

The second direct connection is from the receiver/decoder 2020 and the input of data is achieved automatically by the end user inserting his own daughter smartcard 3020 thus relieving the end user of the job of having to input the relevant data which in turn reduces the time taken and the likelihood of errors in making that input.

A further important feature which arises out of the above discussion is concerned with reducing the time taken for the resulting EMM to be transmitted to the end user in order to initiate viewing by the end user of the selected product.

In broad terms, and with reference to Figure 12, the feature is again achieved by

providing the end user's receiver/decoder 2020 with the capability of communicating directly with the Communications Servers 3022 associated with the SAS 3002.

As described earlier the integrated receiver/decoder 2020 is connected directly to the Communications Servers 3022 by the modemmed back channel 4002 so that commands from the decoder 2020 are processed by the SAS 3002, messages generated (including EMMs) and then sent back directly to the decoder 2020 through the back channel 4002. A protocol is used in the communication between the CS 3022 and the receiver/decoder 2020 (as described later), so that the CS receive acknowledgement of receipt of the relevant EMM, thereby adding certainty to the procedure.

Thus, for example, in the case of a pre-book mode the SAS 3002 receives messages from the end user via the smartcard and decoder 2020 via its modem and via the telephone line 4002, requesting access to a specific event/product, and returns a suitable EMM via the telephone line 4002 and modem to the decoder 2020, the modem and decoder being preferably located together in a Set-Top-Box (STB). This is thus achieved without having to transmit the EMM in the MPEG-2 data stream 2002 via the multiplexer and scrambler 2004, the uplink 2012, satellite 2014 and datalink 2016 to enable the end user to view the event/product. This can save considerably on time and bandwidth. Virtual certainty is provided that as soon as the subscriber has paid for his purchase the EMM will arrive at the receiver/decoder 2020.

In the third type of mode of operation of the OCS 3207 described above, the OCS deals with callback requests issued by the SAS. This is illustrated with reference to Figure 13. Typical callback requests have the purpose of ensuring that the receiver/decoder 2020 calls back the SAS via the modemmed back channel 4002 with the information that the SAS requires of the receiver /decoder.

As instructed by the Command Interface 3102, the subscription chain Message Generator 3106 generates and sends to the receiver/decoder 202 a callback EMM. This EMM is ciphered by the Ciphering Unit 3008 for security reasons. The EMM may contain the time/date at which the receiver/decoder should wake up and perform

a callback on its own, without being explicitly solicited; the EMM may also typically contain the phone numbers which the terminal must dial, the number of further attempts after unsuccessful calls and the delay between two calls.

When receiving the EMM, or at the specified time-date, the receiver/decoder connects to the Communications Servers 3022. The OCS 3207 first identifies the caller, using the AS 3202, and verifies certain details, such as smartcard operator and subscriber details. The OCS then asks the smartcard 3020 to send various ciphered information (such as the relevant session numbers, when the session was watched, how many times the subscriber is allowed to view the session again, the way in which the session was viewed, the number of remaining tokens, the number of prebooked sessions, etc). This information is deciphered by the PPV chain Message Generator 3210, again using the Ciphering Unit 3008. The OCS adds this information to a callback information storage file 3214 for later processing and passing to the SMS 3004. The information is ciphered for security reasons. The whole procedure is repeated until there is nothing more to be read from the smartcard.

One particular preferred feature of the callback facility is that before reading the smartcard (so just after the identification of the caller using the AS 3202 as described above) a check is made by the SAS 3002 that the receiver/decoder is indeed a genuine one rather than a pirated version or computer simulation. Such a check is carried out in the following manner. The SAS generates a random number, which is received by the receiver/decoder, ciphered, and then returned to the SAS. The SAS deciphers this number. If the deciphering is successful and the original random number is retrieved, it is concluded that the receiver/decoder is genuine, and the procedure continues. Otherwise, the procedure is discontinued.

Other functions which may occur during the callback are erasure of obsolete sessions on the smartcard, or filling of the wallet (this latter also being described later under the section entitled "Smartcard").

Also as regards the Pay Per View Chain area 3200, description is now made of the

Communications Servers 3022. At the hardware level, these comprise in the preferred embodiment a DEC Four parallel processor machine. At the software architecture level, with reference to Figure 14, in many respects the Communications Servers are conventional. One particular divergence from conventional designs arises from the fact that the Servers must serve both receiver/decoders 2020 and voice communication with conventional telephones 4001, as well possibly as MINITEL or like systems.

It will be noted in passing that two Order Centralized Servers 3207 are shown in Figure 14 (as "OCS1" and "OCS2"). Naturally any desired number may be provided.

The Communication Servers include two main servers ("CS1" and "CS2") as well as a number of frontal servers ("Frontal 1" and "Frontal 2"); whilst two frontal servers are shown in the figure, typically 10 or 12 may be provided per main server. Indeed, although two main servers CS1 and CS2 and two frontal servers, Frontal 1 and Frontal 2, have been shown, any number could be used. Some redundancy is usually desirable.

CS1 and CS2 are coupled to OCS1 and OCS2 via high level TCP/IP links 3230, whilst CS1 and CS2 are coupled to Frontal 1 and Frontal 2 via further TCP/IP links 3232.

As illustrated, CS1 and CS2 comprise servers for "SENDR" (transmission), "RECVR" (reception), "VTX" (MINITEL, PRESTEL or the like), "VOX" (voice communication), and "TRM" (communication with the receiver/decoder). These are coupled to the "BUS" for communication of signals to the Frontal servers.

CS1 and CS2 communicate directly with the receiver/decoders 2020 via their modemmed back channels 4002 using the X25 public network common protocol. The relatively low-level protocol between the Communications Servers 3022 and the receiver/decoders 3020 is in one preferred embodiment based upon the V42 standard international CCITT protocol, which provides reliability by having error detection and data re-transmission facilities, and uses a checksum routine to check the integrity of

the re-transmission. An escape mechanism is also provided in order to prevent the transmission of disallowed characters.

On the other hand, voice telephone communication is carried out via the Frontal Communications Servers, each capable of picking up, say, 30 simultaneous voice connections from the connection 3234 to the local telephone network via the high speed "T2" (E1) standard telephony ISDN lines.

Three particular functions of the software portion of the Communications Servers (which could of course alternatively be implemented fully in hardware) are firstly to convert the relatively low level protocol information received from the receiver/decoder into the relatively high level protocol information output to the OCS, secondly to attenuate or control the number of simultaneous connections being made, and thirdly to provide several simultaneous channels without any mixing. In this last regard, the Communications Servers play the role of a form of multiplexer, with the interactions in a particular channel being defined by a given Session ID (identifier), which is in fact used throughout the communication chain.

Finally as regards the Pay Per View Chain area 3200, and with reference again to Figure 5, the Server for Programme Broadcast (SPB) 3208 is coupled to one or more Programme Broadcasters 3250 (which would typically be located remotely from the SAS) to receive programme information. The SPB filters out for further use information corresponding to PPV events (sessions).

A particularly important feature is that the filtered programme event information is passed by the SPB to the MG which in turn sends a directive (control command) to the ME to change the rate of cyclic emission of the EMMs in given circumstances; this is done by the ME finding all EMMs with the relevant session identifier and changing the cycle rate allocated to such EMMs. This feature might be thought of as a dynamic allocation of bandwidth for specific EMMs. Cyclic EMM emission is discussed in more detail in the section below concerned with the EMM Injector.

The circumstances in which the cycle rate is changed are now described with reference to Figure 15, which demonstrates how cycle rate 3252 is raised a short while (say 10 minutes) before a particular PPV programme event until the end of the event from a slow cycle rate of say once every 30 minutes to a fast cycle rate of say once every 30 seconds to 1 minute in order to meet the anticipated extra user demand for PPV events at those times. In this way bandwidth can be allocated dynamically according to the anticipated user demand. This can assist in reducing the overall bandwidth requirement.

The cycle rate of other EMMs may also be varied. For example the cycle rate of subscription EMMs may be varied by the Multiplexer and Scrambler 2004 sending the appropriate bitrate directive.

EMM Injector

Concerning the EMM Injector 3300, details of the Message Emitters 3302 to 3308, forming part of the EMM Injector and acting as output means for the Message Generator, are now described with reference to Figure 16. Their function is take the EMMs and to pass them cyclically (in the manner of a carousel) via respective links 3314 and 3316 to the Software Multiplexers 3310 and 3312 and thence to the hardware multiplexers and scramblers 2004. In return the software multiplexers and scramblers 2004 generate a global bitrate directive to control the overall cycling rate of the EMMs; to do so, the MEs take into account various parameters such as the cycle time, the size of EMM, and so on. In the figure, EMM_X and EMM _Y are group EMMs for operators X and Y, whilst EMM_Z are other EMMs for either operator X or operator Y.

Further description proceeds for an exemplary one of the Message Emitters; it will be appreciated that the remaining MEs operate in similar fashion. The ME operates under control of directives from the MG, most notably transmission start and stop time and emission rate, as well as session number if the EMM is a PPV EMM. In relation to the emission rate, in the preferred embodiment the relevant directive may take one of five values from Very fast to Very slow. The numeric values are not specified in

the directive, but rather the ME maps the directive to an actual numeric value which is supplied by the relevant part of the SAS. In the preferred embodiment, the 5 emission rates are as follows:-

1.	Very fast	- every 30 seconds
2.	Fast	- every minute
3.	Medium	- every 15 minutes
4.	Slow	- every 30 minutes
5.	Very slow	- every 30 minutes

The ME has first and second databases 3320 and 3322. The first database is for those EMMs which have not yet achieved their broadcast date; these are stored in a series of chronological files in the database. The second database is for EMMs for immediate broadcast. In the event of a system crash, the ME is arranged to have the ability to re-read the relevant stored file and perform correct broadcast. All the files stored in the databases are updated upon request from the MG, when the MG wishes to maintain consistency between incoming directives and EMMs already sent to the ME. The EMMs actually being broadcast are also stored in Random Access Memory 3324.

A combination of the FIFOs 3162 and 3164 in the Message Generator and the databases 3320 and 3322 in the Message Emitter means that the two can operate in standalone mode if the link 3166 between them is temporarily broken; the ME can still broadcast EMMs.

The Software Multiplexers (SMUX) 3310 and 3312 provide an interface between the MEs and the hardware multiplexers 2004. In the preferred embodiment, they each receive EMMs from two of the MEs, although in general there is no restriction on the number of MEs that can be connected with one SMUX. The SMUXs concentrate the EMMs and then pass them according to the type of EMM to the appropriate hardware multiplexer. This is necessary because the hardware multiplexers take the different types of EMMs and place them at different places in the MPEG-2 stream. The

SMUX's also forward global bitrate directives from the hardware multiplexers to the MEs.

One particularly important feature of the ME is that it emits EMMs in random order. The reason for this is as follows. The Message Emitter has no ability to sense or control what it emits to the multiplexer. Hence it is possible that it may transmit two EMMs which are to be received and decoded by the receiver/decoder 2020 back to back. In such circumstances, further, it is possible that if the EMMs are insufficiently separated the receiver/decoder and smartcard will be unable to sense and decode properly the second of the EMMs. Cyclically emitting the EMMs in random order can solve this problem.

The manner in which randomization is achieved is now described with reference to Figure 17; in the preferred embodiment the necessary software logic is implemented in the ADA computer language. A particularly important part of the randomization is the correct storage of the EMMs in the databases 3320 and 3322 (which are used for backup purposes) and in the RAM 3324. For a particular cycle rate and operator, the EMMs are stored in a two-dimensional array, by rank 3330 (going say from A to Z) and number in the rank 3332 (going from 0 to N). A third dimension is added by cycle rate 3334, so that there are as many two-dimensional arrays as there are cycle rates. In the preferred embodiment there are 256 ranks and typically 200 or 300 EMMs in each rank; there are 5 cycle rates. A final dimension to the array is added by the presence of different operators; there are as many three-dimensional arrays as there are operators. Storage of the data in this fashion can permit rapid retrieval in the event that the MG wants to delete a particular EMM.

Storage of the EMMs takes place according to the "hash" algorithm (otherwise known as the "one-way hash function". This operates on a modulo approach, so that successive ranks are filled before a higher number in the rank is used, and the number of EMMs in each rank remains roughly constant. The example is considered of there being 256 ranks. When the MG sends the ME an EMM with identifer (ID) 1, the rank "1" is assigned to this EMM, and it takes the first number 3332 in the rank 3330.

The EMM with ID 2 is assigned the rank "2", and so on, up to the rank 256. The EMM with ID 257 is assigned the rank "1" again (based on the modulo function), and takes the second number in the first rank, and so on.

Retrieval of a specific EMM, for example when deletion of a specific EMM is requested by the MG, is effected by means of the inverse of the above. The hash algorithm is applied to the EMM ID to obtain the rank, after which the number in the rank is found.

The actual randomization occurs when the EMMs are, on a cyclical basis, retrieved from RAM 3324 using the randomization means 3340 which is implemented in the hardware and/or software of the Message Emitter. The retrieval is random, and again based on the hash algorithm. Firstly, a random number (in the above example initially in the range 1 to 256) is chosen, to yield the particular rank of interest. Secondly, a further random number is chosen to yield the particular number in the rank. The further random number is selected according to the total number of EMMs in a given rank. Once a given EMM has been selected and broadcast, it is moved to a second identical storage area in the RAM 3324, again using the hash function. Hence the first area diminishes in size as the EMMs are broadcast, to the extent that, once a complete rank has been used, this is deleted. Once the first storage area is completely empty, it is replaced by the second storage area before a new round of EMM broadcast, and vice versa.

In the above fashion, after two or three cycles of the EMMs, statistically the chances of any two EMMs destined for the same end user being transmitted back to back is negligible.

At regular intervals whilst the EMMs are being stored the computer 3050 computes the number of bytes in storage and from this computes the bitrate of emission given the global bitrate directive from the multiplexer and software multiplexer.

Reference was made above to the backup databases 3320 and 3322. These are in fact

in the preferred embodiment sequential file stores, which hold a backup version of what is in the RAM 3324. In the event of failure of the Message Emitter and subsequent restart, or more generally when the ME is being restarted for whatever reason, a link is made between the RAM and the databases, over which the stored EMMs are uploaded to RAM. In this way, the risk of losing EMMs in the event of failure can be removed.

Similar storage of PPV EMMs occurs to that described above in relation to subscription EMMs, with the rank typically corresponding to a given operator and the number in the rank corresponding to the session number.

Smartcard

A daughter, or "subscriber", smartcard 3020 is schematically shown in Figure 18 and comprises an 8 bit microprocessor 110, such as a Motorola 6805 microprocessor, having an input/output bus coupled to a standard array of contacts 120 which in use are connected to a corresponding array of contacts in the card reader of the receiver/decoder 2020, the card reader being of conventional design. The microprocessor 110 is also provided with bus connections to preferably masked ROM 130, RAM 140 and EEPROM 150. The smartcard complies with the ISO 7816-1, 7816-2 and 7816-3 standard protocols which determine certain physical parameters of the smartcard, the positions of the contacts on the chip and certain communications between the external system (and particularly the receiver/decoder 2020) and the smartcard respectively and which will therefore not be further described here. One function of the microprocessor 110 is to manage the memory in the smartcard, as now described.

The EEPROM 150 contains certain dynamically-created operator zones 154, 155, 156 and dynamically-created data zones which will now be described with reference to Figure 19.

Referring to Figure 19, EEPROM 150 comprises a permanent "card ID" (or manufacturer) zone 151 of 8 bytes which contains a permanent subscriber smartcard

identifier set by the manufacturer of the smartcard 3020.

When the smartcard is reset, the microprocessor 110 issues a signal to receiver/decoder 2020, the signal comprising an identifier of the conditional access system used by the smartcard and data generated from data stored in the smartcard, including the card ID. This signal is stored by the receiver/decoder 2020, which subsequently utilises the stored signal to check whether the smartcard is compatible with the conditional access system used by the receiver/decoder 2020.

The EEPROM 150 also contains a permanent "random number generator" zone 152 which contains a program for generating pseudo-random numbers. Such random numbers are used for diversifying transaction output signals generated by the smartcard 3020 and sent back to the broadcaster.

Below the random number generator zone 152 a permanent "management" zone 153 of 144 bytes is provided. The permanent management zone 153 is a specific operator zone utilised by a program in the ROM 130 in the dynamic creation (and removal) of zones 154, 155, 156... as described below. The permanent management zone 153 contains data relating to the rights of the smartcard to create or remove zones.

The program for dynamically creating and removing zones is responsive to specific zone creation (or removal) EMMs which are transmitted by the SAS 3002 and received by the receiver/decoder 2020 and passed to the subscriber smartcard 3020. In order to create the EMMs the operator requires specific keys dedicated to the management zone. This prevents one operator from deleting zones relating to another operator.

Below the management zone 153 is a series of "operator ID" zones 154, 155, 156 for operators 1, 2 ..... N respectively. Normally at least one operator ID zone will be preloaded into the EEPROM of the subscriber smartcard 3020 so that the end user can decrypt programmes broadcast by that operator. However further operator ID zones can subsequently be dynamically created using the management zone 153 in response

to a transaction output signal generated via his smartcard 3020 by the end user (subscriber), as will subsequently be described.

Each operator zone 154, 155, 156 contains the identifier of the group to which the smartcard 3020 belongs, and the position of the smartcard within the group. This data enables the smartcard (along with the other smartcards in its group) to be responsive to a broadcast "group" subscription EMM having that group's address (but not the smartcard's position in the group) as well as to an "individual" (or commercial offers subscription) EMM addressed only to that smartcard within the group. There can be 256 member smartcards of each such group and this feature therefore reduces significantly the bandwidth required for broadcasting EMMs.

In order to reduce further the bandwidth required for broadcasting "group" subscription EMMs, the group data in each operator zone 154, 155, 156 and all similar zones in the EEPROM of smartcard 3020 and the other daughter smartcards is continually updated to enable a particular smartcard to change its position in each group to fill any holes created by e.g. deletion of a member of the group. The holes are filled by the SAS 3002 as in the STM server 3104 there is a list of such holes.

In this manner fragmentation is reduced and each group's membership is maintained at or near the maximum of 256 members.

Each operator zone 154, 155, 156 is associated with one or more "operator data objects" stored in the EEPROM 150. As shown in Figure 19, a series of dynamically created "operator data" objects 157–165 are located below the operator ID zones. Each of these objects is labelled with:

a) an "identifier" 1, 2, 3 .... N corresponding to its associated operator 1, 2, 3 ... N as shown in its left hand section in Figure 19;

b) an "ID" indicating the type of object; and

c) a "data" zone reserved for data, as shown in the right hand section of each relevant operator object in Figure 19. It should be understood that each operator is associated with a similar set of data objects so that the following description of the

types of data in the data objects of operator 1 is also applicable to the data objects of all the other operators. Also it will be noted that the data objects are located in contiguous physical regions of the EEPROM and that their order is immaterial.

Deletion of a data object creates a "hole" 166 in the smartcard, that is, the number of bytes that the deleted objects had previously occupied are not immediately occupied. The thus "freed" number of bytes, or "hole" are labelled with:

a) an "identifier" 0; and

b) an "ID" indicating that the bytes are free to receive an object.

The next data object created fills the hole, as identified by the identifier 0. In this manner the limited memory capacity (4 kilobytes) of the EEPROM 150 is efficiently utilised.

Turning now to the set of data objects associated with each operator, examples of the data objects are now described.

Data object 157 contains an EMM key used for decrypting encrypted EMM's received by the receiver/decoder 2020. This EMM key is permanently stored in the data object 157. This data object 157 may be created prior to distribution of the smartcard 3020, and/or may be created dynamically when creating a new operator zone (as described above).

Data object 159 contains ECM keys which are sent by the associated operator (in this case operator 1) to enable the end user to decrypt the particular "bouquet" of programs to which he has subscribed. New ECM keys are sent typically every month, along with a group subscription (renewal) EMM which renews the end user's overall right to view the broadcast from (in this case) operator 1. The use of separate EMM and ECM keys enables viewing rights to be purchased in different ways (in this embodiment by subscription and individually (Pay Per View)) and also increases security. The Pay Per View (PPV) mode will be described subsequently.

Since new ECM keys are sent periodically, it is essential to prevent a user from usin old ECM keys, for example by switching off the receiver/decoder or re-setting a clock to prevent expiry of an old ECM key so that a timer in the receiver/decoder 2020 could be overridden. Accordingly operatot zone 154 comprises an area (typically having a size of 2 bytes) containing an obsolescence date of the ECM keys. The smartcard 3020 is arranged to compare this date with the current date which is contained in received ECMs and to prevent decryption if the current date is later than the obsolescence date. The obsolescence date is transmitted via EMMs, as described above.

Data object 161 contains a 64 bit subscription bitmap which is an exact representation of the broadcast operator's programs to which the subscriber has subscribed. Every bit represents a program and is set to "1" if it is subscribed to and "0" if it is not.

Data object 163 contains a quantity of tokens which can be used by the consumer in PPV mode to buy viewing rights to an imminent broadcast e.g. in response to a free preview or other advertisement. Data object 163 also contains a limit value, which may be set to e.g. a negative value to allow credit to the consumer. Tokens can be purchased e.g. by credit and via the modemmed back channel 4002, or by using a voice server in combination with a credit card, for example. A particular event can be charged as one token or a number of tokens.

Data object 165 contains a description of a PPV event, as shown with reference to table 167 of Figure 20.

The PPV event description 167 contains a "session ID" 168 identifying the viewing session (corresponding to the program and the time and date of broadcasting) a "session mode" 169 indicating how the viewing right is being purchased (e.g. in pre-book mode), a "session index" 170 and a "session view" 171.

In respect of receiving a programme in PPV mode, the receiver decoder 2020 determines whether the programme is one sold in PPV mode. If so, the decoder 2020

checks, using the items stored in the PPV event description 167 whether the session ID for the programme is stored therein. If the session ID is stored therein, the control word is extracted from the ECM.

If the session ID is not stored therein, by means of a specific application the receiver/decoder 2020 displays a message to the end user indicating that he has the right to view the session at a cost of, say, 25 tokens, as read from the ECM or to connect to the communications servers 3022 to purchase the event. Using the tokens, if the end user answers "yes" (by means of remote controller 2026 (see Figure 2)) the decoder 2020 sends the ECM to the smartcard, the smartcard decreases the wallet of the smartcard 3020 by 25 tokens, writes the session ID 168, the session mode 169, the session index 170 and the session view 171 in the PPV event description 167 and extracts and deciphers the control word from the ECM.

In the "pre-book" mode, an EMM will be passed to the smartcard 3020 so that the smartcard will write the session ID 168, the session mode 169, the session index 170 and the session view 171 in the PPV event description 167 using the EMM.

The session index 170 can be set to differentiate one broadcast from the other. This feature permits authorization to be given for a subset of broadcasts, for example, 3 times out of 5 broadcasts. As soon as an ECM with a session index different from the current session index 170 stored in the PPV event description 167 is passed to the smartcard, the number of the session view 171 is decreased by one. When the session view reaches zero, the smartcard will refuse to decipher an ECM with a different session index to the current session index.

The initial value of the session view depends only on the way in which the broadcast supplier wishes to define the event to which it relates; the session view for a respective event may take any value.

The microprocessor 110 in the smartcard implements a counting and a comparison program to detect when the limit to the number of viewings of a particular program

has been reached.

All of the session ID 168, the session mode 169, the session index 170 and the session view 171 in the PPV event description 167 may be extracted from the smartcard using the "call-back" procedure as described previously.

Each receiver/decoder 2020 contains an identifier which may either identify uniquely that receiver/decoder or identify its manufacturer or may classify it in some other way in order to enable it to work only with a particular individual smartcard, a particular class of smartcards made by the same or a corresponding manufacturer or any other class of smartcards which are intended for use with that class of receiver/decoders exclusively.

In this manner the receiver/decoders 2020 which have been supplied by one broadcast supplier to the consumer are protected against the use of non-authorised daughter smartcards 3020.

Additionally or alternatively to this first "handshake" between the smartcard and the receiver, the EEPROM of the smartcard 3020 could contain a field or bitmap describing the categories of receiver/decoders 2020 with which it can function. These could be specified either during the manufacture of the smartcard 3020 or by a specific EMM.

The bitmap stored in the smartcard 3020 typically comprises a list of up to 80 receiver/decoders, each identified with a corresponding receiver/decoder ID with which the smartcard may be used. Associated with each receiver/decoder is a level "1" or "0" indicating whether the smartcard may be used with the receiver/decoder or not, respectively. A program in the memory 2024 of the receiver/decoder searches for the identifier of the receiver/decoder in the bitmap stored in the smartcard. If the identifier is found, and the value associated with the identifier is "1", then the smartcard is "enabled"; if not, then the smartcard will not function with that receiver/decoder.

In addition, if, typically because of an agreement between operators, it is desired to authorize the use of other smartcards in a particular receiver/decoder, specific EMMs will be sent to those smartcards to change their bitmap via the transponder 2014.

Each broadcast supplier may differentiate his subscribers according to certain predetermined criteria. For example, a number of subscribers may be classed as "VIPs". Accordingly, each broadcast supplier may divide his subscribers into a plurality of subsets, each subset comprising any number of subscribers.

The subset to which a particular subscriber belongs is set in the SMS 3004. In turn, the SAS 3002 transmits an EMM to the subscriber which writes information (typically of length 1 byte) concerning the subset to which the subscriber belongs into the relevant operator data zone, say 154, of the EEPROM of the smartcard. In turn, as events are broadcast by the broadcast supplier, an ECM, typically of 256 bits, is transmitted with the event and indicating which of the subsets of subscribers may view the event. If, according to the information stored in the operator zone, the subscriber does not have the right to view the event, as determined by the ECM, programme viewing is denied.

This facility may be used, for example, to switch off all of a given operator's smartcards in a particular geographical region during the transmission of a particular program, in particular a program relating to a sports fixture taking place in that geographical region. In this manner football clubs and other sport bodies can sell broadcasting rights outside their locality whilst preventing local supporters from viewing the fixture on television. In this manner the local supporters are encouraged to buy tickets and attend the fixture.

Each of the features associated with zones 151 to 172 is considered to be a separate invention independent of the dynamic creation of zones.

It will be understood that the present invention has been described above purely by way of example, and modifications of detail can be made within the scope of the

invention.

Each feature disclosed in the description, and (where appropriate) the claims and drawings may be provided independently or in any appropriate combination.

In the aforementioned preferred embodiments, certain features of the present invention have been implemented using computer software. However, it will of course be clear to the skilled man that any of these features may be implemented using hardware. Furthermore, it will be readily understood that the functions performed by the hardware, the computer software, and such like are performed on or using electrical and like signals.

Cross reference is made to our co-pending applications, all bearing the same filing date, and entitled Signal Generation and Broadcasting (Attorney Reference no. PC/ASB/19707), Smartcard for use with a Receiver of Encrypted Broadcast Signals, and Receiver (Attorney Reference No. PC/ASB/19708), Broadcast and Reception System and Conditional Access System therefor (Attorney Reference No. PC/ASB/19710), Downloading a Computer File from a Transmitter via a Receiver/Decoder to a Computer (Attorney Reference No. PC/ASB/19711), Transmission and Reception of Television Programmes and Other Data (Attorney Reference No. PC/ASB/19712), Downloading Data (Attorney Reference No. PC/ASB/19713), Computer Memory Organisation (Attorney Reference No. PC/ASB/19714), Television or Radio Control System Development (Attorney Reference No. PC/ASB/19715), Extracting Data Sections from a Transmitted Data Stream (Attorney Reference No. PC/ASB/19716), Access Control System (Attorney Reference No. PC/ASB/19717), Data Processing System (Attorney Reference No. PC/ASB/19718), and Broadcast and Reception System, and Receiver/Decoder and Remote Controller therefor (Attorney Reference No. PC/ASB/19720). The disclosures of these documents are incorporated herein by reference. The list of applications includes the present application.

CLAIMS

1. A conditional access system comprising:

means for generating a plurality of messages; and

means for receiving the messages, said receiving means being adapted to communicate with said generating means via a communications server connected directly to said generating means.

2. A conditional access system according to Claim 1, wherein said message is an entitlement message for transmission to the receiving means, said generating means being adapted to generate entitlement messages in response to data received from said receiving means.

3. A conditional access system according to Claim 1 or 2, wherein said generating means is arranged to transmit a message as a packet of digital data to said receiving means either via said communications server or via a satellite transponder.

4. A conditional access system according to any preceding claim, wherein said receiving means is connectable to said communications server via a modem and telephone link.

5. A conditional access system for affording conditional access to subscribers, comprising:

a subscriber management system;

a subscriber authorization system coupled to the subscriber management system; and

a communications server; said server being connected directly to the subscriber authorization system.

6. A conditional access system according to Claim 5, further comprising a receiver/decoder for the subscriber, the receiver/decoder being connectable to said communications server, and hence to said subscriber authorization system, via a

modem and telephone link.

7. A broadcast and reception system including a conditional access system according to any preceding claim.

8. A broadcast and reception system comprising:

means for generating a plurality of entitlement messages relating to broadcast programs;

means for receiving said messages from said generating means; and

means for connecting the receiving means to the generating means to receive said messages, said connecting means being capable of effecting a dedicated connection between the receiving means and the generating means.

9. A broadcast and reception system comprising:

means for generating a plurality of entitlement messages relating to broadcast programs;

means for receiving said messages from said generating means via a modem; and

means for connecting said modem to said generating means and said receiving means.

10. A broadcast and reception system according to Claim 9, wherein said generating means is connected to said modem via a communications server.

11. A broadcast and reception system according to Claim 9 or 10, wherein said receiving means is adapted to communicate with said generating means via said modem and connecting means.

12. A broadcast and reception system according to Claim 11, wherein said receiving means comprises means for reading a smartcard insertable thereinto by an end user, the smartcard having stored therein data to initiate automatically the transmission of a message from said receiving means to said generating means upon

insertion of the smartcard by the end user.

13. A broadcast and reception system according to Claim 11or 12, further comprising a voice link to enable the end user of the broadcast and reception system to communicate with the generating means.

14. A broadcast and reception system according to any of Claims 8 to 13, wherein said receiving means comprises a receiver/decoder comprising means for receiving a compressed MPEG-type signal, means for decoding the received signal to provide a television signal and means for supplying the television signal to a television.

15. A broadcast and reception system, comprising, at the broadcast end:

a broadcast system including means for broadcasting a callback request;

and at the reception end:

a receiver including means for calling back the broadcast system in response to the callback request.

16. A system according to Claim 15, wherein the means for calling back the broadcast system includes a modem connectable to a telephone system.

17. A system according to Claim 15 or 16, wherein the means for calling back the broadcast system is arranged to transfer to the broadcast system information concerning the receiver.

18. A system according to Claim 17, wherein the broadcast system includes means for storing the information.

19. A system according to any of Claims 15 to 18, wherein the broadcast means is arranged to broadcast a callback request which includes a command that the callback be made at a given time, and the means for calling back the broadcast system is arranged to respond to said command.

20. A system according to any of Claims 15 to 19, wherein the broadcasting means is arranged to broadcast as the callback request one or more entitlement messages for broadcast.

21. A system according to any of Claims 15 to 20, wherein the broadcast system includes means for generating a check message and passing this to the receiver, the receiver includes means for encrypting the check message and passing this to the broadcast system, and the broadcast system further includes means for decrypting the check message received from the receiver and comparing this with the original check message.

22. A conditional access system or a broadcast and reception system substantially as herein described with reference to and as illustrated in the accompanying drawings, and especially Figures 12, 13 or 14 thereof.

Fig.1.

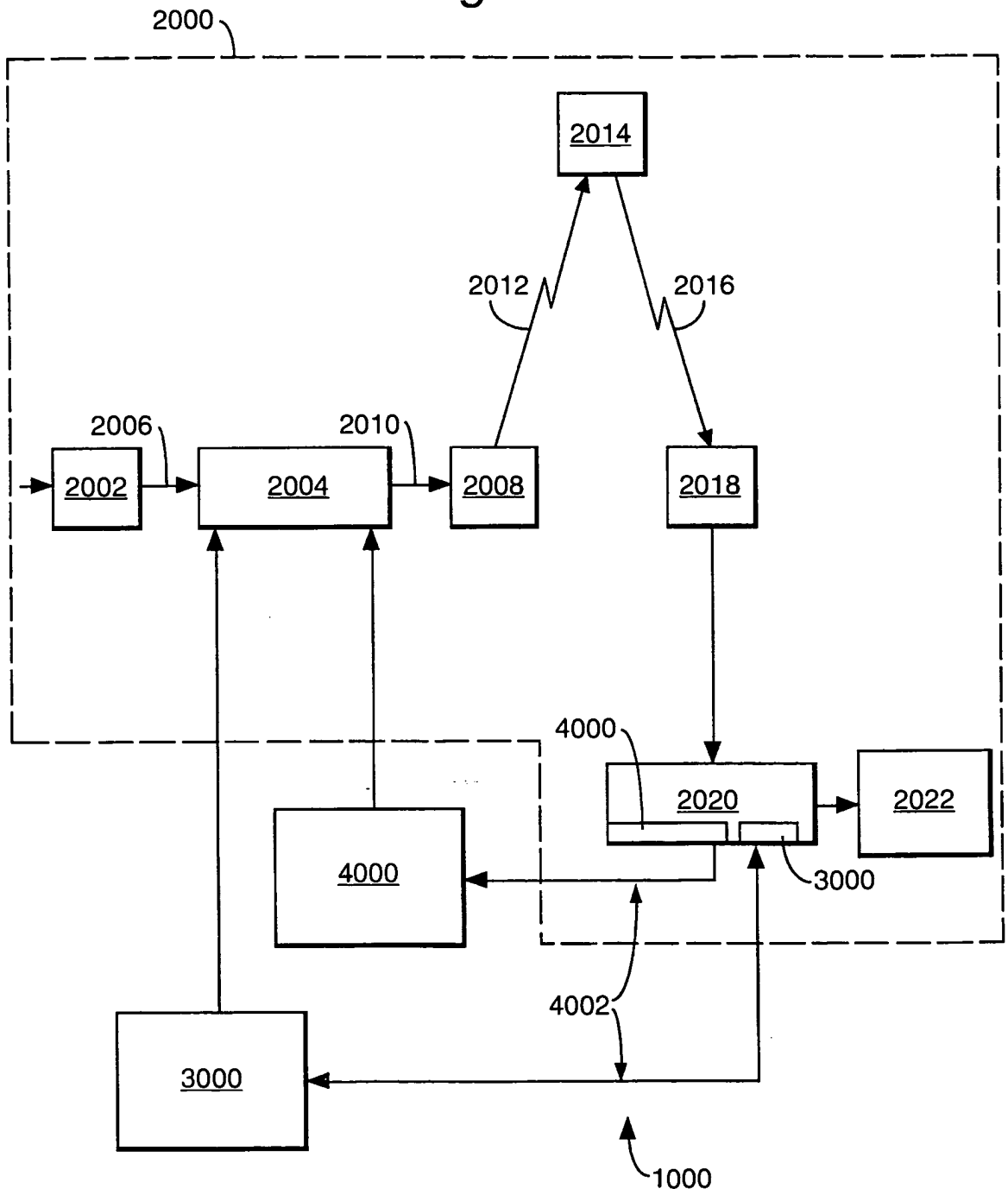
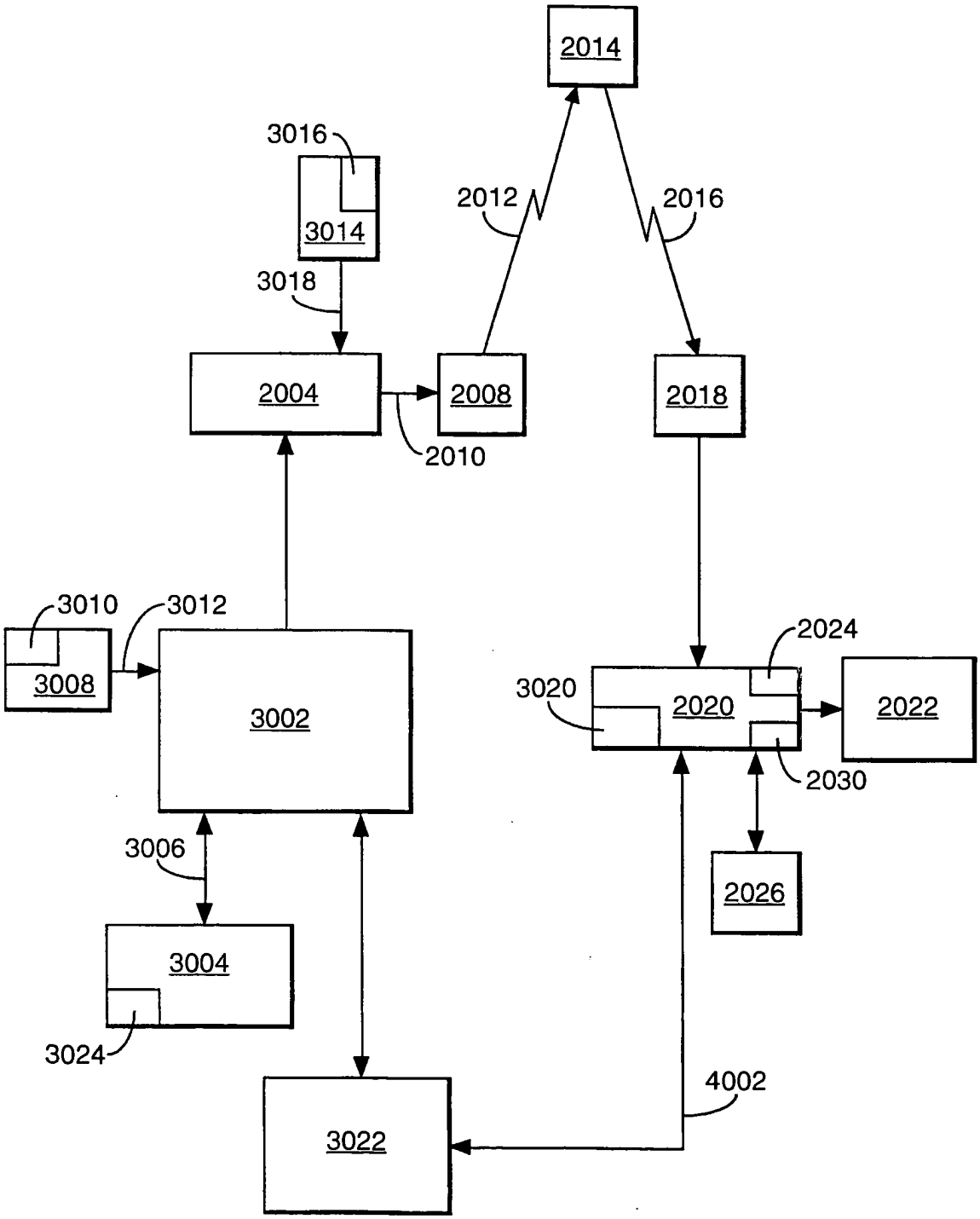


Fig.2.



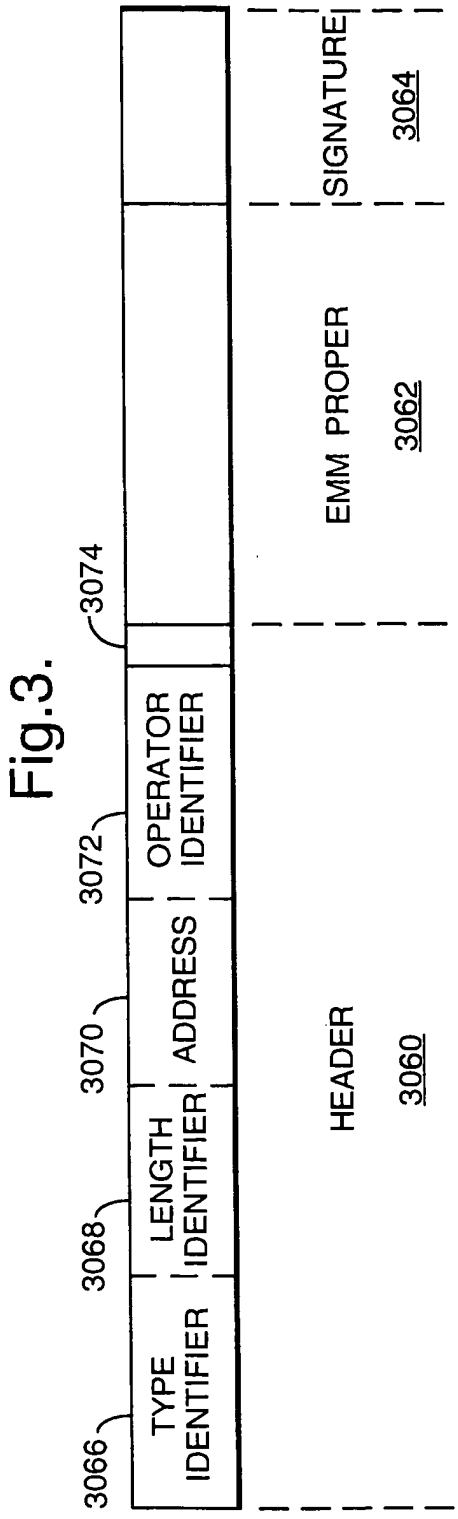


Fig.3.

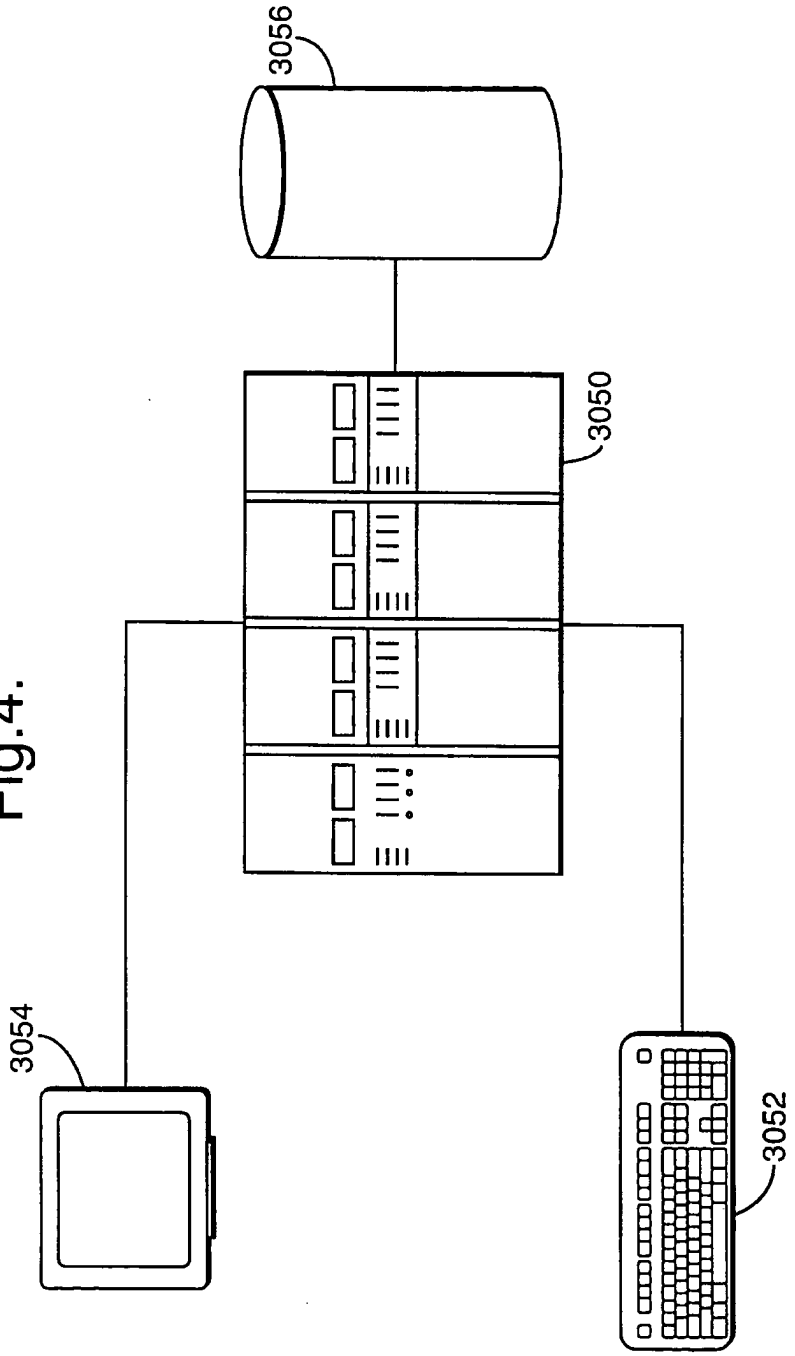


Fig.4.

Fig.5.

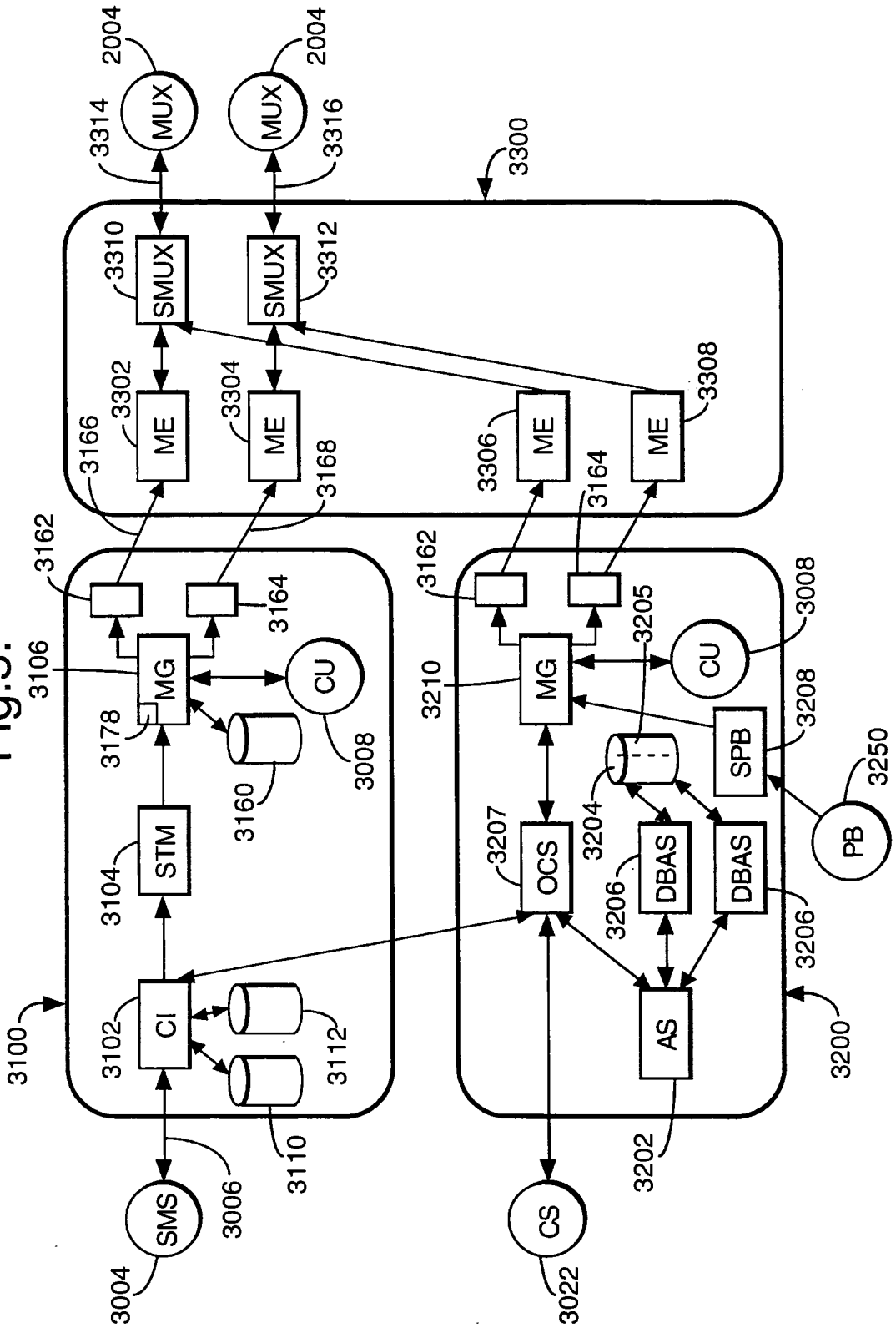


Fig.6.

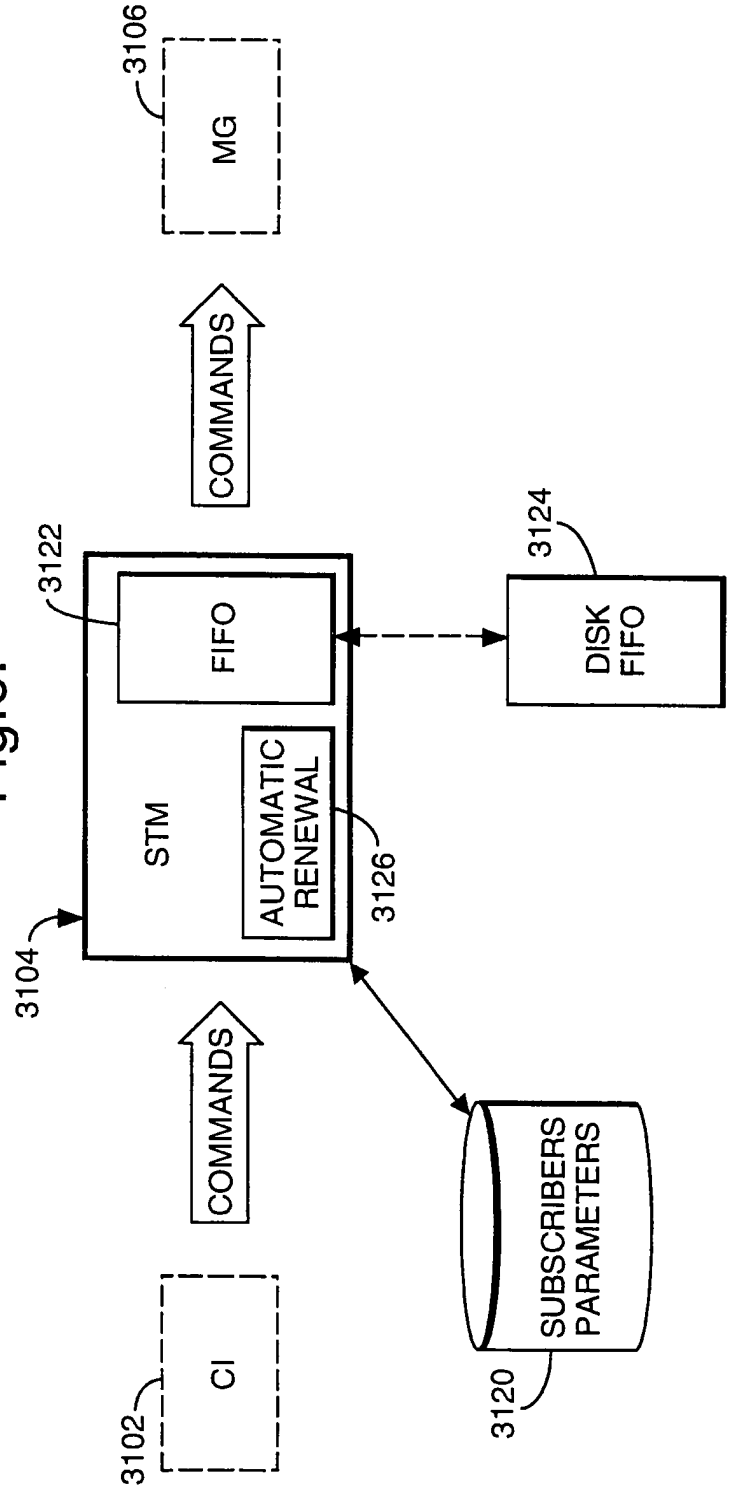


Fig.7.

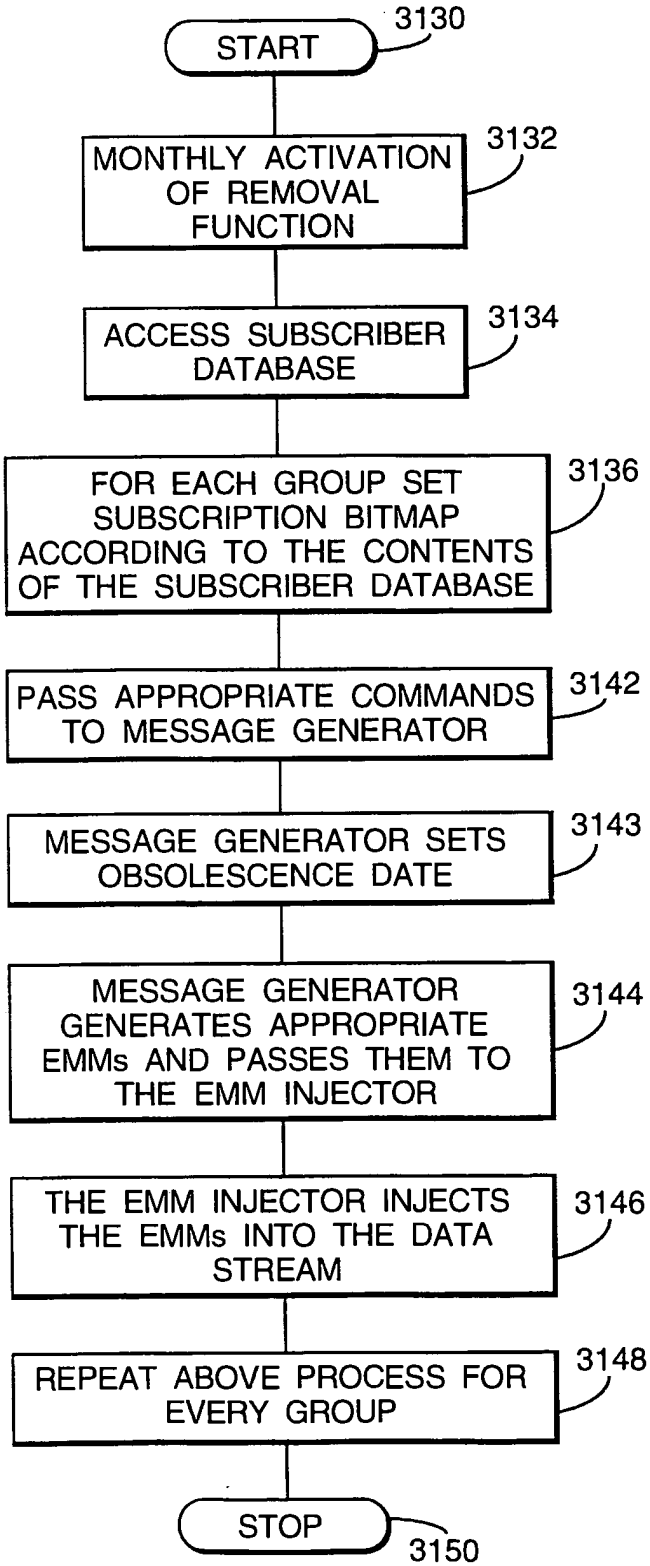


Fig.8.

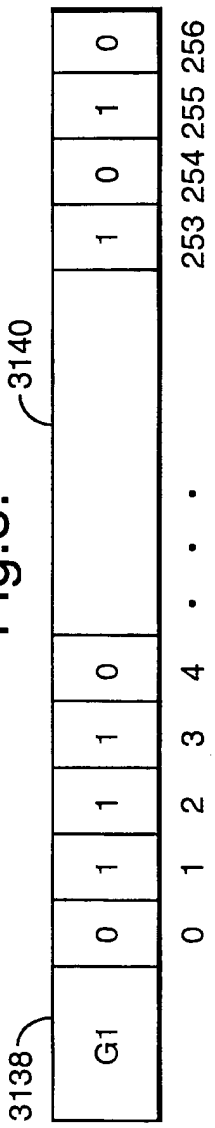


Fig.9.

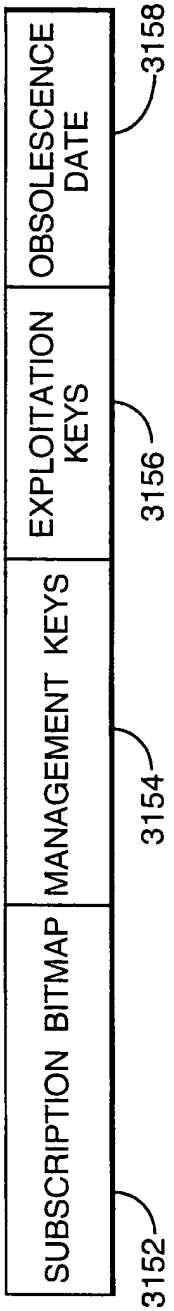
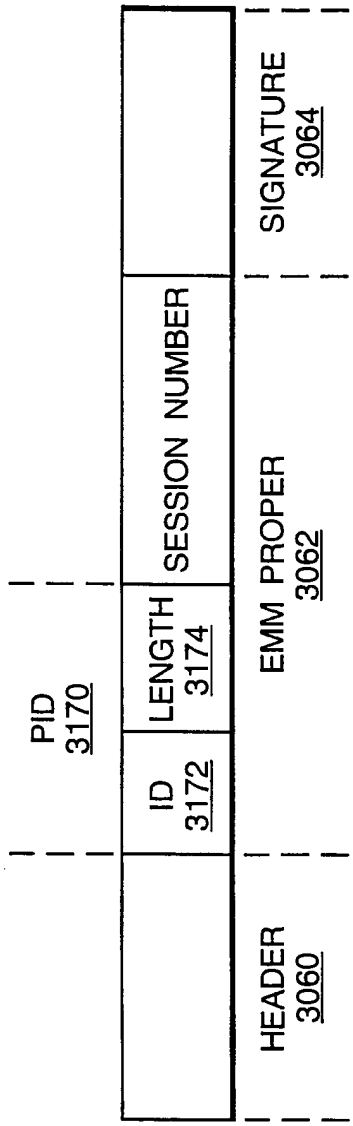
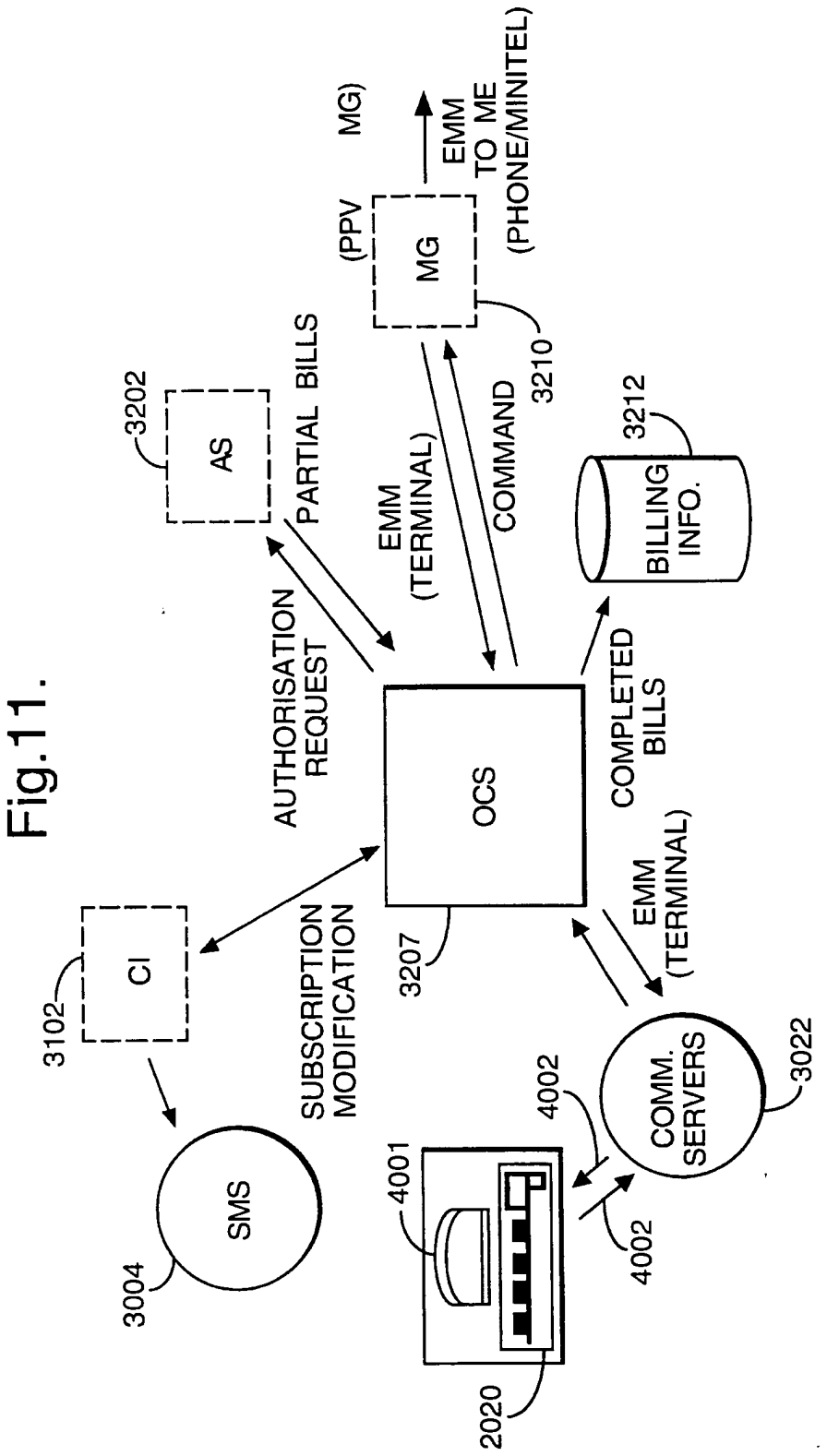


Fig.10.





SUBSTITUTE SHEET (RULE 26)

Fig.12.

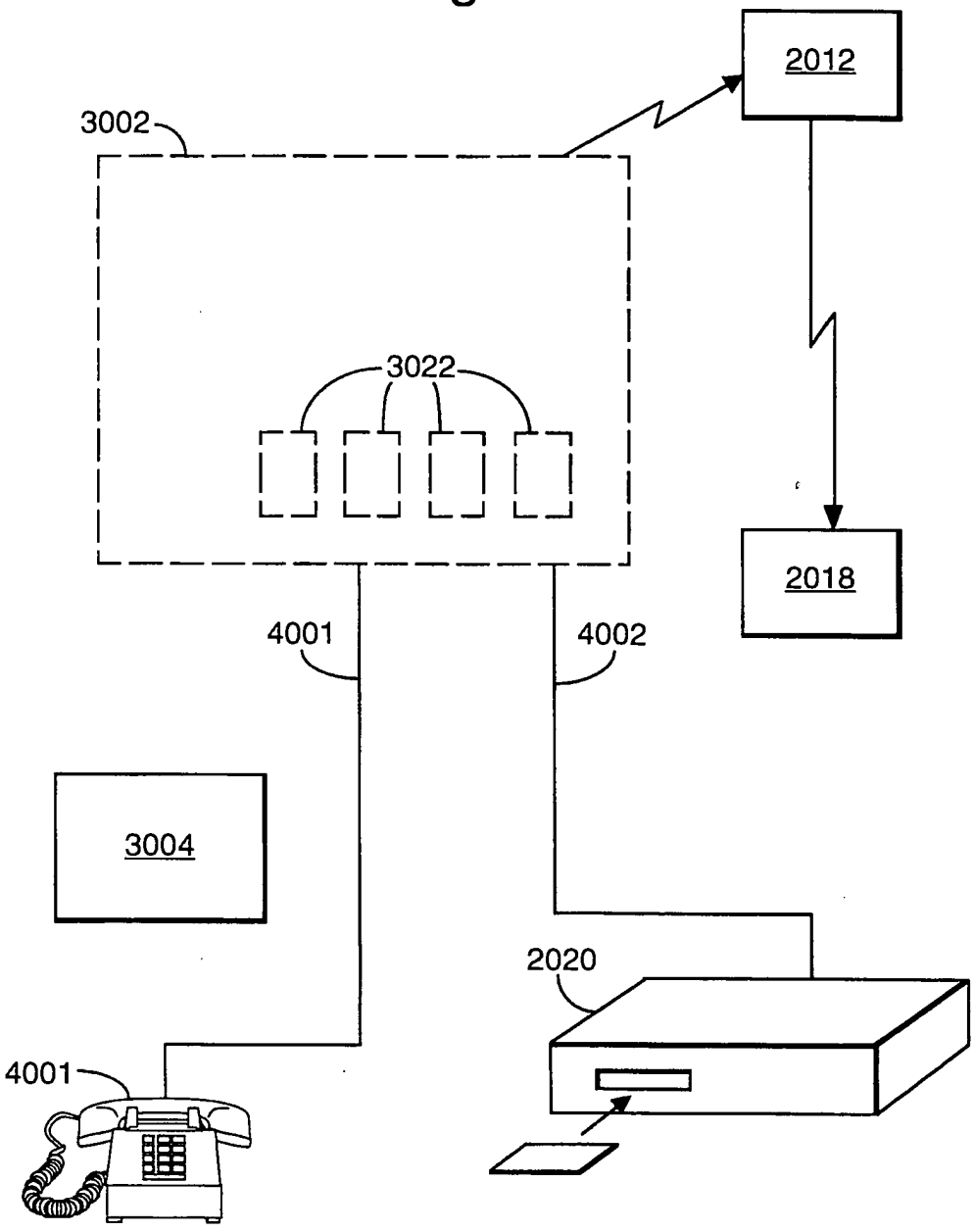


Fig.13.

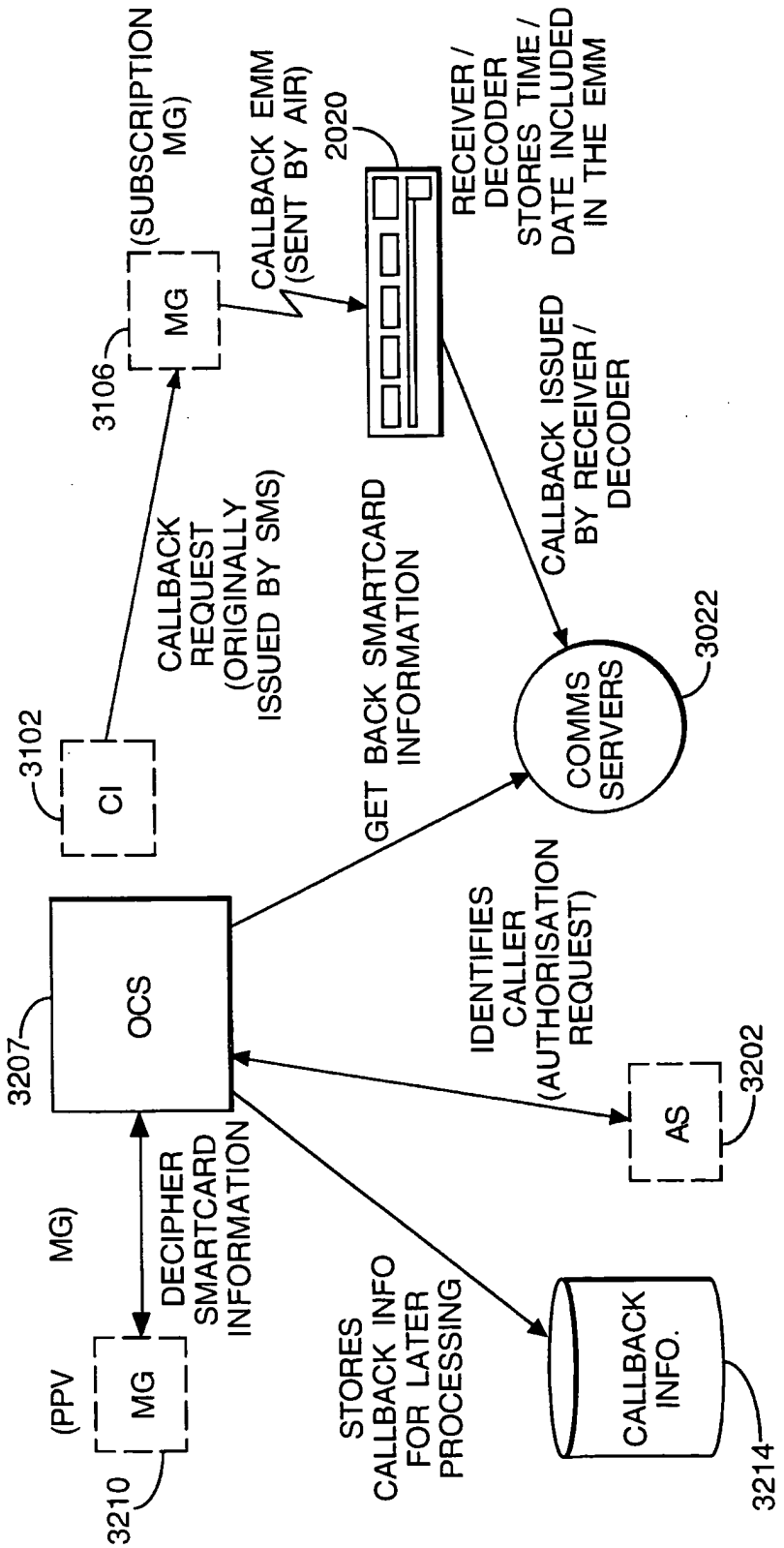


Fig.14.

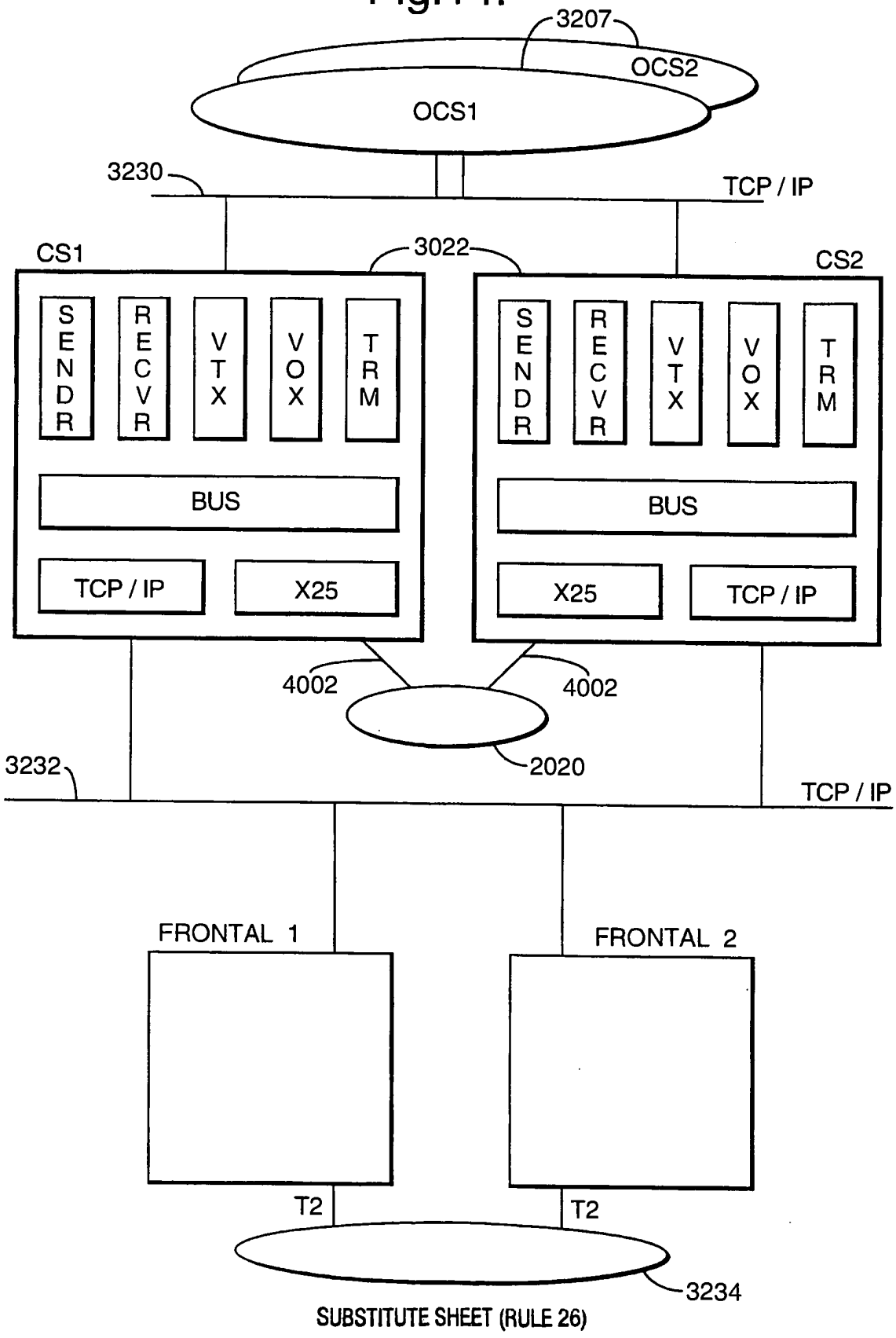


Fig.15.

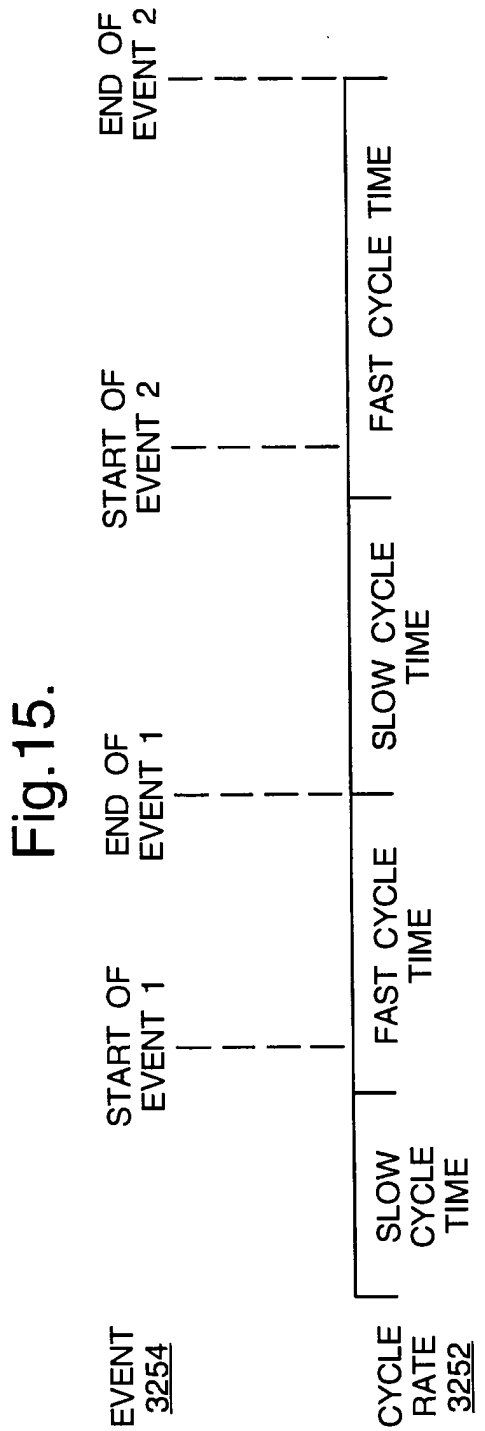


Fig.16.

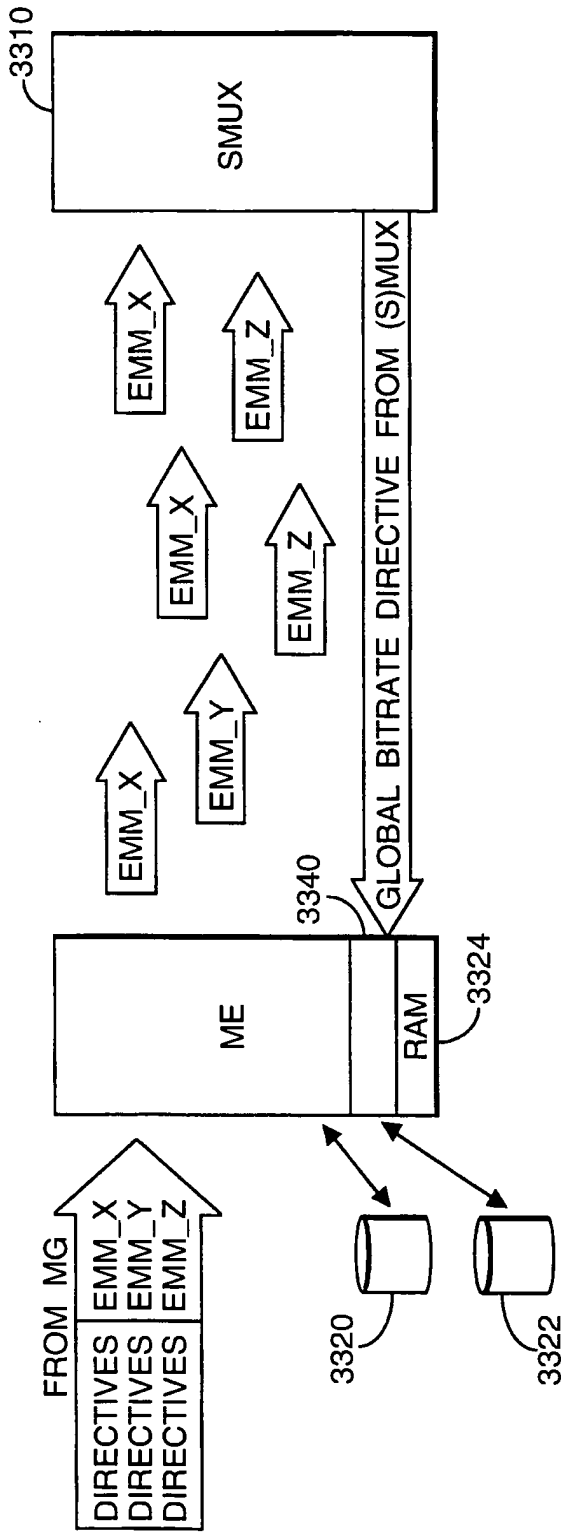


Fig.17.

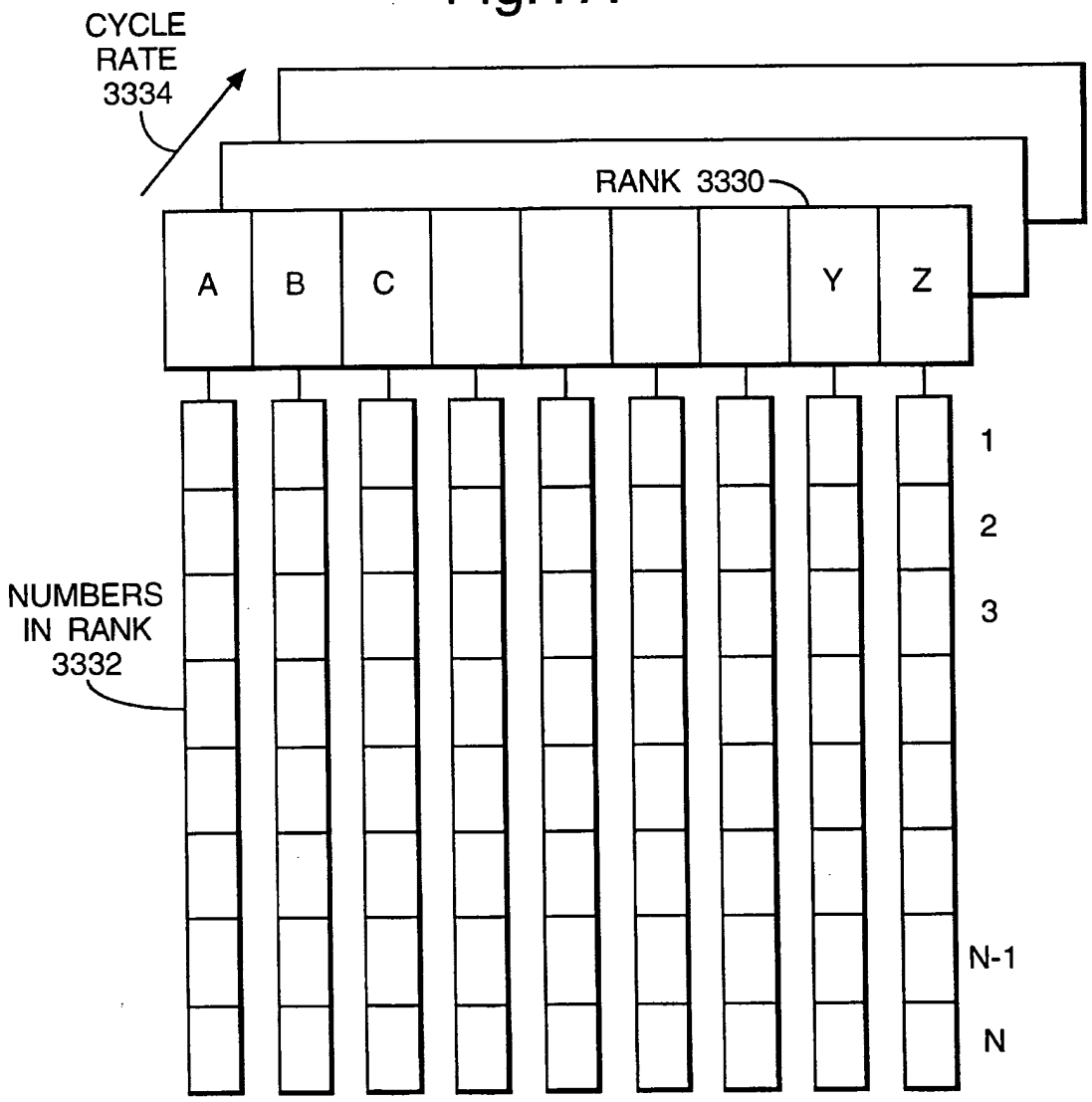


Fig.18.

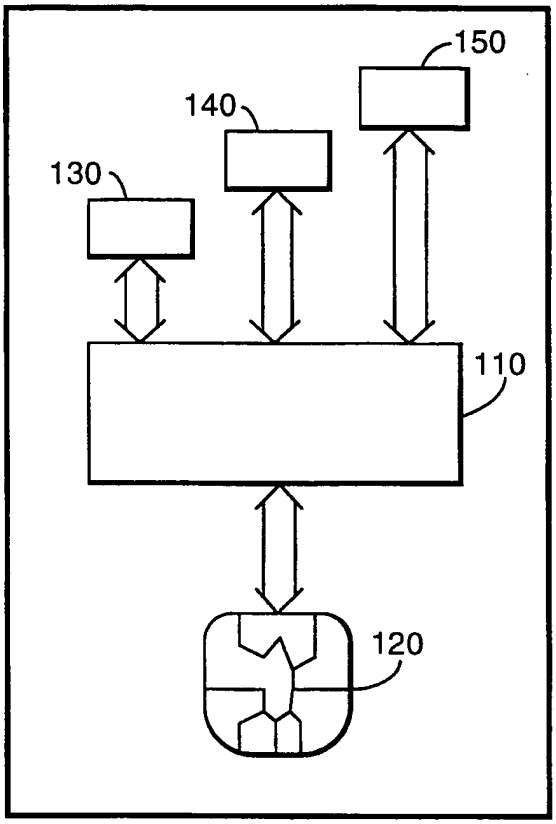


Fig.20.

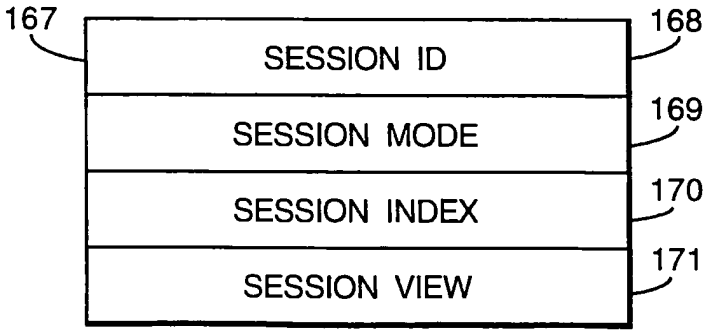
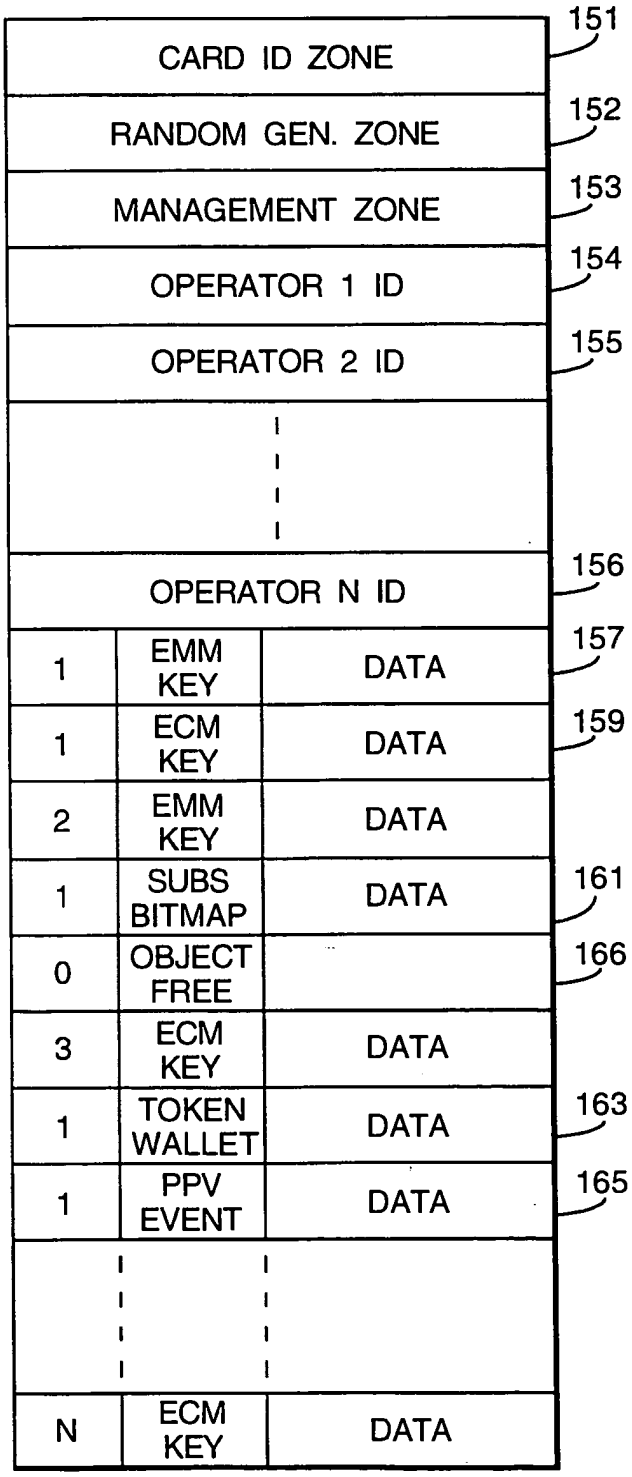


Fig.19.



INTERNATIONAL SEARCH REPORT

Intern: al Application No
PCT/EP 97/02108

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04N7/16 H04N7/167

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)
IPC 6 H04N

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"FUNCTIONAL MODEL OF A CONDITIONAL ACCESS SYSTEM" EBU REVIEW- TECHNICAL, no. 266, 21 December 1995, pages 64-77, XP000559450 see the whole document ---	1-12, 15-19, 21,22
X	WO 94 14284 A (DISCOVERY COMMUNICAT INC) 23 June 1994 see page 8, line 8 - page 14, line 23 see page 18, line 28 - page 21, line 19 see page 24, line 25 - page 29, line 31 see page 33, line 8 - line 17 see figures 1-11 --- -/--	1-12, 14-17

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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

1

Date of the actual completion of the international search	Date of mailing of the international search report
11 November 1997	18. 11. 97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Van der Zaal, R

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/EP 97/02108

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 144 663 A (KUDELSKI ANDRE ET AL) 1 September 1992 see column 2, line 5 - line 23 see column 3, line 6 - column 4, line 65 see column 5, line 62 - column 8, line 58 see figures 1-11 -----	1-12

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Internal Application No
PCT/EP 97/02108

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9414284 A	23-06-94	AU 5732994 A	04-07-94
		AU 5733094 A	04-07-94
		AU 5733194 A	04-07-94
		AU 5733294 A	04-07-94
		AU 5736394 A	04-07-94
		AU 5845894 A	22-06-94
		AU 5869894 A	04-07-94
		CA 2151458 A	23-06-94
		CN 1093211 A	05-10-94
		CN 1090451 A	03-08-94
		CN 1090452 A	03-08-94
		CN 1096151 A	07-12-94
		CN 1090453 A	03-08-94
		CN 1090454 A	03-08-94
		EP 0673578 A	27-09-95
		EP 0673579 A	27-09-95
		EP 0673580 A	27-09-95
		EP 0673581 A	27-09-95
		EP 0673582 A	27-09-95
		EP 0673583 A	27-09-95
		EP 0674824 A	04-10-95
		IL 107908 A	10-01-97
		IL 107909 A	15-04-97
		IL 107910 A	10-06-97
		IL 107912 A	18-02-97
		IL 107913 A	15-04-97
		JP 8510869 T	12-11-96
		JP 8506938 T	23-07-96
		JP 8506939 T	23-07-96
		JP 8506940 T	23-07-96
		JP 8506941 T	23-07-96
		JP 8506942 T	23-07-96
		NZ 259146 A	26-05-97
		NZ 259147 A	26-05-97
		NZ 259148 A	26-11-96
		WO 9413107 A	09-06-94
		WO 9414279 A	23-06-94
		WO 9414280 A	23-06-94
		WO 9414281 A	23-06-94
		WO 9414282 A	23-06-94

INTERNATIONAL SEARCH REPORT

...ormation on patent family members

Interna... Application No
PCT/EP 97/02108

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9414284 A		WO 9414283 A	23-06-94
		US 5559549 A	24-09-96
		US 5600364 A	04-02-97
		US 5659350 A	19-08-97
US 5144663 A	01-09-92	AU 599646 B	26-07-90
		AU 7157887 A	22-10-87
		DE 3751410 D	24-08-95
		DE 3751410 T	11-04-96
		EP 0243312 A	28-10-87
		EP 0626793 A	30-11-94
		ES 2076931 T	16-11-95
		JP 2610260 B	14-05-97
		JP 63023488 A	30-01-88
		JP 2520217 B	31-07-96
		JP 5244591 A	21-09-93

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 25/02 — A1

(11) International Publication Number: **WO 98/19431**

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/19676

(22) International Filing Date: 27 October 1997 (27.10.97)

(30) Priority Data:
08/741,273 30 October 1996 (30.10.96) US

(71) Applicant: QUALCOMM INCORPORATED [US/US]; 6455 Lusk Boulevard, San Diego, CA 92121 (US).

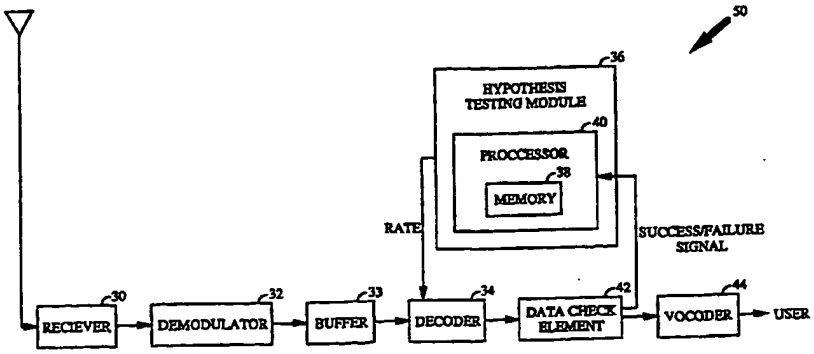
(72) Inventors: TIEDEMANN, Edward, G., Jr.; 4350 Bromfield Avenue, San Diego, CA 92122 (US). LIN, Yu-Chuan; 585 W. 63rd Avenue, Vancouver, British Columbia V6P 2G7 (CA).

(74) Agents: OGROD, Gregory, D. et al.; Qualcomm Incorporated, 6455 Lusk Boulevard, San Diego, CA 92121 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

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: METHOD AND APPARATUS FOR DECODING VARIABLE RATE DATA



RECIEVING SYSTEM

(57) Abstract

A system and method for determining the data rate of a frame of data at a receiver (50) of a variable rate communications system. A vocoder at a transmitter encodes a frame of data at one of the rates of a predetermined set of rates. The data rate is dependent on the speech activity during the time frame of the data. The data frame is also formatted with overhead bits, including bits for error detection and detection. At the receiver (50), the data rate for the frame is determined based on hypothesis testing. Because the data rate is based on speech activity, a hypothesis test may be designed based on the statistics of speech activity. The received data frame is first decoded by a decoder (34) into information bits at the most probable rate as provided by the hypothesis testing module (36). Data check element (42) generates error metrics for the decoded information bits. If the error metrics indicate that the information bits are of good quality, then the information bits are presented to a vocoder (44) at the receiver to be processed for interface with the user. If the error metrics indicate that the information bits have not been properly decoded, then decoder (34) decodes the received data frame at the other rates of the set of rates until the actual data rate is determined.

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.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR DECODING VARIABLE RATE DATA

BACKGROUND OF THE INVENTION

I. Field of the Invention

The present invention relates to digital communications. More particularly, the present invention relates to a novel and improved system and method for determining, at a receiver of a variable rate communication system, the rate at which data has been encoded for transmission.

II. Description of the Related Art

The use of code division multiple access (CDMA) modulation techniques is one of several techniques for facilitating communications in which a large number of system users are present. Although other techniques such as time division multiple access (TDMA), frequency division multiple access (FDMA), and AM modulation schemes such as amplitude companded single sideband (ACSSB) are known, CDMA has significant advantages over these other techniques. The use of CDMA techniques in a multiple access communication system is disclosed in U.S. Pat. No. 4,901,307, entitled "SPREAD SPECTRUM MULTIPLE ACCESS COMMUNICATION SYSTEM USING SATELLITE OR TERRESTRIAL REPEATERS," assigned to the assignee of the present invention and incorporated by reference herein.

CDMA systems often employ a variable rate vocoder to encode data so that the data rate can be varied from one data frame to another. An exemplary embodiment of a variable rate vocoder is described in U.S. Pat. No. 5,414,796, entitled "VARIABLE RATE VOCODER," assigned to the assignee of the present invention and incorporated by reference herein. The use of a variable rate communications channel reduces mutual interference by eliminating unnecessary transmissions when there is no useful speech to be transmitted. Algorithms are utilized within the vocoder for generating a varying number of information bits in each frame in accordance with variations in speech activity. For example, a vocoder with a set of four rates may produce 20 millisecond data frames containing 16, 40, 80, or 171 information bits, depending on the activity of the speaker. It is desired to transmit each data frame in a fixed amount of time by varying the transmission rate of communications.

Additional details on the formatting of the vocoder data into data frames are described in U.S. Pat. No. 5,511,073, entitled "METHOD AND APPARATUS FOR THE FORMATTING OF DATA FOR TRANSMISSION," assigned to the assignee of the present invention and herein incorporated by reference. The data frames may be further processed, spread spectrum modulated, and transmitted as described in U.S. Pat. No. 5,103,459, entitled "SYSTEM AND METHOD FOR GENERATING WAVEFORMS IN A CDMA CELLULAR TELEPHONE SYSTEM," assigned to the assignee of the present invention and incorporated by reference herein.

Variable rate systems can be developed which include explicit rate information. If the rate is included as part of a variable rate frame, then the rate is not recoverable until after the frame has already been properly decoded, at which point the rate has already been determined. Rather than including the rate in a variable rate frame, the rate could instead be sent in a non-variable rate portion of the frame. However, only a few bits are typically needed to represent the rate, and these bits cannot be efficiently encoded and interleaved in order to provide error protection for fading communications channels. Furthermore, the rate information is only available after some decoding delay and are subject to error.

Alternatively, variable rate systems can be developed which do not include explicit rate information. One technique for the receiver to determine the rate of a received data frame where the rate information is not explicitly included in the frame is described in copending U.S. Patent Application Serial No. 08/233,570, entitled "METHOD AND APPARATUS FOR DETERMINING DATA RATE OF TRANSMITTED VARIABLE RATE DATA IN A COMMUNICATIONS RECEIVER," filed April 26, 1994, assigned to the assignee of the present invention, and incorporated by reference. Another technique is described in copending U.S. Patent Application Serial No. 08/126,477, entitled "MULTIRATE SERIAL VITERBI DECODER FOR CODE DIVISION MULTIPLE ACCESS SYSTEM APPLICATIONS," filed Sept. 24, 1993, assigned to the assignee of the present invention, and incorporated by reference. According to these techniques, each received data frame is decoded at each of the possible rates. Error metrics, describing the quality of the decoded symbols for each frame decoded at each rate, are provided to a processor. The error metrics may include Cyclic Redundancy Check (CRC) results, Yamamoto Quality Metrics, and Symbol Error Rates. These error metrics are well-known in communications systems. The processor analyzes the error metrics and

determines the most probable rate at which the incoming symbols were transmitted.

Decoding each received data frame at each possible data rate will eventually generate the desired decoded data. However, the search through all possible rates is not the most efficient use of processing resources in a receiver. Also, as higher transmission rates are used, power consumption for determining the transmission rate also increases. This is because there are more bits per frame to be processed. Furthermore, as technology evolves, variable rate systems may utilize larger sets of data rates for communicating information. The use of larger sets of rates will make the exhaustive decoding at all possible rates infeasible. In addition, the decoding delay will not be tolerable for some system applications. Consequently, a more efficient rate determination system is needed in a variable rate communications environment. These problems and deficiencies are clearly felt in the art and are solved by the present invention in the manner described below.

SUMMARY OF THE INVENTION

The present invention is a novel and improved system and method for determining the transmission rate of communications in a variable rate communications system. In a variable rate system, the data rate at which a data frame is encoded may be based on the speech activity during the time frame. Because the characteristics of speech are known, probability functions may be defined for the data rates which are dependent on the characteristics of speech. The probability functions may in addition be dependent on the measured statistics of the received data frames. Furthermore, hypothesis tests can be designed based on the probability functions to determine the most likely data rate of a received frame of data. These probability functions may be dependent on the selected service option. For example, the probability functions for data services will be different than for voice services.

At the receiver of the present invention, a processor causes a decoder to decode the received frame of data into information bits at the most probable rate as determined by the hypothesis test. The most probable rate may, for example, be the rate of the previous frame of data. The decoder also generates error metrics for the decoded information bits. The decoded bits and the error metrics are provided to a data check element which checks the decoded bits for correctness. If the error metrics indicate that the

decoded information bits are of good quality, then the information bits are provided to a vocoder which further processes the data and provides speech to the user. Otherwise, a failure signal is presented to the processor. The processor then causes the decoder to decode the received frame of data at other data rates until the correct data rate is found.

BRIEF DESCRIPTION OF THE DRAWINGS

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify correspondingly throughout and wherein:

FIG. 1 is a schematic overview of an exemplary CDMA cellular telephone system;

FIG. 2 is a block diagram of a variable rate receiving system with particular reference to the rate determination features of the present invention;

FIGS. 3 and 4 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test designates the rate of the previous frame of data as the most probable rate for the current frame of data;

FIGS. 5 and 6 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test is based on the a priori probability distribution of the data rates; and

FIGS. 7 and 8 are flow charts illustrating two embodiments of the processing steps involved in rate determination wherein the hypothesis test is based on the conditional probability distribution of the data rates.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

An exemplary cellular mobile telephone system in which the present invention is embodied is illustrated in FIG. 1. For purposes of example this system is described herein within the context of a CDMA cellular communications system. However, it should be understood that the invention is applicable to other types of communication systems such as personal communication systems (PCS), wireless local loop, private branch exchange (PBX) or other known systems. Furthermore systems utilizing other well known transmission modulation schemes such as TDMA and

FDMA as well as other spread spectrum systems may employ the present invention.

An exemplary cellular system in which the rate determination system of the present invention may be implemented is illustrated in FIG. 1. In FIG. 1, system controller and switch **10** typically include appropriate interface and processing hardware for providing system control information to the cell-sites. Controller **10** controls the routing of telephone calls from the public switched telephone network (PSTN) to the appropriate cell-site for transmission to the appropriate mobile unit. Controller **10** also controls the routing of calls from the mobile units via at least one cell-site to the PSTN. Controller **10** may direct calls between mobile users via the appropriate cell-site stations since such mobile units do not typically communicate directly with one another.

Controller **10** may be coupled to the cell-sites by various means such as dedicated telephone lines, optical fiber links or by radio frequency communications. In FIG. 1, two exemplary cell-sites, **12** and **14**, along with two exemplary mobile units, **16** and **18**, which include cellular telephones, are illustrated. Arrows **20a-20b** and **22a-22b** respectively define the possible communication links between cell-site **12** and mobile units **16** and **18**. Similarly, arrows **24a-24b** and arrows **26a-26b** respectively define the possible communication links between cell-site **14** and mobile units **18** and **16**.

The cellular system illustrated in FIG. 1 may employ a variable rate data channel for communications between cell-sites **12, 14** and mobile units **16, 18**. By example, a vocoder (not shown) may encode sampled voice information into symbols at four different rates according to the IS-95-A standard. The IS-95-A Mobile Station-Base Station Compatibility Standard for Dual Mode Wideband Spread Spectrum Cellular System has been provided by the telecommunications industry association (TIA) for CDMA communications. According to IS-95-A, speech is encoded at approximately 8,550 bits per second (bps), 4,000 bps, 2,000 bps, and 800 bps based on voice activity during a 20 millisecond (ms) frame of data. Each frame of vocoder data is then formatted with overhead bits as 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps data frames for transmission. The 9,600 bps frame is referred to as a full rate frame; the 4,800 bps data frame is referred to as a half rate frame; a 2,400 bps data frame is referred to as a quarter rate frame; and a 1,200 bps data frame is referred to as an eighth rate frame. Although this example describes a set of four data rates of the IS-95-A standard, it should be recognized that the present invention is equally applicable in systems

utilizing different transmission rates and/or a different number of variable rates.

By encoding each frame of data based on speech activity, data compression is achievable without impacting the quality of the reconstructed speech. Since speech inherently contains periods of silence, i.e. pauses, the amount of data used to represent these periods can be reduced. Variable rate vocoding most effectively exploits this fact by reducing the data rate for these periods of silence. In a system with a set of four rates as described above, periods of active speech will generally be encoded at full rate, while periods of silence will generally be encoded at eighth rate. Most frames (about 80-90%) are encoded at full or eighth rate. Transitions between active speech and periods of silence will typically be encoded at half or quarter rate. An exemplary encoding technique which compresses data based on speech activity is described in U.S. Pat. No. 5,511,073 mentioned above.

The data frames are also formatted with overhead bits, which generally will include additional bits for error correction and detection, such as Cyclic Redundancy Check (CRC) bits. The CRC bits can be used by the decoder to determine whether or not a frame of data has been received correctly. CRC codes are produced by dividing the data block by a predetermined binary polynomial as is described in detail in IS-95-A.

In a preferred embodiment, each frame of symbol data is interleaved by an interleaver, preferably on a bit level basis, to increase time diversity for purposes of error detection. The formatted data frames undergo further processing, which include modulation, frequency upconversion to the radio frequency (RF) and amplification of the signals of data frames, before transmission.

When signals of the variable rate data frames are received by a receiver, the receiver must determine the rate of transmission in order to properly decode the signals. However, the rate of the received frame is not known by the mobile station a priori. Therefore, some other method of ascertaining the rate is necessary.

The present invention accomplishes rate determination through the use of hypothesis testing. Hypothesis tests are designed based on the probability distribution of the data rates of the frames of speech. Although the data rate of each received frame is not known a priori, the probability of receiving a frame at a given rate can be determined. As mentioned above, a variable rate vocoder encodes each frame of speech at one of a set of predetermined rates based on the speech activity during the time frame.

Since the characteristics of speech activity can be modeled, probabilistic functions of the data rates which depend on speech activity can be derived from the model. Hypothesis tests can then be designed based on the probabilistic functions of data rates to determine the most likely data rate for each received frame of data.

The use of hypothesis testing for rate determination in a variable rate receiving system may be better appreciated by referring to FIG. 2. In a CDMA environment, for example, the receiving system 50 of FIG. 2 may be implemented in either a mobile unit or a cell site in order to determine the data rate of received signals. The present invention offers particular advantages because it avoids the exhaustive decoding at all rates. By choosing a hypothesis and checking the hypothesis for correctness, the average amount of processing for each received frame is reduced. This is especially important in the mobile unit because reduced processing, and thereby power consumption, in the decoding process can extend battery life in the receiver.

The variable rate receiving system 50 illustrated in FIG. 2 includes receiver 30 for collecting transmitted signals, including the data signal of interest. Receiver 30 amplifies and frequency downconverts the received signals from the RF frequency band to the intermediate frequency (IF) band.

The IF signals are presented to demodulator 32. The design and implementation of demodulator 32 are described in detail in U.S. Pat. No. 5,490,165, entitled "DEMODULATION ELEMENT ASSIGNMENT IN A SYSTEM CAPABLE OF RECEIVING MULTIPLE SIGNALS," issued Feb. 6, 1996, and assigned to the assignee of the present invention, the disclosure of which is incorporated by reference herein. Demodulator 32 demodulates the IF signal to produce a data signal consisting of the symbols of one frame of data. Demodulator 32 generates the data signal by despreading and correlating the IF signal addressed to the receiver. The demodulated data signal is then fed to buffer 33. Buffer 33 stores the demodulated data signal, or the received symbols, until it is properly decoded. Buffer 33 may also be the deinterleaver if the data frame had been interleaved for transmission. Buffer 33 provides the demodulated symbol data to decoder 34.

Hypothesis testing module 36 implements the hypothesis test for determining the data rate of a received frame of data. Hypothesis testing module 36 comprises processor 40, which includes memory 38. The information needed in hypothesis testing such as the decoded rates from the previous frames and the probabilities are stored in memory 38. For each data frame received, processor 40 determines the most probable rate based

on the information stored in memory 38. Processor 40 then presents the most probable data rate to decoder 34 which decodes the data signal at this most probable rate to produce decoded bits.

In the exemplary embodiment, decoder 34 is a trellis decoder capable of decoding data of varying rates, such as a Viterbi decoder. The design and implementation of a multirate Viterbi decoder which exhaustively decodes a received signal at all rates of a set of rates is described in the aforementioned U.S. Patent Applications 08/233,570 and 08/126,477. It will be understood by one skilled in the art that the multirate Viterbi decoder may be modified to decode at a selected rate. This may be accomplished by having the Viterbi decoder receive a rate indicator input, in response to which the decoder decodes the data signal according to the rate indicator. Thus, the modified Viterbi decoder may decode a received data frame based on a rate indicator supplied by processor 40 of hypothesis testing module 36.

Decoder 34 generates information data bits and error metrics characterizing the information bits. The error metrics include the previously described CRC bits, which were added into the data frames as overhead bits. Decoder 34 may also generate other error metrics, such as the Yamamoto Quality Metric and the Symbol Error Rate (SER). The Yamamoto metric is determined by comparing the differences in the metrics of remerging paths in each step of the Viterbi decoding with a threshold and labeling a path as unreliable if the metric difference is less than a quality threshold. If the final path selected by the Viterbi decoder has been labeled as unreliable at any step, the decoder output is labeled as unreliable. Otherwise, it is labeled as reliable. The Symbol Error Rate is determined by taking the decoded bits, re-encoding these bits to provided re-encoded symbols, and comparing these re-encoded symbols against the received symbols which are stored in buffer 33. The SER is a measure of the mismatching between the re-encoded symbols and the received symbols. The decoded information bits and the error metrics are provided to data check element 42, which determines if the information bits have been correctly decoded.

In a preferred embodiment, data check element 42 first checks the CRC bits. If the CRC check fails, then data check element 42 provides a signal indicative of the failure to processor 40. If the CRC check passes, then data check element 42 determines if the re-encoded SER is below a certain threshold. If the SER is above the threshold, then a signal indicative of failure is provided to processor 40. Otherwise, the data rate provided by hypothesis testing module 36 is determined to be correct, and a success

signal is provided to processor 40, whereupon no further decoding is performed on the data frame. The properly decoded data signal is presented to variable rate vocoder 44.

When processor 40 receives a failure signal indicating that data symbols have not been properly decoded into information bits, processor 40 will determine at least one other data rate from the set of data rates at which to decode the data symbols. Processor 40 provides the rate information to decoder 34, which decodes the data symbols at the rate provided. For each data rate at which the data signal is decoded, data check element 42 will determine the quality of the decoded information bits. Upon determination by data check element 42 that the correct data rate has been found, a signal of decoded information bits is provided to variable rate vocoder 44. Vocoder 44 will then process the information bits for interface with the user.

Hypothesis testing module 36 may implement any of a number of hypothesis tests for determining the data rate of a received frame of data. For example the hypothesis test may be based on known statistics of speech activity. It is known that for a set of four rates using 20 ms frames, a full rate frame will usually be followed by another full rate frame, while an eighth-rate frame will usually be followed by another eighth rate frame. Further, it is also known that most frames will either be full or eighth rate rather than half or quarter rate, because the periods of speech and silence do not occur in 20 ms bursts. Based on these characteristics, the hypothesis test may designate the rate of the previous frame of data as the most probable rate for the currently received frame of data.

In an exemplary implementation, the rate of the previous frame of data is stored in memory 38 of hypothesis testing module 36. When a data frame is received, processor 40 of hypothesis testing module 36 obtains the rate of the previous frame from memory 38 and presents it to decoder 34. Decoder 34 decodes the received data frame at the rate of the previous frame to produce information bits. Decoder 34 also generates error metrics which are then presented to data check element 42 along with the information bits. If data check element 42 determines from the error metrics that the decoded bits are of good quality, then the information bits are presented to vocoder 44. Otherwise, a failure indication is sent from data check element 42 to processor 40. Processor 40 may then have decoder 34 exhaustively decode the data frame at all other rates before determining the data rate. A flow chart illustrating some of the steps involved in rate determination as described in the embodiment above is shown in FIG. 3.

Alternatively, processor 40 may have decoder 34 sequentially decode the data frame according to a ranking from the next most likely rate to the least likely rate. The ranking may be determined in a number of ways, such as according to the probability distributions described below. For each decoding, error metrics are generated by decoder 34 and checked by data check element 42 for correctness. When correctly decoded, the decoded frame is passed on to vocoder 44. A flow chart illustrating some of the processing steps of this embodiment is shown in FIG. 4.

Another implementation of hypothesis testing module 36 is based upon the a priori probability distribution of data rates. For a set of four rates, the a priori probability distribution (P) of the data rates may be defined as:

$$P = \text{Prob}\{R_t\}, \tag{3}$$

where R_t refers to the full, half, quarter, or eighth rate at time t. The likelihood of receiving a frame at each of the different data rates of a set of rates are maintained in memory 38 of processor 40. Generally, the probability distribution of the data rates are determined based on the theoretical statistics or the empirical statistics of speech activity. The likelihood of receiving a frame at the different rates are then permanently stored in memory 38 for determining the rate of every received frame of data. In a more sophisticated embodiment, the likelihood of the rates stored in memory 38 may be updated based on the actual statistics of the received frames of data.

For each new frame of data received, processor 40 obtains the most probable rate from memory 38 and presents the most probable rate to decoder 34. Decoder 34 decodes the data signal at this most probable data rate and presents the decoded data to data check element 42. Error metrics, including the CRC, are also generated by decoder 34 and presented to data check element 42. Other error metrics may also be generated for checking by data check element 42. If the error metrics indicate that the decoded bits are of good quality, then the information bits are presented to vocoder 44. Otherwise, a failure indication is sent from data check element 42 to processor 40. Then, processor 40 obtains the second most likely data rate from memory 38 and presents it to decoder 34, and the process of decoding and error checking is continued until the correct data rate is found. A flow chart of the processing steps of this embodiment is illustrated in FIG. 5. Alternatively, upon receipt of a failure signal by processor 40, processor 40 may cause decoder 34 to exhaustively decode the data frame at each of the

other data rates of the set of rates, and error metrics are checked for each decoding in order to determine the actual rate of transmission. A flow chart of the processing steps of this embodiment is illustrated in FIG. 6.

Instead of designing the hypothesis test based on the simple probability distribution of the data rates, conditional probabilities may be used to improve on the accuracy of the rate determination. For example, the probability of receiving a data frame at a given rate may be defined to be conditioned on the actual rates of the previous frames of data. Conditional probabilities based on the previous rates work well because transition characteristics of the data signals are well known. For example, if the rate two frames ago was eighth rate and the rate for the previous frame was half rate, then the most likely rate for the current frame is full rate, because the transition to half rate indicates the onset of active speech. Conversely, if the rate two frames ago was full rate and the rate for the previous frame was quarter rate, then the most likely rate for the present frame might be eighth rate, because the rate transition indicates the onset of silence.

The probability distribution of the data rates conditioned on the rates of the previous n frames of data may be defined as:

$$P = \text{Prob}\{ R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}\} \quad (4)$$

where R_t again refers to the rate at time t, and R_{t-1}, R_{t-2}, . . ., R_{t-n} refers to rate(s) of the previous n frame(s) of data, for $n \geq 1$. The likelihood of receiving a frame at each of the different data rates of a set of rates conditioned on the previous n actual rates are stored in memory **38** of processor **40**. In addition, the actual data rates of the previous n frames of data are maintained by processor **40**, and may be stored in memory **38** as the rates are determined.

For each received frame of data, processor **40** will determine the most probable data rate conditioned on the previous n actual data rates and present it to decoder **34**. Decoder **34** will decode the frame at this most probable data rate and present the decoded bits to data check element **42**. In addition, error metrics are generated by decoder **34** and presented to data check element **42**. If the error metrics indicate that the decoded bits are of good quality, then the information bits are presented to vocoder **44**. Also, processor **40** is informed of the rate decision so that it can maintain the history of chosen rates. That is, processor **40** is supplied R_t so that it can be used in determining $\text{Prob}\{ R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}\}$ for the next frame. If error metrics indicate an unsuccessful decoding, then a failure indication signal is

sent from data check element 42 to processor 40, and processor 40 determines the second most probable data rate conditioned on the previous n actual data rates to decode the data frame. As in the simple probabilities case, the process of decoding and error checking is continued until the correct data rate is found. Some of the processing steps of this embodiment are illustrated in a flow chart in FIG. 7. Also as in the simple probabilities case, after a failed decoding at the most likely rate, decoder 34 may exhaustively decode the data frame at all of the other data rates and have error metrics checked for all decoding in order to determine the data rate. Some of the processing steps of this embodiment are illustrated in a flow chart in FIG. 8.

It should be understood that the conditional probability distribution of the data rates may depend on statistics other than the actual rates of the previous frames of data. For example, the probability distribution may be conditioned on one or more frame quality measurements. The probability distribution is then defined to be:

$$P = \text{Prob}\{R_t \mid X_1, X_2, \ldots, X_k\}, \quad (5)$$

where R_t is the rate at time t, and $X_1, X_2, \ldots X_k$ are one or more frame quality measurements. The k frame quality measurements may be measurements performed on the current frame of data, or measurements performed on previous frame(s) of data, or a combination of both. An example of a frame quality measurement is the SER error metric mentioned above. Thus, the probability of receiving a frame at a given rate is conditioned on the SER obtained from the previous decoding if a previous decoding had been performed.

The conditional probability distribution may also depend on a combination of the actual rates of the previous frames of data and the frame quality measurements. In this case, the probability distribution of the data rates is defined as:

$$Pt = \text{Prob}\{ R_t \mid R_{t-1}, R_{t-2}, \ldots, R_{t-n}, X_1, X_2, \ldots, X_k \}, \quad (6)$$

where R_t is the rate at time t, $R_{t-1}, R_{t-2}, \ldots, R_{t-n}$ are the rates of the previous frames of data and $X_1, X_2, \ldots, X_k$ are the frame quality measurements.

In the cases where the probability distribution is based on frame quality measurements, the frame quality measurements should be maintained in processor 40 of hypothesis testing module 36. As can be seen from the above description, the hypothesized frame rate may be conditioned

on a number of different statistics, and the rates of the previous frames and the frame quality measurements are examples of these statistics. For each data frame received, processor **40** uses the statistics to determine the rate at which to decode the frame.

A further refinement to the determination of the rate at which to decode a received frame of data considers the processing costs of decoding the frame at the various rates in conjunction with hypothesis testing. In this embodiment, an optimum test sequence of the rates is established based on both the probability distribution of the data rates and the cost of decoding at each of the data rates. The optimum test sequence is maintained by processor **40**, which causes decoder **34** to sequentially decode a received frame of data according to the optimum sequence until the correct rate is found. The optimum test sequence is established to minimize the total expected cost of the rate search. Denoting P_i to be the probability that the rate search will stop at test T_i, and C_i to be the cost for conducting test T_i, the total expected cost of the rate search using test sequence $T_1, T_2, \ldots, T_M$, where M is the number of possible rates in the system and $1 \leq i \leq M$, can be modeled as:

$$C_{total} = C_1 * P_1 + (C_1 + C_2)* P_2 + \ldots (C_1 + C_2 + \ldots + C_M)* P_M. \quad (7)$$

The optimum test sequence is found by minimizing the total expected cost C_{total}.

In Equation (7), the cost C_i for conducting test T_i will generally be the processing power required for decoding a frame at the rate specified by test T_i. The cost may be assigned to be proportional to the frame rate specified by the test T_i because the computational complexity of decoder **34** is in general approximately proportional to the number of bits per frame. The probabilities P_i may be assigned by the unconditioned a priori probability distribution of data rates as defined by Equation (3), or any of the conditional probability distributions defined by Equations (4), (5), or (6) above.

In a variable rate communications system where data frames are transmitted at 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps, the following example illustrates the formulation of the optimum test sequence for rate determination of a received frame. The costs to decode the 9,600 bps, 4,800 bps, 2,400 bps, and 1,200 bps frames are assumed to be 9.6, 4.8, 2.4, and 1.2, respectively. Further, the probability of receiving a frame at each of the four

rates is assumed to be the unconditioned a priori probabilities having the following values:

$$\text{Prob}(9{,}600 \text{ bps}) = 0.291, \quad (8)$$

$$\text{Prob}(4{,}800 \text{ bps}) = 0.039, \quad (9)$$

$$\text{Prob}(2{,}400 \text{ bps}) = 0.072, \text{ and} \quad (10)$$

$$\text{Prob}(1{,}200 \text{ bps}) = 0.598. \quad (11)$$

The probabilities given in Equations (8)-(11) are derived from steady state empirical data.

A listing of all possible test sequences for rate determination in the system transmitting frames at 9,600, 4,800, 2,400, and 1,200 bps is shown in Table I below. In Table I, column 1 lists all possible test sequences T_1, T_2, T_3, T_4, where $T_i = 1$ refers to the test of decoding at 9,600 bps, $T_i = 1/2$ refers to the test of decoding at 4,800 bps, $T_i = 1/4$ refers to the test of decoding at 2,400 bps, and $T_i = 1/8$ refers to the test of decoding at 1,200 bps. Columns 2 and 3 list the probability P_1 and the cost C_1 of performing the test T_1, columns 4 and 5 list the probability P_2 and the cost C_2 of performing the test T_2, columns 6 and 7 list the probability P_3 and the cost C_3 of performing the test T_3, and columns 8 and 9 list the probability P_4 and the cost C_4 of performing the test T_4. The total cost C_{total} of performing the test sequence T_1, T_2, T_3, T_4 is listed in column 10.

T_1, T_2, T_3, T_4	P_1	C_1	P_2	C_2	P_3	C_3	P_4	C_4	C_{total}
1, 1/2, 1/4, 1/8	0.291	9.6	0.039	4.8	0.072	2.4	0.598	1.2	15.33
1, 1/2, 1/8, 1/4	0.291	9.6	0.039	4.8	0.598	1.2	0.072	2.4	13.98
1, 1/4, 1/2, 1/8	0.291	9.6	0.072	2.4	0.039	4.8	0.598	1.2	15.08
1, 1/4, 1/8, 1/2	0.291	9.6	0.072	2.4	0.598	1.2	0.039	4.8	12.25
1, 1/8, 1/2, 1/4	0.291	9.6	0.598	1.2	0.039	4.8	0.072	2.4	11.16
1, 1/8, 1/4, 1/2	0.291	9.6	0.598	1.2	0.072	2.4	0.039	4.8	10.90
1/2, 1, 1/4, 1/8	0.039	4.8	0.291	9.6	0.072	2.4	0.598	1.2	16.35
1/2, 1, 1/8, 1/4	0.039	4.8	0.291	9.6	0.598	1.2	0.072	2.4	15.00
1/2, 1/4, 1, 1/8	0.039	4.8	0.072	2.4	0.291	9.6	0.598	1.2	16.36
1/2, 1/4, 1/8, 1	0.039	4.8	0.072	2.4	0.598	1.2	0.291	9.6	10.97
1/2, 1/8, 1, 1/4	0.039	4.8	0.598	1.2	0.291	9.6	0.072	2.4	9.61
1/2, 1/8, 1/4, 1	0.039	4.8	0.598	1.2	0.072	2.4	0.291	9.6	9.62
1/4, 1, 1/2, 1/8	0.072	2.4	0.291	9.6	0.039	4.8	0.598	1.2	15.08
1/4, 1, 1/8, 1/2	0.072	2.4	0.291	9.6	0.598	1.2	0.039	4.8	12.26
1/4, 1/2, 1, 1/8	0.072	2.4	0.039	4.8	0.291	9.6	0.598	1.2	16.11
1/4, 1/2, 1/8, 1	0.072	2.4	0.039	4.8	0.598	1.2	0.291	9.6	10.71
1/4, 1/8, 1/2, 1	0.072	2.4	0.598	1.2	0.039	4.8	0.291	9.6	7.89
1/4, 1/8, 1, 1/2	0.072	2.4	0.598	1.2	0.291	9.6	0.039	4.8	6.87
1/8, 1, 1/4, 1/2	0.598	1.2	0.291	9.6	0.072	2.4	0.039	4.8	5.51
1/8, 1, 1/2, 1/4	0.598	1.2	0.291	9.6	0.039	4.8	0.072	2.4	5.76
1/8, 1/2, 1, 1/4	0.598	1.2	0.039	4.8	0.291	9.6	0.072	2.4	6.79
1/8, 1/2, 1/4, 1	0.598	1.2	0.039	4.8	0.072	2.4	0.291	9.6	6.79
1/8, 1/4, 1/2, 1	0.598	1.2	0.072	2.4	0.039	4.8	0.291	9.6	6.54
1/8, 1/4, 1, 1/2	0.598	1.2	0.072	2.4	0.291	9.6	0.039	4.8	5.52

Table I

As shown in Table I, the optimum test sequence is the sequence 1/8, 1, 1/4, 1/2 shown in the 19th row. This test sequence offers the lowest total expected cost of processing. Therefore, the rate determination system would decode a received frame of data at 1,200 bps first. If the decoding at 1.200 bps is not successful, then the frame would be decoded sequentially at 9,600 bps, 2,400 bps, and 4,800 bps until the correct rate is found. In a preferred embodiment, the optimum test sequence is maintained by processor **40** of hypothesis testing module **36**. For each frame of data received, processor **40** causes decoder **34** to decode the frame sequentially according to the optimum test sequence, with each decoding checked by data check element **42**, until the correct data rate is found. Processing resources are efficiently utilized in this rate determination system because the decoding is performed sequentially according to an optimum search sequence.

Based on the embodiments described above, it will be understood by one skilled in the art that the present invention is applicable to all systems

in which data has been encoded according to a variable rate scheme and the data must be decoded in order to determine the rate. Even more generally, the invention is applicable to all systems in which the encoded data E is a function of the data D and some key k, and there exists some information in D or E which permits the verification of the correct D by the receiver. The sequence k may be time varying. The encoded data is represented as:

$$E = f(D,k), \tag{1}$$

where k is from a small set K of keys and where some probability function exists on the set of keys. The inverse of the encoding, or the decoding, can be represented as:

$$D = f^{-1}(E,k), \tag{2}$$

where k is chosen so that D is correct.

As an example, assume that D is data composed of fixed-length sequence D1 and fixed length sequence D2 so that D = D1, D2. Sequence D2 is the Cyclic Redundancy Code (CRC) of D1, so that D2 = fcrc(D1). Assume also that the encoding function, f(D,k), is an exclusive-OR of a fixed-length D with the fixed length sequence k. Then, the decoding, $f^{-1}(E,k)$, would be the exclusive-OR of E with the correct k. The correct k is verified by checking whether D2 = fcrc(D1). The correct k can be found by decoding all possible k's in K and then determining whether the CRC check passes. Alternatively, it can be done by sequentially decoding using one k at a time, with no further decoding once the "correct" k is found. According to the present invention, the order of sequential decoding is to be determined by hypothesis testing. A number of hypothesis tests, including the tests described above, may be utilized. The order of sequential decoding may in addition depend on the cost of processing, as described above. The use of hypothesis testing and/or cost functions in formulating a test sequence for rate determination reduces the average amount of processing as fewer k's will have to be tried.

The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the

embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

CLAIMS

1. In a variable rate communications system, a sub-system for determining, at a receiver, the data rate of a received data frame, comprising:

a processor for generating a signal indicating the most likely rate of said received data frame in accordance with a predetermined hypothesis test; and

a decoder for receiving said most likely rate signal and for decoding said received data frame into a decoded frame of bits at said most likely rate.

2. The rate determination sub-system of claim 1 wherein said most likely rate is the rate of the previous data frame.

3. The rate determination sub-system of claim 1 wherein said hypothesis test is based on an a priori probability distribution of data rates.

4. The rate determination sub-system of claim 1 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

5. The rate determination sub-system of claim 1 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

6. The rate determination sub-system of claim 1 further comprising a data check element for receiving said decoded bits, generating error metrics characterizing said decoded bits, and generating a quality indication based on said error metrics for said decoded bits.

7. The rate determination sub-system of claim 6,

further comprising a vocoder for receiving said decoded bits and processing said decoded bits to provide speech to an user upon generation of a positive indication of said quality; and

wherein upon generation of a negative indication of said quality, said processor further causes said decoder to perform additional decoding of said received data frame in accordance with at least one rate other than said most likely rate.

8. The rate determination sub-system of claim 7,

wherein said additional decoding is performed sequentially in accordance with a predetermined test sequence of data rates;

wherein said data check element generates error metrics for each said additional decoding and generates a quality indication based on said error metrics for each said additional decoding; and

wherein said additional decoding terminates upon generation of a positive indication of said quality.

9. The rate determination sub-system of claim 7,

wherein said additional decoding comprises exhaustive decoding of said received data frame at all rates of a rate set except said most likely rate; and

wherein said data check element generates error metrics for each said additional decoding and determines the rate of said received data frame in accordance with said error metrics.

10. The rate determination sub-system of claim 6 wherein said error metrics include a Cyclic Redundancy Check result.

11. The rate determination sub-system of claim 6 wherein said error metrics include a Symbol Error Rate metric.

12. The rate determination sub-system of claim 6 wherein said error metrics include a Yamamoto quality metric.

13. The rate determination sub-system of claim 1 wherein said processor comprises a memory for storing said most likely rate.

14. The rate determination sub-system of claim 1 wherein said decoder is a Viterbi decoder.

15. In a variable rate communications system, a sub-system for determining, at a receiver, the data rate of a received data frame, comprising:

a processor for generating a test sequence of data rates for determining the rate of a received data frame, said test sequence being generated in accordance with a predetermined hypothesis test;

a decoder for decoding said received data frame sequentially according to said test sequence and generating a decoded frame of bits for each rate at which said received data frame is decoded;

a data check element for generating error metrics characterizing said decoded bits and for generating a quality indication based on said error metrics for each rate at which said received data frame is decoded; and

wherein no further decoding is performed upon generation of a positive indication of said quality.

16. The rate determination sub-system of claim 15 wherein said hypothesis test is based on an a priori probability distribution of data rates.

17. The rate determination sub-system of claim 15 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

18. The rate determination sub-system of claim 15 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

19. The rate determination sub-system of claim 16 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

20. The rate determination sub-system of claim 17 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

21. The rate determination sub-system of claim 18 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

22. The rate determination sub-system of claim 15 further comprising a vocoder for receiving said decoded bits and processing said decoded bits to provide speech to an user upon generation of a positive indication of said quality.

23. The rate determination sub-system of claim 15 wherein said error metrics include a Cyclic Redundancy Check result.

24. The rate determination sub-system of claim 15 wherein said error metrics include a Symbol Error Rate metric.

25. The rate determination sub-system of claim 15 wherein said error metrics include a Yamamoto quality metric.

26. The rate determination sub-system of claim 15 wherein said processor comprises a memory for storing said test sequence of data rates.

27. The rate determination sub-system of claim 15 wherein said decoder is a Viterbi decoder.

28. A method for determining the rate of a received data frame in a variable rate communications system, comprising the steps of:

receiving a wide-band signal;

demodulating said wide-band signal to produce a data signal, wherein said data signal has been transmitted at one of a set of possible transmission rates;

generating a test sequence of data rates for determining the rate of said data signal, said test sequence being generated in accordance with a predetermined hypothesis test;

decoding said data signal sequentially according to said test sequence to generate a decoded frame of bits for each rate at which said data signal is decoded;

generating error metrics characterizing said decoded frame of bits for each rate at which said data signal is decoded;

generating a quality indication based on said error metrics for each rate at which said data signal is decoded; and

upon generation of a positive indication of said quality, providing said decoded frame of bits to a vocoder which processes said decoded bits to provide speech to an user.

29. The method of claim 28 wherein said hypothesis test is based on an a priori probability distribution of data rates.

30. The method of claim 28 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on the rate of at least one previous data frame.

31. The method of claim 28 wherein said hypothesis test is based on a conditional probability distribution of data rates conditioned on at least one frame quality measurement.

32. The method of claim 29 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

33. The method of claim 30 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

34. The method of claim 31 wherein said test sequence is generated further in accordance with the cost of decoding said received data frame at each of said data rates.

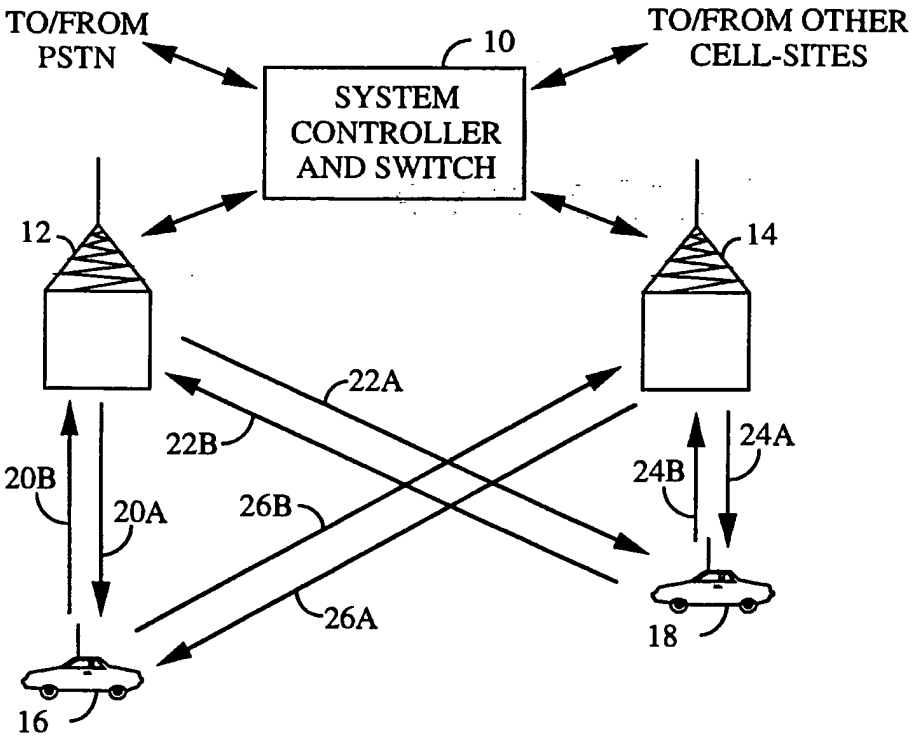

TO/FROM
PSTN
10
TO/FROM OTHER
CELL-SITES
SYSTEM
CONTROLLER
AND SWITCH
12
14
22A
22B
20B
20A
26B
24A
24B
26A
18
16

FIG. 1

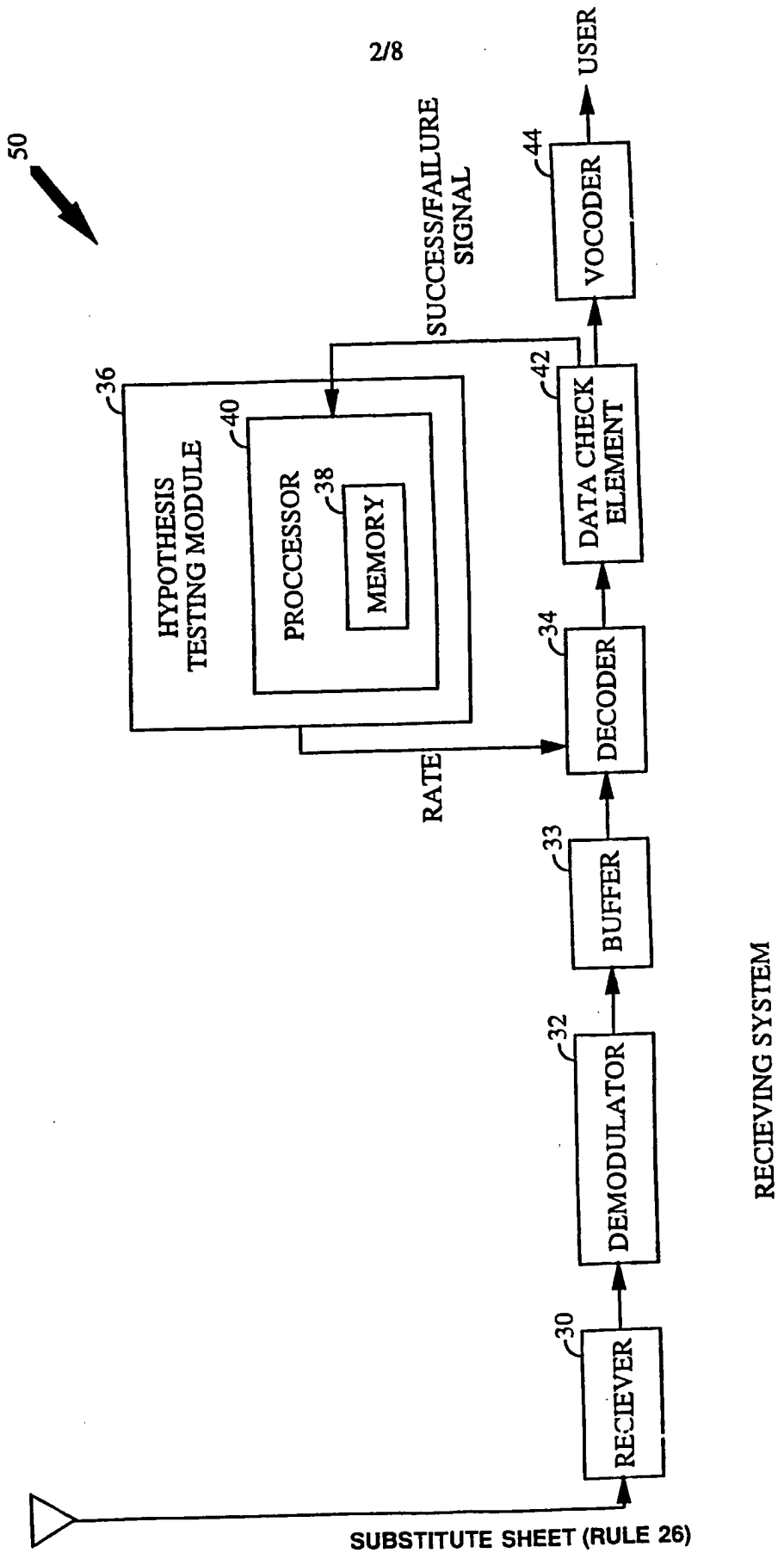


FIG. 2

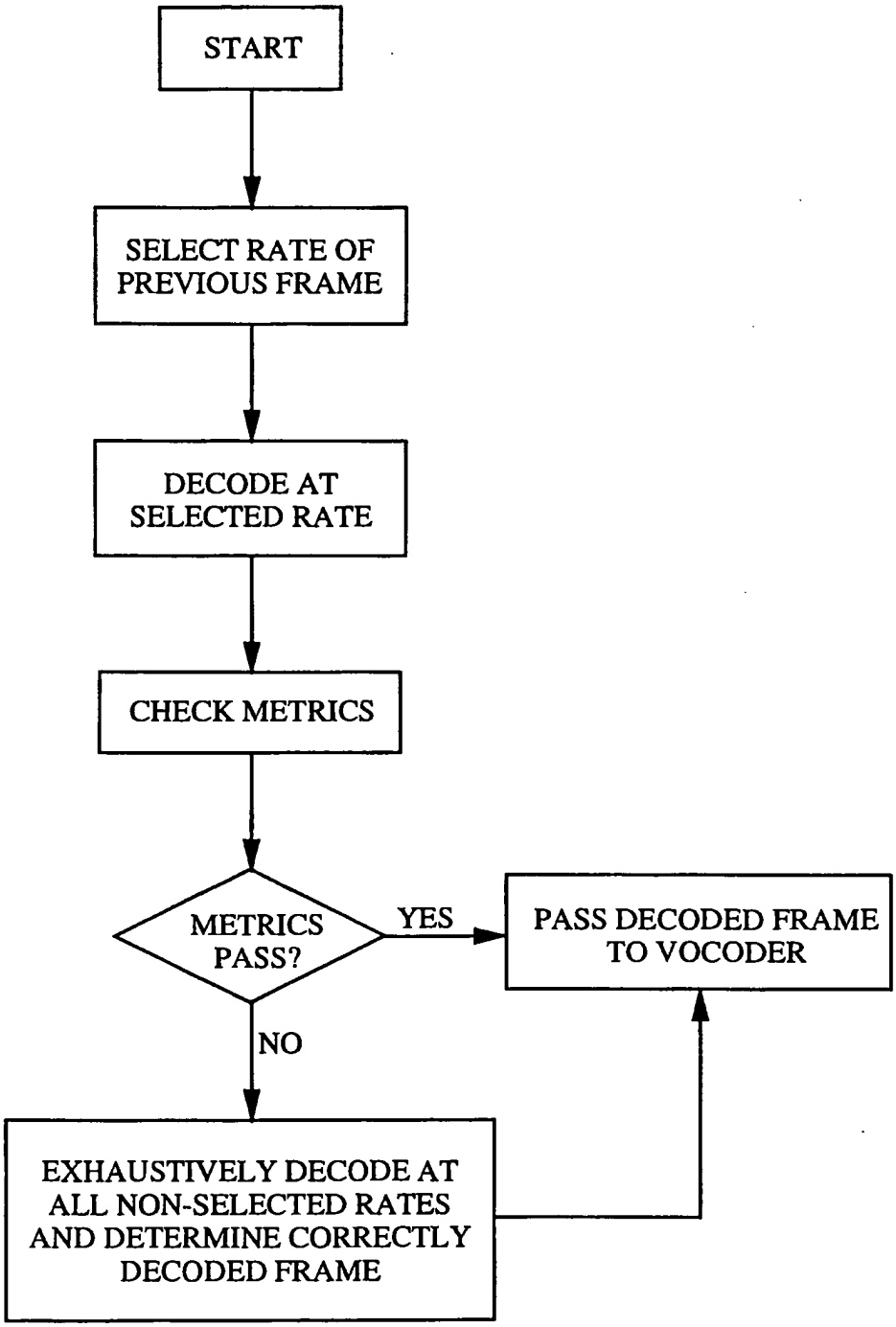

START
SELECT RATE OF PREVIOUS FRAME
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 3

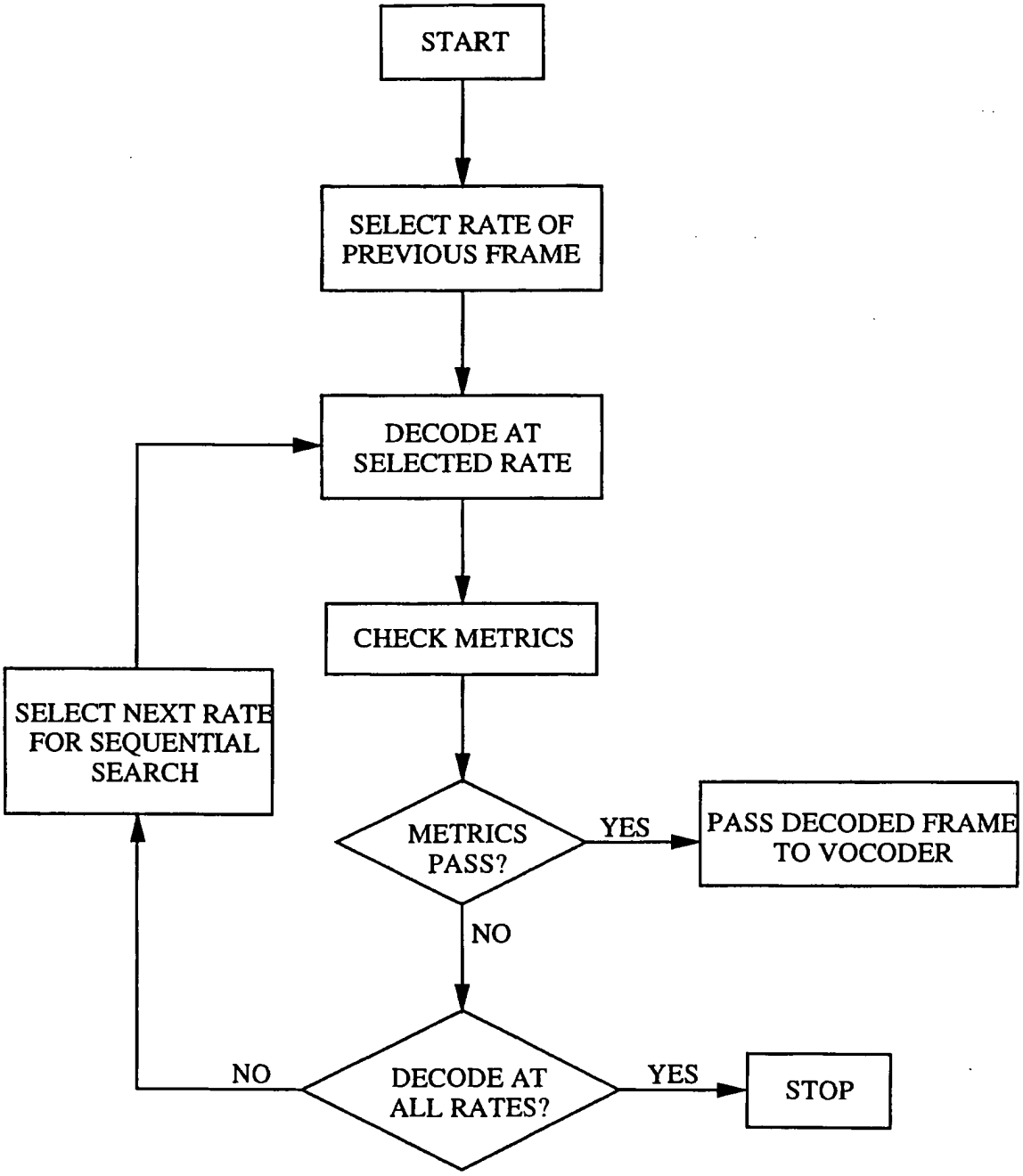

START
SELECT RATE OF PREVIOUS FRAME
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
DECODE AT ALL RATES?
NO
SELECT NEXT RATE FOR SEQUENTIAL SEARCH
YES
STOP

FIG. 4

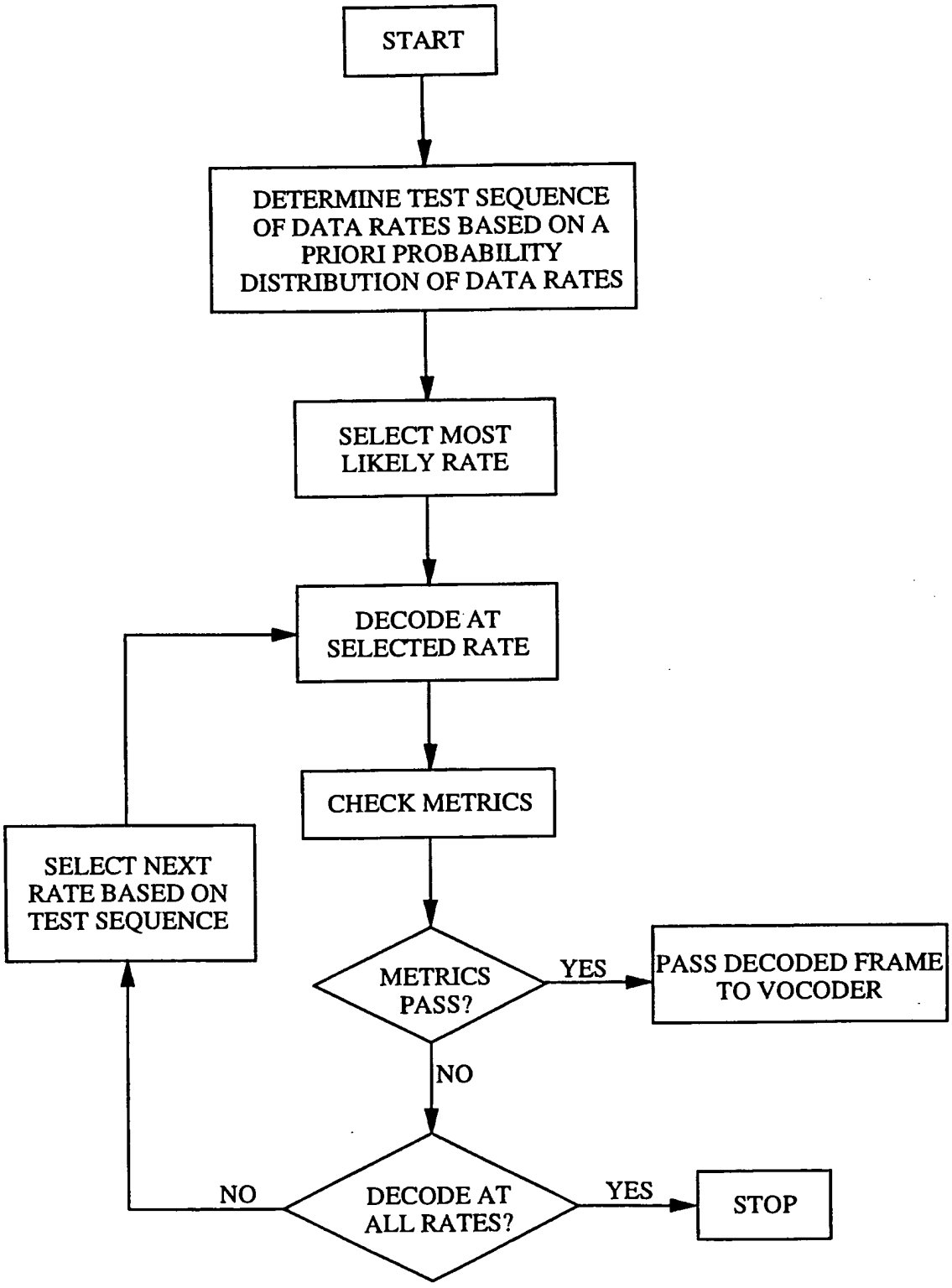

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON A PRIORI PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
SELECT NEXT RATE BASED ON TEST SEQUENCE
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
DECODE AT ALL RATES?
NO
YES
STOP

FIG. 5

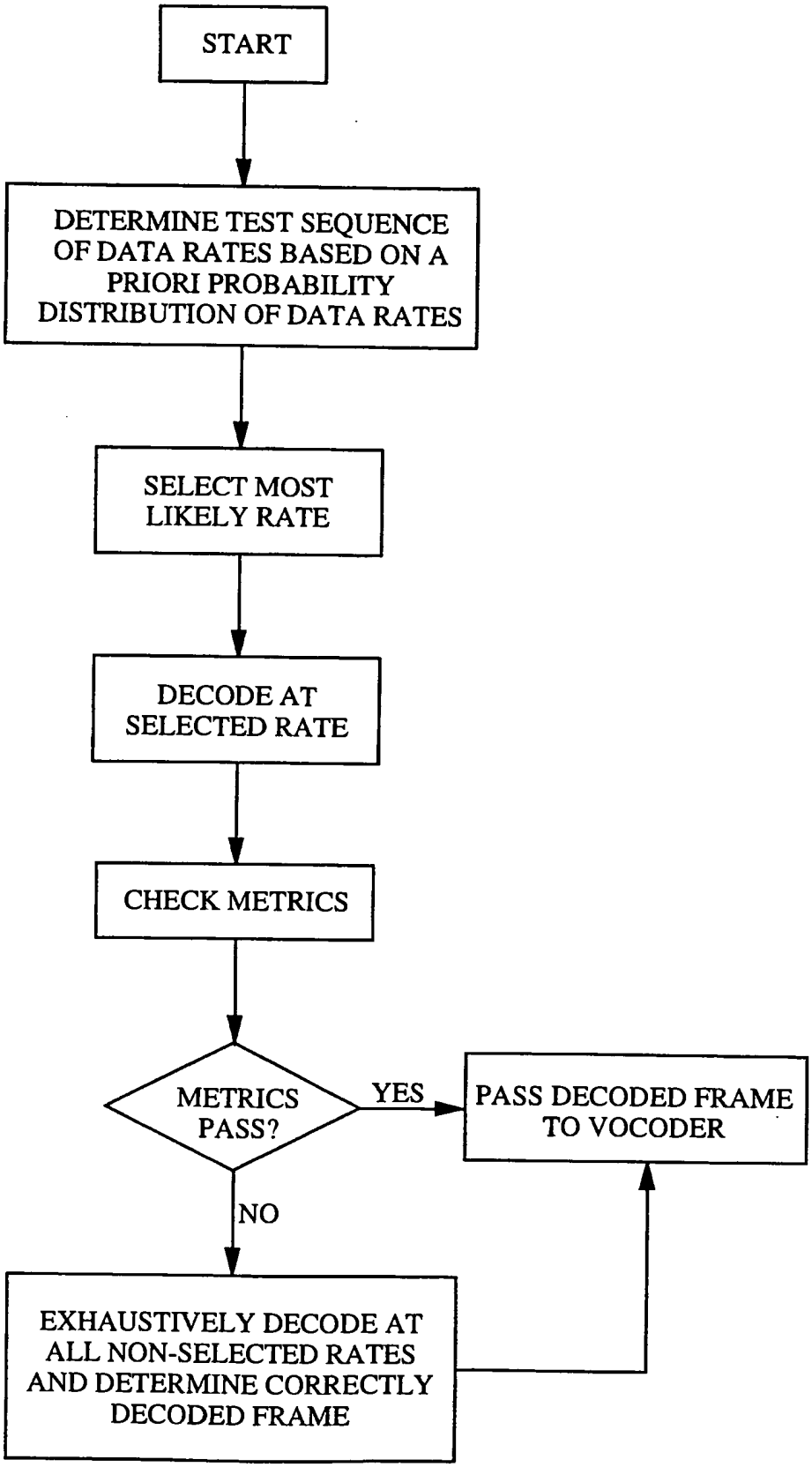

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON A PRIORI PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 6

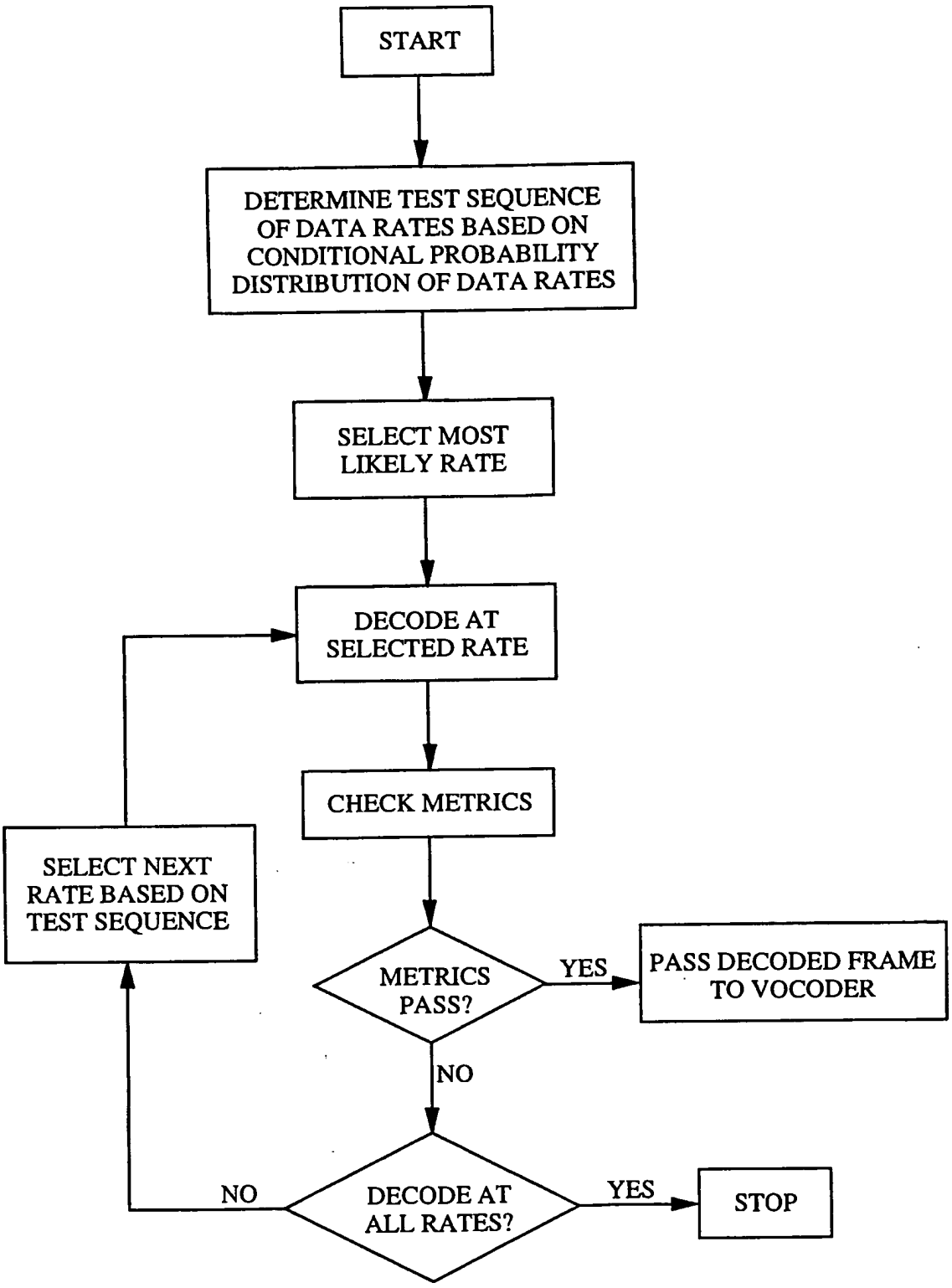

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON CONDITIONAL PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
DECODE AT ALL RATES?
NO
SELECT NEXT RATE BASED ON TEST SEQUENCE
YES
STOP

FIG. 7

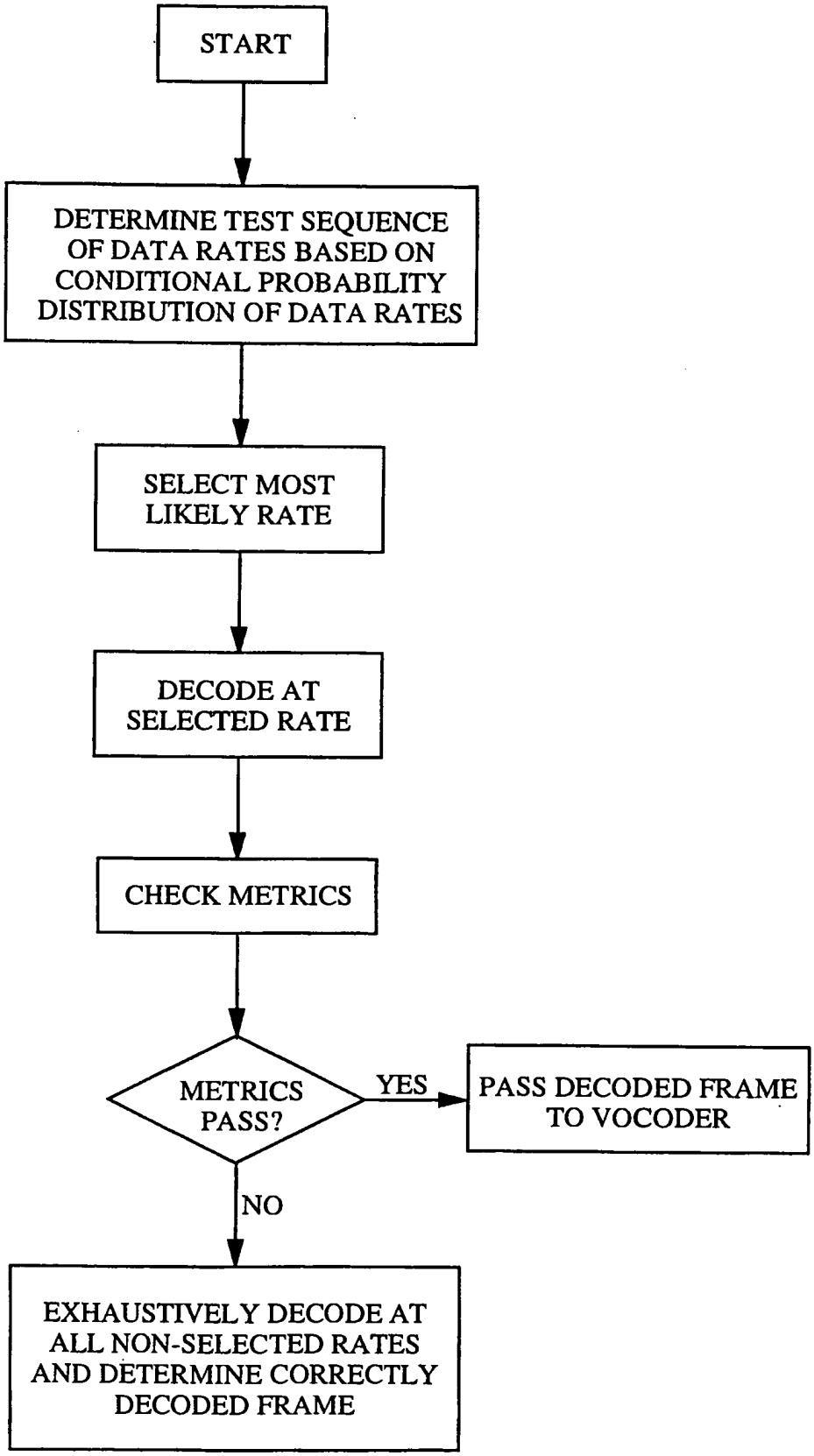

START
DETERMINE TEST SEQUENCE OF DATA RATES BASED ON CONDITIONAL PROBABILITY DISTRIBUTION OF DATA RATES
SELECT MOST LIKELY RATE
DECODE AT SELECTED RATE
CHECK METRICS
METRICS PASS?
YES
PASS DECODED FRAME TO VOCODER
NO
EXHAUSTIVELY DECODE AT ALL NON-SELECTED RATES AND DETERMINE CORRECTLY DECODED FRAME

FIG. 8

INTERNATIONAL SEARCH REPORT

Inte. ional Application No
PCT/US 97/19676

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L25/02

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)
IPC 6 H04L

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 711 056 A (NIPPON ELECTRIC CO) 8 May 1996 see abstract see page 2, column 2, line 46 - page 3, column 3, line 26 see page 4, column 5, line 13 - column 6, line 23; figure 2 ---	1,14,15, 22,27,28
A	EP 0 713 305 A (NIPPON ELECTRIC CO) 22 May 1996 see abstract see page 2, column 2, line 45 - page 3, column 3, line 12 see page 3, column 3, line 33 - column 4, line 2; figure 1 see page 3, column 4, line 44 - page 4, column 5, line 13 --- -/--	1,14,15, 22,27,28

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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 publicationdate 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

1

Date of the actual completion of theinternational search	Date of mailing of the international search report
27 February 1998	09/03/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bossen, M

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/US 97/19676

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 566 206 A (BUTLER BRIAN K ET AL) 15 October 1996 cited in the application see column 2, line 37 - line 47 see column 5, line 57 - column 6, line 32; figure 2 see column 7, line 35 - line 52; figure 4	7,9-12, 14, 22-25,27

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int. Jonal Application No
PCT/US 97/19676

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0711056 A	08-05-96	JP 8130535 A	21-05-96
		AU 3662595 A	09-05-96
		CA 2163134 A	03-05-96
EP 0713305 A	22-05-96	JP 2596392 B	02-04-97
		JP 8149567 A	07-06-96
		AU 686026 B	29-01-98
		AU 3787595 A	23-05-96
		CA 2162417 A	17-05-96
		FI 955398 A	17-05-96
US 5566206 A	15-10-96	AT 158910 T	15-10-97
		AU 683479 B	13-11-97
		AU 7113694 A	17-01-95
		BR 9406891 A	26-03-96
		DE 69405997 D	06-11-97
		EP 0705512 A	10-04-96
		FI 956091 A	16-02-96
		JP 9501548 T	10-02-97
		WO 9501032 A	05-01-95
		CN 1108834 A	20-09-95
		IL 109842 A	30-09-97
		MX 9404610 A	31-01-95
		ZA 9404032 A	09-03-95

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/00

A1

(11) International Publication Number: WO 98/11690

(43) International Publication Date: 19 March 1998 (19.03.98)

(21) International Application Number: PCT/US97/16223

(22) International Filing Date: 12 September 1997 (12.09.97)

(30) Priority Data:

60/025,991	12 September 1996 (12.09.96)	US
08/887,723	3 July 1997 (03.07.97)	US

(71)(72) Applicant and Inventor: GLOVER, John, J. [US/US]; 26 Amaranth Avenue, Medford, MA 02155 (US).

(74) Agent: GORDON, Peter, J.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

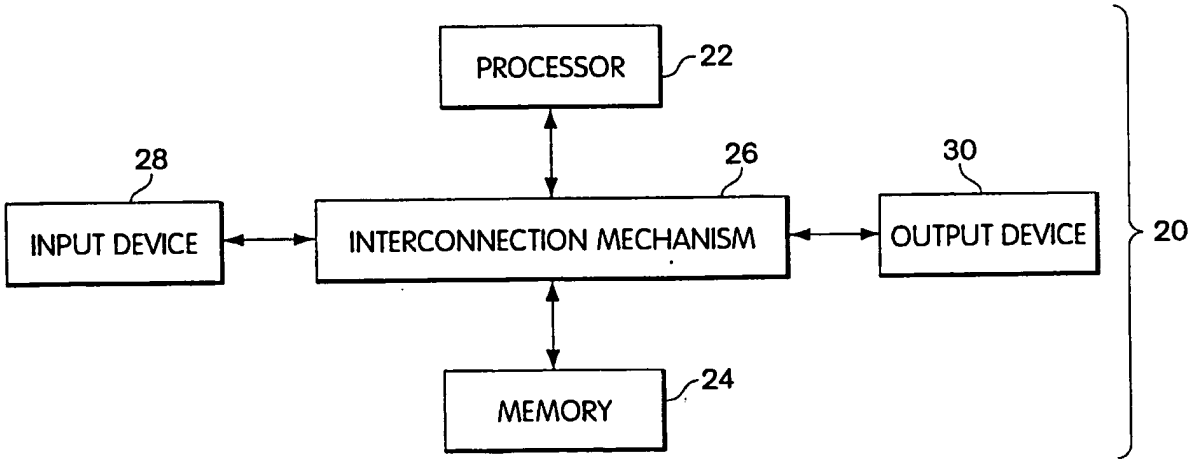
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

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: SELF-DECRYPTING DIGITAL INFORMATION SYSTEM AND METHOD



(57) Abstract

The claimed data protection device (20) includes a processor (22) connected to a memory system (24) through an interconnection mechanism (26). An input device (28) is also connected to the processor (22) and memory system (24) through the interconnection mechanism (26). The interconnection mechanism (26) is typically a combination of one or more buses and one or more switches. The output device (30) may be a display, and the input device (28) may be a keyboard and/or mouse or other cursor control device.

*(Referred to in PCT Gazette No. 32/1998, Section II)

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.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SELF-DECRYPTING DIGITAL INFORMATION SYSTEM AND METHOD

Field of the Invention

The present invention is related to mechanisms for protecting digital information from being copied. In particular, the present invention is related to mechanisms which permit authorized execution of computer program code or access to other digital information which is encrypted or otherwise encoded.

Background of the Invention

A serious problem which faces the electronic publishing and software industries is the ease with which digital information can be copied without authorization from the publisher. Digital information also may be used or modified without authorization. For example, computer software may be reverse engineered or attacked by computer viruses.

There are many mechanisms available which may be used to limit or prevent access to digital information. Such mechanisms often either restrict the ability of the user to make back-up copies or involve the use of special purpose hardware to limit access to the digital information. For example, some mechanisms restrict the use of digital information to a particular machine. See, for example, U.S. Patent 4,817,140. Other mechanisms require the digital information to be stored on a particular recording medium in order to be used. See, for example, U.S. Patent 5,412,718. Yet other mechanisms allow only a certain number of uses of the digital information. See for example, U.S. Patent 4,888,798. Many of these access control mechanisms cause distribution to be more costly.

Several other patents describe a variety of systems for encryption, compression, licensing and royalty control and software distribution such as: U.S. Pat. No. 4,405,829, U.S. Pat. No. 4,864,616, U.S. Pat. No. 4,888,800, U.S. Pat. No. 4,999,806, U.S. Pat. No. 5,021,997, U.S. Patent No. 5,027,396, U.S. Pat. No. 5,033,084, U.S. Pat. No. 5,081,675, U.S. Pat. No. 5,155,847, U.S. Pat. No. 5,166,886, U.S. Pat. No. 5,191,611, U.S. Pat. No. 5,220,606, U.S. Pat. No. 5,222,133, U.S. Pat. No. 5,272,755, U.S. Pat. No. 5,287,407, U.S. Pat. No. 5,313,521, U.S. Pat. No. 5,325,433, U.S. Pat. No. 5,327,563, U.S. Pat. No. 5,337,357, U.S. Pat. No. 5,351,293, U.S. Pat. No. 5,341,429, U.S. Pat. No. 5,351,297, U.S. Pat. No. 5,361,359, U.S. Pat. No. 5,379,433, U.S. Pat. No. 5,392,351, U.S. Pat. No. 5,394,469, U.S. Pat. No. 5,414,850, U.S. Pat. No. 5,473,687, U.S. Pat. No. 5,490,216, U.S. Pat. No. 5,497,423, U.S. Pat. No. 5,509,074, U.S.

Pat. No. 5,,511,123, U.S. Pat. No. 5,524,072, U.S. Pat. No. 5,532,920, U.S. Pat. No. 5,555,304, U.S. Pat. No. 5,557,346, U.S. Pat. No. 5,557,765, U.S. Pat. No. 5,592,549, U.S. Pat. No. 5,615,264, U.S. Pat. No. 5,625,692, and U.S. Pat. No. 5,638,445.

Computer programs or other digital information also may be encrypted in order to prevent an individual from making a useful copy of the information or from reverse engineering a program. Even with such encryption, however, a computer program must be decrypted in order for a computer to load and execute the program. Similarly, other digital information must be decrypted before it can be accessed and used. Generally, digital information is decrypted to disk, and not to main memory of the computer which is more protected by the operating system, because decryption to main memory results in a significant loss of memory resources. If the purpose for using encryption is to prevent users from copying the digital information, then decryption of the information to accessible memory for use defeats this purpose.

One way to protect digital information using encryption has been made available by International Business Machines (IBM) and is called a "CRYPTOLOPE" information container. This technology is believed to be related to U.S. Patent Nos. 5,563,946 and 5,598,470 (to Cooper et al.), and published European patent applications 0679977, 0679978, 0679979 and 0681233. The CRYPTOLOPE system requires a user to have a "helper application" and a key. The CRYPTOLOPE information container is generated by IBM. The content provider submits data to IBM, which in turn encrypts and packages the data in a CRYPTOLOPE information container. The helper application is a form of memory resident program, called a terminate and stay resident (TSR) program, which is a form of input/output (I/O) device driver installed in the operating system and which monitors requests from the operating system for files on specified drives and directories. Because the TSR program must know the directory, and/or file name to be accessed, that information also is available to other programs. Other programs could use that information to manipulate the operation of the TSR program in order to have access to decrypted contents of the information container. The encrypted information container includes an executable stub which is executed whenever the application is run without the installed TSR program or from a drive not monitored by the TSR program to prevent unpredictable activity from executing encrypted code. This stub may be used to install decryption and cause the application be executed a second time, or to communicate with the TSR program to instruct the TSR program to monitor the drive. It may be preferable from the point of view of the content provider however to maintain an encryption process and keys independently of any third party.

Multimedia content, such as a movie or hypertext presentation also may be stored on a digital versatile disk (DVD), sometimes called a digital video disk, compact disk read-only memory (CD-ROM), rewriteable compact disks (CD-RW) or other medium in an encrypted digital format for use with special-purpose devices. For example, concern about illegal copying of content from digital video disks or other digital media has resulted in a limited amount of content being available for such devices. This problem has caused representatives of both multimedia providers and digital video disk manufacturers to negotiate an agreement on an encryption format for information stored on DVDs. This copy protection scheme is licensed through an organization called the CSS Interim Licensing organization. However, in this arrangement, the content provider is limited to using the agreed upon encryption format and a device manufacturer is limited to using a predetermined decryption system.

Encryption has also been used to protect and hide computer viruses. Such viruses are typically polymorphic, i.e., they change every time they infect a new program, and are encrypted. The virus includes a decryption program that executes to decrypt the virus every time the infected program is run. Such viruses are described, for example, in "Computer Virus-Antivirus Coevolution" by Carey Nachenberg, Communications of the ACM, Vol. 40, No. 1, (Jan. 1997), p. 46 et seq. Such viruses include decryption keys within them since, clearly, their execution is not carried out by the user and a user would not be asked for authorization keys to permit execution of the viruses. Additionally, such viruses are typically only executed once at the start of execution of an infected program and permanently return control to the infected program after execution.

Summary of the Invention

Some of these problems with digital information protection systems may be overcome by providing a mechanism which allows a content provider to encrypt digital information without requiring either a hardware or platform manufacturer or a content consumer to provide support for the specific form of corresponding decryption. This mechanism can be provided in a manner which allows the digital information to be copied easily for back-up purposes and to be transferred easily for distribution, but which should not permit copying of the digital information in decrypted form. In particular, the encrypted digital information is stored as an executable computer program which includes a decryption program that decrypts the encrypted information

to provide the desired digital information, upon successful completion of an authorization procedure by the user.

In one embodiment, the decryption program is executed as a process within a given operating system and decrypts the digital information within the memory area assigned to that process. This memory area is protected by the operating system from copying or access by other processes. Even if access to the memory area could be obtained, for example through the operating system, when the digital information is a very large application program or a large data file, a copy of the entire decrypted digital information is not likely to exist in the memory area in complete form.

By encrypting information in this manner, a platform provider merely provides a computer system with an operating system that has adequate security to define a protected memory area for a process and adequate functionality to execute a decryption program. The content provider in turn may use any desired encryption program. In addition, by having a process decrypt information within a protected memory area provided by the operating system, the decrypted information does not pass through any device driver, memory resident program or other known logical entity in the computer system whose behavior may be controlled to provide unauthorized access to the data. The ability to reverse engineer or attack a computer program with a computer virus also may be reduced.

In another embodiment, the decryption program is part of a dynamically loaded device driver that responds to requests for data from the file containing the encrypted data. When the digital information product is first executed, this device driver is extracted from the file and is loaded into the operating system. The executed digital information product then informs the loaded device driver of the location of the hidden information in the file, any keys or other passwords, and the name of a phantom directory and file to be called that only the digital information product and the device driver know about. The name of this directory may be generated randomly. Each segment of hidden information in the digital information product may be assigned its own unique file name in the phantom directory. The digital information product then makes a call to the operating system to execute one of the files in the phantom directory. The loaded driver traps these calls to the operating system, accesses the original file, decrypts the desired information and outputs the desired information to the operating system.

In combination with other mechanisms that track distribution, enforce royalty payments and control access to decryption keys, the present invention provides an improved method for

identifying and detecting sources of unauthorized copies. Suitable authorization procedures also enable the digital information to be distributed for a limited number of uses and/or users, thus enabling per-use fees to be charged for the digital information.

Accordingly, one aspect of the invention is a digital information product including a computer-readable medium with digital information stored thereon. The digital information includes computer program logic having a first portion of executable computer program logic and a second portion of digital information. The first portion of executable program logic, when executed, defines a mechanism for responding to requests for digital information from an operating system of a computer. This mechanism, when used to access the second portion of the encrypted digital information, decrypts the encrypted digital information, and provides the encrypted digital information to the operating system.

In the foregoing aspect of the invention, the digital information may be executable computer program logic. Hence, one aspect of the invention is a computer program product, including a computer readable medium with computer program logic stored thereon. The computer program logic includes a first portion of executable computer program logic and a second portion of encrypted computer program logic. The first portion of executable computer program logic, when executed, defines a mechanism for responding to requests for computer program logic from an operating system of a computer. This mechanism accesses the second portion of encrypted computer program logic, decrypts the encrypted computer program logic, and provides the decrypted computer program logic to the operating system.

Another aspect of the present invention is a computer program product, a computer system and a process which produce a computer program or digital information product in accordance with other aspects of the invention, using executable program code for the first and second portions of the desired computer program product.

Another aspect of the present invention is a computer program product including a self-decrypting encrypted executable computer program. The product includes a computer readable medium having computer program logic stored thereon. The computer program logic defines first, second and third modules, wherein the third module defines the encrypted executable computer program. The first module, when executed by a computer, defines a mechanism for loading the second module into memory of the computer. The second module, when executed by a computer, defines a mechanism for communicating with an operating system of the computer to receive requests for program code from the encrypted executable computer program from the

third module, and for processing the requests to access and decrypt the encrypted executable computer program and for providing the decrypted executable code from the third module to the operating system.

Another aspect of the invention is a process for executing encrypted executable computer programs on a computer system having a processor, memory and operating system. The process involves receiving computer program logic having a first module defining a start up routine, a second module, and a third module containing the encrypted executable computer program. The first module of the received computer program logic is executed using the processor. When the first module is executed, the second module is caused to be loaded into the memory of the computer system. Requests are generated from the operating system for data from the encrypted executable computer program and are received by the second module. The second module accesses and decrypts the encrypted executable computer program in response to these requests and returns the decrypted executable computer program to the operating system.

These and other aspects, advantages and features of the present invention and its embodiments will be more apparent given the following detailed description.

Brief Description of the Drawing

In the drawing,

Fig. 1 is a block diagram of a typical computer system with which the present invention may be implemented;

Fig. 2 is a block diagram of a memory system in the computer system of Fig. 1;

Fig. 3 is a diagram of a computer program or digital information product which may be recorded on a computer readable and writable medium, such as a magnetic disc;

Fig. 4 is a flowchart describing how the computer program or digital information product of Fig. 3 is used;

Fig. 5 is a flowchart describing operation of an example unwrap procedure as shown in Fig. 3 in one embodiment of the invention;

Fig. 6 is a flowchart describing operation of an example device driver as shown in Fig. 3 in one embodiment of the invention;

Fig. 7 is a block diagram of a computer system in the process of executing a computer program product in accordance with one embodiment of the invention;

Fig. 8 is a flowchart describing operation of an example unwrap procedure in another embodiment of the invention; and

Fig. 9 is a flowchart describing how a computer program product such as shown in Fig. 3 is constructed.

Detailed Description

The present invention will be more completely understood through the following detailed description which should be read in conjunction with the attached drawing in which similar reference numbers indicate similar structures.

Embodiments of the present invention may be implemented using a general purpose digital computer or may be implemented for use with a digital computer or digital processing circuit. A typical computer system 20 is shown in Fig. 1, and includes a processor 22 connected to a memory system 24 via an interconnection mechanism 26. An input device 28 also is connected to the processor and memory system via the interconnection mechanism, as is an output device 30. The interconnection mechanism 26 is typically a combination of one or more buses and one or more switches. The output device 30 may be a display and the input device may be a keyboard and/or a mouse or other cursor control device.

It should be understood that one or more output devices 30 may be connected to the computer system. Example output devices include a cathode ray tube (CRT) display, liquid crystal display (LCD), television signal encoder for connection to a television or video tape recorder, printers, communication devices, such as a modem, and audio output. It also should be understood that one or more input devices 28 may be connected to the computer system. Example input devices include a keyboard, keypad, trackball, mouse, pen and tablet, communication device, audio or video input and scanner. It should be understood that the invention is not limited to the particular input or output devices used in combination with the computer system or to those described herein.

The computer system 20 may be a general purpose computer system, which is programmable using a high level computer programming language, such as "C++," "Pascal," "VisualBasic." The computer system also may be implemented using specially programmed, special purpose hardware. In a general purpose computer system, the processor is typically a commercially available processor, such as the Pentium processor from Intel Corporation. Many other processors are also available. Such a processor executes a program called an operating

system, such as Windows 95 or Windows NT 4.0, both available from Microsoft Corporation, which controls the execution of other computer programs and provides scheduling, debugging, input output control, accounting compilation, storage assignment, data management and memory management, and communication control and related services. Other examples of operating systems include: MacOS System 7 from Apple Computer, OS/2 from IBM, VMS from Digital Equipment Corporation, MS-DOS from Microsoft Corporation, UNIX from AT&T, and IRIX from Silicon Graphics, Inc.

The computer system 20 also may be a special purpose computer system such as a digital versatile disk or digital video disk (DVD) player. In a DVD player, there is typically a decoder controlled by some general processor which decodes an incoming stream of data from a DVD. In some instances, the DVD player includes a highly integrated DVD decoder engine. Such devices generally have a simple operating system which may be modified to include the capabilities described and used herein in connection with the typical operating systems in a general purpose computer. In particular, some operating systems are designed to be small enough for installation in an embedded system such as a DVD player, including the WindowsCE operating system from Microsoft Corporation and the JavaOS operating system from SunSoft Corporation. The operating system allows a content provider to provide its own programs that define some of the content, which is particularly useful for interactive multimedia. This capability also can be used to provide encryption and decryption, in accordance with the invention.

The processor and operating system define a computer platform for which application programs in a programming language such as an assembly language or a high level programming language are written. It should be understood that the invention is not limited to a particular computer platform, operating system, processor, or programming language. Additionally, the computer system 20 may be a multi-processor computer system or may include multiple computers connected over a computer network.

An example memory system 24 will now be described in more detail in connection with Fig. 2. A memory system typically includes a computer readable and writable non-volatile recording medium 40, of which a magnetic disk, a flash memory, rewriteable compact disk (CD-RW) and tape are examples. The recording medium 40 also may be a read only medium such as a compact disc-read only memory (CD-ROM) or DVD. A magnetic disk may be removable, such as a "floppy disk" or "optical disk," and/or permanent, such as a "hard drive." The disk,

which is shown in Fig. 2, has a number of tracks, as indicated at 42, in which signals are stored, in binary form, i.e., a form interpreted as a sequence of 1's and 0's, as shown at 44. Such signals may define an application program to be executed by the microprocessor, or information stored on the disk to be processed by the application program. Typically, in the operation of a general purpose computer, the processor 22 causes data to be read from the non-volatile recording medium 40 into an integrated circuit memory element 46, which is typically a volatile random access memory, such as a dynamic random access memory (DRAM) or static random access memory (SRAM). The integrated circuit memory element 46 allows for faster access to the information by the processor than disk 40, and is typically called the system or host memory. The processor generally causes the data to be manipulated within the integrated circuit memory 46 and may copy the data to the disk 40, if modified, when processing is completed. A variety of mechanisms are known for managing data movement between the disk 40 and the integrated circuit memory 46, and the invention is not limited thereto. It should also be understood that the invention is not limited to a particular memory system.

The file system of a computer generally is the mechanism by which an operating system manages manipulation of data between primary and secondary storage, using files. A file is a named logical construct which is defined and implemented by the operating system to map the name and a sequence of logical records of data to physical storage media. An operating system may specifically support various record types or may leave them undefined to be interpreted or controlled by application programs. A file is referred to by its name by application programs and is accessed through the operating system using commands defined by the operating system. An operating system provides basic file operations provided by for creating a file, opening a file, writing a file, reading a file and closing a file.

In order to create a file, the operating system first identifies space in the storage media which is controlled by the file system. An entry for the new file is then made in a directory which includes entries indicating the names of the available files and their locations in the file system. Creation of a file may include allocating certain available space to the file. Opening a file returns a handle to the application program which it uses to access the file. Closing a file invalidates the handle.

In order to write data to a file, an application program issues a command to the operating system which specifies both an indicator of the file, such as a file name, handle or other descriptor, and the information to be written to the file. Given the indicator of the file, the

operating system searches the directory to find the location of the file. The directory entry stores a pointer, called the write pointer, to the current end of the file. Using this pointer, the physical location of the next available block of storage is computed and the information is written to that block. The write pointer is updated in the directory to indicate the new end of the file.

In order to read data from a file, an application program issues a command to the operating system specifying the indicator of the file and the memory locations assigned to the application where the next block of data should be placed. The operating system searches its directory for the associated entry given the indicator of the file. The directory may provide a pointer to a next block of data to be read, or the application may program or specify some offset from the beginning of the file to be used.

A primary advantage of using a file system is that, for an application program, the file is a logical construct which can be created, opened, written to, read from and closed without any concern for the physical storage used by the operating system.

The operating system also allows for the definition of another logical construct called a process. A process is a program in execution. Each process, depending on the operating system, generally has a process identifier and is represented in an operating system by a data structure which includes information associated with the process, such as the state of the process, a program counter indicating the address of the next instruction to be executed for the process, other registers used by process and memory management information including base and bounds registers. Other information also may be provided. The base and bounds registers specified for a process contain values representing the largest and smallest addresses that can be generated and accessed by an individual program. Where an operating system is the sole entity able to modify these memory management registers, adequate protection from access to the memory locations of one process from another process is provided. As a result, this memory management information is used by the operating system to provide a protected memory area for the process. A process generally uses the file system of the operating system to access files.

The present invention involves storing encrypted digital information, such an audio, video, text or an executable computer program, on a computer readable medium such that it can be copied easily for back-up purposes and transferred easily for distribution, but also such that it cannot be copied readily in decrypted form during use. In particular, the digital information is stored as a computer program that decrypts itself while it is used to provide the digital information, e.g., to provide executable operation code to the operating system of a computer, as

the digital information is needed. Any kind of encryption or decryption may be used and also may include authorization mechanisms and data compression and decompression. In one embodiment of the present invention, decrypted digital information exists only in memory accessible to the operating system and processes authorized by the operating system. When the digital information is a large application program, a copy of the entire decrypted application program is not likely to exist in the main memory at any given time, further reducing the likelihood that a useful copy of decrypted code could be made. The decryption operation also is performed only if some predetermined authorization procedure is completed successfully.

One embodiment of the invention, in which the decryption program is a form of dynamically loaded device driver, will first be described. Fig. 3 illustrates the structure of digital information as stored in accordance with one embodiment of the present invention, which may be stored on a computer readable medium such as a magnetic disc or compact disc read only memory (CD-ROM) to form a computer program product. The digital information includes a first portion 50, herein called an unwrap procedure or application, which is generally unencrypted executable program code. The purpose of the unwrap procedure is to identify the locations of the other portions of the digital information, and may perform other operations such as verification. In particular, the unwrap procedure identifies and extracts a program which will communicate with the operating system, herein called a virtual device driver 52. The unwrap procedure may include decryption and decompression procedures to enable it to decrypt/decompress the driver, and/or other content of this file. The program 52 need not be a device driver. The virtual device driver 52 typically follows the unwrap procedure 50 in the file container, the digital information. The virtual device driver, when executed, decrypts and decodes the desired digital information such as an executable computer program code from hidden information 54, which may be either encrypted and/or encoded (compressed). It is the decrypted hidden information which is the desired digital information to be accessed. This hidden information may be any kind of digital data, such as audio, video, text, and computer program code including linked libraries or other device drivers.

In this embodiment of the computer program product, labels delineate the boundaries between the device driver and the hidden files. These labels may or may not be encrypted. A first label 56 indicates the beginning of the code for the virtual device driver 52. A second label 58 indicates the end of the virtual device driver code. Another label 60 indicates the beginning of the hidden information and a label 62 indicates the end of that application. There may be one

or more blocks of such hidden information, each of which can be given a different name. It may be advantageous to use the name of the block of information in its begin and end tags. This computer program product thus contains and is both executable computer program code and one or more blocks of digital information. A table of locations specifying the location of each portion of the product could be used instead of labels. Such a table could be stored in a predetermined location and also may be encrypted.

The overall process performed using this computer program product in one embodiment of the invention will now be described in connection with Fig. 4. This embodiment may be implemented for use with the Windows95 operating system and is described in more detail in connection with Figs. 5-7. An embodiment which may be implemented for use on the WindowsNT 4.0 operating system is described in more detail below in connection with Fig. 8. In both of these described embodiments, the digital information is an executable computer program which is read by the operating system as data from this file and is executed. The same principle of operation would apply if the data were merely audio, video, text or other information to be conveyed by a user. In the embodiment of Fig. 4, the computer program is first loaded into memory in step 70, and the unwrap procedure 50 is executed by the operating system, as any typical executable computer program is executed. The unwrap procedure may perform authorization, for example by checking for a required password or authentication code, and may receive any data needed for decryption or decompression, for example keys or passwords, in step 72. Suitable authorization procedures may provide the ability to distribute software for single use. The unwrap procedure locates the virtual device driver 52 within the computer program in step 74, and then locates the hidden application in step 76. The virtual device driver 52 is then extracted by the unwrap procedure from the computer program, copied to another memory location and loaded for use by the operating system in step 78. An advantage of an operating system like Windows95 is that it allows such device drivers to be loaded dynamically without restarting the computer.

The executed unwrap procedure 50, in step 80, informs the loaded virtual device driver 52 of the location of the hidden information in the file, any keys or other passwords, and a name of a phantom directory and file to be called that only the unwrap procedure and the virtual device driver know about. The name of this phantom directory may be generated randomly. Each segment information hidden in the digital information product may be assigned its own unique file name in the phantom directory.

After the loaded virtual device driver 52 receives all communications from the unwrap procedure, it opens the original application file for read only access in step 82. The unwrap procedure then makes a call to the operating system in step 84 to execute the file in the phantom directory for which the name was transmitted to the loaded virtual device driver. One function of the loaded virtual device driver 52 is to trap all calls from the operating system to access files in step 86. Any calls made by the operating system to access files in the phantom directory are processed by the virtual device driver, whereas calls to access files in other directories are allowed to proceed to their original destination. In response to each call from the operating system, the virtual device driver obtains the bytes of data requested by the operating system from the original computer program file in step 88. These bytes of data are then decrypted or decompressed in step 90 and returned to the operating system. When processing is complete, the phantom application is unloaded from the operating system in step 92, and may be deleted from the memory.

A more detailed description of the process of Fig. 4 will now be described in connection with Figs. 5-7. Fig. 5 is a flowchart describing the operation of one embodiment of the unwrap procedure in more detail. The first step performed by this procedure is identifying the operating system being used, in step 100. This step is useful because different methods may be used with different operating systems. All code that may be used to run in various operating systems may be placed in this unwrap procedure. This procedure also may contain the decompression/decryption code, for example or any other computer program code to be executed.

The executed application then opens the original executable file as a data file and searches for the begin and end tags of the device driver and hidden files in step 102. The device driver code is copied into memory and loaded into the operating system in step 104. The unwrap procedure then informs the device driver of the name of the original application file, offsets of the hidden files and the name of a phantom directory, which is typically randomly generated (step 106). This communication may be performed using a "DeviceIOControl" function call in the Windows95 operating system. The unwrap procedure then makes a call to the operating system to execute the hidden file in the phantom directory, in step 108.

The operation of one embodiment of a device driver will now be described in connection with Fig. 6. After the device driver is loaded into the operating system, it hooks into a position between the operating system and a file system driver (FSD), in step 110, to intercept

calls made by the operating system to the FSD for data from files in the phantom directory. The FSD is the code within the operating system that performs physical reading and writing of data to disk drives. The operating system makes requests to the FSD for data from files in directories on the disk drives. The driver then receives information from the unwrap procedure including the name of the original file, the location of hidden files within the original file, and the name of the phantom directory created by the unwrap procedure (step 112). The device driver opens the original file as a read only data file. The device driver now traps calls, in step 114, made from the operating system for files in the phantom directory. Calls to other directories are ignored and passed on to the original destination. The device driver then reads the data from the original data file, decrypts and decompresses it, and returns the decrypted/decompressed data to the operating system in step 116.

For example, if the offset for the hidden application in the original data file is 266,270 bytes and the operating system asks for 64 bytes starting at offset 0 of the hidden application in the phantom directory, the device driver reads 64 bytes from the original file starting at offset 266,270, decrypts/decompresses those 64 bytes, and returns the first 64 decrypted/decompressed bytes back to the operating system. From the point of view of the operating system, the 64 bytes appear to have come from the file in the phantom directory. Steps 114 and 116 are performed on demand in response to the operating system.

A block diagram of the computer system in this embodiment, with a device driver loaded and in operation, will now be described in more detail in connection with Fig. 7. Fig. 7 illustrates the operating system 120, the loaded device driver 122, a file system driver 124, the original executable file 126 as it may appear on disk and the unwrap procedure 128. The executable file may in fact be on a remote computer and accessed through a network by the device driver. The unwrap procedure causes the operating system to begin execution of the hidden file by issuing an instruction to execute the file in the phantom directory, as indicated at 130. This command is issued after the device driver 122 is informed of the file name of the original executable file 126, offsets of the hidden files within that file and the name of the phantom directory, as indicated at 132. The operating system then starts making calls to the phantom directory as indicated at 134. The device driver 122 traps these calls and turns them into requests 136 to the file system driver to access the original executable file 126. Such requests actually are made to the operating system 120, through the device driver 122 to the file system driver 124. The file system driver 124 returns encrypted code 138 to the device driver

122. The encrypted code 138 actually passes back through the device driver 122 to the operating system 120 which in turn provides the encrypted code 138 to the device driver 122 as the reply to the request 136 for the original file. The device driver 122 then decrypts the code to provide decrypted code 140 to the operating system 120.

Another embodiment of the invention will now be described in connection with Fig. 8. This embodiment may be implemented using the WindowsNT 4.0 operating system, for example. In this embodiment, the device driver portion 52 of the computer program product is not used. The unwrap procedure for this embodiment begins by identifying the operating system being used similar, which is step 100 in Fig. 5. If the operating system is Windows NT 4.0, for example, a different unwrap procedure for this embodiment is performed. Before describing this unwrap procedure, a brief description of some of the available operating system commands will be provided.

Currently, under all versions of the Window operating system or operating environment from Microsoft Corporation (such as Windows 3.1, Windows 95 and Windows NT 3.51 and 4.0) all executable files (.exe) or dynamic link library (.dll and .ocx) files, which are executable files with different header and loading requirements than .exe files, that are loaded into memory by the operating system must reside as a file either locally, e.g., on a disk drive or remotely, e.g., over a network or communications port. All further references herein to loading an executable will be using the Win32 function calls used in Windows 95 and NT 3.51 and 4.0 operating systems. The CreateProcess() function which loads files with an .exe extension takes ten parameters:

```
BOOL CreateProcess(// Prototype from Microsoft Visual C++ Help Documentation
  LPCTSTR lpApplicationName,                  // pointer to name of executable module
  LPTSTR lpCommandLine,                       // pointer to command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes,  // pointer to process security attributes
  LPSECURITY_ATTRIBUTES lpThreadAttributes,   // pointer to thread security attributes
  BOOL bInheritHandles,                       // handle inheritance flag
  DWORD dwCreationFlags,                      // creation flags
  LPVOID lpEnvironment,                       // pointer to new environment block
  LPCTSTR lpCurrentDirectory,                 // pointer to current directory name
  LPSTARTUPINFO lpStartupInfo,                // pointer to STARTUPINFO
  LPPROCESS_INFORMATION lpProcessInformation  // pointer to PROCESS_INFORMATION
);
```

Three of these parameters are pointers to strings that contain an application file name, command line parameters, and the current directory. The other parameters are security, environmental, and process information. The LoadLibrary() function takes one parameter that is a pointer to a string that contains the application file name:

```
HINSTANCE LoadLibrary(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpLibFileName      // address of filename of executable module
);
```

The LoadLibraryEx() function takes three parameters the first being the same as LoadLibrary(), the second parameter must be null, and the third tells the operating system whether to load the file as an executable or as a data file in order to retrieve resources such as icons or string table data from it and not load it as an executable:

```
HINSTANCE LoadLibraryEx(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpLibFileName,     // points to name of executable module
 HANDLE hFile,              // reserved, must be NULL
 DWORD dwFlags              // entry-point execution flag
);
```

The CreateFile() function is used to create and open files and to load files such as device drivers. This function also requires a pointer to a string that contains the name of a physical file:

```
HANDLE CreateFile(// Prototype from Microsoft Visual C++ Help Documentation
 LPCTSTR lpFileName,                          // pointer to name of the file
 DWORD dwDesiredAccess,                       // access (read-write) mode
 DWORD dwShareMode,                           // share mode
 LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // pointer to security descriptor
 DWORD dwCreationDistribution,                // how to create
 DWORD dwFlagsAndAttributes,                  // file attributes
 HANDLE hTemplateFile                         // handle to file with attributes to copy
);
```

There are other functions such as MapViewOfFile() and MapViewOfFileEx() that map areas of memory to an already opened physical file through a handle to that file. They have the following parameters:

```
LPVOID MapViewOfFile(// Prototype from Microsoft Visual C++ Help Documentation
  HANDLE hFileMappingObject,        // file-mapping object to map into address space
  DWORD dwDesiredAccess,            // access mode
  DWORD dwFileOffsetHigh,           // high-order 32 bits of file offset
  DWORD dwFileOffsetLow,            // low-order 32 bits of file offset
  DWORD dwNumberOfBytesToMap        // number of bytes to map
 );

LPVOID MapViewOfFileEx(// Prototype from Microsoft Visual C++ Help Documentation
  HANDLE hFileMappingObject,        // file-mapping object to map into address space
  DWORD dwDesiredAccess,            // access mode
  DWORD dwFileOffsetHigh,           // high-order 32 bits of file offset
  DWORD dwFileOffsetLow,            // low-order 32 bits of file offset
  DWORD dwNumberOfBytesToMap,       // number of bytes to map
  LPVOID lpBaseAddress              // suggested starting address for mapped view
 );
```

All of the foregoing functions directly use a pointer to a string that is a physical file. The only file functions that do not directly use a physical filename are functions like CreateNamedPipe(), which has the following parameters:

```
HANDLE CreateNamedPipe(// Prototype from Microsoft Visual C++ Help Documentation
  LPCTSTR lpName,                          // pointer to pipe name
  DWORD dwOpenMode,                        // pipe open mode
  DWORD dwPipeMode,                        // pipe-specific modes
  DWORD nMaxInstances,                     // maximum number of instances
  DWORD nOutBufferSize,                    // output buffer size, in bytes
  DWORD nInBufferSize,                     // input buffer size, in bytes
  DWORD nDefaultTimeOut,                   // time-out time, in milliseconds
  LPSECURITY_ATTRIBUTES lpSecurityAttributes  // pointer to security attributes structure
 );
```

The string to which CreateNamedPipe() points using the first parameter is a string that both an existing executable and the operating system know about and does not exist physically. Unfortunately both of the executables that "know" this private name could only be loaded using one of the other procedures that required a physical file. Currently it is not possible to load an executable using a "named pipe" name. Both of or any executables that use the name of the "named pipe" already must have been loaded into memory.

All of the foregoing functions require a physical file because all of them use "file mapping" processes. File mapping allows large executable files to appear to be loaded rapidly since they are rarely completely loaded into memory but rather are mapped into memory. The detriment to this mapping capability is that executable code must remain in physical memory in a file in unencrypted form in order to be loaded, unless there is a middle layer or file system driver that the operating system uses as a physical layer and that decrypts the executable code to the operating system on demand. The potential weakness here is that another file system driver can hook into the operating system to monitor traffic between the operating system and all file system drivers and capture decrypted executable code passing from the file system driver to the operating system. Some operating systems allow such monitoring more than others. Many anti-viral software packages use this technique to prevent computer virus attacks.

One method of loading and executing encrypted executable computer program code is to use a stub executable having two parts. The first part is the normal front end loader code that all executables have. In addition, the first part would perform any authorization which may include receiving a password from the user, then allocate enough memory to hold hidden encrypted code when it is decrypted, either in its entirety or a portion of it, copy the encrypted code into that area of protected (and preferably locked so no disk swapping occurs) memory, decrypt it once it is in memory and only in memory, and then have the operating system load the code only from memory therefore bypassing any file system drivers or TSRs so they have access to only encrypted code.

Some of the file functions listed above and similar functions on other operating systems could be modified easily by a programmer having access to source code for those operating systems, or a new operating system may be made to provide functions which allow direct loading of executable code from memory rather than physical files. For example, in the Win32 commands, a command similar to CreateProcess() command could be provided. The command should have a few extra parameters including the process identifier of the process that contains

the now decrypted executable code, the memory address of the start of the decrypted code, and the size of the decrypted code. The command could also contain a parameter specifying a "call back" function within the first process that would provide decrypted code on demand directly to the operating system through a protected buffer, therefore allowing only a portion of the encrypted code to be decrypted at any one time instead of in its entirety, for better protection and less memory use. The second parameter of the LoadLibraryEx() command that now needs to be NULL could be expanded to hold a structure that contained the same information. Both of these and other similar functions could be changed or created to allow loading executable code either as an .exe, .dll, or other extensions or identifiers, such as by using a "named pipe" name that only the operating system and process that holds decrypted code know about and having the operating system load from the named pipe.

Alternatively, without having such additional capabilities in the operating system, an application program can be divided into two parts. The first part is code that is common to all applications such as code for allocating memory off the heap and code that provides some interaction with the user. This kind of code is generally not code that the content provider is concerned about copying. The second part is the code that the content provider believes is valuable. Typically this valuable code is a business logic code or what would be considered a middle tier of a three-tier environment. A content provider would like to protect this second part of the code, at least much more that the first part of the code. The content provider would place all of the important code to be protected inside a dynamic link library and the code that is not that important would reside in the front end "stub" executable. Both of these would be combined into another executable containing the .dll in encrypted form only, along with any other files, data, information, and/or tables for holding, for example, hardware identifiers. This other executable is the final digital information product.

The first part of the digital information product, i.e., the executable stub, would load and execute normally like any other application. It then would perform any authorization procedures. Once the proper authorization or password was completed successfully, an unwrap procedure would be performed as will now be described in connection with Fig. 8, it would then allocate enough protected memory using a function like VirtualAlloc() as shown in step 150:

```
DWORD nFileSize = 0;
DWORD nPhantomFileSize = 0;
```

```
DWORD exeOffset = 0;
DWORD nPreferredLoadAddress = GetPreCompiledLoadAddress();
CString cCommandFile = UnwrapGetNTCommandFile();
exeOffset = UnwrapGetDllOffset(cCommandFile);
nFileSize = UnwrapGetDllSize(cCommandFile);
nPhantomFileSize = nFileSize + 0x3000; // add any needed extra space
// Increase buffer size to account for page size (currently Intel page size).
DWORD nPageSize = GetPageSize();
nPhantomFileSize += (nPageSize -(nPhantomFileSize % nPageSize));
// Allocate the memory to hold the decrypted executable.
LPVOID lpvBlock = VirtualAlloc((LPVOID) nPreferredLoadAddress,
                nPhantomFileSize,
                        MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
```

This function can request a particular address space. Preferably, this address space is the preferred load address space to which the .dll was linked in order to minimize any needed relocation and fix up code. The stub executable may lock that area of memory in step 152, for example by using VirtualLock() to prevent any memory writes to a swap file, depending on the operating system, as shown below:

```
BOOL bVLock = VirtualLock((LPVOID) nPreferredLoadAddress, nPhantomFileSize);
```

The memory area still should be secure even without this preventive step since the Windows 95 and NT operating systems do not allow any user access to swap files.

The encrypted code is then copied from the digital information product into the allocated protected memory in step 154, for example by using the following command:

```
UnwrapCopyHiddenExeToMem(cCommandFile, exeOffset, nFileSize, (char *) lpvBlock);
```

Once in memory, the stub would then decrypt the code to that same portion of memory in step 156, for example by using the following commands:

```
CwrapDecryptSeed(cPassword.GetBuffer(0), cPassword.GetLength());
CwrapDecrypt((unsigned char *) lpvBlock, 0, nFileSize);
```

Any "fix up and relocation" type services would then be performed in step 158, for example by using the following command:

```
UnwrapFixUpAndRelocateDll(lpvBlock);
```

Possibly, the memory protection may be changed to execute only in step 160, for example by using the VirtualProtect() command as follows:

```
DWORD lpflOldProtect; // variable to get old protection
BOOL bVProtect = VirtualProtect((LPVOID) nPreferredLoadAddress,
                                nPhantomFileSize,
                                PAGE_EXECUTE,
                                &lpflOldProtect);
```

Function calls then can be made into that area of memory that now contains the decrypted code:

```
UnwrapDoDllAlgorithms();
```

Some of the "fix up" operations to be performed above include placing the addresses of external or stub.exe functions into the address place holders of the decrypted .dll or internal code, by using commands similar to the following:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);
```

For instance a wrapper function could be created in the outer stub.exe that received a size parameter, allocated that amount of memory off of the heap, and passed back the starting address of that block of memory. Another example would be to have encrypted algorithms within the hidden, encrypted .dll which would be called at run time from the front end stub once decrypted

within protected memory. The dynamic link library would be compiled and linked to expect a pointer to a function that took that parameter and/or returned a value by including prototypes in the header file as follows:

```
void (*lpCallBackFunc1)();
void (*lpCallBackFunc2)(unsigned long);
```

Function calls to "external" functions also could be added as follows:

```
(*lpCallBackFunc1)();
unsigned long z = x * x;
(*lpCallBackFunc2)(z);
```

At run time the "fix up" code would take the run time address of that "wrapper function" and place it into the pointer address within the .dll block of code as follows:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);
```

This information is readily available using the .cod output files from the compiler, an example of which follows:

```
_TestSum PROC NEAR                                          ; COMDAT
; Line 8
 00000    56          push   esi
; Line 23
 00001    ff 15 00 00 00
          00          call   DWORD PTR_lpCallBackFunc1
; Line 24
 00007    8b 44 24 08 mov    eax, DWORD PTR_a$[esp]
 0000b    50          push   eax
 0000c e8 00 00 00 00 call   _TestSquare
```

```
    00011       83 c4 04        add     esp, 4
    00014       8b f0           mov     esi, eax
    ; Line 25
    00016       8b 44 24 0c     mov     eax, DWORD PTR _b$[esp]
    0001a       50              push    eax
    0001b       e8 00 00 00 00  call    _TestSquare
    00020       83 c4 04        add     esp, 4
    00023       03 c6           add     eax, esi
    ; Line 28
    00025       5e              pop     esi
    00026       c3              ret     0
    _TestSum ENDP
    _TEXT       ENDS
    ;       COMDAT_TestSquare
    _TEXT       SEGMENT
    _x$ = 8
    _TestSquare PROC NEAR                                   ; COMDAT
    ; Line 30
    00000       56              push    esi
    ; Line 32
    00001       8b 74 24 08     mov     esi, DWORD PTR _x$[esp]
    00005       0f af f6        imul    esi, esi
    ; Line 34
    00008       56              push    esi
    00009       ff 15 00 00 00
                00              call    DWORD PTR _lpCallBackFunc2
    0000f 83 c4 04      add     esp, 4
    00012       8b c6           mov     eax, esi
    ; Line 36
    00014       5e              pop     esi
    00015       c3              ret     0
    _TestSquare ENDP
```

Such information also is available from .map output files from the linker where the "f" between the address (i.e., 0a406100) and the object file (i.e. Algorithms.obj) means it is a "flat" address (i.e., hard coded by the linker) and the lack of an "f" means that it is an address pointer to be supplied at run time (load time) where the address that is contained in that address location is used and not the actual address location (i.e., the address that is contained at address location 0a406100 and not 0a406100 itself):

```
0001:00000000      _TestSum                0a401000 f Algorithms.obj
0001:00000030      _TestSquare             0a401030 f Algorithms.obj

0003:00001100      _lpCallBackFunc2        0a406100   Algorithms.obj
0003:00001104      _lpCallBackFunc1        0a406104   Algorithms.obj
```

When the code inside the .dll makes a "call" to a dereferenced pointer, it would jump to the correct function in the outer code and return the expected return value (if any). For example:

```
void CallBackFunction1(){
// This is the first function that exists in the Stub executable
// whose address has been placed at the appropriate location inside the "dll" code
// that has now been decrypted in a block of memory. The code inside the "dll"
// makes a function call to this function. In its encrypted state, the "dll" does not contain
// this address, but merely has a placeholder for the address. The “dll” has enough space
allocated to hold an
// address of this size. After the “dll” has been decrypted at run time, its address is
// placed in that location so the code inside the "dll" that references (or more
// appropriately dereferences) that address can jump (which is function call) to this
// address.
AfxMessageBox(
_T("This is the FIRST Stub.exe call back function being called from the dll."));
        return;
}
```

```
void CallBackFunction2(DWORD nNumber){
// See comment for CallBackFunction1 except this function receives a parameter off
// of the stack.  It could also return a value as well.
        CString
        cString(
T("This is the SECOND Stub.exe call back function being called from the dll"));

        har buffer[20];
        ltoa(nNumber, buffer, 10);

        cString += _T(" with a parameter of ");
        cString += buffer;
        cString.+= _T(".");
        AfxMessageBox(cString.GetBuffer(0));
        return;
}
```

The outer stub.exe would make the same kinds of jumps or function calls into the now protected decrypted code block as follows:

```
DWORD c;

// This command declares a function pointer.  This command is different for different function
// calls.  Here the called function takes two integer parameters and
// passes back a DWORD.
DWORD (*lpFunc)(DWORD,DWORD);

// The function pointer is then pointed to the starting address of the function in the
// block of memory that now holds the decrypted DLL.
lpFunc = (DWORD (*)(DWORD,DWORD)) UnwrapFixUpAndRelocateDll();

// Now call that "function" which is really like all function calls, i.e., a jump to
```

```
// the address where that function exists. In this case, two
// variables are passed to that function and returning a value from that function. This function
illustrates that the function call
// can be more complicated than merely a simple jump
// to an address. Inline assembler code may be used to push the variables onto
// the stack frame and return the variable from the eax register, but this function enables
// the C++ compiler to do the same function.
c = (DWORD) (*lpFunc)(a, b);
```

This mechanism requires the unwrap procedure and the now decrypted code to have intimate knowledge about procedural interfaces of each other but no knowledge about each other's implementation. This is the way most executable .exe files and .dll files behave but with the addition of a series of "wrapper" functions on either side for communication. This method works under Windows 95 and Windows NT 4.0 operating systems and should work under Windows NT 3.51 and other operating systems.

Another modified version of this mechanism that works under the Windows NT 4.0 operating system because of functions specific to Windows NT 4.0 would be to have another hidden and/or encrypted executable within the digital information product. This executable would be copied to a physical disk in an unencrypted form, launched or loaded with the CreateProcess() command in its current form but called with a parameter to load the executable in suspended mode:

```
BOOL success = CreateProcess(cFrontEndExe.GetBuffer(0), 0, 0, 0, TRUE,
                    CREATE_NEW_CONSOLE | CREATE_SUSPENDED,
                0, 0, &startUpInfo, &processInfo);
```

Then the first process would copy the encrypted dll into its own process and decrypt it, allocate enough memory using VirtualAllocEx() in its current form in the second process that has just loaded the expendable front end executable in a suspended state as follows:

```
LPVOID lpvBlockEx = VirtualAllocEx(processInfo.hProcess,
```

```
(LPVOID) nPreferredLoadAddress, nPhantomFileSize,
MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
```

The decrypted code is copied from the first process to the second suspended process using WriteProcessMemory() in its current form:

```
BOOL bWriteProcessMemory = WriteProcessMemory((HANDLE) processInfo.hProcess,
(LPVOID) lpvBlockEx, (LPVOID) nPreferredAddress,
(DWORD) nPhantomFileSize, (LPDWORD) &nBytesWritten);
```

The primary thread of the previously launched second process is then resumed:

```
DWORD nResumed = ResumeThread(processInfo.hThread);
```

Any necessary function pointers are then placed in the correct locations by the second process, the area of memory is locked to prevent any writes to a swap file, and the memory protection is changed to execute only as follows:

```
WriteAddress((char*) 0x0a406104, (DWORD) &CallBackFunction1);
WriteAddress((char*) 0x0a406100, (DWORD) &CallBackFunction2);

BOOL bVLock = VirtualLock((LPVOID) nPreferredLoadAddress, nPhantomFileSize);
DWORD lpflOldProtect; // variable to get old protection
BOOL bVProtect = VirtualProtect((LPVOID) nPreferredLoadAddress,
nPhantomFileSize, PAGE_EXECUTE, &lpflOldProtect);
```

The program can continue running by making and receiving calls to and from the decrypted dynamic link library that now resides in the protected memory of its process using commands such as the following:

```
DWORD c;
```

```
DWORD (*lpFunc)(DWORD,DWORD);
lpFunc = (DWORD (*)(DWORD,DWORD)) ExpendableGetEntryAddress();
c = (DWORD) (*lpFunc)(a, b);
```

The first process can either close down or launch another instance of that same process.

In either of these implementations using the same process or launching into a second process, the hidden encrypted code never passes through a file system driver or memory resident program in decrypted form. Code can be split up among different dynamic link libraries so that no two would reside in memory at the same time in order to protect code further. Both of these systems can be implemented using the Win32 function calls. If additional functions, similar to a CreateProcess() command or a LoadLibrary() command but that take a process identifier and address location in memory to load in an executable instead of a physical file, are provided in an operating system then the entire executable and dynamic link library can be hidden, encrypted, and protected on the physical disk and then decrypted within protected memory and use the operating system loader to load it directly to the operating system from memory without residing in decrypted form on any physical medium.

Having described the operation and use of the computer program product in accordance with the invention, embodiments of which are described above in connection with Figs. 3-8, and the operation of the unwrap procedure and device driver it contains, the process of constructing such a computer program product will now be described in more detail. Referring now to Fig. 9, an embodiment of this process for creating a computer program product is shown. This process can be applied to any digital information including an arbitrary executable computer program, dynamic link libraries and related files of data. All digital information is treated as mere data by this process. Each separate data file is combined into a single file by this process, with an executable program for performing the unwrap procedure, and optionally executable program code for a virtual device driver, into the computer program product. Each file of hidden information has a unique location and is identified by its own begin and end markers as shown in Fig. 3. The first step of this process is opening a new data file for the computer program using a name that will be used to indicate an executable file (step 200). For example, an executable word processing program may be named "word_processor.exe" in the Windows95 operating system.

The three portions of the computer program product are then inserted into the open data file. First, the unwrap procedure is inserted at the beginning of the file in an executable format in step 202. The begin tag for the optional device driver is then inserted in step 204. The executable device driver program code is then inserted in step 206, followed by its corresponding end tag in step 208. For each hidden file to be inserted into this computer program product, steps 210 to 216 are performed. First, the begin tag is inserted in step 210. The begin tag also may include an indication of a unique name of the file which will be used as its name in the phantom directory created by the unwrap procedure. The hidden file is then encrypted and/or compressed in step 212 and inserted into the data file in step 214. The end tag for the hidden file is then inserted in step 216. The device driver and all of the tags may be encrypted also if the unwrap procedure has suitable decryption procedures. The computer program file is closed when the last hidden file is processed.

Using the present invention digital information, such as executable program code or various kinds of data, is loaded and unloaded as needed, and thus does not take up any more memory than is necessary. At no time does unencrypted digital information, such as computer program code, exist on disk in accessible and complete decrypted form. Because the original digital information is available as a read only file in one embodiment of the invention accessible only to the device driver, the digital information may be accessed over networks, from a CD-ROM or from a DVD, and can be made to have a limited number of uses. This mechanism is particularly useful for controlling distribution of computer programs, digitized movies or other information while reducing the cost of such distribution and control. For example, software may be distributed over a network on a single use basis, and charges may be levied on a per use basis. The ability to reverse engineer an application program also may be reduced.

One benefit with this system over some other systems for preventing unauthorized access to digital information is that the content provider maintains control of the encryption applied to the information how it may be decrypted. Any need for either a centralized facility or a predetermined decryption program is eliminated. An operating systems manufacturer or other platform vendor merely provides the capability for the information to be accessed and decrypted on the fly. Since the valuable information and any other tables of authorization codes, passwords, or hardware identifiers that the content provider may use to secure the information resides in one large encrypted file, it becomes difficult, if not impossible, for someone to determine just where any of this information exists.

A potential scenario with authorization procedure in which the present invention may be used is the following. A consumer purchases a DVD disk containing a movie. The user puts the disk into the player. This is the first time the disk is installed. The content provider's functions are loaded into the DVD chip, which looks in the encrypted table and sees that this is the first time this disk is being played. The player then displays on a screen a numeric identifier and toll free phone number. The consumer calls the toll free phone number and inputs the numeric identifier that was displayed on the screen. The content provider provides a numeric password based on the numeric identifier that the user inputs into the DVD. The content provider may develop a database of information about its consumers that also may be used to detect pirating of the digital information product. Now that this authorization has taken place, the software that the content provider wrote, and is now in the DVD chip, takes a hardware identifier from the DVD and encrypts it and puts it in the encrypted and buried table on the disk. Alteratively, the data may be decrypted in memory and re-encrypted back onto the disk using the hardware identifier as part of a key. Now that disk will run and show the movie and will only run on that DVD and no other. The content provider could allow for a table of hardware id's so they could limit the number of DVD's that disk would run on or a limited number of times it can be shown. It should be understood that many other authorization procedures may be used.

In the foregoing scenario, the movie is encrypted on the same disk inside of the encrypted file that contains the table and functions the content provider distributed. The movie is decrypted by the decryption functions contained in the file directly to the DVD chip. At no time does the movie reside anywhere in decrypted form. The content provider can protect the movie with any desired level of security (for both encryption and authorization).

In the present invention, the onus of protection of content does not reside with a hardware manufacturer or platform provider but in the hands of the content provider. The hardware manufacturer only provides the mechanism to protect the digital information through the operating system. The technique and implementation of protection resides in the hands of the content provider. This mechanism allows the content providers to change the level of security as needed without any modifications to the hardware. The security of the content is provided by the encryption/decryption algorithms, public/private keys, and authorization methods which are determined by the content provider. Even each individual product can have its own encryption/decryption algorithms and/or public/private keys. All of these can be changed and enhanced as the market demands.

The present invention also could be used for on-line or live use of digital information. For example, a movie could be retrieved on demand and recorded by a consumer. A set top box could receive the digital information, decrypt it, and then re-encrypt and store the information using, for example, a hardware identifier of the set top box. Since home movies digitally recorded would be encrypted using the hardware identifier of the device used in recording, that home movie could not be played on another or only on a limited number of other devices and/or for only a specified number of times depending on the wishes of the content provider. Since the algorithms are downloaded at the time of recording from a service provider, e.g., the cable company, the content provider (movie company) would provide the encrypted data to the service provider to present to their customers. The service provider need not be concerned with the encryption/decryption and authorization functions used by the content provider. Similar uses are possible with other data transmission systems including, but not limited to, telephone, cellular communications, audio transmission including communication and the like.

In another embodiment, the stub executable program is a first process that is implemented similar to a debugging tool such as the SoftIce debugger from NuMega Technologies or the WinDebug debugger from Microsoft Corporation for Ring 0 kernel level debugging for an Intel processor based architecture, or the CodeView debugger for ring 3 application level debugging. Such a debugger controls execution of a program to be debugged as a second process and steps through each program statement or opcode of the debugged program. The debugging tool could be modified to monitor each opcode that indicates a jump to a program fragment, such as each instruction or a block code. If the program fragment to be executed is not decrypted, the modified debugger decrypts the program fragment before the jump command is allowed to execute. Each program fragment may be re-encrypted after execution. Clearly, unnecessary debugging commands may be omitted from the modified debugger.

Having now described a few embodiments of the invention, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by way of example only. Numerous modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention as defined by the appended claims and equivalent thereto.

CLAIMS

1. A computer-implemented process for executing encrypted computer program logic while maintaining protection against copying of corresponding decrypted executable computer program logic, wherein the encrypted computer program logic is stored in association with first executable computer program logic, the process comprising the steps of:

through an operating system of a computer, reading, loading and executing the first executable computer program logic as a first process having a protected memory area defined by the operating system;

the first process decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process causing loading and execution of the decrypted second computer program logic in the protected memory area.

2. The process of claim 1, wherein the encrypted computer program logic and the first executable computer program logic are stored in a single data file accessible through the operating system.

3. The process of claim 1, wherein the execution of the decrypted second computer program logic is performed as a second process having a second protected memory area defined by the operating system.

4. A digital information product including a computer readable medium having digital information stored thereon, the digital information including computer program logic defining first executable computer program logic, wherein the first executable computer program logic when executed performs the following steps:

storing the encrypted computer program logic in a data file accessible through an operating system of a computer, wherein the data file also includes first executable computer program logic;

through the operating system, reading, loading and executing the first executable computer program logic from the data file as a first process having a protected memory area;

the first process decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process causing loading and execution of the decrypted second computer program logic in the protected memory area.

5. A computer system comprising:

a processor for executing computer program logic;

a main memory operatively connected to the processor for storing digital information including executable computer program logic at memory locations addressed by the processor; and

an operating system defined by executable computer program logic stored in the memory and executed by the processor and having a command which when executed by the processor defines means for creating a process in response to a request specifying a process identifier and a memory location in the main memory, wherein the process identifier indicates the process making the request and the memory location stores executable computer program logic which when executed defines the process.

6. A computer system having an operating system, for decrypting digital information, comprising:

means for storing the encrypted computer program logic in a data file accessible through the operating system, wherein the data file also includes first executable computer program logic;

means, invokable through the operating system, for reading, loading and executing the first executable computer program logic from the data file as a first process having a protected memory area;

the first process defining means for decrypting the encrypted computer program logic into second executable computer program logic and storing the second executable computer program logic in the protected memory area; and

the first process defining means for causing loading and execution of the decrypted second computer program logic in the protected memory area.

7. The computer system of claim 6, wherein the encrypted computer program logic and the first executable computer program logic are stored in a single data file accessible through the operating system.

8. The computer system of claim 6, wherein the execution of the decrypted second computer program logic is performed as a second process having a second protected memory area defined by the operating system.

9. A digital information product, including a computer readable medium with computer readable information stored thereon, wherein the computer readable information comprises:

a first portion of executable computer program logic; and

a second portion of encrypted digital information; and

wherein the first portion of executable program logic, when executed, defines means, operative in response to requests for digital information, for accessing the second portion of encrypted digital information, for decrypting the encrypted digital information, and for outputting the decrypted digital information.

10. The digital information product of claim 9, wherein the encrypted digital information is encrypted executable computer program logic.

11. A computer program product including a self-decrypting encrypted executable computer program, comprising:

a computer readable medium having computer program logic stored thereon, wherein the computer program logic defines:

a first module,

a second module,

wherein the first module, when executed by a computer, defines means for loading the second module into memory of the computer, and

a third module defining the encrypted executable computer program,

wherein the second module, when executed by a computer, defines means for communicating with an operating system of the computer to receive requests for program code from the encrypted executable computer program from the third module, and for processing the

requests to access and decrypt the encrypted executable computer program and for providing the decrypted executable code from the third module to the operating system.

12. A process for executing encrypted executable computer programs on a computer system having a processor, memory and operating system, comprising the steps of:

receiving computer program logic having a first module defining a start up routine, a second module, and a third module containing the encrypted executable computer program;

executing the first module of the received computer program logic using the processor,

wherein the step of executing causes the second module to be loaded into the memory of the computer system, and

generating requests from the operating system for data from the encrypted executable computer program which are received by the second module, and

accessing and decrypting the encrypted executable computer program and returning the decrypted executable computer program to the operating system.

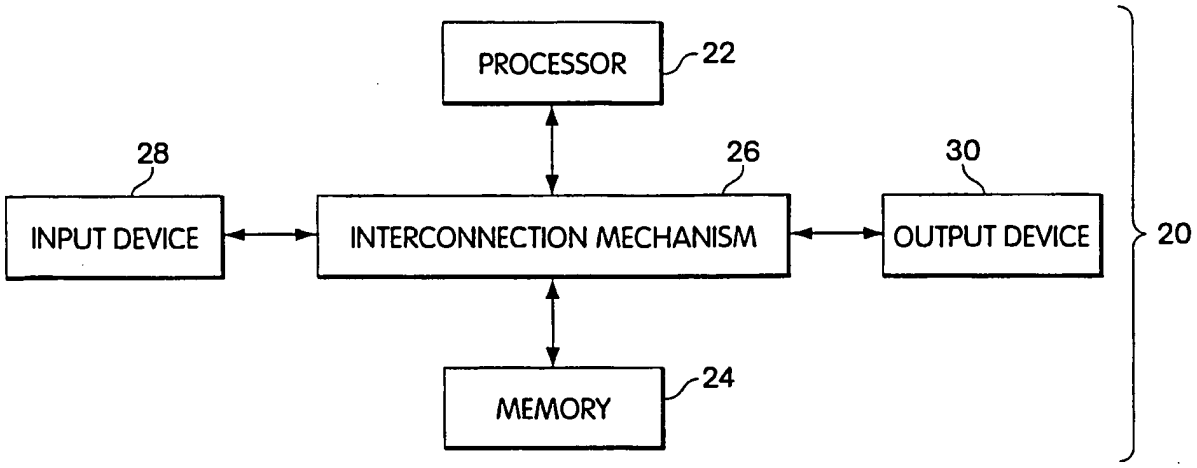


Fig. 1

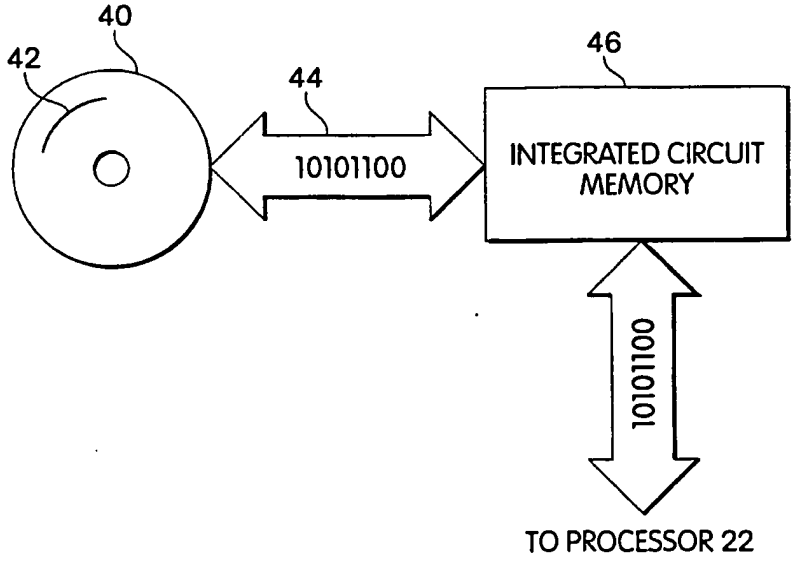


Fig. 2

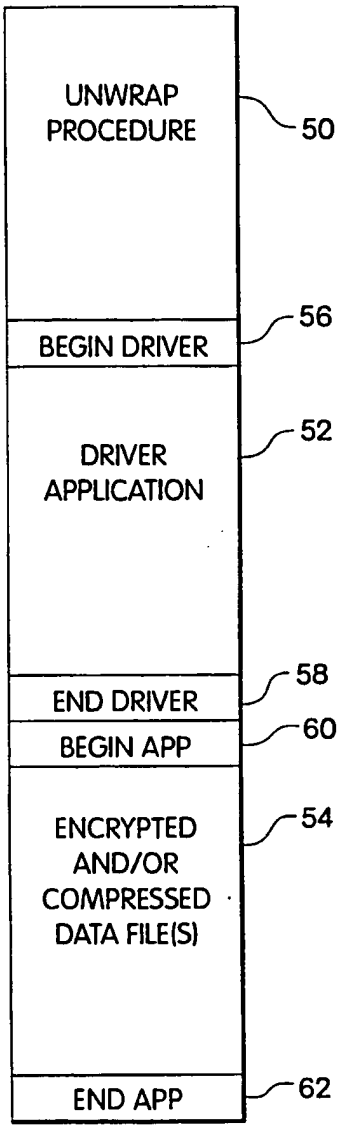


Fig. 3

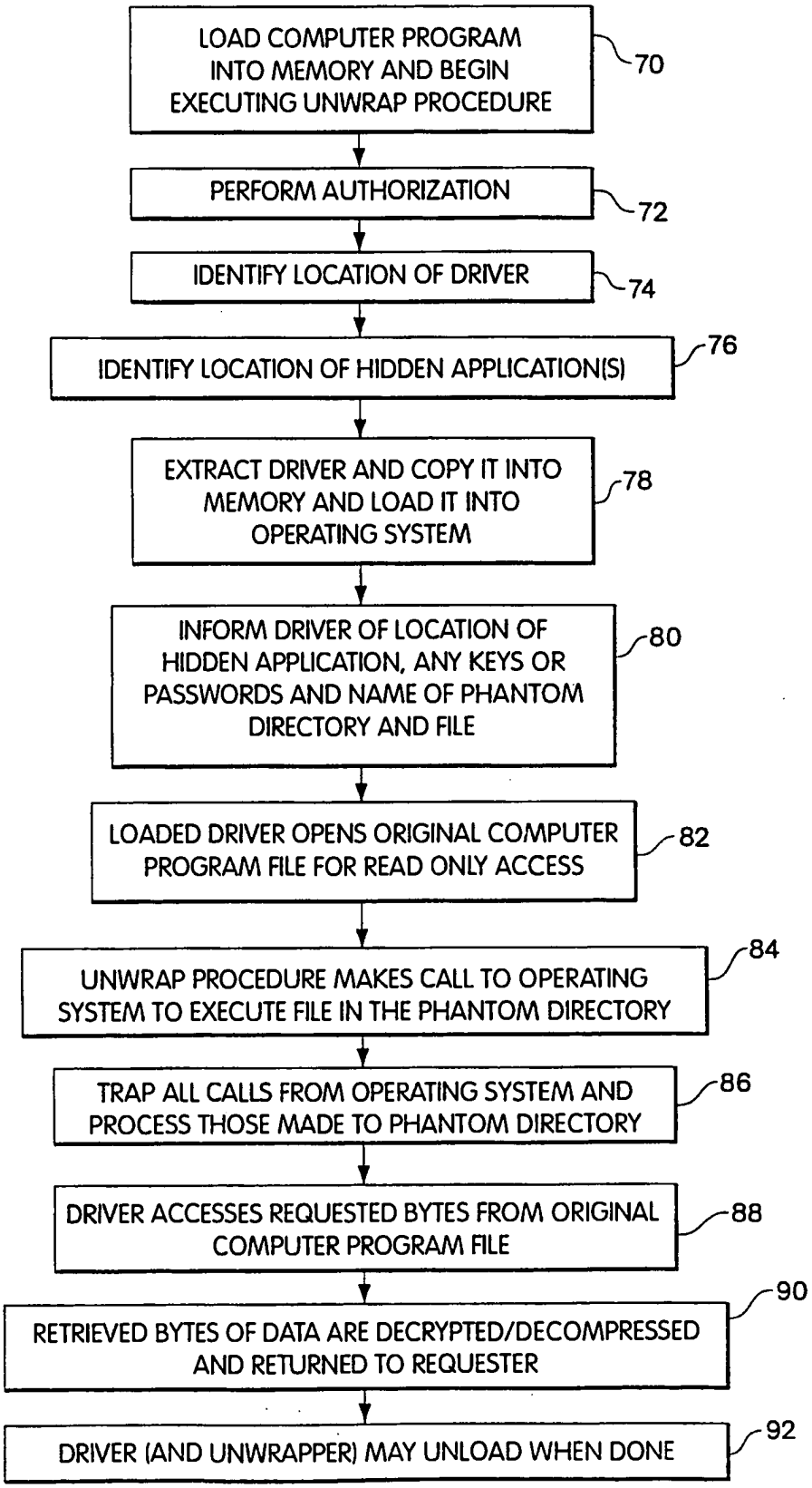


Fig.4

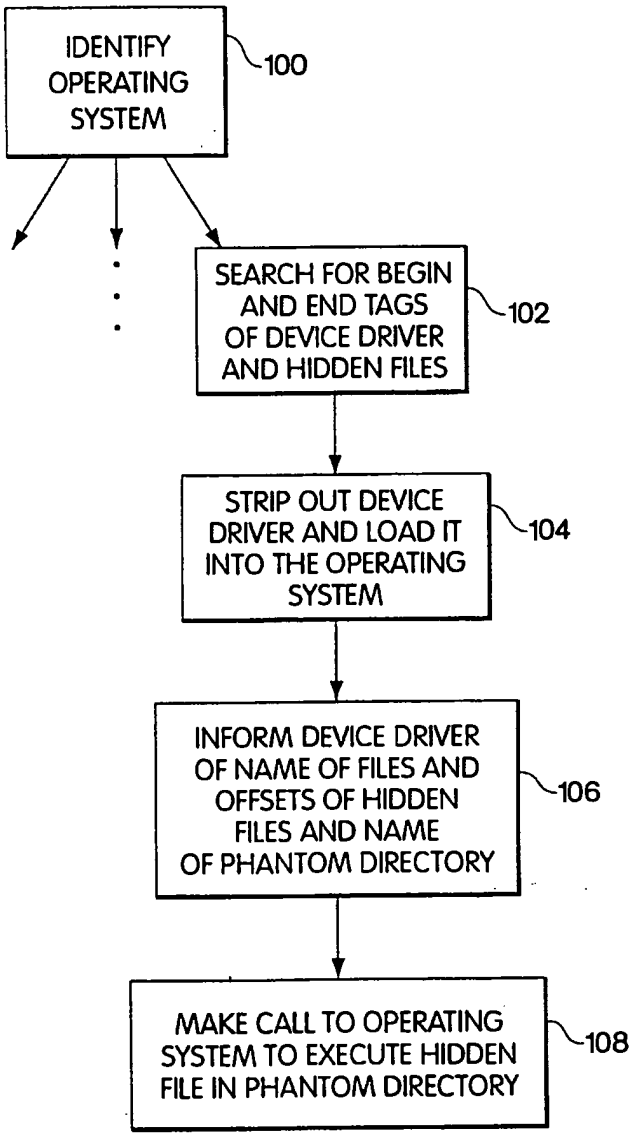


Fig. 5

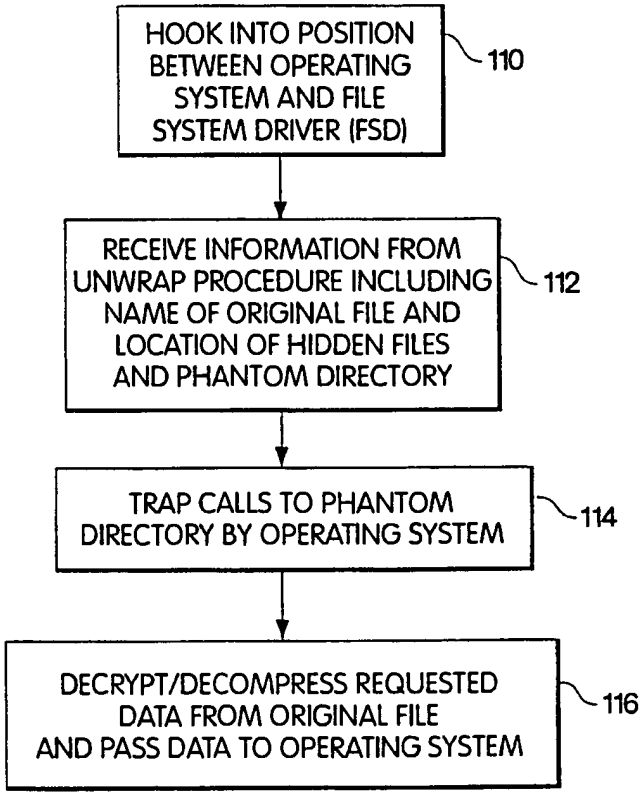


Fig. 6

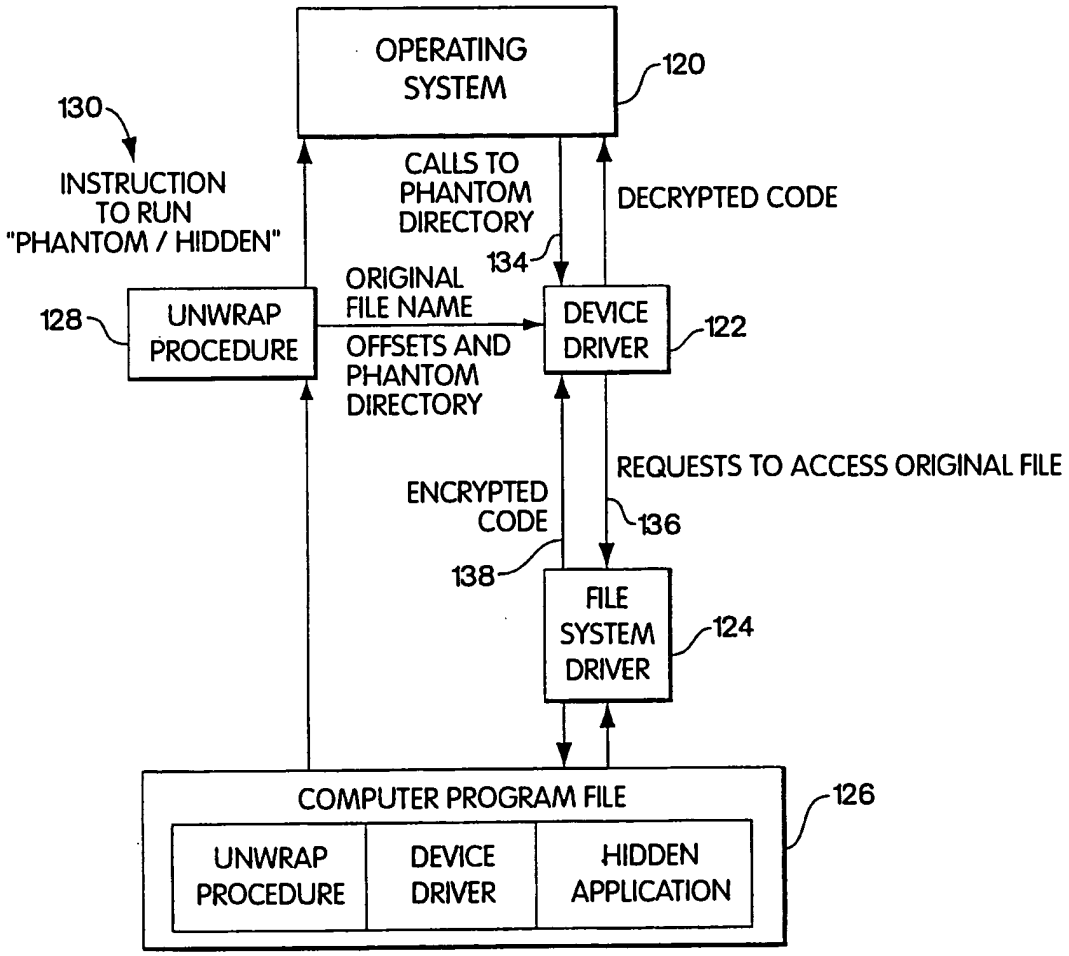


Fig. 7

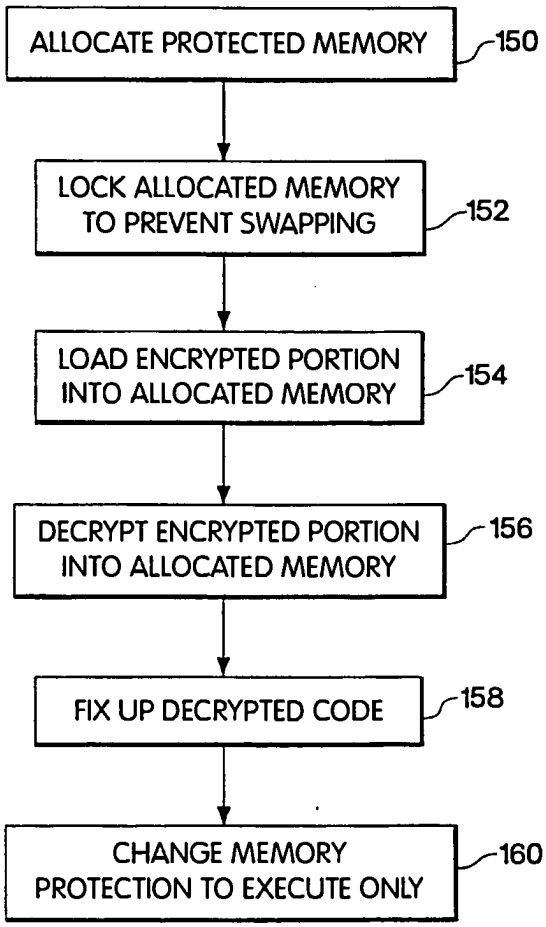


Fig. 8

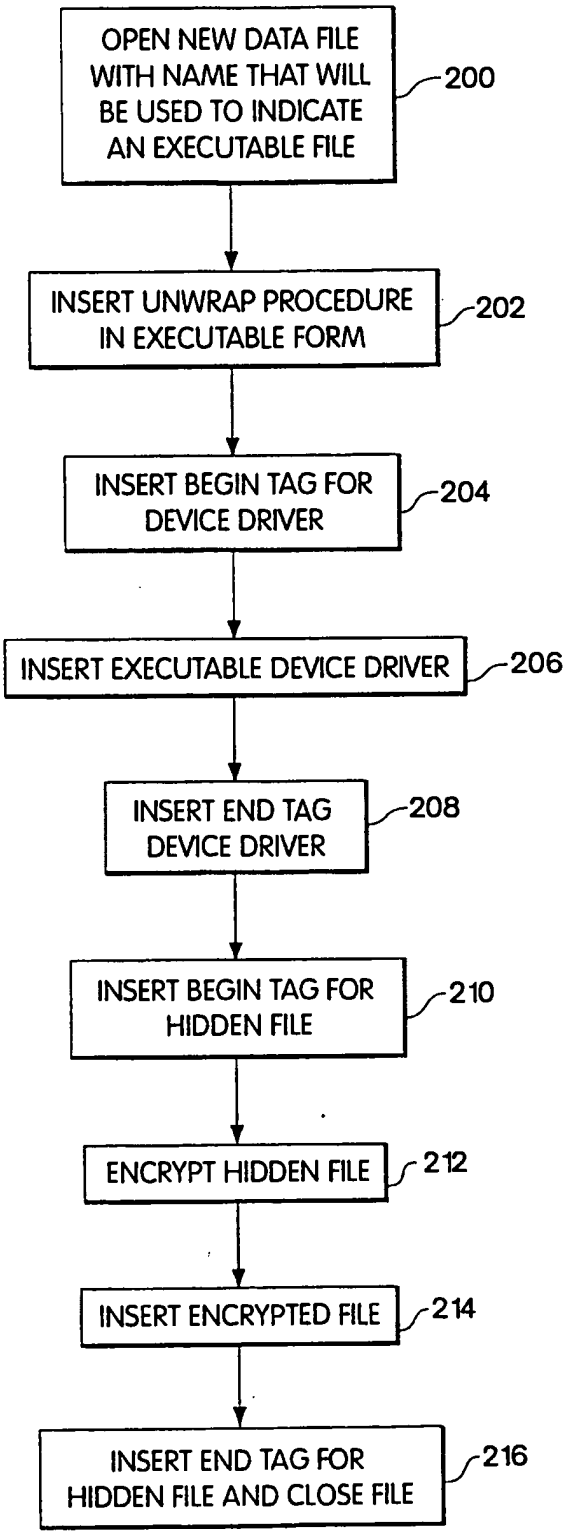


Fig. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/16223

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :H04L 9/00
US CL : 380/4
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. : 380/4,9,23,25,49,50,59

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)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 4,937,861 A (CUMMINS) 26 June 1990, see Abstract.	1-12
A	US 5,007,082 A (CUMMINS) 09 April 1991, see Abstract.	1-12
A	US 5,144,659 A (JONES) 01 September 1992, see Abstract.	1-12
A	US 5,155,827 A (GHERING) 13 October 1992, see Abstract.	1-12
A	US 5,396,609 A (SCHMIDT et al) 07 March 1995, see Abstract.	1-12

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

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance	"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
"B"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search 20 JANUARY 1998	Date of mailing of the international search report 18 FEB 1998
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 BERNARR EARL GREGORY Telephone No. (703) 306-4153

Form PCT/ISA/210 (second sheet)(July 1992)★

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 9/08, H04Q 7/32

A1

(11) International Publication Number: **WO 98/10561**

(43) International Publication Date: 12 March 1998 (12.03.98)

(21) International Application Number: PCT/SE97/01407

(22) International Filing Date: 26 August 1997 (26.08.97)

(30) Priority Data:
08/708,796 9 September 1996 (09.09.96) US

(71) Applicant: TELEFONAKTIEBOLAGET LM ERICSSON (publ) [SE/SE]; S-126 25 Stockholm (SE).

(72) Inventor: RUNE, Johan; Motionsvägen 5, S-181 30 Lidingö (SE).

(74) Agents: BANDELIN, Hans et al.; Telefonaktiebolaget LM Ericsson, Patent and Trademark Dept., S-126 25 Stockholm (SE).

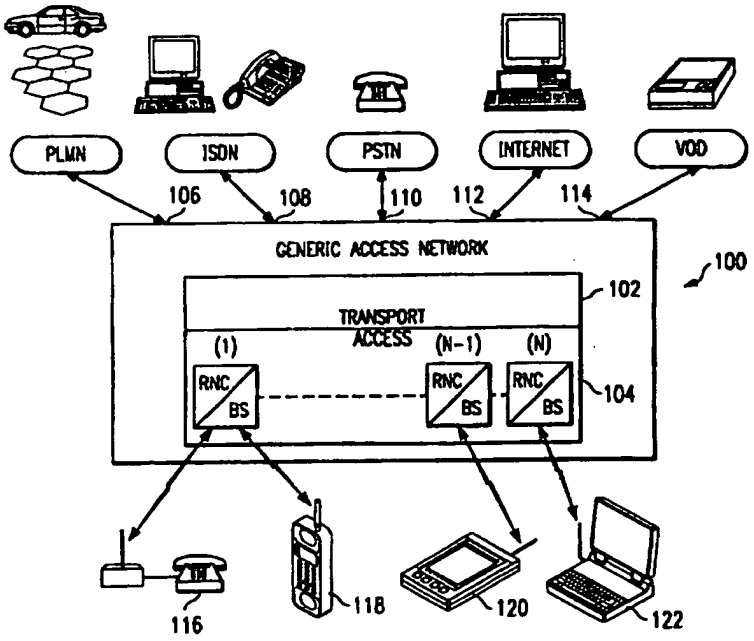
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR ENCRYPTING RADIO TRAFFIC IN A TELECOMMUNICATIONS NETWORK

(57) Abstract

A generic communications network (100) provides an encrypted communications interface between service networks (130, 132, 134) and their subscribers. When communications are initiated between a subscribing communications terminal (118) and the generic network (100), the terminal (118) compares a stored network identifier associated with a stored public key, with a unique identifier broadcast by the generic network (100). If a match is found, the terminal (118) generates a random secret key, encrypts the secret key with the stored public key, and transmits the encrypted secret key. The generic communications network (100) decrypts the secret key using a private key associated with the public key. The secret key is used thereafter by the terminal (118) and the generic network (100) to encrypt and decrypt the ensuing radio traffic. Consequently, the network (100) can maintain secure communications with the terminal (118) without ever knowing the terminal's identity.



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.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD AND APPARATUS FOR ENCRYPTING RADIO TRAFFIC IN A TELECOMMUNICATIONS NETWORK

BACKGROUND OF THE INVENTION

Technical Field of the Invention

The present invention relates generally to the field of wireless radio communications and, in particular, to a method and apparatus for encrypting radio traffic between terminals and a mobile communications network.

Description of Related Art

The need for increased mobility and versatility in telecommunications networks requires the networks to cover larger geographical areas and provide a broader range of telecommunications services to subscribers. These telecommunications services include teleservices and bearer services. A teleservice provides the necessary hardware and software for a subscriber to communicate with another subscriber (e.g., terminal, etc.). A bearer service provides the capacity required to transmit appropriate signals between two access points (e.g., ports) that provide an interface with a network. Telecommunications services can be provided to subscribers by a number of service networks, such as, for example, public land mobile telecommunications networks (PLMNs), public switched telephone networks (PSTNs), integrated services digital networks (ISDNs), the so-called "Internet" access networks, video on demand (VOD) networks, and other proprietary service networks.

In response to the need for increased mobility and versatility, a new mobile radio telecommunications network is being developed, which has a generic interface through which a service network subscriber can be connected with that service network regardless of the subscriber's geographic location. This generic mobile radio network is referred to as the "Generic Access Network" (GAN). In order to more readily understand the present invention, which deals primarily with encrypting communications traffic between terminals and a GAN, a brief description of such a GAN is provided below with respect to FIGURE 1.

FIGURE 1 is a perspective view of an exemplary GAN connected to a plurality of service networks and service network subscribers. The GAN (10) illustrated by FIGURE 1 includes an access network interconnected with a transport network. The access network includes a plurality of base stations (e.g., BS1 and BS2). Each base station includes a radio transmitter and receiver that provides communications coverage for a respective geographical area (e.g., a so-called cell, C1 and C2). The base stations are connected to a radio network controller (RNC) 12. Although not shown explicitly, certain of the base stations can be connected to RNC 12 (e.g., BS1 and BS2), and certain other of the base stations can be connected to one or more other RNCs. A plurality of the RNCs can be interconnected to provide a communications path therebetween. The RNCs distribute signals to and from the connected base stations.

A plurality of service networks (e.g., VOD network, PLMN, PSTN, Internet) are connected through respective access input ports (14, 16, 18, 20, 22, 24 and 26) to the access network of GAN 10. Each service network uses its own standard signaling protocol to communicate between its internal signaling nodes. For example, the Global System for Mobile communications (GSM), which is a digital cellular PLMN that has been fielded throughout Europe, uses the Multiple Application Part (MAP) signaling protocol. As illustrated by FIGURE 1, the RNCs in the access network are connected through at least one of the access input ports to a service network. As shown, RNC 12 is connected through access ports 20 and 24, respectively, to the PLMN and PSTN service networks.

Mobile terminals 28 and 30 are located within the radio coverage area of GAN 10, and can establish a connection with each of the base stations (e.g., BS2) in the access network. These mobile terminals can be, for example, a cellular phone, mobile radiotelephone, personal computer (notebook, laptop, etc.) possibly connected to a digital cellular phone, or mobile television receiver (for VOD). Signal transport between a mobile terminal and a selected service network takes place over specified signal carriers. For example, signals are transported between the cellular phone (28) and the PLMN service network over signal carriers SC1 and SC2.

The mobile terminals (e.g., 28 and 30) include an access section and service network section. The access section of a mobile terminal is a logical part of the access network and handles the signaling required to establish the signal carrier (e.g., SC2 and SC4) between the mobile terminals and RNC 12. The service network section of a mobile terminal is a logical part of the service network to which that terminal's user subscribes. The service network section of a mobile terminal receives and transmits signals, in accordance with the specified standards of its related service network, via the established signal carriers SC1 and SC2 (or SC4). The radio interface portion of the signal carrier SC2 or SC4 (between the mobile terminal and base station) can be time division multiple access (TDMA), code division multiple access (CDMA), or any other type of multiple access interface.

The service network subscribers can access their respective service network through the GAN. The GAN provides a signal carrier interface that allows a message to be transported transparently over a signal carrier (e.g., SC1 and SC2) between the service network part of a mobile terminal and its service network. The GAN accomplishes this function by matching the characteristics of the signaling connections and traffic connections of all of the service networks that connect to it. Consequently, the GAN can extend the coverage of existing service networks and also increase the subscribers' degree of mobility.

A unique characteristic of a GAN is that it has no subscribers of its own. The mobile users of the GAN are permanent subscribers to their own service networks, but they are only temporary users of the GAN. Consequently, a GAN does not know (or need to know) the identity of these users. However, a problem arises in attempting to encrypt radio traffic between the mobile terminals and the GAN.

Radio traffic (e.g., speech information or data) between mobile terminals and base stations is typically encrypted to ensure that the information being passed remains confidential. Although some service networks (e.g., GSM) encrypt traffic, most other service networks do not. Consequently, a GAN should be capable of encrypting traffic for those service networks that do not have that capability.

However, since a GAN does not know the identity of its users (the service network subscribers), it must be capable of encrypting radio traffic using encryption keys that are created without knowing a subscribing terminal's identity or authenticity. Unfortunately, most existing mobile communications networks use encryption techniques that generate encryption keys by using authentication parameters. In other words, to encrypt radio traffic in a conventional mobile communications network, the user terminal's identity must be known.

SUMMARY OF THE INVENTION

It is an object of the present invention to encrypt communications between a mobile terminal and a communications network without requiring the network to know the identity of the terminal.

It is also an object of the present invention to encrypt communications between a plurality of mobile terminals and a communications network without requiring the network to maintain individual encryption keys for each of the terminals.

It is another object of the present invention to encrypt communications between a mobile terminal and a communications network without requiring the terminal to permanently store a secret encryption key.

It is yet another object of the present invention to minimize call setup time, minimize transmission delays, and maximize data throughput, while encrypting communications between a mobile terminal and a communications network.

In accordance with one aspect of the present invention, a method is provided for encrypting communications between a communications network and a communications terminal, by storing a public key associated with the network at the terminal, generating a secret key at the terminal, encrypting the secret key with the stored public key at the terminal, transmitting the encrypted secret key from the terminal, receiving the encrypted secret key at the network, decrypting the received encrypted secret key with a private key, where the private key is associated with the public key, and encrypting the ensuing traffic with the secret key. If a public key has not been stored at the terminal, then the terminal transmits a request to the

network for a public key. As such, the network is not required to know the identity of the terminal in order to maintain encrypted communications with the terminal.

In accordance with another aspect of the present invention, the foregoing and other objects are achieved by a method and an apparatus for encrypting traffic between a communications network and a communications terminal by broadcasting a (asymmetric) public key from the network. The public key is received by the terminal. The network maintains a private key that can be used to decrypt information encrypted with the public key. The terminal generates and stores a naturally occurring random number as a secret session (symmetric) key, encrypts the symmetric session key with the public key, and transmits the encrypted session key to the network. The network decrypts the session key with the private key, and both the network and terminal encrypt the ensuing communications with the secret session key. Again, the communications network is not required to know the identity of the terminal in order to maintain encrypted communications with the terminal.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the method and apparatus of the present invention may be had by reference to the following detailed description when taken in conjunction with the accompanying drawings wherein:

FIGURE 1 is a perspective view of an exemplary generic access network connected to a plurality of service networks and service network subscribers;

FIGURE 2 is a top level schematic block diagram of a generic access network in which a method of encrypting radio traffic between service networks and service network subscribers can be implemented, in accordance with a preferred embodiment of the present invention;

FIGURE 3 is a schematic block diagram of the access network illustrated in FIGURE 2;

FIGURE 4 is a sequence diagram that illustrates a method that can be used to encrypt radio communications between a generic access network and a terminal, in accordance with a preferred embodiment of the present invention; and

FIGURE 5 is a block diagram of a method that can be used to certify the authenticity of a public key and the owner of the key with a digital signature, in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

The preferred embodiment of the present invention and its advantages are best understood by referring to FIGUREs 1-5 of the drawings, like numerals being used for like and corresponding parts of the various drawings.

Essentially, in accordance with a preferred embodiment of the present invention, a mobile terminal stores at least one public key, along with a unique identification character of at least one GAN associated with that public key, in a memory location. A GAN continuously broadcasts its unique identification character in all cells connected to that GAN. When contact is initiated between the terminal and that GAN, the terminal compares the received identifier with the stored identifier(s), and if a match can be made, the terminal generates a random secret key, encrypts the secret key with the public key associated with that GAN's identifier, and transmits the encrypted secret key. The GAN decrypts the secret key using a private key associated with the public key. The secret key is used thereafter by the terminal and the GAN to encrypt and decrypt the ensuing radio traffic. Notably, the GAN can maintain secure communications with the terminal without ever knowing the terminal's identity. Furthermore, since the GAN does not need to know the identity of such a terminal, the GAN is not required to maintain a database of individual terminal encryption keys. Additionally, the terminal is not required to store its own secret key, because it can generate a new secret key for each communications session.

FIGURE 2 is a top level schematic block diagram of a generic access network in which a method of encrypting radio traffic between service networks and service network subscribers can be implemented, in accordance with a preferred embodiment of the present invention. A GAN 100 is shown, which includes a transport network 102 interconnected with an access network 104. A plurality of service networks (e.g., PLMN, ISDN, PSTN, INTERNET, VOD) are connected

through respective access ports (e.g., 106, 108, 110, 112, 114) to transport network 102 and access network 104. Access network 104 includes a plurality of RNCs and associated base stations (e.g., RNC(1)-RNC(N)). The plurality of RNCs and associated base stations are connected by a respective radio interface to a plurality of mobile transceivers (terminals) 116, 118, 120 and 122. A user of each mobile terminal is a subscriber to at least one of the service networks PLMN, etc. The mobile terminals can communicate with their respective service networks in the manner described above with respect to FIGURE 1. More specifically, the RNCs control communications between the terminals and their respective service networks. Notably, although a plurality of mobile terminals (116, etc.) are shown in FIGURE 2, this is for illustrative purposes only. One or more fixed radio terminals may also be connected to GAN 100 and are thus capable of communicating with at least one of the service networks.

FIGURE 3 is a schematic block diagram of access network 104 illustrated in FIGURE 2. Access network 104 includes a plurality of RNCs (e.g., RNC(1)-RNC(N)). However, although a plurality of RNCs is shown for this embodiment, the present invention can be implemented with only one RNC. At least one service network (e.g., 130, 132, 134) is connected through at least one respective access port (e.g., AP1, AP(N-1), AP(N)) to at least one RNC. At least one base station (e.g., BS(1), BS(N)) is connected to a respective RNC (e.g., RNC(1), RNC(N)). Although a plurality of base stations is shown, the present invention can be implemented with only one base station.

A mobile terminal (e.g., cellular phone 118) is connected by a radio interface to base station BS(1). It should be readily understood that one terminal (118) is shown for illustrative purposes only and that one or more additional terminals could be shown. The RNCs (e.g., RNC(1)-RNC(N)) are interconnected by communications lines (136, 138) for communications therebetween. Consequently, terminal 118 can establish communications with any of the service networks (e.g., 130, 132, 134) through access network 104 and GAN 100 (FIGURE 2). Notably, the coverage provided for each service network can be enlarged by switching to a different access port of access network 104. In other words, terminal 118 can

communicate with service network 132 through RNC(1), interconnecting line 136, and RNC(N-1). Alternatively, if service network 132 is switched to access port AP(1), terminal 118 can communicate with service network 132 through RNC(1).

FIGURE 4 is a sequence diagram that illustrates a method that can be used to encrypt radio communications between a generic access network and a terminal, in accordance with a preferred embodiment of the present invention. The method 200 of encrypting communications can begin at the GAN or the terminal. For example, in this embodiment, at step 204, the GAN (e.g., 10) continuously broadcasts a unique identification character in all cells connected to that GAN. The terminal (e.g., 118) contains a non-volatile memory located in a GAN section of the terminal. The terminal stores at least one public key in the non-volatile memory. Along with each public key, the terminal also stores a respective expiration date for the key, and a GAN identification character that identifies a specific GAN associated with that key. In other words, each public key stored in the terminal's memory is thereby associated with a specific GAN. The terminal initiates contact by registering with a GAN (but not necessarily setting up a call). A processor in the terminal compares the received GAN identifier with the stored identifiers, and if a match can be made (and the key has not expired), the processor retrieves the stored public key associated with the identified GAN. However, in the event that no such match is found, the terminal sends a request for the GAN to transmit a public key. The transmitted public key (and its expiration date) is stored in the terminal and can be used to encrypt a secret key in the current and ensuing communication sessions.

At step 206, the terminal generates a (symmetric) secret key (described in detail below). At step 208, the terminal uses the retrieved public key to encrypt the secret key. At step 210, the terminal transmits the encrypted secret key to the identified GAN. At step 212, the GAN decrypts the secret key, which, at step 214, is used by the GAN and the terminal for encrypting traffic during the ensuing communications session (described in detail below).

Alternatively, at the end of a session with a GAN, the terminal stores the public key used for that session. When the terminal or a GAN begins a new communications session, the terminal retrieves the public key stored from the last

session with a GAN, and uses that public key to encrypt a secret key to be used for the ensuing session. If the use of that stored public key is unsuccessful, the terminal then sends a request to the GAN for a new public key. This technique advantageously increases network throughput, because a network channel is not tied up transmitting a public key. However, if a public key has not been stored from a past session with a particular GAN, the terminal can still receive the public key by requesting it from the GAN and using it to encrypt a secret key that will be used for the ensuing session. In any event, by storing the relatively large (bit-wise) public keys in the terminal, as opposed to transmitting them from the GAN, radio transmission delays can be reduced significantly, a substantial amount of network transmission time can be saved, and data throughput will be increased.

FIGURE 4 also illustrates a method that can be used to encrypt radio communications between a generic access network and a mobile terminal, in accordance with another embodiment of the present invention. For example, when communications are desired between a service network and a terminal (e.g., PLMN and terminal 118), the service network or terminal can initiate communications with a call setup message. At step 202, as the initial connection between the GAN and the terminal is established, the service network can request that the ensuing traffic will be encrypted. If so, at step 204, still during the initial call setup process, the terminal receives a public key which is continuously broadcast from one or more base stations (e.g., BS(1)-BS(N)).

In this embodiment, all of the RNCs maintain at least one public key/private key pair (the same pair in every RNC) in a memory storage location. The public key that was broadcast by the GAN is received by the terminal (118) that has initiated contact with that GAN. Preferably, both the call setup procedure and the procedure to transfer the public key is performed by an RNC, which is connected through an access port to the service network of interest (e.g., RNC(1) to AP(1) to PLMN 130). Alternatively, a base station (e.g., BS1) can be configured to maintain public/private key pairs and broadcast or otherwise transfer a public key to a terminal.

The RNC can broadcast the public key in all cells in the RNC's coverage area. Consequently, since the GAN broadcasts the public key instead of having the terminal request the key from the GAN, the terminal can register with the GAN much faster, and a call can be set up in a substantially shorter period of time. Alternatively, instead of broadcasting the public key in a plurality of cells, the RNC can transfer the public key directly through the base station that has established contact with the terminal. However, the method of broadcasting the public key in a plurality of cells before call setup advantageously decreases the load on the GAN's dedicated traffic channels.

For all embodiments, as long as the terminal is registered with the GAN, the same public key can be used for all subsequent communications with that GAN, because the same key is stored at the GAN and also at the terminal. Alternatively, the public key can be changed periodically in accordance with a predetermined scheme or algorithm, or even at the whim of the GAN operator. If an operator desires to change public keys periodically, storing each public key's expiration date at the terminal facilitates their use in that regard. Furthermore, in the preferred embodiment, when the public key is changed, it can be broadcast by the GAN for a predetermined period of time, to minimize the number of terminal requests for a new public key.

As described earlier, at step 202, the GAN can maintain one or more asymmetric public key/private key pairs. In that event, a so-called "RSA Algorithm" can be used to create the public key/private key pairs. The RSA Algorithm combines the difficulty of factoring a prime number with the ease of generating large prime numbers (using a probabilistic algorithm) to split an encryption key into a public part and a private part.

Specifically, assuming that the letters P and Q represent prime numbers, the letter M represents an unencrypted message, and the letter C represents the encrypted form of M, the RSA Algorithm can be expressed as follows:

$$M^E \bmod PQ => C \text{ (encrypted message M)} \quad (1)$$

$$C^D \bmod PQ => M \text{ (decrypted message C)} \quad (2)$$

where the term (DE-1) is a multiple of (P-1)(Q-1). In this embodiment, the exponent E is set to 3. The private and public keys are each composed of two numbers. For example, the numbers represented by (PQ, D) make up the private key, and the numbers represented by (PQ, E) make up the public key. Since the same value for E is used consistently, only the PQ portion of the number need be sent on request or broadcast and used for the public key (e.g., at step 204). By knowing the private key, any message encrypted with the public key can be decrypted.

Returning to FIGURE 4, at step 206, the terminal (118) receives and/or stores the asymmetric public key. The terminal generates a random symmetric secret key. The random secret key, which is used to encrypt communications preferably for the complete session, can be generated in at least one of four ways. Using one method, the terminal takes several samples from measurements of the strength of the received signal, concatenates the lower order bits of the several samples, and processes the result to produce a random number. Since the lower order bits of the received signal are well within the noise level of the received signal, a naturally occurring, truly random number is generated. A second random number generating method is to use the random noise signal created at the input of an A/D converter connected to a microphone. Again, using this method, a naturally occurring, truly random number can be generated for the secret key. A third random number generating method is for the terminal to take samples from phase measurements of the received signal, concatenate the lower order bits of the samples, and process the result to produce a random number. A fourth random number generating method is for the terminal to take samples from the encoding section of the speech codec, concatenate the lower order bits of the samples, and process the result to produce the random number.

Alternatively, a random number generated at the terminal can be used as a seed for a pseudorandom number generator. The seed is encrypted with the public key from the GAN, and transmitted to the GAN. The seed is used simultaneously in the GAN and the terminal to produce a pseudorandom number. The

pseudorandom number thus generated can be used by the GAN and the terminal as the secret key for the ensuing communications session.

The session key can be changed periodically to a different number in the pseudorandom number sequence. For example, the session key can be changed for a number of reasons, such as after a predetermined amount of data has been encrypted, or after traffic has been encrypted for a predetermined amount of time. The terminal or the GAN can initiate a change of the secret key, or the key can be changed according to a predetermined scheme or algorithm. For example, a request to change the secret session key can be implemented by transmitting a "session key change request" message, or by setting a "session key change request" bit in the header of a transmitted message.

Additionally, shorter session keys can be generated and less complicated encryption algorithms can be used with the pseudorandom number generation method described above. Consequently, a substantial amount of processing power can be saved in the GAN and especially in the terminal. The terminal can be configured to select the length of the session key to be used, in order to address trade offs between security and computational requirements. For example, the terminal's processor can select the length of a secret session key by generating a session key at that length, or by specifying the number of bits to be used from the output of the pseudorandom number generator. Alternatively, the terminal can specify the range of the output of the pseudorandom number generator to set a predetermined length.

Other alternative methods may be used to generate a pseudorandom number for a secret session key. For example, using a "Lagged Fibonacci" type of pseudorandom number generator, the n^{th} number in the pseudorandom number sequence, N_n, can be calculated as follows:

$$N_n=(N_{n-k}-N_{n-l}) \bmod M \qquad (3)$$

where k and l are the so-called lags, and M defines the range of the pseudorandom numbers to be generated. For optimum results, the largest lag should be between 1000 and 10000. If a relatively long key is desired, a plurality of the pseudorandom numbers produced by equation 3 can be concatenated to produce a longer key. If

the pseudorandom numbers produced by equation 3 are to be floating point numbers between 0 and 1, M can be set to 1. The bit patterns of such floating point pseudorandom numbers can be used as symmetric encryption keys.

Another pseudorandom number generator that can be used to create a secret session key is based on an algorithm that produces pseudorandom numbers uniformly distributed between 0 and 1. Specifically, the seeds X_0, Y_0 and Z_0 of the pseudorandom numbers N_n are initially set to integer values between 1 and 30000. The pseudorandom numbers N_n are then calculated as follows:

$$X_n = 171*(X_{n-1} \bmod 177)-(2*X_{n-1}/177) \quad (4)$$

$$Y_n = 172*(Y_{n-1} \bmod 176)-(35*Y_{n-1}/176) \quad (5)$$

$$Z_n = 170*(Z_{n-1} \bmod 178)-(63*Z_{n-1}/178) \quad (6)$$

If any of the values of X_n, Y_n or Z_n are less than zero, respectively, then X_n is set equal to $X_n+30269$, Y_n is set equal to $Y_n+30307$, or Z_n is set equal to $Z_n+30323$. The pseudorandom numbers N_n are then equal to $((X_n/30269 + Y_n/30307 + Z_n/30323) \text{ amod } 1)$, where X_n, Y_n and Z_n are floating point numbers, and "amod" means that these numbers can be fractions. The floating point numbers generated with this algorithm form bit patterns that are suitable for use as symmetric encryption keys. The length of such keys can be extended by concatenating a plurality of the pseudorandom numbers generated.

Returning to the method illustrated by FIGURE 4, at step 208, preferably using the above-described RSA Algorithm, the terminal encrypts the secret symmetric key with the public key. For example, assume that the secret symmetric key generated at the terminal is represented by the letters SK. Using equation 1 of the RSA Algorithm, the secret key is encrypted as follows:

$$M^E \bmod PQ => C$$

where (PQ, E) represents the public key, M is equal to SK, and C is the encrypted version of SK. The exponent E is set to 3.

In the preferred embodiment, the terminal places the encrypted secret key into a message format, which includes a header and message field. The header provides control information associated with the encrypted secret key that follows in the message field. A bit in the header can be set to indicate that the message field

that follows the header is encrypted. In other words, only the secret key field of the message is encrypted. The header of the message is transmitted in the clear. Consequently, a substantial amount of network processing time can be saved at the RNC, since the header indicates whether the subsequent message field is encrypted, and if so, only that portion of the message is to be decrypted.

At step 210, the terminal (118) transmits the encrypted secret key (C) to the GAN via the contacted base station (e.g., BS(1)). In the preferred embodiment, this secret key is used for the ensuing communications. Alternatively, at any time during the ensuing communications session, the terminal can generate a new secret key, encrypt it with the public key, and transmit the new encrypted secret key to the GAN. The security of the session is thereby increased, because by reducing the amount of time that a particular secret key is used for a session, the likelihood that the secret key will be broken by an unauthorized user is also reduced.

At step 212, the RNC (e.g., RNC(1)) receives the encrypted secret key (C) from the base station, and decrypts the secret key using the private key part of the RSA Algorithm. For example, using equation 2 (above) of the RSA Algorithm, the received encrypted secret key (C) is decrypted as follows:

$$C^D \bmod PQ => M$$

where (PQ, D) represents the private key, and M is equal to SK (secret key).

At step 214, the ensuing radio traffic between the RNC and the terminal is encrypted and decrypted with the secret key, which is now known to both the RNC and the terminal. A known symmetric encryption algorithm can be used to encrypt and decrypt the ensuing radio traffic with the secret key, such as, for example, a one, two or three pass Data Encryption Standard (DES) algorithm, or a Fast Encipherment Algorithm (FEAL).

As yet another encryption alternative, instead of using the RSA Algorithm to create a public/private key pair, a so-called Diffie-Hellman "exponential key exchange" algorithm can be used to let the terminal and the GAN agree on a secret session key. In using this encryption scheme, two numbers (α, q) are stored at the GAN. At the beginning of a communications session, the RNC transmits the two numbers directly (or broadcasts the numbers) to the terminal. The numbers α and

q are required to meet the following criteria: q is a large prime number that defines the finite (Galios) field GF(q) = 1, 2..., q-1; and α is a fixed primitive element of GF(q). In other words, the exponents (x) of (α^x mod q) produce all of the elements 1,2..., q-1 of GF(q). In order to generate an agreed to secret session key, the two numbers (α, q) are transmitted directly (or broadcast) from the GAN to the terminal. Alternatively, the two numbers can be already resident in the terminal's non-volatile memory. The terminal (118) generates the random number $X_T(1<X_T<q-1)$, and computes the value of $Y_T=\alpha^{X}{}_T$ mod q. The GAN (e.g., the RNC or base station) generates the random number $X_G(1<X_G<q-1)$, and computes the value of $Y_G=\alpha^{X}{}_G$ mod q. The random numbers can be generated at the terminal using the methods described above with respect to generating naturally occurring, truly random numbers.

Y_T and Y_G are transferred unencrypted to the respective GAN and terminal. Upon receipt of the number Y_G, the terminal calculates the value of $K_S=Y_G{}^{X}{}_T$ mod $q=\alpha^{X}{}_G{}^{X}{}_T$ mod q. Upon receipt of the number Y_T, the GAN calculates the value of $K_S=Y_T{}^{X}{}_G$ mod $q=\alpha^{X}{}_T{}^{X}{}_G$ mod q. The number X_T is kept secret at the terminal, the number X_G is kept secret at the GAN, but the value of K_S is now known at both the terminal and the GAN. The number K_S is therefore used by both as the communications session encryption key. An unauthorized user would not know either X_T or X_G and would have to compute the key K_S from Y_T and Y_G, which is a prohibitive computational process. A significant security advantage of using the exponential key exchange algorithm is that the GAN is not required to maintain secret private key data on a permanent basis.

In summary, when a communications session is first initiated between a GAN and a terminal, the terminal receives an asymmetric public key that has been continuously broadcast by the GAN, retrieved from the terminal's internal memory, or requested from the GAN. The GAN maintains a private key that can be used to decrypt information encrypted with the public key. The terminal generates and stores a naturally occurring random number as a secret session (symmetric) key, encrypts the symmetric session key with the public key, and transmits the encrypted session key to the GAN. The GAN decrypts the session key with the private key,

and both the GAN and terminal encrypt the ensuing communications with the secret session key. A primary technical advantage of transferring a public key from a GAN to a terminal at the onset of communications is that the GAN is not required to know the identity of the terminal in order to have encrypted communications with the terminal. However, a problem can arise if an unauthorized user attempts to impersonate a GAN and transmits a public key to the terminal. In that event, as described below, the terminal can be configured to authenticate the received public key and the identity of the GAN.

For example, when a public key is to be transferred from a GAN to a terminal, the key can be transferred with a public key "certificate". This certificate provides proof that the associated public key and the owner of that key are authentic. A "trusted" third party can issue the public key along with the certificate, which includes a "digital signature" that authenticates the third party's identity and the public key. The certificate can also contain the GAN's identity and the expiration date of the certificate, if any.

In one aspect of the invention, the GAN transmits the certificate and public key to the terminal. In that case, the public key of the third party is pre-stored (a priori) at the subscribing terminals.

FIGURE 5 is a block diagram of a method that can be used to certify the authenticity of a public key and the owner of the key with a digital signature, in accordance with the present invention. The method (300) of digitally signing a public key certificate and verifying its authenticity begins at step 302. At step 302, a "certificate" containing unencrypted information about the owner of the public key to be transferred to a terminal is prepared by a trusted third party. The unencrypted information also includes the public key and the expiration date of the certificate. At step 304, the resulting "unsigned" certificate is processed with an irreversible algorithm (e.g., a hashing algorithm) to produce a message digest at step 306, which is a digested or shortened version of the information included on the certificate. At step 308, the digest information is encrypted with a private key of a different public/private key pair. Preferably, an RSA algorithm similar to equations 1 and 2 above is used to derive this key pair. At step 310, a digitally signed public key

certificate is thereby produced that contains the originally unencrypted information (including the public key to be used for the communications session) and the digest information, which is now encrypted with the certificate issuer's private key. The digitally signed public key certificate is then transferred to the terminal that has initiated contact with the GAN.

At step 312, upon receiving the digitally signed certificate, the terminal's processor analyzes the unencrypted and encrypted portions of the document. At step 314, the unencrypted information is processed using an algorithm identical to the hashing algorithm used at step 304. At step 316, a second digested version of the unencrypted information is produced at the terminal. At step 318, the terminal's processor retrieves the pre-stored certificate issuer's public key from memory, and using an RSA algorithm, decrypts the encrypted digest information from the certificate. Another version of the unencrypted digested information is thereby produced at step 320. At step 322, the terminal compares the two versions of the unencrypted digested information, and if the compared information is identical, the certificate's signature and the session public key are assumed to be authentic. That certified public key can now be used by the terminal to encrypt the secret session key.

Although a preferred embodiment of the method and apparatus of the present invention has been illustrated in the accompanying Drawings and described in the foregoing Detailed Description, it will be understood that the invention is not limited to the embodiments disclosed, but is capable of numerous rearrangements, modifications and substitutions without departing from the spirit of the invention as set forth and defined by the following claims.

WHAT IS CLAIMED IS:

1. A method for encrypting communications traffic between a mobile communications network and a communications terminal, comprising the steps of:

storing a public key and a first identifier associated with said mobile communications network at said communications terminal;

comparing said first identifier stored at said communications terminal with a second identifier received from said mobile communications network and producing a first predetermined result;

generating a secret key at said communications terminal;

encrypting said secret key with said stored public key at said communications terminal; and

transmitting said encrypted secret key from said communications terminal.

2. The method according to Claim 1, further comprising the steps of:

receiving said encrypted secret key at said mobile communications network;

decrypting said received encrypted secret key with a private key, said private key associated with said public key; and

encrypting said communications traffic with said secret key.

3. The method according to Claim 1, wherein the step of storing a public key comprises the step of a priori pre-storing the public key.

4. The method according to Claim 1, further comprising the step of transmitting said public key from said mobile communications network upon receiving a public key request from said communications terminal.

5. The method according to Claim 4, wherein the step of transmitting said public key further comprises the step of transmitting information to authenticate said public key.

6. The method according to Claim 4, further comprising the step of transmitting said request from said communications terminal upon said comparing step producing a second predetermined result.

7. The method according to Claim 1, wherein the steps of receiving and decrypting said encrypted secret key are performed at a radio base station in said mobile communications network.

8. The method according to Claim 1, wherein the step of decrypting said received encrypted secret key is performed at a radio network controller in said mobile communications network.

9. The method according to Claim 1, wherein said mobile communications network comprises a generic communications network.

10. The method according to Claim 1, wherein said communications terminal comprises a mobile terminal.

11. The method according to Claim 1, wherein said communications terminal comprises a fixed terminal.

12. The method according to Claim 1, wherein said communications terminal comprises an unidentified communications terminal.

13. The method according to Claim 1, wherein said mobile communications network comprises a cellular phone network.

14. The method according to Claim 1, further comprising the steps of:

connecting a plurality of service networks to said mobile communications network, a user of said communications terminal being a subscriber to at least one of said plurality of service networks; and

providing a communications path between said communications terminal and said at least one of said plurality of service networks.

15. The method according to Claim 1, wherein said private key and said public key are associated by an RSA Algorithm.

16. The method according to Claim 1, wherein said secret key comprises a symmetric encryption key.

17. The method according to Claim 1, wherein the step of generating a secret key comprises the step of generating a naturally occurring random number.

18. The method according to Claim 1, wherein the step of generating a secret key comprises the steps of:

detecting a received signal in digital form at said communications terminal; and

extracting at least one low order bit from said detected received signal.

19. The method according to Claim 1, wherein the step of generating a secret key comprises the steps of:

detecting a signal at an output of a microphone A/D converter; and

extracting at least one low order bit from said detected output signal.

20. The method according to Claim 1, wherein the step of generating a secret key comprises the steps of:

detecting a signal at an output of a speech codec; and

extracting at least one low order bit from said detected output signal.

21. The method according to Claim 1, wherein the step of generating a secret key comprises the steps of:

generating a seed for a pseudorandom number; and

generating a pseudorandom number from said seed.

22. The method according to Claim 1, wherein a length of said secret key is predetermined at said communications terminal.

23. The method according to Claim 1, wherein said secret key further comprises a plurality of concatenated numbers.

24. The method according to Claim 1, wherein the step of storing said public key and said first identifier further comprises storing an expiration date associated with said public key.

25. The method according to Claim 24, wherein said communications terminal transmits a public key request to said mobile communications network if said public key has expired.

26. The method according to Claim 1, further comprising the steps of:

changing said public key at said mobile communications network; and

storing said changed public key at said communications terminal.

27. The method according to Claim 26, wherein the step of changing said public key further comprises the step of broadcasting said changed public key from said mobile communications network for a predetermined period of time.

28. A method for encrypting traffic between a generic communications network and a first communications terminal, comprising the steps of:

broadcasting a public key from said generic communications network to a plurality of communications terminals, said plurality of communications terminals including said first communications terminal;

generating a secret key at said first communications terminal;

encrypting said secret key with said public key at said first communications terminal;

transmitting said encrypted secret key from said first communications terminal;

receiving said encrypted secret key at said generic communications network;

decrypting said received encrypted secret key with a private key, said private key associated with said public key; and

encrypting said traffic with said secret key.

29. The method according to Claim 28, wherein the broadcasting step further comprises the steps of:

transferring said public key from a radio network controller to at least one base station in said generic communications network; and

transmitting said public key from said at least one base station.

30. The method according to Claim 28, wherein said broadcasting step comprises the step of transmitting said public key from a plurality of base stations in said generic communications network.

31. The method according to Claim 28, wherein said first communications terminal comprises an unidentified communications terminal.

32. The method according to Claim 28, wherein the step of broadcasting said public key further comprises the step of broadcasting information to authenticate said public key.

33. The method according to Claim 28, wherein the step of broadcasting said public key further comprises the step of transmitting, on request, information to authenticate said public key.

34. A method for encrypting communications traffic between a mobile communications network and a communications terminal, comprising the steps of:

storing two numbers associated with a Diffie-Hellman exponential key exchange algorithm and a first identifier associated with said mobile communications network at said communications terminal;

comparing said first identifier stored at said communications terminal with a second identifier received from said mobile communications network and producing a first predetermined result;

generating a first random number at said communications terminal;

generating a second random number at said mobile communications network; and

using said first and second random numbers as inputs to said Diffie-Hellman exponential key exchange algorithm, generating a third number to be used as a secret key by said communications terminal and said mobile communications network.

35. The method according to Claim 34, wherein the step of storing two numbers comprises the step of a priori pre-storing said two numbers.

36. The method according to Claim 34, further comprising the step of transmitting said two numbers from said mobile communications network upon receiving a request for said two numbers from said communications terminal.

37. The method according to Claim 36, further comprising the step of transmitting said request from said communications terminal upon said comparing step producing a second predetermined result.

38. The method according to Claim 34, wherein the step of storing said two numbers and said first identifier further comprises storing an expiration date associated with said two numbers.

39. The method according to Claim 38, wherein said communications terminal transmits a request for two new numbers associated with said Diffie-Hellman exponential key exchange algorithm if said two numbers has expired.

40. The method according to Claim 34, further comprising the steps of:

changing said two numbers associated with a Diffie-Hellman exponential key exchange algorithm at said mobile communications network; and

storing said changed two numbers at said communications terminal.

41. The method according to Claim 40, wherein the step of changing said two numbers further comprises the step of broadcasting said changed two numbers from said mobile communications network for a predetermined period of time.

42. A method for encrypting traffic between a generic communications network and a first communications terminal, comprising the steps of:

broadcasting two numbers associated with an exponential key exchange algorithm from said generic communications network to a plurality of communications terminals, said plurality of communications terminals including said first communications terminal;

generating a first random number at said first communications terminal;

generating a second random number at said generic communications network;

using said first and second random numbers as inputs to said exponential key exchange algorithm, generating a third number to be used as a secret key by said first communications terminal and said generic communications network;

and encrypting said traffic with said secret key.

43. A system for use in encrypting traffic between a generic communications network and a communications terminal, comprising:

an access network included in said generic communications network; and

access network means coupled to said communications terminal and associated with said access network, for storing a public encryption key associated

with said generic communications network, generating a secret key, encrypting said secret key with said stored public encryption key, and transmitting said encrypted secret key to said generic communications network.

44. A system for use in encrypting traffic between a generic communications network and a communications terminal, comprising:

first network means for storing a private encryption key, distributing a public encryption key, and decrypting an encrypted secret session key;

second network means connected to said first network means, for broadcasting said distributed public encryption key, said first and second network means associated with an access network of said generic communications network; and

access network means coupled to said communications terminal and associated with said access network of said generic communications network, for receiving said broadcast public encryption key, generating a secret key, encrypting said secret key with said received public encryption key, and transmitting said encrypted secret key to said generic communications network.

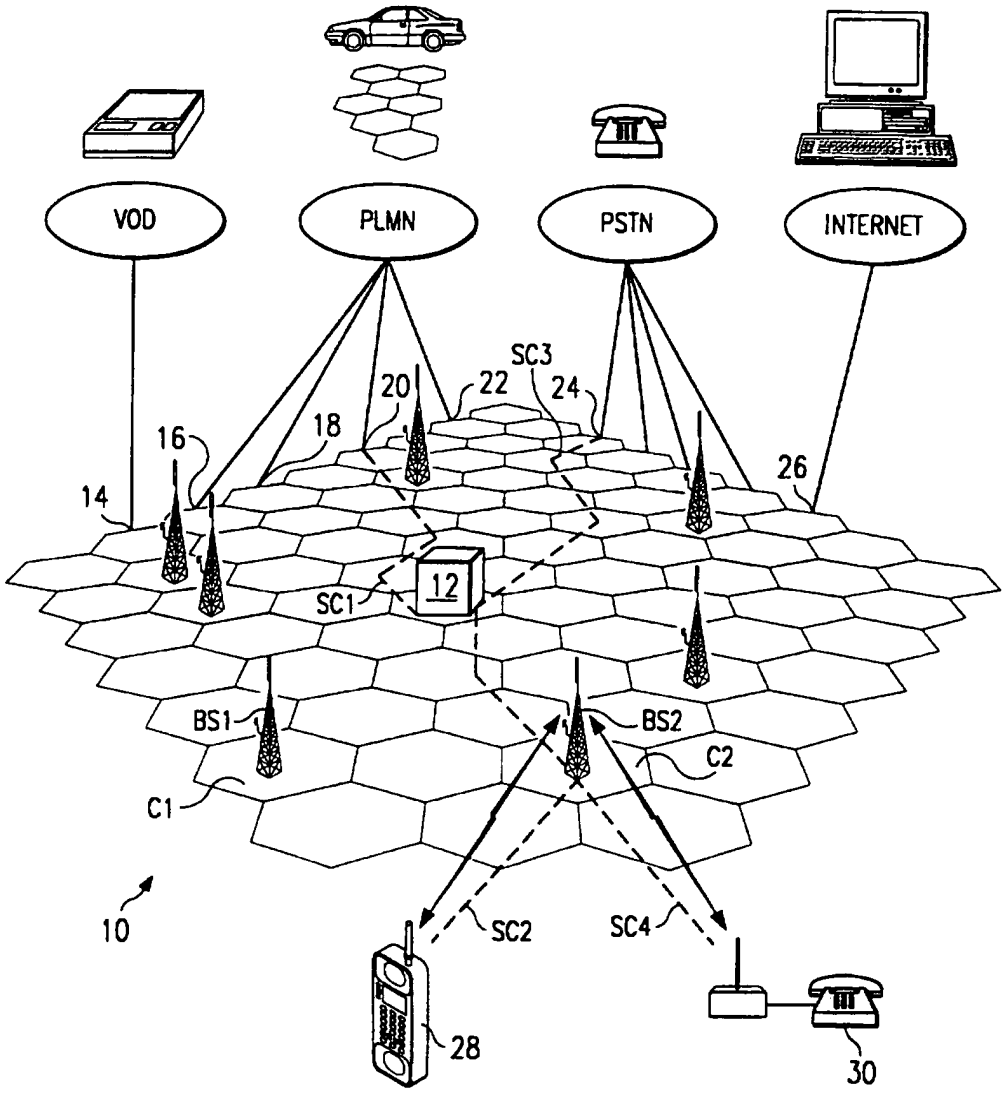

VOD
PLMN
PSTN
INTERNET
SC3
20
22
24
18
16
26
14
12
SC1
BS1
BS2
C2
C1
10
SC2
SC4
28
30

FIG. 1

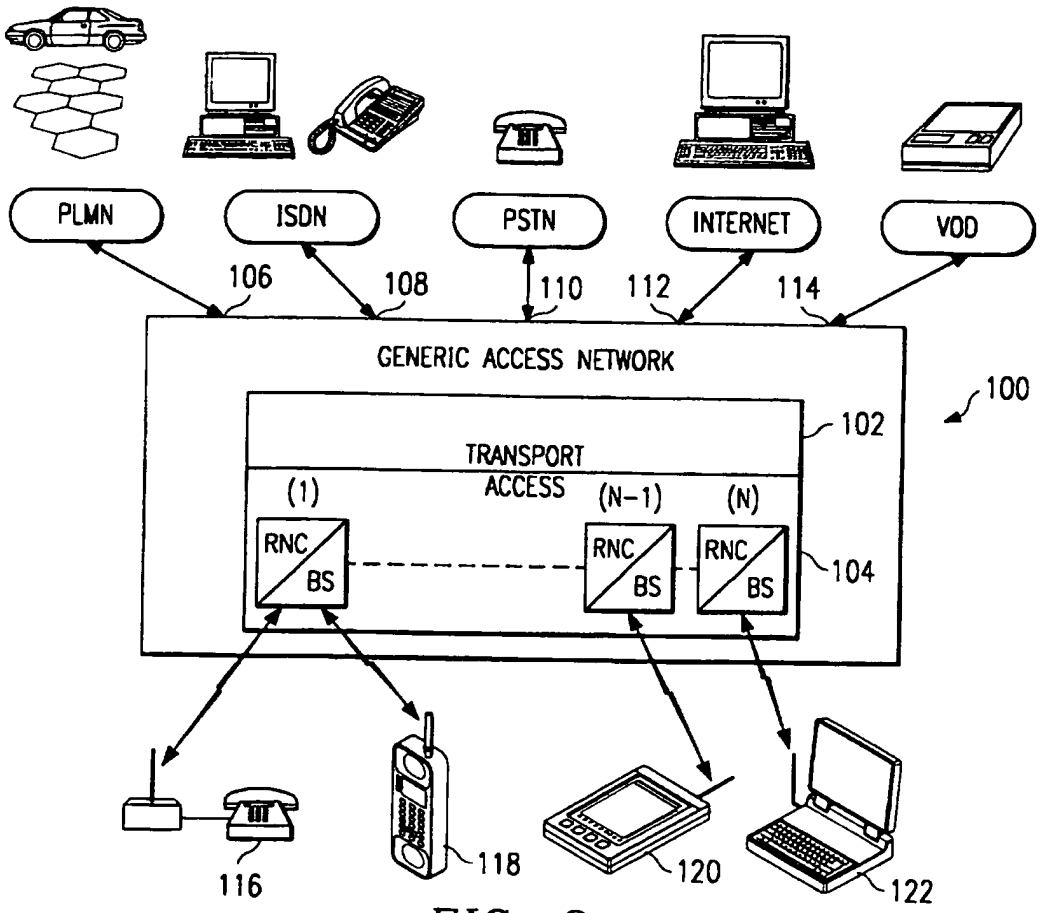

PLMN
ISDN
PSTN
INTERNET
VOD
106
108
110
112
114
GENERIC ACCESS NETWORK
100
102
TRANSPORT ACCESS
(1)
(N-1)
(N)
RNC
BS
104
116
118
120
122

FIG. 2

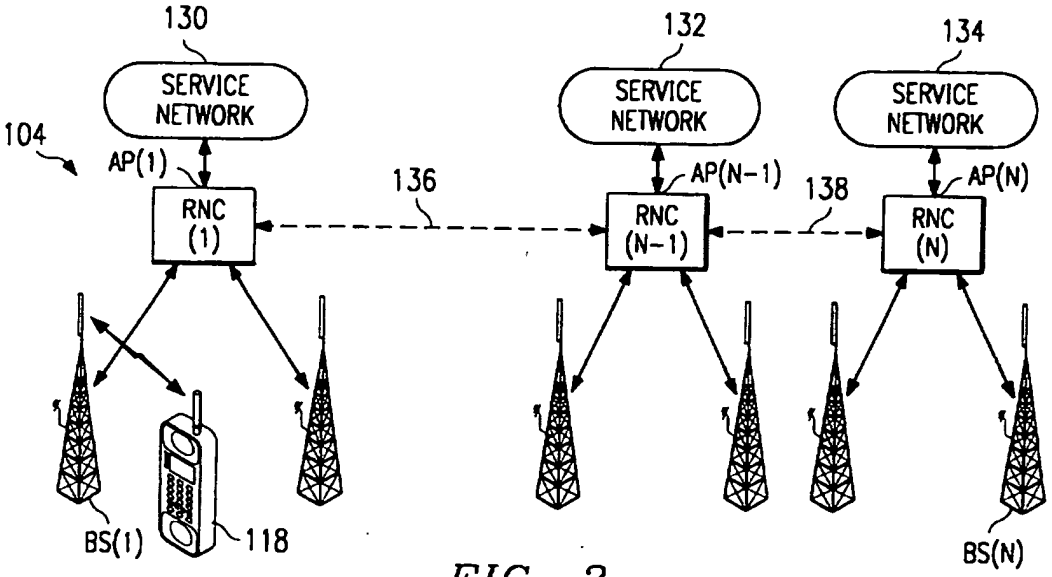

130
132
134
SERVICE NETWORK
104
AP(1)
AP(N-1)
AP(N)
136
138
RNC (1)
RNC (N-1)
RNC (N)
BS(1)
118
BS(N)

FIG. 3

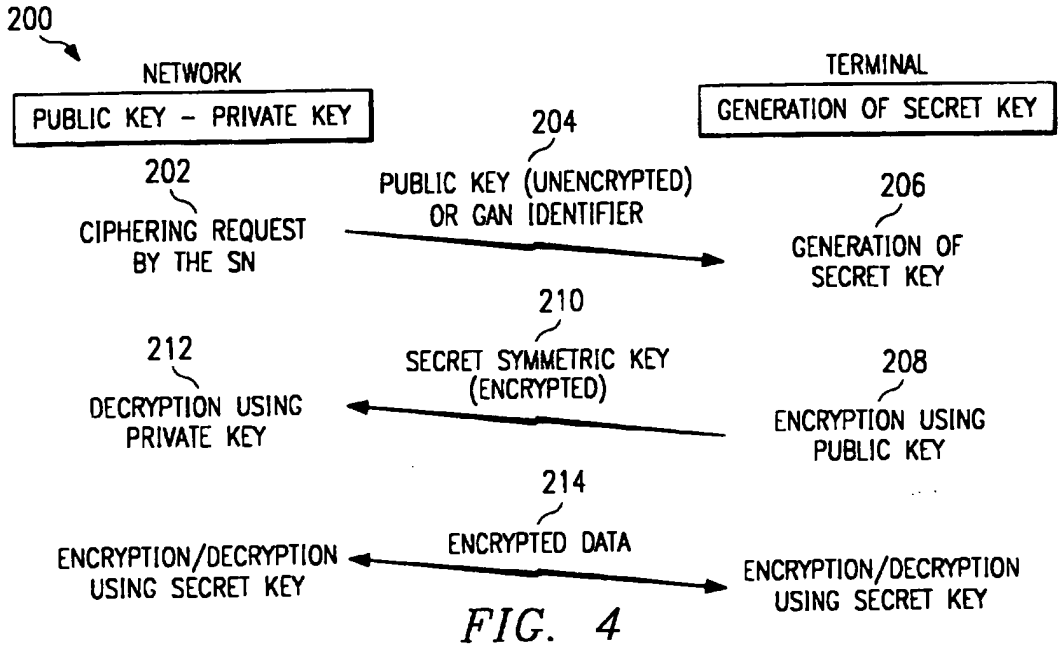

200
NETWORK
PUBLIC KEY - PRIVATE KEY
TERMINAL
GENERATION OF SECRET KEY
204
202
PUBLIC KEY (UNENCRYPTED) OR CAN IDENTIFIER
206
CIPHERING REQUEST BY THE SN
GENERATION OF SECRET KEY
210
212
SECRET SYMMETRIC KEY (ENCRYPTED)
208
DECRYPTION USING PRIVATE KEY
ENCRYPTION USING PUBLIC KEY
214
ENCRYPTED DATA
ENCRYPTION/DECRYPTION USING SECRET KEY
ENCRYPTION/DECRYPTION USING SECRET KEY

FIG. 4

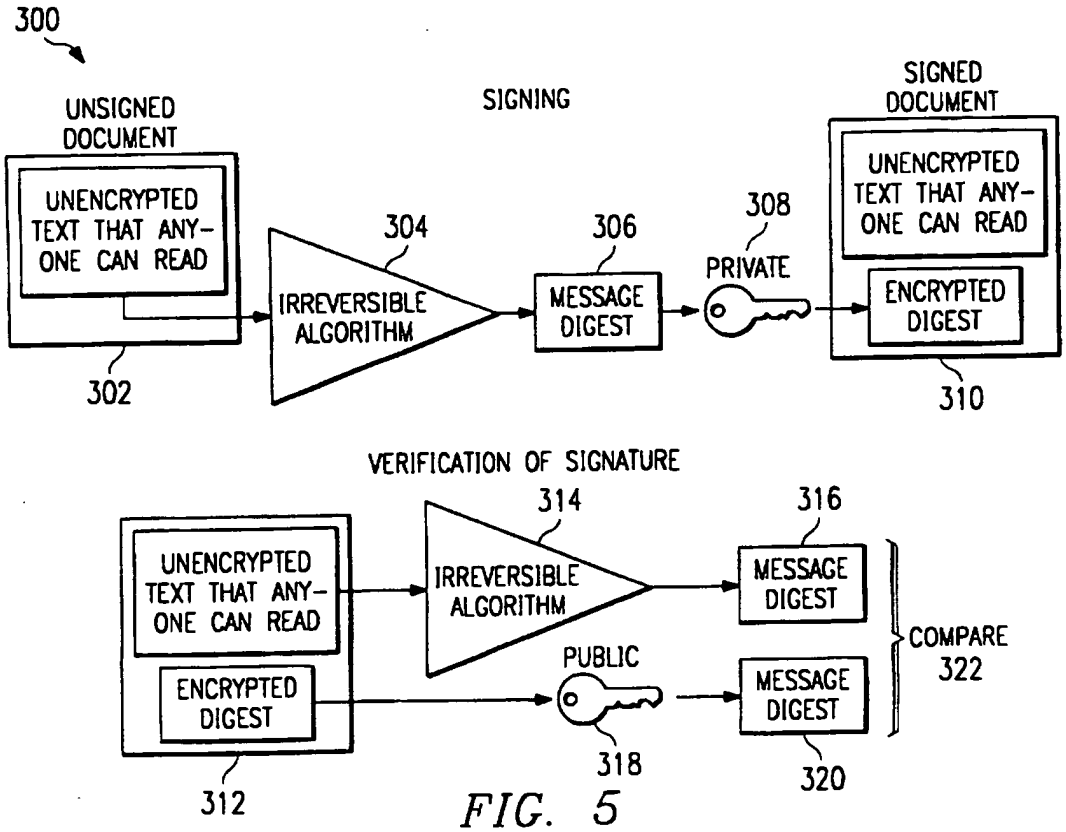

300
UNSIGNED DOCUMENT
SIGNING
SIGNED DOCUMENT
UNENCRYPTED TEXT THAT ANY-ONE CAN READ
304
306
308
PRIVATE
UNENCRYPTED TEXT THAT ANY-ONE CAN READ
IRREVERSIBLE ALGORITHM
MESSAGE DIGEST
ENCRYPTED DIGEST
302
310
VERIFICATION OF SIGNATURE
314
316
UNENCRYPTED TEXT THAT ANY-ONE CAN READ
IRREVERSIBLE ALGORITHM
MESSAGE DIGEST
COMPARE
322
PUBLIC
ENCRYPTED DIGEST
MESSAGE DIGEST
318
312
320

FIG. 5

INTERNATIONAL SEARCH REPORT

International Application No
PCT/SE 97/01407

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L9/08 H0407/32

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)
IPC 6 H04L H04Q

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 222 140 A (BELLER ET AL.) 22 June 1993 see column 4, line 57 - column 5, line 3 see column 5, line 13 - line 37 ---	1,2,7, 10,28, 29,34, 42-44
A	GB 2 297 016 A (KOKUSAI DENSHIN DENWA) 17 July 1996 see page 19, line 11 - page 21, line 2; figure 7 --- -/--	28,44

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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
19 November 1997	02/12/1997

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Holper, G

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/SE 97/01407

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MEVEL F ET AL: "Distributed communication services in the Masix system" CONFERENCE PROCEEDINGS OF THE 1996 IEEE FIFTEENTH ANNUAL INTERNATIONAL PHOENIX CONFERENCE ON COMPUTERS AND COMMUNICATIONS (CAT. NO.96CH35917), CONFERENCE PROCEEDINGS OF THE 1996 IEEE FIFTEENTH ANNUAL INTERNATIONAL PHOENIX CONFERENCE ON COMPUTERS AND, ISBN 0-7803-3255-5, 1996, NEW YORK, NY, USA, IEEE, USA, pages 172-178, XP000594787 see page 174, right-hand column, line 25 - line 29 see page 176, right-hand column, line 26 - page 177, left-hand column, last line; figure 5 ---	28,43,44
A	PATENT ABSTRACTS OF JAPAN vol. 95, no. 008 & JP 07 203540 A (N T T IDOU TSUUSHINMOU KK), 4 August 1995, see abstract ---	1,34
A	EP 0 067 977 A (SIEMENS) 29 December 1982 see abstract; figure 2 -----	24

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/SE 97/01407

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5222140 A	22-06-93	NONE	
GB 2297016 A	17-07-96	JP 8195741 A	30-07-96
EP 67977 A	29-12-82	DE 3123167 C	24-02-83

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00	A1	(11) International Publication Number: WO 98/45768
		(43) International Publication Date: 15 October 1998 (15.10.98)

(21) International Application Number: PCT/CA98/00241

(22) International Filing Date: 18 March 1998 (18.03.98)

(30) Priority Data:
08/831,696 10 April 1997 (10.04.97) US

(71) Applicant: NORTHERN TELECOM LIMITED [CA/CA]; Station A, P.O. Box 6123, Montreal, Quebec H3C 3J5 (CA).

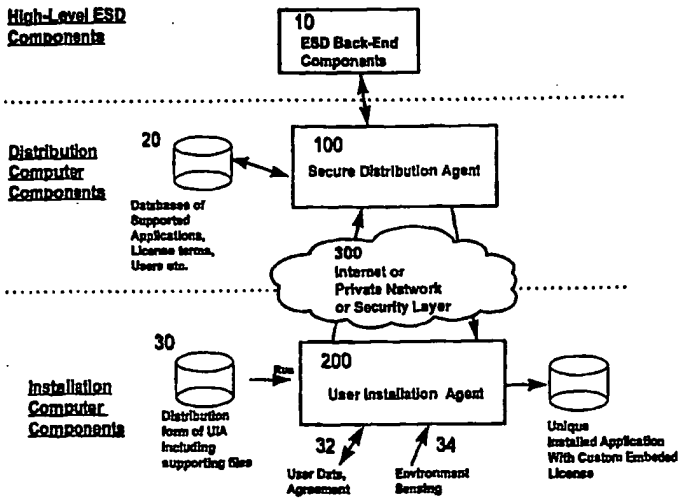
(72) Inventors: LAROSE, Gordon, Edward; 2417 Baseline Road, Ottawa, Ontario K2C 0E3 (CA). ALLAN, David, Ian; 852 Forest Street, Ottawa, Ontario K2B 5P9 (CA).

(74) Agents: MCGRAW, James et al.; Smart & Biggar, 900–55 Metcalfe Street, P.O. Box 2999, Station D, Ottawa, Ontario K1P 5Y6 (CA).

(81) Designated States: AU, CA, CN, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: METHOD AND SYSTEM FOR NETWORKED INSTALLATION OF UNIQUELY CUSTOMIZED, AUTHENTICABLE, AND TRACEABLE SOFTWARE APPLICATIONS



(57) Abstract

A method to create, distribute and install on an installation computer a uniquely customised instance of a software application that is authenticable and traceable to a particular user. A secure distribution agent resident on a distribution computer collects indentifying information, and calculates a cryptographic signature of the software application and identifying information. The identifying information and cryptographic signature are embedded in the software application by the secure distribution agent. The software application with embedded data is transmitted via a distribution channel to the installation computer. A user installation agent resident on the installation computer manages the installation of the software application with embedded data on the installation computer. Prior to installation, the user installation agent may use the cryptographic signature to verify that the software application, and the indentifying information, are authentic and have not been tampered with.

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.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND SYSTEM FOR NETWORKED INSTALLATION OF UNIQUELY CUSTOMIZED, AUTHENTICABLE, AND TRACEABLE SOFTWARE APPLICATIONS

FIELD OF THE INVENTION

This invention relates to a method and system for the electronic distribution and installation to users via a network of software applications that are uniquely customized, authenticable and traceable to the individual user.

BACKGROUND OF THE INVENTION

With the increasing importance and reliance on networked computer environments such as the Internet, Electronic Software Distribution (ESD) is assuming an increased importance as a means of distributing software applications to users. The on-line infrastructures currently in place enable users to purchase and install software applications without the need for physical delivery of shrink-wrapped software. Typically, a software publisher will prepare a master of the software application for electronic distribution. A customer will then go on-line and submit an order to purchase the software application, which will be received and fulfilled by the publisher. The customer will then download the software application and install it to his/her own computer.

A disadvantage of the current on-line infrastructure is that it delivers software applications to users in a form that is identical with those found in retail stores and catalogues. Absent cryptographic protection, users can freely share the distribution form of the software amongst themselves.

Even where cryptographic protection are present, the potential for unauthorized copying is still significant because all the users possess identical copies (necessarily having identical encryption schemes) of a software application. There is in all such cases a single underlying decryption key, and in most cases this key, or an equivalent variant of it, is entered by the user, who can then share it with other users who can use it to obtain unlicensed usage of the program. There exist today bulletin boards and Internet sites devoted to the sharing of such keys, which are visited by persons interested in

obtaining unpaid for usage of programs by applying such keys to copies of the applications they have obtained.

Further, even where more subtle anti-piracy schemes are in place in a software application, it is not uncommon for software "hackers" to produce "crack" programs which can be used to process a freely-distributed, limited functionality version of a program to produce a revised, fully-functional version of the same program which can be used without purchasing a license. Even the most ingenious forms of single-key mass distribution, which might involve input of one-time-only responses to a dynamic challenge to infer the key, are vulnerable to a "crack" which simply causes the application of the "true" universal decryption key. Although such "crack" involves more technical sophistication than sharing of keys as above, the distribution channels and potential effect on the product's revenues are very similar.

In addition, software applications distributed by conventional ESD techniques provide no means to police their own integrity to prevent unauthorized tampering.

Portland Software has produced an electronic software distribution system sold under the trade-mark ZipLock™ that packages software for electronic distribution over the Internet. The ZipLock™ system discloses a system that distributes, from a secure server to a client resident on the user's computer, a standard executable software application that is protected by means of a cryptographic key. Data input by the users is transmitted to the secure server and is used to construct a customized digital licence certificate that is transmitted to the user in a separate computer file. The Ziplock™ system does not provide a mechanism to detect tampering done to the executable software application itself, nor does it provide traceability if the digital licence certificate is not included with an unauthorized redistribution of the software application.

The prior art discloses a number of other systems and methods to protect unauthorized use of software electronically distributed to users. In Choudhury U.S. Pat. No. 5,509,074,

there is disclosed a method of protecting electronically published materials using cryptographic protocols. A first described embodiment requires special purpose hardware to decrypt the document that is transmitted to the user. This eliminates the method from general use with personal computers used by the general public. In a second method, there is no requirement for special purpose hardware. In this method, the publisher modifies the inter-line or inter-word spaces of the document to make each document unique for each user. The unique document is then encrypted and transmitted to the user's computer. Upon receipt of the encrypted document, the user's computer will prompt the user to enter his/her secret key which is used to decrypt the document for viewing. The method disclosed by this reference does not prevent piracy, it only discourages piracy by making the pirated document traceable to the user. In addition, this reference pertains only to data files, not to the protection of executable files of any type.

In Cane U.S. Pat No. 5,416,840, there is disclosed a method and system for protecting computer program distribution in a broadcast medium, such as radio frequency public broadcast or computer network. In this reference, the method involves encrypting at least a portion of a computer program, the user being supplied with a password for use in decrypting the computer program so that the computer program can be installed and used. A unique password is generated and transmitted to the user for subsequent use in decrypting the selected software program contained on the medium. While there is disclosed a method and system for the generation, transmission and use of unique passwords that cannot be shared among different users of the software application, this reference requires the user to own proprietary hardware that eliminates it from general use with personal computer owned by the general public.

In Yuval U.S. Pat No. 5,586,186, there is disclosed a method and system for controlling unauthorized access to software distributed to users. The main components of the system are an encryptor, a user key generator, and a decryptor. The encryptor generates encryption and decryption keys,

encrypts the software using the encryption keys, and stores the encrypted forms of the software of the broadcast medium, such as CD ROM. The user key generator generates a unique key using numeric representations of identifying information supplied by users and the decryption keys. The decryptor is responsible for decrypting the encrypted forms of the software using the identifying information supplied by the user, and the unique user keys. The decryption method disclosed by this reference enables a large number of different but logically similar keys to be used as decryption keys, each of which is unique to a particular user. However, this reference does not disclose a means to customize a software application with user-specific data such that the software application itself can be authenticated. Furthermore, this reference does not prevent piracy by sharing of keys; it only discourages it through traceability of keys.

SUMMARY OF THE INVENTION

The present invention pertains to a method for the electronic distribution of a software application from a distribution computer to an installation computer comprising the steps of receiving at said distribution computer identifying information, embedding said identifying information in said software application at said distribution computer to form an identifiable software application, generating a cryptographic signature for said identifiable software application, embedding said cryptographic signature in said identifiable software application to form an identifiable and authenticable software application, and transferring said identifiable and authenticable software application from said distribution computer to said installation computer.

The method and system of the present invention discloses an on-line software customization, delivery and installation scheme. Instead of distributing a software application to a user that results in the installation of a totally generic, untraceable executable file on the installation computer, the method and system disclosed herein discloses a means to create, distribute and install on an

installation computer a uniquely customised instance of a software application that is authenticable and traceable to a particular user.

The method and system disclosed herein provides for a user installation agent (UIA) resident on an installation computer to establish a connection through a distribution channel to a secure distribution agent (SDA) resident on a distribution computer. The UIA and/or SDA prompt the user to input identifying information that, together with business related information such as licensing terms, etc., is used to create a unique data set that is embedded in the desired software application by the SDA. By the use of a cryptographic hash algorithm, and private/public key cryptography wherein a private key is only known to the SDA, a cryptographic signature of the desired software application and embedded data set is calculated and also embedded into the software application. The software application with embedded data and cryptographic signature is transmitted via a distribution channel to the installation computer where it is installed on the installation computer. Optionally, the installation computer may use the cryptographic signature to verify that neither the software application, nor the embedded data have been tampered with. Public key(s) used to decrypt the cryptographic signatures may be transmitted to the installation computer with the software application, or by any other means, such as e-mail, Internet bulletin boards, etc. Following installation, the embedded data and cryptographic signature are used in a variety of ways, such as to provide a means to trace the software application to the user, to police the continued integrity of the software application, to ensure that license conditions continue to be met, to perform virus checking, or automatic upgrading of the software application itself.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a system overview showing the various inputs and components of the system and method of the present invention;

Figure 2 is a data flow diagram of the structure and operation of the Secure Distribution Agent employed by the present invention;

Figure 3A is a block diagram showing details of the construction of the aggregate distribution file using a one-step cryptographic process;

Figure 3B is a block diagram showing details of the construction of an aggregate distribution file using a two-step cryptographic process;

Figure 3C is a block diagram showing details of the construction of an aggregate distribution file using a cryptographic process that is a variant of the two-step cryptographic process shown in Figure 3B;

Figure 4 is a block diagram of the structure and operation of the User Installation Agent employed by the present invention;

Figure 5 is a block diagram showing the means of extracting and authenticating embedded data from an installed distribution file;

Figure 6 is a flow chart of a first embodiment of the present invention that authenticates embedded data by means of a common encryption key;

Figure 7 is a flow chart of a second embodiment of the present invention that authenticates embedded data by means of a unique per-user encryption key; and,

Figure 8 is a block diagram showing the various uses of the installed software application delivered to the user by means of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 shows the various inputs and components of the system and method of the present invention. At the top level is shown a representation of the Electronic Software Distribution (ESD) back-end components 10, which include clearinghouses of software, software manufacturers, publishers, credit card servers, etc., all of which interact with a Secure Distribution Agent (SDA) 100 resident on a distribution computer that forms an essential part of the present invention.

The SDA 100 interfaces with these ESD back-end components 10 via the Internet or private computer network to provide payment methods support, loading of software applications from publishers, etc. The exact nature of the ESD back-end components 10 may vary without affecting the method and system of the present invention.

The SDA 100 is comprised of a system of co-operating software programs, which run in a secure environment. The nature of the secure environment is immaterial to the invention as long as it ensures the ability to protect the privacy of user data, authentication of users and possibly other third-parties, and suitable limitations on the operations which can be accessed externally. This environment might or might not be physically separated from an installation computer. The structure and operation of the SDA 100 is more fully described in Figure 2.

One of the inputs to the SDA 100 is a set of databases 20 of supported software applications, license terms, licensed users, etc. The SDA 100 transmits and receives relevant data to/from the databases 20 prior to, and during the operation of the present invention. The exact nature and content of the databases 20 is not an essential feature of the invention.

A distribution channel 300 is illustrated in Figure 1 that can comprise computer networks such as the Internet, or private network, or a security layer as required to maintain security if the SDA 100 were located in close proximity with a user installation agent (UIA) 200. Alternatively, it may contain some combination of these elements. The distribution channel 300 is used to connect the UIA 200 to the SDA 100 (and thus connect the distribution computer to the installation computer) so that information may be exchanged between these two agents, and so that an aggregate distribution file 170 (shown in Figure 2) can be distributed from the SDA 100 to the UIA 200. Though the distribution channel 300 is illustrated between the SDA 100 and the UIA 200, the system of the present

invention does not require that the SDA 100 be physically distant from the UIA 200.

At a user's end is the UIA 200 which is an installation/automatic upgrade software program resident on the installation computer. This program is used to communicate via the distribution channel 300 to the SDA 100, and to perform the required operations, more fully described below, on the installation computer. Though normally one UIA 200 would be required for each supported software application, persons skilled in the art would be familiar with the capability to develop a UIA 200 that would support multiple software applications. Also shown in Figure 1 is a distribution form 30 of the UIA 200, including support files. The nature of the distribution form 30 of the UIA 200 is immaterial to the operation of the present invention. Any of CD ROM, World Wide Web (WWW) download, floppy diskette, etc. could be used.

The UIA 200 accepts data 32 input from the user, such as name, address, payment options, etc., as well as data pertaining to the acceptance of an end-user license. Environment sensing data 34 such as speed of CPU, size of hard disk, speed of modem, etc. may also be input to the UIA 200 for processing. The identifying information processed by the UIA 200 may include any information pertaining to the purchaser, the seller, the installation agent, date, serial number, license specifics, etc. This data may be used for the automatic registration of the desired software application with a publisher or its commercial proxies.

As noted above, the identifying data 32, 34 constitutes identifying information concerning the user, its computer, etc. The identifying data 32, 34 is processed by the UIA 200 and transmitted to the SDA 100 via the distribution channel 300. Of course, it is understood that the identifying information does not necessarily have to be transmitted to the SDA 100 by means of the distribution channel 300. For example, the identifying information may be locally entered into the SDA 100 by an agent using information received verbally, in writing, or in some other non-electronic manner. The SDA 100

combines the identifying data 32, 34 with the data stored in the databases 20 to produce an aggregated distribution file 170 that is uniquely customized, authenticable, and traceable to the user. The aggregate distribution file 170 is transmitted via the distribution channel 300 to the UIA 200. The output from the UIA 200 is a uniquely customized software application 15 (to be referred to below as an "installed aggregate distribution file") installed on the installation computer, with identifying information embedded therein.

Though the description of the present invention implies that the "user" is an individual user of the software application 15 to be installed on a personal computer, persons skilled in the art will appreciate that the present invention would also operate in the context of a networked end-user environment, where the "user" was a network administrator responsible for installing software on a central server for use by a number of end users.

Figure 2 is a data flow diagram of the structure and operation of the SDA 100 employed by the present invention. An original distribution file 130 is shown as an input to a conversion program 110. In the envisioned implementation, the original distribution file 130 is input to the SDA 100 by the databases 20 shown in Figure 1. It is understood that the original distribution file 130 does not necessarily have to be input to the SDA 100 by the databases 20, since the original distribution file 130 may already be resident on the distribution computer containing the SDA 100. The conversion program 110 has, as additional inputs, the data 140 to be embedded in the distribution file 130, and required public/private cryptographic key pairs 150. The embedded data 140 is produced by a user interaction program 120 which interacts with the user through the UIA 200 to receive identifying data 32, 34 (shown in Figure 1) as well as data from the databases 20 of supported software applications, license terms, licensed users, etc.

While the embedded data 140 can be of any form and content, it is anticipated that the embedded data 140 will

contain information enabling the software application 15 to be traceable to an individual user and license transaction. For example, the embedded data 140 can include a unique serial number used to identify the aggregate distribution file 170 to be distributed to the user. The would eliminate serial number fraud that is common in the software industry, whereby current software applications can only perform simple validity checks, which can be fooled by widespread fraudulent re-use of a single valid serial number. The embedded data 140 may take the form of a complete license agreement customized to the individual user, including user name, address, software serial no., license terms, etc. Records of the user information collected by the user interaction program 120 may be kept by the databases 20.

The output of the conversion program 110 is an aggregate distribution file 170 which contains both the contents of the original distribution file 130, the embedded data 140, as well as a cryptographic signature of the embedded data 140 and the original distribution file 130. The aggregate distribution file 170 is then transmitted via the distribution channel 300 to the UIA 200. The UIA 200 then installs the aggregate distribution file 170 on the installation computer. Once the aggregate distribution file 170 is installed, it takes the form of an installed aggregate distribution file 15.

By means of its connection with the UIA 200, the SDA 100 can negotiate arbitrary license terms with the user, display an End-User License Agreement (EULA), confirm acceptance of that agreement, and automatically perform on-line registration of the software based on the already-established identity of the user and the specific license terms. Subject to commercial and legal considerations, an SDA 100 could offer different pricing and license terms, and possibly different executable versions, to users in different countries, for example. In addition, differential pricing based on attributes of the installation computer such as CPU power could be provided.

The SDA 100 does not require intelligence within itself for functions such as establishing that a user's stated address and credit-card number are valid, consistent, and within a given geographical area. Such functions may be undertaken by the high-level ESD components 10 illustrated in Figure 1.

Figure 3A shows the procedure for constructing the aggregate distribution file 170 in greater detail. For the sake of illustration, the original distribution file 130 is assumed to have a structure including header information and different types of internal sections for code, static data and so on, such as a Windows™ 'Portable Executable' (PE) program file. One of ordinary skill in the art can appreciate that the method and system of the present invention can be applied to a number of different file formats. Similarly, the inputs 140, 151 to the conversion program 110, and output 170 from the conversion program 110 are illustrated to be computer files, but they could be in-memory images, streams from other processors, etc.

A typical sequence of steps run by the SDA 100 to construct the aggregate distribution file 170 is described below.

1. The conversion program 110 is run, as a result of the user interaction program 120 having determined that a conversion is required i.e. that a delivery of a particular aggregate distribution file 170 according to the method of the present invention is authorized, and that the required embedded data block 140 has been constructed. All subsequent steps are executed by the conversion program 110 unless otherwise indicated.

The object is to obtain what is often referred to as a "digital signature", or "cryptographic signature" which inherently has two aspects:

(i) By the use of a cryptographic hash algorithm, the production of a cryptographic fingerprint that uniquely corresponds to the data "ed" 130, 140; and

(ii) Protection of that cryptographic fingerprint by encrypting it with a private key, such that the recipient of the cryptographic fingerprint may, by using a public key and the cryptographic algorithm, verify that the data "ed" 130, 140 is intact, without having the ability to generate a new cryptographic fingerprint, and plausibly change the data.

These two steps are essential to realize the advantage of the present invention, since without both steps a third party may intervene and alter data without the recipient being able to detect it. This procedure is to be distinguished from simply encrypting the data "ed" 130, 140, which is a step that is not necessary to the operation of the present invention since there is no way to plausibly alter the data 130, 140 without such alterations being detectable.

2. The input/output logic 111 of the conversion program 110 reads in the desired original distribution file 130, its corresponding cryptographic private key 151, and the data to be embedded 140. Though not required by the conversion program 110, a public key 152 may be passed through in order that it may be added to the aggregate distribution file 130. Utilizing cryptographic hash algorithms 112 and Public-Private key (PPK) encryption algorithms 113, a cryptographic signature 174 is produced. The basic steps of this process are:

2.1 Apply a one-way hash function "hf" to the data "ed" 130, 140 producing a cryptographic fingerprint "edh", that is edh = hf(ed). The requirements for this cryptographic fingerprint are as follows: (i) that it produce a reasonably compact result i.e. size(edh) << size(ed), and preferably a fixed-length result; (ii) that the fingerprint alone cannot be used to ascertain the original data block back i.e. there is no back-hash function "bhf" such that bhf(edh) = ed; (iii) that it be extremely sensitive to changes in "ed"; say, that a single-bit change in "ed" changes on average 50% of the bits in "edh", and (iv) that it is extremely difficult to

construct a false embedded data block "fed" which produces the same fingerprint as "ed", that is hf(ed) = hf(fed). There are a number of algorithms which satisfy these requirements, such as MD5 (Message Digest 5) and SHA (Secure Hash Algorithm). Other algorithms that also meet the above criteria that may be employed by the present invention.

2.2 Encrypt the cryptographic fingerprint "edh" using the private key 151 "prk" and a public/private encryption function "ppef" to produce a cryptographic signature "edf" 174, that is: edf = ppef(prk, edh). The requirements for the encryption function "ppef" are as follows: (i) that it produce a result not substantially larger than its input; (ii) that it effectively protect relatively short data sets, since "edh" will be bytes long rather than kilobytes long; (iii) that is computationally infeasible to use the public key 151 ("puk") and the cryptographic signature "edf" 174, or multiple instances of "edf" 174 (which will be visible on the installation computer) to infer the private key "prk", that is, there is no cracking function "cf" such that: puk = cf(edf,puk); (iv) that there is no conceivable means of replicating the behaviour of "ppef" using "prk" without in fact possessing both "ppef" and "prk". In principle, "ppef" can be inferred from its corresponding decryption function, so "prk" is the important secret in practical terms; (v) that the corresponding public-key decryption function "ppdf" have acceptable performance on a typical installation computer for the pertinent file sizes. Note that if a specific ppef/ppdf is chosen for security reasons and does not yield acceptable performance, the encryption could be applied to only a portion of the selected files and still offer the same benefits; (vi) that it be suitable (preferably, via established cryptanalysis) for specific application in this domain i.e. digital signatures. There are a number of algorithms which might satisfy these requirements, such RSA and those of Rabin and ElGamal. The

careful selection of implementation parameters can help attain required security and performance.

3. The cryptographic signature 174 from step 2.1, and the data to be embedded 140 are inserted into the original distribution file 130 to produce the aggregate distribution file 170. This insertion is not a simple copying of bits into the middle of a file, since it must be compliant to the format requirements of the particular file types. For example, headers may have to be updated to identify the new data etc.

The system and method of the present invention does not require that the embedded data 171 or the cryptographic signature 174 be positioned in any particular manner in the aggregate distribution file 170. What is necessary is that: (i) the software on the installation computer, and the UIA 200 in particular, be able to locate the embedded data 171 and cryptographic signature 174, and (ii) that the aggregate distribution file 170, after it is installed on the installation computer, be able to perform its intended function; for example, if it is an executable file, that it still conform to structural and other platform requirements so it can load and run on the installation computer as it might have before the conversion process. For example, if the file were in a format common to current computers containing an Intel™ microprocessor and running a Microsoft™ Windows™ operating system, the conversion program 110 could inspect the "header" section of the original distribution file 130 to determine where there were sections containing static data so as to avoid sections containing executable code. A static data section would be selected and a suitable location for the embedded data block 171 and cryptographic signature 174 would be found or created. This would be done by, for example, (i) determining that an existing static data block had unused capacity sufficient to add the data, (ii) allocating a new static data block, or (iii) expanding an existing static data block.

The method illustrated in Figure 3A discloses a one-step process wherein cryptographic signature 174 is ascertained

for the original distribution file 130 and the embedded data 140. An optional method, such as that illustrated in Figure 3B, would be to employ a two-step process wherein a cryptographic signature 172 of the embedded data 171 is first produced using the same algorithm described in step 2 above. This embedded data cryptographic signature 172 is then itself embedded into the original distribution file 130. The original distribution file 130, embedded data 171, and embedded data cryptographic signature 172 are then input to the second cryptographic step, wherein an overall cryptographic signature 176 is ascertained using the same algorithm described in step 2 above. The benefit of the two step process is that it augments the capabilities of the system and method of the present invention to authenticate and detect tampering in the software application installed on the installation computer. For example, separate cryptographic public/private key pairs could be provided for the two cryptographic signatures 172, 176. Furthermore, the two-step process allows the embedded data 171 to be extracted and authenticated, even if the original file contents 173a, 173b have been corrupted.

Another alternative is to construct the aggregate distribution file 170 using a variation of the two-step cryptographic process wherein a first cryptographic signature 175 is made of only the original file contents 173a, 173b, and a second cryptographic signature 172 is made of the embedded data 171. This is illustrated in Figure 3C. This scheme has all the advantages of the two-step process illustrated in Figure 3B, and also allows for separate authentication of the embedded data 171 and the original file contents 173a, 173b. This would allow a user to verify that original distribution file 130 provided by the publisher had not been altered by the on-line installation process disclosed by the present invention.

One of ordinary skill in the art will appreciate that any of the cryptographic signatures 172, 174, 175, 176 shown in Figures 3A, 3B and 3C do not have to be produced using the same set of cryptographic public/private key pairs, or even the same

cryptographic algorithms. As well, it is not necessary that the cryptographic signatures 172, 174, 175, 176 be calculated each time an aggregate distribution file 170 is distributed to a user. The SDA 100 could maintain a database of partially-precomputed signatures to speed up the related calculations. The availability of cryptographic hardware support such as RSA co-processors in the installation computer, could be used to attain good responsiveness with maximal security. As well, it is not essential that the aggregate distribution file 170 be constructed in its entirety by the SDA 100. What is necessary is that the aggregate distribution file 170 be derivable in its entirety by the UIA 200.

Figure 4 illustrates the structure and operation of the UIA 200 which consists of a transient installation index 204, a transient installation input fileset 205, and a UIA proper executable software program 203. One skilled in the art will appreciate that there are many ways in which the UIA program 203 could be implemented. Since a significant part of the UIA's 200 functionality involves user interaction and dialog with the SDA 100, options for the implementation of the UIA 200 include either making it an adjunct to a World Wide Web (WWW) browser, or implementing it as a stand-alone program which itself embeds or invokes already-present browser capability on the installation computer.

A typical execution sequence of the UIA 200 is described below:

1. After the UIA program 203 and its support data 204, 205 have been copied onto the installation computer, the user runs the UIA program 203. Note that the UIA program 203 could also have been initiated remotely e.g. sent as an active program within a browser framework by a WWW server. Unless otherwise stated, all subsequent steps are executed by the UIA program 203.

2. The installation index 204 and installation input fileset 205 are read by the installation computer to determine the particular default SDA 100 appropriate for the installation

of the desired software application (known as the "installed aggregate distribution file") 15.

3. The installation computer is examined to determine the probable means of establishing communications with the SDA 100, for example, the presence of TCP/IP network interfaces, modems etc. If no such means are found, the program optionally assists the user to find parameters which will work properly, then fails with a warning. This is because access to the SDA 100 is essential for operation of the invention.

4. The user 1 is prompted with the default data from steps (2) and (3) above, i.e. informed where the UIA program 203 will look for the desired SDA 100, and over what sort of distribution channel 300. The user 1 is then given an opportunity to change this information, either for commercial reasons (e.g. maybe an SDA has changed names or locations), or for technical reasons (e.g. the user does not have working TCP/IP connectivity and wants to use a straight modem link, perhaps via an 800 number.)

5. Via the distribution channel 300, the UIA program 203 establishes contact with SDA 100. If this cannot be done, the UIA program 203, after optionally helping the user determine parameters which will work properly, fails with a warning. While the security of the distribution channel 300 is optional to the operation of this invention, it is expected that the distribution channel 300 will support appropriate protocols to protect the SDA 100 from fraud. A common protocol supporting authentication and privacy, such as Secure Sockets Layer (SSL) is appropriate.

6. The UIA program 203 acts as an intermediary between the user and the SDA 100, enabling the user 1 to establish any legitimate agreement which the SDA 100 supports with respect to the desired installed aggregate distribution file 15. The UIA program 203 also has the ability to determine whether the available system resources of the installation computer meet the requirements of the desired installed aggregate distribution file 15.

There are no technical limits to the variety of options that can be displayed to the user, the questions the user might be asked, the data that might be gathered about the installation computer, etc. Since the SDA 100 is being run throughout the data gathering, data embedding, software distribution and software installation process, the system and method of the present invention can employ various levels of cryptography without the user ever being informed of the cryptographic keys, or any information from which they could be derived. This is unlike other electronic delivery systems which typically require subsequent off-line entry of 'secret keys' or derivates thereof which have been explicitly divulged to the user. Of course, public keys used for the authentication of cryptographic signatures are an exception in that the user may be able to determine them easily, however this is not a security issue since they have no fraudulent application.

7. Assuming that the user 1 meets all the criteria set out by the SDA 100, the SDA 100 will determine a specific set of files that must be transmitted to the UIA 200 to complete installation on the installation computer, notably including at least one aggregate distribution file 170 (shown in Figures 3A-3C). It is immaterial to the system and method of the present invention what is the nature of the agreement entered into between the user 1 and SDA 100, or how it is validated. That is the responsibility of the SDA 100 and its subtending commercial systems 10, if any. Most importantly, the UIA 200 does not and cannot itself decide whether an agreement has been reached between the user and the SDA 100. The UIA 200 does not have, and should not have, access to all the information required to complete the installation, except through interaction with the SDA 100.

8. The SDA 100 transmits an index of the required distribution files to the UIA 200 via the distribution channel 300. The UIA 200 uses this index to augment its own local index 204 forming a complete index for the upcoming installation.

9. The SDA 100 constructs one or more aggregate distribution files 170 and any other files required for the installation, and transmits these files to the UIA 200 via the distribution channel 300.

10. Using its local index and support files 204, 205 the UIA program 203 completes the installation of the installed aggregate distribution file 15 in a manner compliant with the platform of the installation computer. In particular, the UIA 200 installs the aggregate distribution file 170 such that the cryptographic signature 174 and the embedded data 171 are unaffected. Once the aggregate distribution file 170 is installed on the installation computer, it is referred to as an installed aggregate distribution file 15. The UIA program 203 will also perform other system updates 212 as necessary, such as updating the operating system registry (in the case of Windows 95™), and installing any additional application files. Other optional operations, such as leaving an appropriate 'uninstall' utility, may also be involved.

12. If an error should occur, the UIA program 203 may signal the SDA 100 to re-initiate the installation. If no error has occurred, the UIA program 203 signals the SDA 100 that all required data has been received. This could, for example, be used as the trigger signal for the SDA 100 to commit to a financial transaction. Leaving the financial commit to this late part of the process minimizes the probability of the user being charged for a software application which has not been successfully installed, thus reducing one cause of customer frustration.

13. The UIA program 203 deletes any transient files, indices etc. that it might have placed on the installation computer.

14. The UIA program 203 disconnects from the SDA 100 and the distribution channel 300 and exits.

Upon successful completion of the optional authentication procedures described in further detail below, the user can then run the installed aggregate distribution file 15 on the installation computer. It should be understood that

the authentication procedures described below can be done either before or after the installation is completed.

The method and system of the present invention would diminish disputes arising from software which is purchased but not successfully installed. The UIA 200 can detect and warn the user if the installation computer had inadequate resources to run the desired software application, before any financial transaction has been made. Further, the final financial commitment to purchase the software application by the user can be done late in the installation process so that the probability of the financial transaction being successful, but the installation itself failing, would be low.

One of ordinary skill in the art will appreciate that the UIA 200 may be distributed to users in a mass-produced media form containing the original distribution file 130, or a derivative thereof not subject to successful fraudulent re-use through simple copying. In this scenario, the SDA 100 would transmit to the UIA 200 only the incremental information which the UIA 200 would require to construct the aggregate distribution file 170 and complete the installation. Any attempts to pirate the software application can be defeated by ensuring that the distribution form of the UIA 200 contains an incomplete set of executable files, thereby requiring essential data from the SDA 100 to be capable of executing on the installation computer.

Figure 5 illustrates the means of authenticating and extracting user data from an installed aggregate distribution file 15 to verify that neither the original file contents 173a, 173b, nor the embedded data 171 have been tampered with. This step is optional to the operation of the present invention because the installed aggregate distribution file 15 may be run by the user without authentication. It should be understood that the authentication procedures described below can be done either before or after the installation is completed. If authentication is done prior to installation on the installation computer, then the following procedures are

directed by the UIA 203 to the aggregate distribution file 170, instead of the installed aggregate distribution file 15.

The process illustrated in Figure 5 is in relation to an installed aggregate distribution file 15 constructed using the two-step process illustrated in Figure 3B. The principles of authenticating and extracting user data from an installed aggregate distribution file 15 constructed using the one-step cryptographic process illustrated in Figure 3A, or the variant of the two-step process illustrated in Figure 3C, are the same as those described below, with appropriate modifications, depending on the nature of the cryptographic signatures to be compared.

Though a separate authentication and reading program 400 is shown performing the functions of authentication and reading of embedded data 171, a person skilled in the art will appreciate that these functions need not be embodied in such a stand-alone program, and could be incorporated as functions of other programs, such as the UIA 200, a license-checker, a virus-checker, a program loader, a copy program, etc. A typical execution sequence of the authentication and reading program 400 is described below:

1. The authentication and reading program 400 is run, either by a user or by automatic invocation from another program such as the UIA 200. Unless otherwise indicated, the following steps are all executed by authentication and reading program 400.

2. Determine which installed aggregate distribution file 15 to process, either by prompting the user or having this passed as a parameter by the UIA 200. Also determine (if derivable therefrom in the particular implementation, as opposed to being contained in the file itself), which particular public key 152 is applicable to this installed aggregate distribution file 15.

3. Open the installed aggregate distribution file 15 in question and check that it meets the applicable format requirements. For example, a given implementation might support executable (EXE) and dynamic link library (DLL) files

in the 'PE' format for Intel™ processors. If the installed aggregate distribution file 15 fails these basic checks, or is not found, the authentication and reading program 400 fails with an appropriate warning.

4. Examine the file to determine the location of the overall cryptographic signature 176, the embedded data cryptographic signature 172, and the embedded data 171. The installed aggregate distribution file 15 can be formatted in various ways to support this, such as including pointers to these sections in the file header. If applicable in the particular implementation, (i.e. the public key 152 is included in the file as opposed to being otherwise determined the authentication and reading program 400), find and extract the required public key 152.

If any of the above steps fail, the authentication and reading program 400 fails with an appropriate warning.

5. Use the public key 152 to decrypt the overall cryptographic signature 176 into its unencrypted form 176a (the decrypted remote overall fingerprint).

6. Using the same known cryptographic signature algorithm as was employed by the SDA 100, calculate a local version 176b (the locally calculated overall fingerprint) of the overall cryptographic signature. This calculation will necessarily exclude the overall cryptographic signature 176 itself i.e. cover all parts of the installed aggregate distribution file 15 except 176, in order that the locally calculated overall fingerprint 176b will not depend on itself.

7. Compare the locally calculated overall fingerprint 176b to the decrypted remote overall fingerprint 176a. If they differ, the authentication and reading program 400 will fail with a warning that the installed aggregate distribution file 15 has been corrupted. At this point, the UIA 200 may be invoked to contact the SDA 100 to re-acquire the installed aggregate distribution file 15.

8. Extract the embedded data 171 and present it graphically to the user, if the program has been user-invoked,

or pass it in message form to the invoker routine, if software-invoked.

9. Use the public key 152 to decrypt the embedded data cryptographic signature 172 into its unencrypted form 172a (the decrypted remote embedded data fingerprint).

10. Calculate a local version 172b (the locally calculated embedded data fingerprint) of the embedded data cryptographic signature 172 using the same known cryptographic signature algorithm as the SDA 100 used.

11. Compare the locally calculated embedded data fingerprint 172b to the decrypted remote embedded data fingerprint 172a. If they differ, the authentication and reading program 400 will fail with a warning that the embedded data 171 has been corrupted.

A similar procedure of comparison would be followed in respect of the cryptographic signature 174 if the one step process illustrated in Figure 3A had been followed. As well, a similar procedure of comparison would be followed in respect of the original file contents cryptographic signature 175 if the variant of the two-step cryptographic process illustrated in Figure 3C had been undertaken.

Figure 6 is a flow-chart of a summary of the procedures described in relation to Figures 2, 3A, 3B, 3C, 4 and 5. It should be noted that the public key 152 used to authenticate the integrity of the installed aggregate distribution file 15 could be delivered to the UIA 200 by any means since it is not a secret and might be useful for more than one purpose. For example, the public key may be embedded in the aggregate distribution file 170, it may be explicitly sent to the user as a separate file or message, or it may be obtained automatically by the installation computer from a network trusted authority (e.g. Verisign™ Inc.)

Figure 7 is a flow-chart of another set of procedures that may be employed in accordance with the present invention, whereby the original file contents 173a, 173b are encrypted using a unique private key calculated by the SDA 100 for this particular transaction. A record of this unique private key is

kept by the SDA 100, and the corresponding unique public key is transmitted with the aggregate distribution file 170 via the distribution channel 300 to the UIA 200. The UIA 200 will decrypt the aggregate distribution file 170 using the public key. For security reasons, it is preferred that this public key not be permanently stored on the installation computer. Instead, the unique public key would exist only in the computer's Random Access Memory (RAM) for the duration of the installation. This makes usable redistribution of the aggregate distribution file 170 practically impossible.

Although the present invention has been described with reference to the preferred embodiments, one of ordinary skill in the art will recognize that a number of variations, alterations and modifications are possible. In Figure 8 there is an illustration showing the various uses of the installed aggregate distribution file 15. After installation and authentication by the UIA 200, the installed aggregate distribution file 15 may run normally without making use of the embedded data 171 in any way. To ensure licence compliance, the installed aggregate distribution file 15 may also be run in association with a license-enforcement program that verifies that any license terms comprising part of the embedded data 171 are being complied with. The embedded data 171 and cryptographic signatures 172, 174, 175, 176 (depending on the manner in which the aggregate distribution file 170 was constructed) may also be used as an input to a virus checker that may perform an integrity check on the installed aggregate distribution file 15 by using the public key 152 and the same known cryptographic signature algorithm as was employed by the SDA 100. Each time the installed aggregate distribution file 15 is run, the authentication and reading program 400 shown in Figure 5 may also be run, either by itself, or in association with an authenticating loader that would reject tampered files, and would not permit a tampered installed aggregate distribution file 15 to be run. The embedded data 171 may also be used simply for display to the user.

The method and system disclosed herein can also be used to upgrade an installed aggregate distribution file 15 present on an installation computer. In this case, the UIA 200 and the SDA 100 would verify of the license status of the installed aggregate distribution file 15 present on the installation computer, and then invoke the method and system disclosed herein to construct, deliver and install an upgraded version of the installed aggregate distribution file 15 to the installation computer. The capability to invoke the upgrading feature of the present invention could be done at the request of the user, or it could be invoked automatically upon detection by the UIA 200 of the availability of a new version of the original distribution file 130.

The uniqueness of the installed aggregate distribution file 15 can be used to restrict its operation to a specific central processing unit (CPU) on the installation computer. The identification of the CPU for these purposes would be done by the UIA 200 during the stage of gathering data 32, 34 for transmission to the SDA 100.

The SDA 100 and UIA 200 disclosed herein are not restricted to being invoked at the time of installation or upgrading of the installed distribution file 170. For example, in a computer game environment, the SDA 100 and UIA 200 could be invoked when the user reaches a certain point in the game, giving the user the option to purchase additional functionalities or levels for the game.

This disclosure does not presuppose that the UIA 200 does not possess added intelligence to increase the functionality of the present invention. For example, the UIA 200 may possess the intelligence to find and recognize separate Personal Digital Certificates on the installation computer which establish his identity for purposes sufficient to authorize all, or part of, the transaction in question. Such Personal Digital Certificates and their method of application would conform to established standards such as those used by commercial certificate provider Verisign™ Inc. In addition, the UIA 200 could possess the intelligence to find and

recognize digital "coupon" certificates which establish that the user has some specific privilege, such as an entitlement to a specific price for a piece of software, or one which establishes his membership in a specific group, such as a company. In addition, the UIA 200 could locate pre-existing files installed according to the method of the present invention, and examine the embedded data 171. If the UIA 200 determines that there is license information present which may affect the terms of the transaction, or which may indicate a user's likely interest in, for example, an upgrade, the UIA 200 can transmit this information to the SDA 100 so that it can suitably mediate the transaction, advertise an upgrade, etc. A typical example of this would be examining a word-processing application installed in accordance with the present invention to determine that the user is entitled to a free upgrade, which the present invention can then proceed to install.

In another set of variations of the invention, the installed aggregate distribution file 15 is one which uses the principles of Nortel Algorithmic Authorization (NAA), as disclosed in U.S. Patent Application No. 08/674,037 to add robust self-policing of its own integrity. In a first variation, the run-time NAA algorithms, which already have the capability of using the installed aggregate distribution file's 15 own code as an input required for proper operation, and thus of forcing catastrophic failure in the event of tampering, have the scope of this input expanded to include an in-memory copy of one or more of the data items added by the SDA 100, such as the overall cryptographic signature 176.

In a second variation, the "launch stub" component could go further, extracting and decoding the embedded data 171 in the installed aggregate distribution file 15, and comparing the license terms therein (e.g. a specific CPU identified by, say, a certain physical Media Access Control address on a network card) to those it found by examining its current environment. In accordance with the principles of Nortel Algorithmic Authorization, the "launch stub" would not have to "decide" whether to proceed, since such decision-

points are obvious attack points for 'hackers' wishing to defeat security mechanisms. Rather, it could modify data upon which proper program operation depended, in such a way that the program would continue to run properly only if said data corresponded to the proper environment per the license. As for the first variation, the application would have been pre-constructed for the specific instance, as per the patent-pending technology, in such a way that its proper flow of control used input data that was initially 'incorrect' in just such a way as to be 'corrected' only by application of the correct license data, or a simple derivative thereof.

The invention disclosed herein does not necessarily alter the functionality of the installed form of the installed aggregate distribution file 15, it only adds information and authenticability to it. However, there are a number of means by which the behaviour of the installed aggregate distribution file 15 can be mediated in new ways enabled by this invention. In one variation, the SDA 100 would have access to a variety of executable forms for a given program, or to software routines which would dynamically construct variant forms, in order to produce a program which meets particular customer function/cost requirements, and/or which actively binds itself to very specific license terms. For example, in the Microsoft Windows™ environment, different behaviour could be embodied by different Dynamic Link Libraries (DLLS) which could be selectively included.

In another variation, the initial executable form of the program file would have specific functional and license-binding choices built-in, and the SDA 100 would inject (possibly authenticable) data into the executable file which caused it to exhibit the desired behaviour. In yet another variation, the SDA 100 could make use of routines with detailed knowledge of specific program structures in order to add variant code to a pre-existing executable program which was not explicitly designed to accommodate such variation.

The described embodiments of the present invention focus on a single "core" file of a specific file type as the

cornerstone of a software application's installation and security. However the method of the present invention may certainly be applied to more than one file or file type in a particular case. For example, all of the static files associated with an installed software application could receive embedded information such that they were all authenticable and associable with the particular application and installation instance.

We Claim:

1. A method for the electronic distribution of a software application from a distribution computer to an installation computer comprising the steps of:

a. receiving at said distribution computer identifying information;

b. embedding said identifying information in said software application at said distribution computer to form an identifiable software application;

c. generating a cryptographic signature for said identifiable software application;

d. embedding said cryptographic signature in said identifiable software application to form an identifiable and authenticable software application; and

e. transferring said identifiable and authenticable software application from said distribution computer to said installation computer.

2. The method of claim 1, wherein the step of generating a cryptographic signature for said identifiable software application includes the steps of

a. applying a one-way hash function "hf" to the identifiable software application "ed" producing a hash result "edh", where edh = hf(ed); and

b. encrypting the hash result "edh" using a cryptographic key to obtain a cryptographic signature.

3. The method of claim 2, wherein the one-way hash function is generated using any one of a Message Digest 5 (MD5) algorithm and a Secure Hash Algorithm (SHA) algorithm.

4. The method of claim 2 or 3, wherein the step of encrypting the hash result "edh" includes the step of using a public/private encryption function "ppef" and a private encryption key "prk" to encrypt the hash result "edh" to produce a cryptographic signature "edf" where edf = ppef(prk, edh).

5. The method of claim 4, wherein the public/private encryption function "ppef" is generated using any one of an RSA algorithm, a Rabin algorithm and an ElGamal algorithm.

6. The method of claim 1, 2, 3, 4 or 5, wherein the distribution computer and the installation computer are connected by the Internet.

7. The method of claim 1, 2, 3, 4, 5, or 6, wherein the identifying information received at said distribution computer is transmitted from said installation computer.

8. A method of receiving electronically at an installation computer a software application distributed from a distribution computer comprising the steps of:

a. receiving an identifiable and authenticable software application from the distribution computer, the identifiable and authenticable software application having embedded therein the identifying information and a cryptographic signature of the identifiable and authenticable software application; and

b. installing the identifiable and authenticable software application at the installation computer.

9. The method of claim 8, wherein prior to the step of receiving an identifiable and authenticable software application from the distribution computer, the installation computer transmits identifying information to the distribution computer.

10. The method of claim 8 or 9, wherein prior to the step of installing the identifiable and authenticable software application, the installation computer authenticates the integrity of the software application.

11. The method of claim 10, wherein the installation computer uses the cryptographic signature to authenticate the integrity of the software application.

12. A method for the electronic distribution of a software application from a distribution computer to an installation computer comprising the steps of:

a. receiving identifying information at said distribution computer;

b. embedding said identifying information in said software application at said distribution computer to form an identifiable software application;

c. generating a cryptographic signature for said identifiable software application;

d. embedding said cryptographic signature in said identifiable software application to form an identifiable and authenticable software application;

e. transferring said identifiable and authenticable software application from said distribution computer to said installation computer; and

f. installing said identifiable and authenticable software application at said installation computer.

13. The method of claim 12, wherein the distribution computer and the installation computer are connected by the Internet.

14. The method of claim 12 or 13, wherein the identifying information received at said distribution computer is transmitted from said installation computer.

15. A software distribution computer for distributing an identifiable and authenticable software application to a user comprising:

a. a communications link between said software distribution computer and said user;

b. a storage device for storing a software application for distribution;

c. a communications interface in communication with said link, for receiving identification data from said user, and for transferring said identifiable and authenticable software application to said user;

d. means for embedding identification data received from said installation computer in said software application to form an identifiable software application;

e. means for generating a cryptographic signature for said identifiable software application; and

f. means for embedding said cryptographic signature in said identifiable software application to form said identifiable and authenticable software application.

16. A software installation computer for receiving an identifiable and authenticable software application distributed by a distribution computer:

a. a communications link between said software installation computer and said software distribution computer;

b. a storage device for storing identification data, and for storing an installed software application;

c. a computer communications interface in communication with said link, for transferring said identification data, and for receiving said identifiable and authenticable software application, the identifiable and authenticable software application having embedded therein the identification data, and a cryptographic signature of the identifiable and authenticable software application;

d. means for installing said identifiable and authenticable software application on said computer storage device.

17. A software distribution computer for distributing an identifiable and authenticable software application from a distribution computer to an installation computer comprising:

a distribution computer;

an installation computer;

a communications link between said installation computer and distribution computer;

said distribution computer comprising:

a. distribution computer storage device for storing a software application for distribution;

b. a distribution computer communications interface in communication with said link, for transferring an identifiable and authenticable software application to said installation computer and for receiving identification data from said installation computer;

c. means for embedding identification data received from said installation computer in said software application to form an identifiable software application;

d. means for generating a cryptographic signature for said identifiable software application; and

e. means for embedding said cryptographic signature in said identifiable software application to form an identifiable and authenticable software application;

said installation computer comprising:

a. an installation computer storage device for storing said identification data, and for storing an installed software application;

b. an installation computer communications interface in communication with said link, for transferring said identification data to said distribution computer and for receiving said identifiable and authenticable software application from said distribution computer; and,

d. means for installing said software application on said installation computer storage device.

FIG. 1

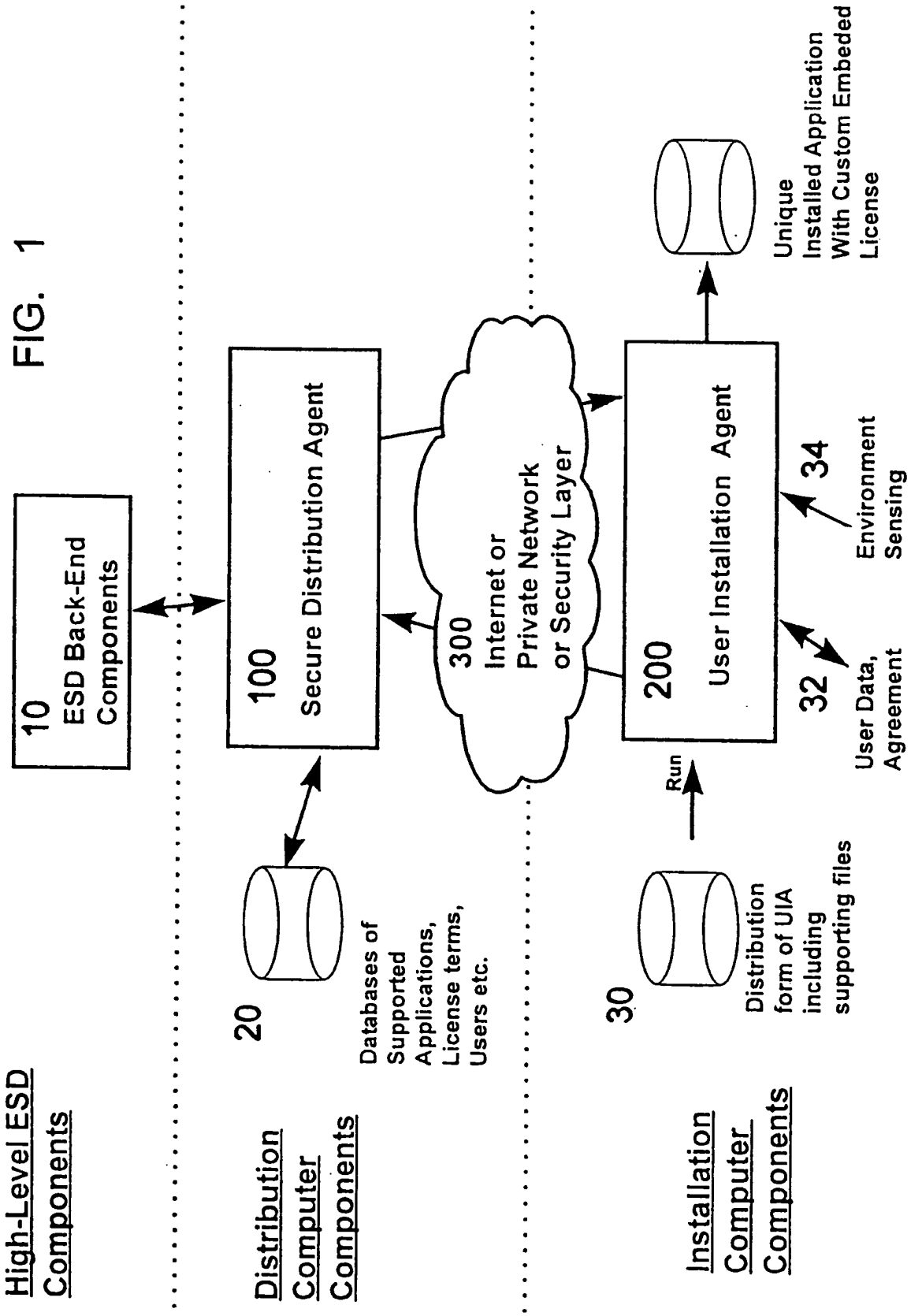
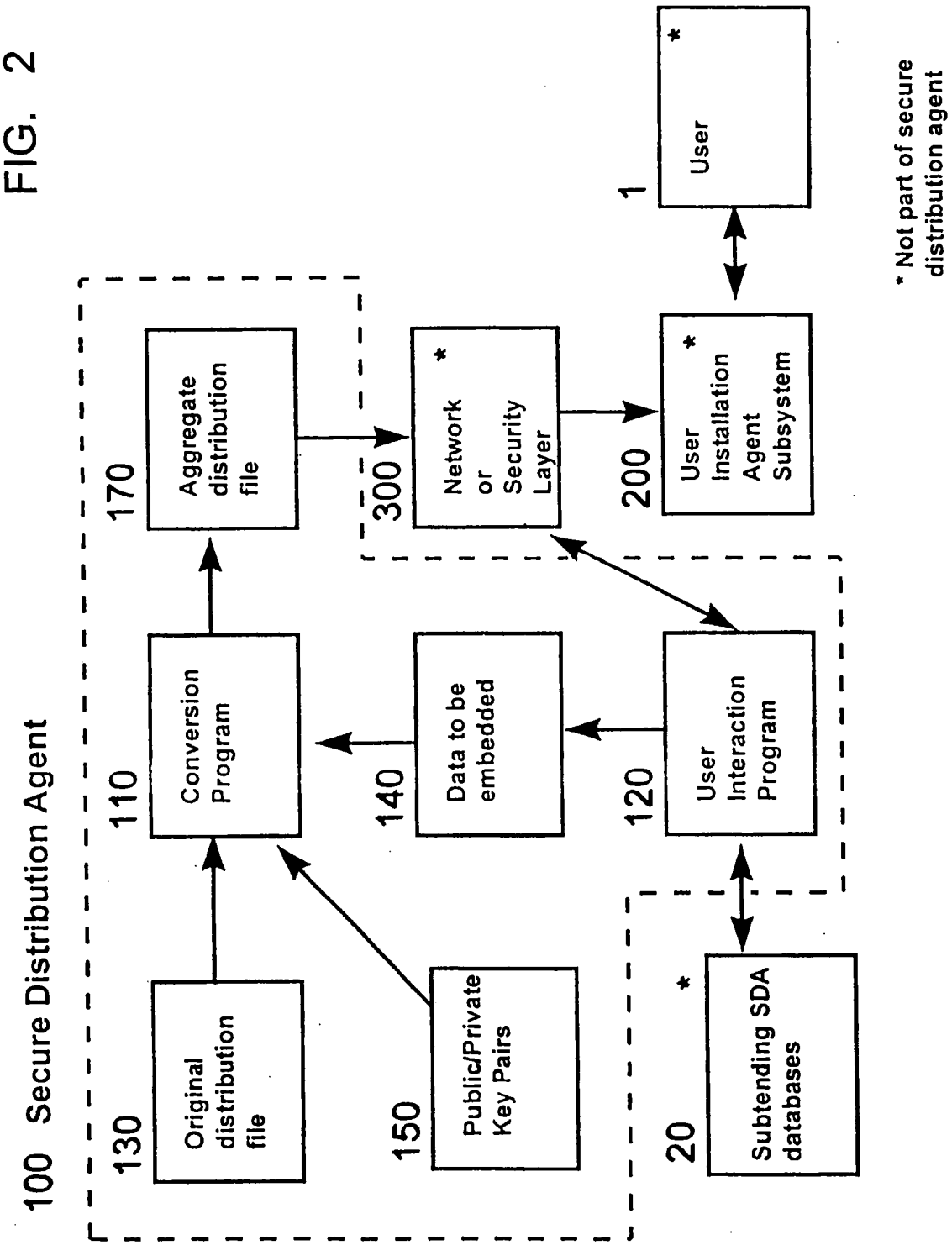

High-Level ESD Components
Distribution Computer Components
Installation Computer Components
10 ESD Back-End Components
100 Secure Distribution Agent
300 Internet or Private Network or Security Layer
200 User Installation Agent
20
Databases of Supported Applications, License terms, Users etc.
30
Distribution form of UIA including supporting files
Run
32
User Data, Agreement
34
Environment Sensing
Unique Installed Application With Custom Embeded License

FIG. 2


100 Secure Distribution Agent
1
User
*
* Not part of secure distribution agent
170
Aggregate distribution file
300
Network or Security Layer
*
200
User Installation Agent Subsystem
*
110
Conversion Program
140
Data to be embedded
120
User Interaction Program
130
Original distribution file
150
Public/Private Key Pairs
20
Subtending SDA databases
*

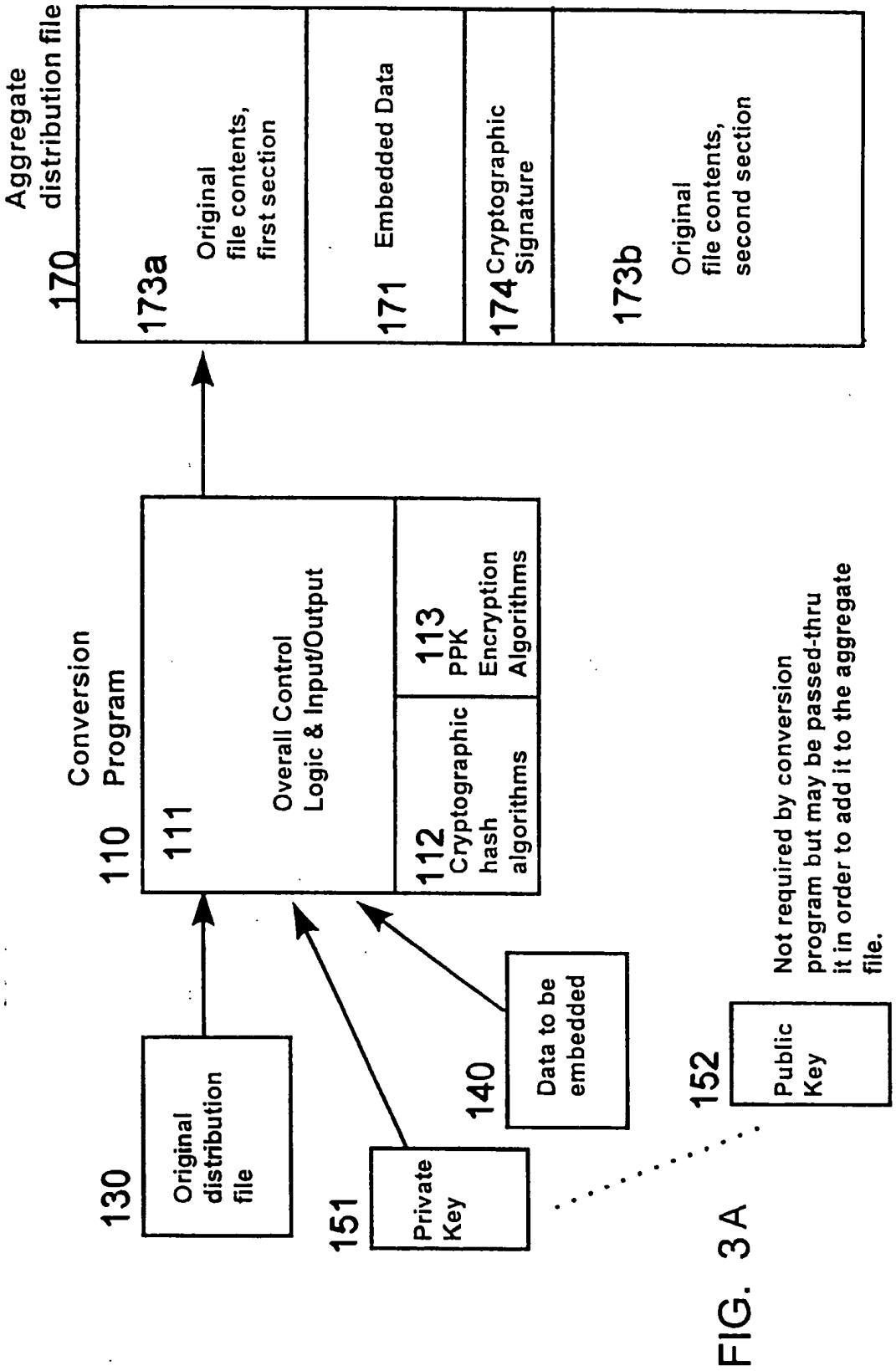

Aggregate distribution file
170
173a
Original file contents, first section
171
Embedded Data
174
Cryptographic Signature
173b
Original file contents, second section
Conversion Program
110
111
Overall Control Logic & Input/Output
112
Cryptographic hash algorithms
113
PPK Encryption Algorithms
130
Original distribution file
151
Private Key
140
Data to be embedded
152
Public Key
Not required by conversion program but may be passed-thru it in order to add it to the aggregate file.

FIG. 3A

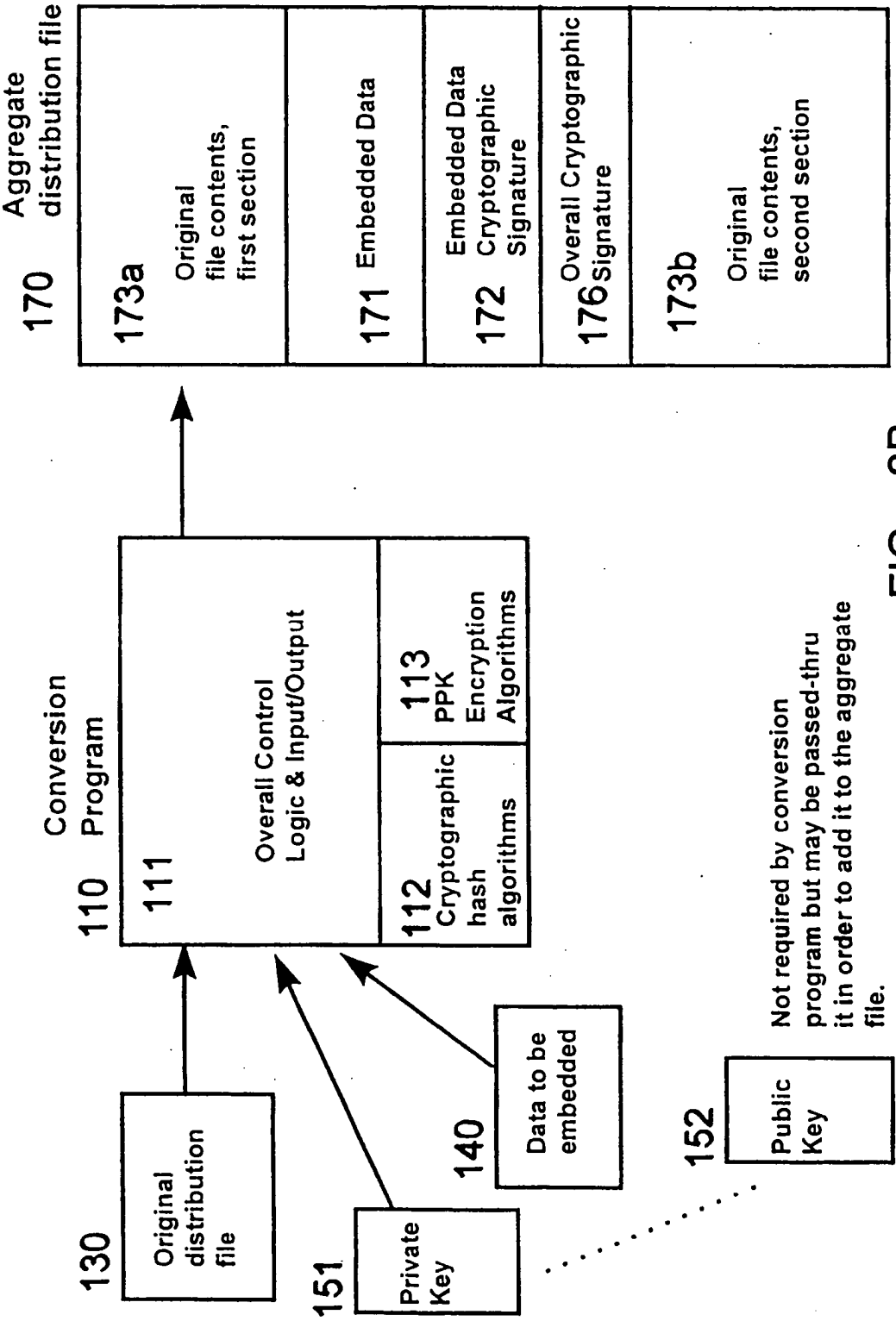

Aggregate distribution file
170
173a
Original file contents, first section
171 Embedded Data
172 Embedded Data Cryptographic Signature
176 Overall Cryptographic Signature
173b
Original file contents, second section
Conversion Program
110
111
Overall Control Logic & Input/Output
112 Cryptographic hash algorithms
113 PPK Encryption Algorithms
130
Original distribution file
151
Private Key
140
Data to be embedded
152
Public Key
Not required by conversion program but may be passed-thru it in order to add it to the aggregate file.

FIG. 3B

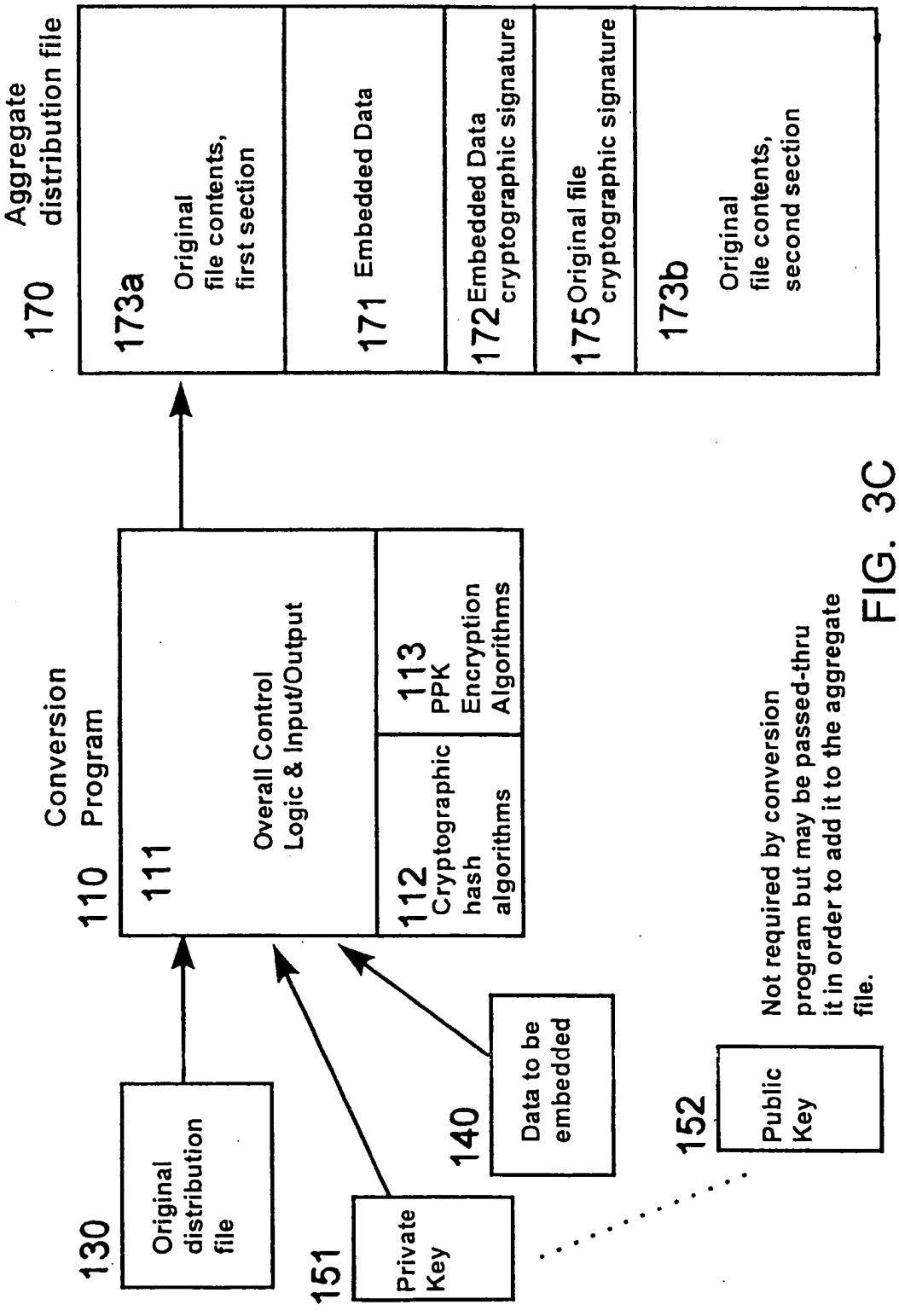

Aggregate distribution file
170
173a
Original file contents, first section
171 Embedded Data
172 Embedded Data cryptographic signature
175 Original file cryptographic signature
173b
Original file contents, second section
Conversion Program
110
111
Overall Control Logic & Input/Output
112 Cryptographic hash algorithms
113 PPK Encryption Algorithms
130
Original distribution file
151
Private Key
140
Data to be embedded
152
Public Key
Not required by conversion program but may be passed-thru it in order to add it to the aggregate file.

FIG. 3C

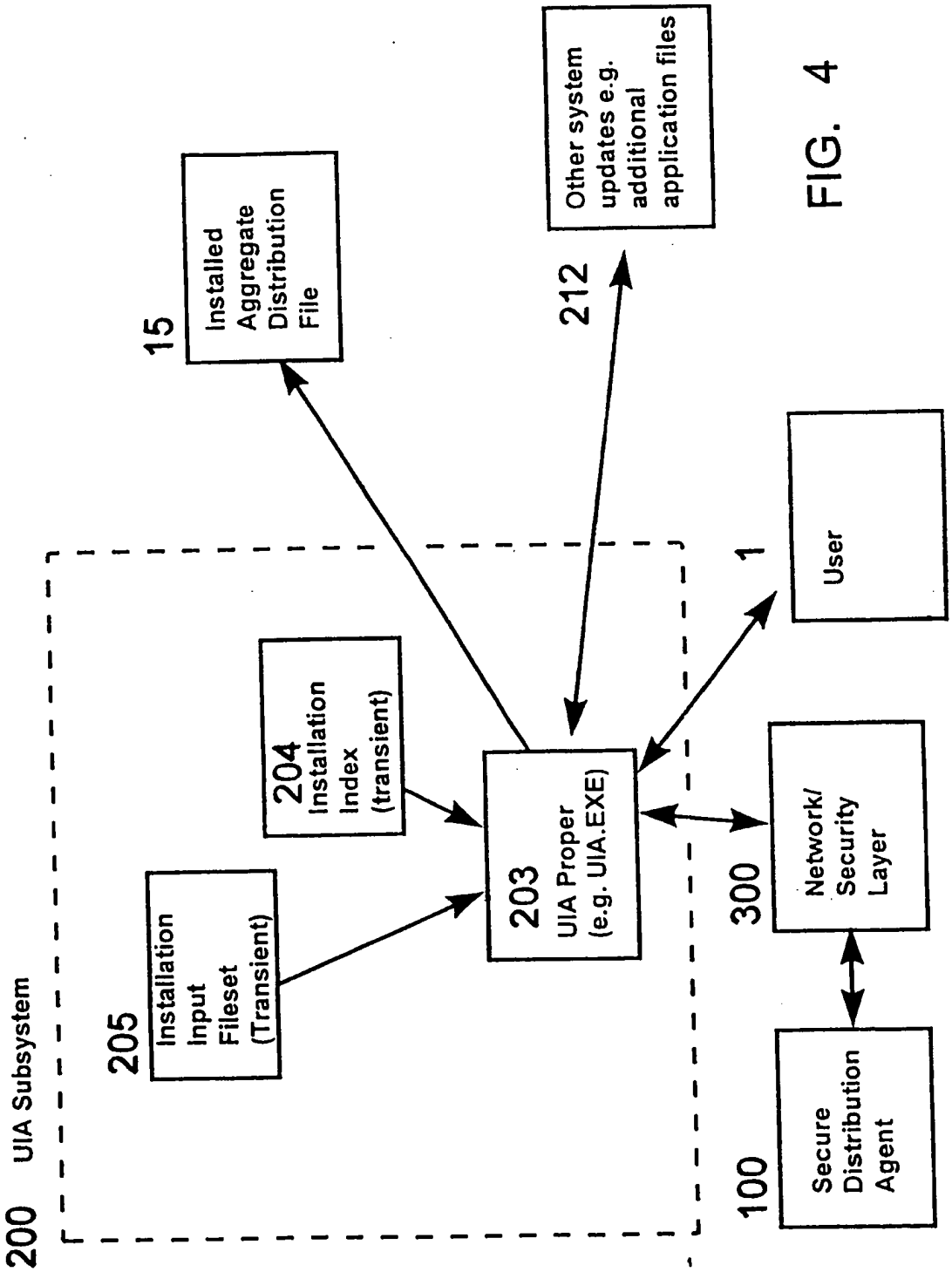

Other system updates e.g. additional application files
212
15
Installed Aggregate Distribution File
User
1
204
Installation Index (transient)
203
UIA Proper (e.g. UIA.EXE)
300
Network/ Security Layer
205
Installation Input Fileset (Transient)
200
UIA Subsystem
100
Secure Distribution Agent

FIG. 4

FIG. 5

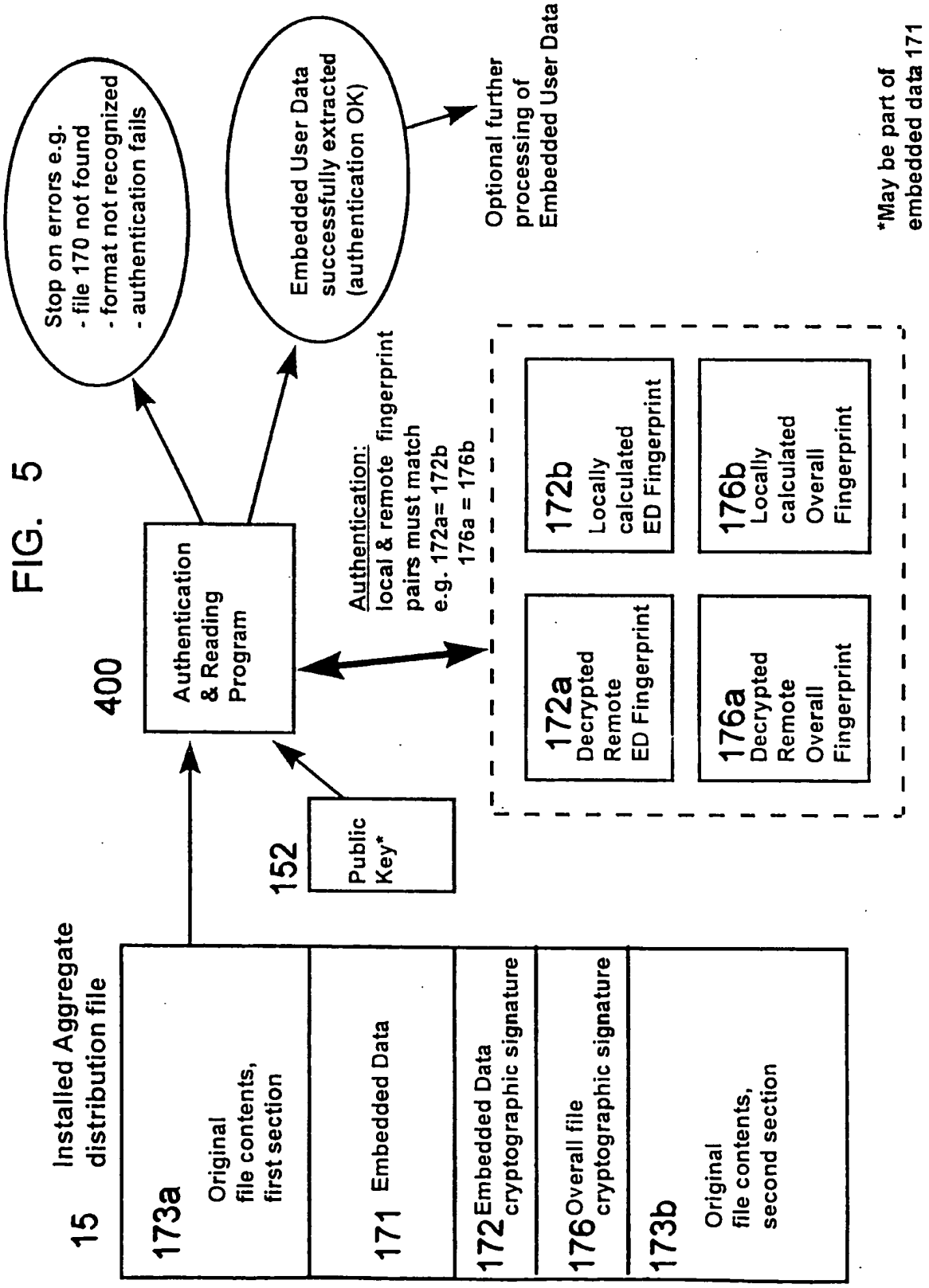

15 Installed Aggregate distribution file
173a Original file contents, first section
171 Embedded Data
172 Embedded Data cryptographic signature
176 Overall file cryptographic signature
173b Original file contents, second section
152 Public Key*
400 Authentication & Reading Program
Authentication: local & remote fingerprint pairs must match e.g. 172a= 172b 176a = 176b
172a Decrypted Remote ED Fingerprint
172b Locally calculated ED Fingerprint
176a Decrypted Remote Overall Fingerprint
176b Locally calculated Overall Fingerprint
Stop on errors e.g. - file 170 not found - format not recognized - authentication fails
Embedded User Data successfully extracted (authentication OK)
Optional further processing of Embedded User Data
*May be part of embedded data 171

FIG. 6

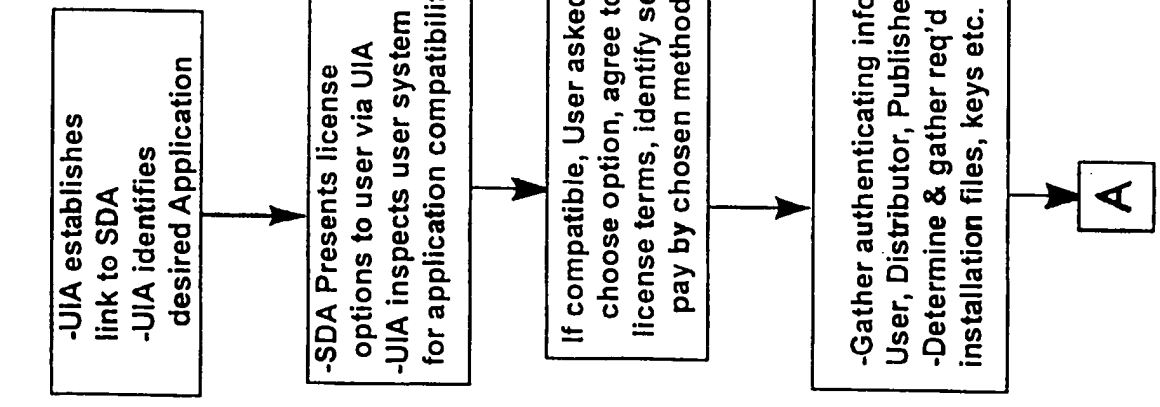

-UIA establishes link to SDA
-UIA identifies desired Application
-SDA Presents license options to user via UIA
-UIA inspects user system for application compatibility
If compatible, User asked to choose option, agree to license terms, identify self, pay by chosen method.
-Gather authenticating info on User, Distributor, Publisher.
-Determine & gather req'd installation files, keys etc.
A
A
-Build application embedded data, encrypt with private key, embed in application main file.
-Build application certificate & embed in file also
- Send to UIA
-Commit Financial Transaction
- UIA automatically completes Application installation
Done

FIG. 7

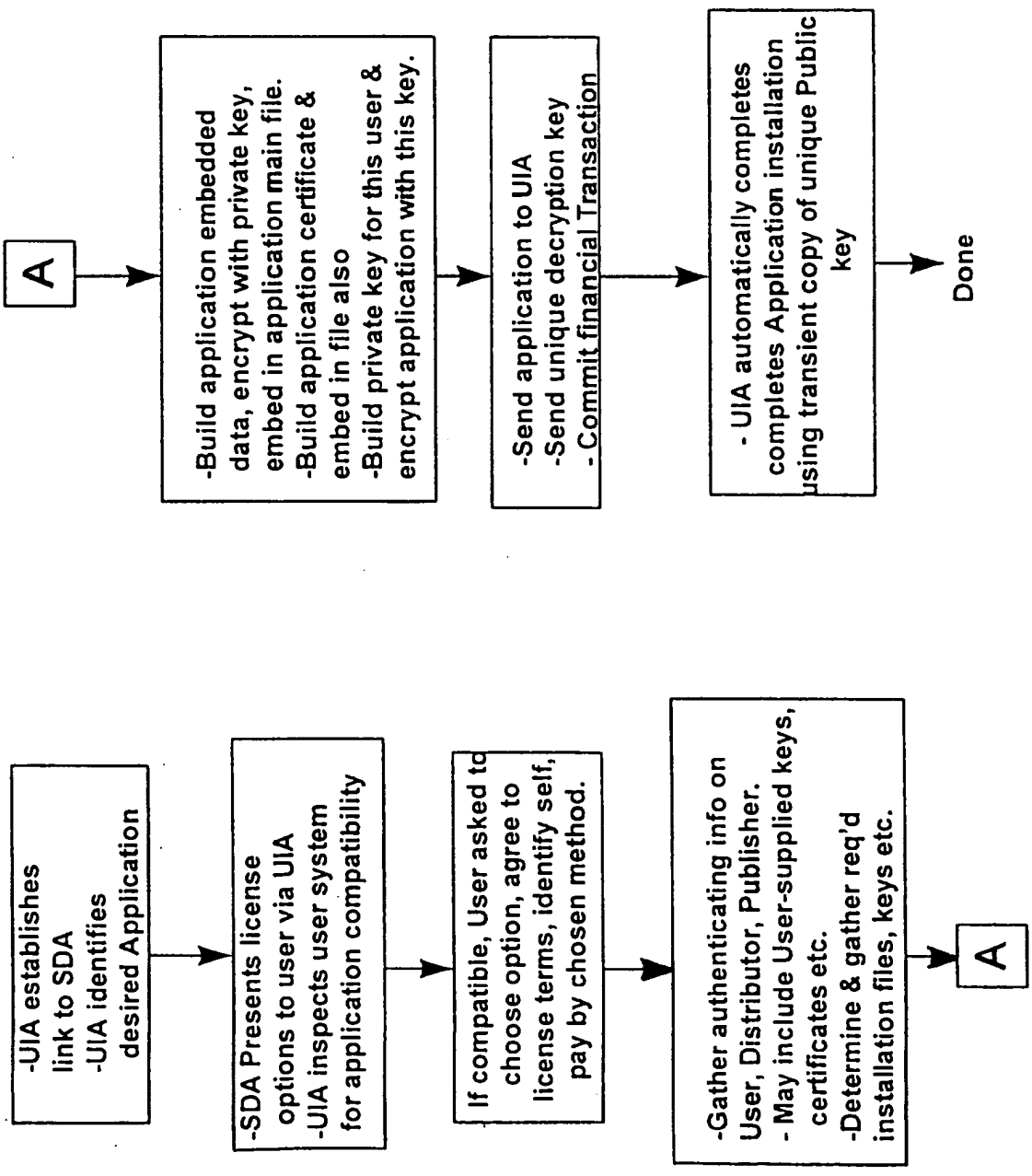
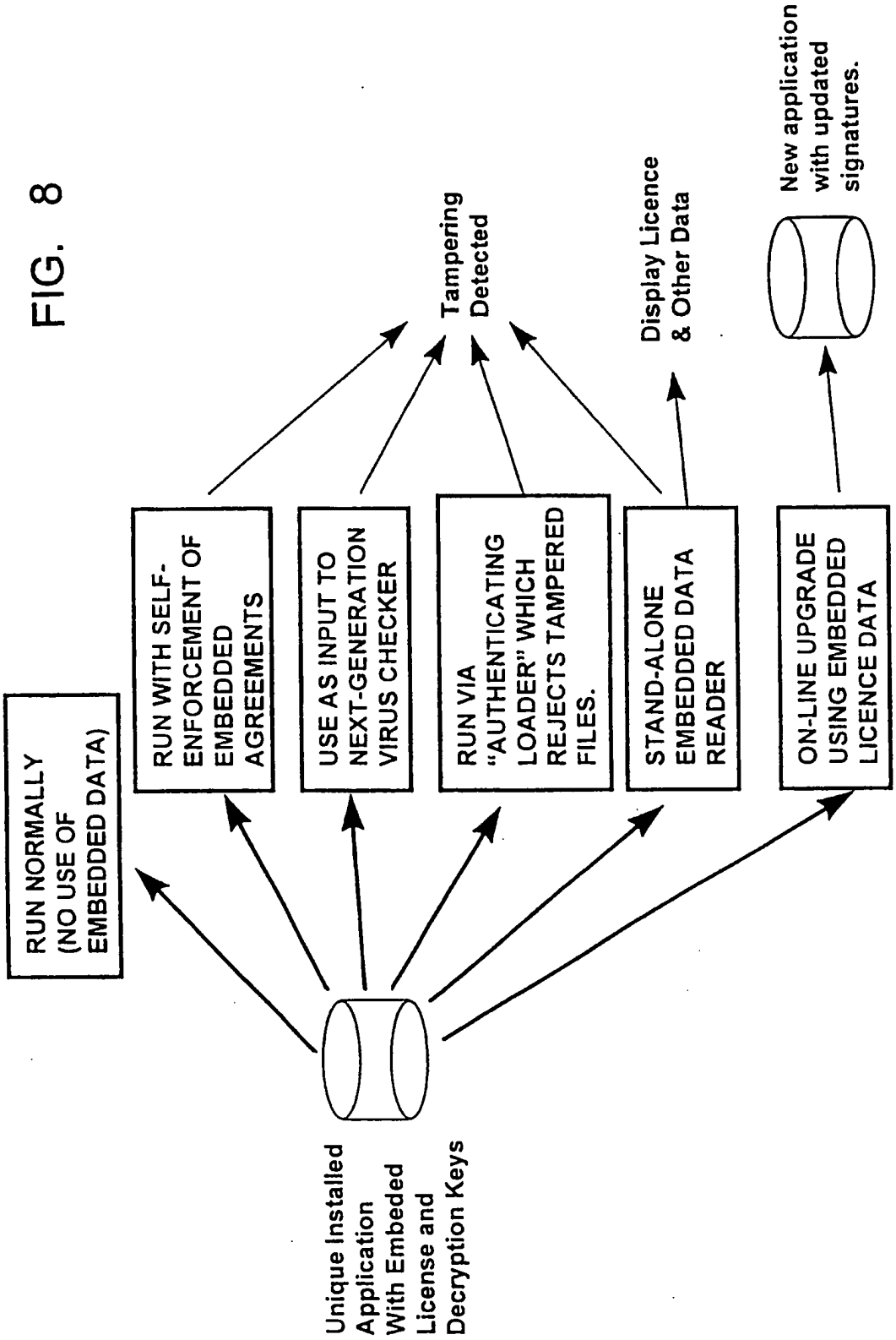

-UIA establishes link to SDA
-UIA identifies desired Application
-SDA Presents license options to user via UIA
-UIA inspects user system for application compatibility
If compatible, User asked to choose option, agree to license terms, identify self, pay by chosen method.
-Gather authenticating info on User, Distributor, Publisher.
- May include User-supplied keys, certificates etc.
-Determine & gather req'd installation files, keys etc.
A
A
-Build application embedded data, encrypt with private key, embed in application main file.
-Build application certificate & embed in file also
-Build private key for this user & encrypt application with this key.
-Send application to UIA
-Send unique decryption key
- Commit financial Transaction
- UIA automatically completes completes Application installation using transient copy of unique Public key
Done

FIG. 8


Unique Installed Application With Embeded License and Decryption Keys
RUN NORMALLY (NO USE OF EMBEDDED DATA)
RUN WITH SELF-ENFORCEMENT OF EMBEDDED AGREEMENTS
USE AS INPUT TO NEXT-GENERATION VIRUS CHECKER
RUN VIA "AUTHENTICATING LOADER" WHICH REJECTS TAMPERED FILES.
STAND-ALONE EMBEDDED DATA READER
ON-LINE UPGRADE USING EMBEDDED LICENCE DATA
Tampering Detected
Display Licence & Other Data
New application with updated signatures.

INTERNATIONAL SEARCH REPORT

Int tional Application No
PCT/CA 98/00241

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F1/00

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)
IPC 6 G06F

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 practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 686 906 A (SUN MICROSYSTEMS INC) 13 December 1995	1-6,8, 10-13, 15-17
A	see column 4, line 1 - column 5, line 5; figures 4,6A,6B	7,9,14

A	LEIN HARN ET AL: "A SOFTWARE AUTHENTICATION SYSTEM FOR INFORMATION INTEGRITY" COMPUTERS & SECURITY INTERNATIONAL JOURNAL DEVOTED TO THE STUDY OF TECHNICAL AND FINANCIAL ASPECTS OF COMPUTER SECURITY, vol. 11, no. 8, 1 December 1992, pages 747-752, XP000332279 see page 750, left-hand column, paragraph 2 see page 750, left-hand column, paragraph 8 - right-hand column, paragraph 5	1-17

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in 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 document but 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 publicationdate 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 theinternational search	Date of mailing of the international search report
8 July 1998	15/07/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Moens, R

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/CA 98/00241

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 509 074 A (CHOUDHURY ABHIJIT K ET AL) 16 April 1996 cited in the application see abstract	1-17
A	EP 0 717 337 A (IBM) 19 June 1996 see column 1, line 1 - column 2, line 40	1,8,12, 15-17

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ional Application No
PCT/CA 98/00241

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0686906 A	13-12-1995	US 5724425 A	03-03-1998
		JP 8166879 A	25-06-1996
US 5509074 A	16-04-1996	CA 2137065 A	28-07-1995
		EP 0665486 A	02-08-1995
		JP 7239828 A	12-09-1995
EP 0717337 A	19-06-1996	JP 8221268 A	30-08-1996
		US 5745678 A	28-04-1998

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 May 2001 (25.05.2001)

PCT

(10) International Publication Number
WO 01/37209 A1

(51) International Patent Classification[7]: G06K 9/36, 9/46

(21) International Application Number: PCT/US00/30825

(22) International Filing Date:
8 November 2000 (08.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/438,666 12 November 1999 (12.11.1999) US

(71) Applicant: TERALOGIC, INC. [US/US]; 1240 Villa Street, Mountain View, CA 94041 (US).

(72) Inventors: CASTOR, Jon, S.; 2160 Stockbridge Avenue, Woodside, CA 94062 (US). CHUI, Charles, K.; 340 Olive Street, Menlo Park, CA 94025 (US).

(74) Agents: WILLIAMS, Gary, S. et al.; Pennie & Edmonds LLP, 1155 Avenue of the Americas, New York, NY 10036 (US).

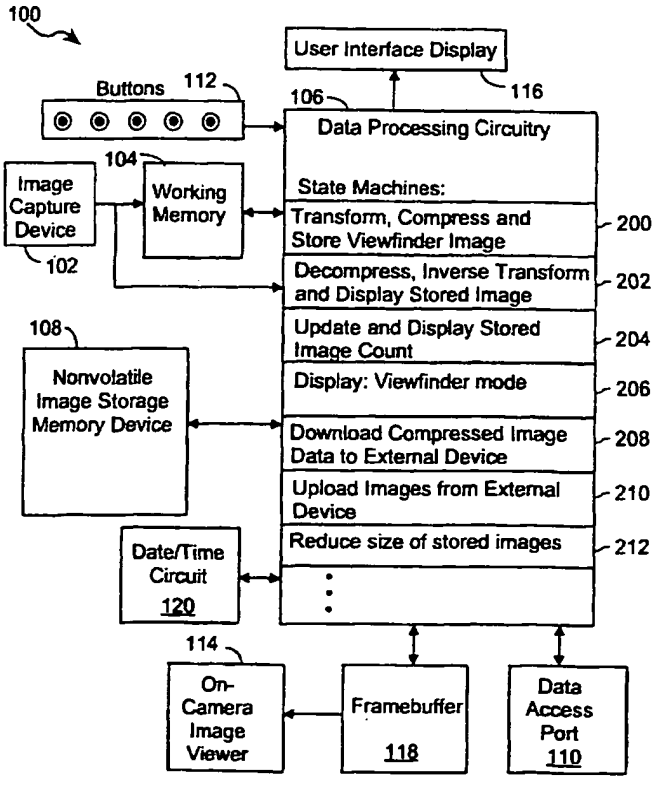
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

[Continued on next page]

(54) Title: PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD



(57) Abstract: An image processing system (100) stores image files in a memory device (108) at a number of incremental quality levels. Each image file has an associated image quality (that is fidelity or resolution) level corresponding to a quality level at which the corresponding image has been encoded. The images are initially encoded by applying a predefined transform, such as a DCT transform or wavelet-like transform (200), to image data received from an image capture mechanism (102) and then applying a data compression method to the transform data (200). The image is regenerated by successively applying a data decompression method and an inverse transform to an image file (202). Image file size reduction circuitry (212) and one or more state machines are used to lower the quality level of a specified one of the image files, including circuitry for extracting a subset of the data in the specified image file and forming a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

WO 01/37209 A1

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PICTURE AND VIDEO STORAGE MANAGEMENT SYSTEM AND METHOD

The present invention relates generally to the storage of still and video images in an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, and particularly to a system and method for storing images at different image quality levels and for enabling users to dynamically compress high quality images into lower quality images in order to free up storage for the storage of additional images.

BACKGROUND OF THE INVENTION

Digital cameras typically include either permanent internal storage, and/or removable storage, for storing images. For instance, many digital cameras use removable flash memory cards for storing images. Each flash memory card has a fixed amount of memory, and when the memory card is full it can be removed from the camera and replaced by another flash memory card. In addition, digital cameras typically have a built-in viewer that enables the user to review the images stored on the flash memory card (and/or in the internal memory) and to delete user specified ones of the stored images. Deleting stored images obviously creates room for storing additional images.

When a digital camera user is "in the field" he/she generally has a limited amount of image storage on hand. If all the available image storage is full, the user has the choice of either not taking any additional pictures, or of deleting pictures from the image storage devices on hand to make room for new images. While this is actually one level better than the situation with film cameras, in which the user is simply out of luck when all the available film has been used, it is the premise of the present invention that the current image storage limitations of digital cameras are caused, in part, by failure to fully exploit the advantages of having images stored in digital format.

Similar storage vs. image quality considerations also apply to digitally encoded video frames. In particular, for any given amount of storage space, such as in a digital video camera, the goal

is to retain the best image quality for the amount of storage required for a given number of video frames. Current devices allow the user to select image quality prior to capturing a digital video image, but do not enable the user to effectively manage the storage space in the video camera with respect to video sequences already taken, other than by deletion.

It is an object of the present invention to provide a digital camera or digital video camera, or other constrained storage device, that can store images at a plurality of image quality (i.e., fidelity or resolution) levels and furthermore can reduce images initially stored at a first image quality level to a lower image quality level so as to reduce the amount of storage occupied by those images.

It is also an object of the present invention to provide space efficient and computationally efficient image and video handling mechanisms for other applications, including network connected image and video libraries, Internet web browsers executed by client computers coupled to server computers, cable television set top boxes having video storage capabilities, and so on.

SUMMARY OF THE INVENTION

In summary, the present invention is an image processing device or system, such as a digital camera or digital video camera or a computer based image storage system, that can store images at a number of different image quality levels. The image processing device includes a memory device and image management logic. The memory device stores image files that each represent a respective image, each image file having an associated image quality level corresponding to a quality level at which the corresponding image has been encoded. The image management logic includes data processing circuitry and state machines for storing and processing image data received from an image capture mechanism. More specifically, the image management logic includes image processing circuitry and one or more state machines for applying a predefined transform, such as a wavelet-like 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 an image file having an associated image quality level.

The image management logic also includes image reconstruction circuitry and one or more state machines for successively applying a data decompression method and an inverse transform to a specified one of the image files so as to generate a reconstructed image suitable for display on an image viewer.

Further, the image management logic includes image file size reduction circuitry and one or more state machines for reducing the size of an image file while minimizing the reduction in image quality level. This circuitry extracts a subset of the data in the specified image file and forms a lower quality version of the specified image file that occupies less space in the memory device than was previously occupied by the specified image data structure. As a result, the amount of space occupied by image files in the memory device can be reduced so as to make room for the storage of additional image files or to allow more rapid transmission in a restricted bandwidth environment.

In a preferred embodiment, the image transform data in an image file is 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. To generate a lower quality image file, the image size reduction circuitry extracts a portion of the image file that excludes the image transform data for at least one bit plane and replaces the image file with an image file containing the extracted portion. Further, the image data is also 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. The image size reduction circuitry can also generate a lower quality image file by extracting a portion of the image file that excludes the image transform data for at least one transform layer and replaces the image file with an image file containing the extracted portion.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

Fig. 1 is a block diagram of a digital camera in accordance with an embodiment of the present invention.

Fig. 2 depicts an image data array divided into a set of smaller analysis arrays for purposes of encoding and data compression.

Fig. 3 schematically depicts the process of transforming a raw image data array into a transform image array and compressing the transform image array into a compressed image file.

Figs. 4A and 4B depict image storage data structures.

Fig. 5 is a conceptual flow chart depicting changes in the state of a digital camera as various operations are performed.

Fig. 6 is a conceptual flow chart depicting changes in the state of a video image sequence as the video image sequence is encoded and compressed.

Figs. 7, 8 and 9 show data structures used in one particular embodiment for video image sequence encoding and compression.

Fig. 10 is a conceptual diagram of an Internet server and client devices that utilize the image or and video image compressing and management features of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In some image processing systems, an image can be stored at a number of discrete resolution levels, typically with each resolution level differing from its "neighbors" by a resolution factor of four. In other words, if the highest resolution representation (at resolution level 1) of the image contains X amount of information, the second resolution level representation contains (for example) X/4 amount of information, the third resolution level representation contains X/16 amount of information, and so on. Thus, an image's "resolution" typically means the

amount of image information, and in the context of digital image processing systems is often expressed in terms of the number of distinct pixel elements. The number of resolution levels and the particular amount of information reduction from one level to the next may vary considerably from one system to another. Further, the present invention would be equally applicable to systems having a continuous range of resolution levels.

Another concept concerning image quality is "fidelity." The fidelity of an image can be compromised even if its resolution, in terms of the number of pixels in the image, is unchanged. An image's fidelity can be reduced by reducing the number of bits used to represent the image data. For instance, if the transform coefficients used to represent an image are represented at full fidelity using 12 bits, and then the number of bits used to represent transform coefficients is reduced to 11, the fidelity of the resulting image will be reduced. In other words, the quality of the image reconstructed from the lower fidelity data will be a little less sharp than an image reconstructed from higher fidelity data.

In this document, the terms "image quality" and "quality level" will be used in the general sense of image quality, encompassing both image "resolution" and image "fidelity." Thus, a reduction in an image's "image quality" from a top image quality level to a next highest image quality level might be accomplished either by reducing the image's resolution or by reducing its fidelity, or both. In other embodiments the terms "image quality" and "quality level" may be used to refer to other aspects of image quality as well.

Digital Camera Architecture

Referring to Fig. 1, there is shown an embodiment of a digital camera system 100 in accordance with the present invention. The digital camera system 100 includes an image capture device 102, such as a CCD or CMOS sensor array or any other mechanism suitable for capturing an image as an array of digitally encoded information. Thus the image capture device is assumed to include analog to digital conversion (ADC) circuitry for converting analog image information into digital values.

A working memory 104, typically random access memory, receives digitally encoded image information from the image capture device 102. More generally, it is used to store a digitally

encoded image while the image is being transformed and compressed and otherwise processed by the camera's data processing circuitry 106. Memory 104 may be integrated on the same integrated circuit as other devices, or may be implemented using separate circuit(s).

The data processing circuitry 106 in one embodiment consists of hardwired logic and a set of state machines for performing a set of predefined image processing operations. In alternate embodiments the data processing circuitry 106 could be implemented in part or entirely using a fast general purpose microprocessor and a set of software procedures. However, at least using the technology available in 1999, it would be difficult to process and store full resolution images (e.g., full color images having 1280 x 840 pixels) fast enough to enable the camera to be able to take, say, twenty pictures per second, which is a requirement for some commercial cameras, as well as digital video cameras. In the future, general purpose microprocessors or general purpose image data microprocessors (e.g., single instruction multiple data (SIMD) processors) may be able to provide the fast image processing needed by digital cameras, in which case the data processing circuit 106 could be implemented using such a general purpose microprocessor or perhaps a hybrid processor system.

Each image, after it has been processed by the data processing circuitry 106, is typically stored as an "image file" in a nonvolatile memory storage device 108, typically implemented using "flash" (i.e., EEPROM) memory technology. The nonvolatile memory storage device 108 is preferably implemented as a removable memory card. This allows the camera's user to remove one memory card, plug in another, and then take additional pictures. However, in some implementations, the nonvolatile memory storage device 108 may not be removable, in which case the camera will typically have a data access port 110 to enable the camera to transfer image files to and from other devices, such as general purpose, desktop computers, computer systems and devices used to warehouse libraries of images, computer systems and devices used to store and distribute image files, and so on. Digital cameras with removable nonvolatile memory 108 may also include a data access port 110.

While the amount of storage in the nonvolatile image memory 108 will vary from one implementation to the next, such devices will typically have sufficient capacity to store 10 to 50 high quality images. Once the nonvolatile image memory 108 is full, if the file size reduction methodology of the present invention is not used, the only way the camera can be

used to take additional pictures is either by deleting images from the nonvolatile image memory 108 or by replacing the nonvolatile image memory 108 with another one. If neither of these options are feasible (e.g., because the user has filled all the memory cards he/she has on hand with images that he/she does not wish to delete), then no further pictures can be taken until a new memory device 108 is inserted or the stored images are transferred to an external device (if available).

The digital camera 100 includes a set of buttons 112 for giving commands to the camera. In addition to the image capture button, there will typically be several other buttons to enable the use to select the quality level of the next picture to be taken, to scroll through the images in memory for viewing on the camera's image viewer 114, to delete images from the nonvolatile image memory 108, and to invoke all the camera's other functions. Such other functions might include enabling the use of a flash light source, and transferring image files to and from a computer. In accordance with the present invention, the user selectable functions, selected by using the buttons 112, further include reducing the size of one or more of the image files stored in the nonvolatile image memory 108 so as to make room for the storage of additional images. The buttons in one embodiment are electromechanical contact switches, but in other embodiments at least some of the buttons may be implemented as touch screen buttons on a user interface display 116, or on the image viewer 114.

The user interface display 116 is typically implemented either (A) as an LCD display device separate from the image viewer 114, or (B) as images displayed on the image viewer 114. Menus, user prompts, and information about the images stored in the nonvolatile image memory 108 may be displayed on the user interface display 116, regardless of how that display is implemented.

After an image has been captured, processed and stored in nonvolatile image memory 108, the associated image file may be retrieved from the memory 108 for viewing on the image viewer. More specifically, the image file is converted from its transformed, compressed form back into a data array suitable for storage in a framebuffer 118. The image data in the framebuffer is displayed on the image viewer 114. A date/time circuit 120 is used to keep track of the current date and time, and each stored image is typically date stamped with the date and time that the image was taken.

Image Data Structures

Referring to Fig. 2, in one embodiment the nonvolatile image memory 108 stores a directory 130 that lists all the image files 132 stored in the memory 108. Preferably, the directory 130 contains information for each stored image file 132, such as the date and time the image was taken, the quality level of the image and the file's location in the memory 108.

To understand the image data structure stored in each image file, it is helpful to first understand how an image file is encoded. Referring to Fig. 3, a raw image data array 140, obtained from the digital camera's image capture mechanism 102 (Fig. 1), is treated as a set of non-overlapping "analysis arrays" 142 of a fixed size, such as 32 x 32, or 64 x 64 (or more generally $2^n \times 2^n$, for some integer value of n). A sufficient number of subarrays are used to cover the entire data array that is to be encoded, even if some of the subarrays overhang the edges of the data array. The overhanging portions of the subarrays are filled with zero data values during the data encoding process. In a preferred embodiment, the origin of the data array is the top left corner, the first coordinate used to identify data array positions is the "Y" axis or vertical coordinate, and the second coordinate used is the "X" axis or horizontal coordinate. Thus, a position of 0,64 indicates a pixel at the top vertical position of the array, 64 pixel positions over to the right from the array origin, while a position of 32,0 indicates a pixel on the left edge of the array, 32 pixel positions vertically down from the array origin.

An appropriate transform is applied to each of the analysis arrays 142, and then the resulting transform coefficients for each analysis array are quantized (e.g., divided by an appropriate value to generate integer valued, quantized transform coefficients) so as to generate a transformed image array 144. In one embodiment the transform applied to the raw image data is a wavelet-like transform. In other embodiments a DCT transform could be used (which is the type of transform used in current JPEG image encoding systems), or other types of wavelet or wavelet-like transforms could be used.

In this document, the terms "wavelet" and "wavelet-like" are used interchangeably. Wavelet-like transforms generally have spatial frequence characteristics similar to those of conventional wavelet transforms, and are losslessly reversible, but have shorter filters that are more computationally efficient.

In one embodiment the transformed image array 144 is generated by successive applications of a wavelet-like decomposition transform. A first application of the wavelet-like decomposition transform to an initial two dimensional array of "raw" image data generates four sets of coefficients, labeled LL, HL1, LH1 and HH1. Each succeeding application of the wavelet-like decomposition transform is applied only to the LL set of coefficients generated by the previous wavelet transformation step and generates four new sets of coefficients, labeled LL, HLx, LHx and HHx, where x represents the wavelet transform "layer" or iteration. After the last wavelet-like decomposition transform iteration only one LL set remains. The total number of coefficients generated is equal to the number of data samples in the original data array. The different sets of coefficients generated by each transform iteration are sometimes called layers. The number of wavelet transform layers generated for an image is typically a function of the resolution of the initial image. Performing five to seven wavelet transformation layers is fairly typical, but more or less may be used depending on such considerations as the size of the analysis arrays, the subject matter of the image, the data processing resources available for image compression, and the like.

For the purposes of explaining the operation of the image encoding and decoding operations of the present invention, the specific type of image transform used and the specific type of data quantization used to transform a raw image file 140 into a transformed image array 142 are not relevant and therefore are not further described herein. However, a preferred embodiment of the wavelet transform and data quantization methods are described in U.S. Patent No. 5,909518, "System and Method for Performing Wavelet and Inverse Wavelet Like Transformations of Digital Data," which is hereby incorporated by reference as background information.

Each transformed image array 144 is compressed and encoded using a sparse data encoding technique. In one embodiment, the method of compressing and encoding the analysis arrays is the method described in detail in U.S. patent application 08/858,035, filed May 16, 1997, entitled "System and Method for Scalable Coding of Sparse Data Sets," now U.S. Patent No. 5,949,911, which is hereby incorporated by reference as background information. The encoded image data for all the analysis arrays of the image are combined and stored as an image file 132.

Referring to Fig. 4A, the image file 132 includes header data 160 and a sequence of data structures 162, each representing one analysis array. The header data 160 indicates the size of the image file and the image file's quality level. The header data also includes a list of analysis array size values indicating the length of each of the analysis array data structures 162, thereby enabling fast indexing into the image data. Storing size values for the analysis arrays enables the camera's data processing circuitry 106 (Fig. 1) to locate the beginning of any analysis array data structure 162 without having to decode the contents of the earlier analysis arrays in the image file 132.

As shown in Fig. 4B, the encoded data 162 representing any one analysis array is stored in "bit layer order". For each analysis array, the encoding procedure determines the most significant non-zero bit in the data to be encoded, which is herein called the y^{th} bit. The value of y is determined by computing the maximum number of bits required to encode the absolute value of any data value in the analysis array. In particular, y is equal to $int(\log_2 V) + 1$, where V is the largest absolute value of any element in the analysis array, and "int()" represents the integer portion of a specified value.

The encoded data 162 representing one analysis array includes (A) header data 170 indicating the maximum number of bits required to encode the absolute value of any data value in the analysis array, and (B) a sequence of data structures 172, each representing one bit plane of the elements in the analysis array. The x^{th} bit plane of the analysis array is the x^{th} bit of the absolute value of each of the elements in the analysis array. A sparse data encoding technique is used so that it takes very little data to represent a bit plane that contains mostly zero values. Typically, higher frequency portions of the transformed, quantized image data will contain more zero values than non-zero values, and further most of the non-zero values will have relatively small absolute value. Therefore, the higher level bit planes of many analysis arrays will be populated with very few non-zero bit values.

In an alternate embodiment, the data structure shown in Fig. 4A is modified slightly. In particular, to facilitate fast extraction of lower-resolution image data from an image file, the boundaries of the analysis arrays are adjusted, if necessary, so as to coincide precisely with the boundaries between the wavelet transform regions shown in Fig. 3 (e.g., the boundary between HL2 and HL1). If the size of the initial image array is not equal to an integer number of

analysis arrays (i.e., if either the height or width of the image array is not an integer multiple of 2^n, where the size of each analysis array is 2^n x 2^n for an integer value of n), at least some of the boundaries between wavelet transform regions will fall in the middle of the analysis regions. For example, for a 800 x 600 pixel image, the LL region might have a size of 50 x 38. If the wavelet transform coefficients are encoded in units of analysis regions of size 32 x 32, the LL region will be encoded in four analysis regions, three of which would normally contain data for neighboring wavelet transform regions. In this alternate embodiment, each analysis array that overlaps a border between wavelet transform regions is replaced by two or four analysis regions (depending on whether the analysis array overlaps one or two region boundaries), with zero values being stored in the appropriate locations so that each analysis array contains data from only one wavelet transform region. The analysis arrays are still stored in "origin sorted order" in the image file 132, with the "origin" now being defined as the coordinate of the coefficient closest to the upper left corner of the analysis array that has not been overwritten with zero values.

In another alternate embodiment, a different transform than the wavelet-like transform could be used, but the resulting image data would still be stored in bit plane order. For instance, a DCT transform could be used.

In some embodiments of the present invention, the raw image array received from the digital camera's image capture mechanism may first be divided into "analysis arrays" and then transformed and quantized. Further, the analysis arrays may each be a thin horizontal strip of the image array. That is, each analysis array may extend the full width of the image array, but have a height of only a few (e.g., 4 to 16) image elements. In yet another embodiment, the image array might not be divided into analysis arrays at all.

Generally, in all embodiments described above, the compressed encoded image data is stored in bit plane order. The reason that bit plane ordered storage is favored is that it makes gradual fidelity reduction very easy: to reduce the fidelity of an image file by a minimum amount, the data for the lowest level bit plane in the file is discarded and the remaining image data is retained, resulting in a smaller file with one bit plane less fidelity.

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 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.

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.

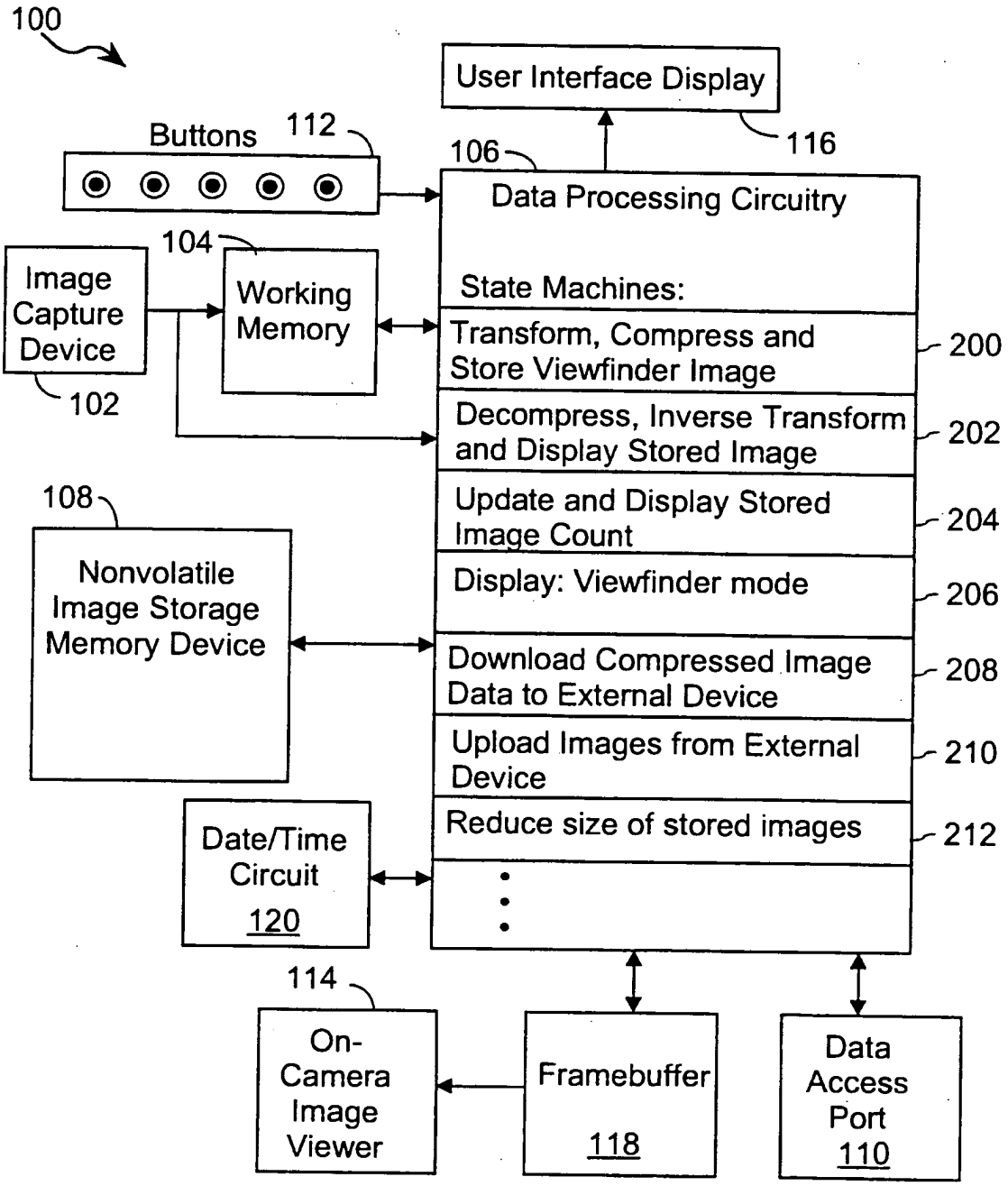

100
User Interface Display
116
Buttons
112
106
Data Processing Circuitry
State Machines:
104
Image Capture Device
Working Memory
102
Transform, Compress and Store Viewfinder Image
200
Decompress, Inverse Transform and Display Stored Image
202
Update and Display Stored Image Count
204
108
Nonvolatile 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

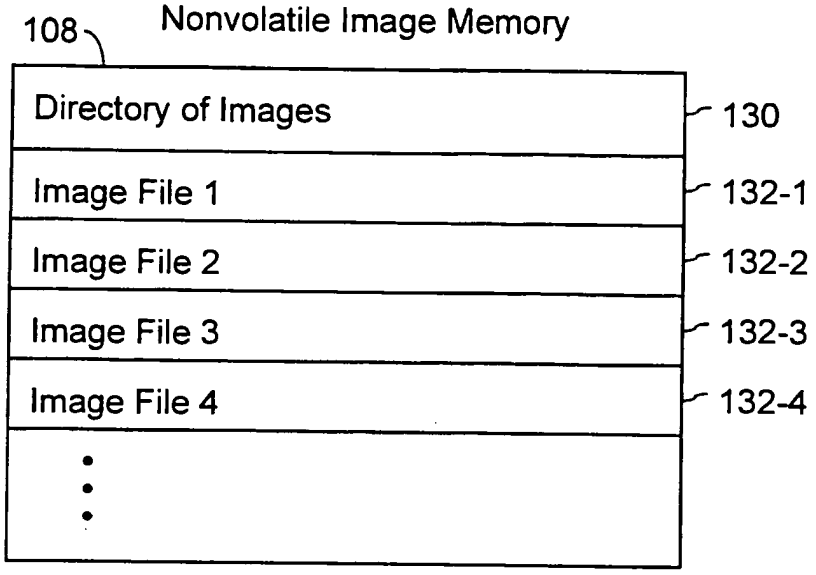

108
Nonvolatile Image Memory
Directory of Images
130
Image File 1
132-1
Image File 2
132-2
Image File 3
132-3
Image File 4
132-4

FIG. 2

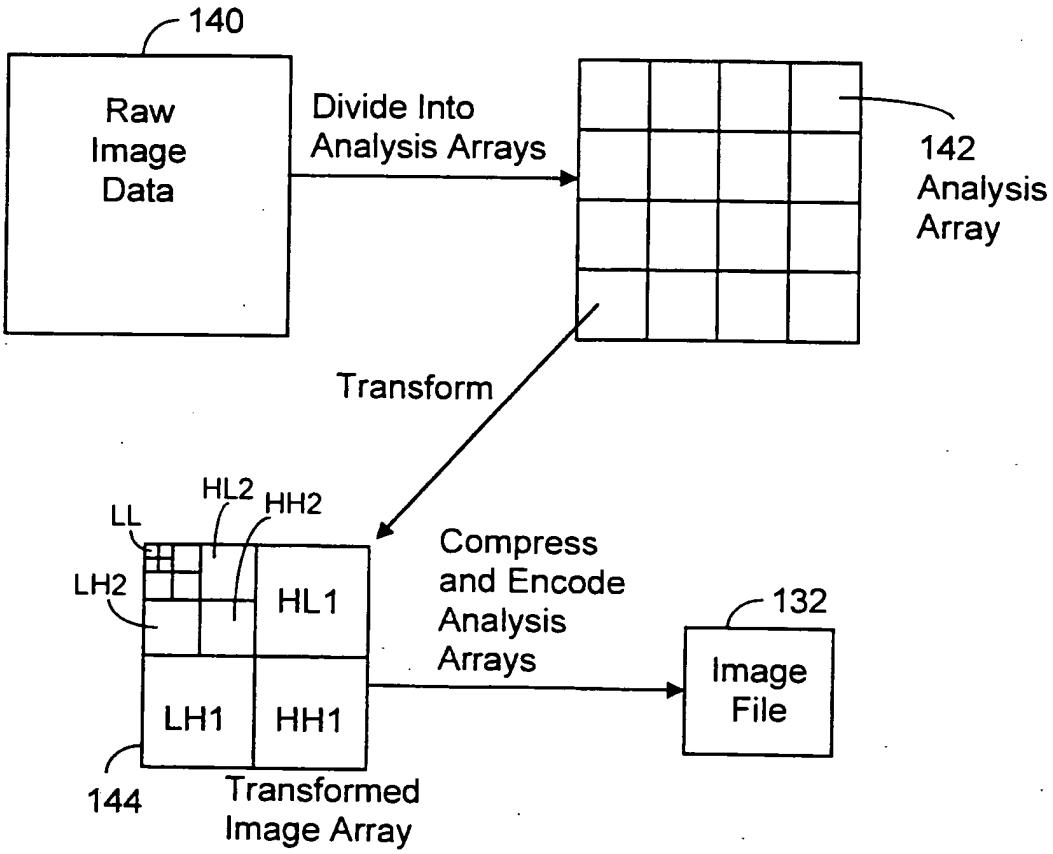

140
Raw Image Data
Divide Into Analysis Arrays
142 Analysis Array
Transform
LL
HL2
HH2
LH2
HL1
LH1
HH1
144
Transformed Image Array
Compress and Encode Analysis Arrays
132
Image File

FIG. 3

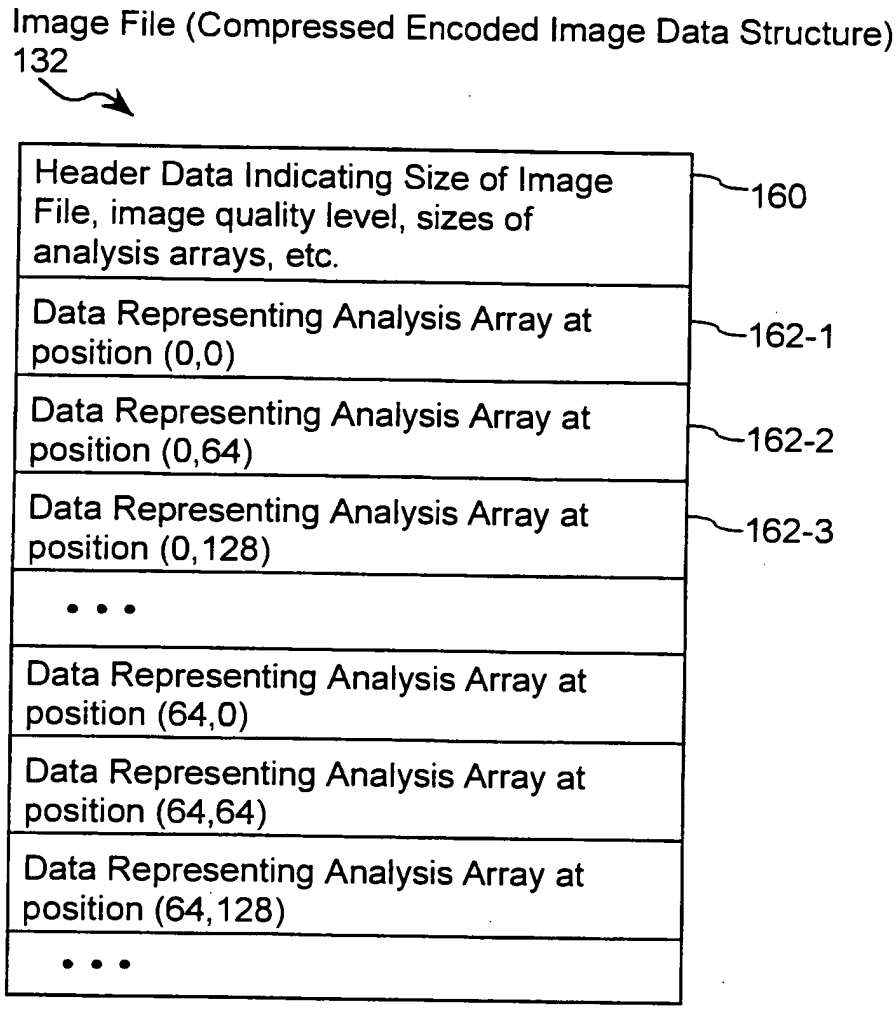


Image File (Compressed Encoded Image Data Structure)
132
Header Data Indicating Size of Image File, image quality level, sizes of analysis arrays, etc.
160
Data Representing Analysis Array at position (0,0)
162-1
Data Representing Analysis Array at position (0,64)
162-2
Data Representing Analysis Array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

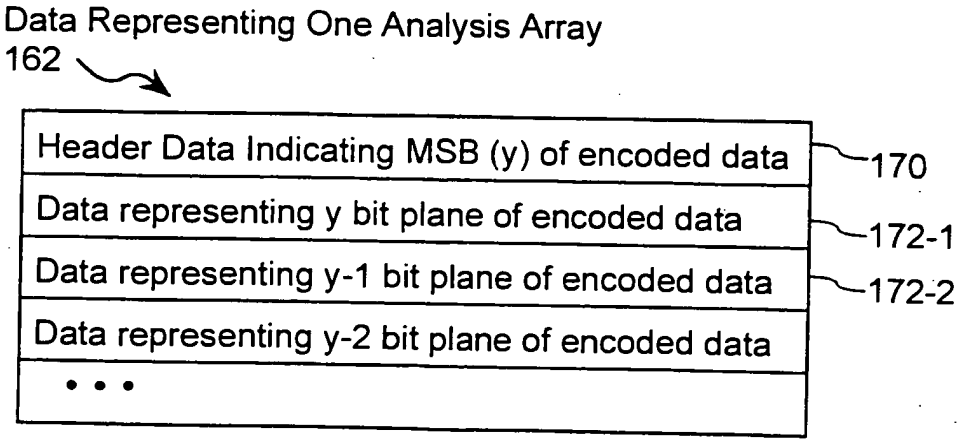

Data Representing One Analysis Array
162
Header Data Indicating MSB (y) of encoded data
170
Data representing y bit plane of encoded data
172-1
Data representing y-1 bit plane of encoded data
172-2
Data representing y-2 bit plane of encoded data
• • •

FIG. 4B

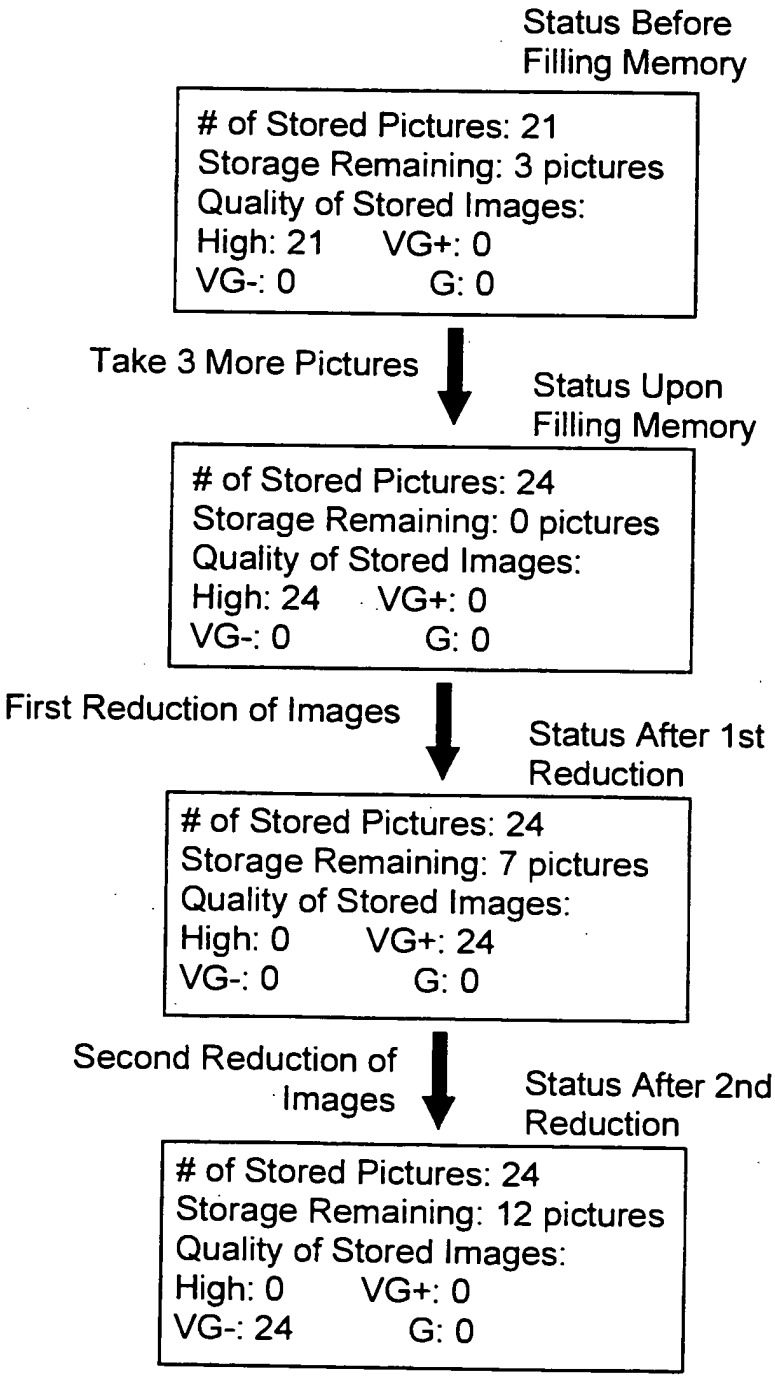

Status Before Filling Memory
# of Stored Pictures: 21
Storage Remaining: 3 pictures
Quality of Stored Images:
High: 21 VG+: 0
VG-: 0 G: 0
Take 3 More Pictures
Status Upon Filling Memory
# of Stored Pictures: 24
Storage Remaining: 0 pictures
Quality of Stored Images:
High: 24 VG+: 0
VG-: 0 G: 0
First 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

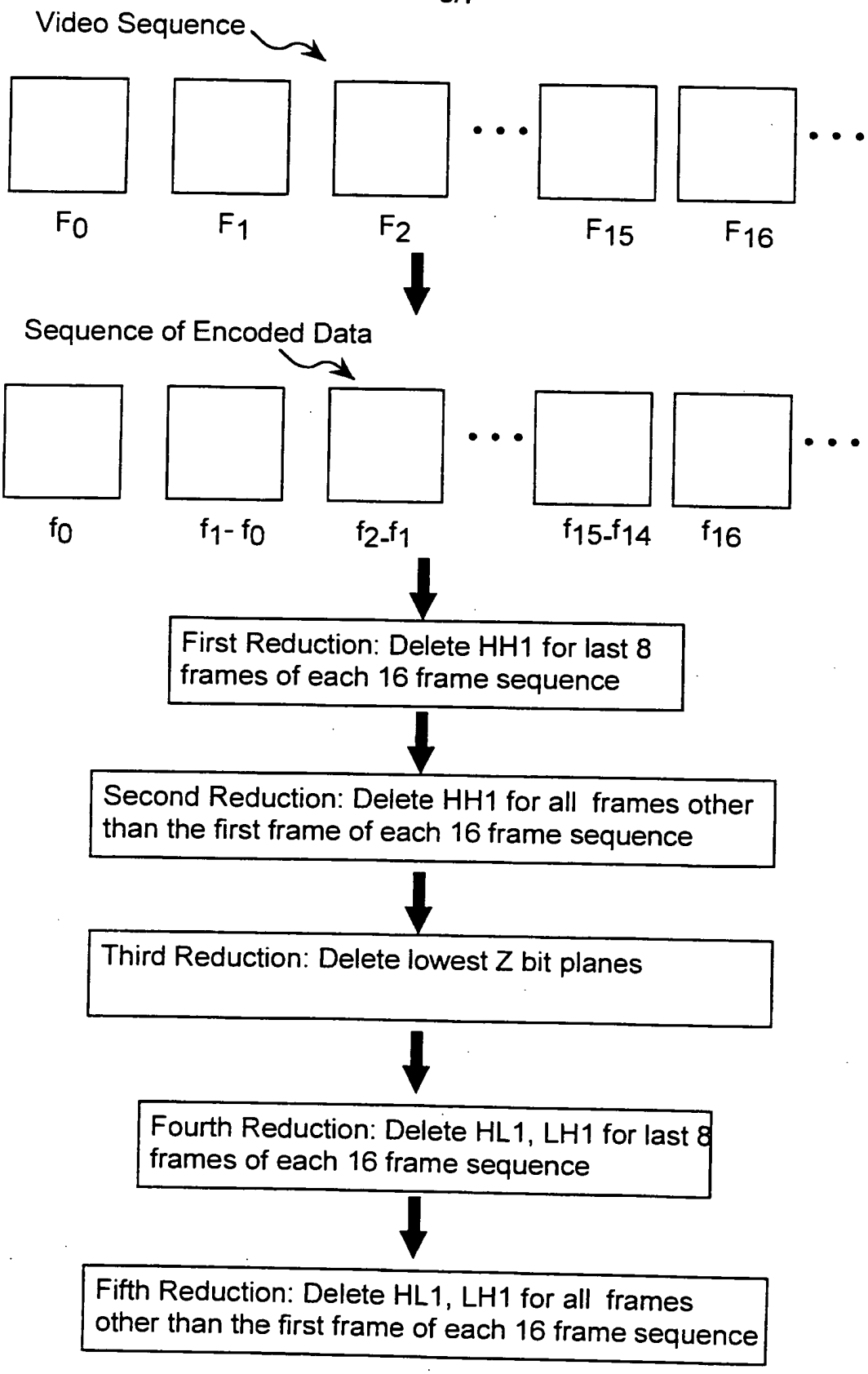

Video Sequence
F0
F1
F2
F15
F16
Sequence of Encoded Data
f0
f1- f0
f2-f1
f15-f14
f16
First Reduction: Delete HH1 for last 8 frames of each 16 frame sequence
Second Reduction: Delete HH1 for all frames other than the first frame of each 16 frame sequence
Third Reduction: Delete lowest Z bit planes
Fourth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

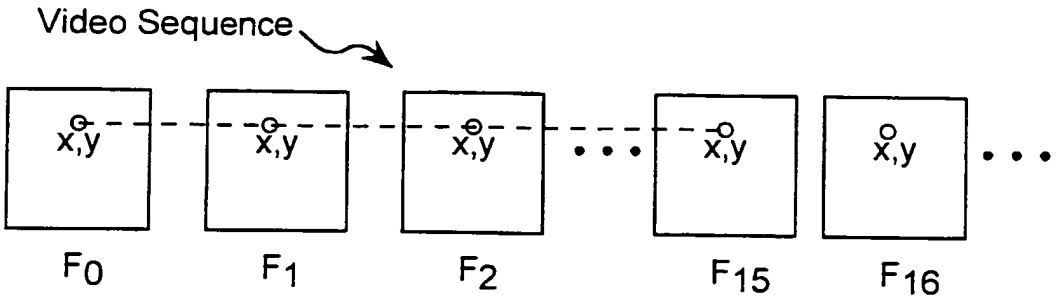


FIG. 7

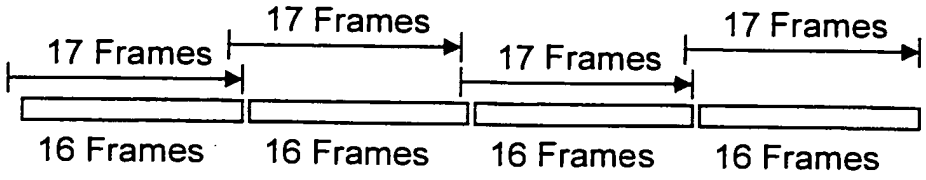


FIG. 8

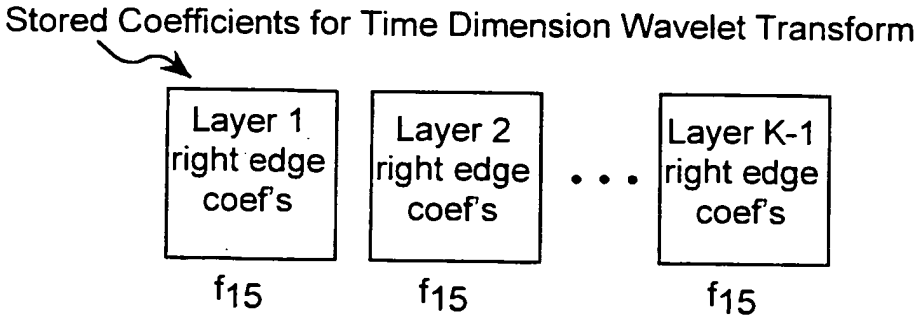


FIG. 9

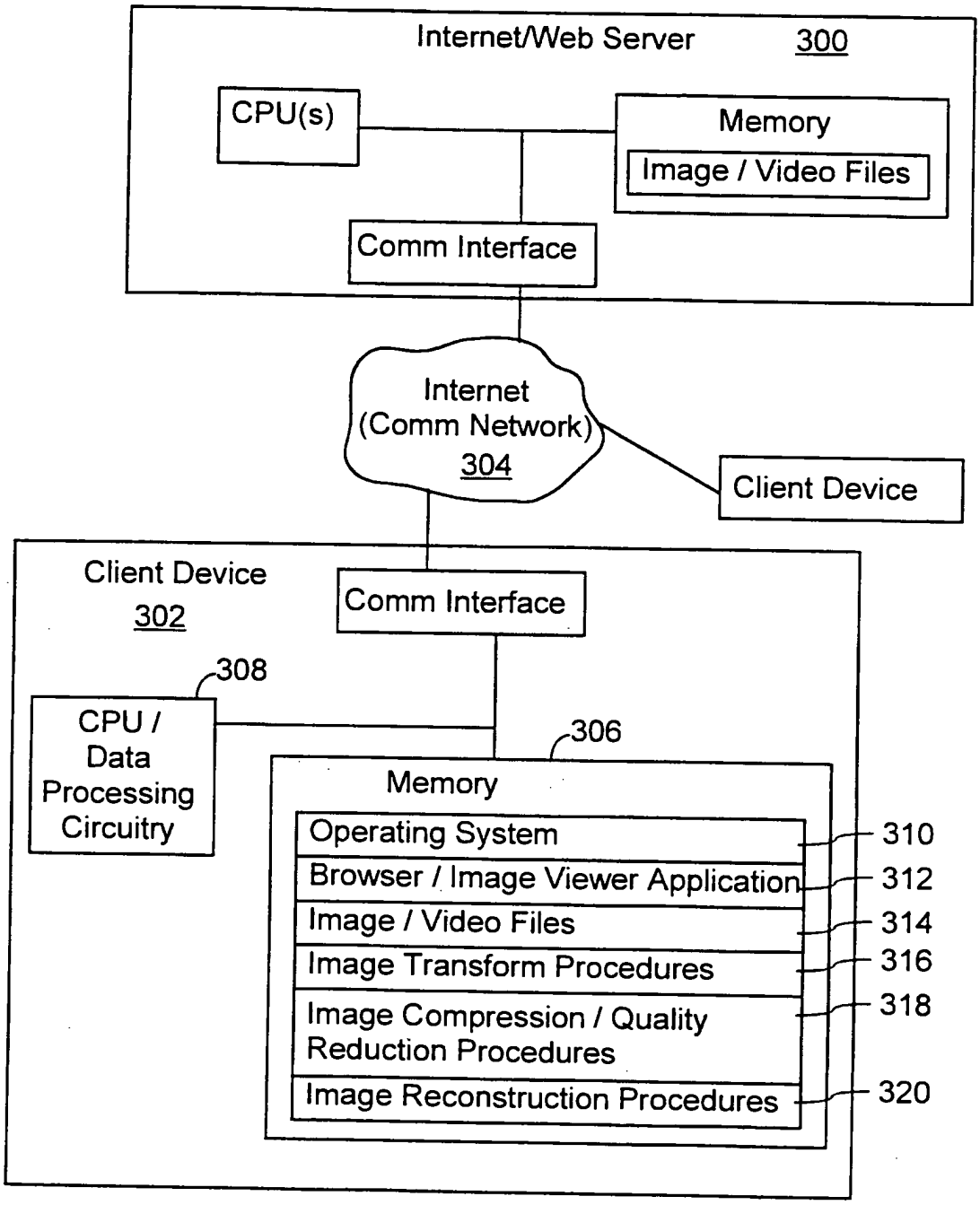

Internet/Web Server 300
CPU(s)
Memory
Image / Video Files
Comm Interface
Internet
(Comm Network)
304
Client Device
Client Device
302
Comm Interface
308
CPU /
Data
Processing
Circuitry
306
Memory
Operating System
Browser / Image Viewer Application
Image / Video Files
Image Transform Procedures
Image Compression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

FIG. 10

INTERNATIONAL SEARCH REPORT

Internal 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[illegible]onal 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: **0 529 261 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)

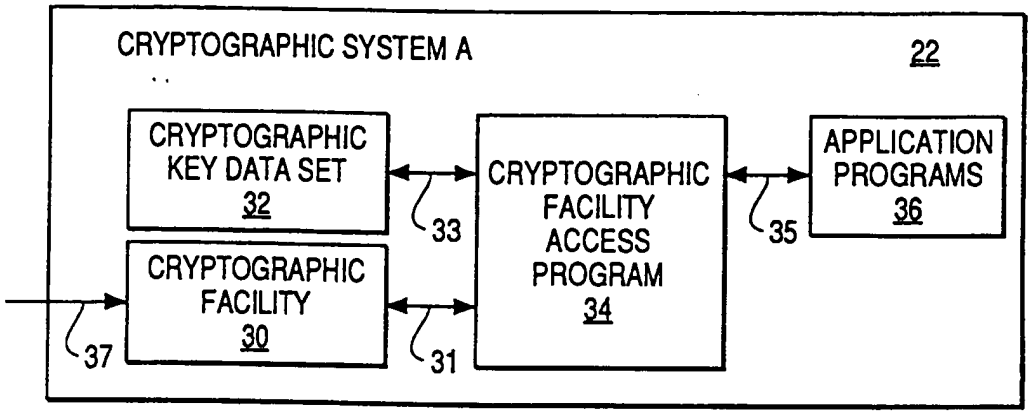
(54) **A hybrid public key algorithm/data encryption algorithm key distribution method based on control vectors.**

(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

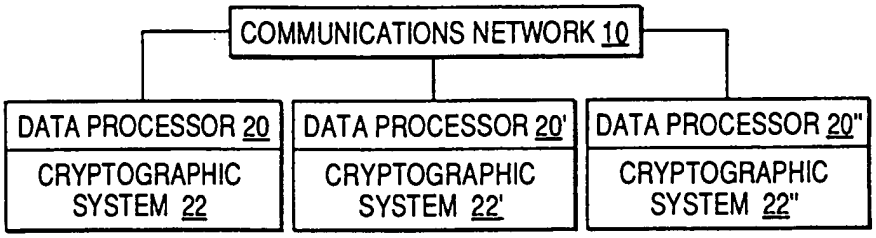


FIG. 2

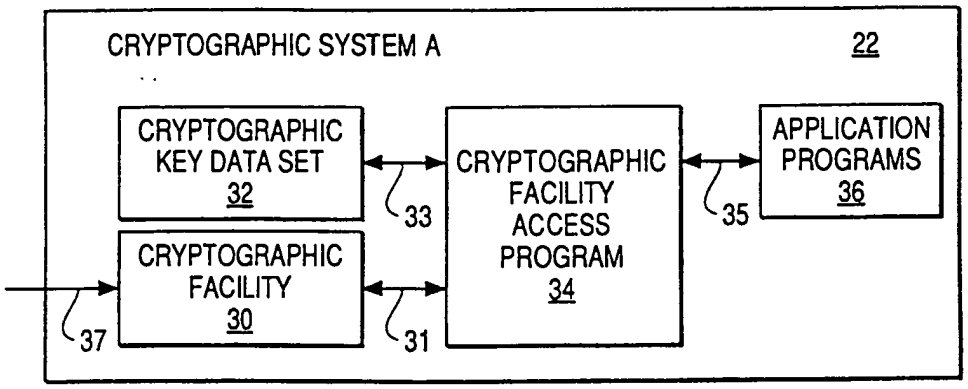


FIG. 3

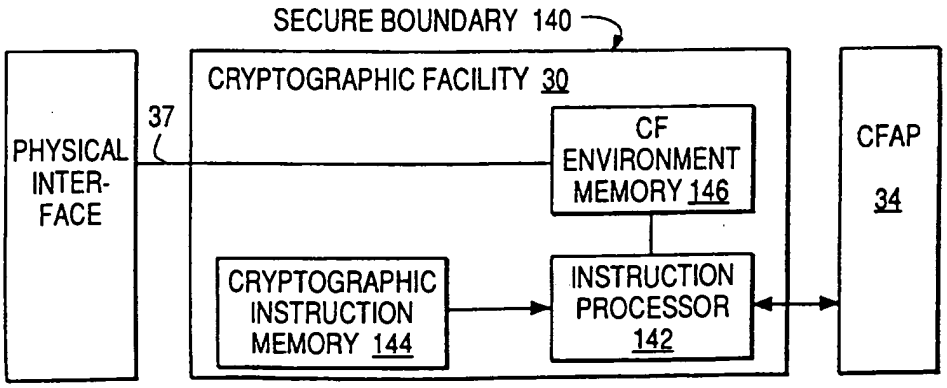


FIG. 4

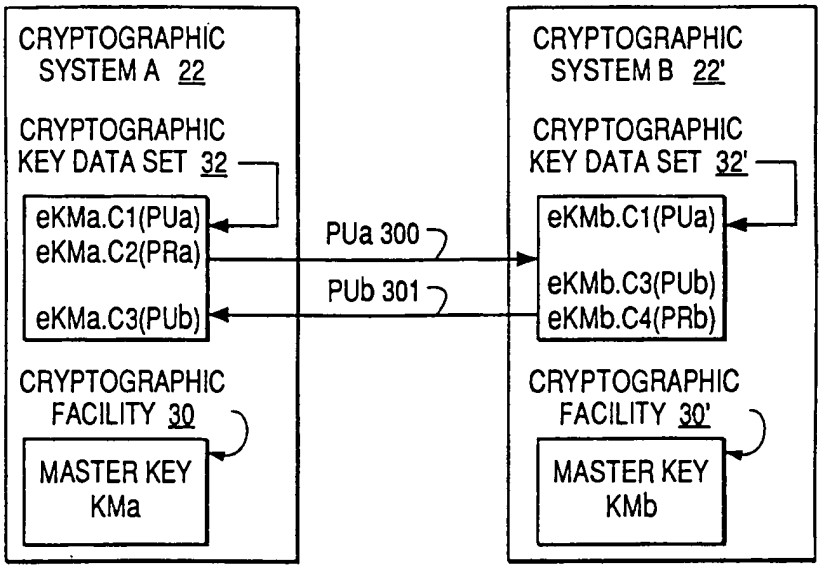


FIG. 5

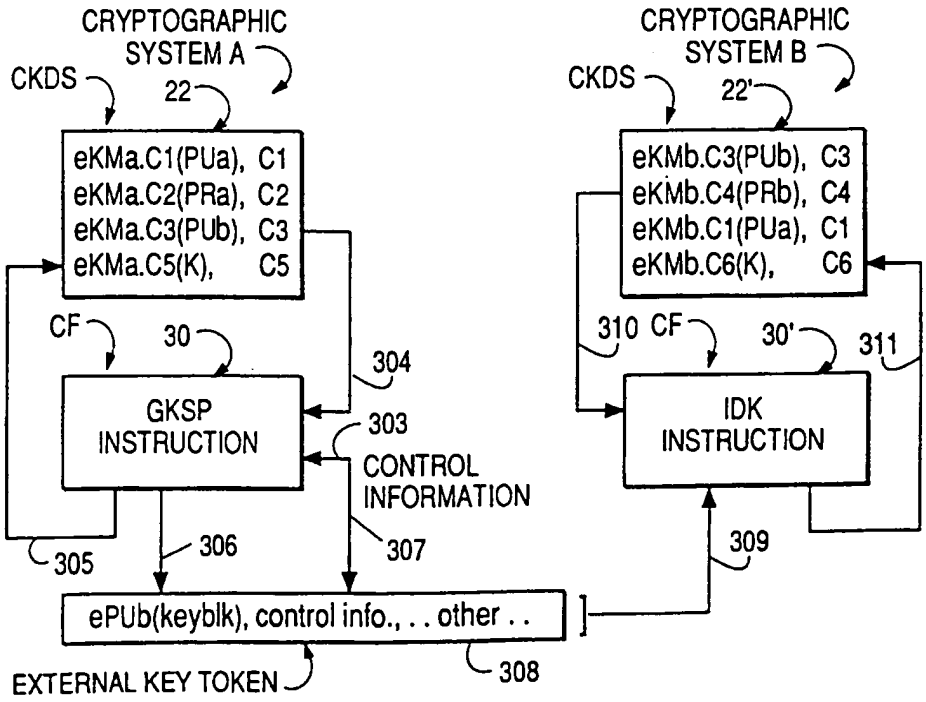


FIG. 6

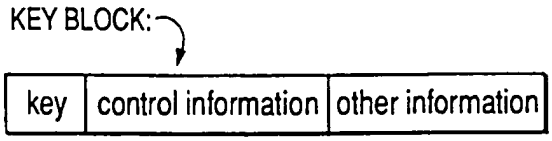


FIG. 7

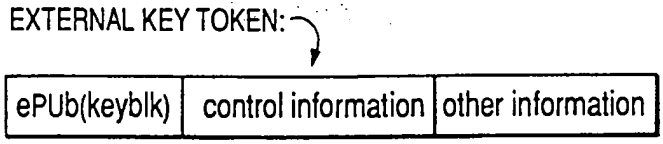


FIG. 8

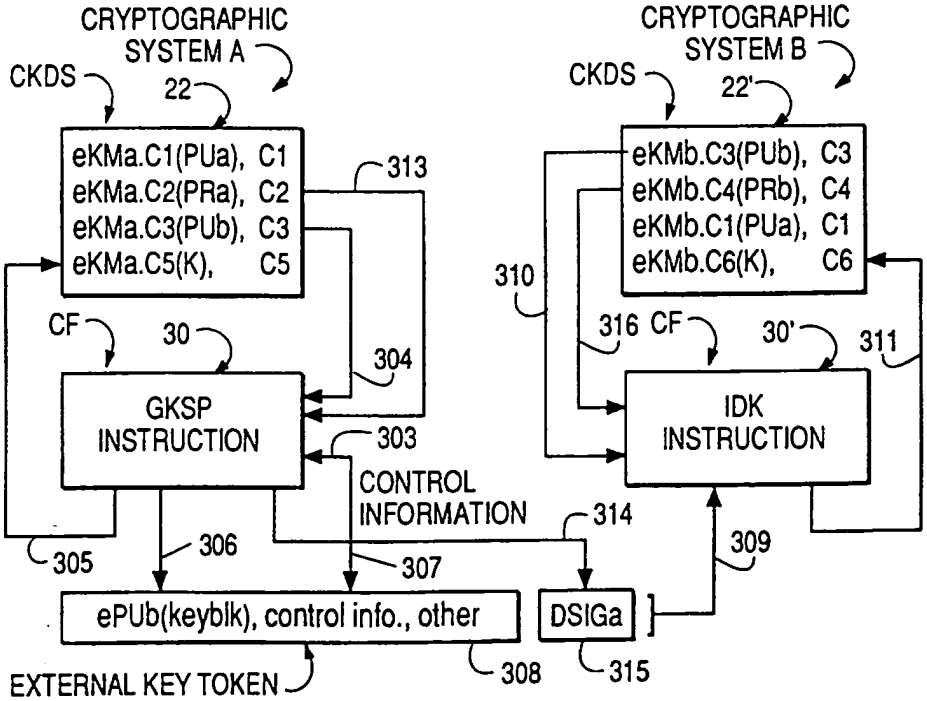


FIG. 9

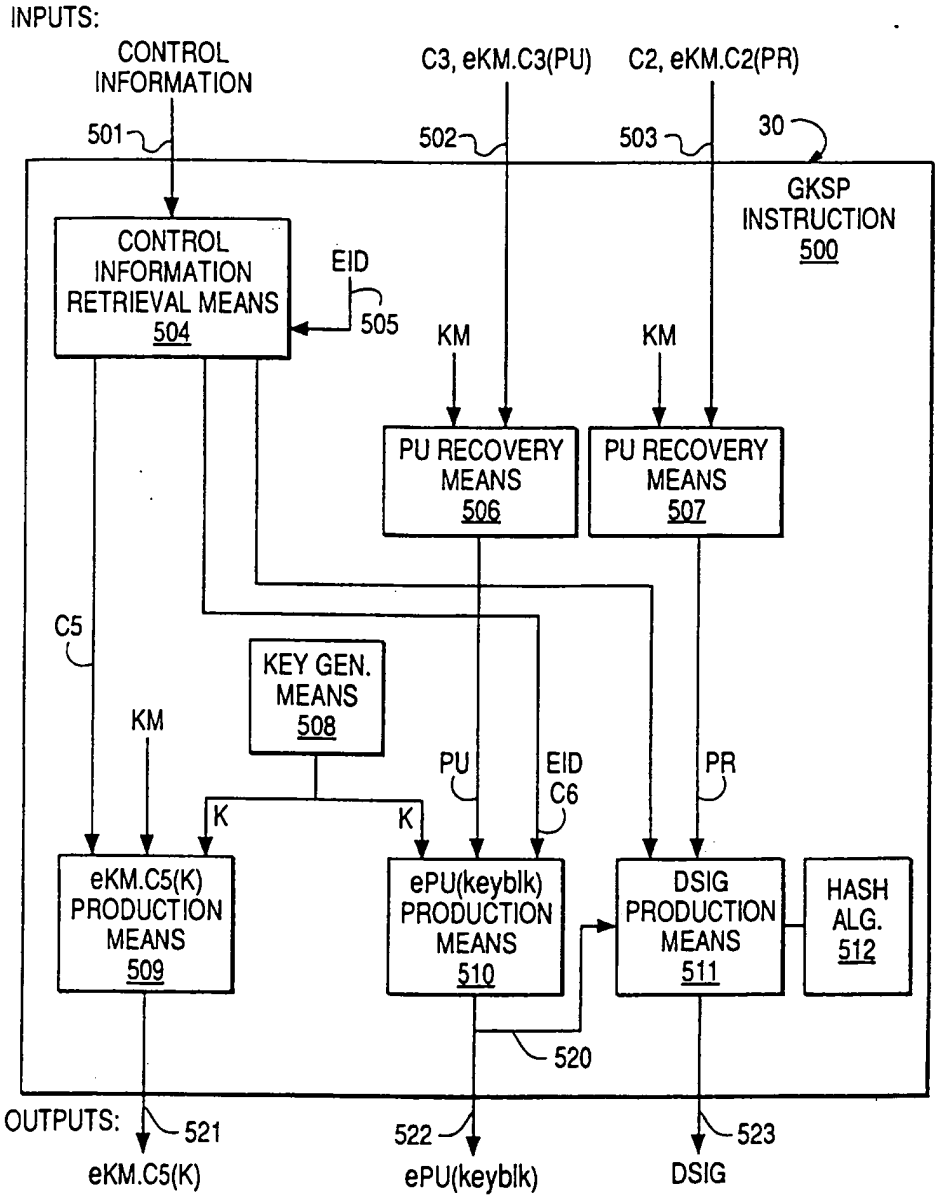


FIG. 10

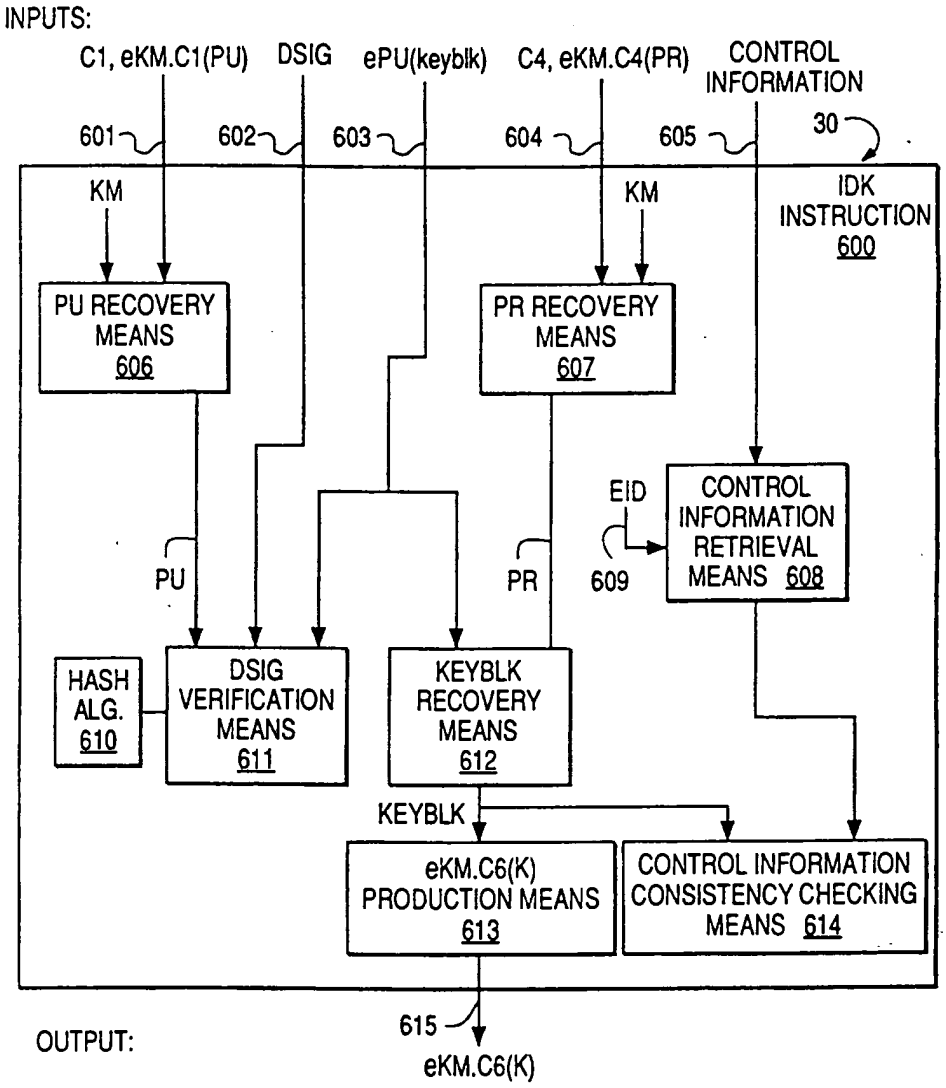


FIG. 11

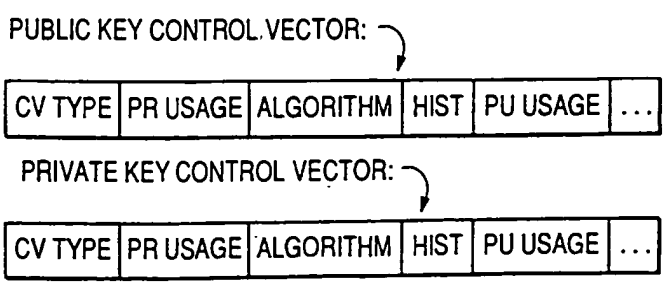


FIG. 12

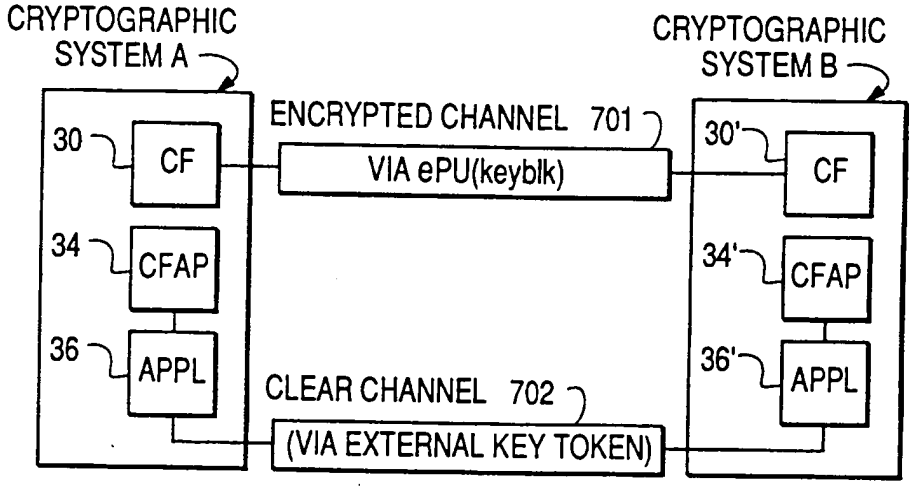


FIG. 13

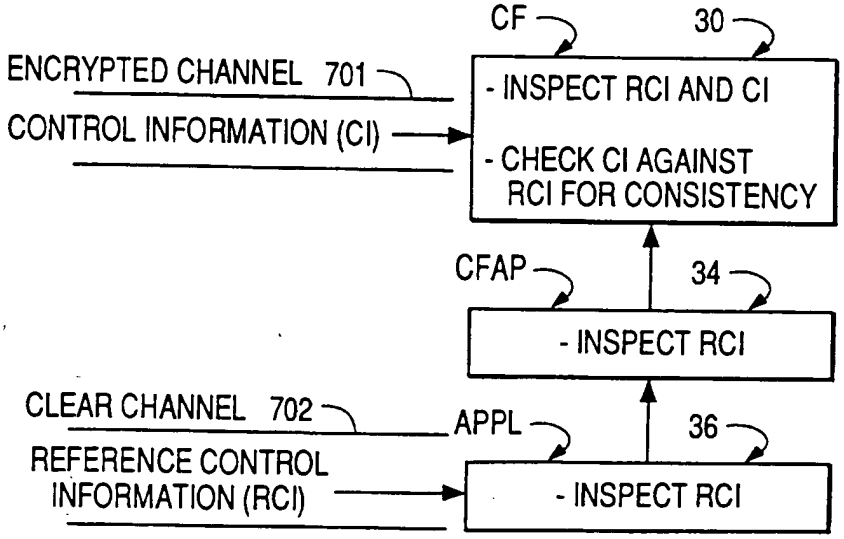


FIG. 14

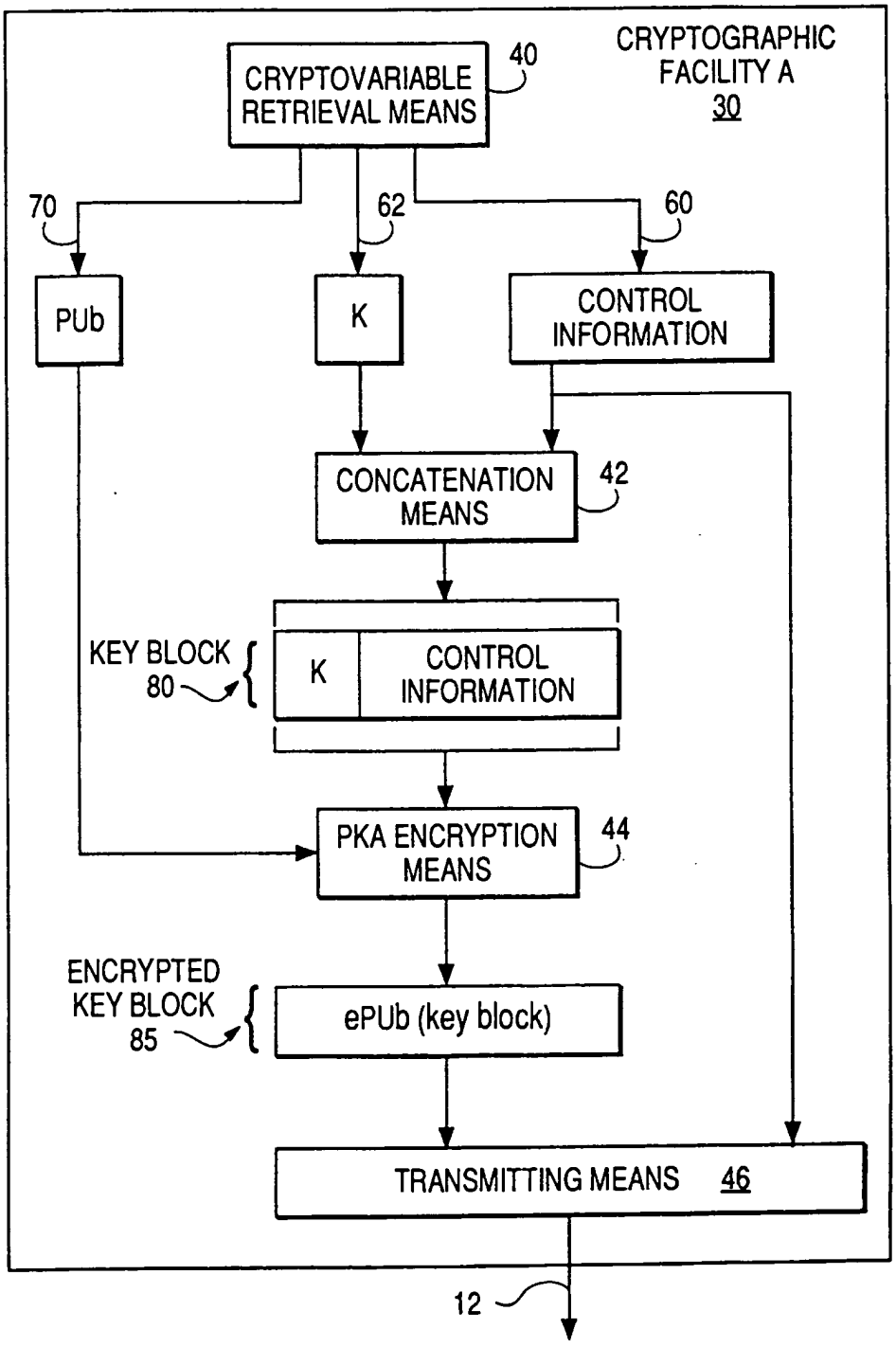


FIG. 15

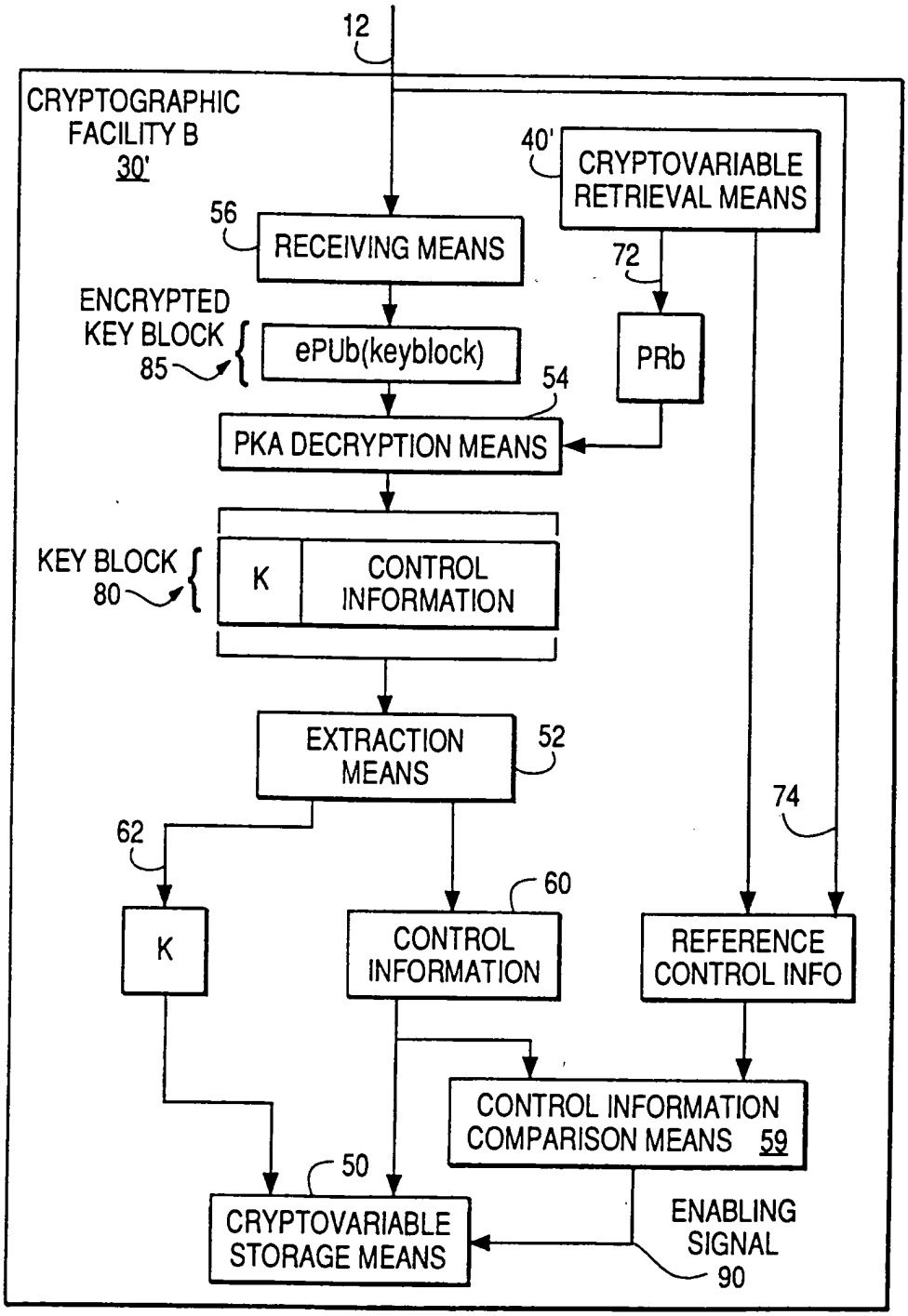
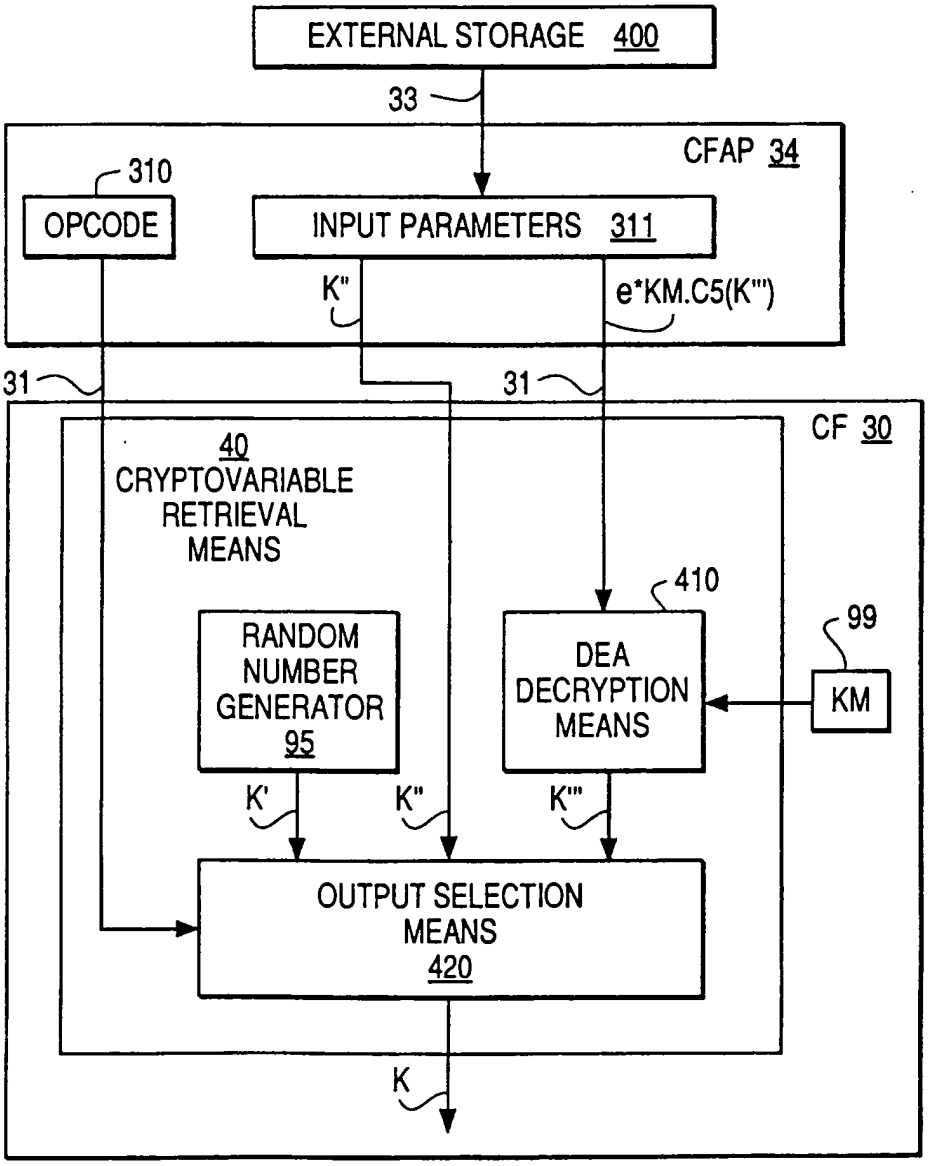


FIG. 16



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

(11) Publication number: **0 450 841 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

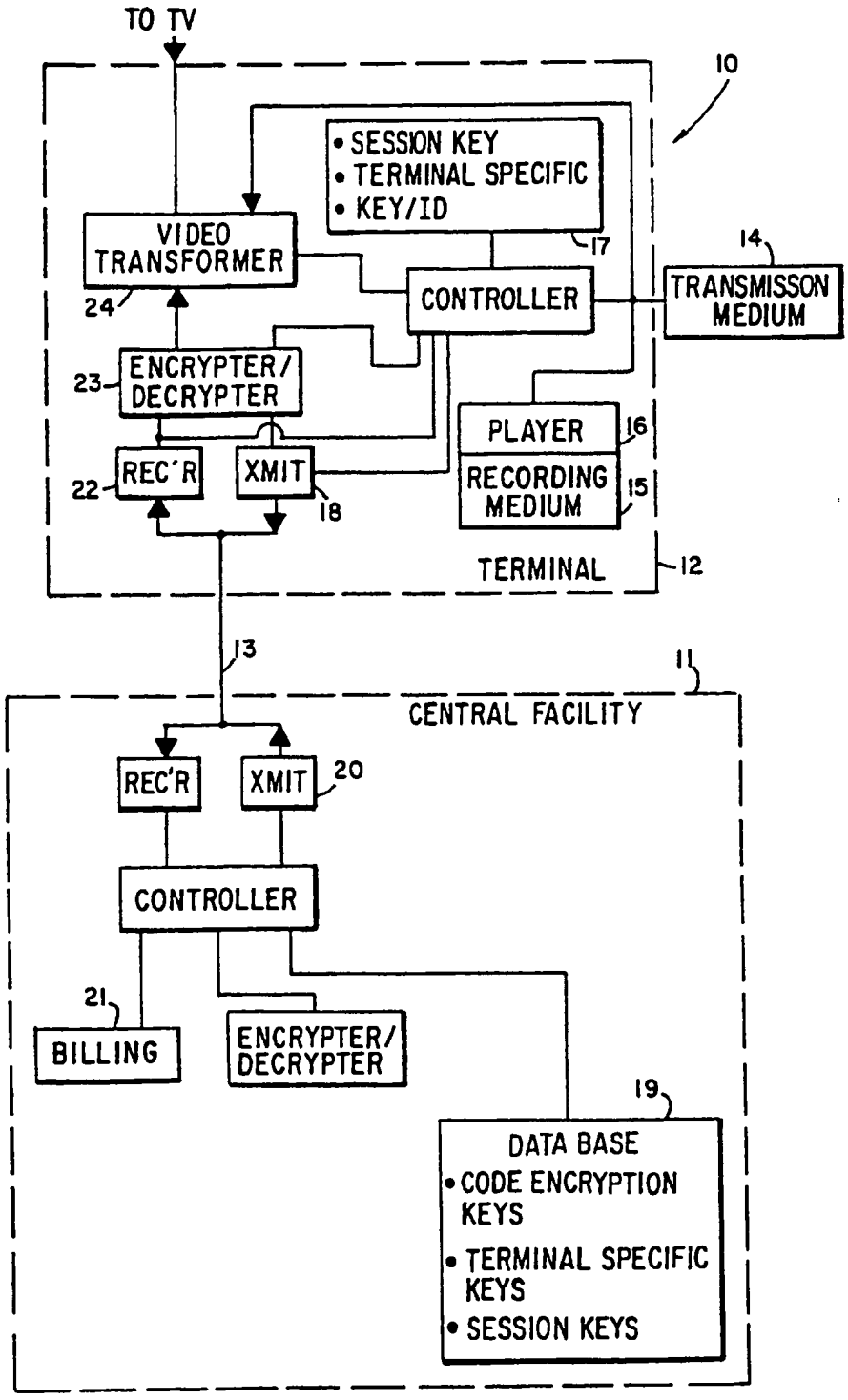

TO TV
10
• SESSION KEY
• TERMINAL SPECIFIC
• KEY/ID
17
VIDEO TRANSFORMER
24
CONTROLLER
14
TRANSMISSON MEDIUM
23
ENCRYPTER/ DECRYPTER
PLAYER
16
RECORDING MEDIUM
15
22
REC'R
XMIT
18
TERMINAL
12
13
11
CENTRAL FACILITY
REC'R
XMIT
20
CONTROLLER
21
BILLING
ENCRYPTER/ DECRYPTER
19
DATA BASE
• CODE ENCRYPTION 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 seid 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.

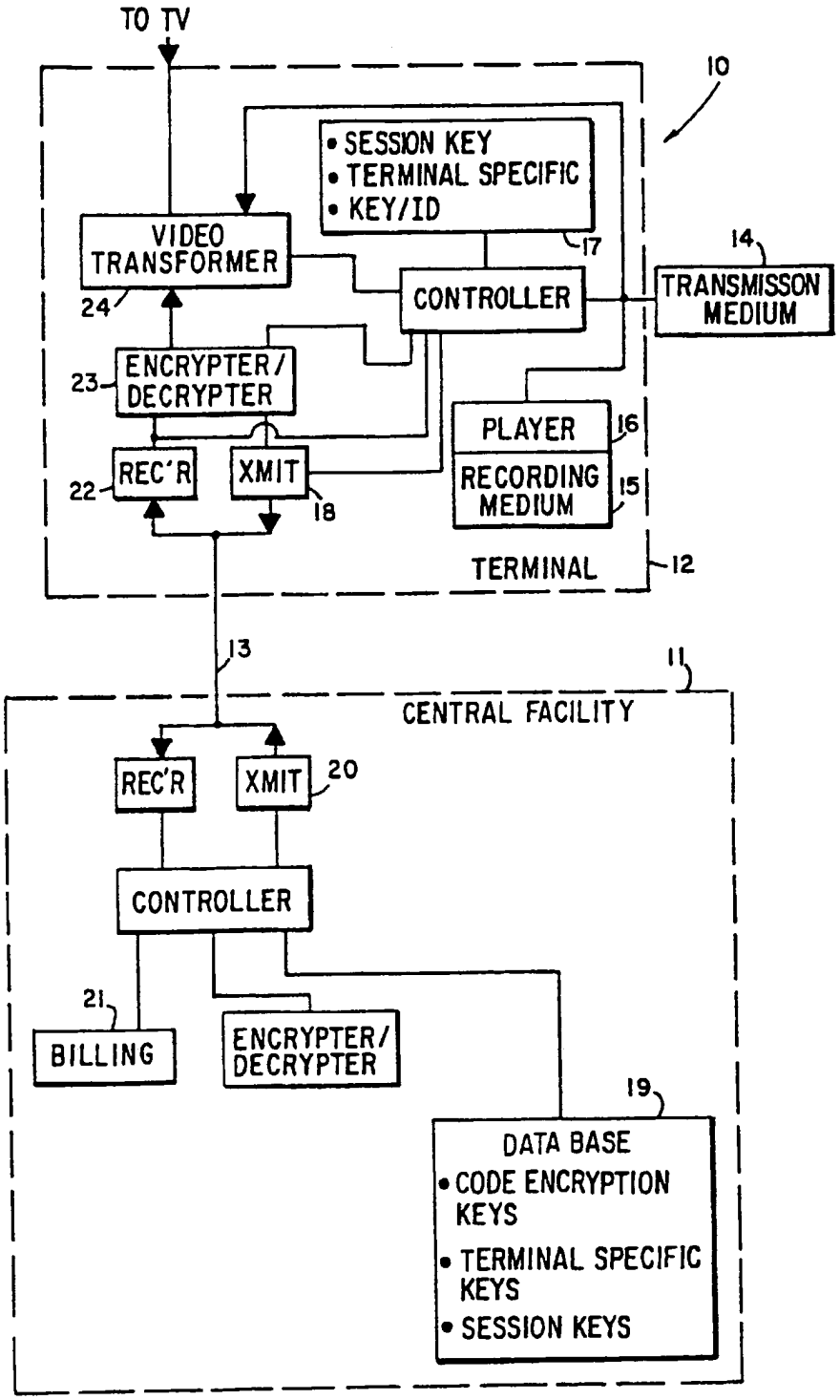


FIG.1

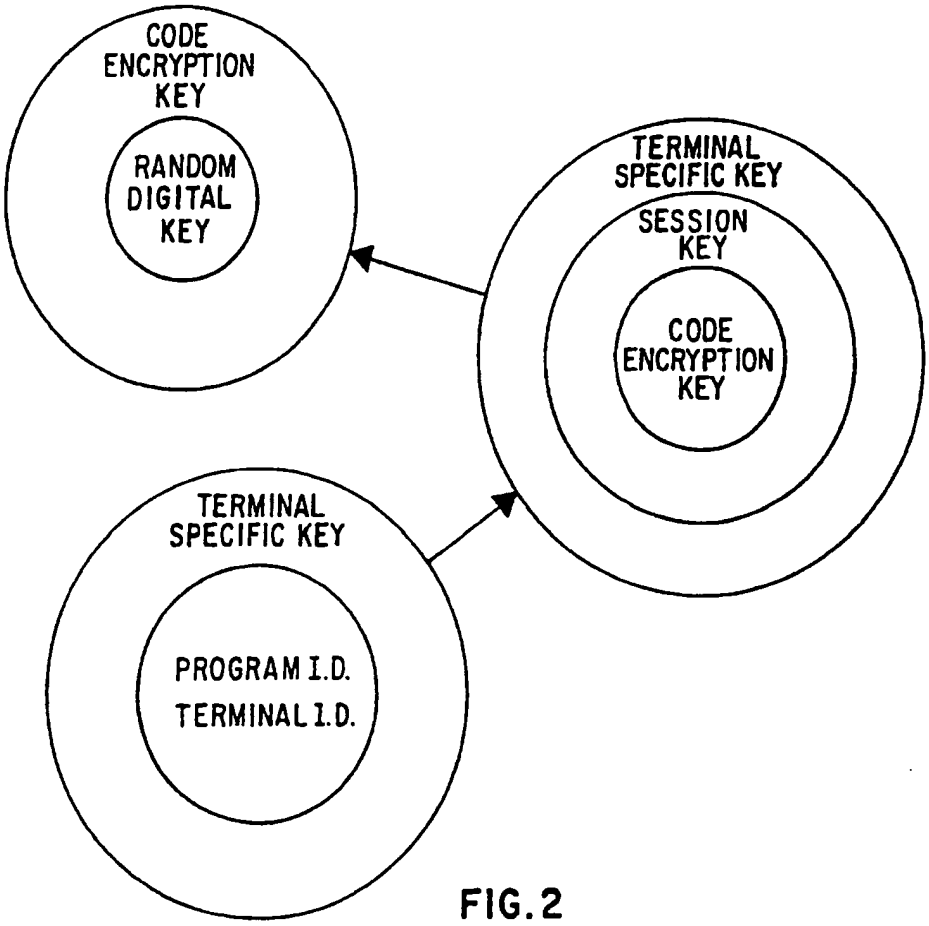


FIG. 2

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

(11) Publication number: **0 393 806 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.

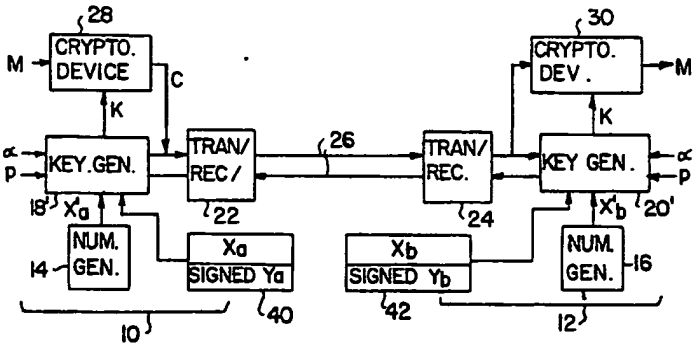


FIG. 3

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.

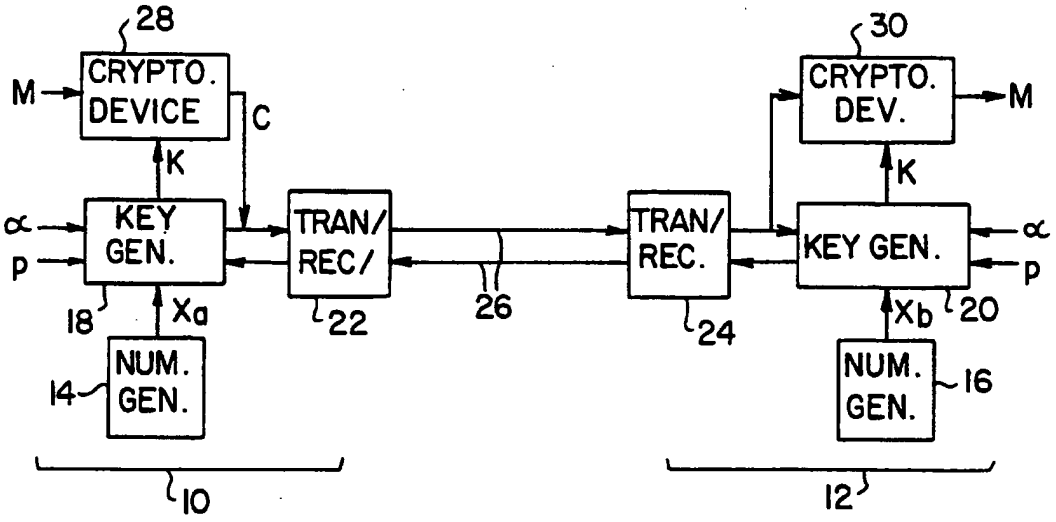


FIG. 1 (PRIOR ART)

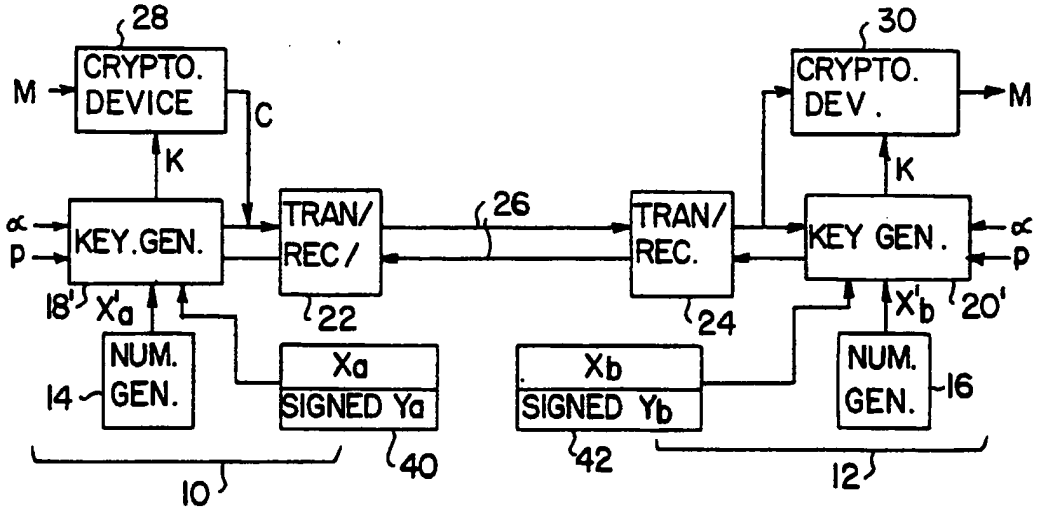


FIG. 3

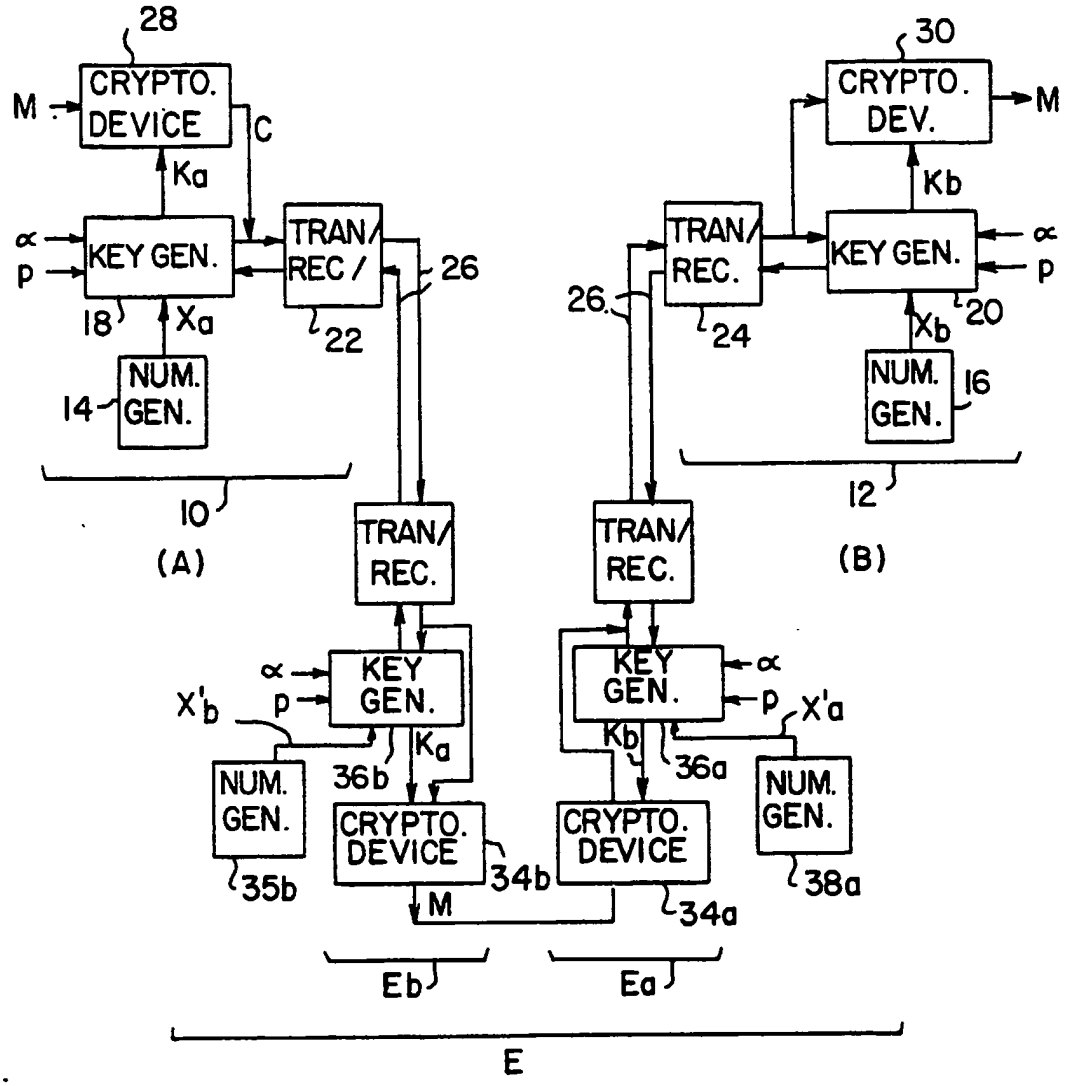
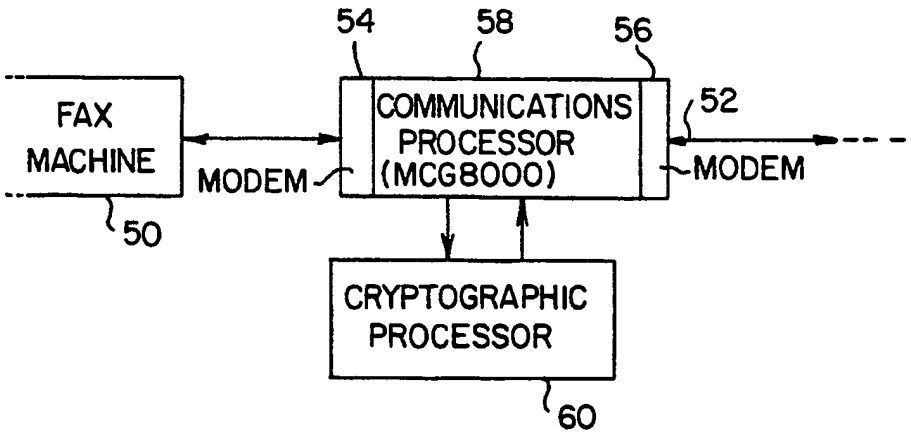


FIG. 2 (PRIOR ART)

FIG. 4


54
58
56
52
FAX MACHINE
MODEM
COMMUNICATIONS PROCESSOR (MCG8000)
MODEM
50
CRYPTOGRAPHIC PROCESSOR
60

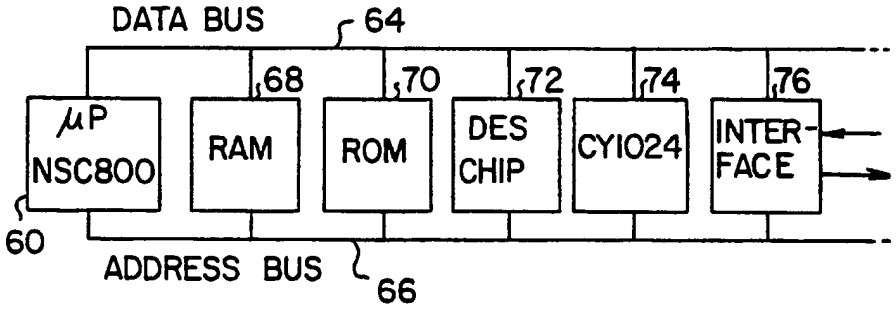

DATA BUS
64
68
70
72
74
76
μP NSC800
RAM
ROM
DES CHIP
CYI024
INTER-FACE
60
ADDRESS BUS
66

FIG. 5

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

(11) Publication number: **0 332 304 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.

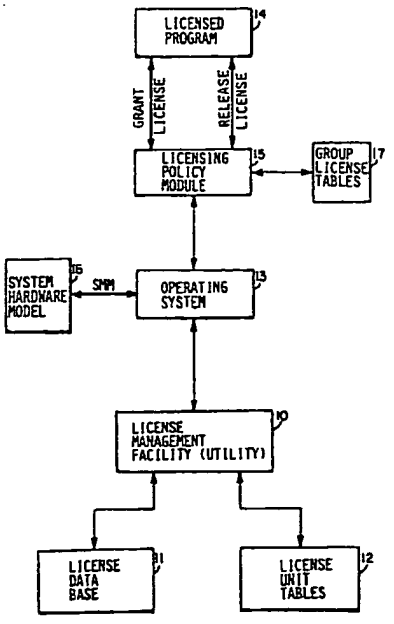


FIG. 1

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.

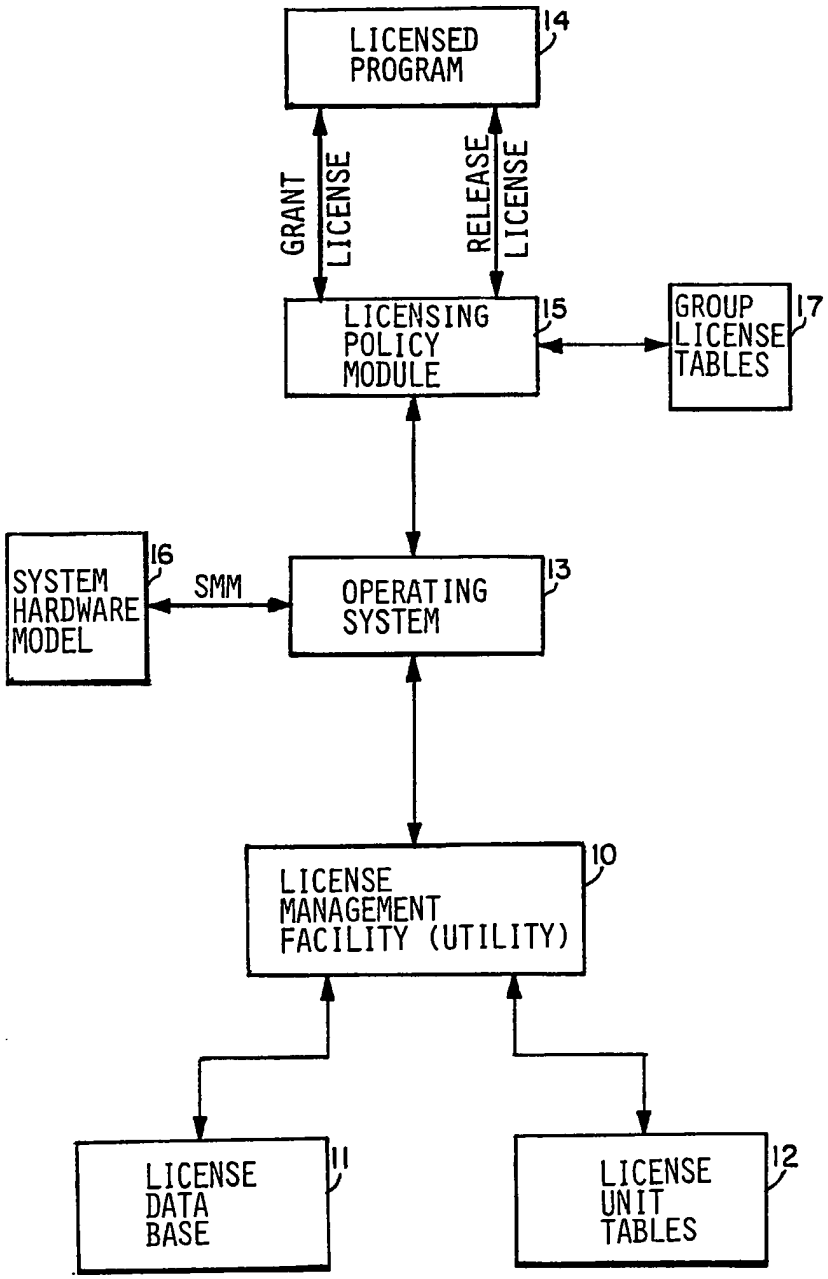


FIG. 1

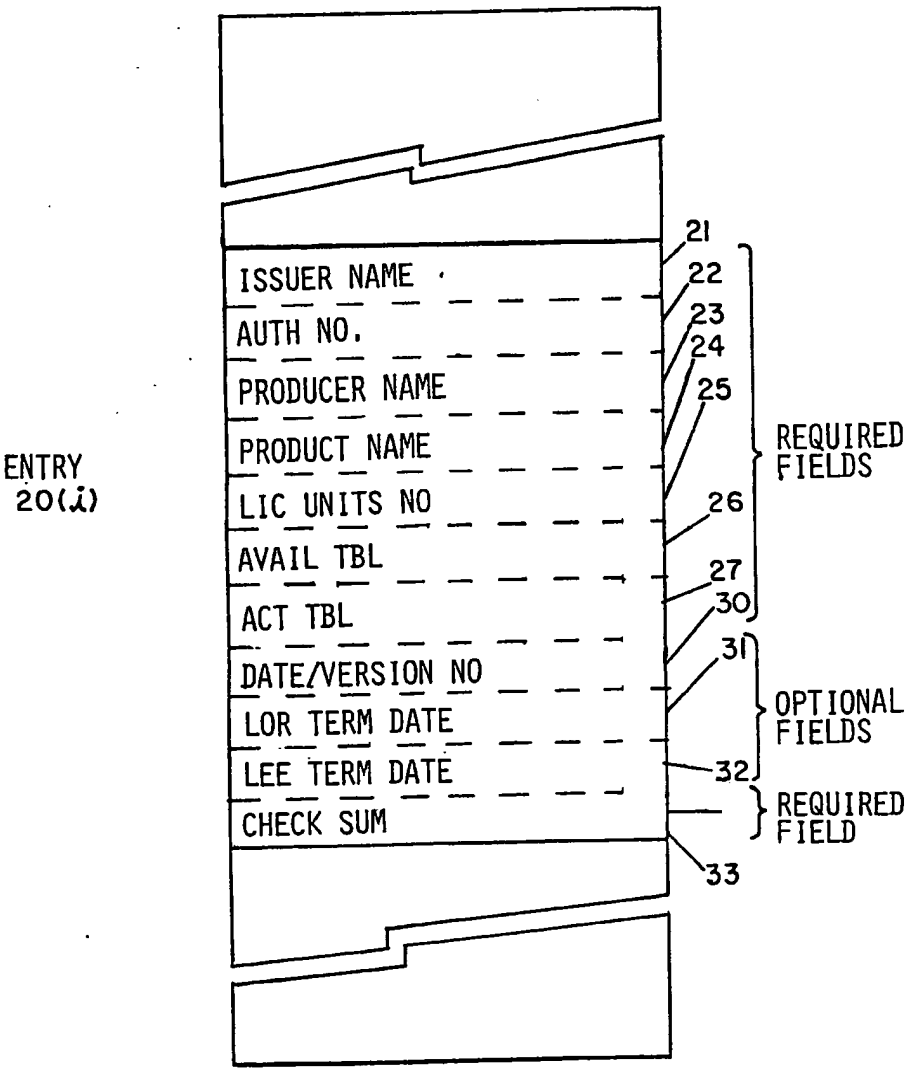
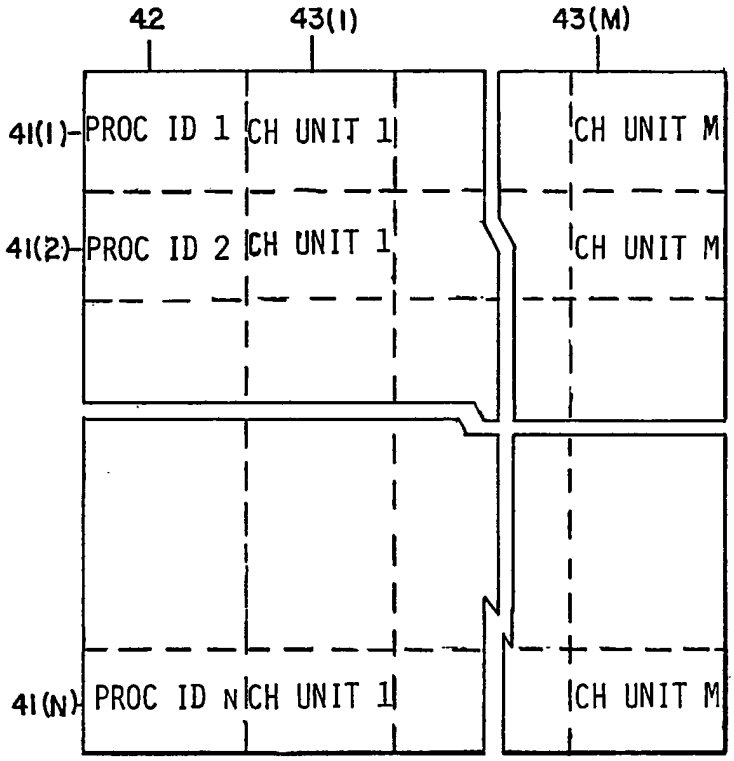


FIG. 2



LICENSE UNIT TABLE 40

FIG. 3

FIG. 4A-1

GRANT LICENSE

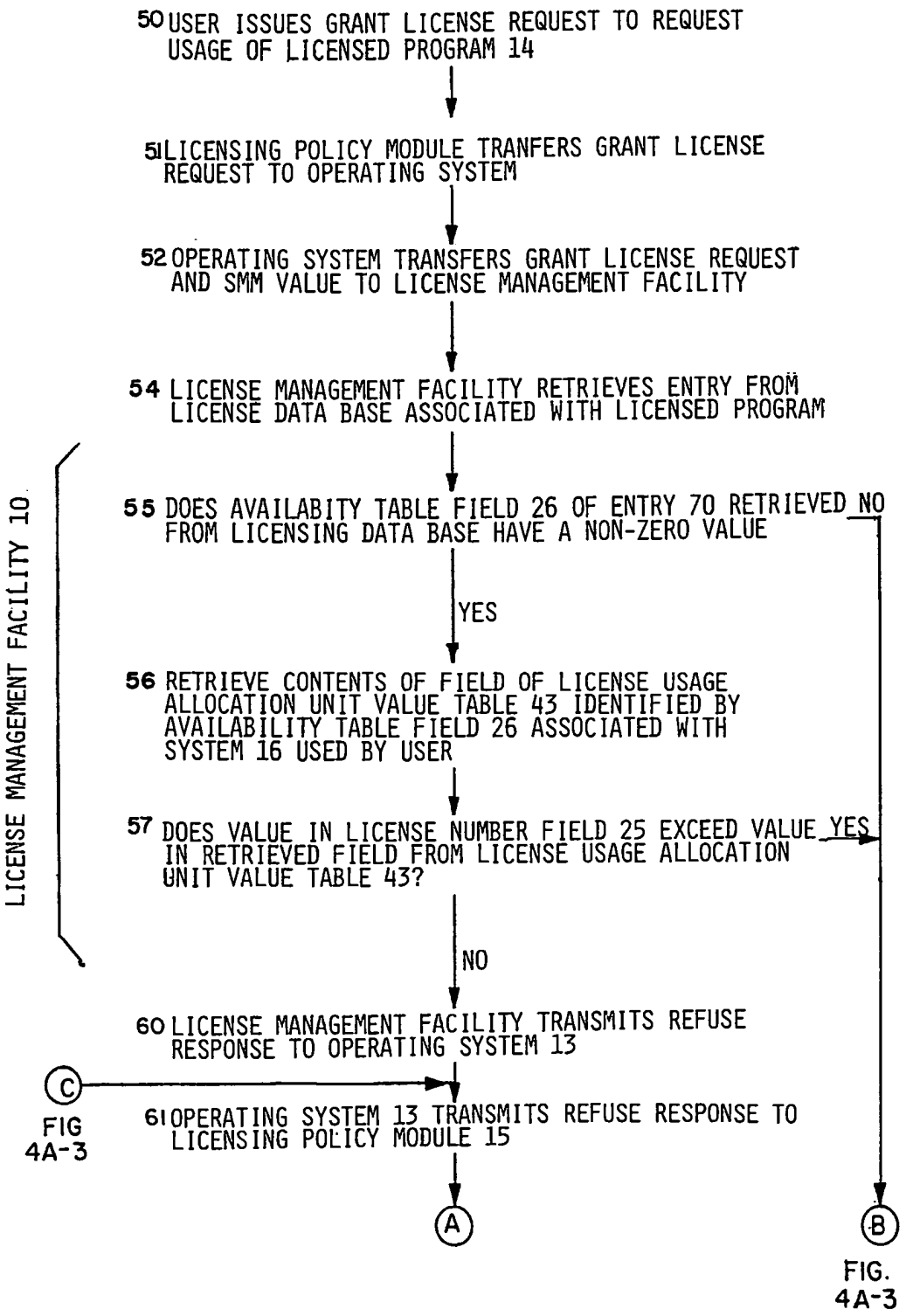


FIG. 4A-3

FIG. 4A-2

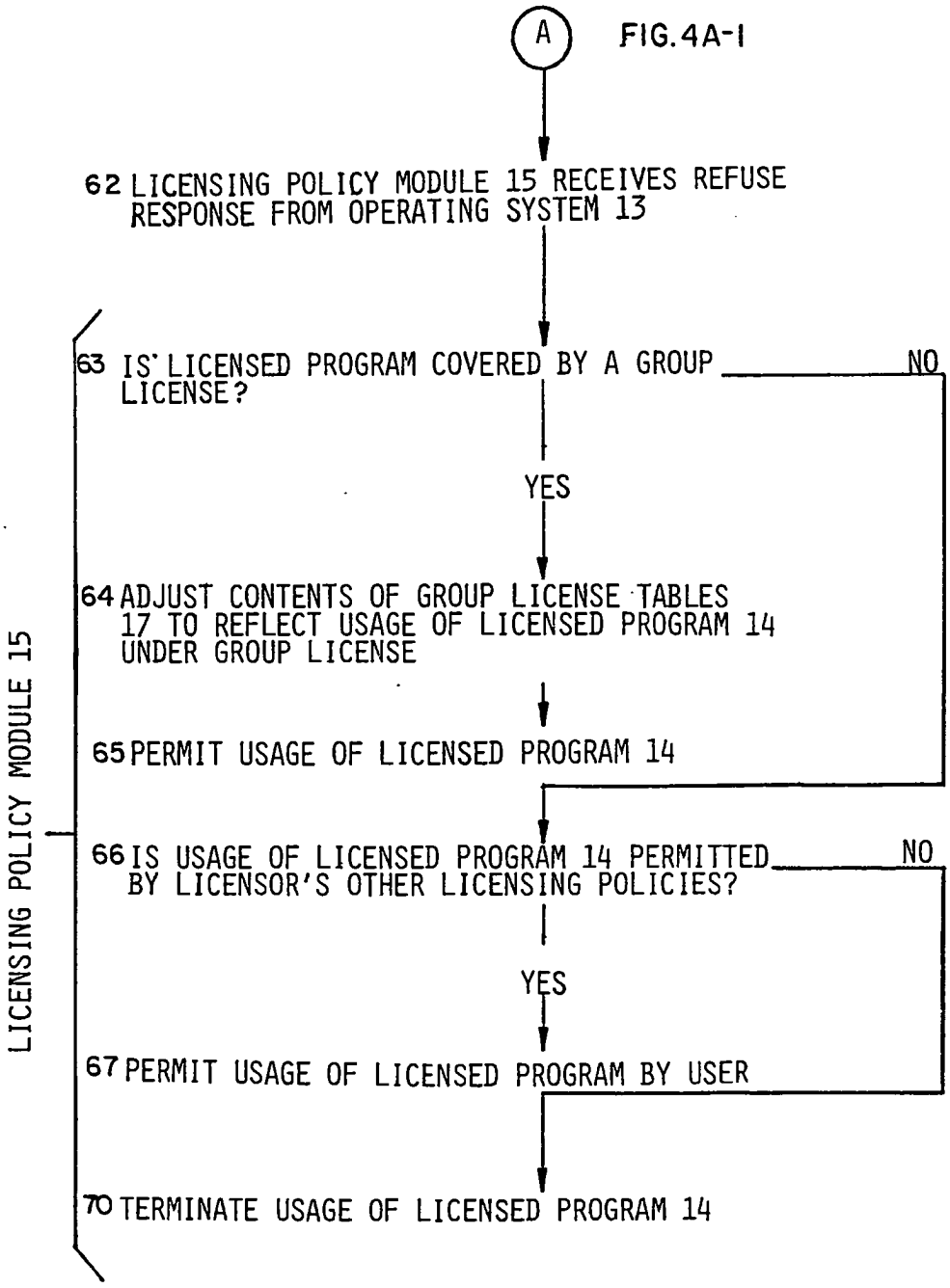


FIG. 4A-3

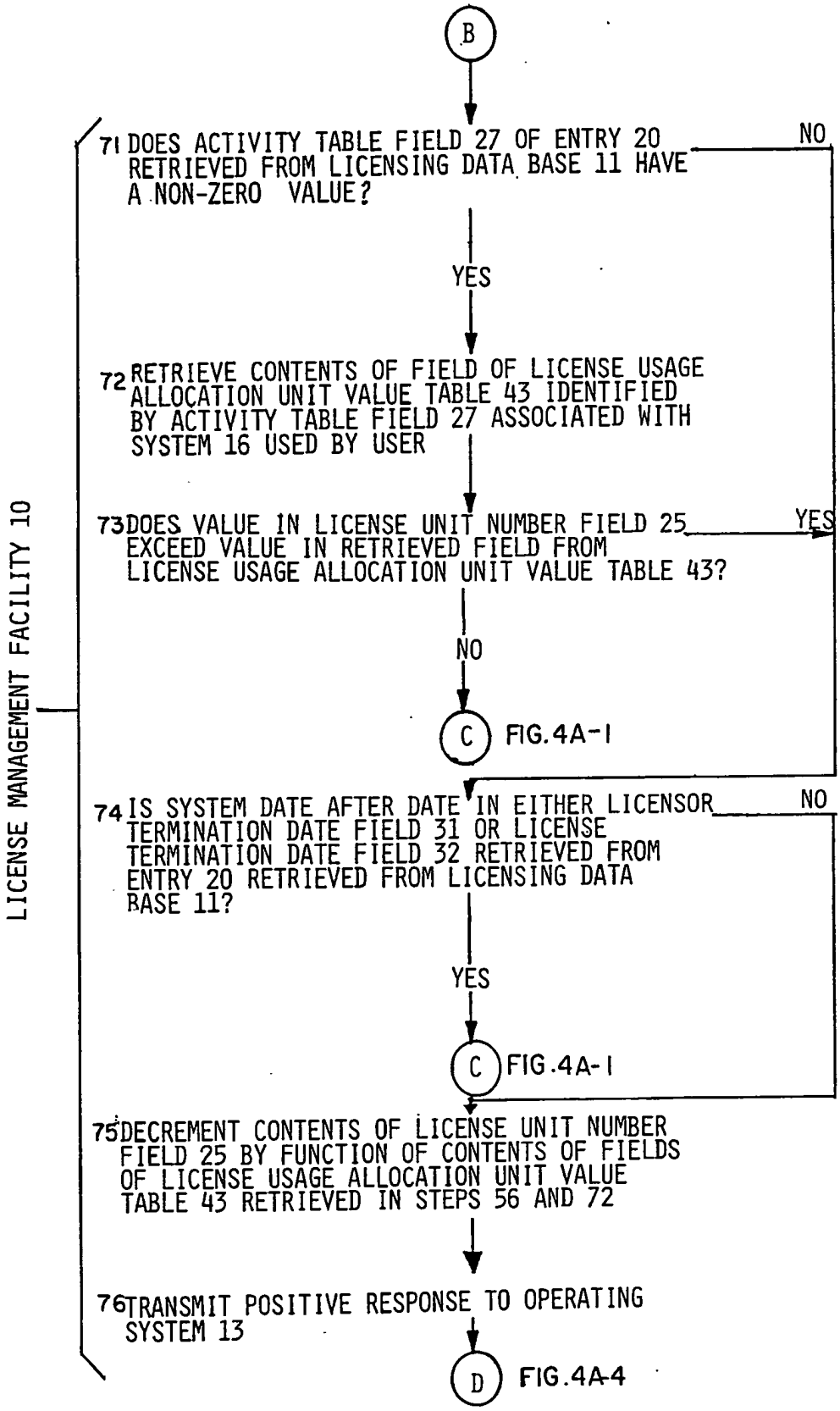

B
LICENSE MANAGEMENT FACILITY 10
71 DOES ACTIVITY TABLE FIELD 27 OF ENTRY 20 RETRIEVED FROM LICENSING DATA BASE 11 HAVE A NON-ZERO VALUE?
NO
YES
72 RETRIEVE CONTENTS OF FIELD OF LICENSE USAGE ALLOCATION UNIT VALUE TABLE 43 IDENTIFIED BY ACTIVITY TABLE FIELD 27 ASSOCIATED WITH SYSTEM 16 USED BY USER
73 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

FIG. 4A-4

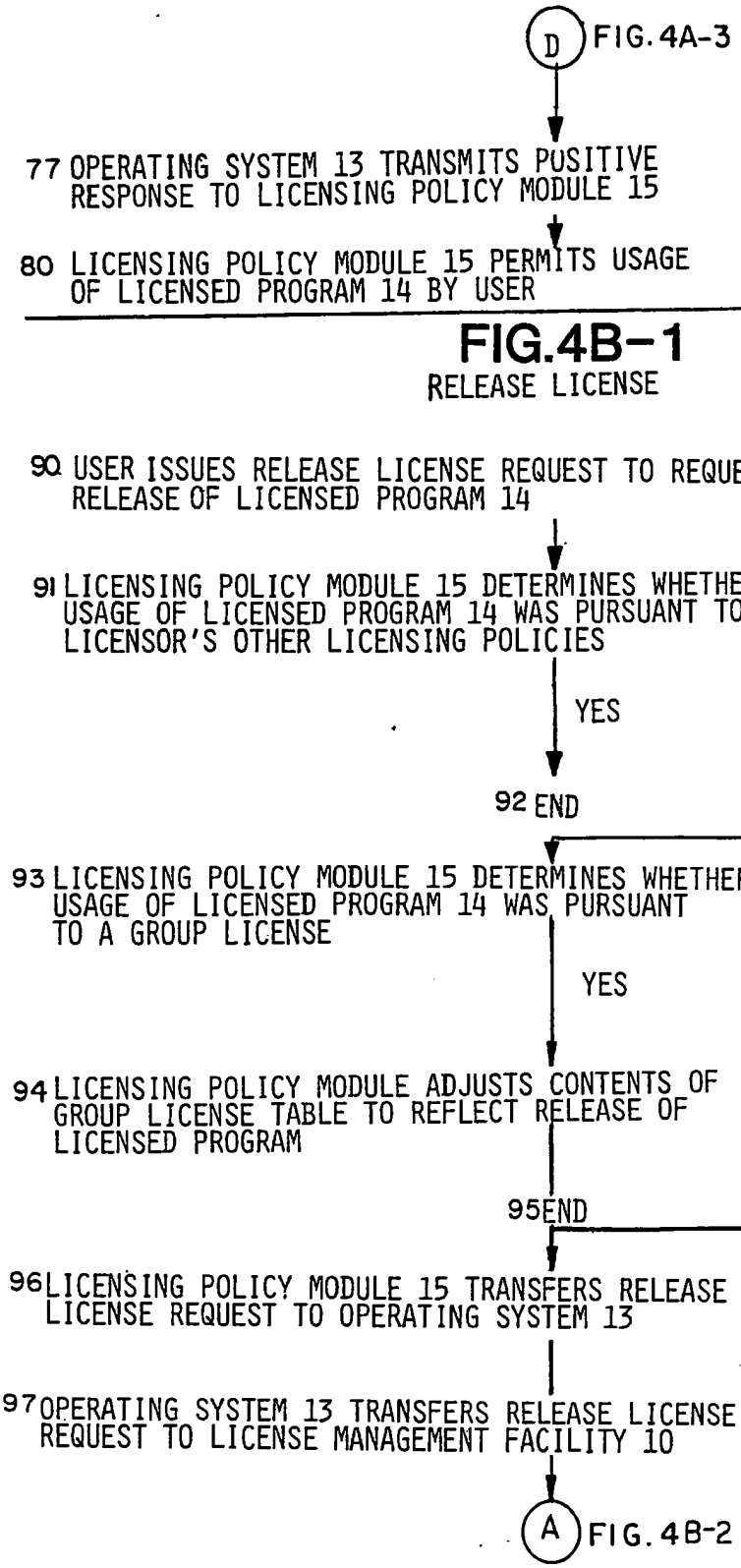
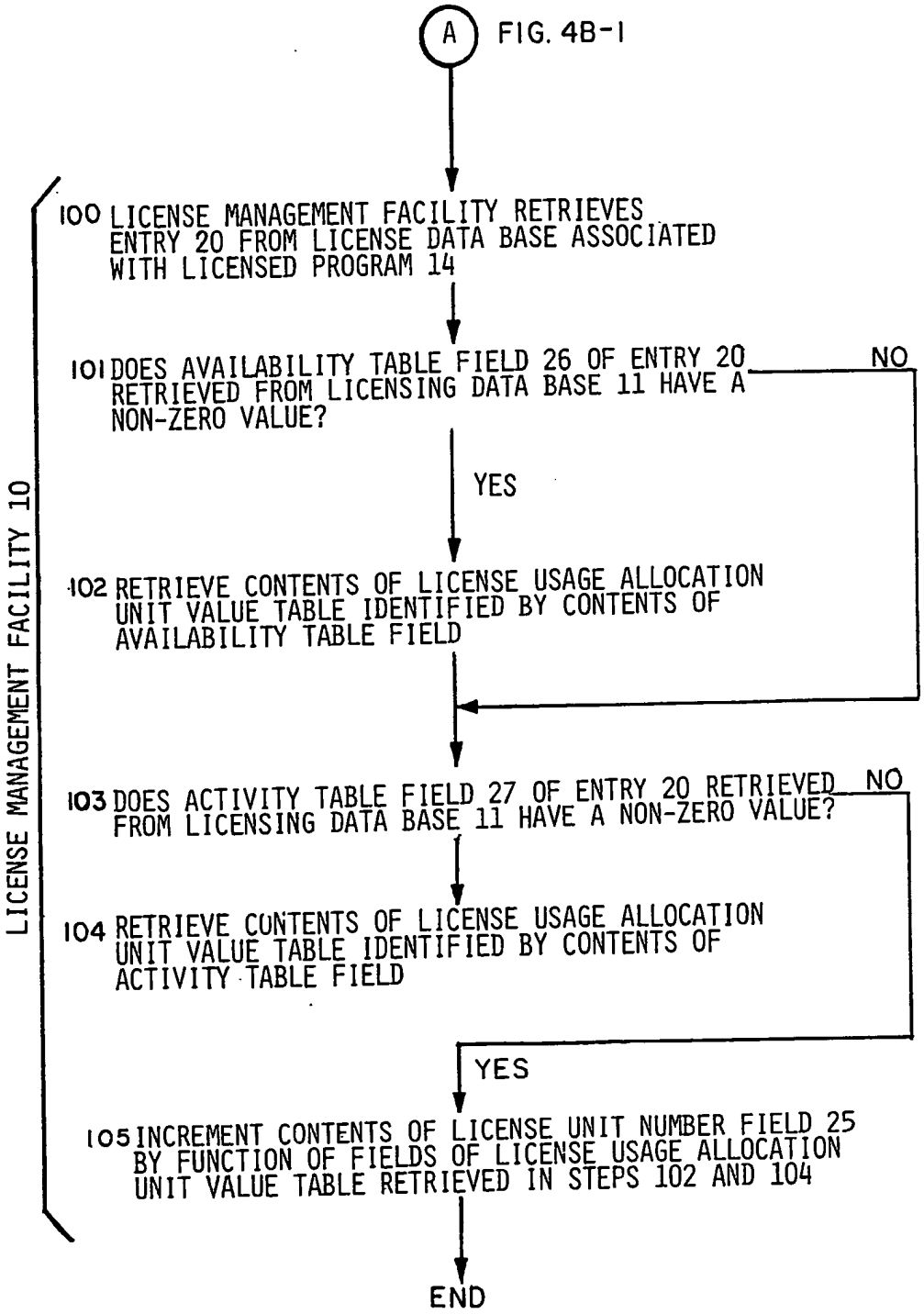


FIG.4B-1

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: **0 257 585**
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.

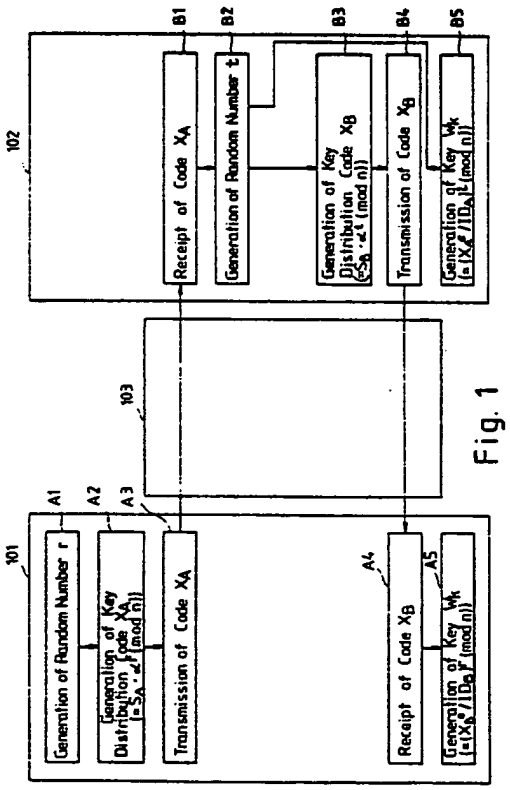


Fig. 1

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 $Y_B{}^{X_A}$ (mod 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 p is a large prime number of about 256 bits in binary representation, which is publicly known. a (mod b) means a remainder of division of the number a by the number b. The converser B also prepares the key wk in accordance to $Y_A{}^{X_B}$ (mod p) in a similar manner. Y_A and Y_B are selected so as to be equal to α^{X_A} (mod p) and α^{X_B} (mod p), respectively. As a result, $Y_B{}^{X_A}$ (mod p) becomes equal to $Y_A{}^{X_B}$ (mod p). It is known that even if Y_A, a and p are known, it is infeasible for anybody except the converser A to obtain X_A which satisfies $Y_A = \alpha^{X}A$ (mod 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 $e.d \pmod{(p-1)\cdot(q-1)} = 1$. 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 $S_A \cdot \alpha^{c \cdot r} \pmod{n}$ (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 $S_B \cdot \alpha^{c \cdot t} \pmod{n}$ (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 $Y_A^e / X_A^c \pmod{n}$ ($= S_A^e \cdot \alpha^{erc} / \alpha^{erc} \pmod{n}$) and $Y_B^e / X_B^c \pmod{n}$ ($= S_B^e \cdot \alpha^{etc} / \alpha^{etc} \pmod{n}$), 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).

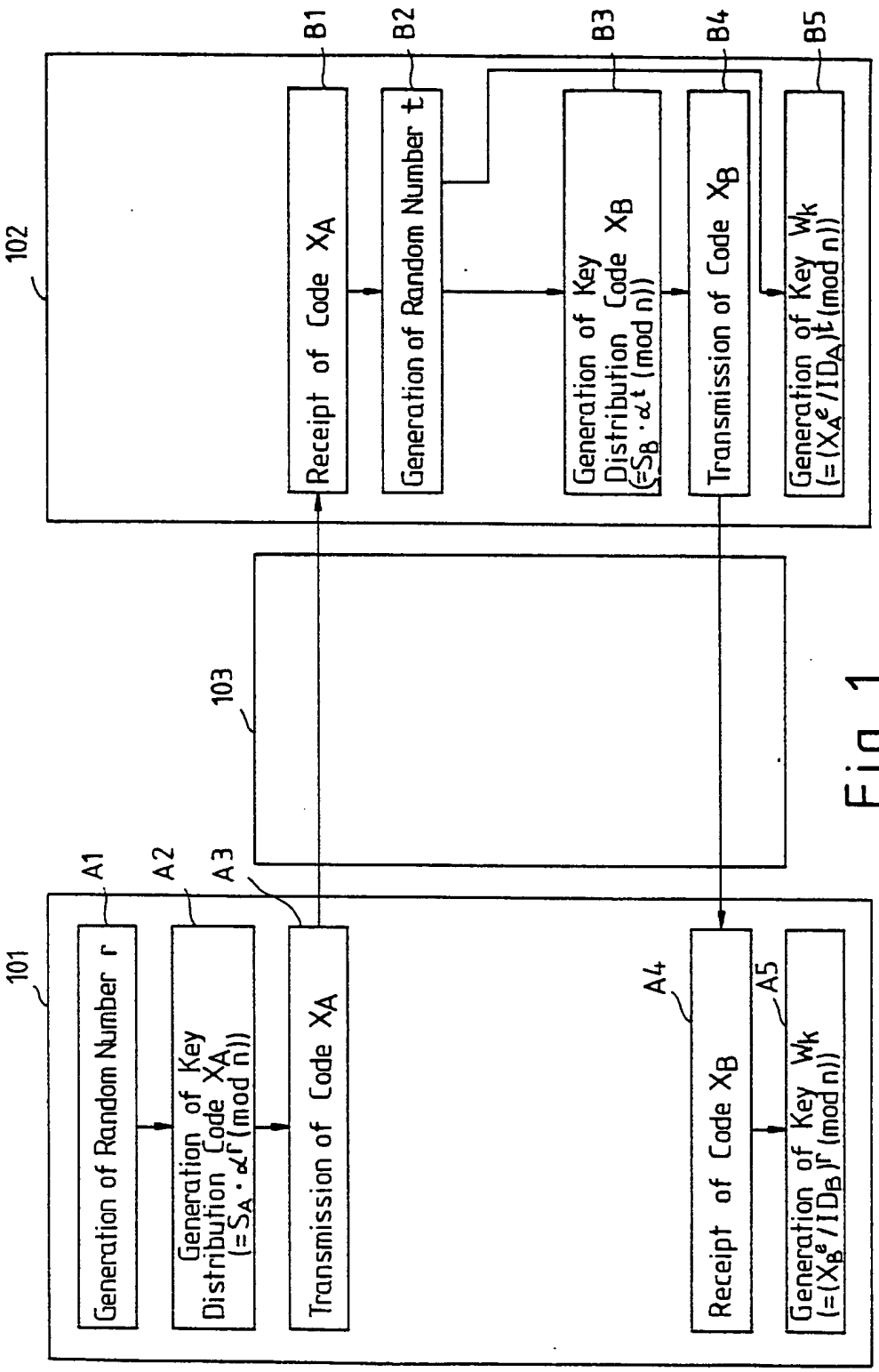

101
102
103
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
Generation of Random Number r
Generation of Key Distribution Code X_A (=$S_A \cdot \alpha^r \pmod n$))
Transmission of Code X_A
Receipt of Code X_B
Generation of Key Wk (=$(X_B^e / ID_B)^r \pmod n$))
Receipt of Code X_A
Generation of Random Number t
Generation of Key Distribution Code X_B (=$S_B \cdot \alpha^t \pmod n$))
Transmission of Code X_B
Generation of Key Wk (=$(X_A^e / ID_A)^t \pmod n$))

Fig. 1

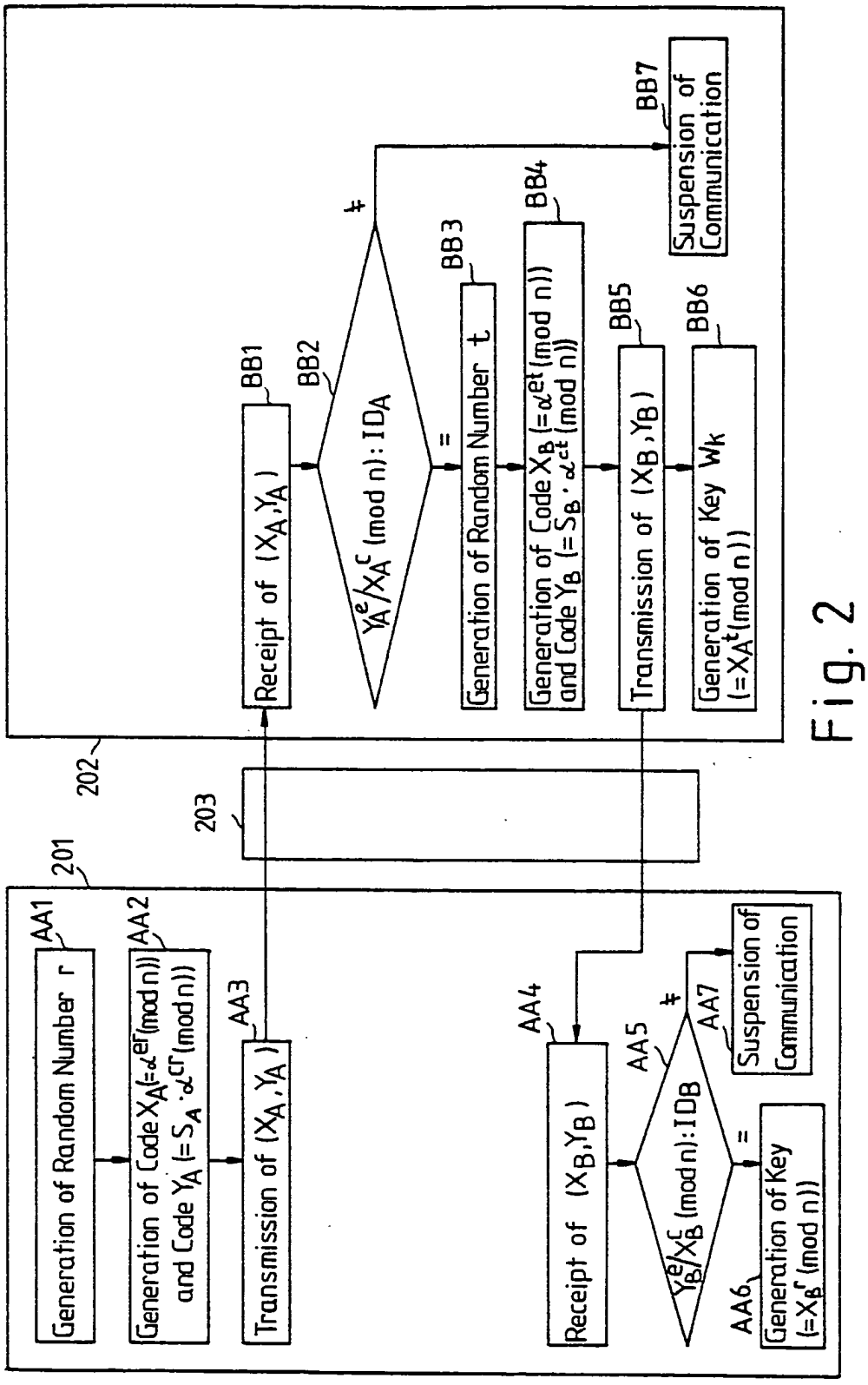

Generation of Random Number r
AA1
Generation of Code $X_A(=\alpha^{er} \pmod n)$ and Code $Y_A(=S_A \cdot \alpha^{cr} \pmod n)$
AA2
Transmission of (X_A, Y_A)
AA3
Receipt of (X_B, Y_B)
AA4
$Y_B^e/X_B^c \pmod n : ID_B$
AA5
Suspension of Communication
AA7
Generation of Key $(=X_B^r \pmod n)$
AA6
201
202
203
Receipt of (X_A, Y_A)
BB1
$Y_A^e/X_A^c \pmod n : ID_A$
BB2
Generation of Random Number t
BB3
Generation of Code $X_B(=\alpha^{et} \pmod n)$ and Code $Y_B(=S_B \cdot \alpha^{ct} \pmod n)$
BB4
Transmission of (X_B, Y_B)
BB5
Generation of Key W_k $(=X_A^t \pmod n)$
BB6
Suspension of Communication
BB7

Fig. 2

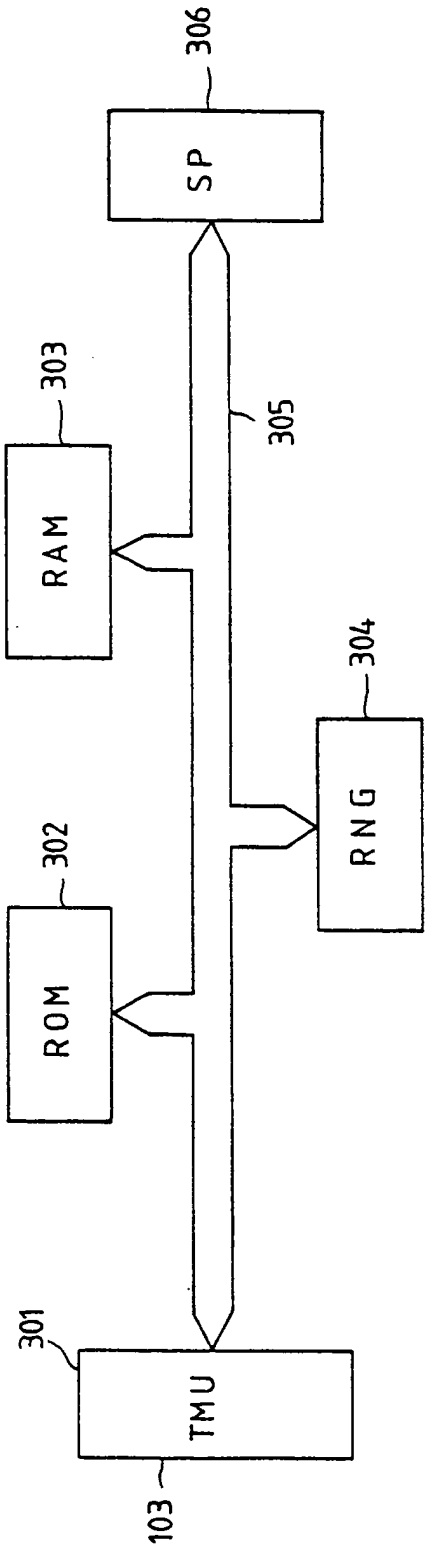


Fig. 3

(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)

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 mentissa 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_t) 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 10114 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), 0 (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, LB, 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 27SO3As.

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 performed 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(26:28)) 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 simllarly 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 significant 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 mcroinstruction 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 attemting 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 SN74S04s.

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 MCW0, 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 CQAS 25512 or of CQAE 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 microinstruction 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 wi11 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 ere 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 0 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 performs 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/0 operation illustrates how the components work together. Each user I/0 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.

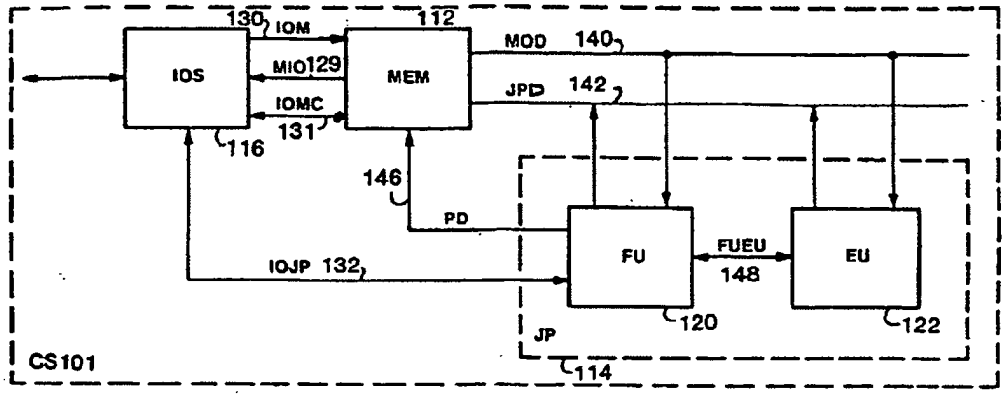


FIG 1

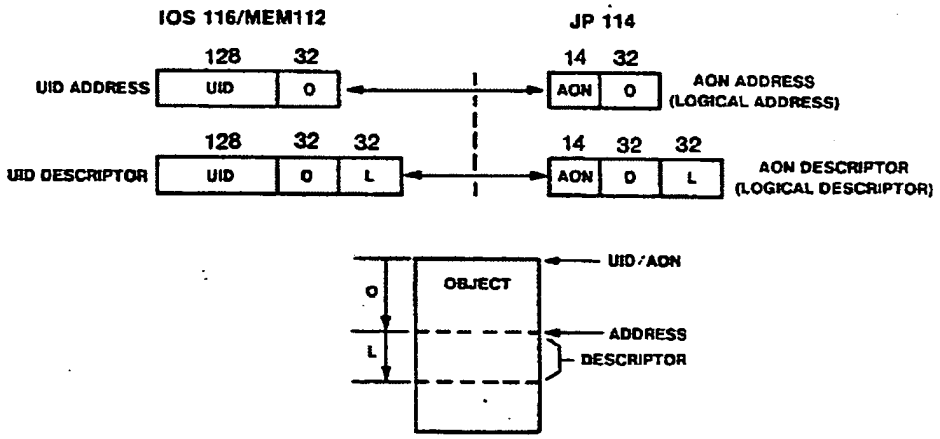


FIG 2

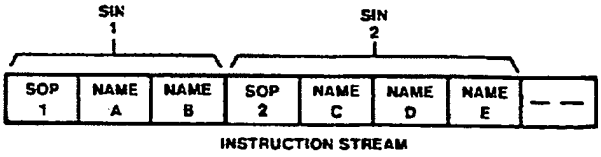


FIG 3

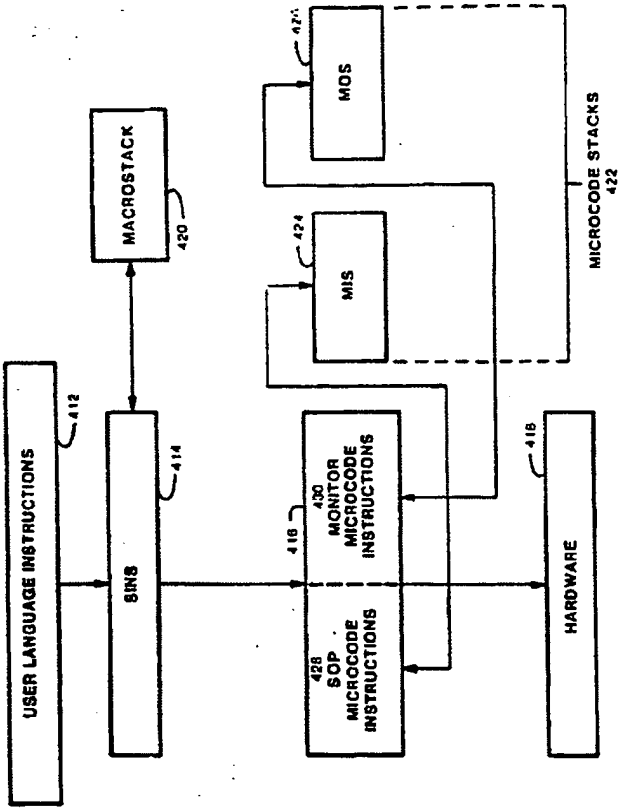


FIG 4A

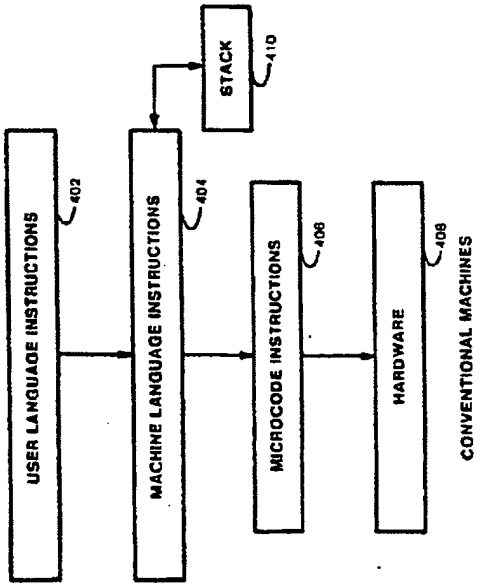


FIG 4

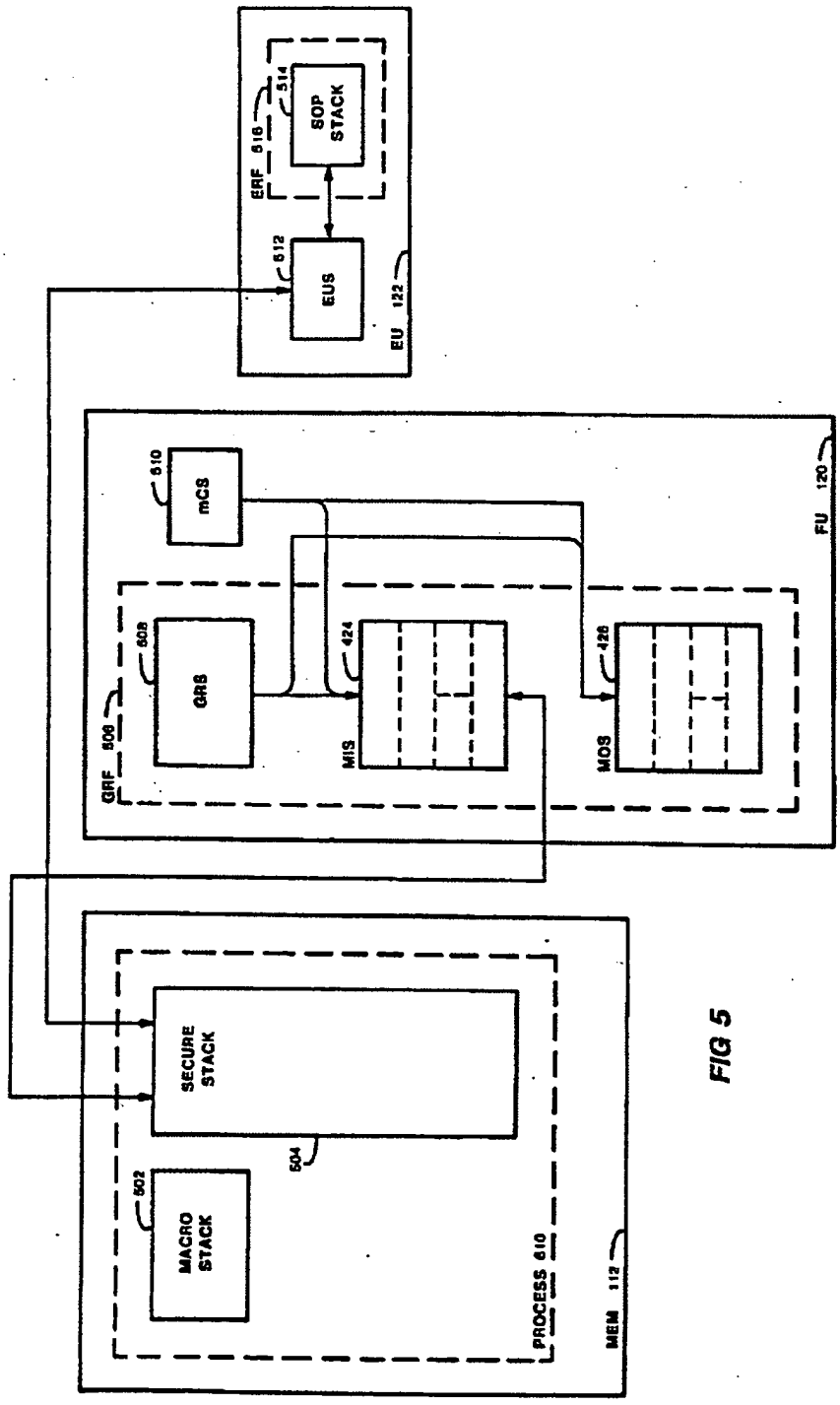


FIG 5

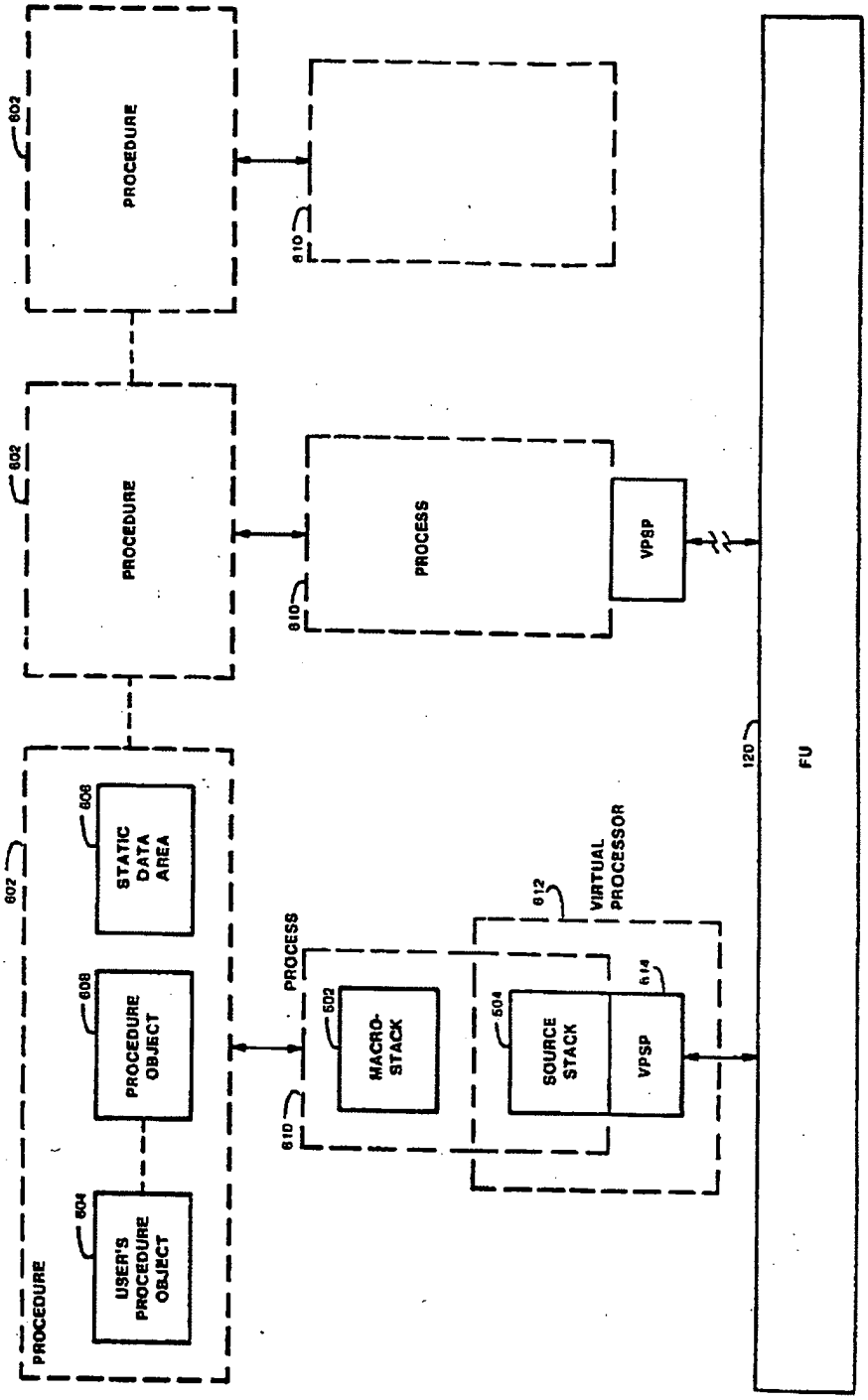


FIG 6

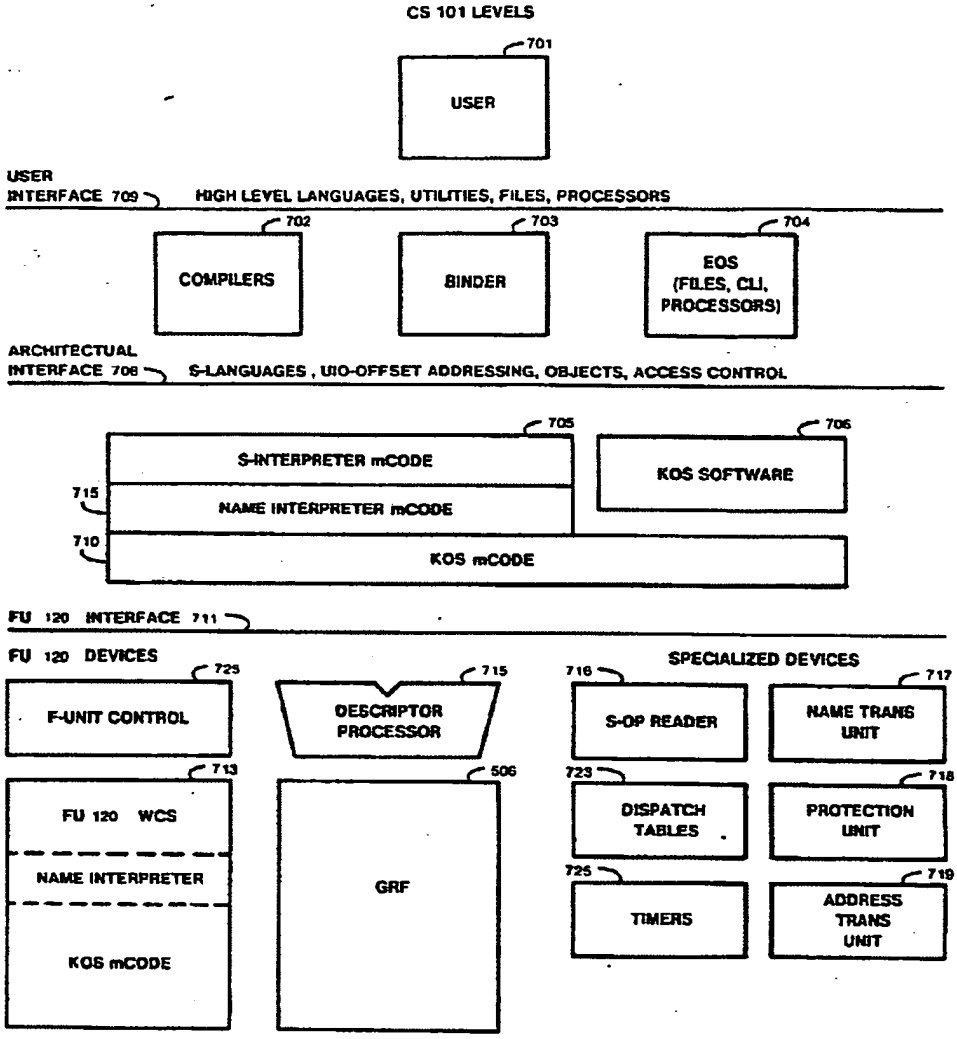


FIG 7

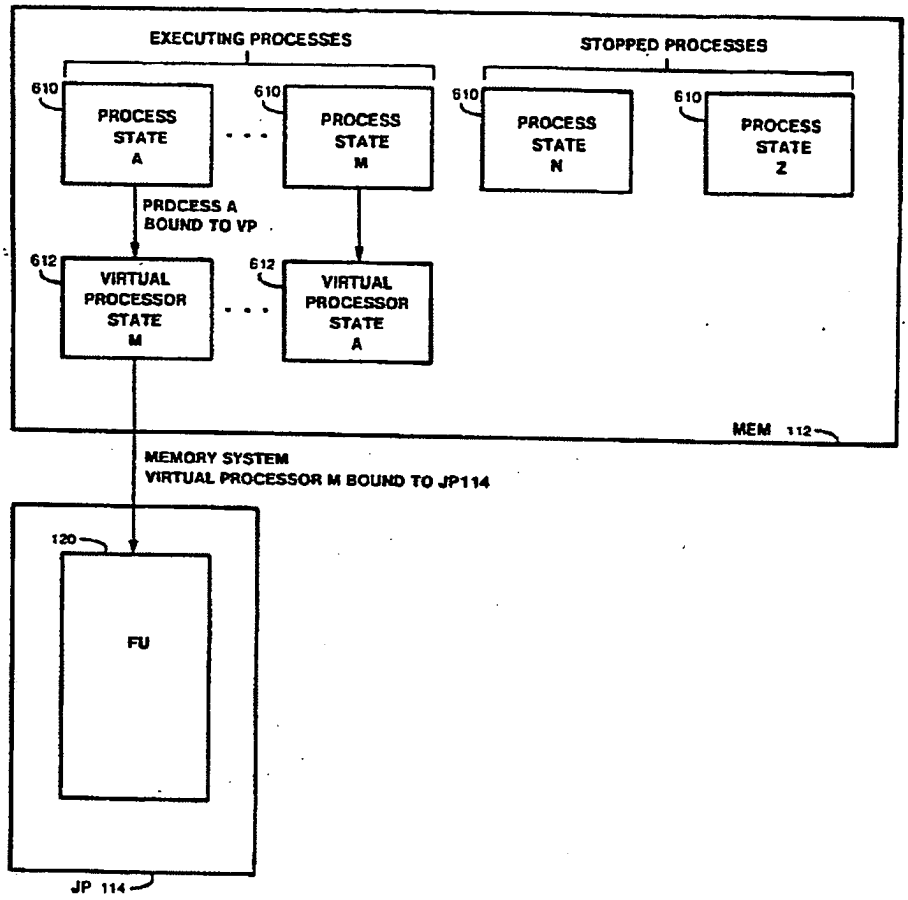


FIG 8

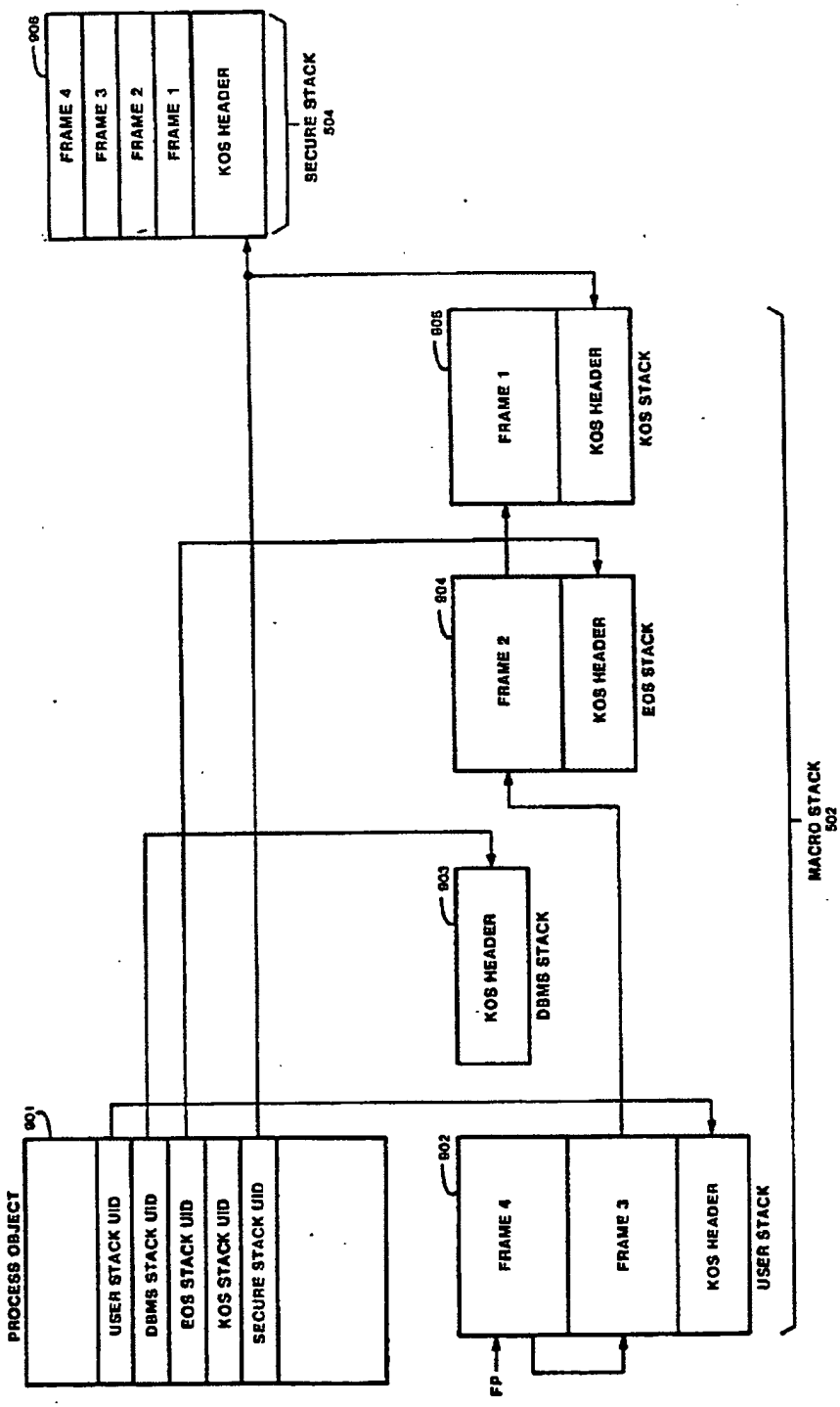


FIG 9

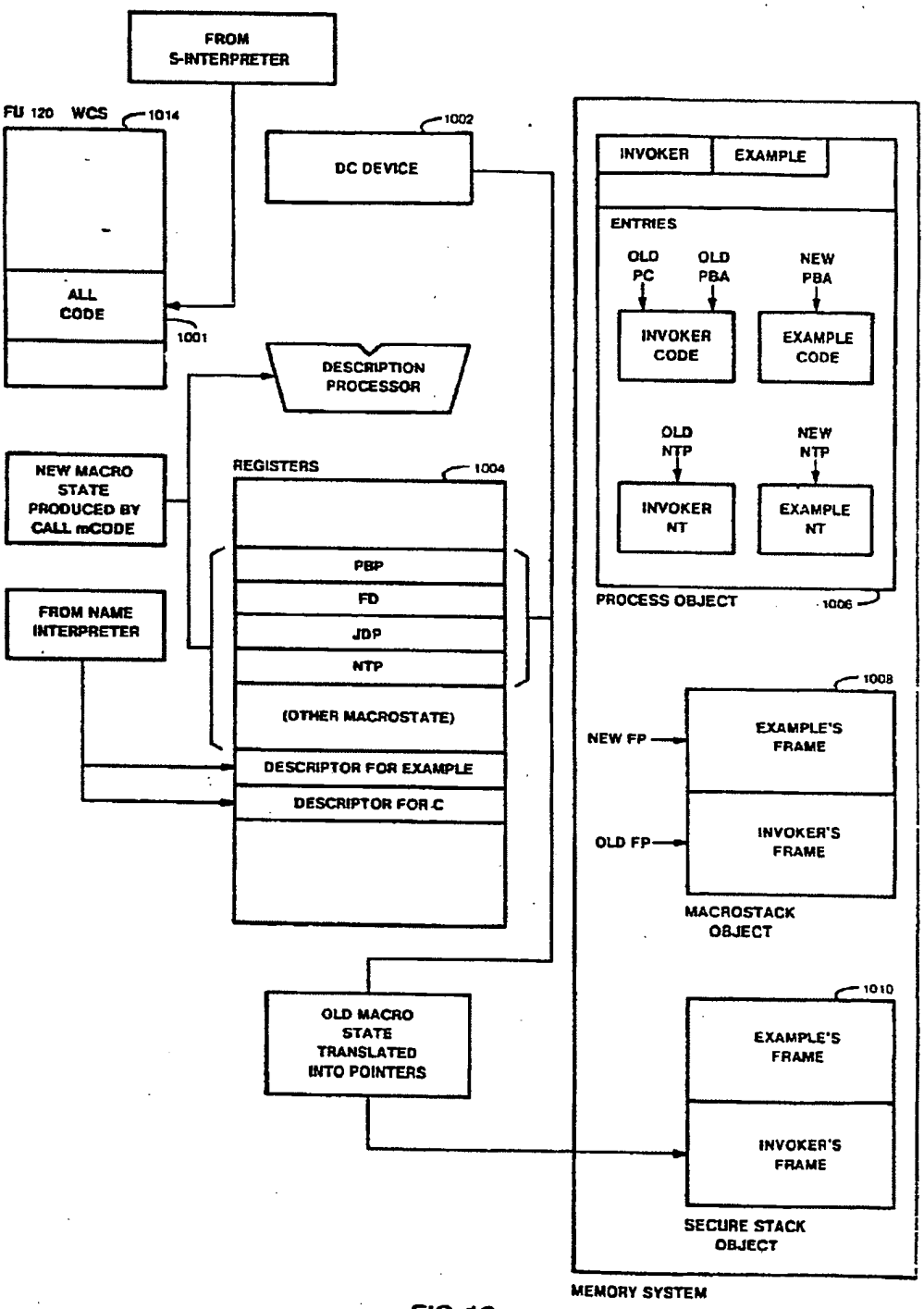


FIG 10

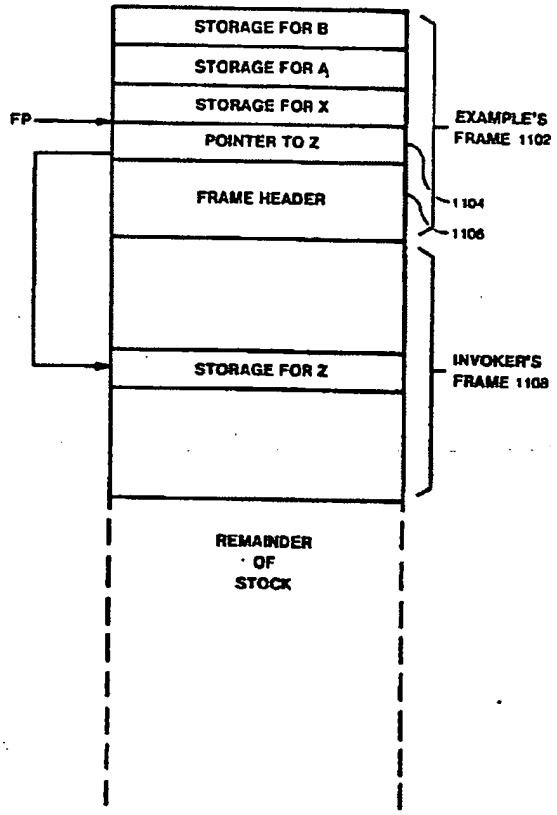


FIG 11

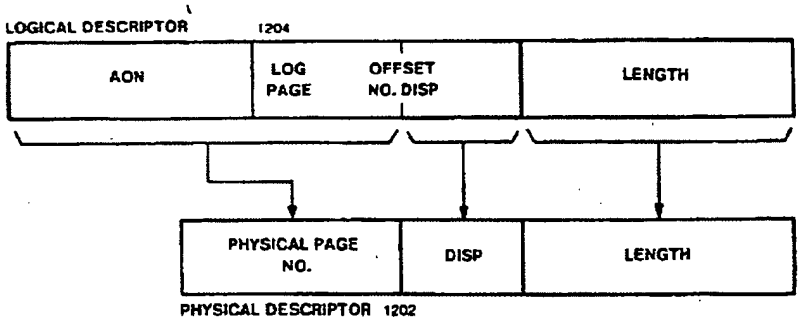


FIG 12

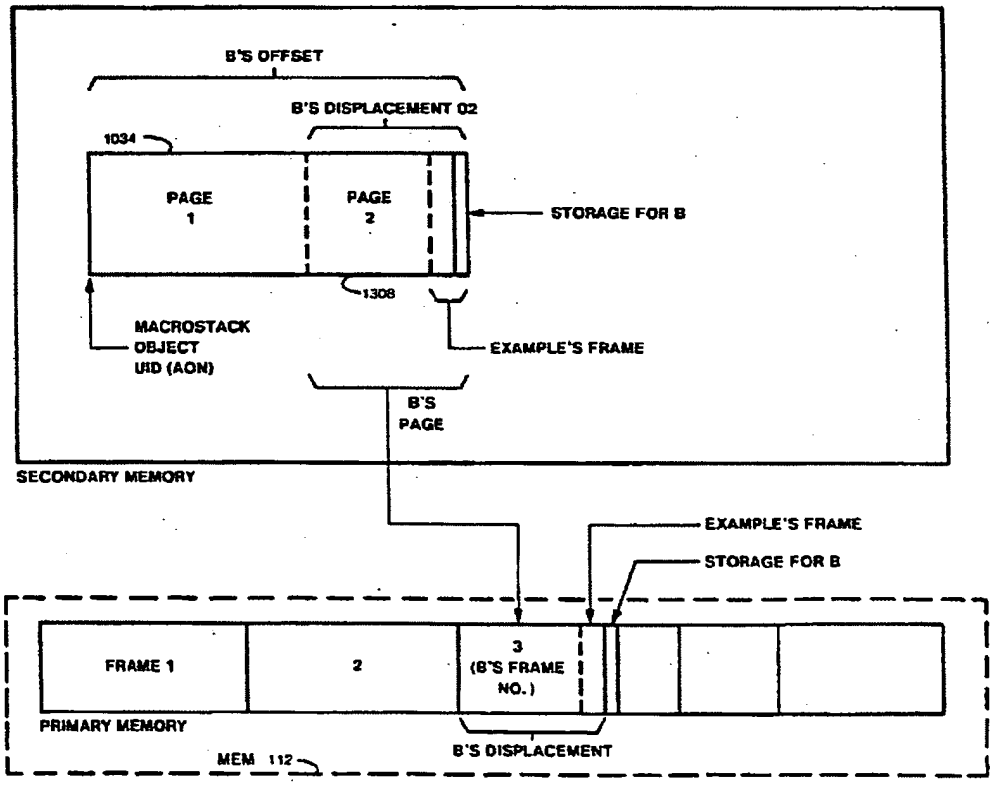


FIG 13

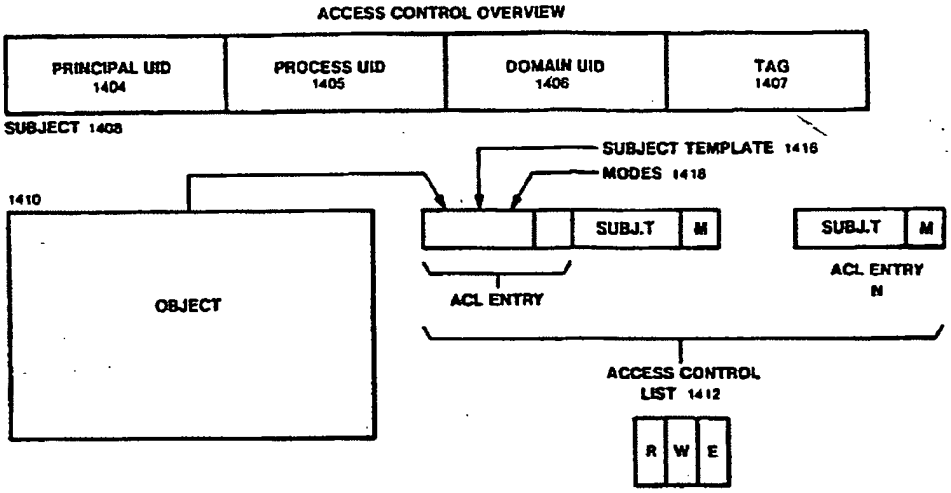


FIG 14

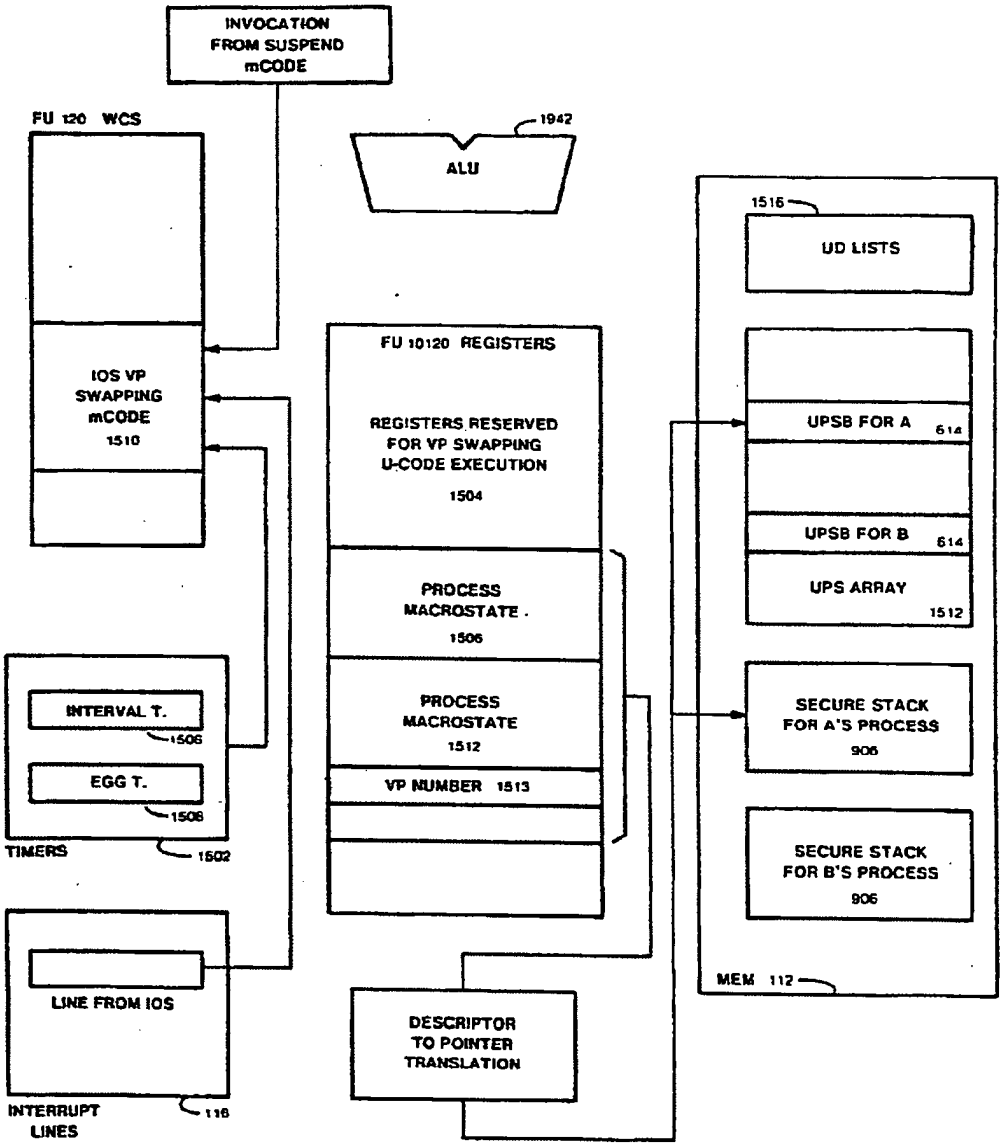


FIG 15

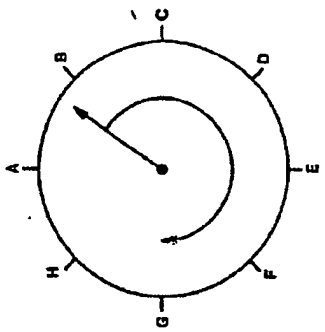


FIG 17

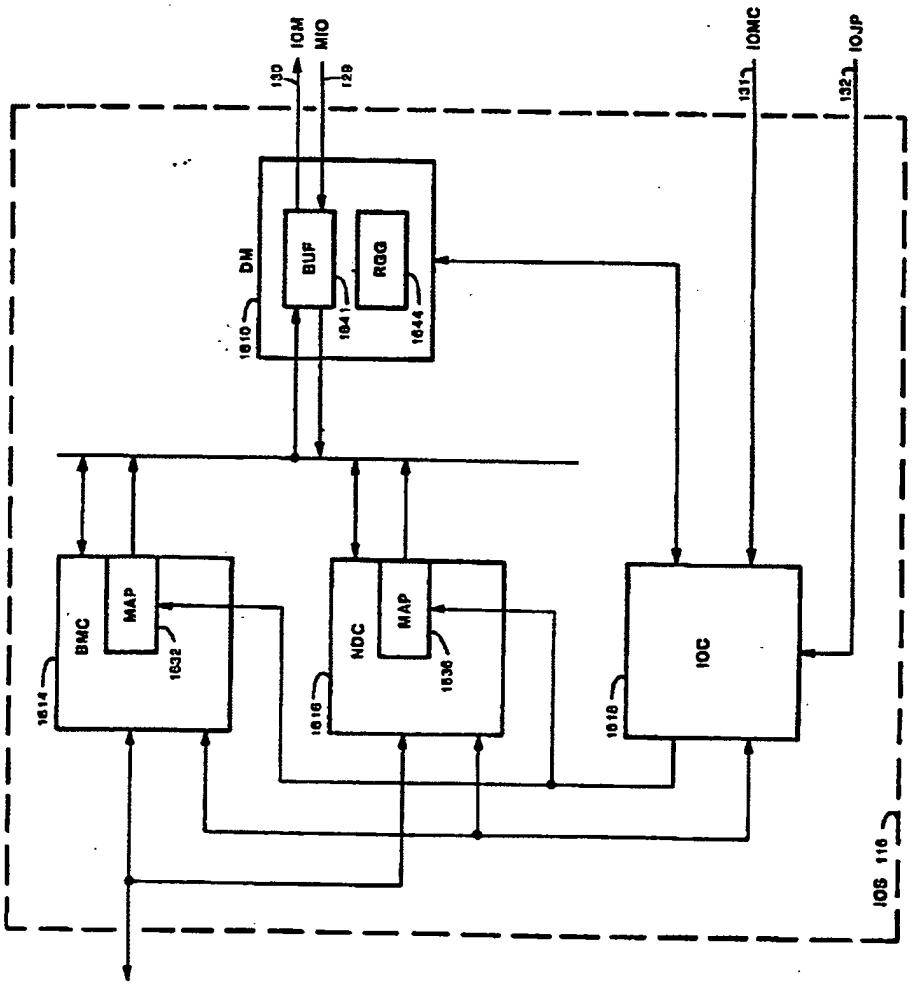


FIG 16

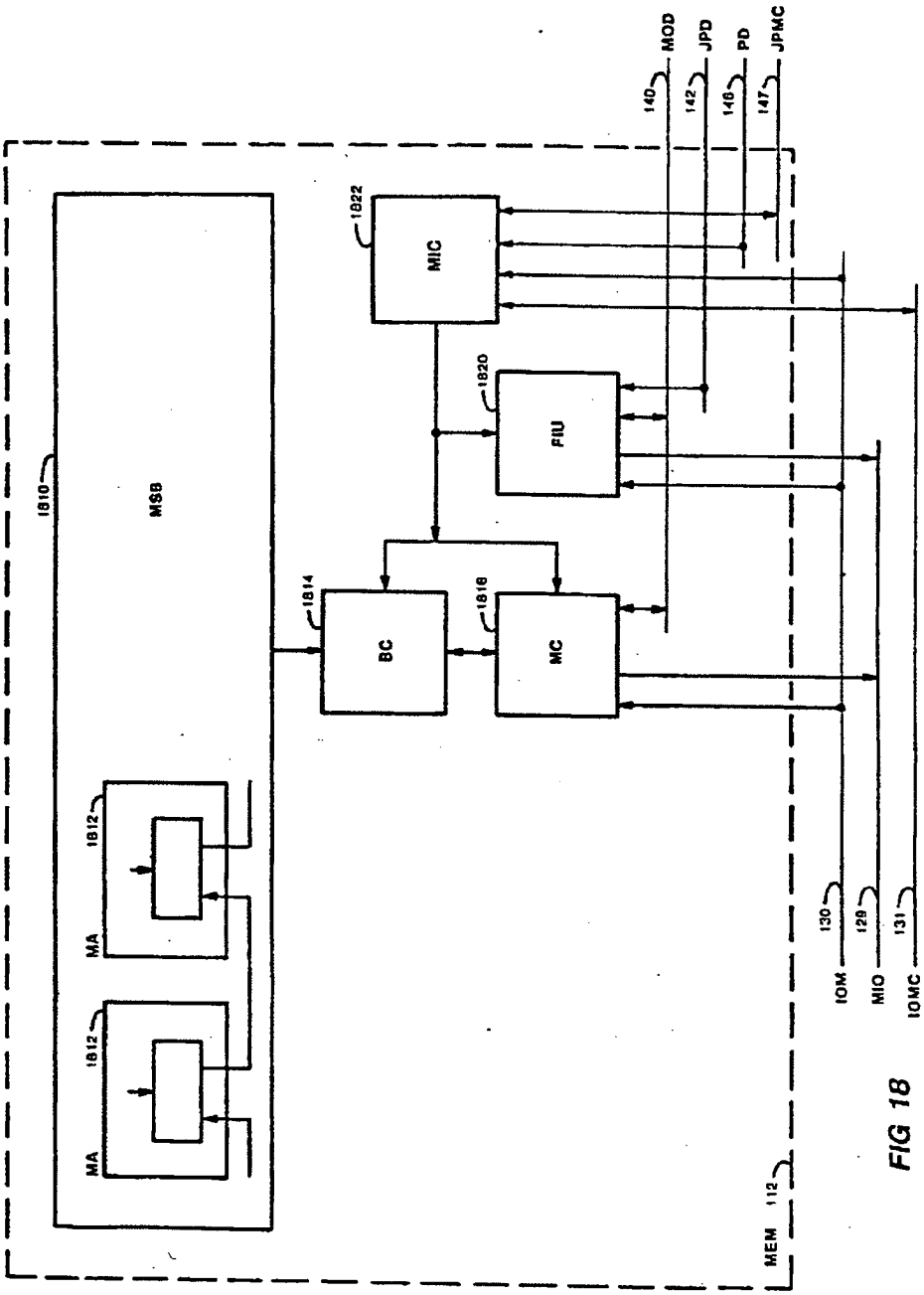


FIG 18

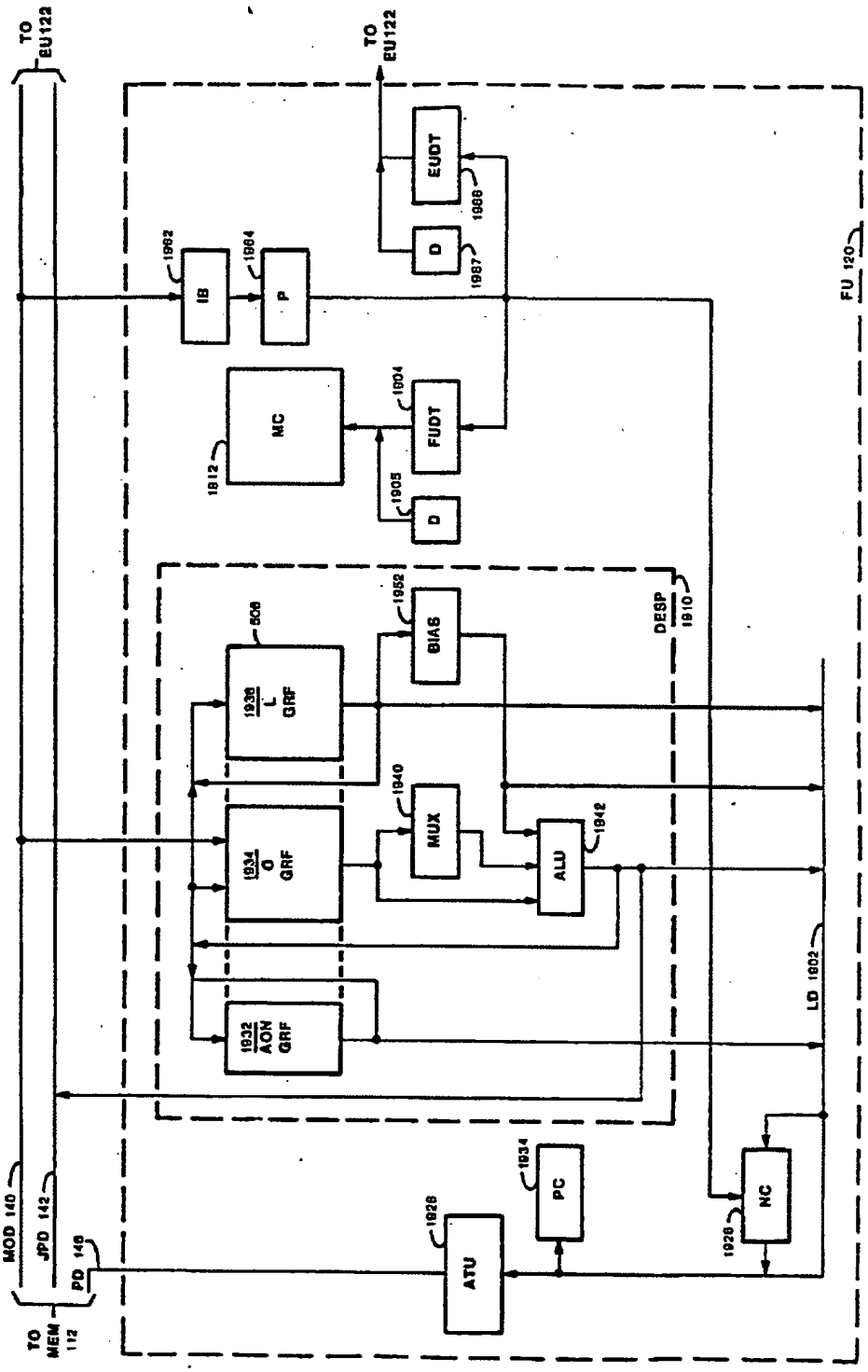


FIG 19

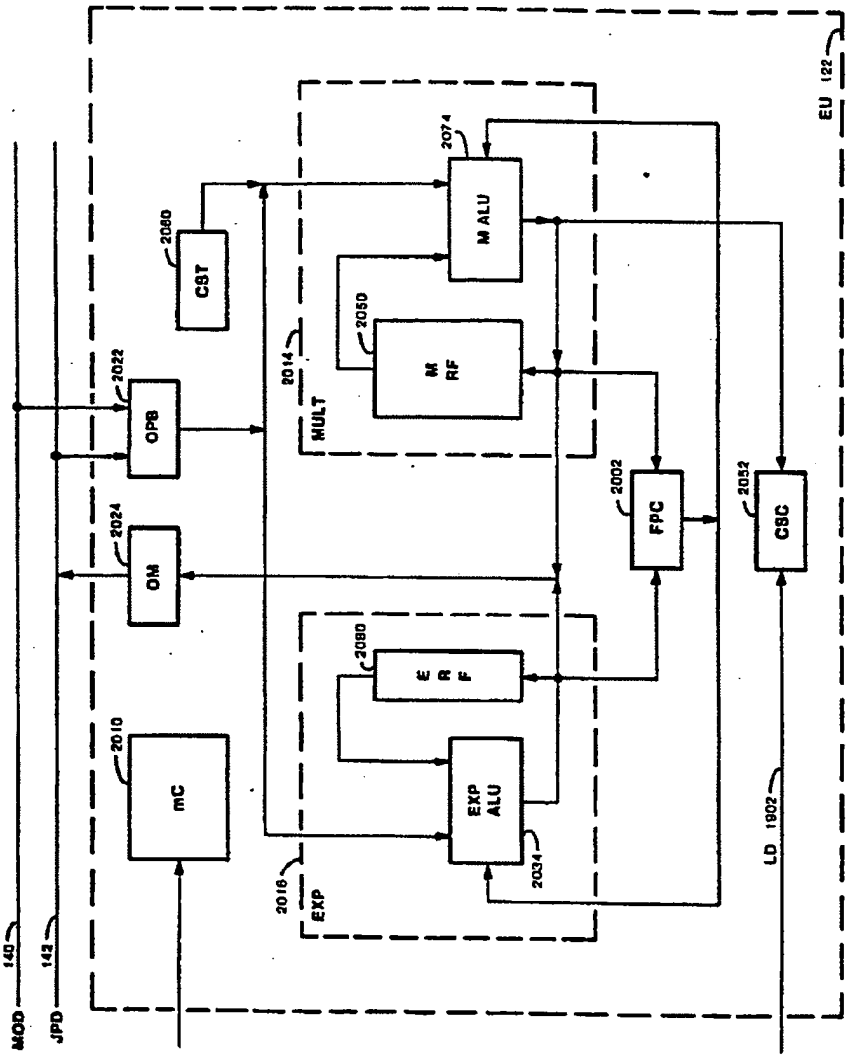


FIG 20

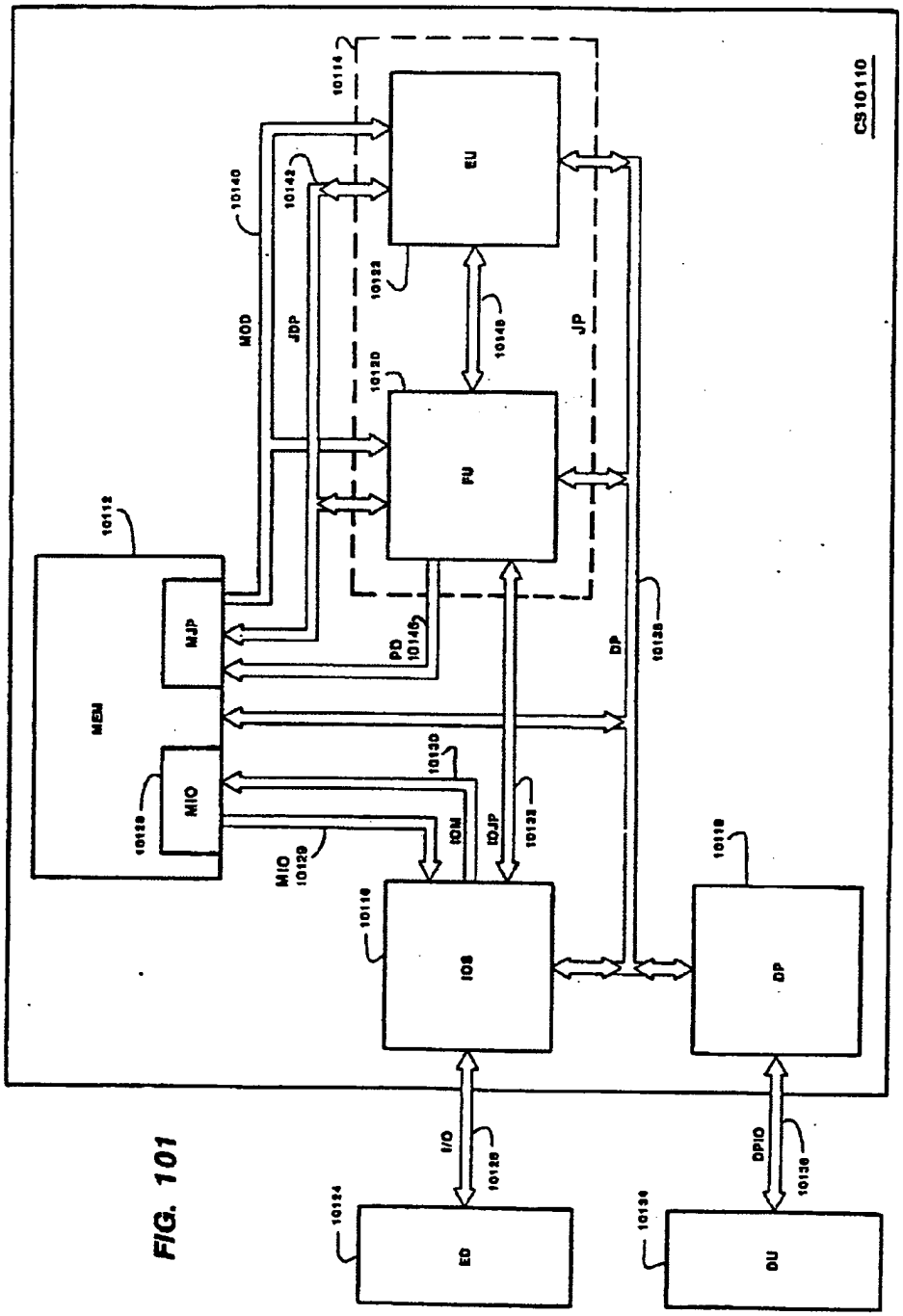


FIG. 101

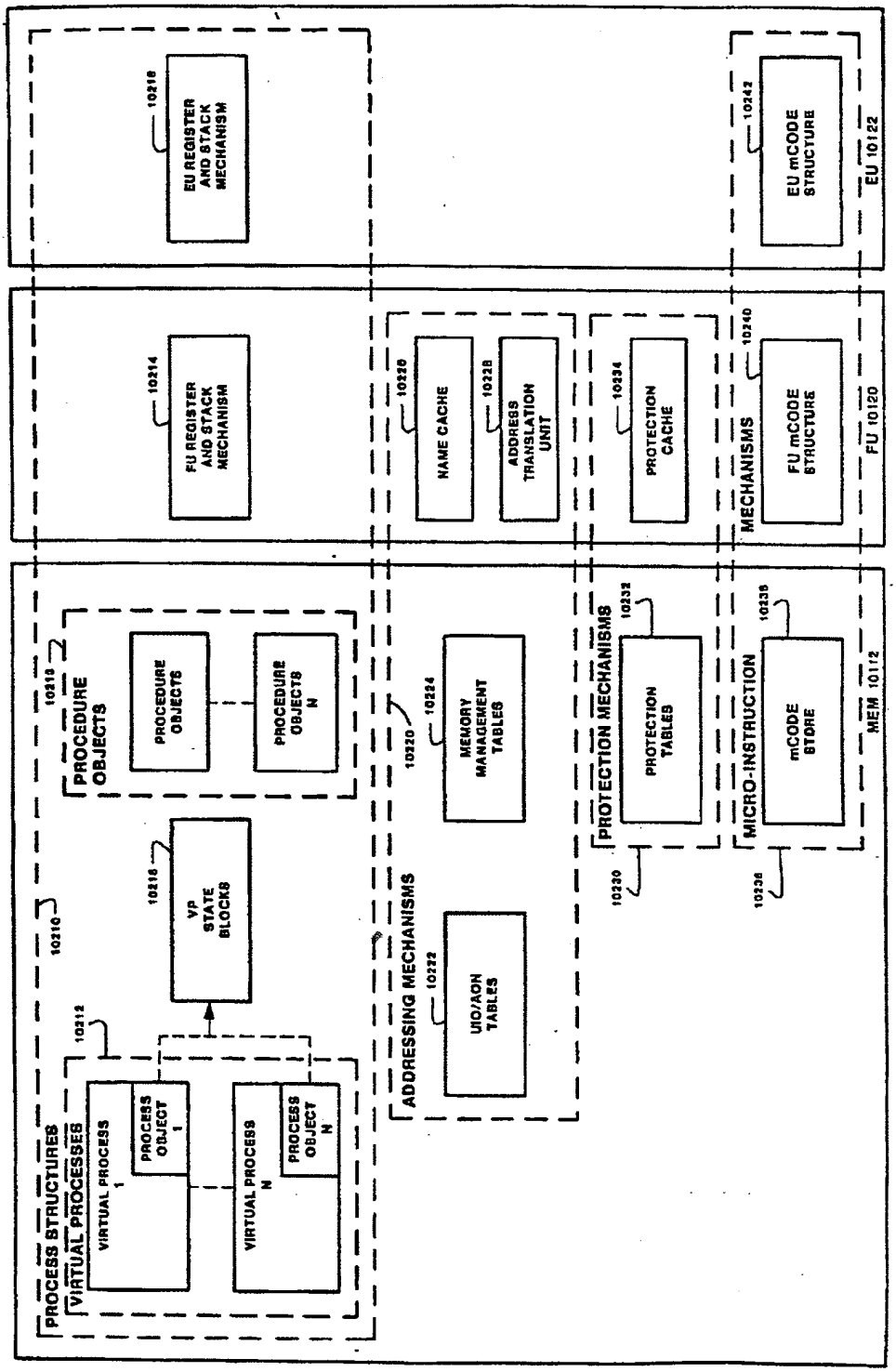


FIG. 102

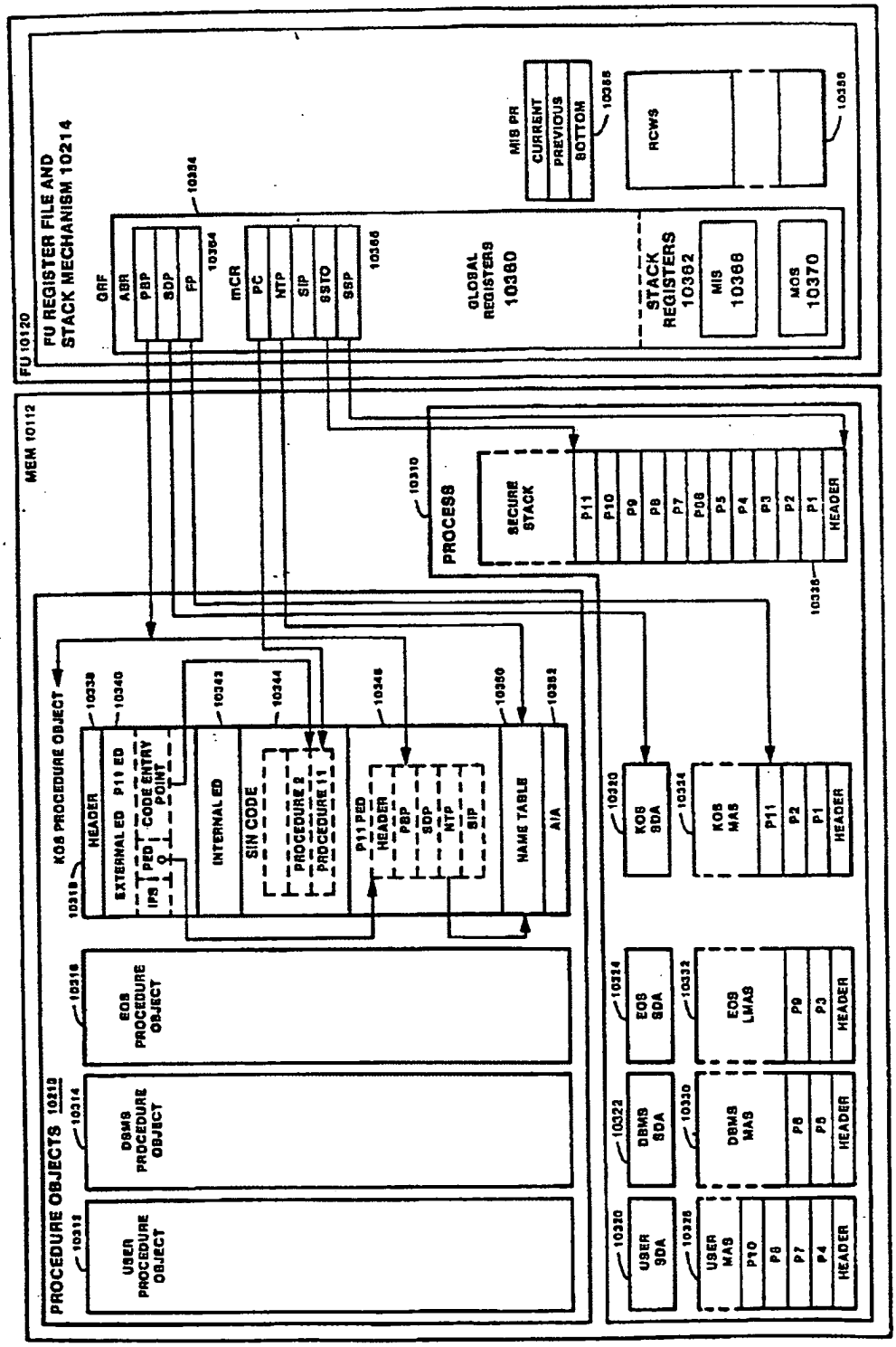


FIG. 103

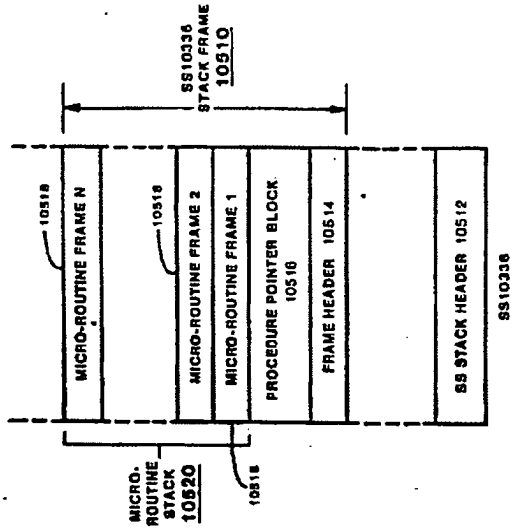


FIG. 105

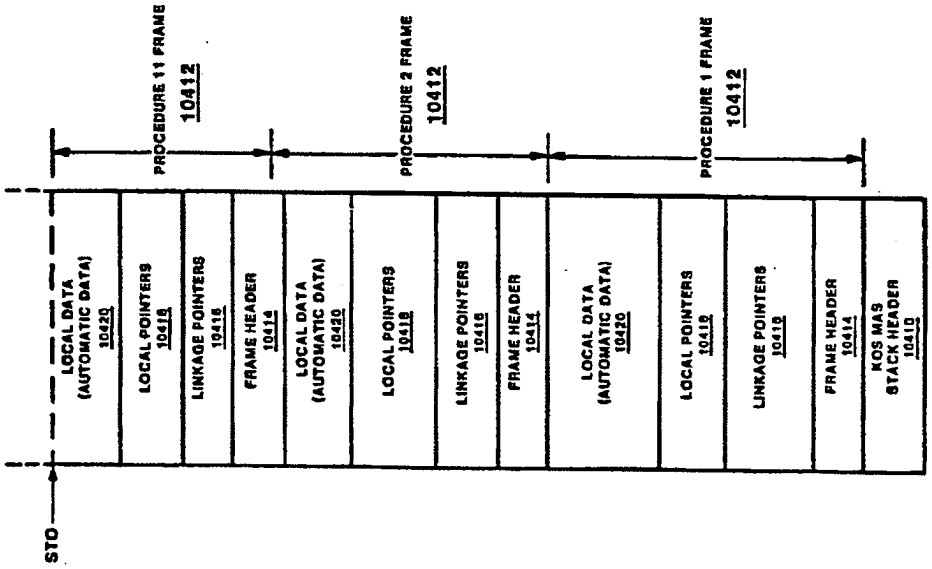


FIG. 104

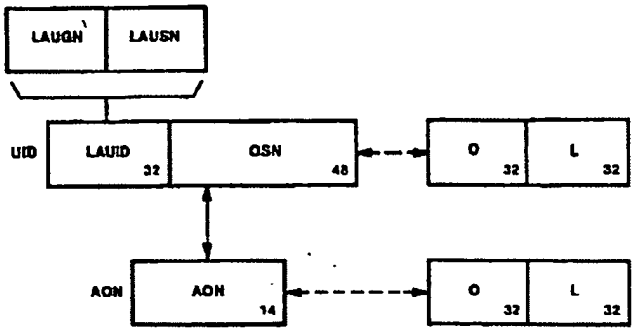


FIG. 106A

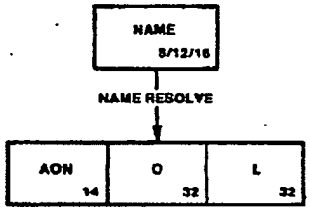


FIG. 106B

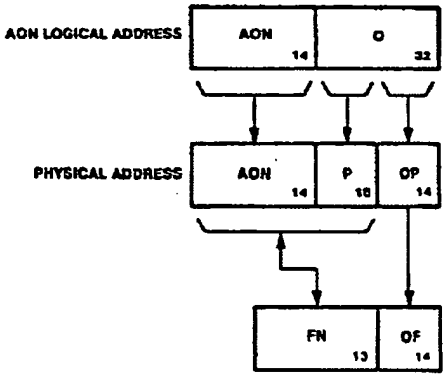


FIG. 106C

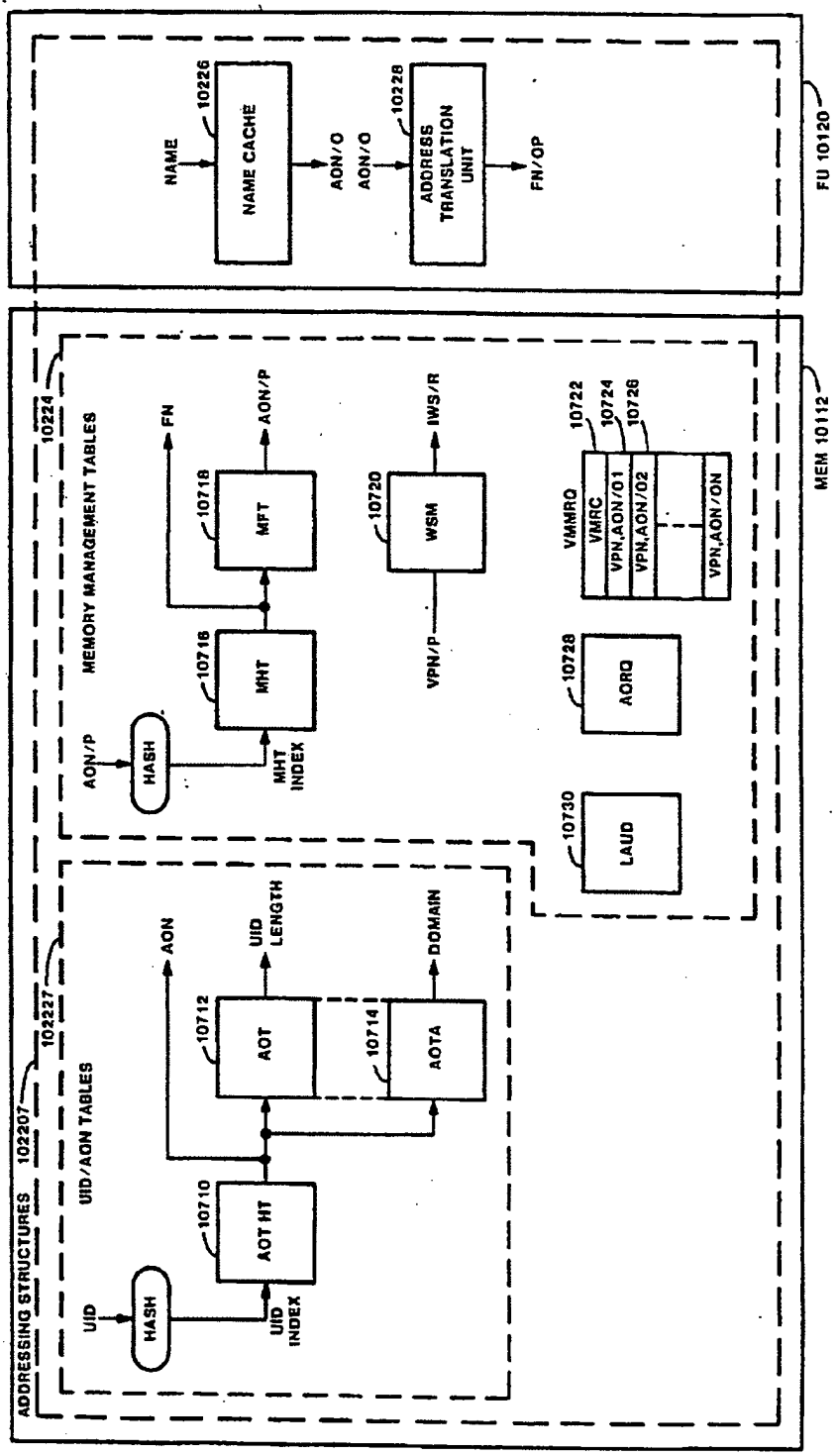


FIG 107

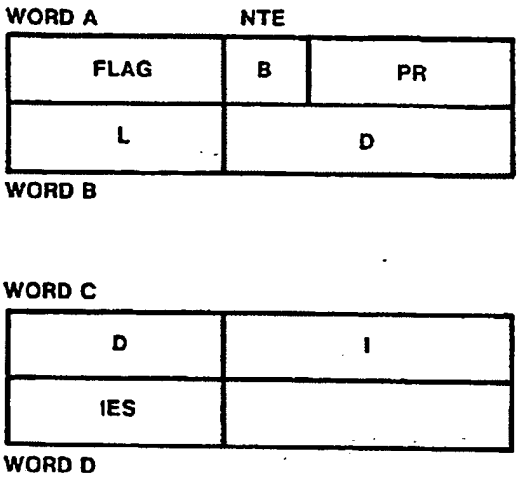


FIG. 108

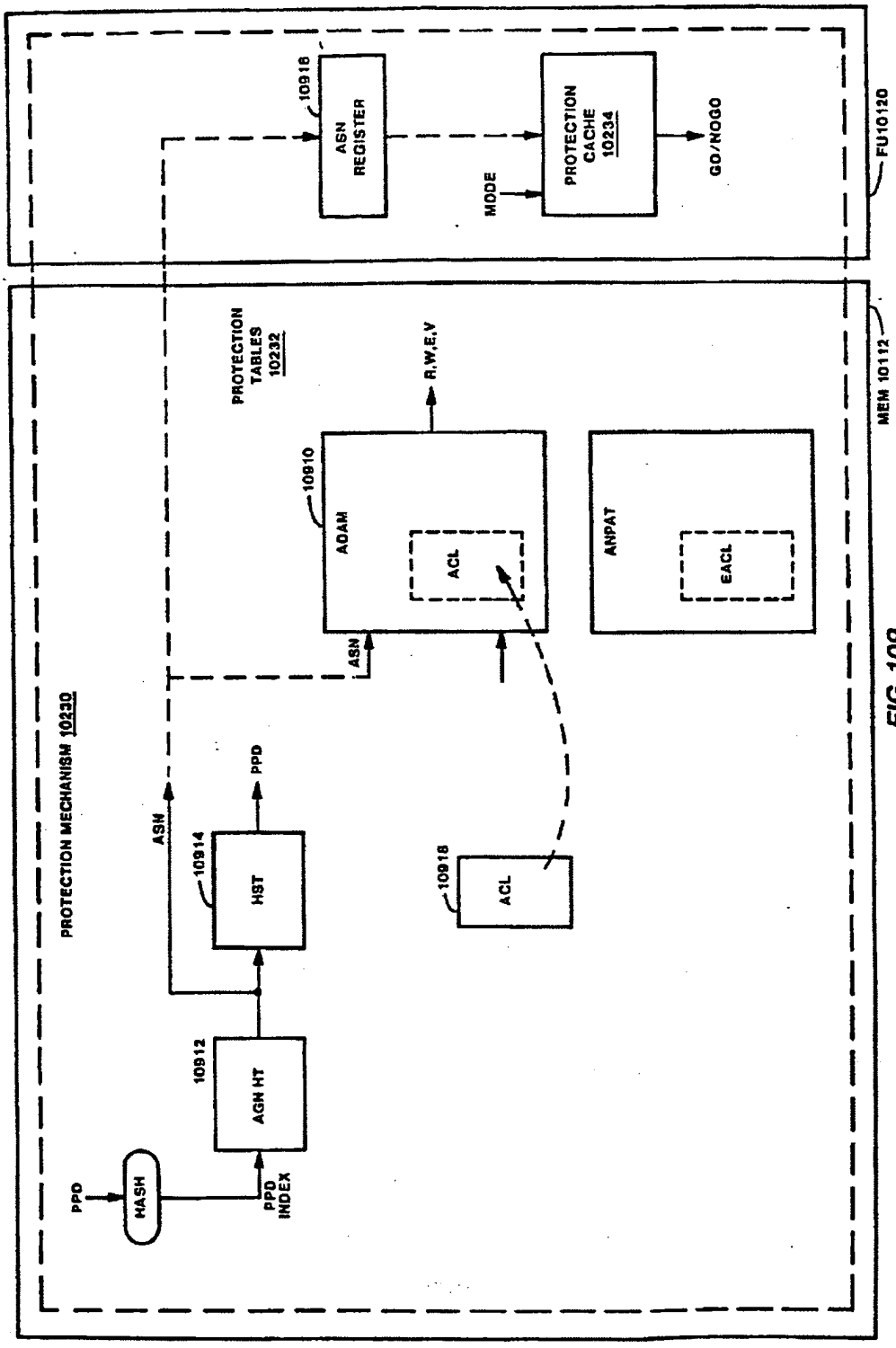


FIG 109

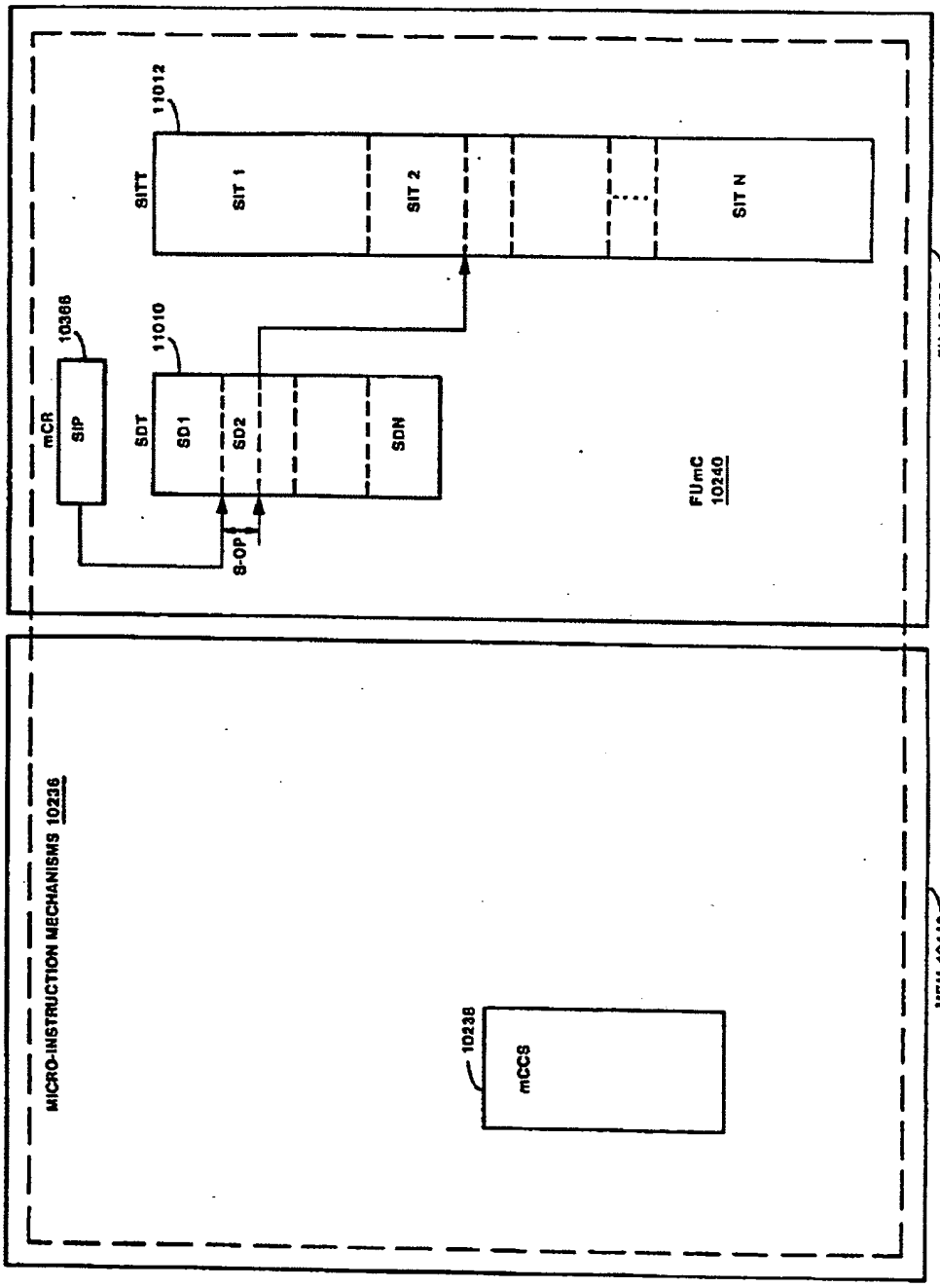


FIG 110

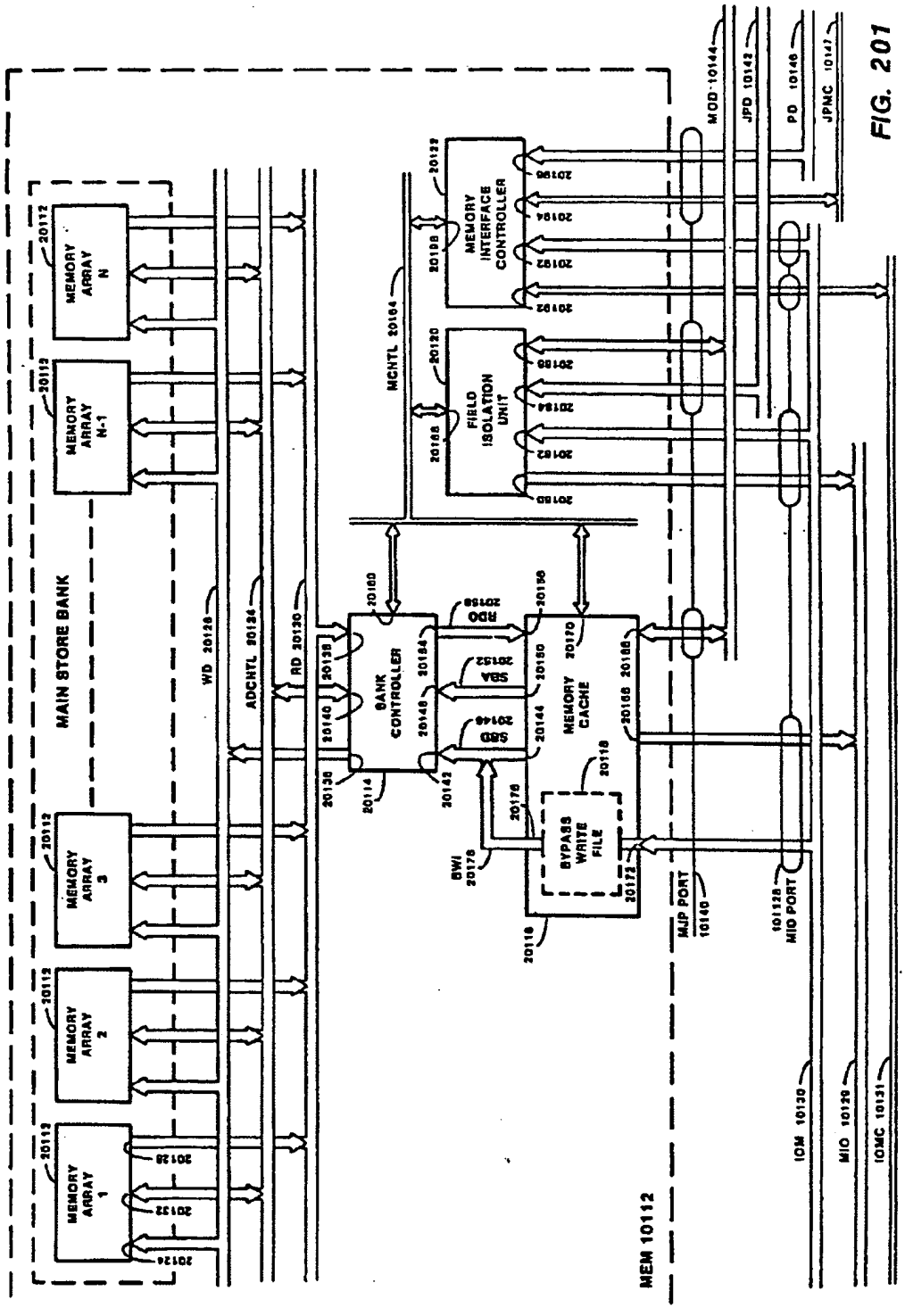


FIG. 201

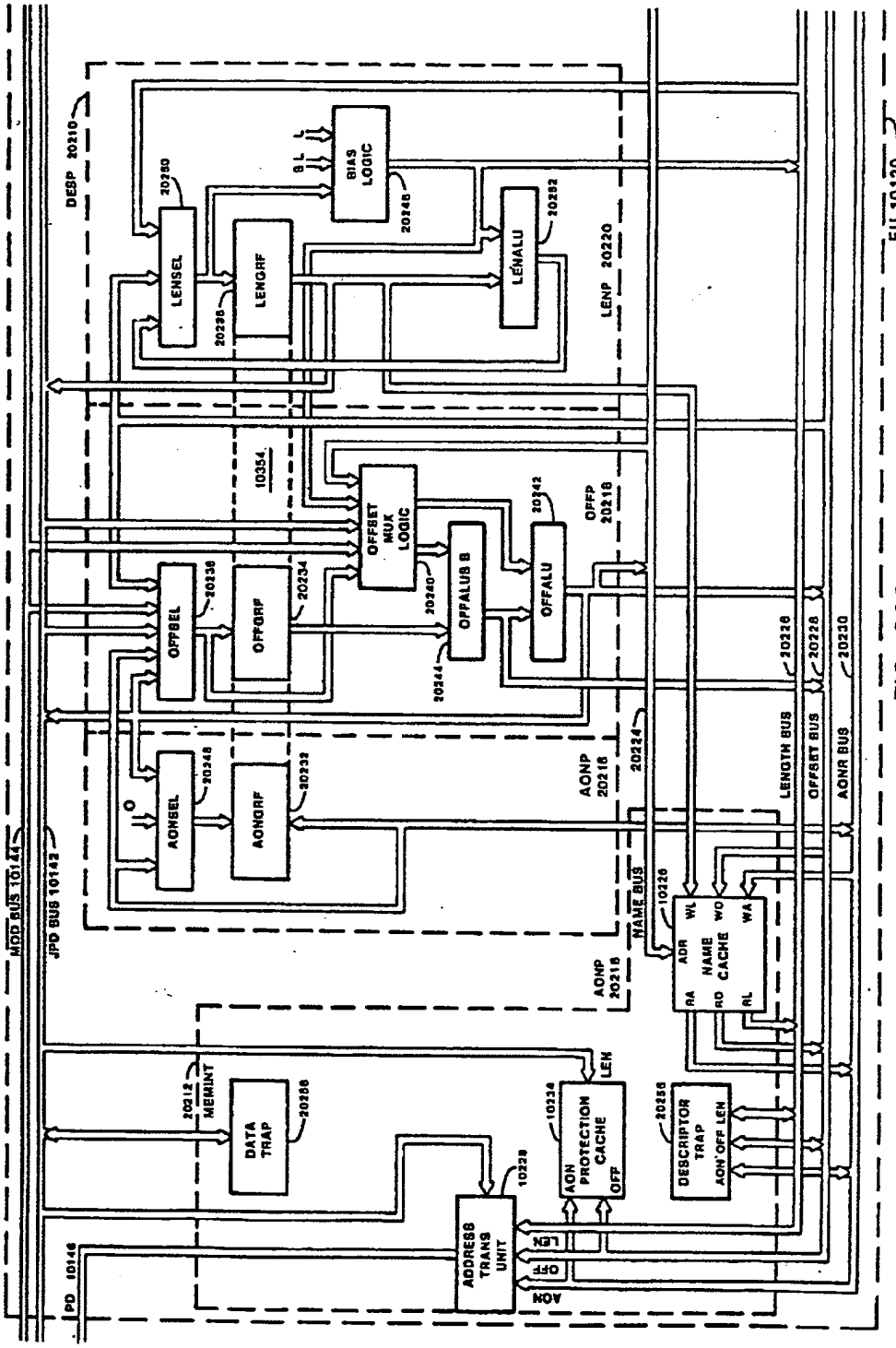


FIG. 202

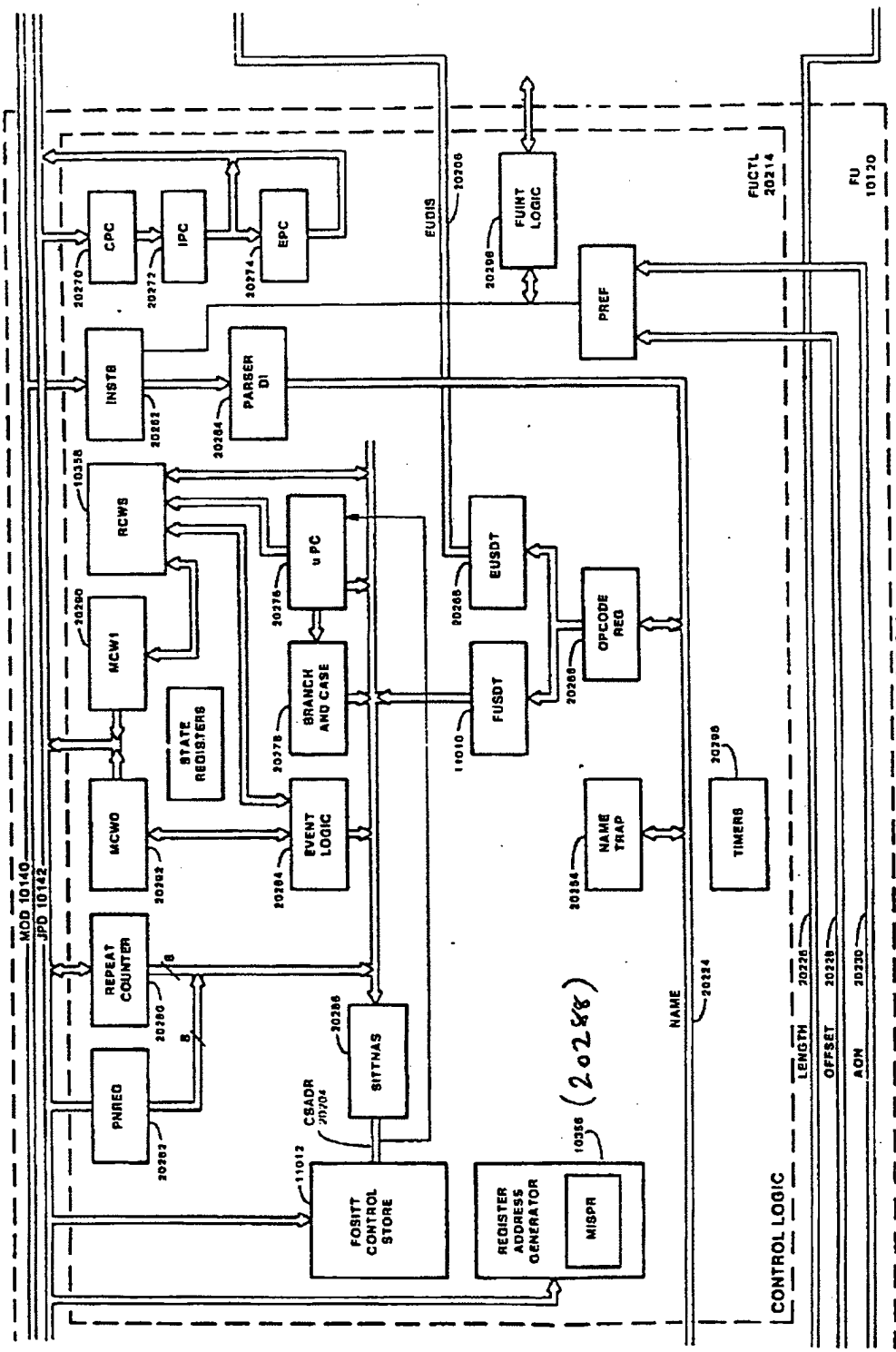


FIG. 202A

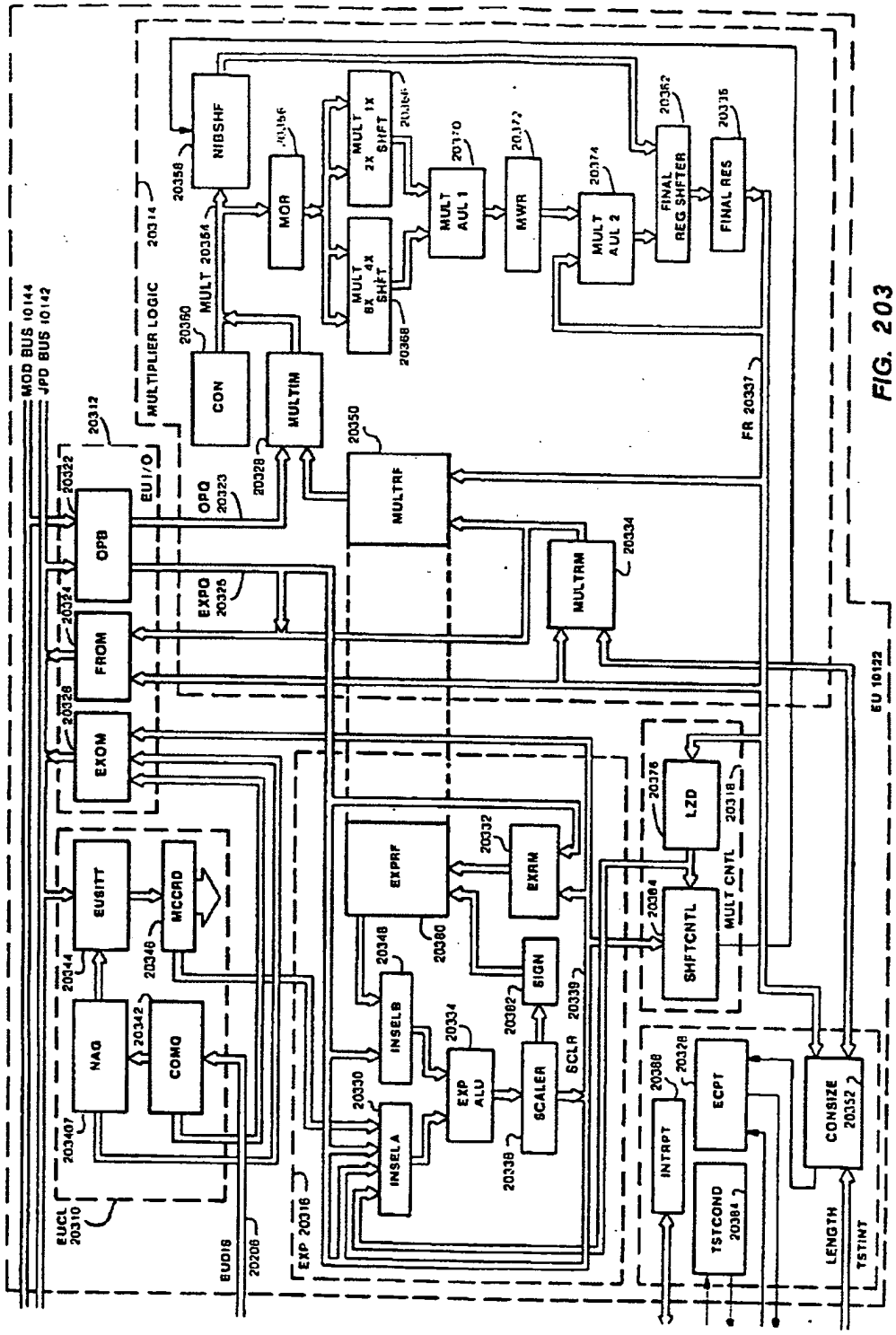


FIG. 203

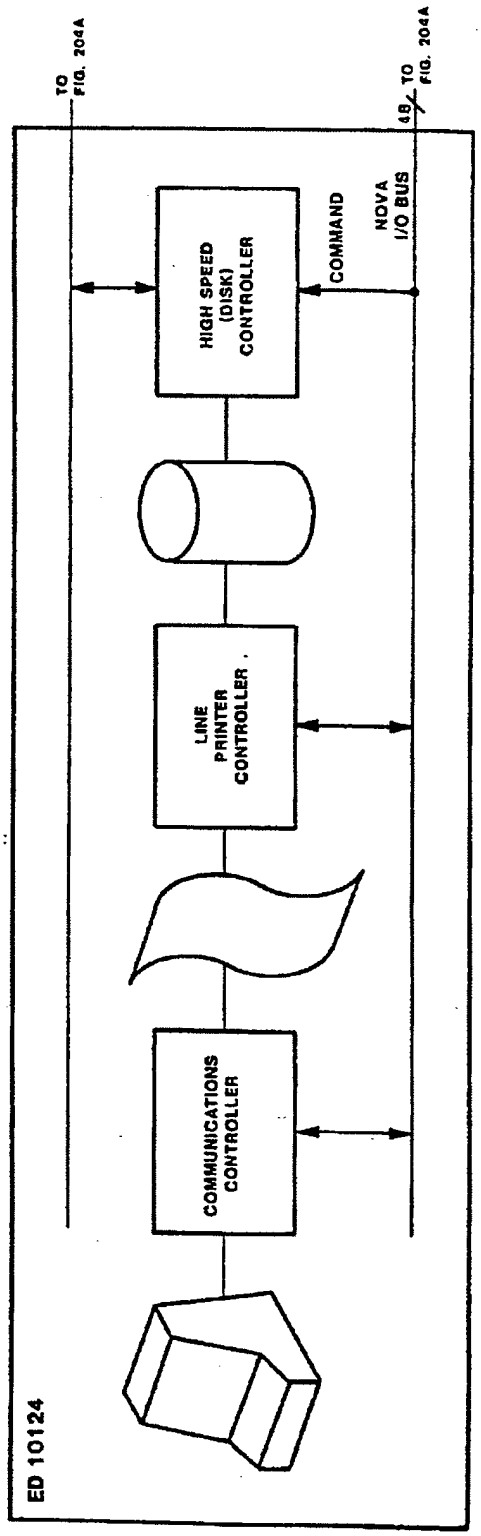


FIG. 204

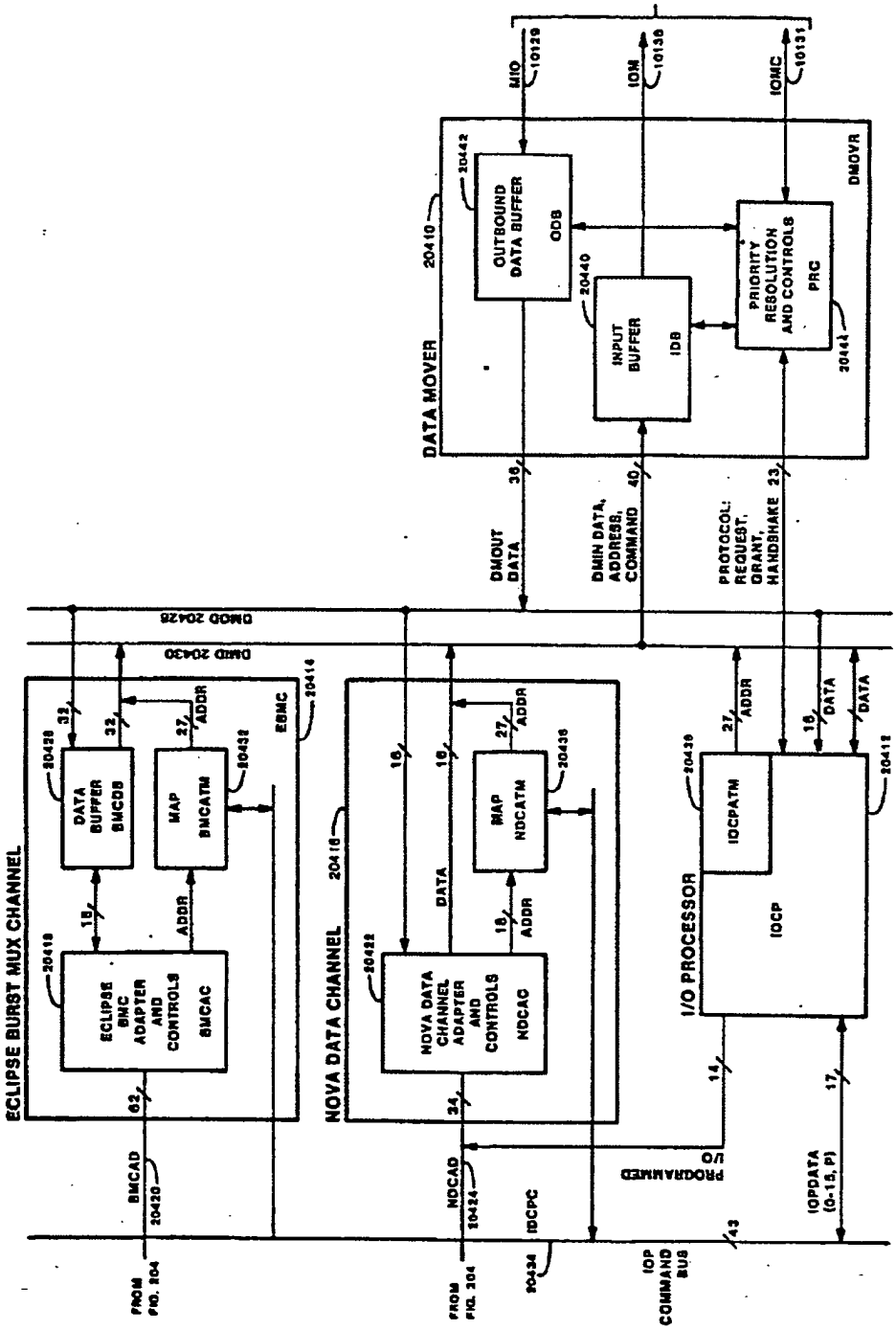


FIG. 204A

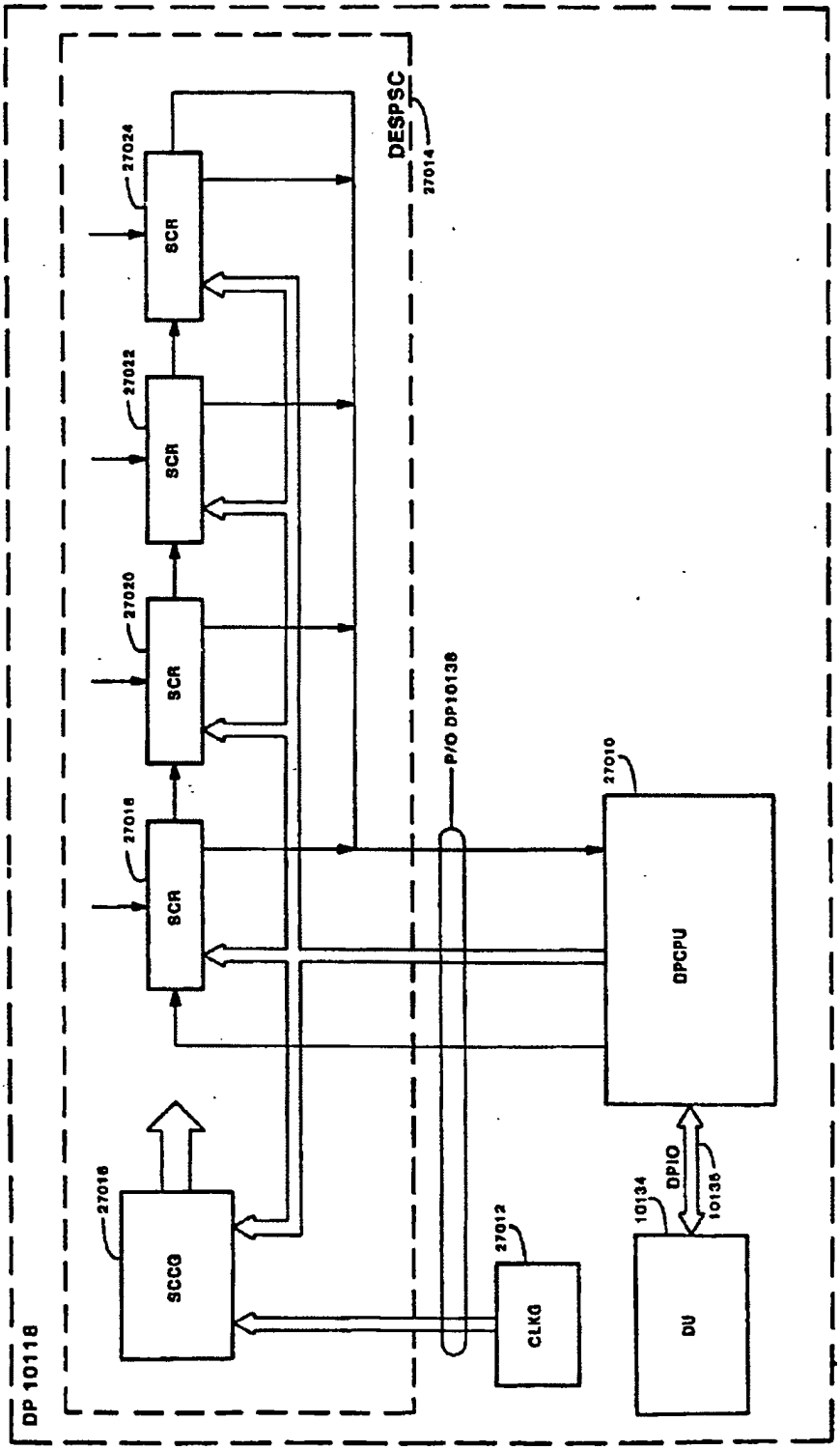


FIG. 205

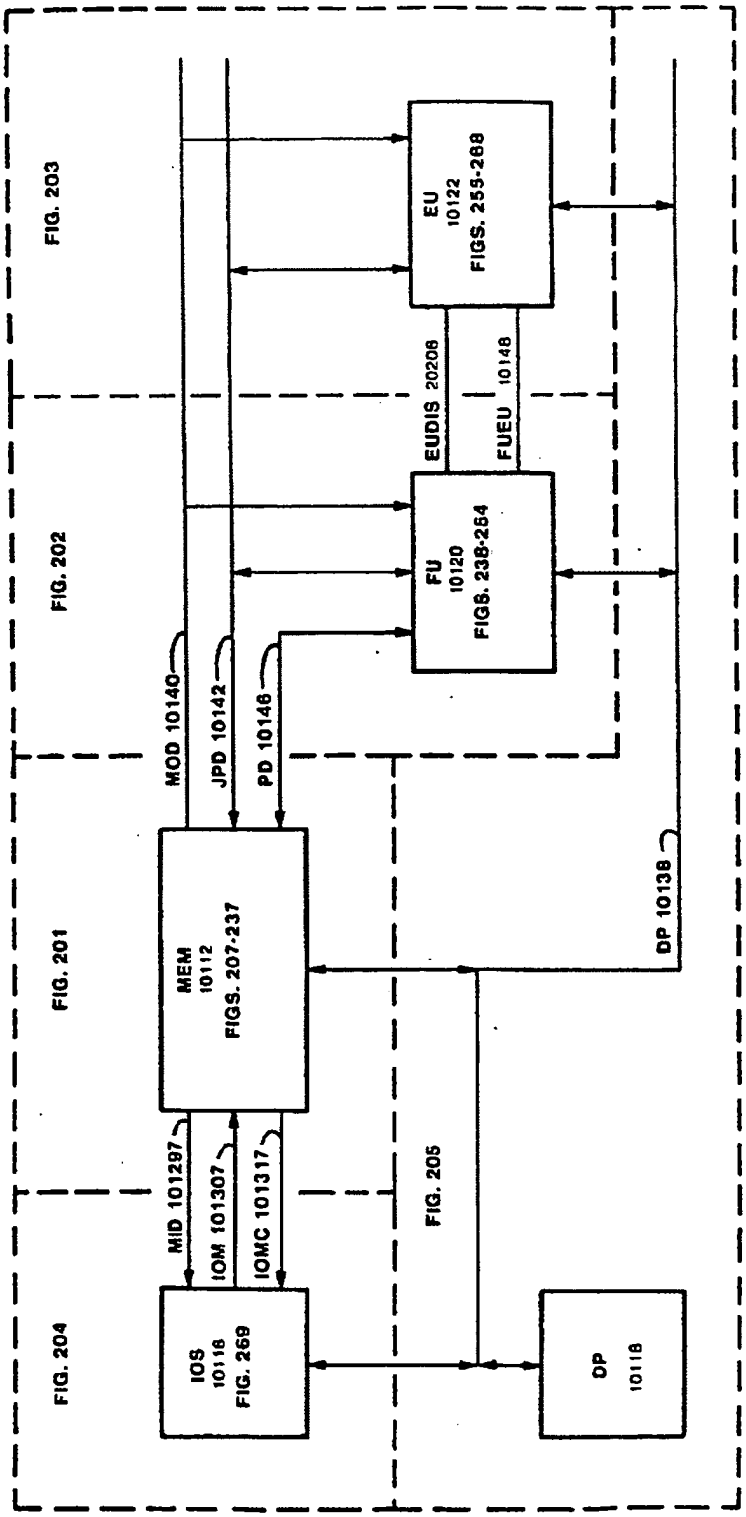


FIG. 206

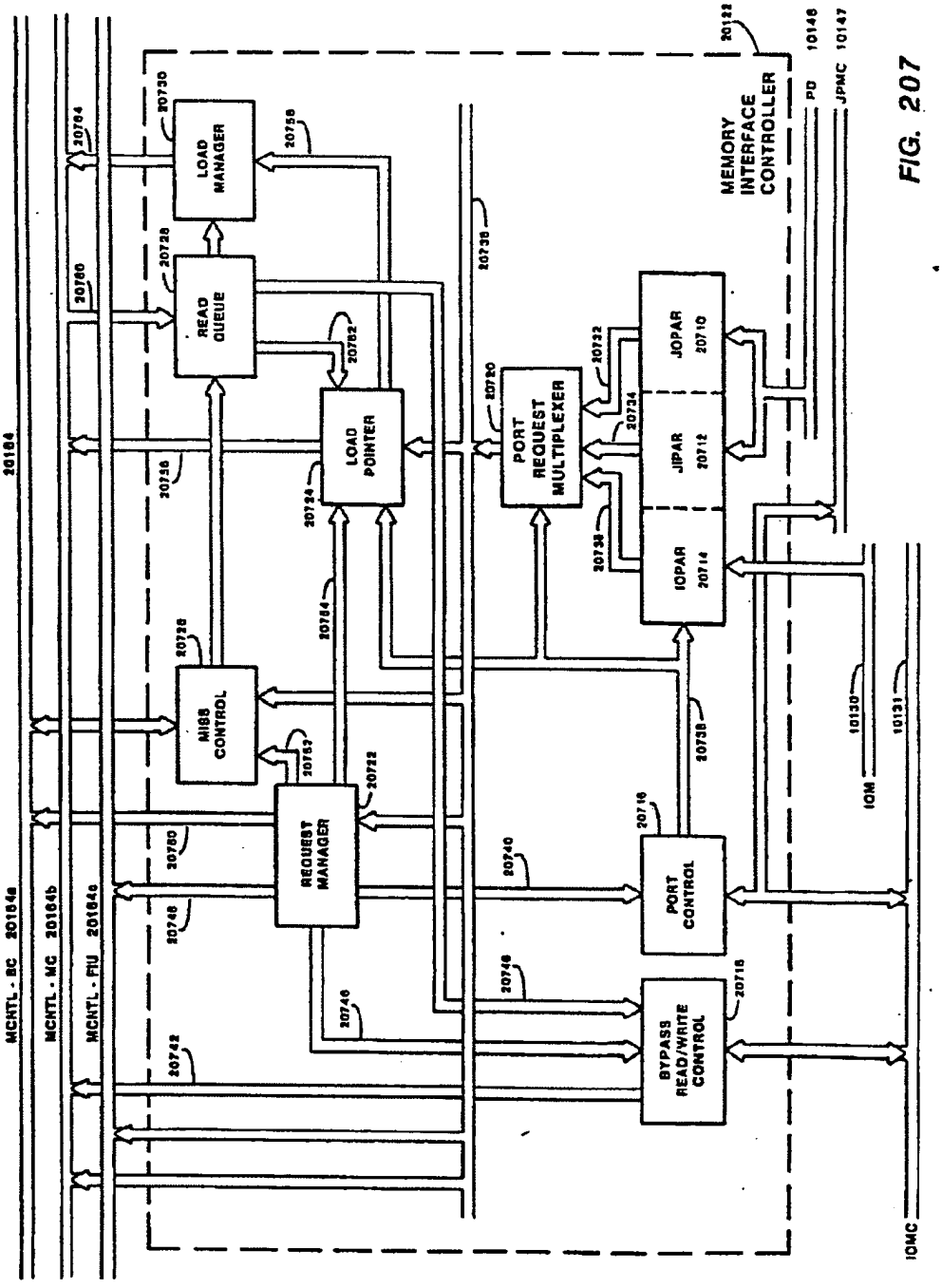


FIG. 207

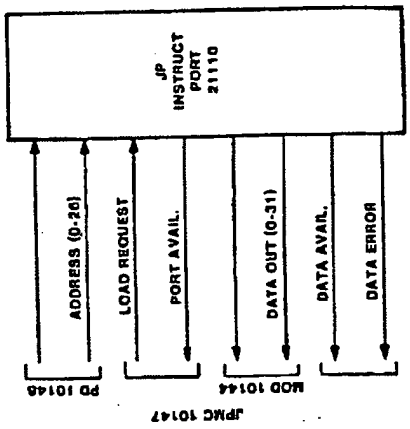


FIG. 211

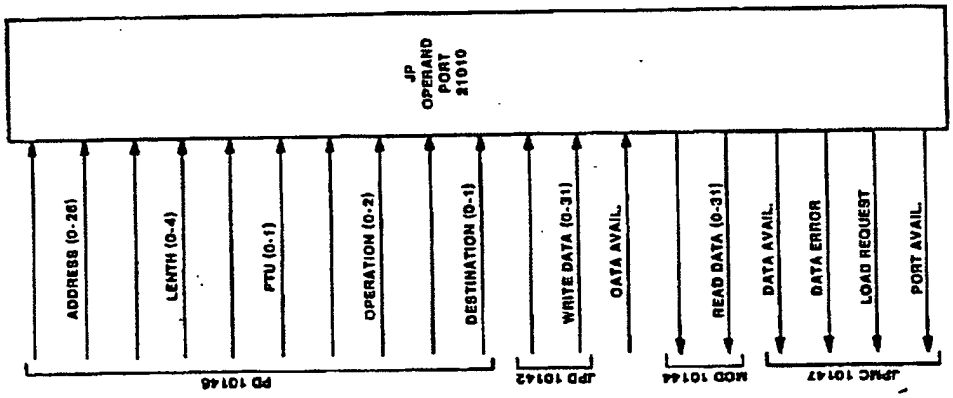


FIG. 210

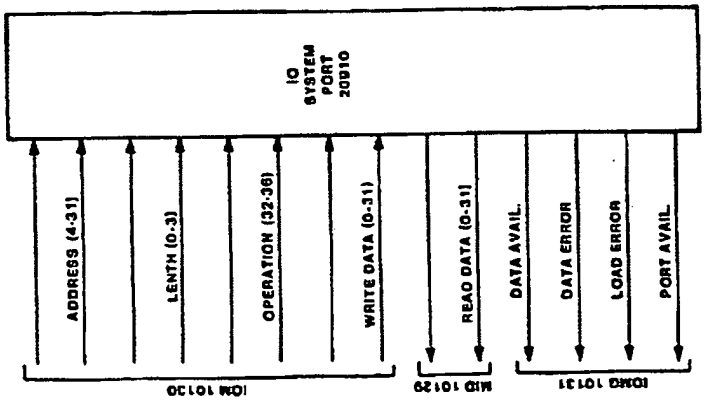


FIG. 209

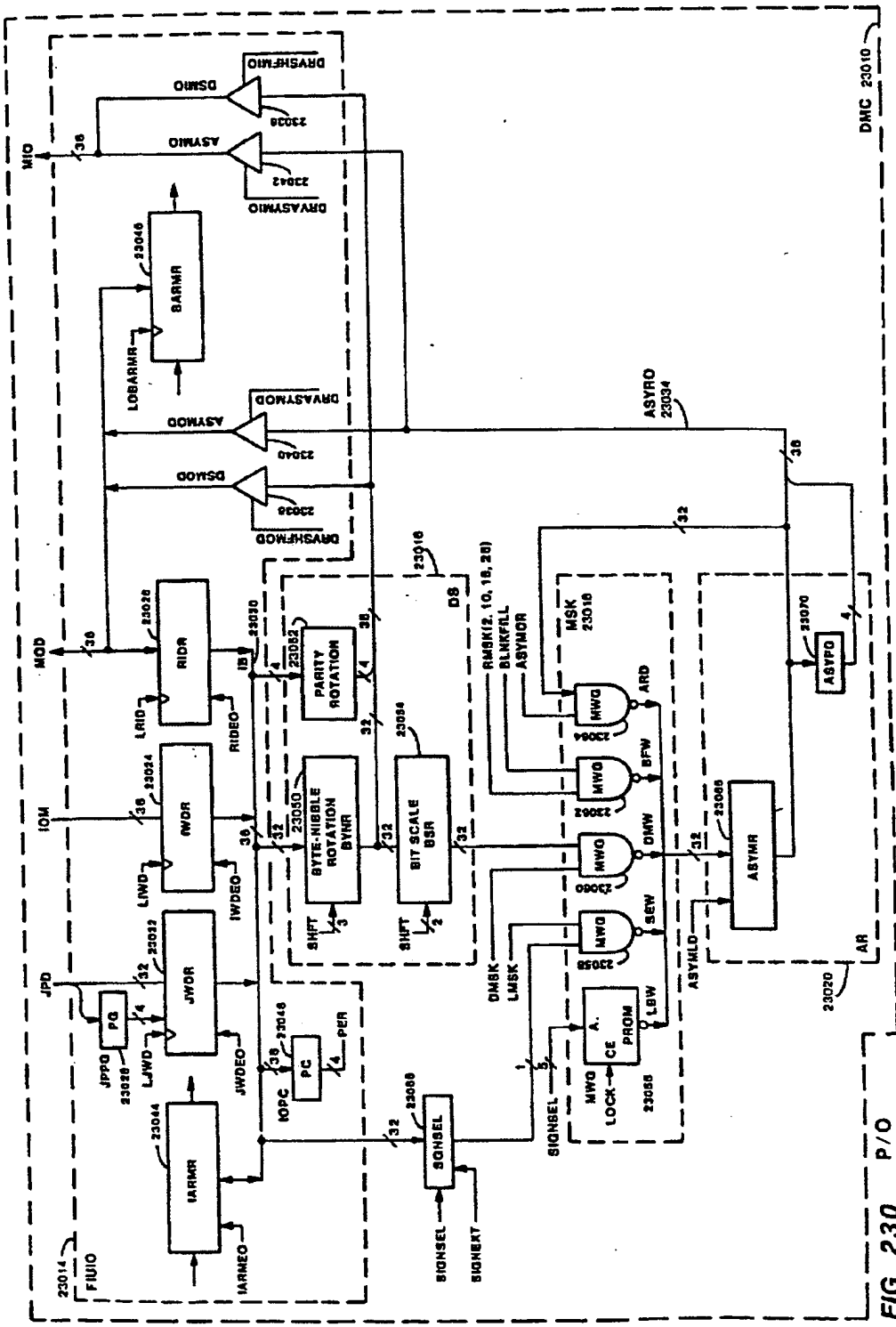


FIG. 230

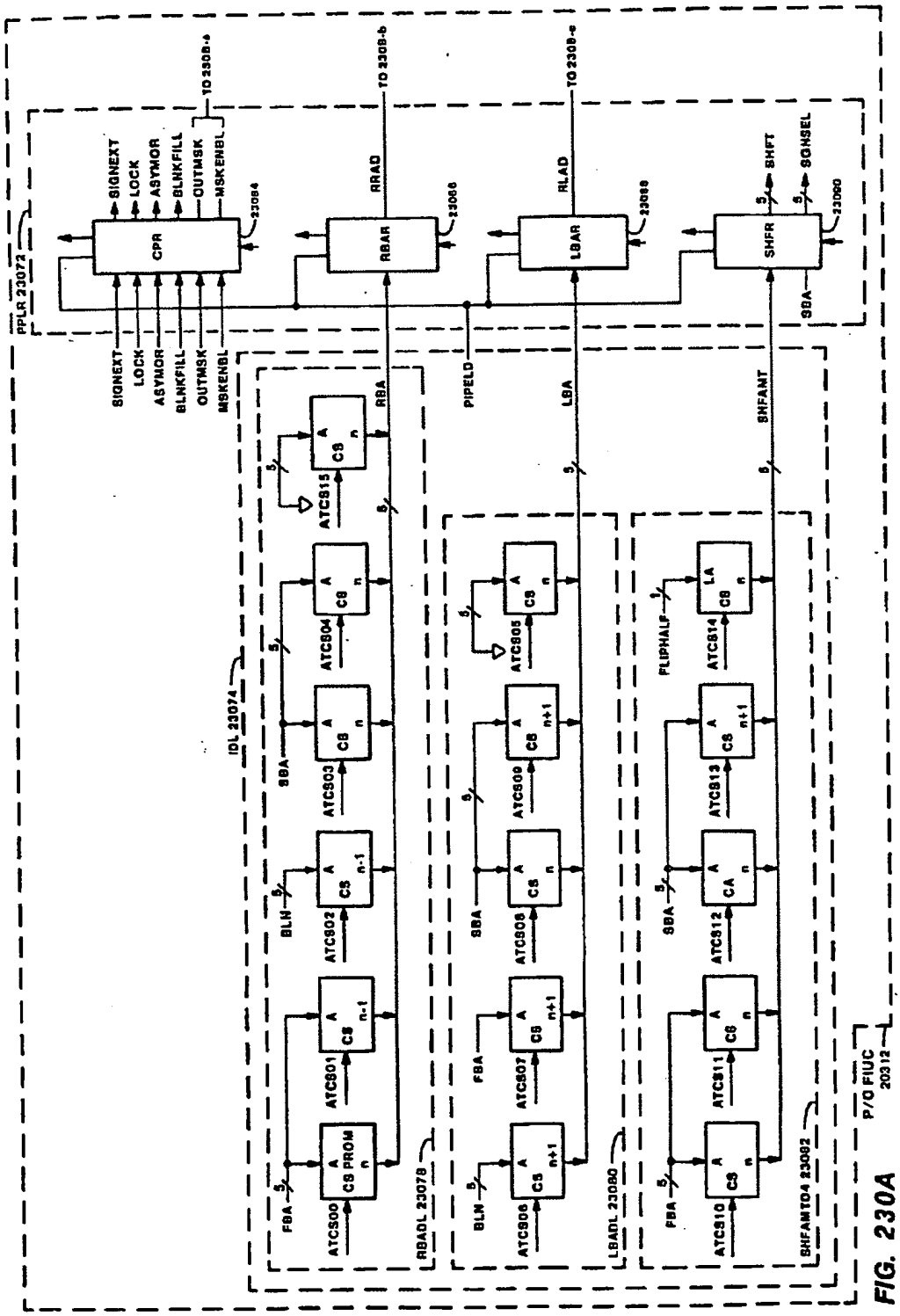


FIG. 230A

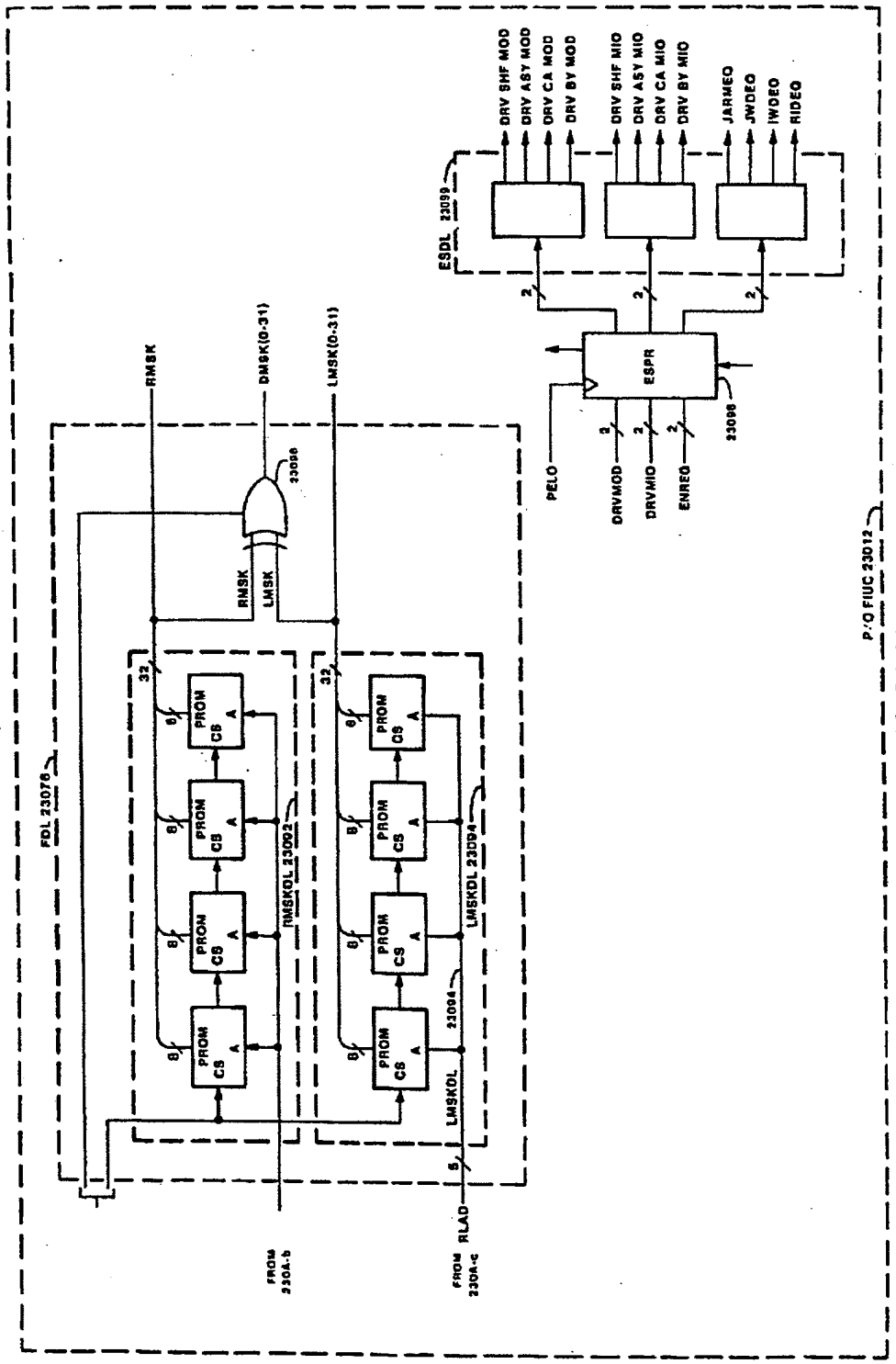


FIG 230B

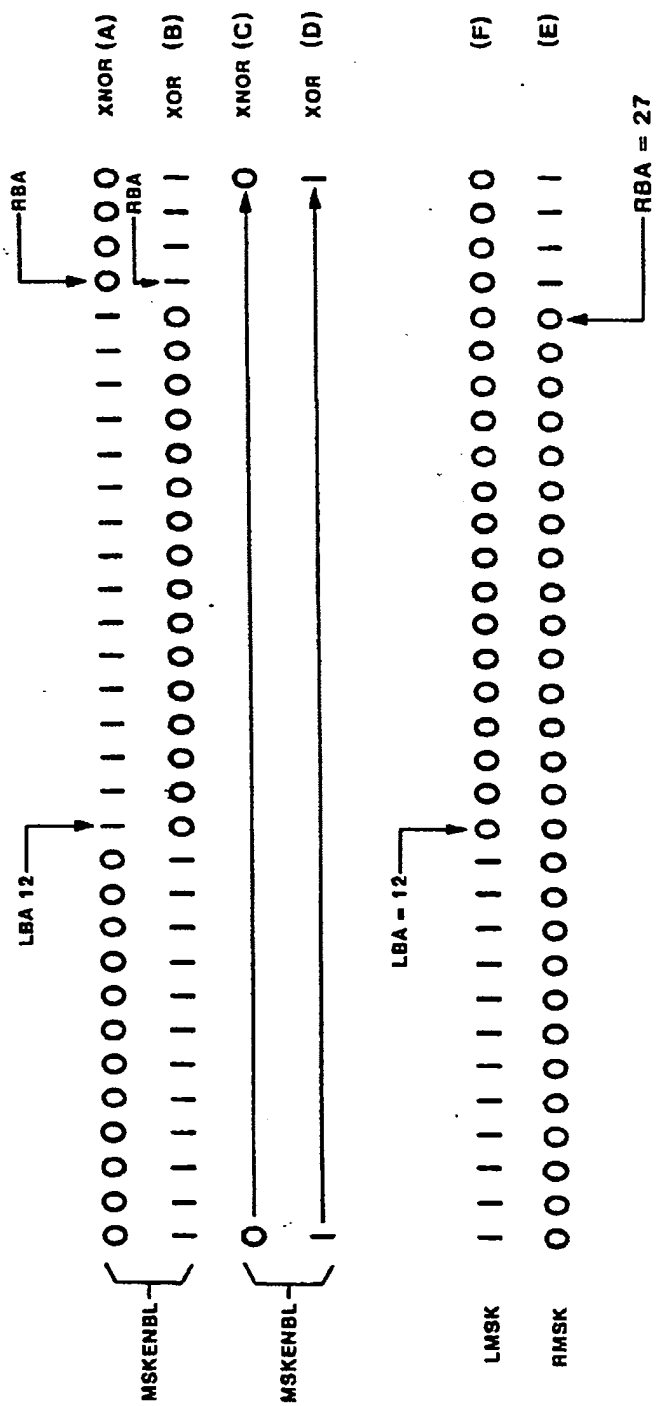


FIG. 231

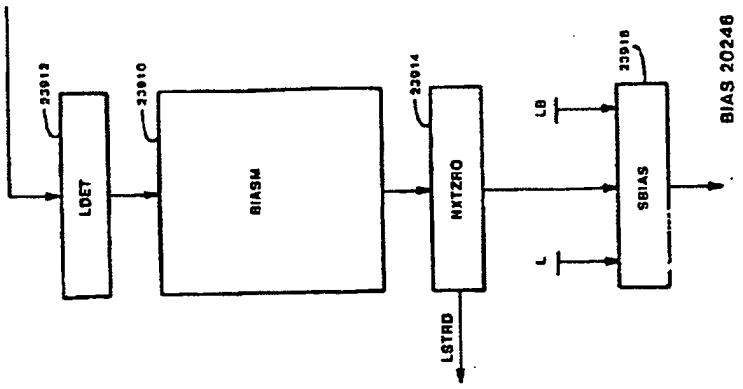


FIG. 239

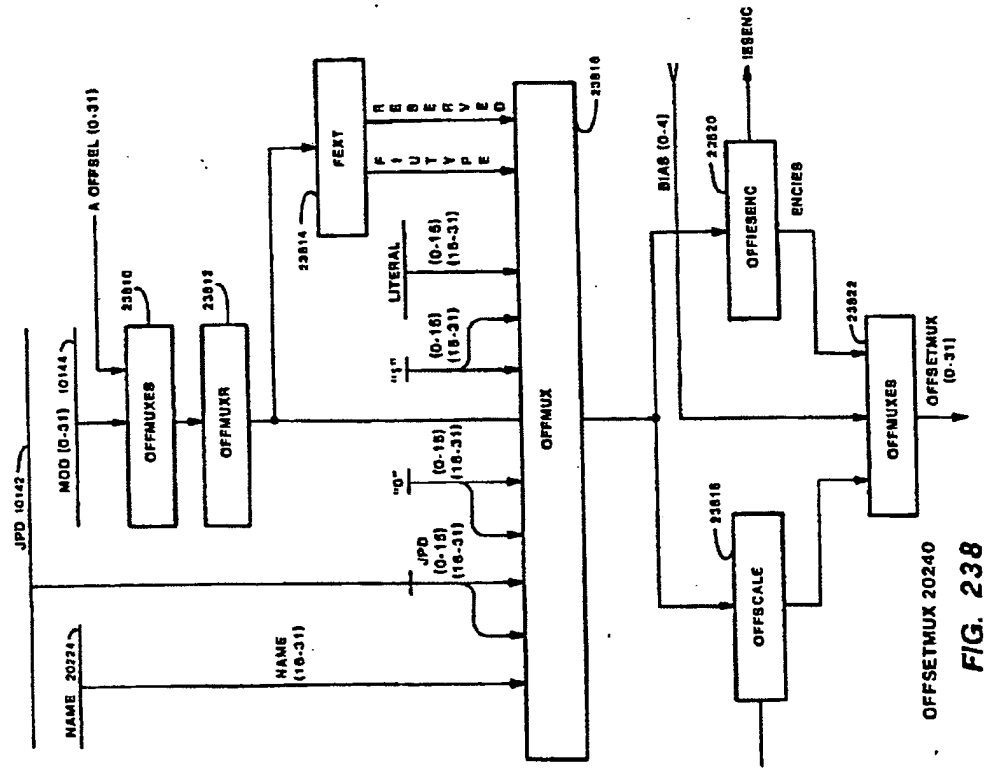
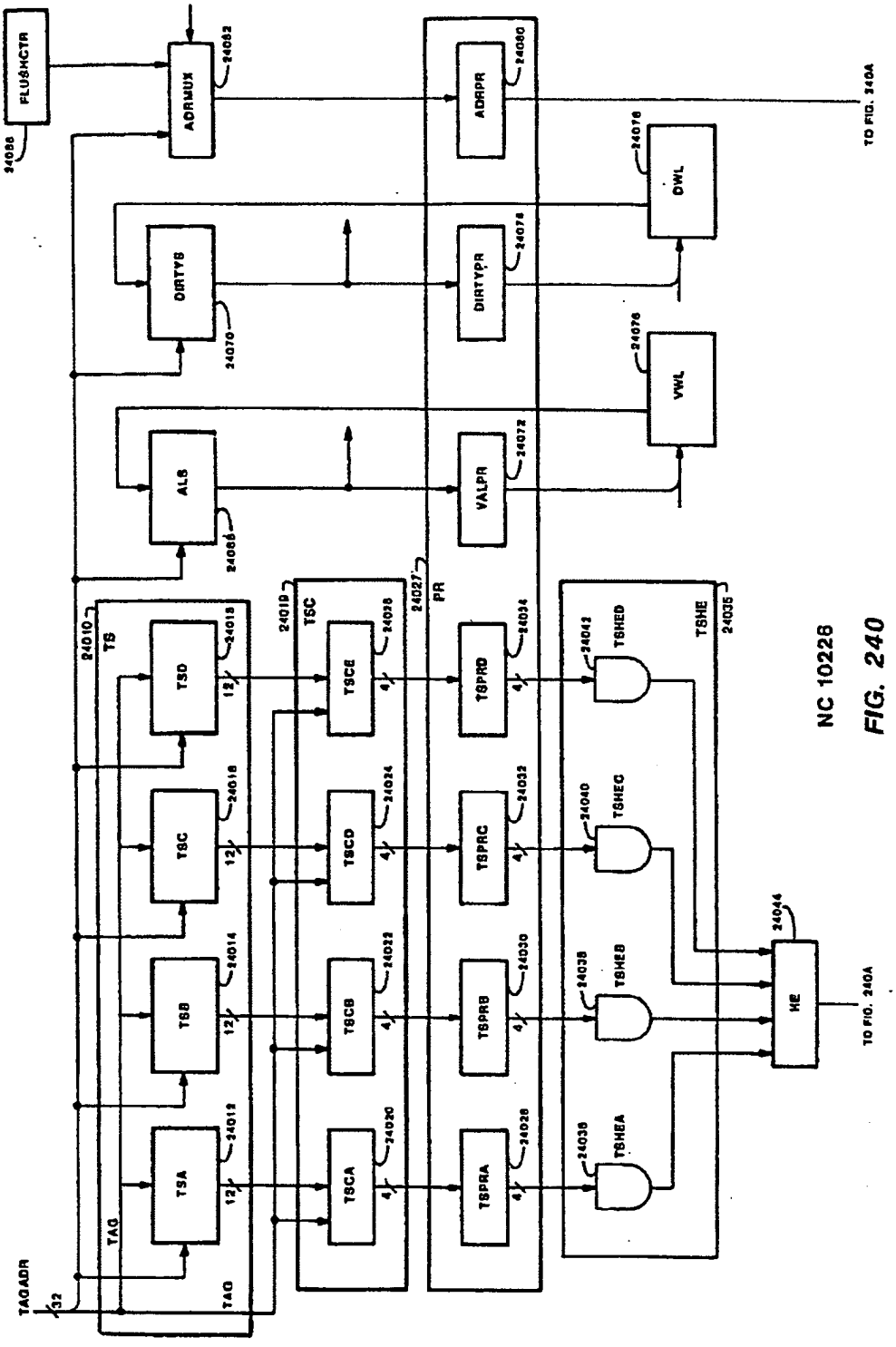
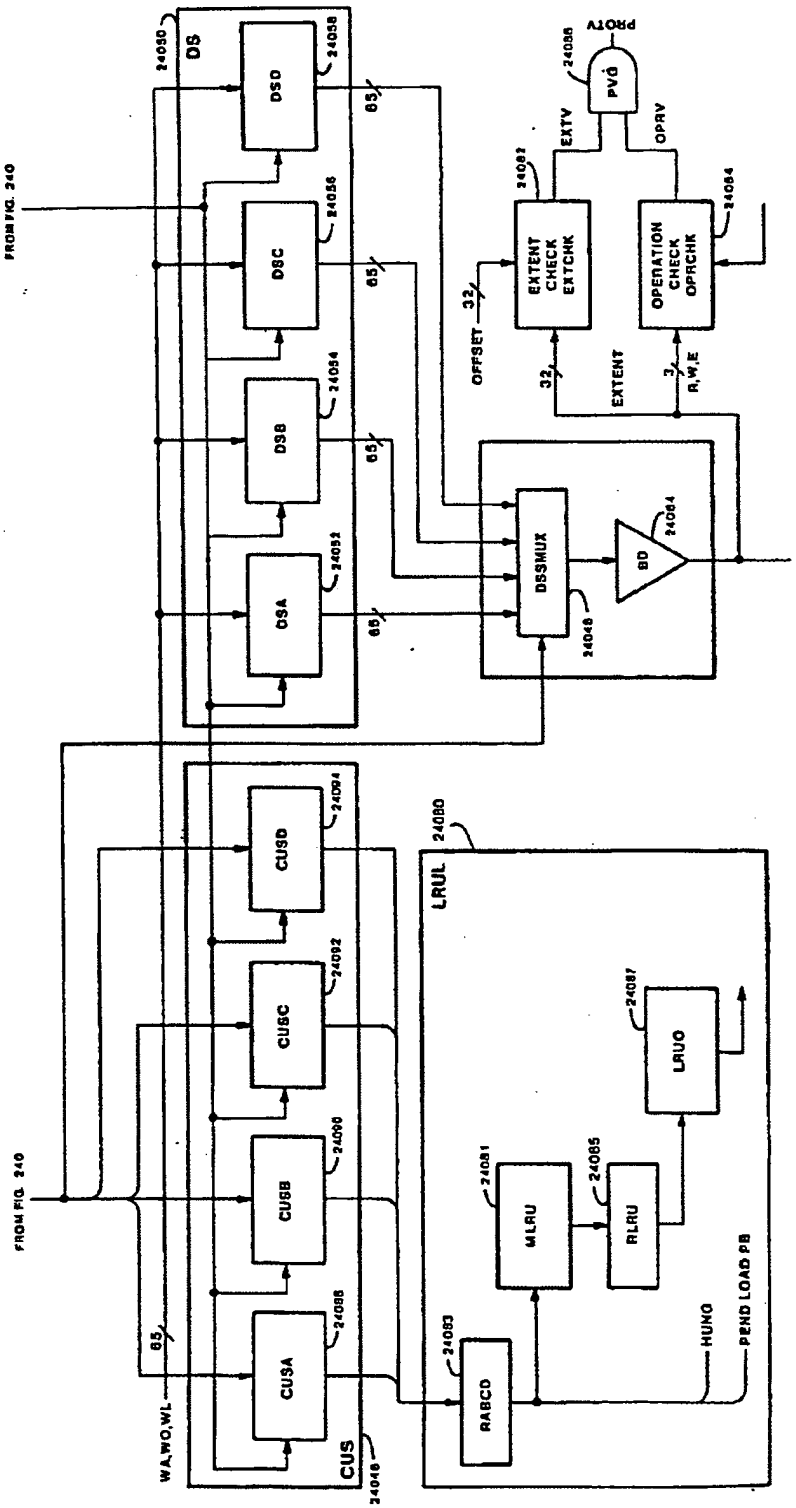


FIG. 238



NC 10228

FIG. 240



NC 10228

FIG. 240A

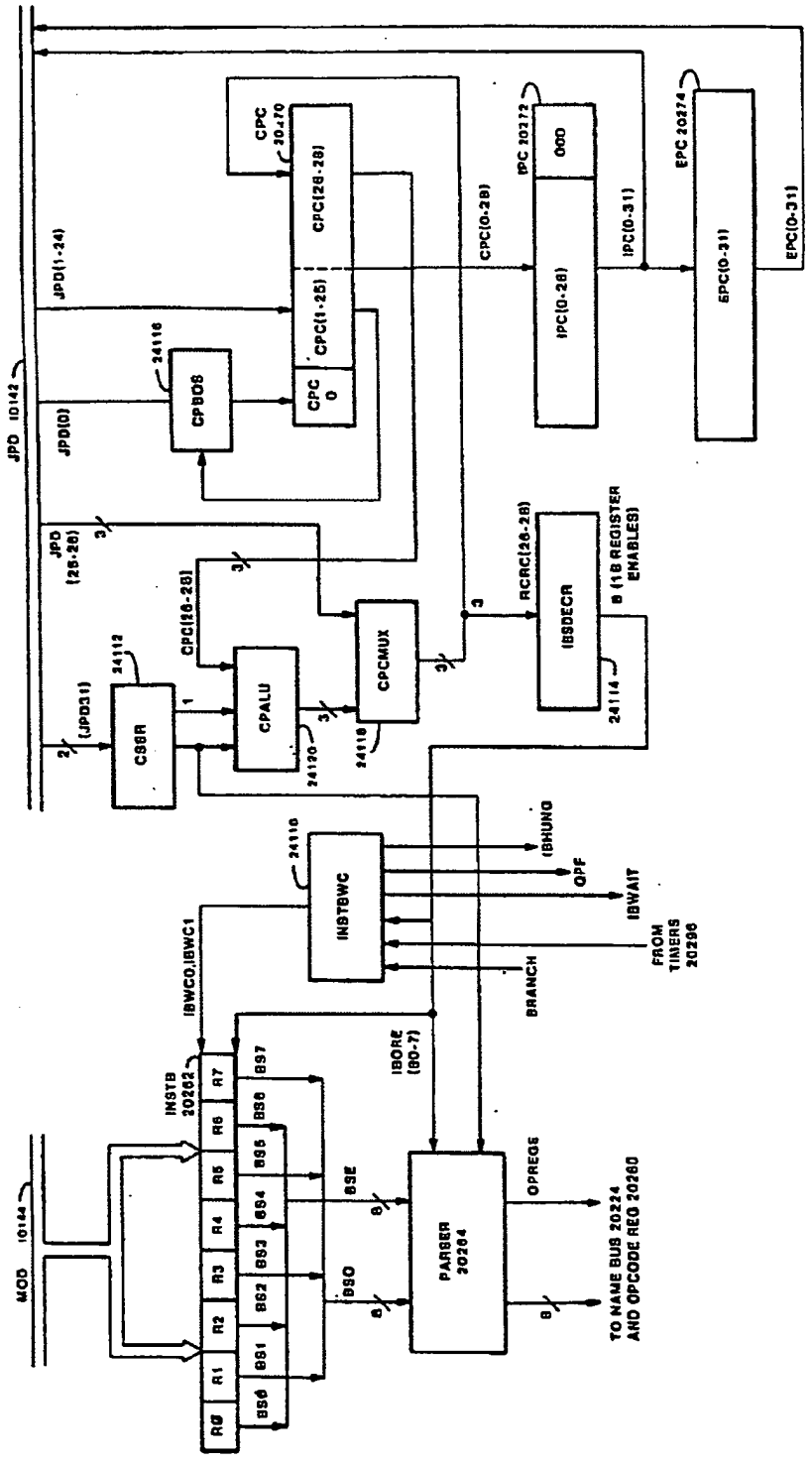


FIG. 241

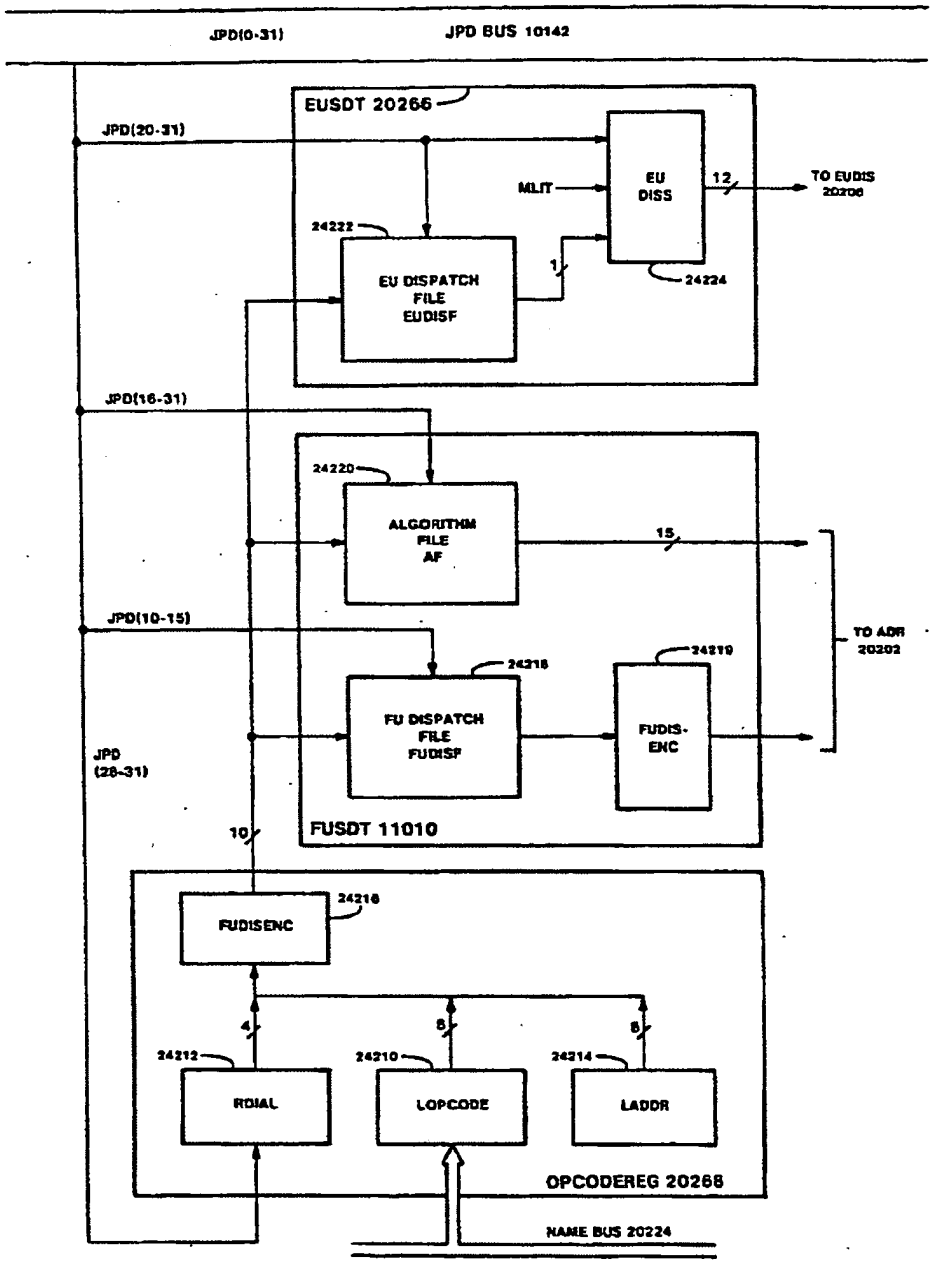


FIG. 242

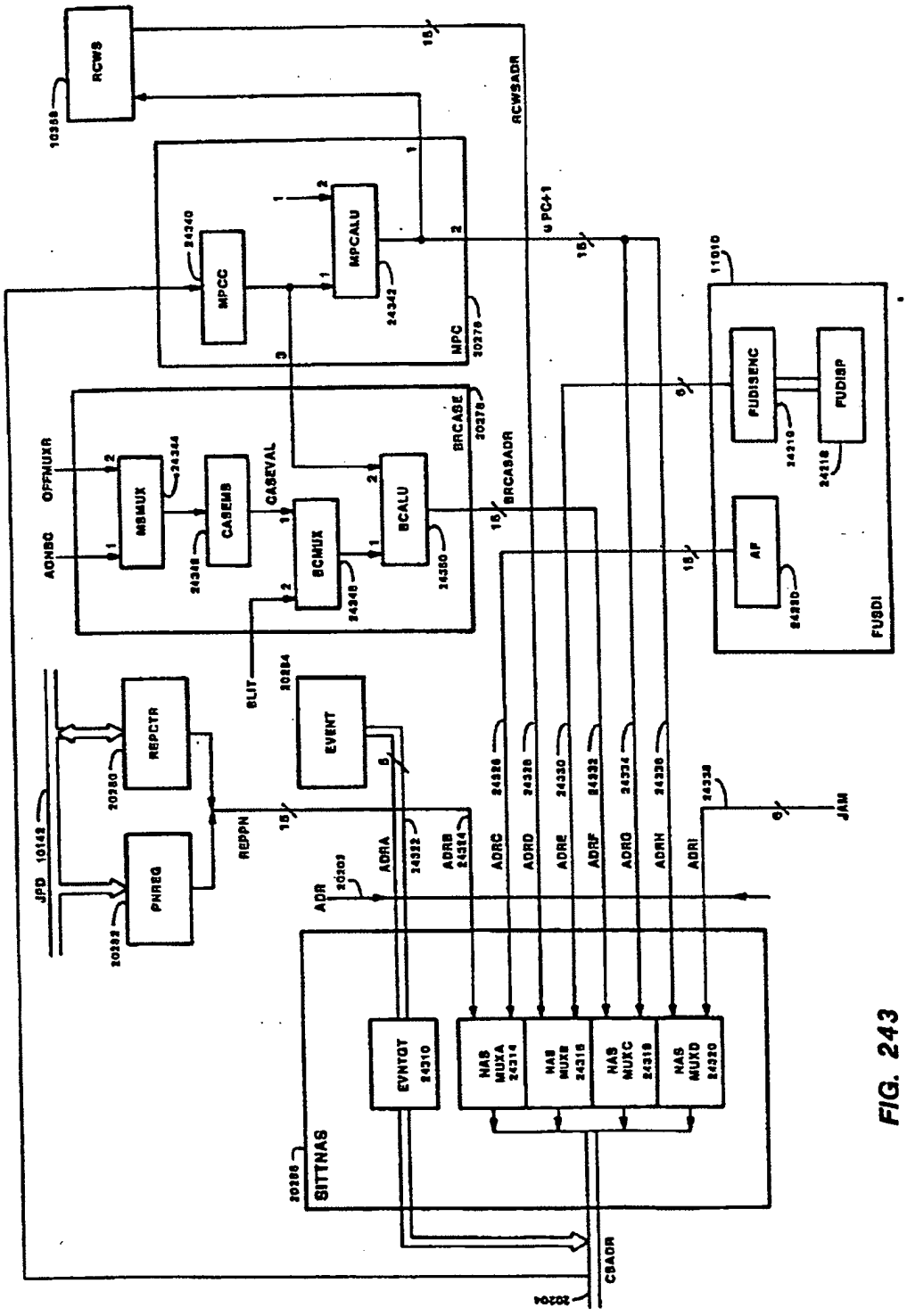


FIG. 243

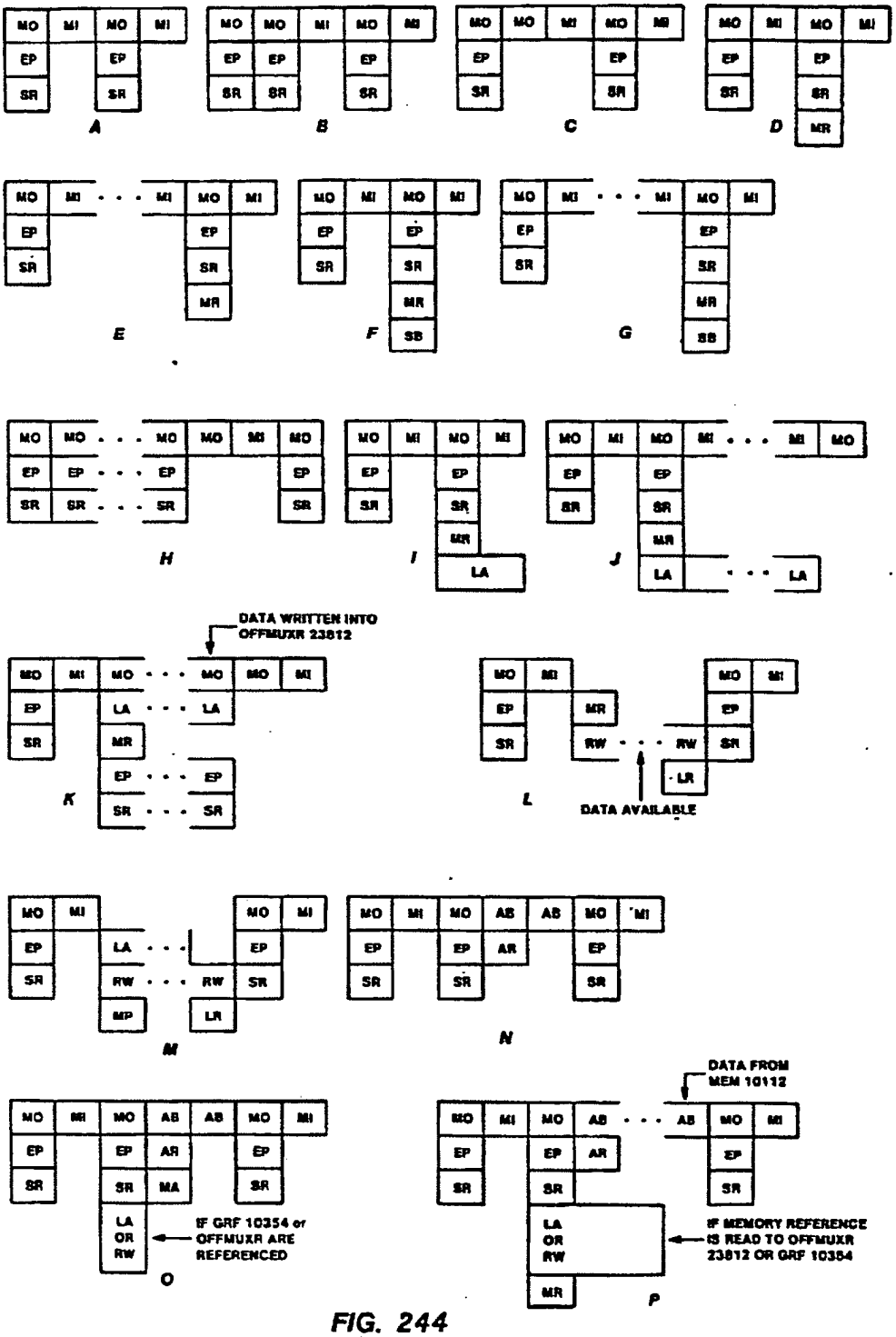


FIG. 244

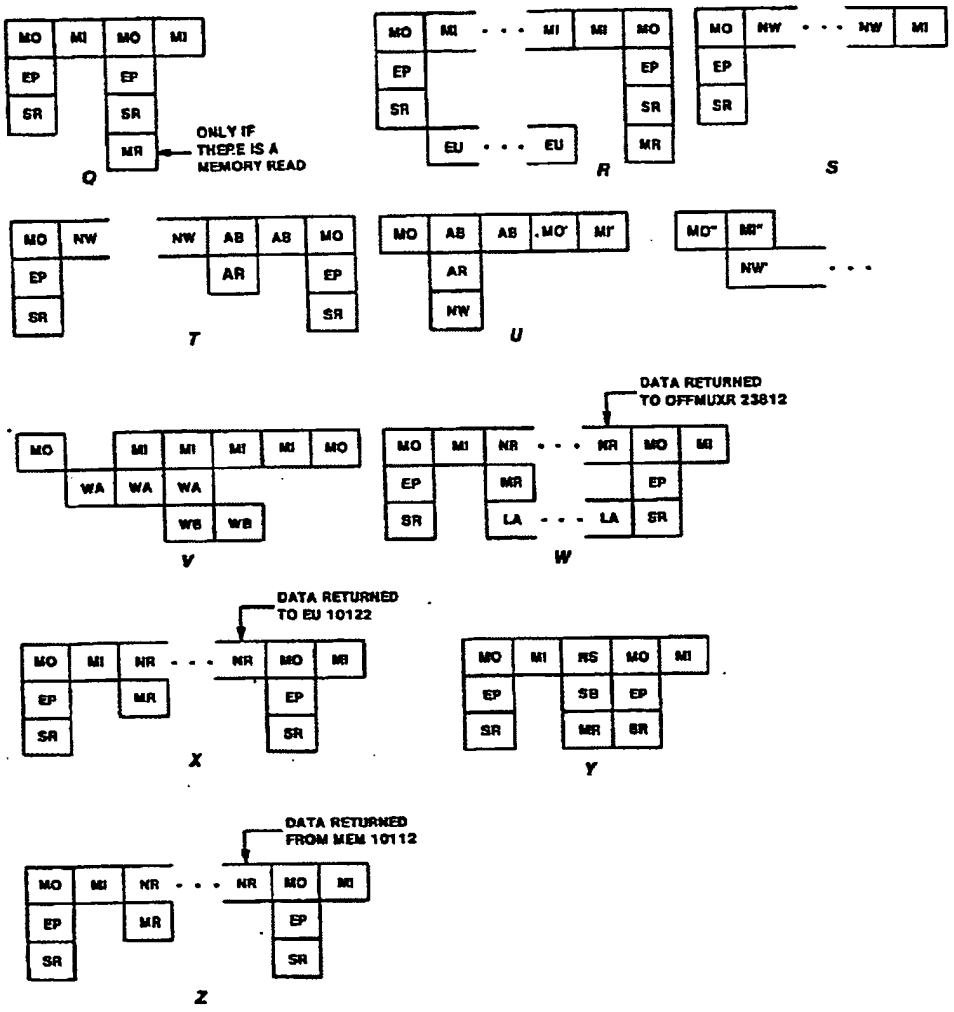


FIG. 244A

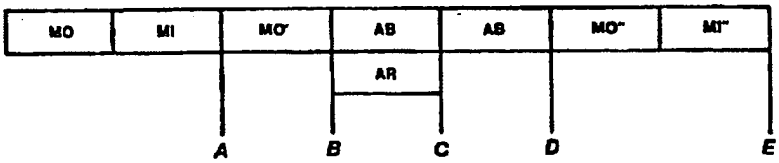


FIG. 245

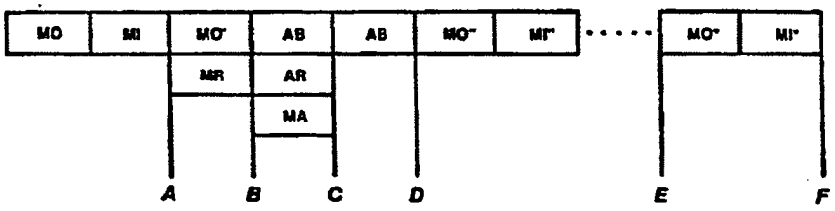


FIG. 246

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

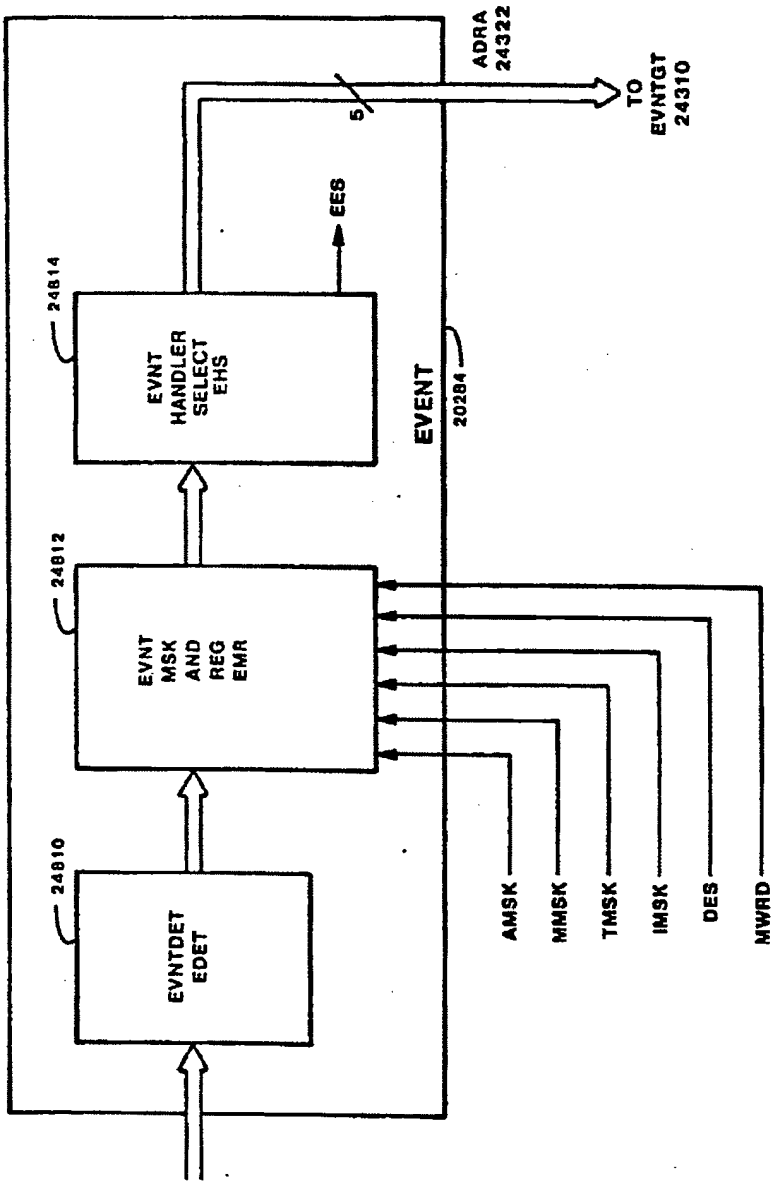


FIG. 248

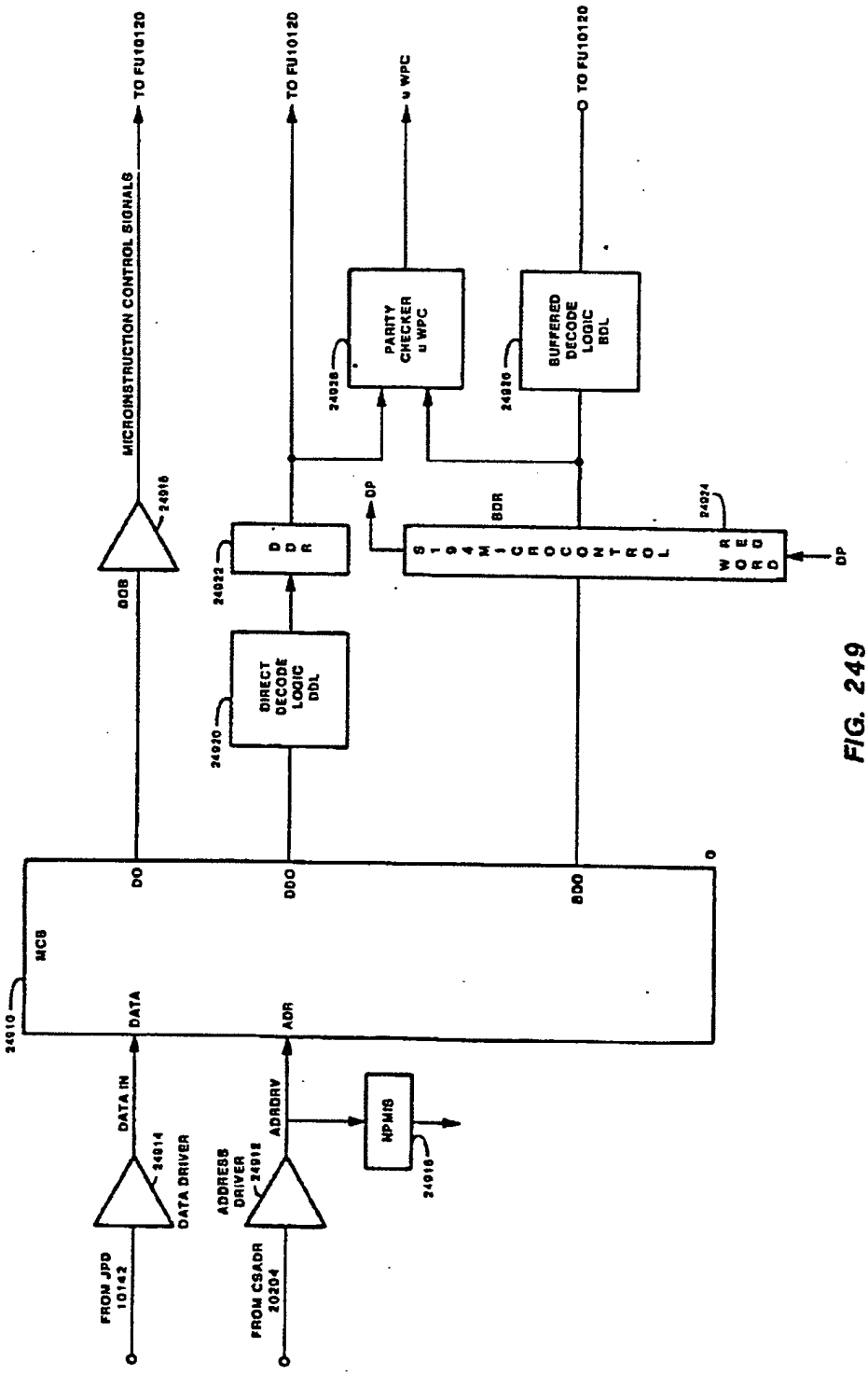


FIG. 249

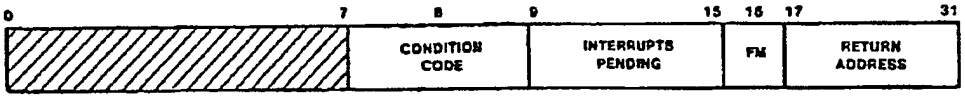


FIG. 251

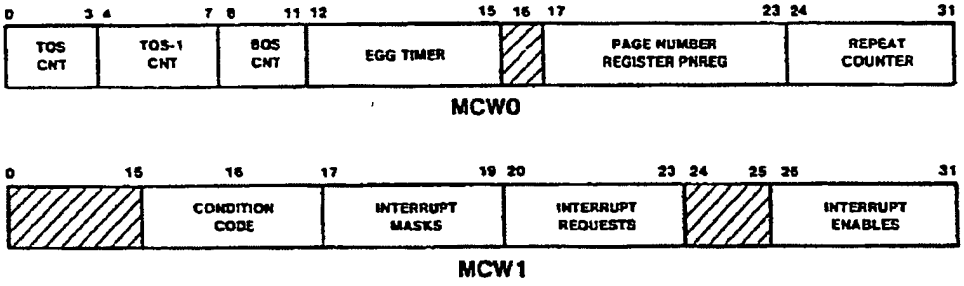


FIG. 252

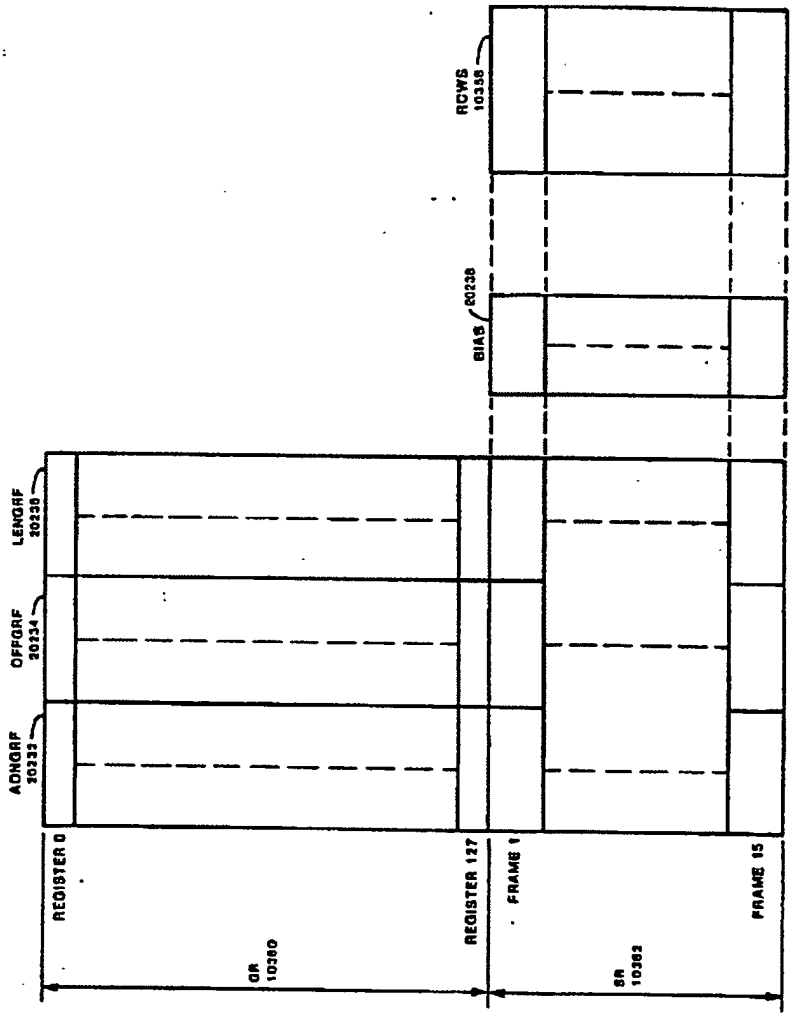


FIG. 253

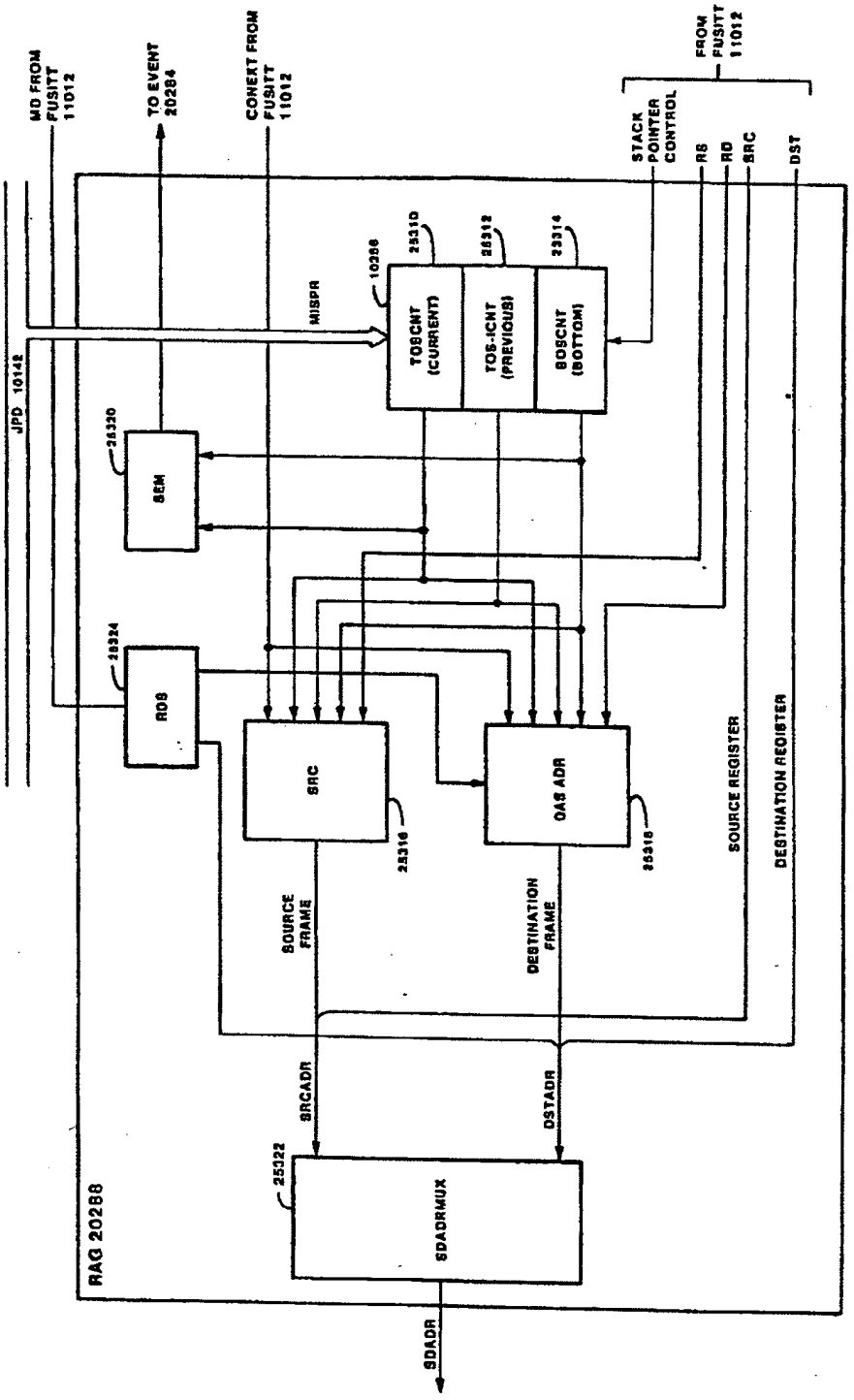


FIG. 253A

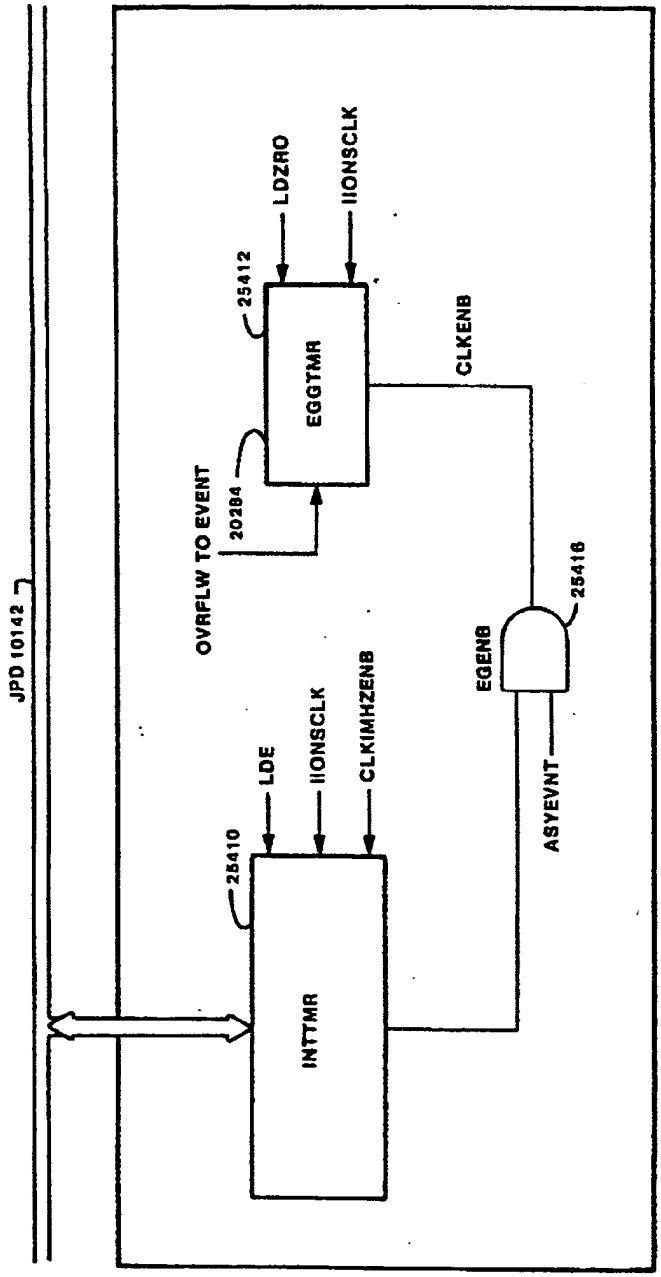


FIG. 254

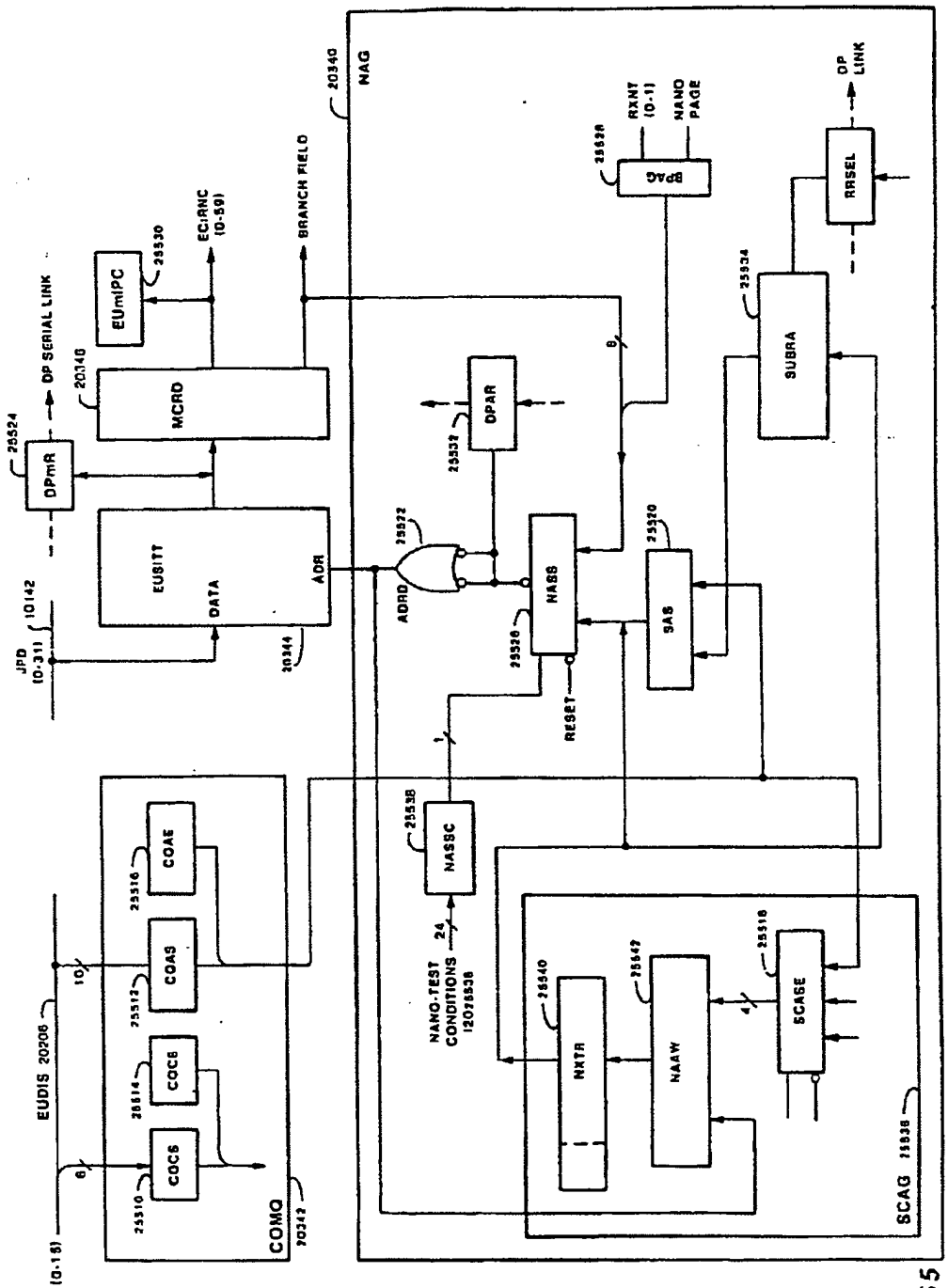


FIG. 255

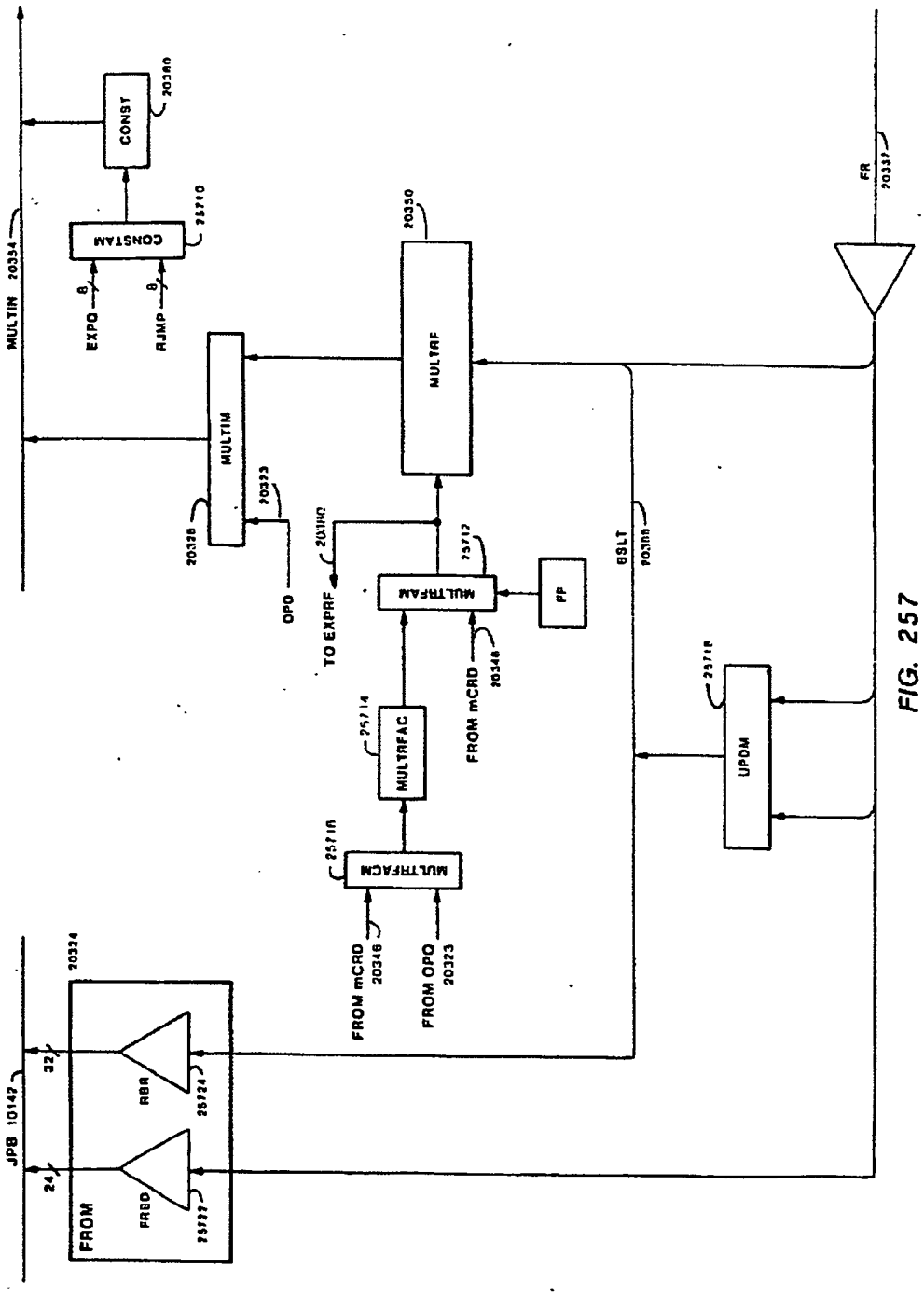


FIG. 257

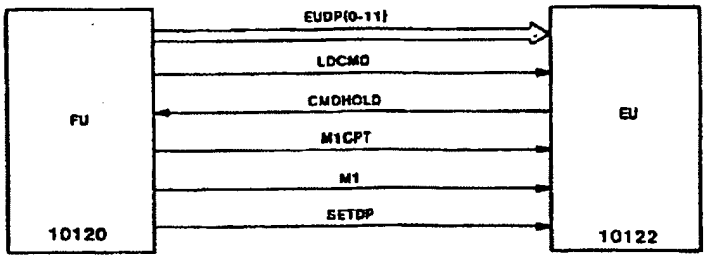


FIG. 260

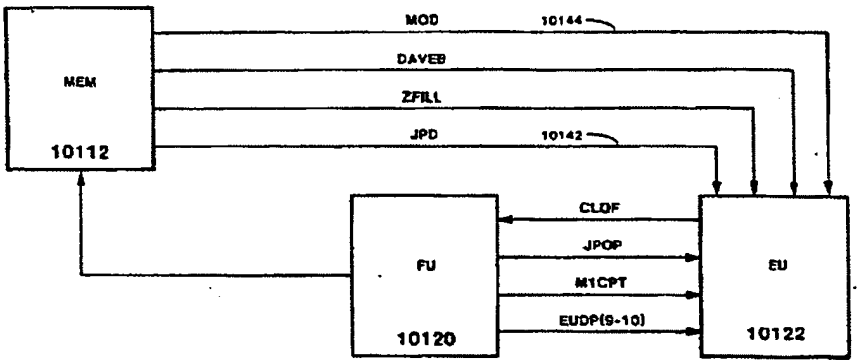


FIG. 261

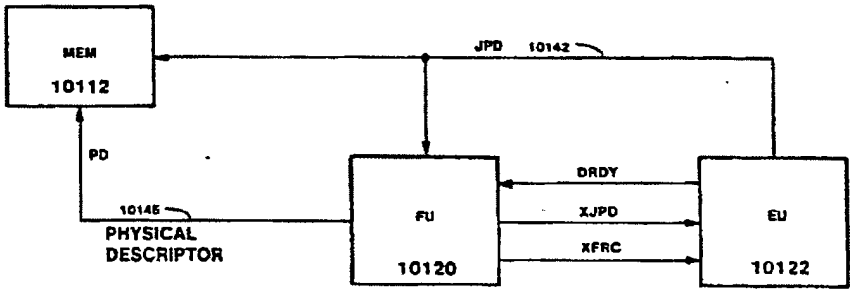


FIG. 262

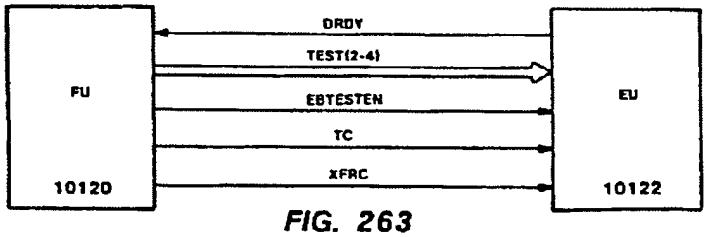


FIG. 263

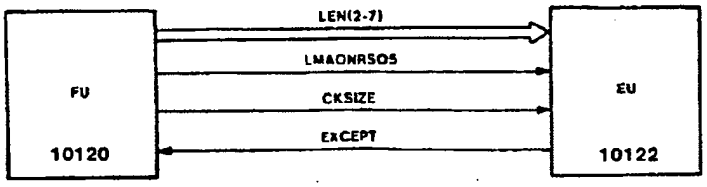


FIG. 264

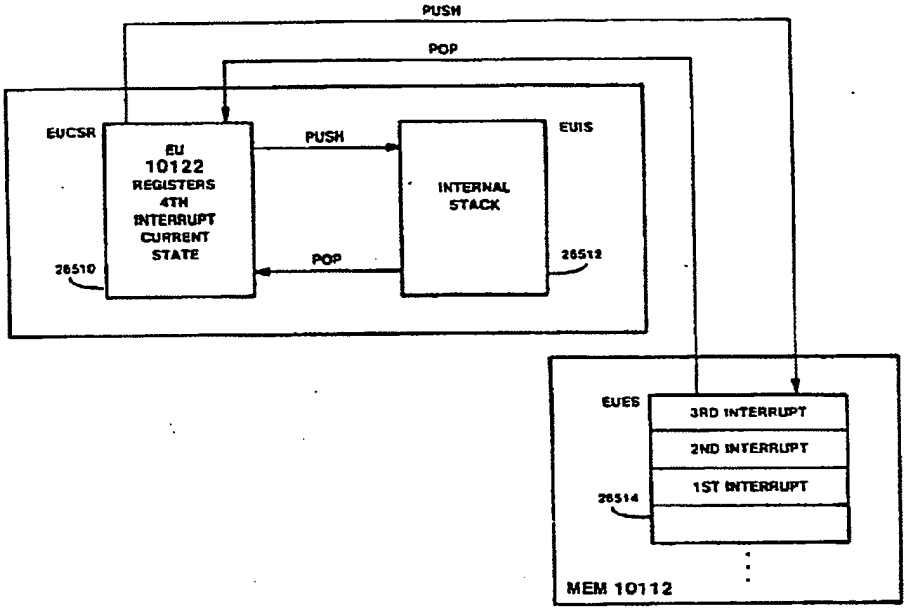


FIG. 265

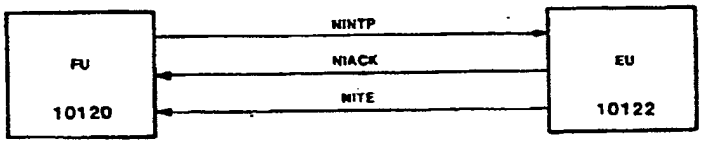


FIG. 266

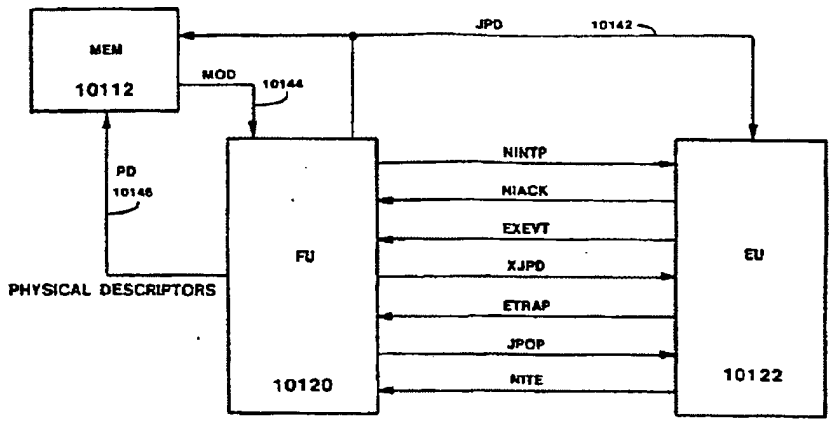


FIG. 267

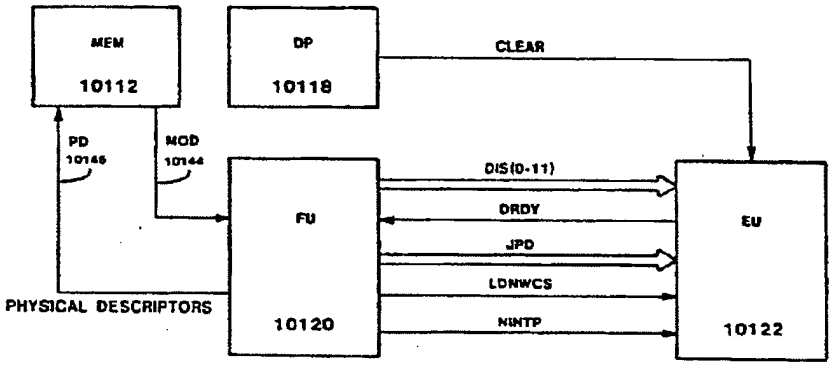


FIG. 268

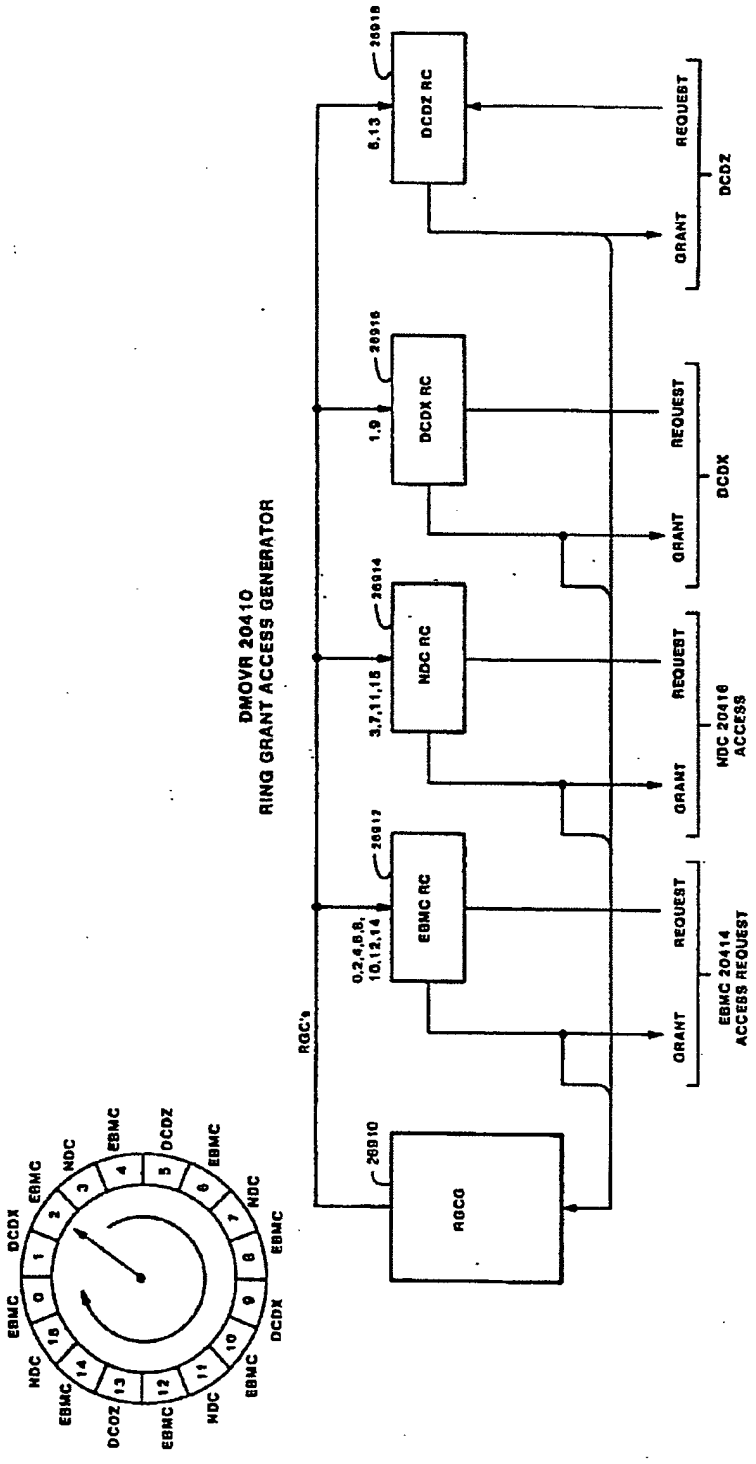


FIG 269

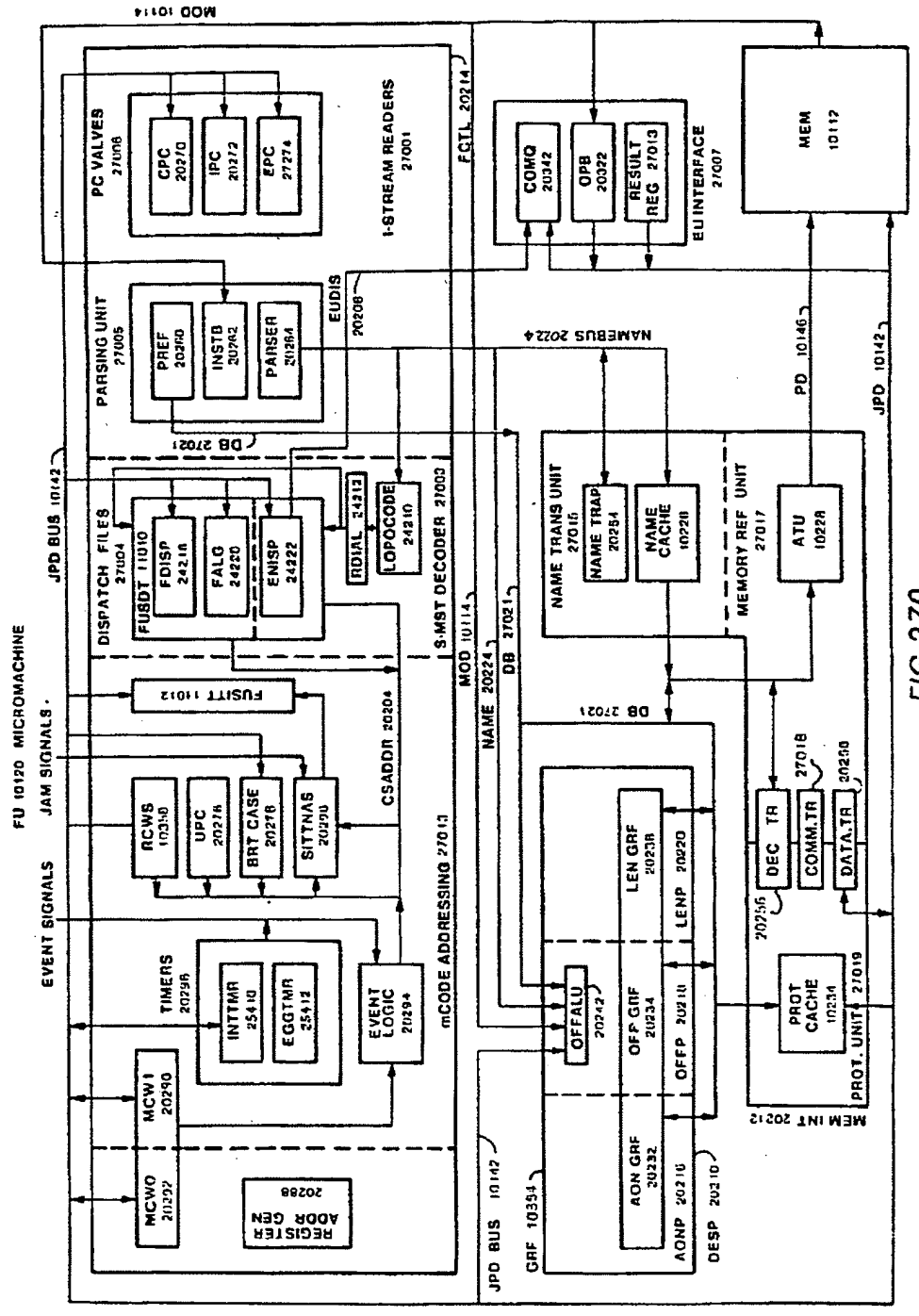


FIG 270

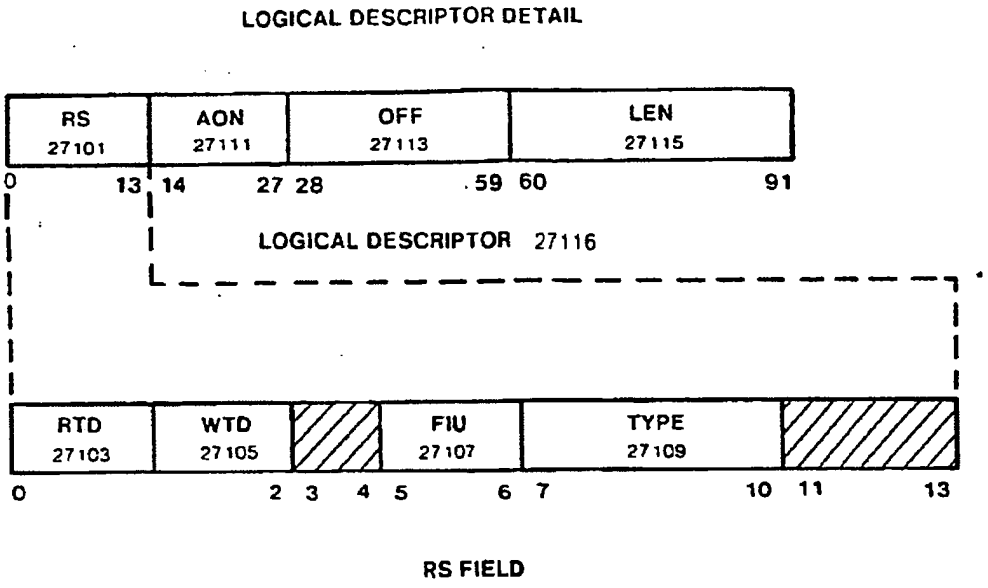


FIG. 271

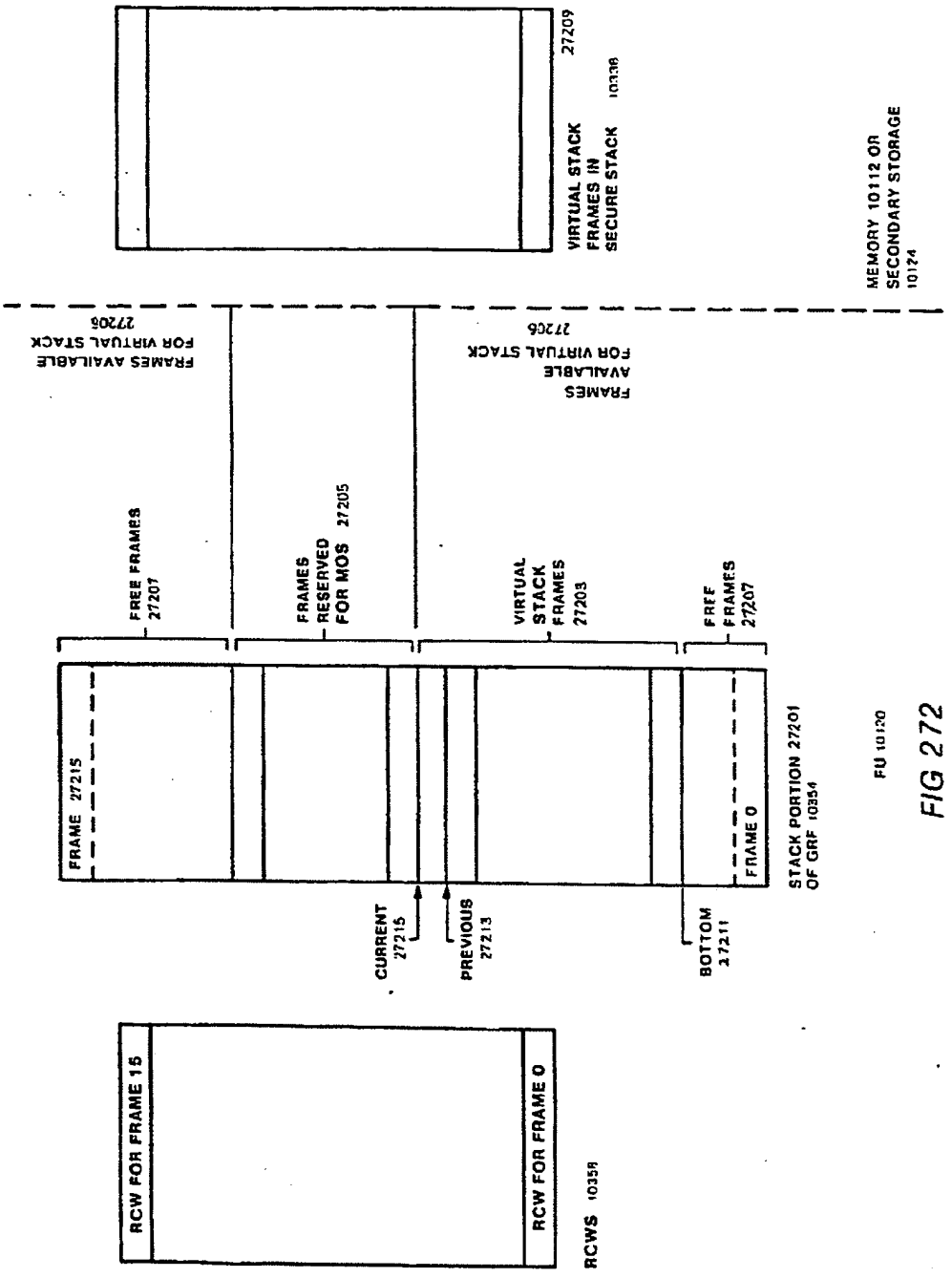


FIG 272

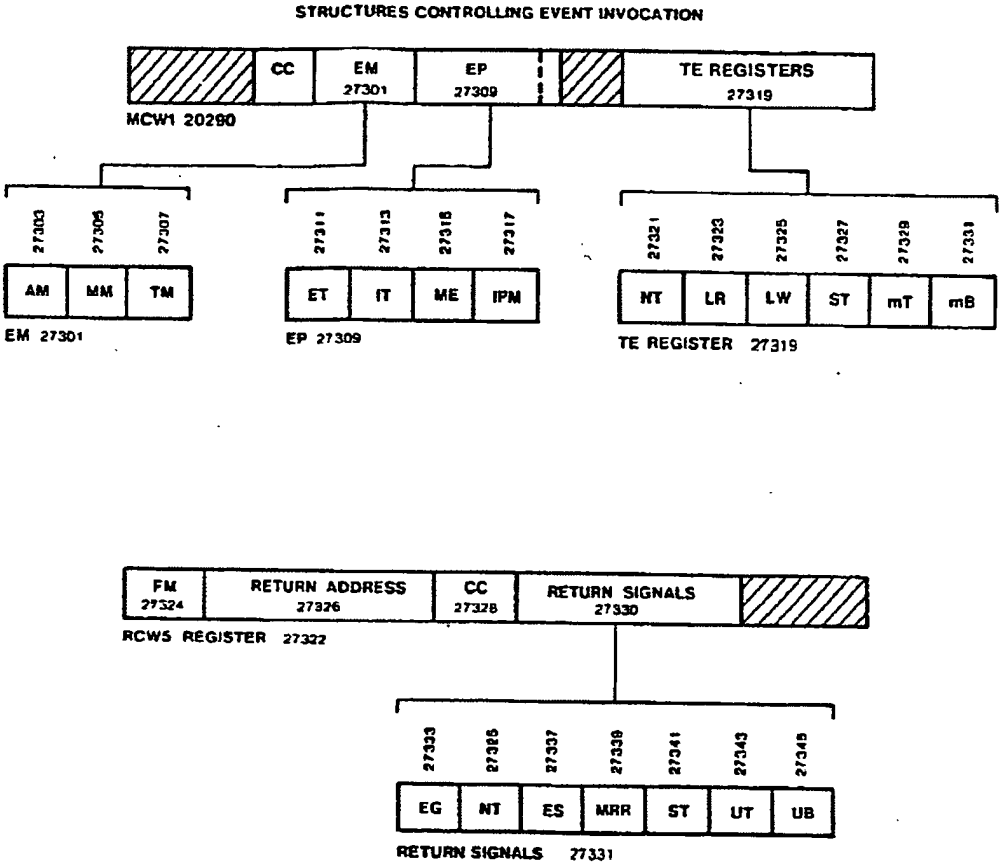


FIG. 273

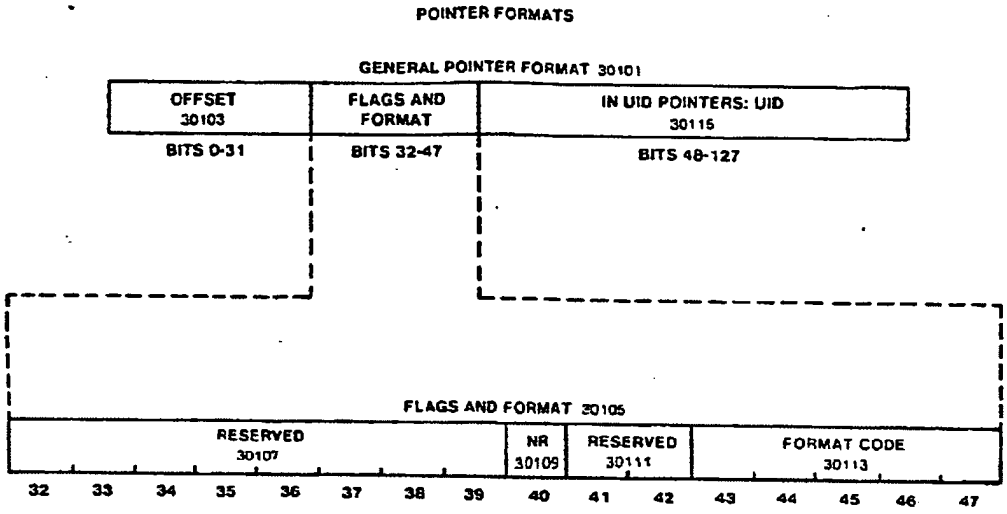


FIG. 301

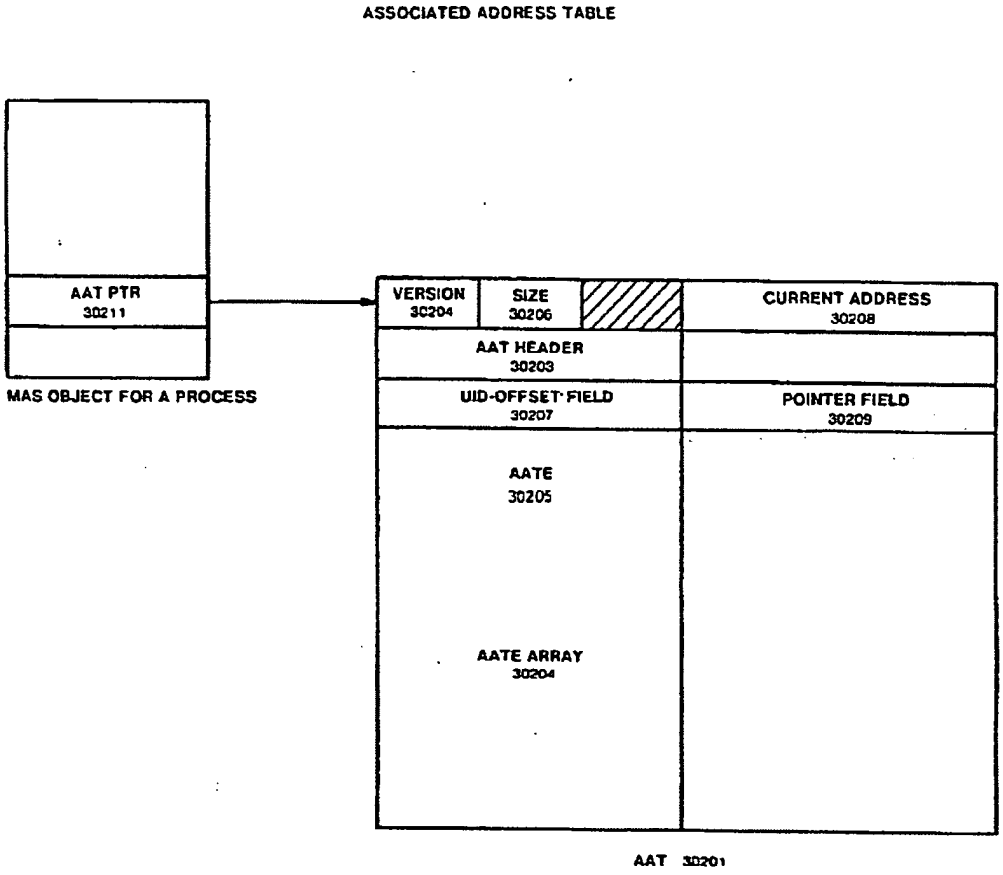


FIG. 302

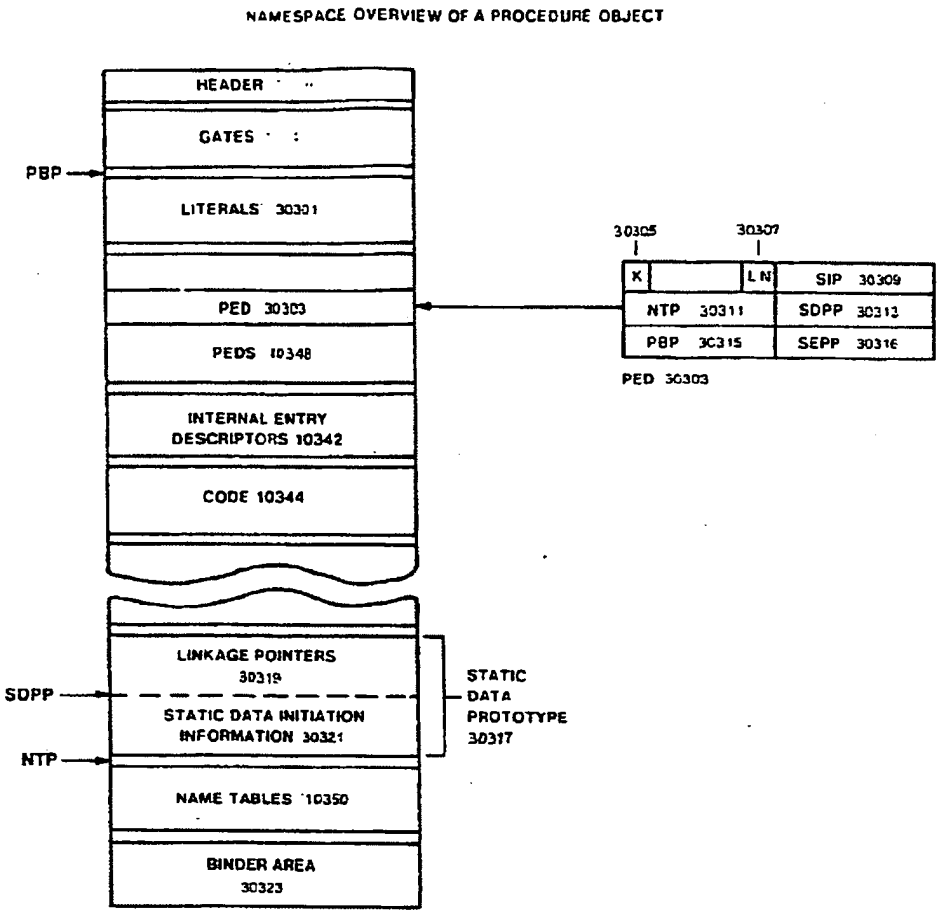


FIG. 303

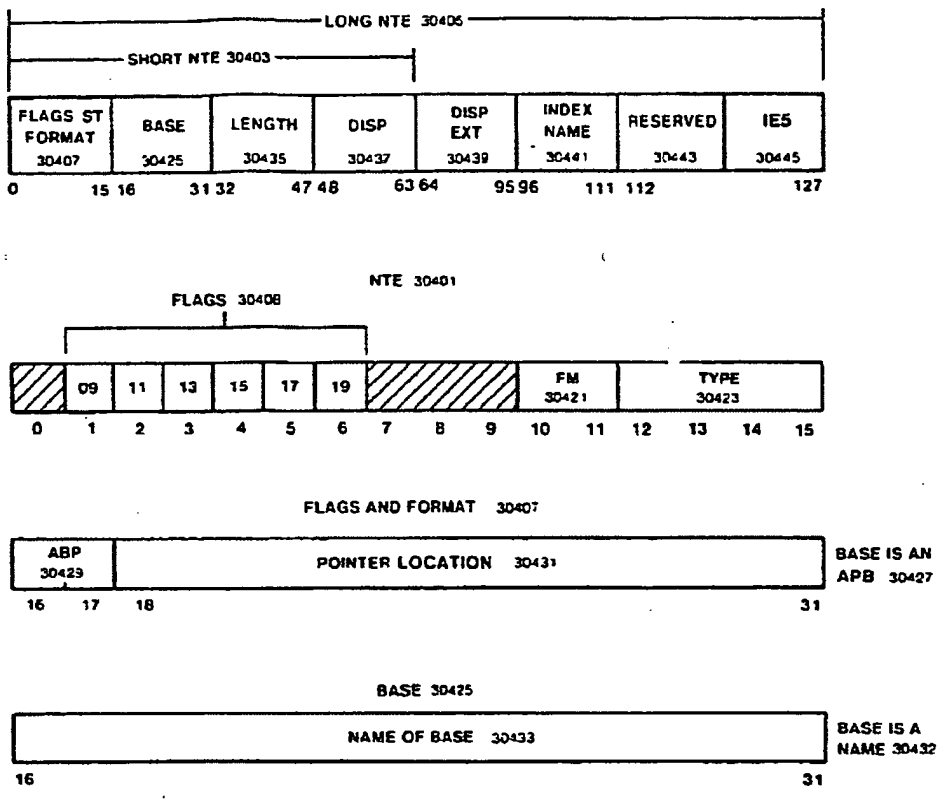


FIG. 304

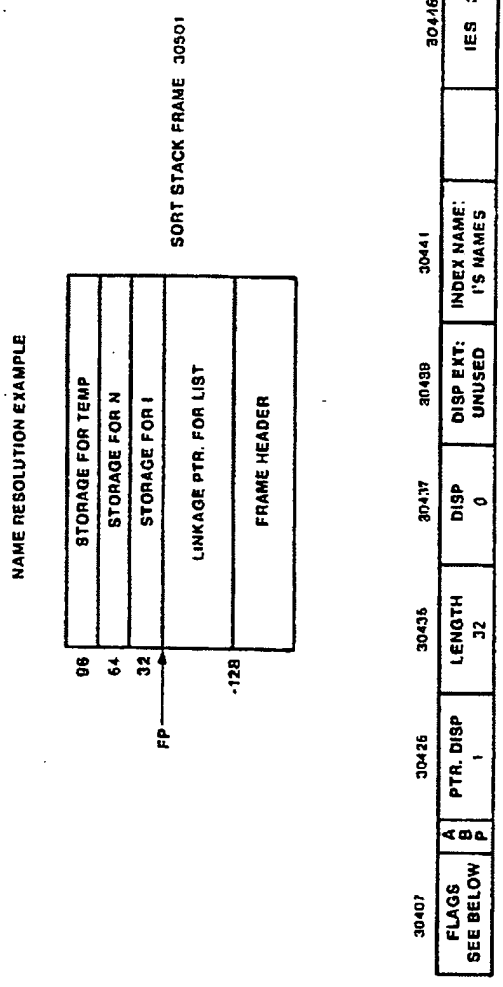


FIG. 305

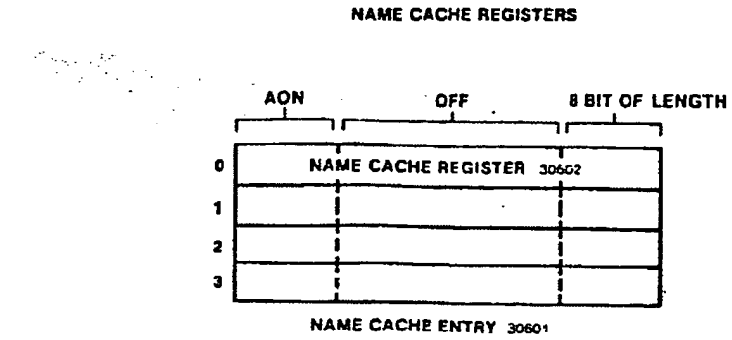


FIG. 306

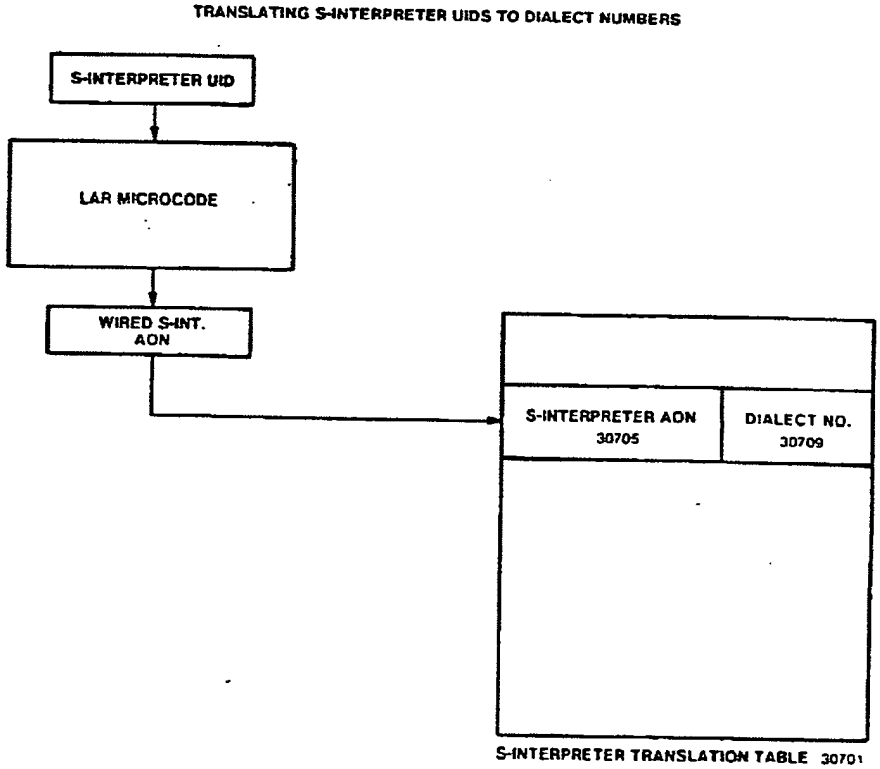


FIG. 307

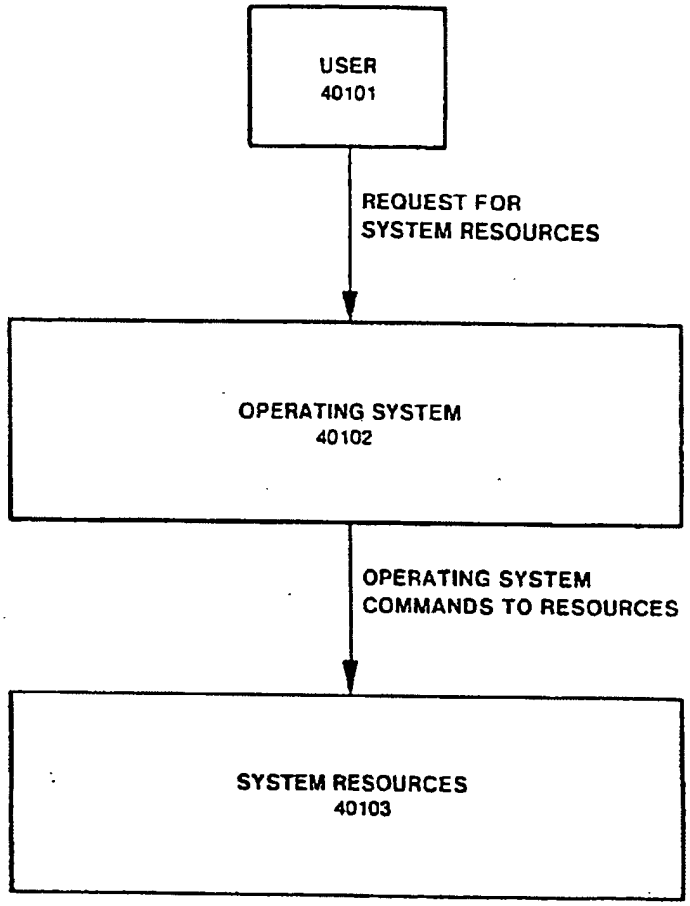


FIG 401

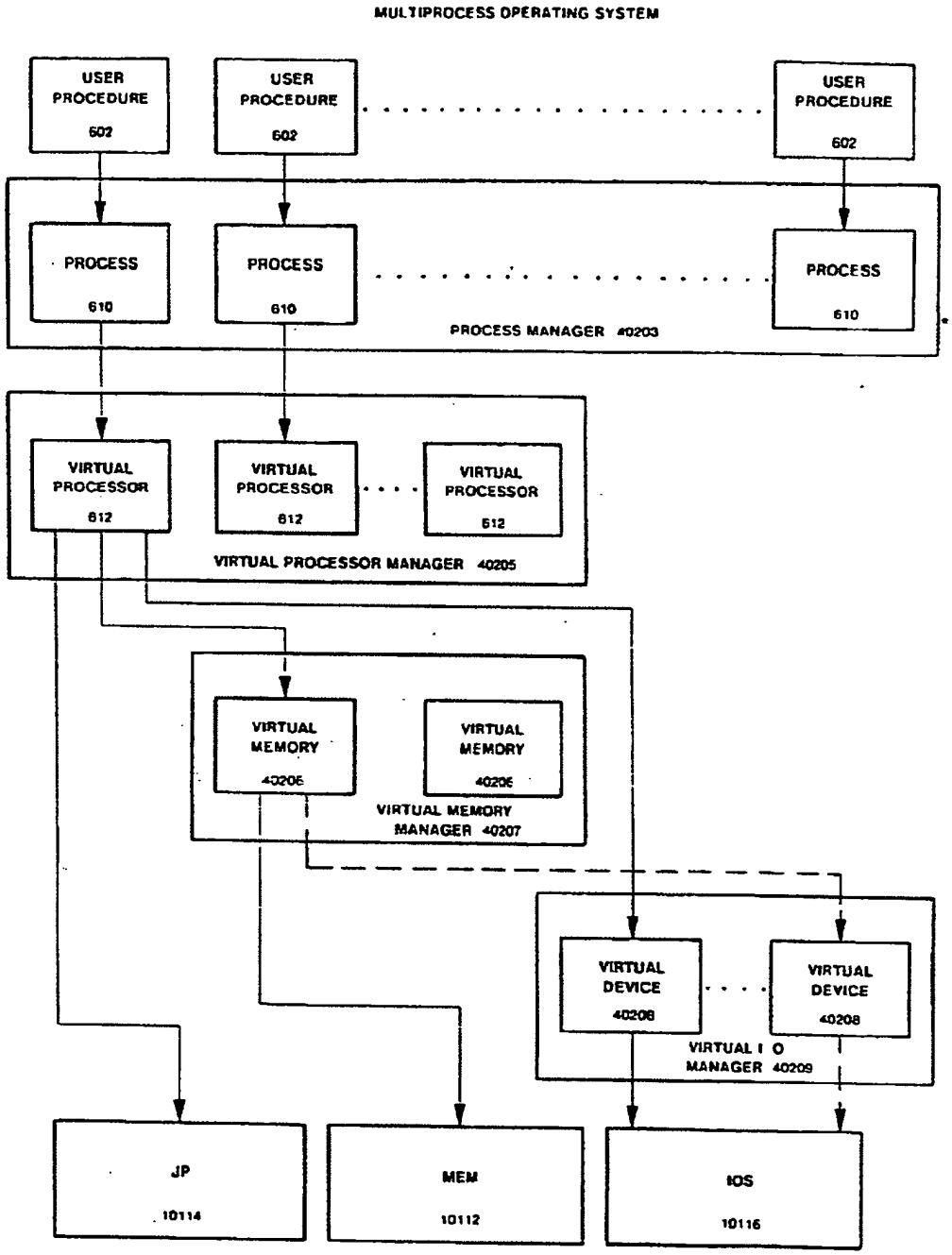


FIG 402

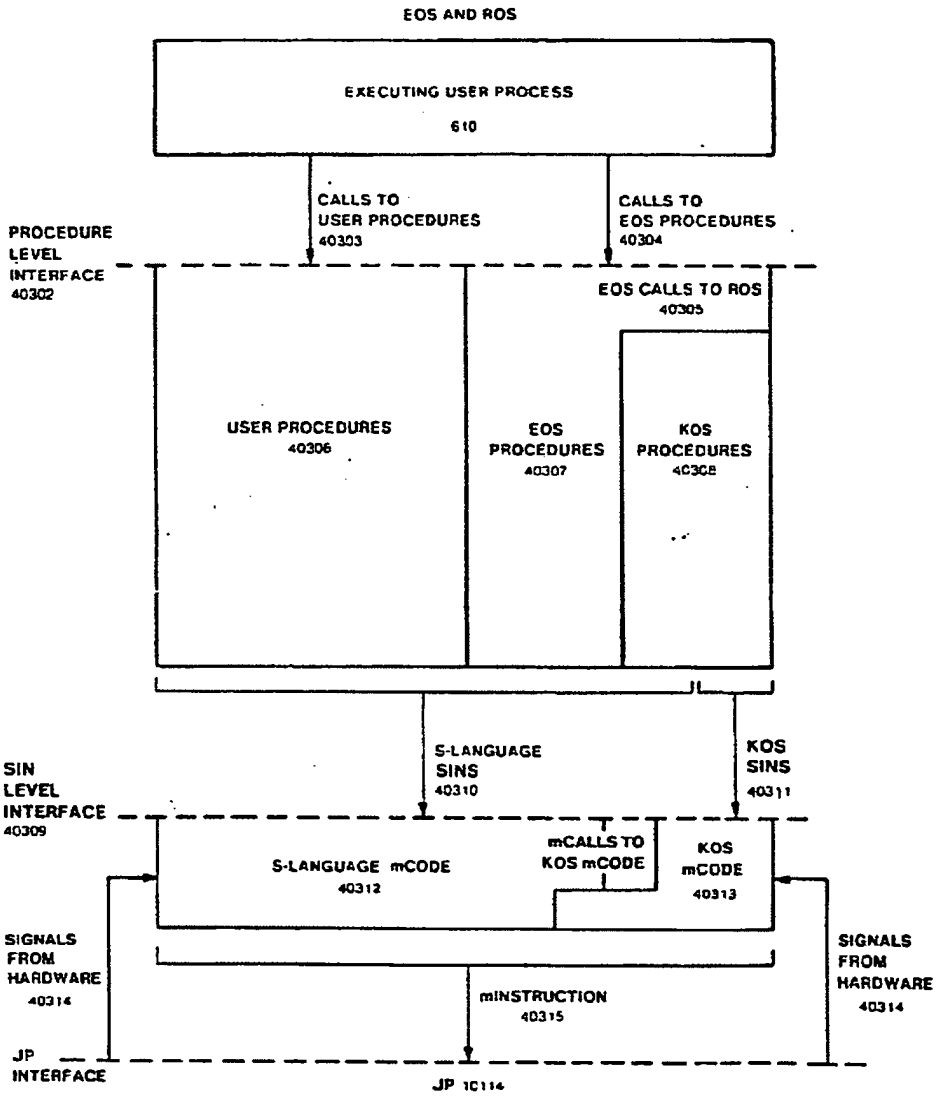


FIG. 403

EOS VIEW OF OBJECTS

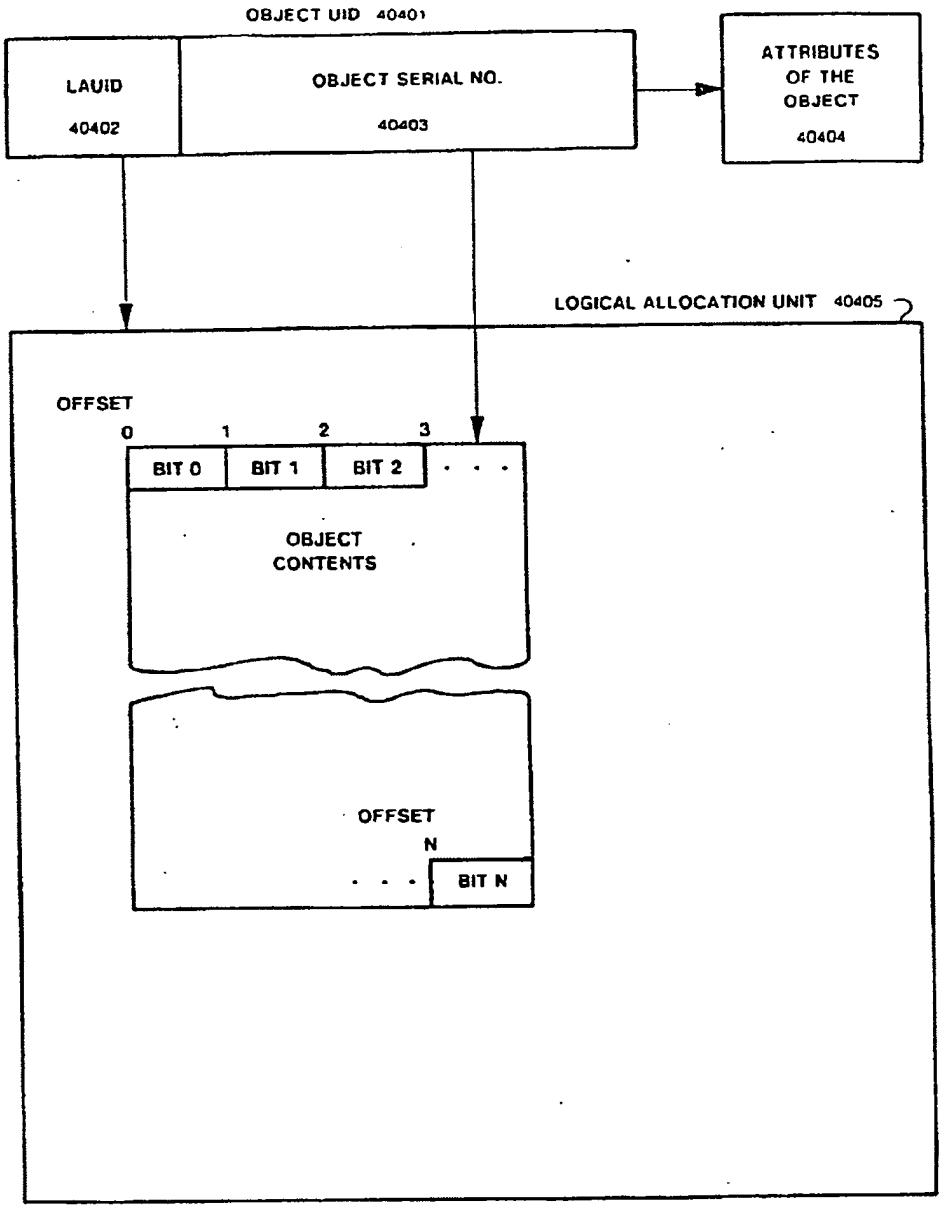


FIG 404

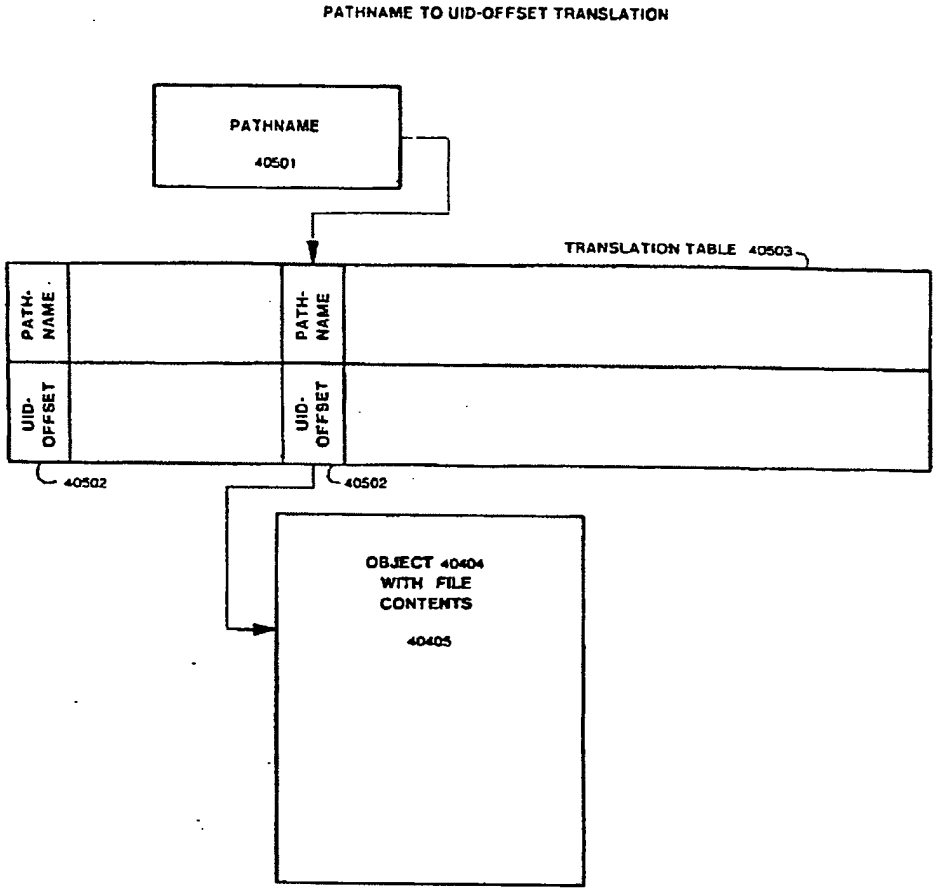


FIG 405

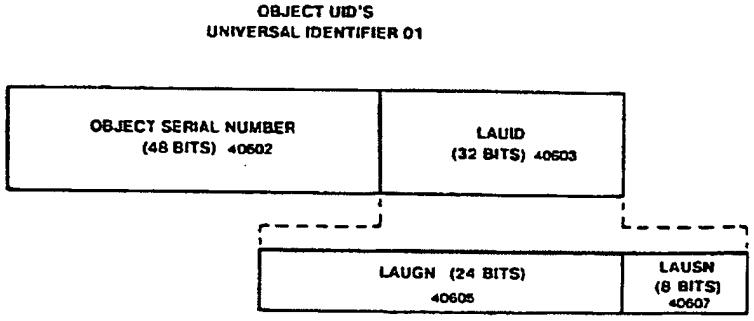


FIG 406

ATU, MHT, AND MEMORY

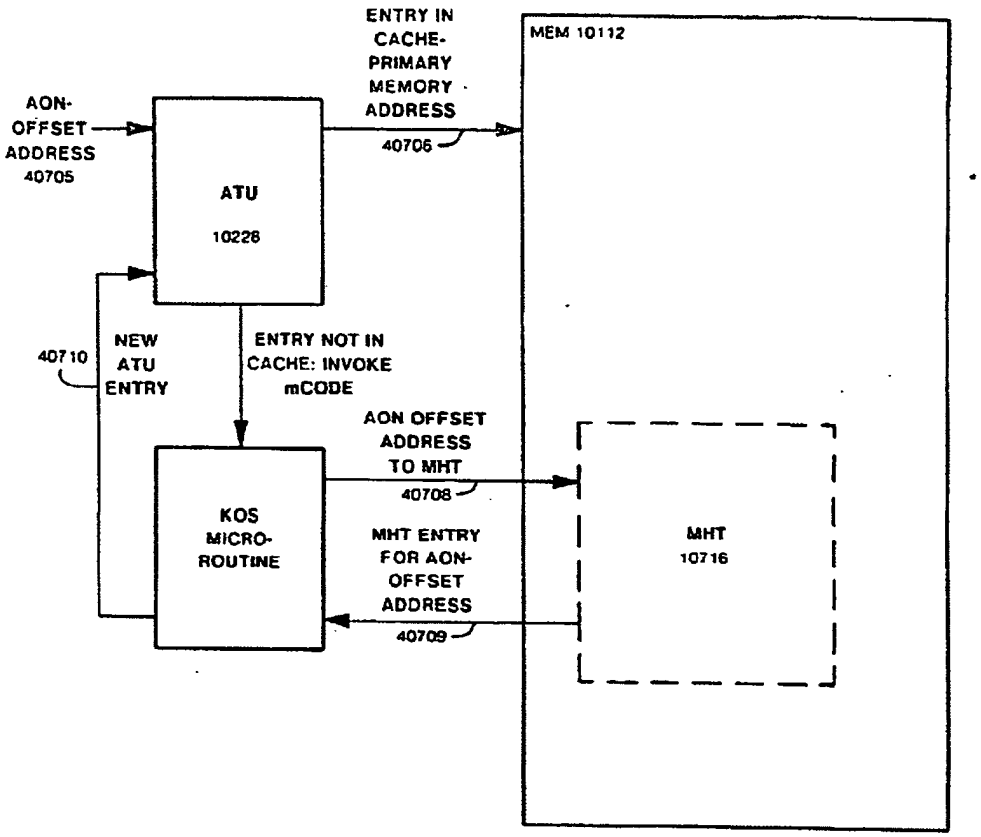


FIG 407

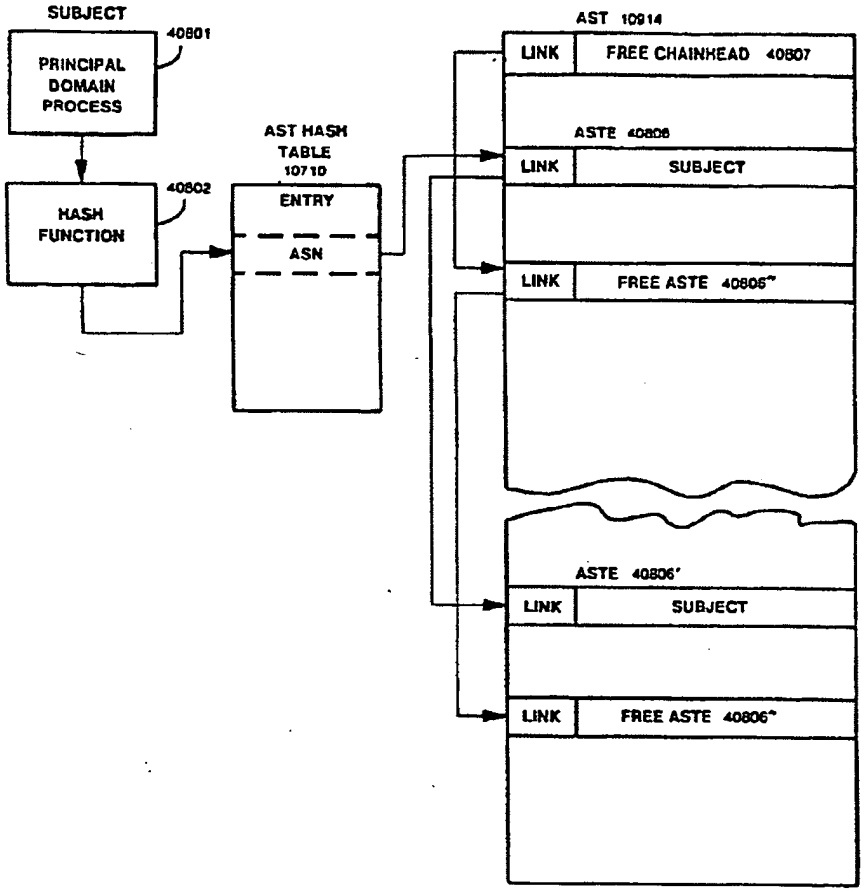


FIG 408

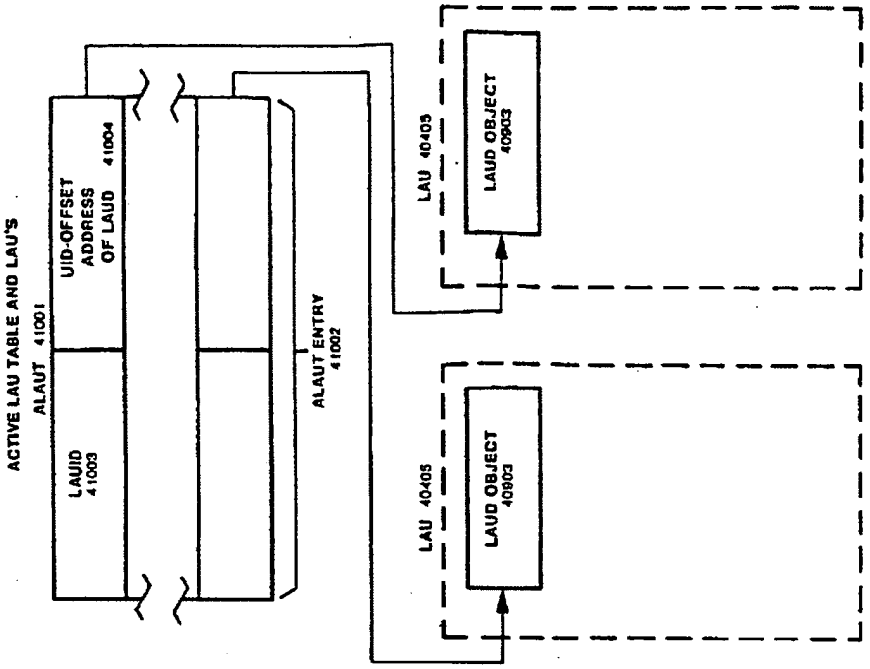


FIG 410

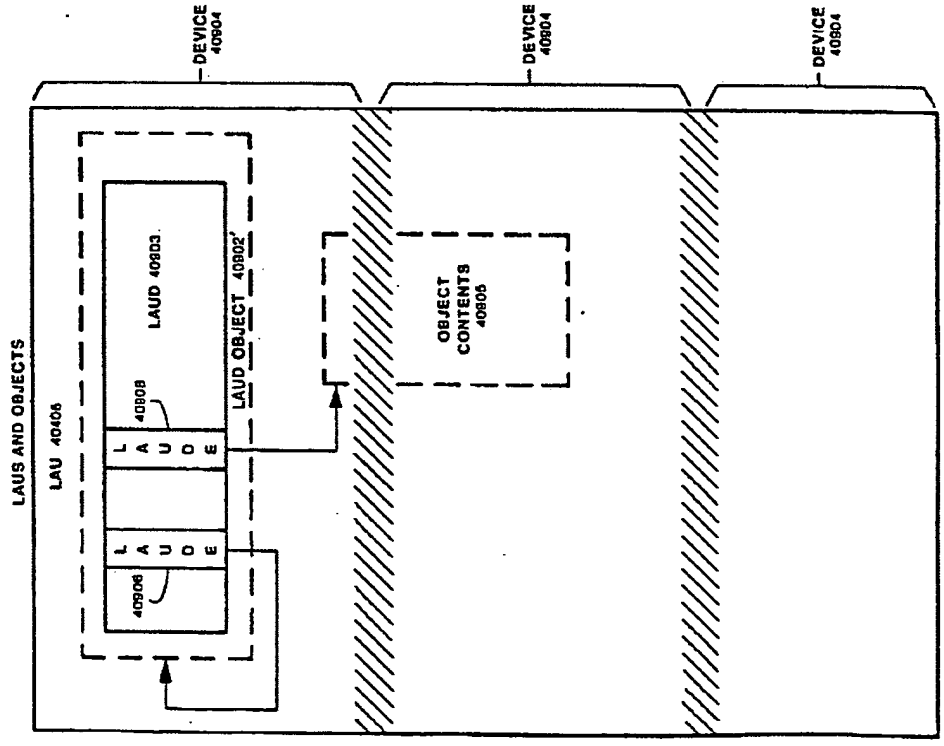


FIG 409

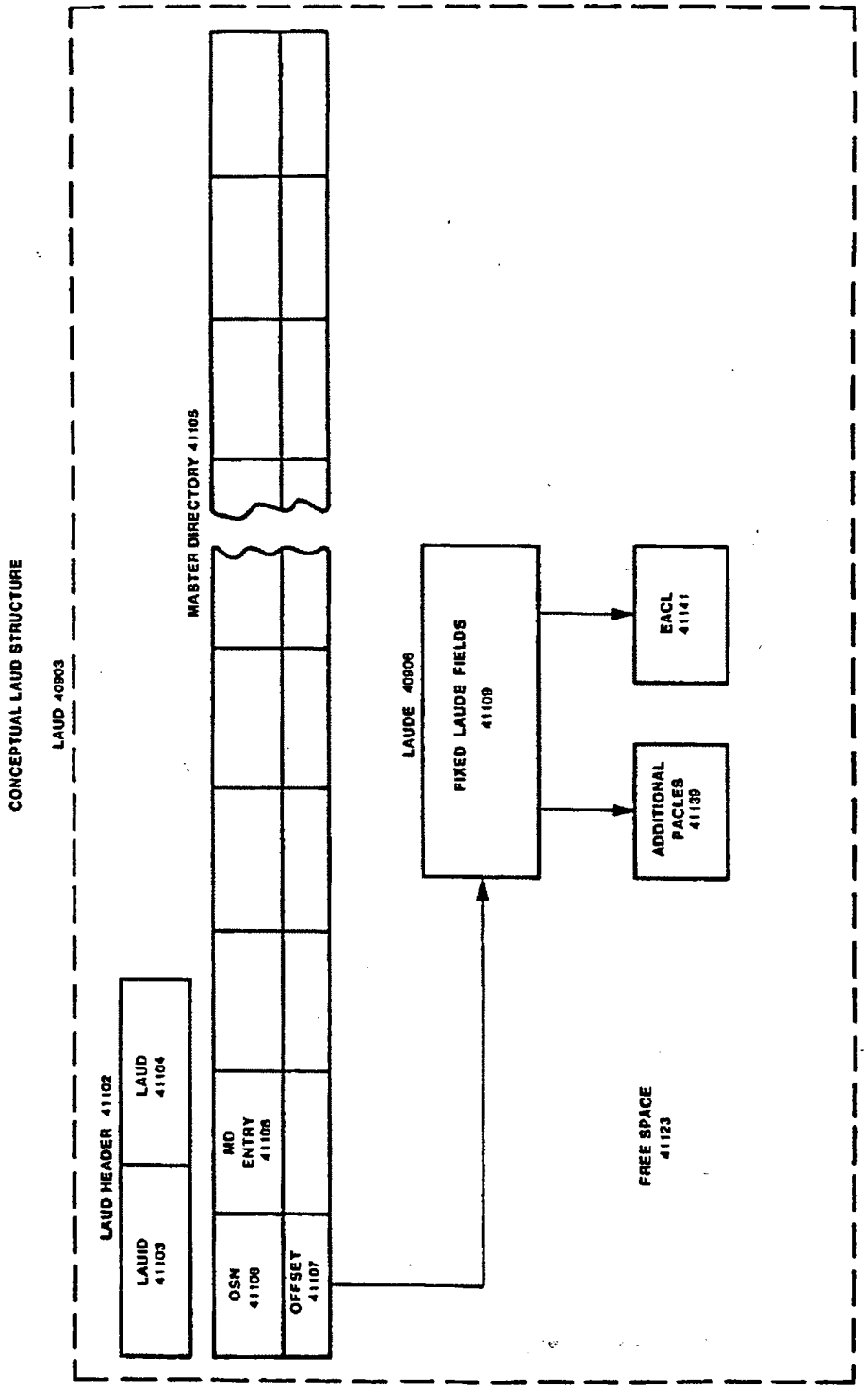


FIG 411

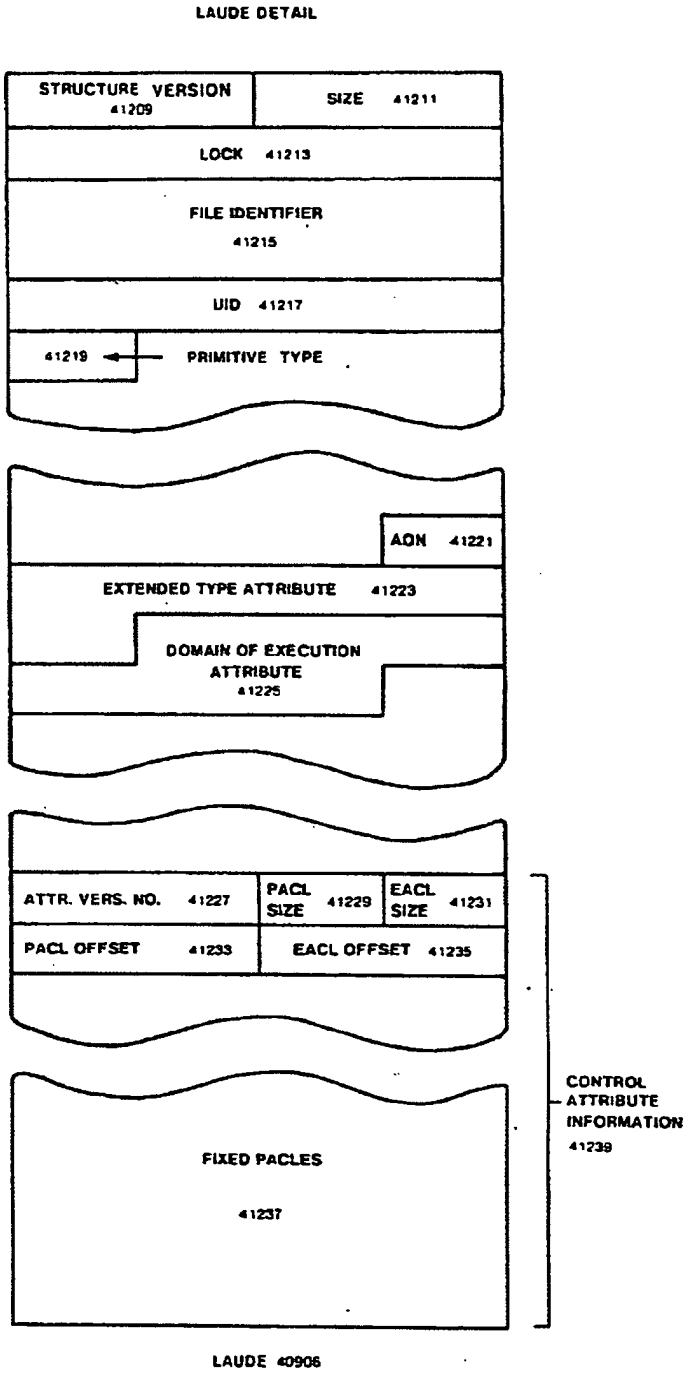


FIG. 412

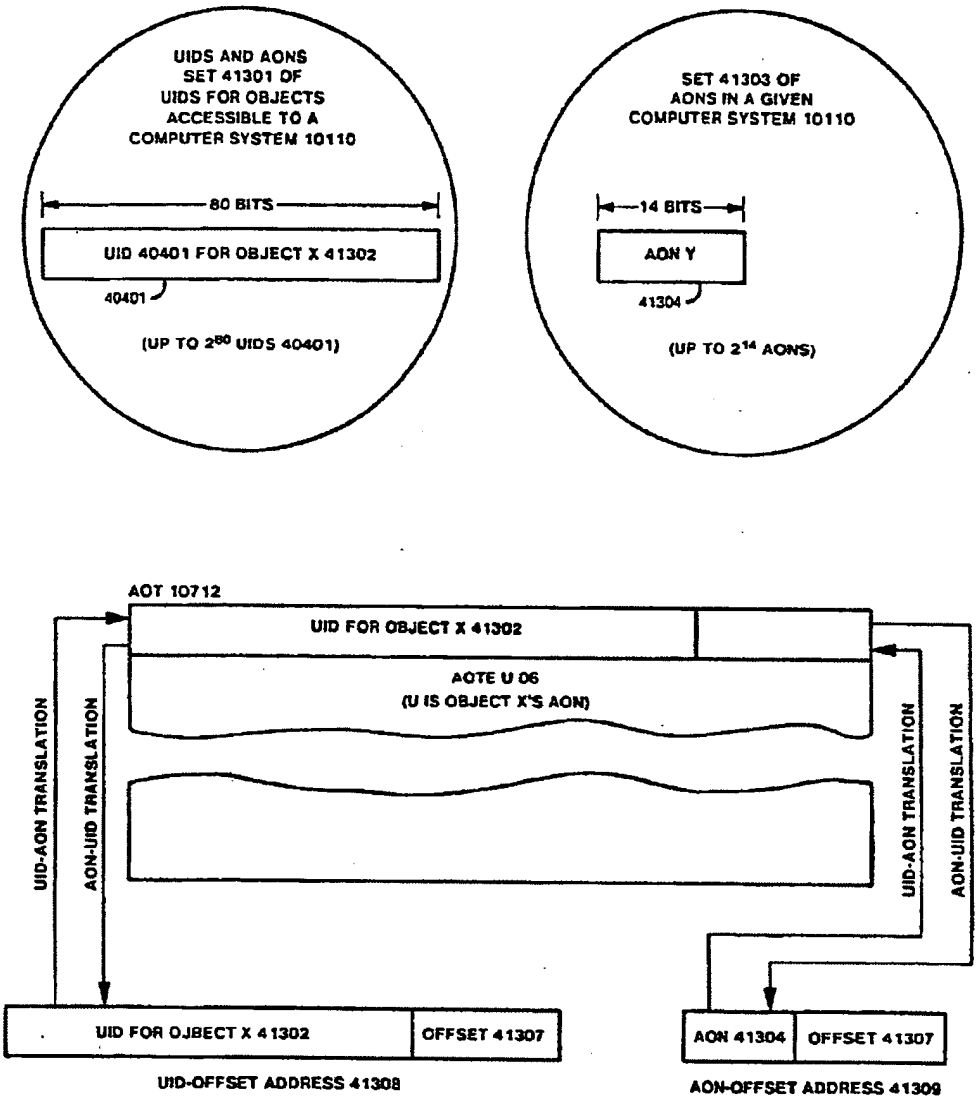


FIG 413

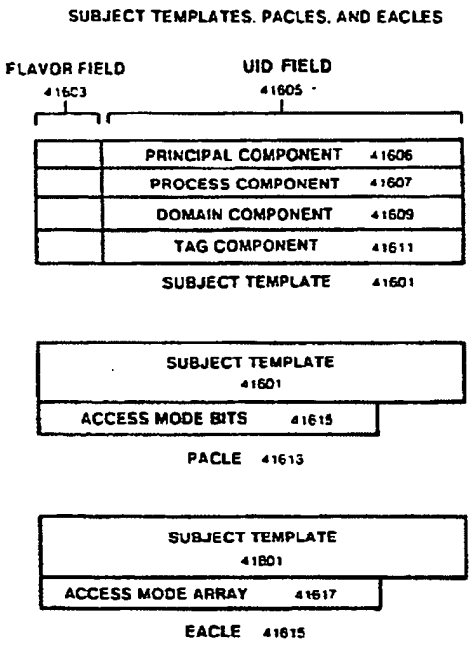


FIG. 416

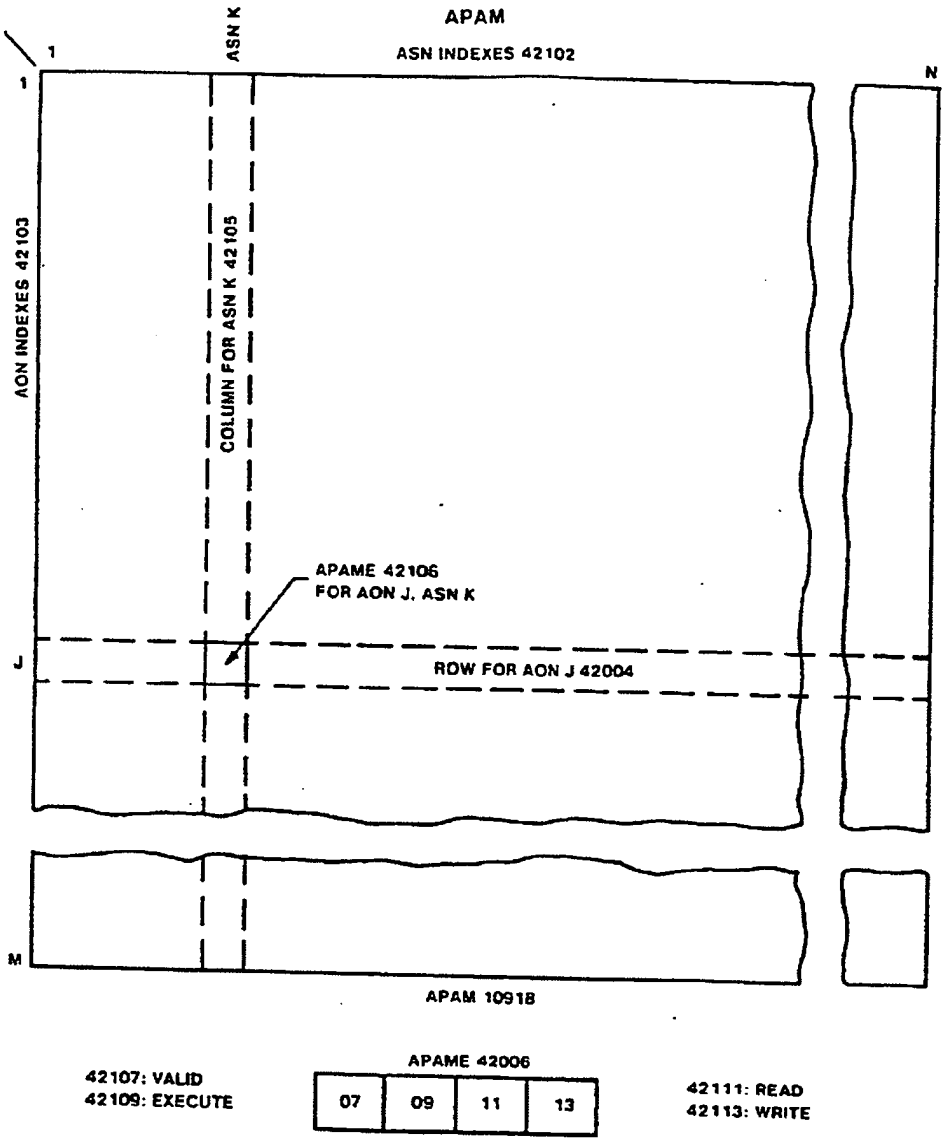


FIG. 421

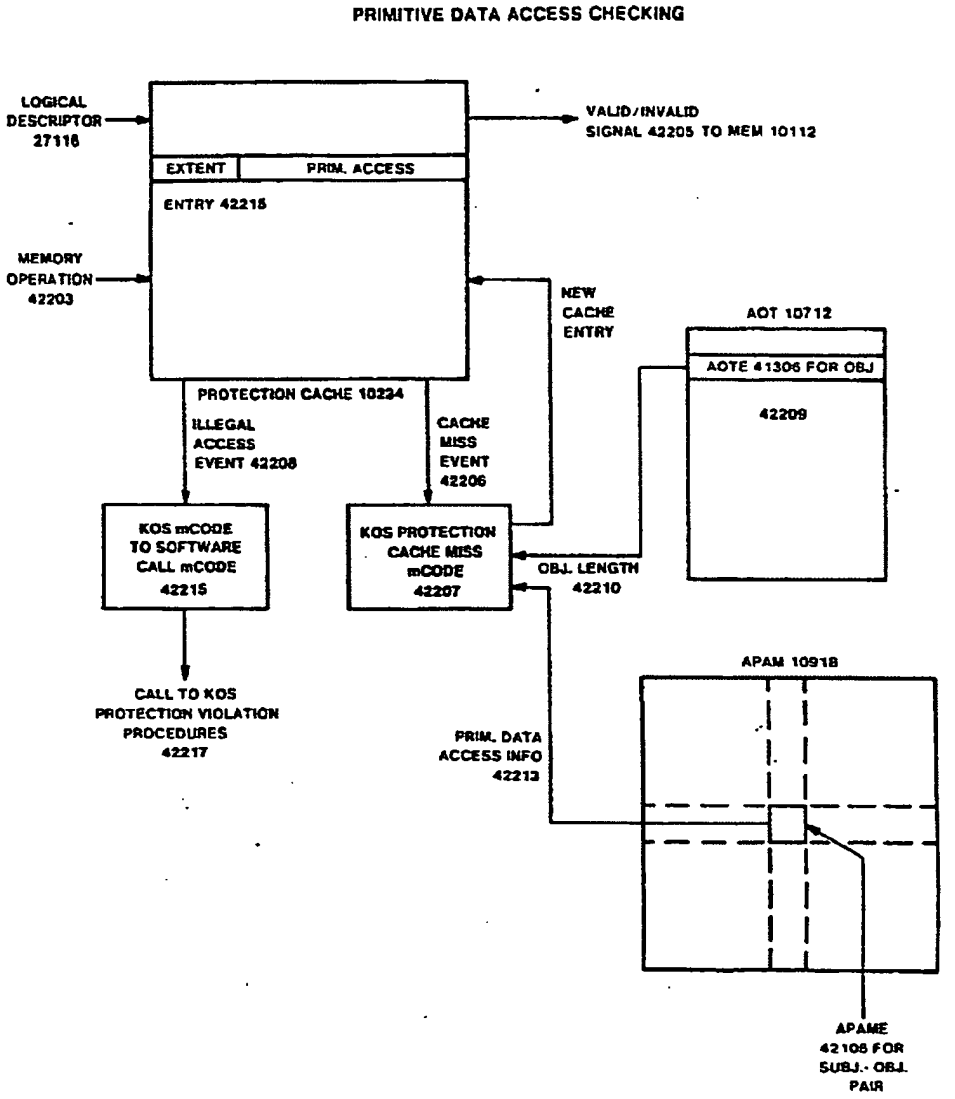


FIG. 422

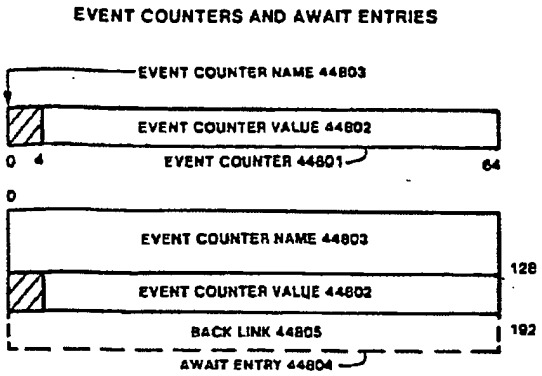


FIG. 448

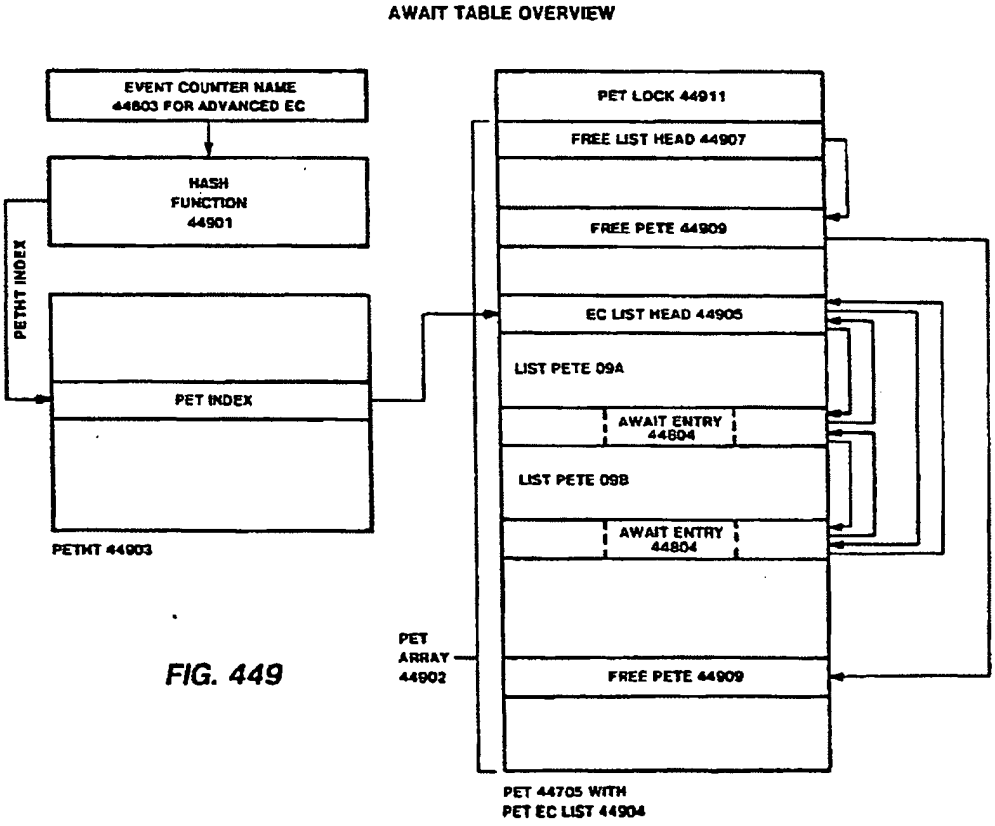


FIG. 449

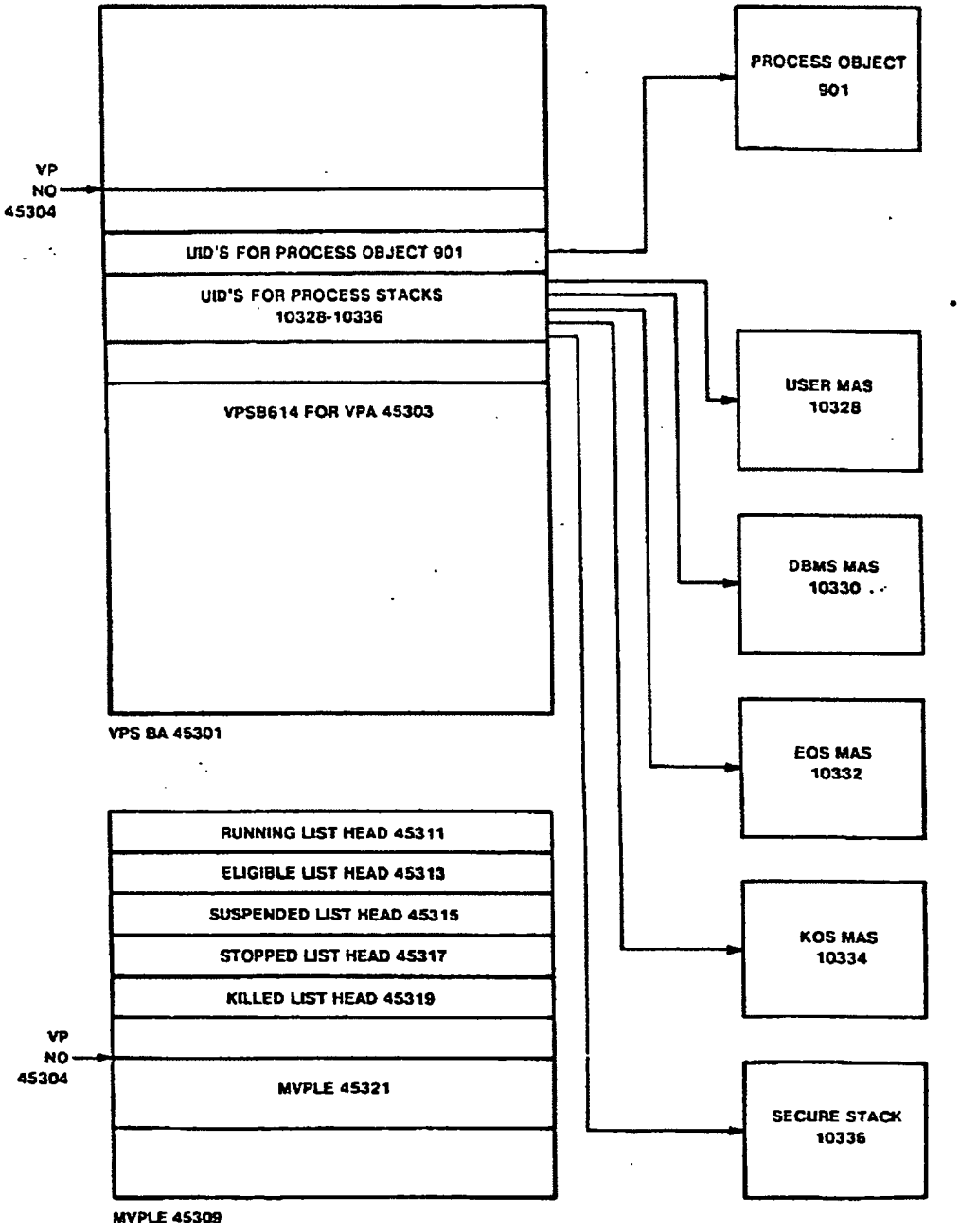


FIG. 453

VIRTUAL PROCESSOR SYNCHRONIZATION

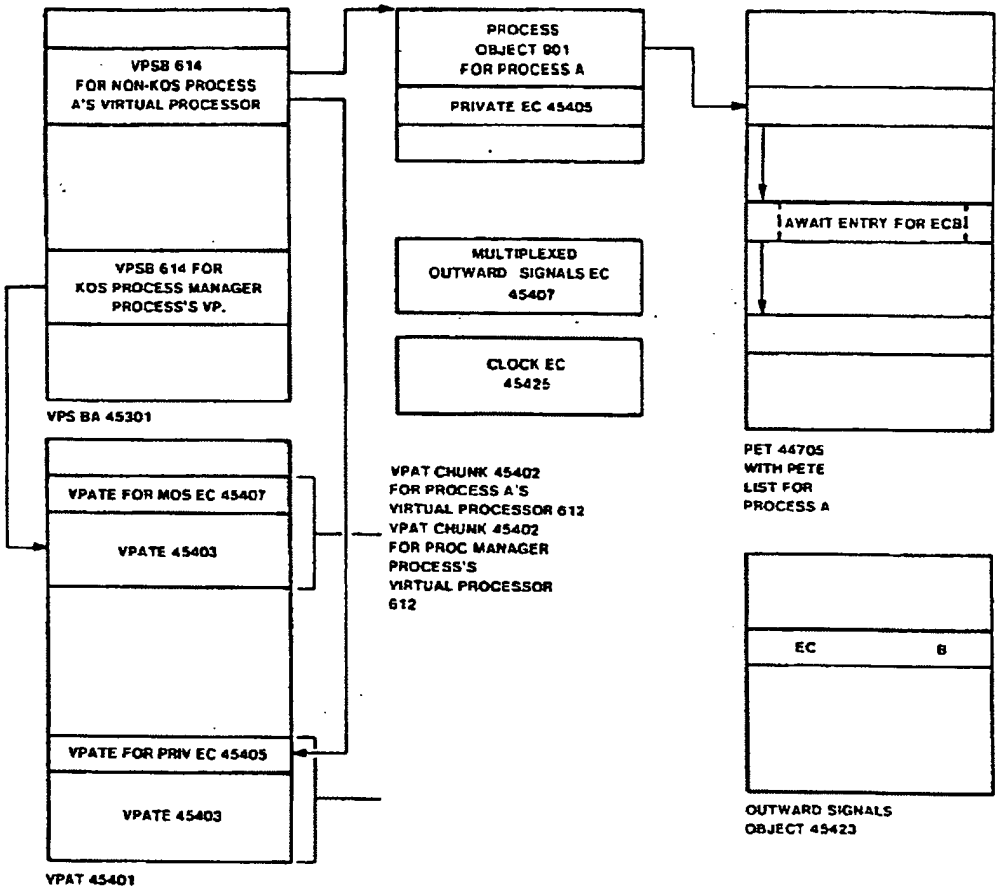


FIG. 454

MAS OBJECT OVERVIEW

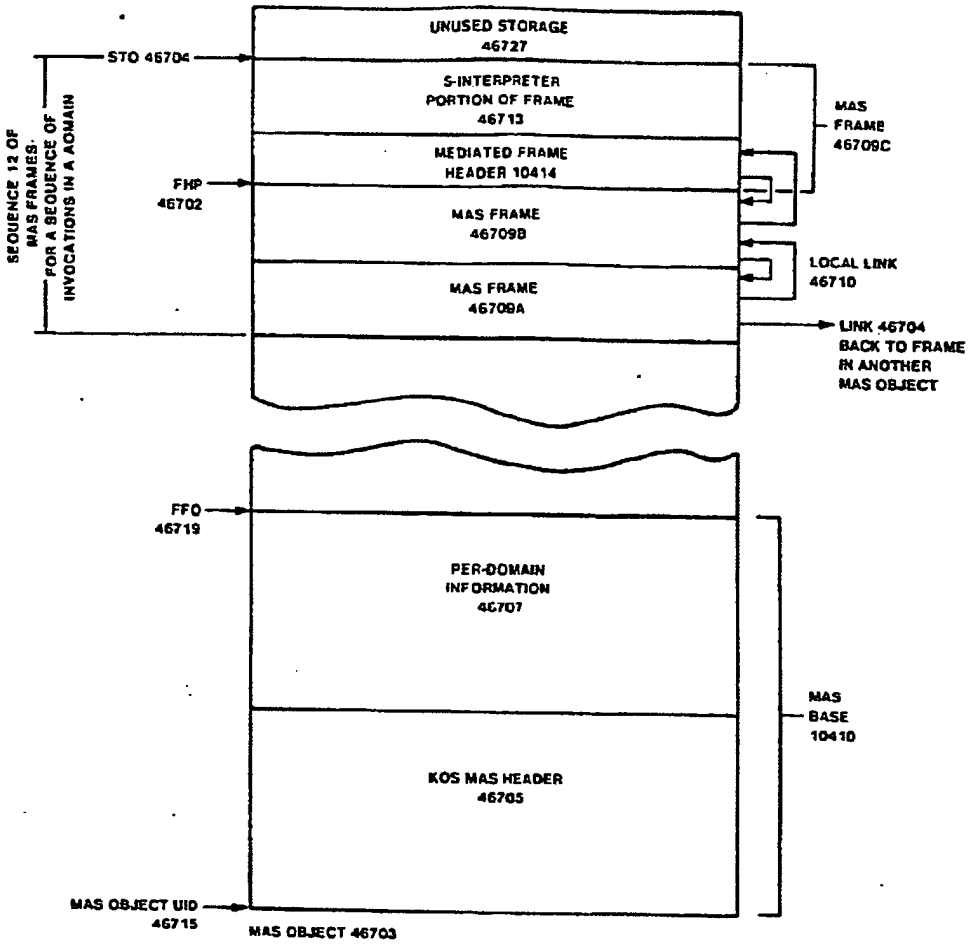


FIG. 467

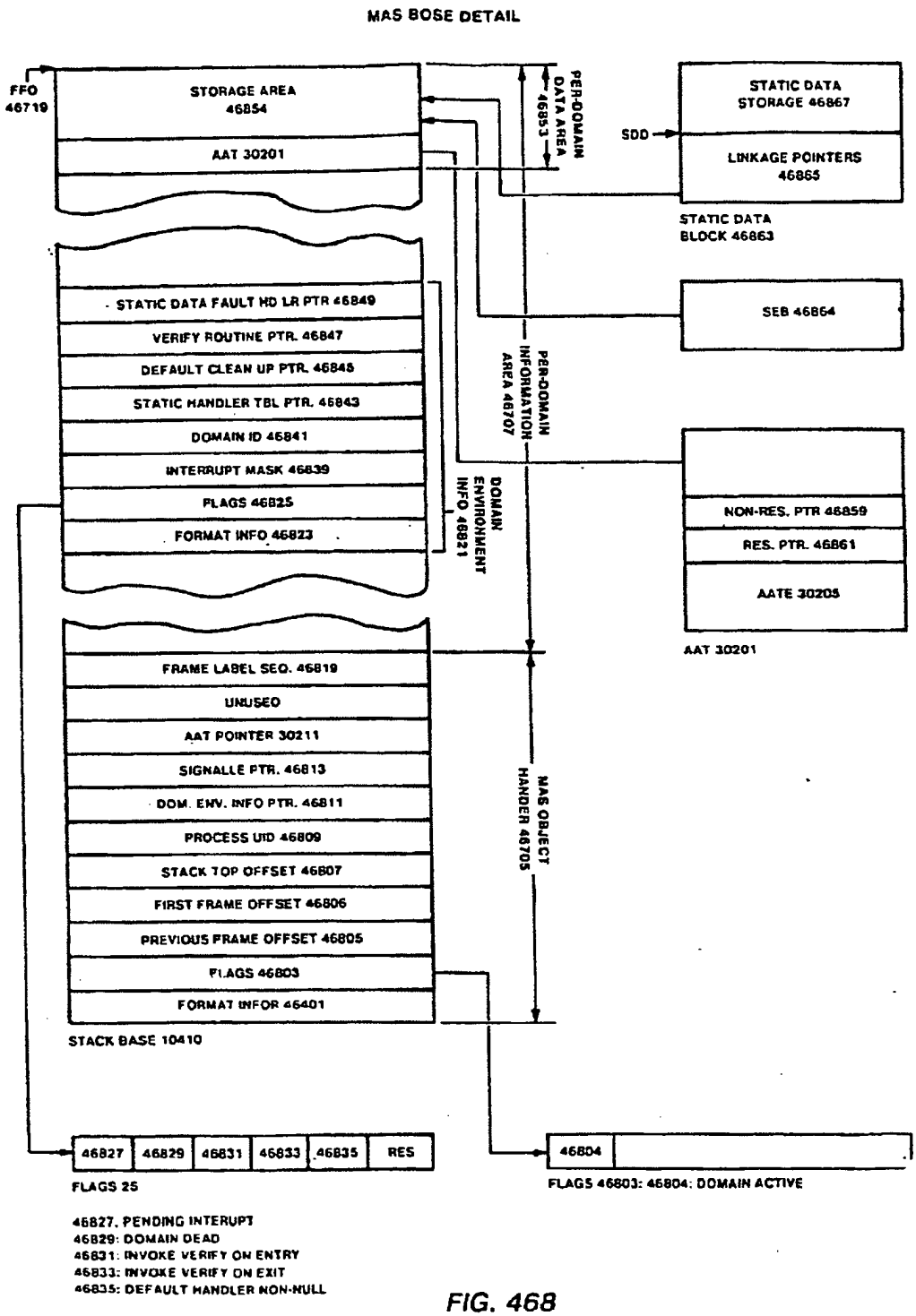


FIG. 468

MAS FRAME DETAIL

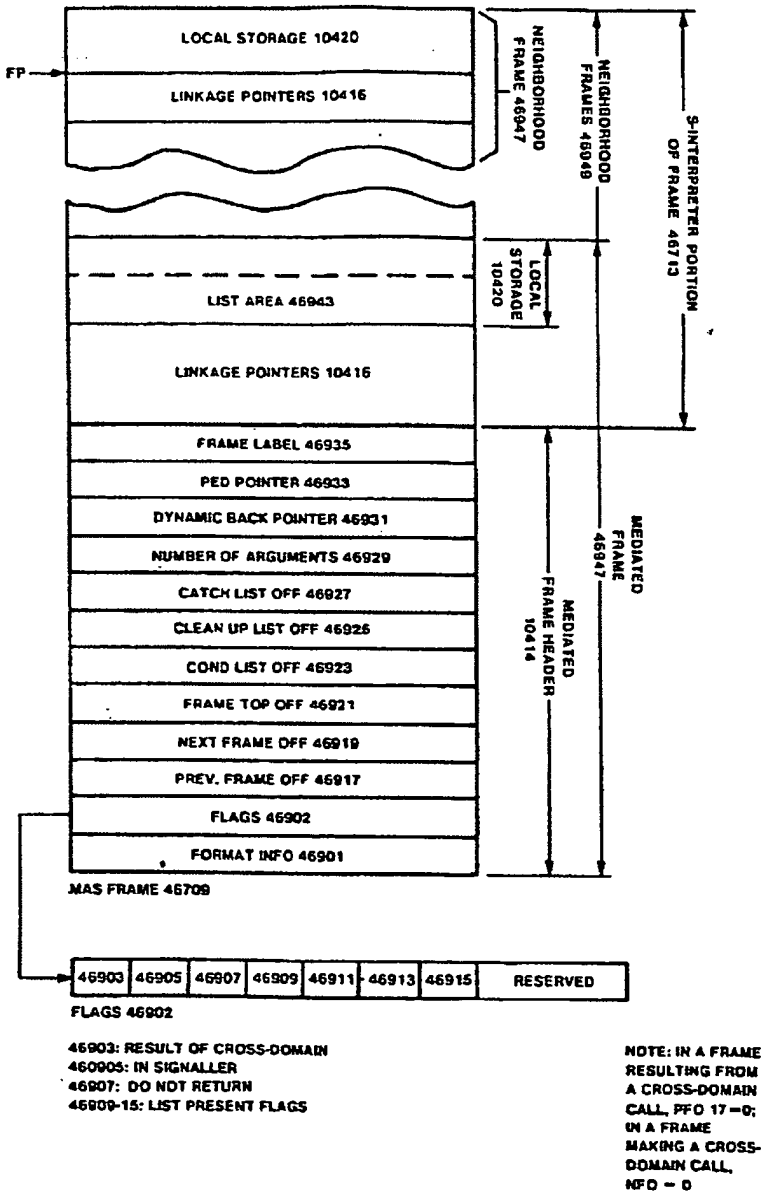


FIG. 469

SS 10336 OVERVIEW

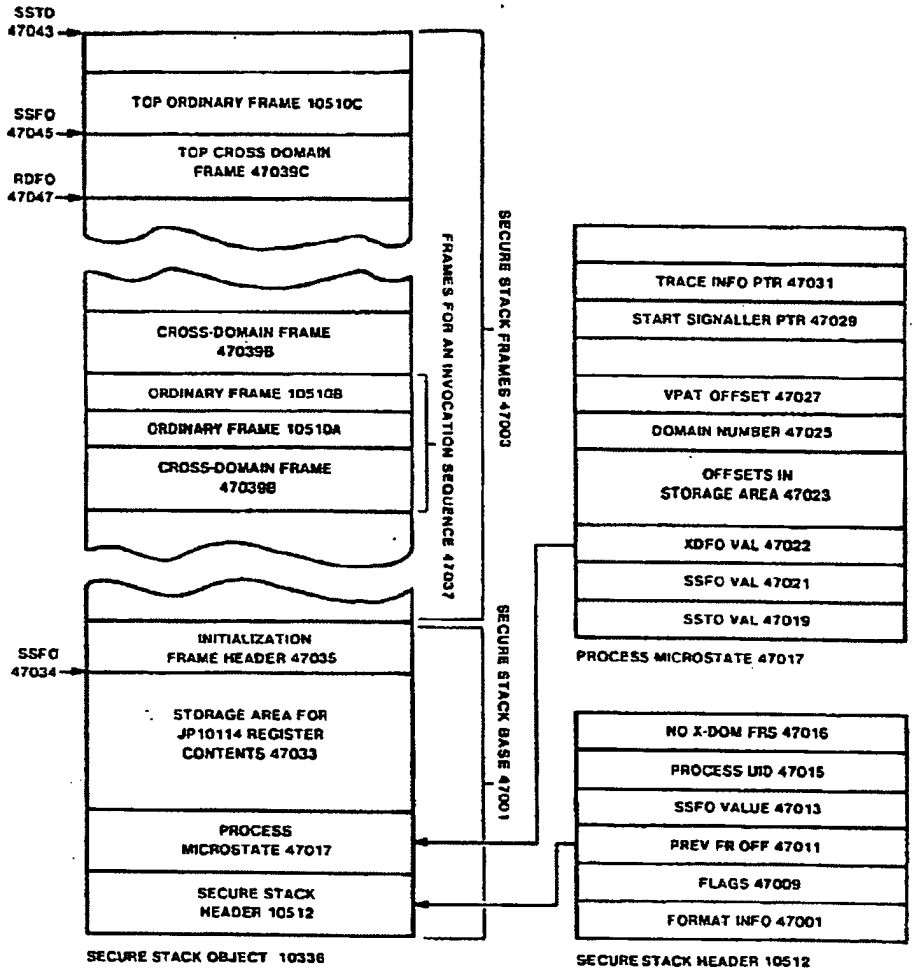


FIG. 470

SECURE STACK 10336 FRAME DETAIL

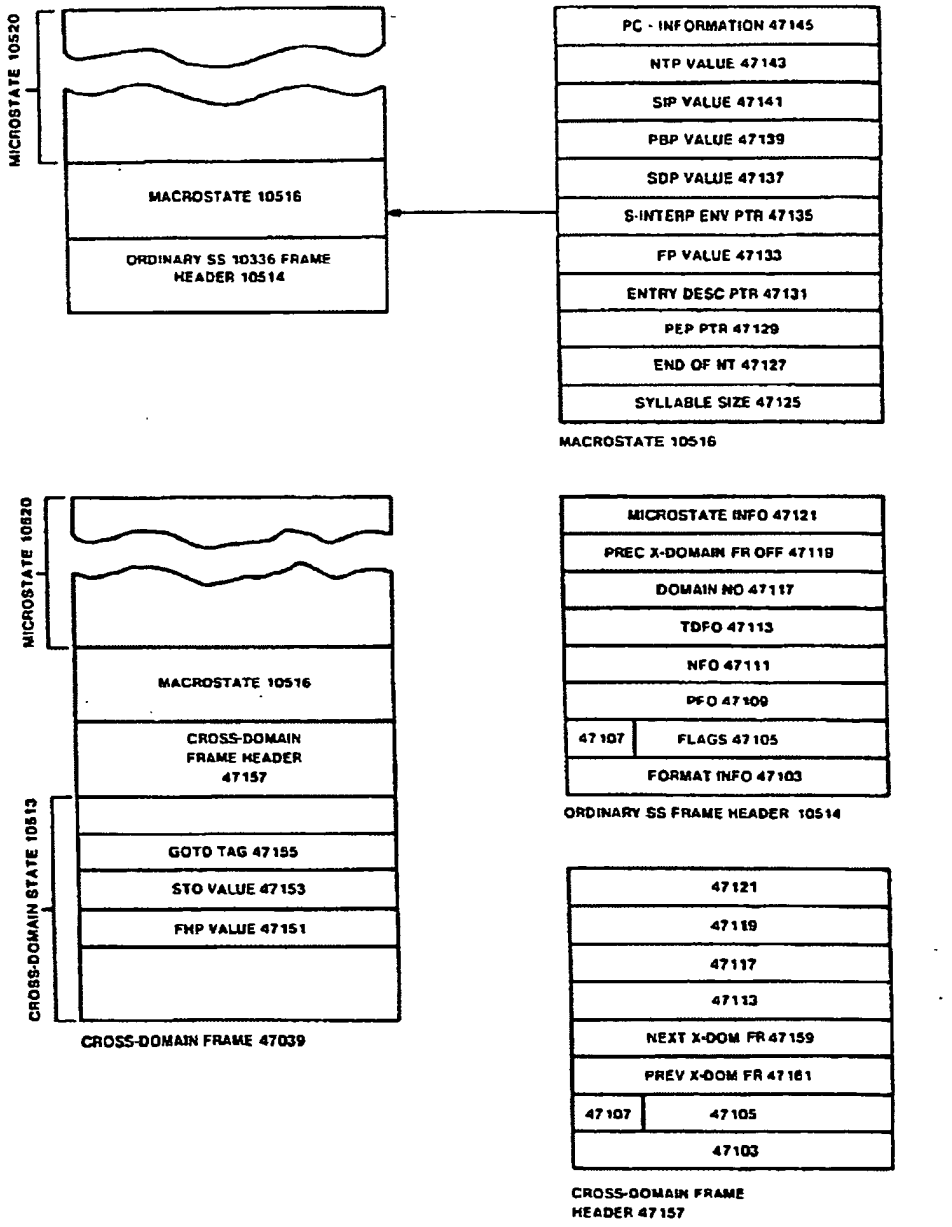


FIG. 471

PROCEDURE OBJECT OVERVIEW

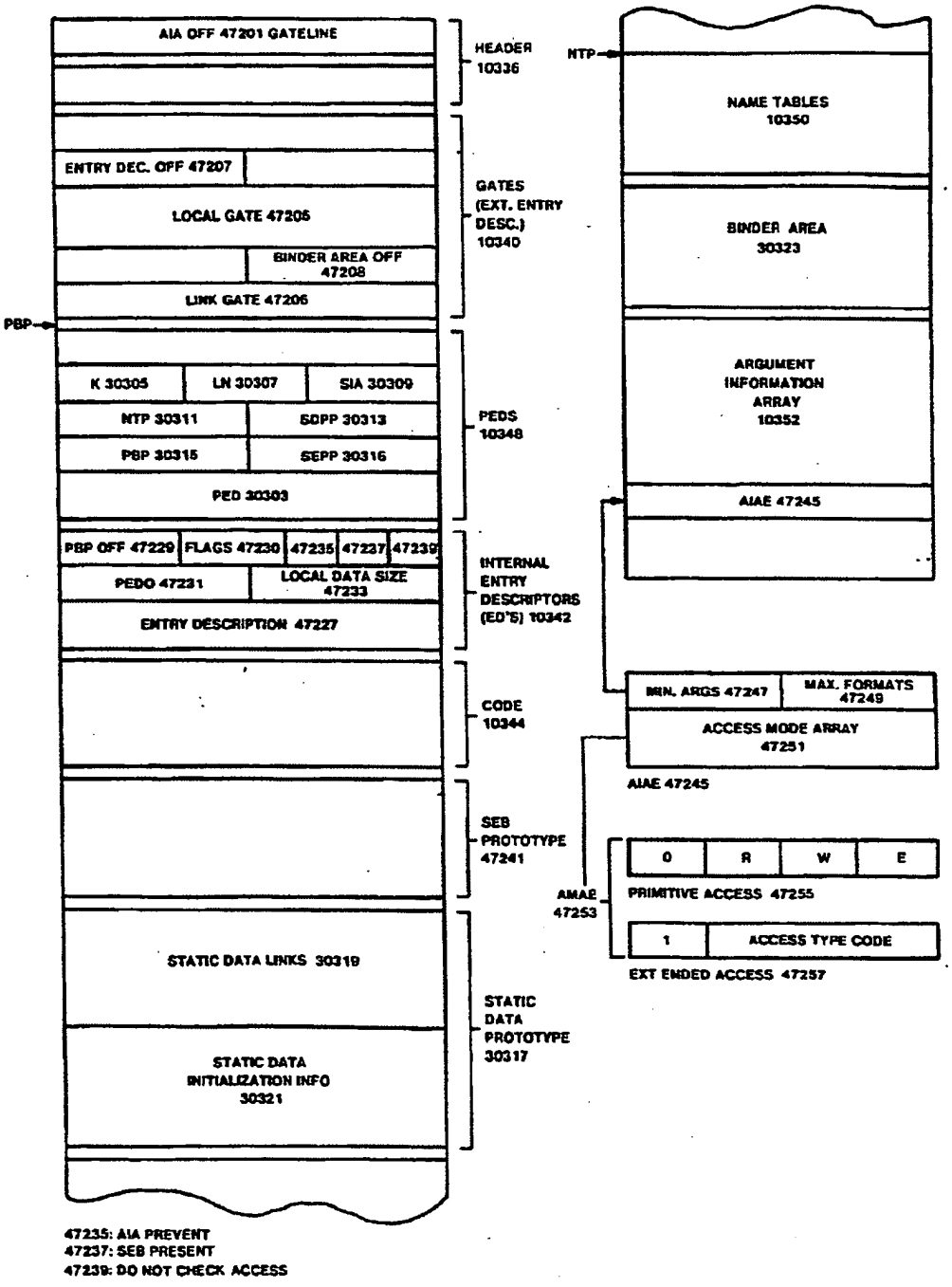


FIG. 472

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

(11) Publication number: **0 613 073 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)

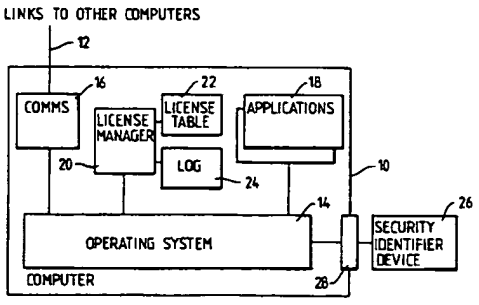
(72) Inventor: **Archer, Barrie**
Lilac Cottage,
Honey Hall
Wokingham, Berkshire RG11 3BA (GB)

(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.

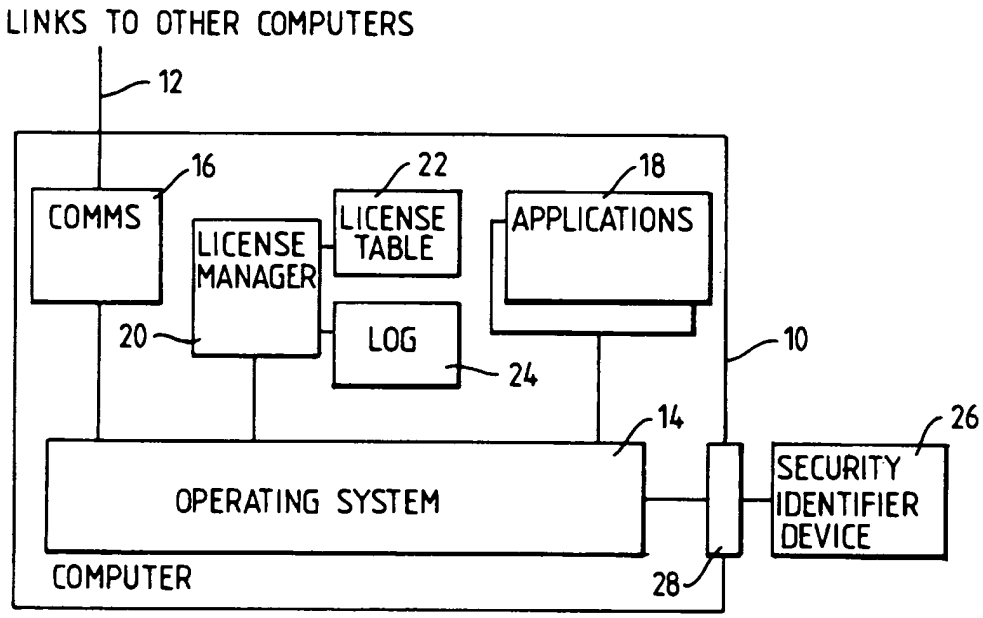


Fig. 2.

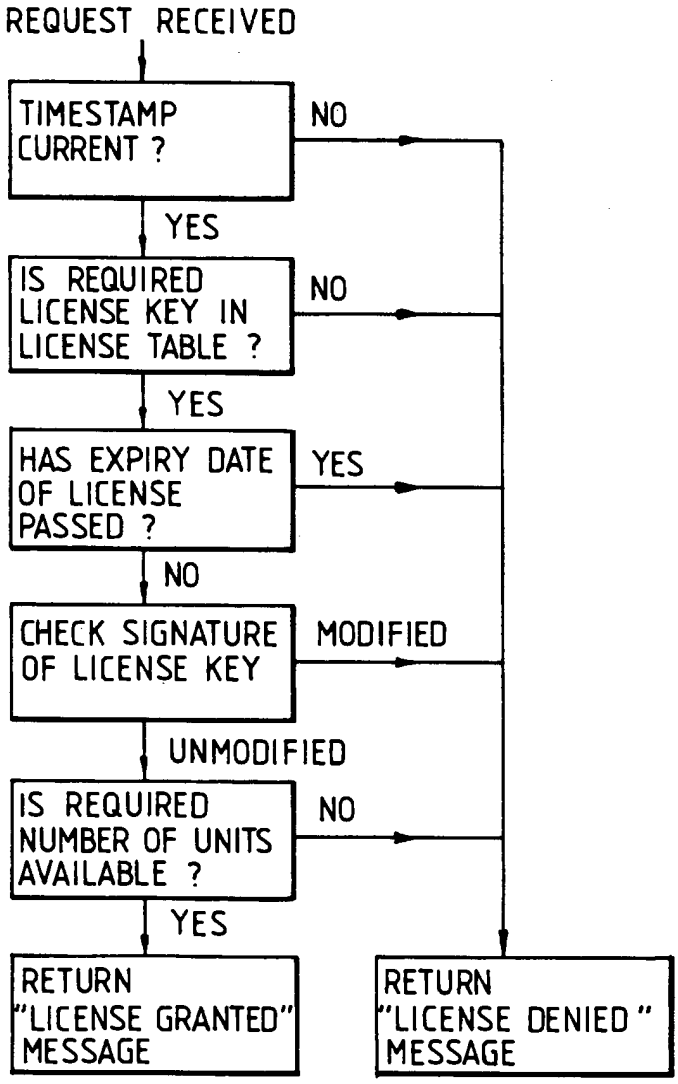
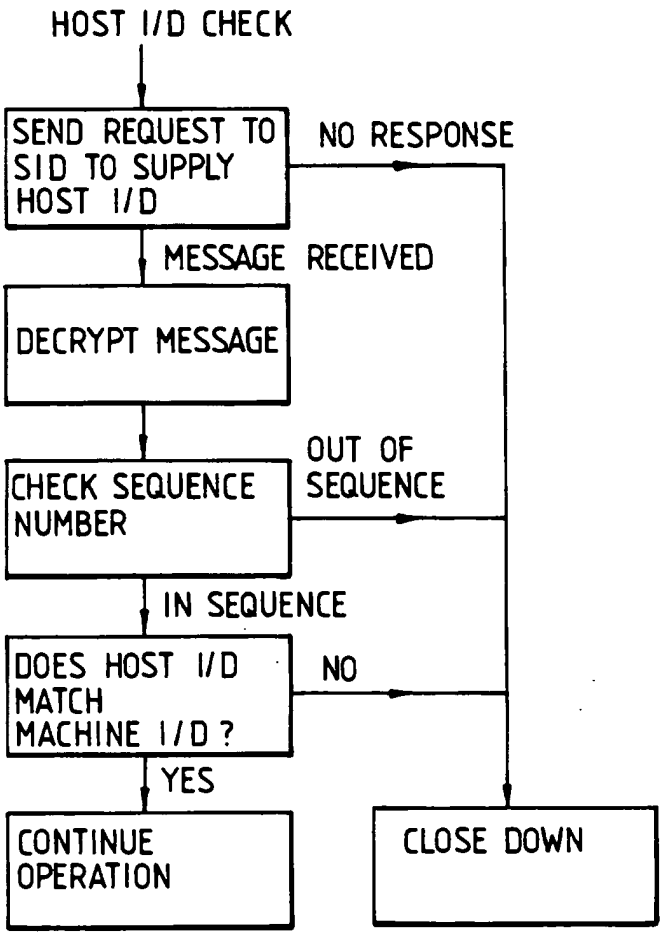


Fig.3.



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 retum 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 retum 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.

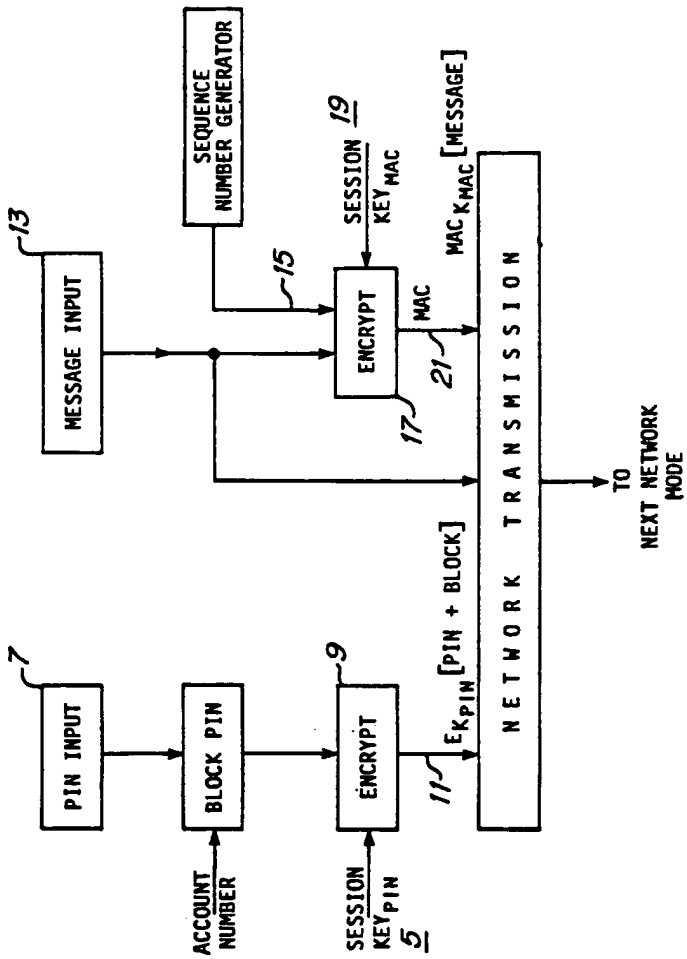


Figure 1
(PRIOR ART)

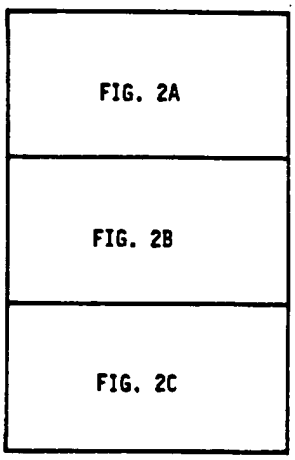


Figure 2

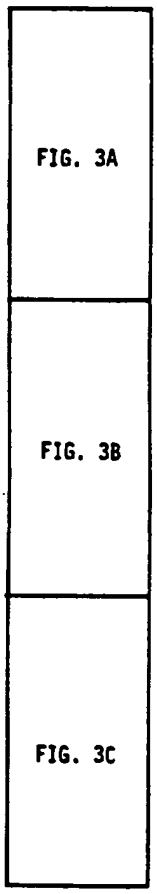


Figure 3

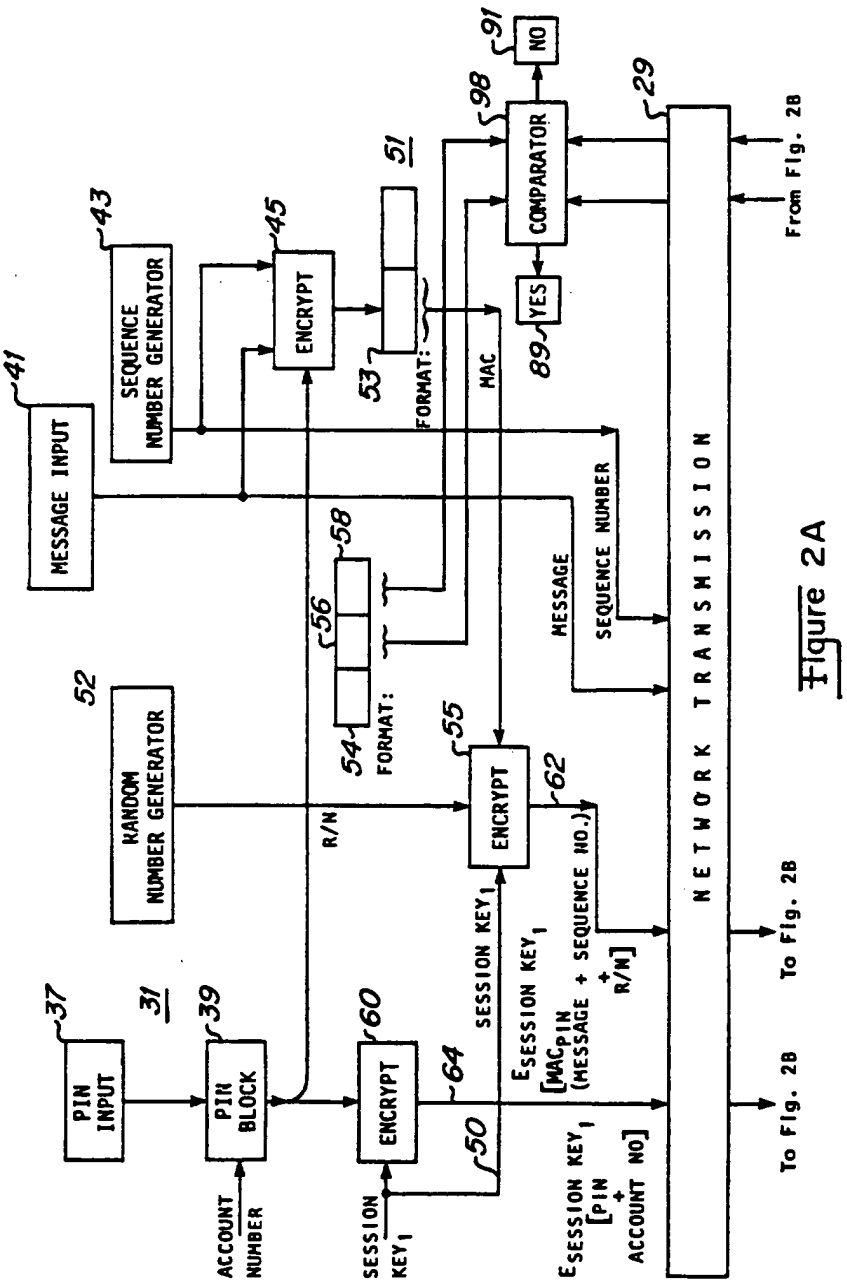


Figure 2A

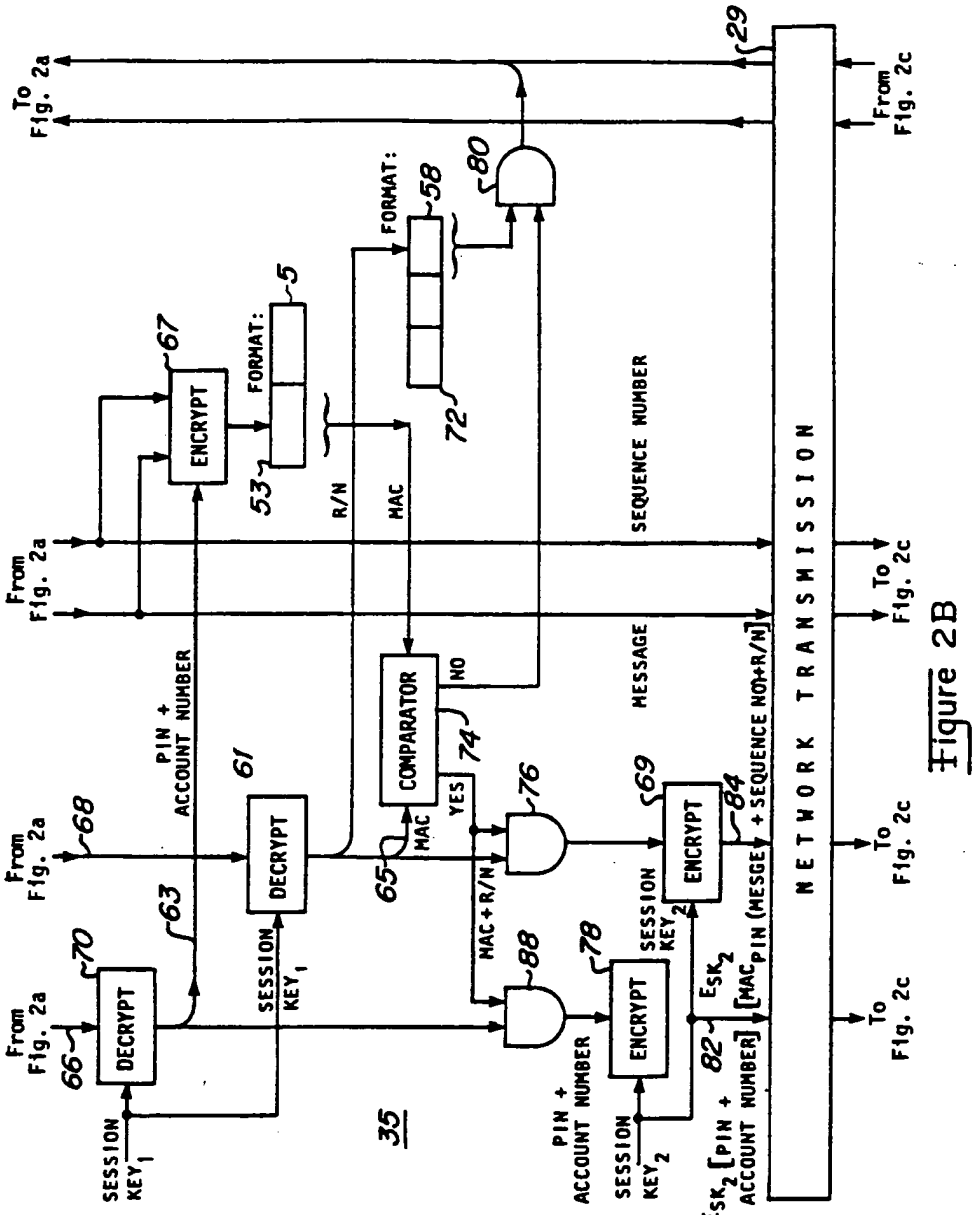


Figure 2B

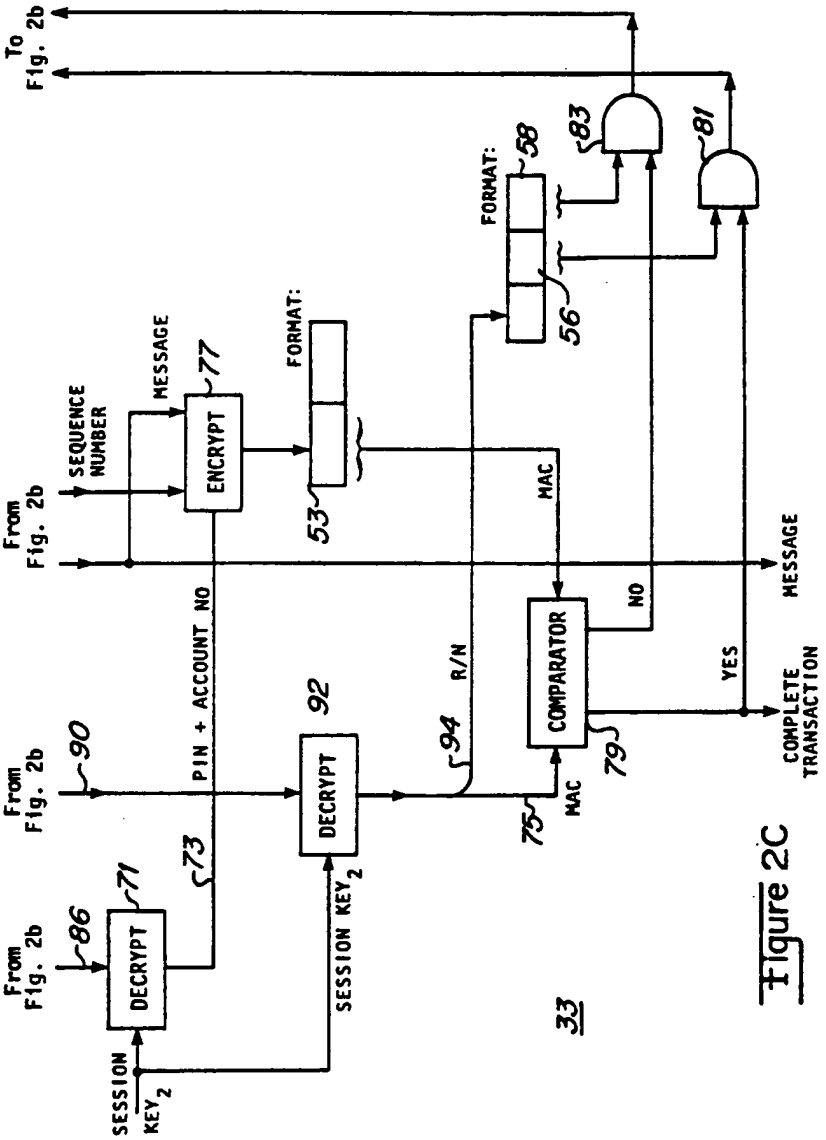


Figure 2C

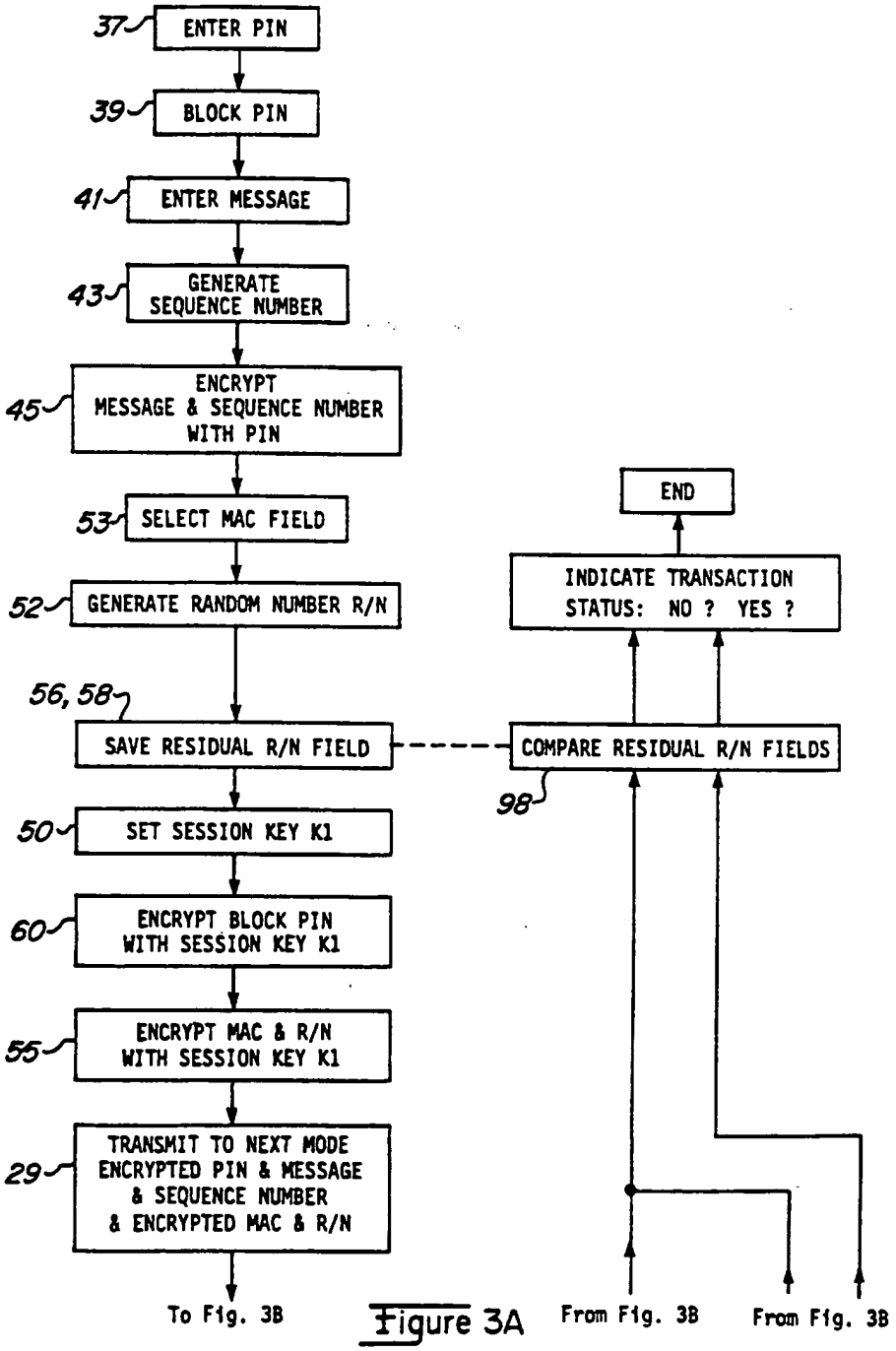


Figure 3A

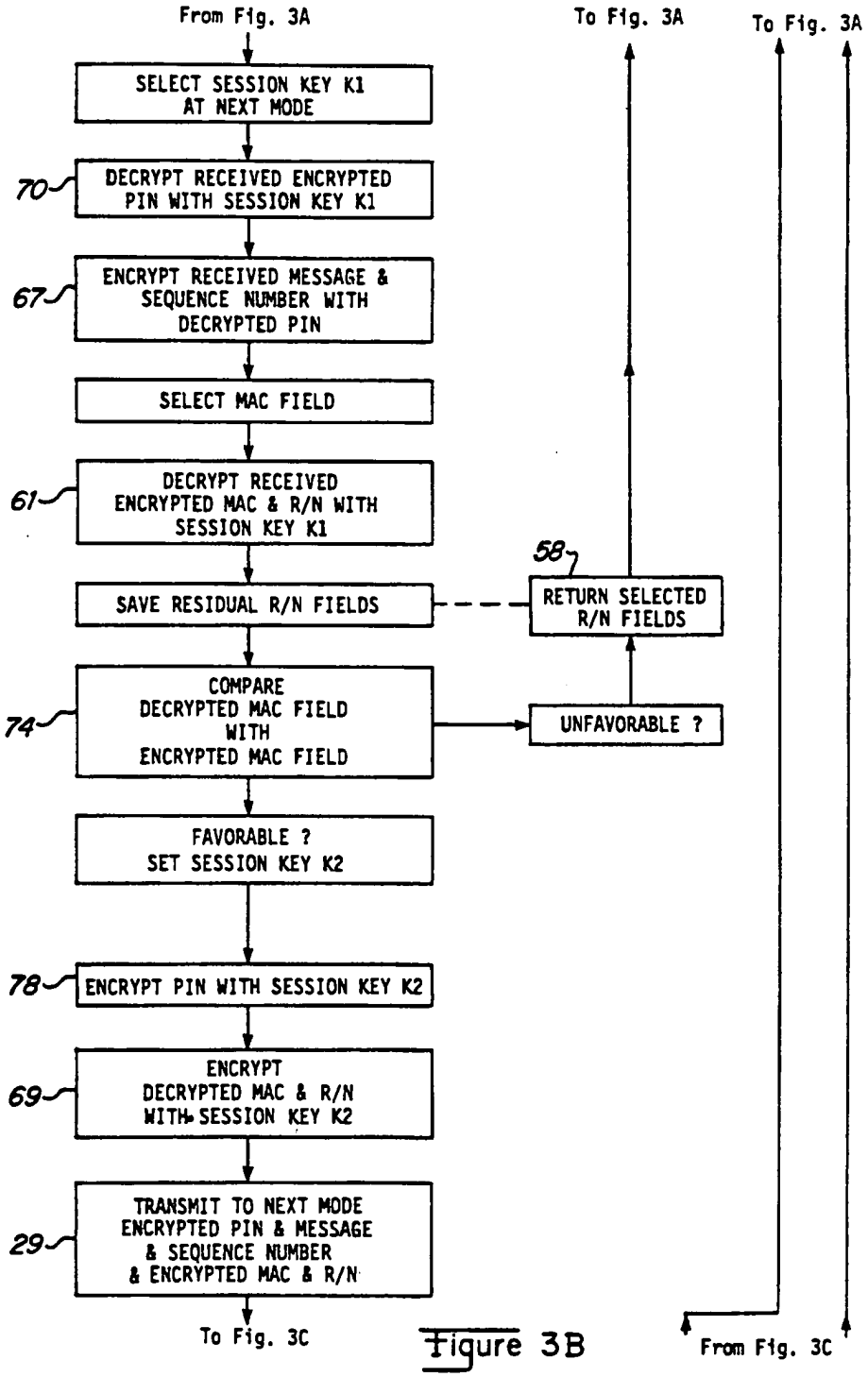


Figure 3B

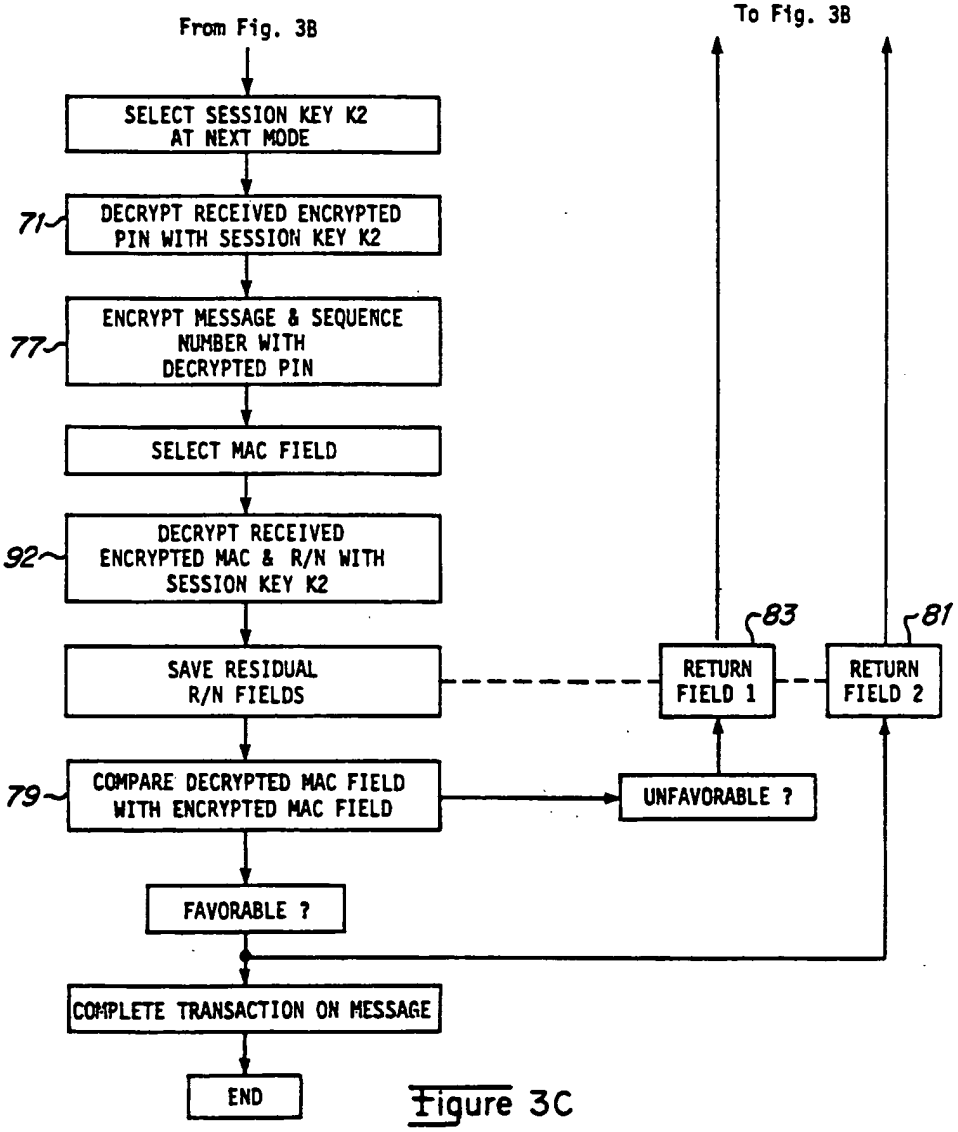


Figure 3C

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)

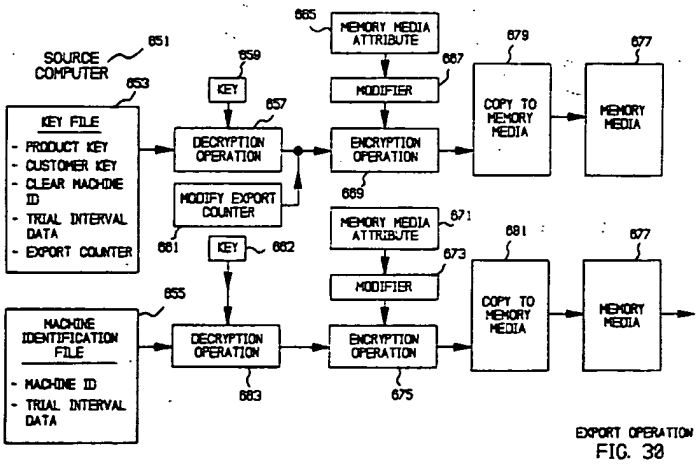
Inventor: **Nagda, Jagdish**
701 Kalmia Avenue
Boulder,
Colorado 80304 (US)

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. 3B

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.

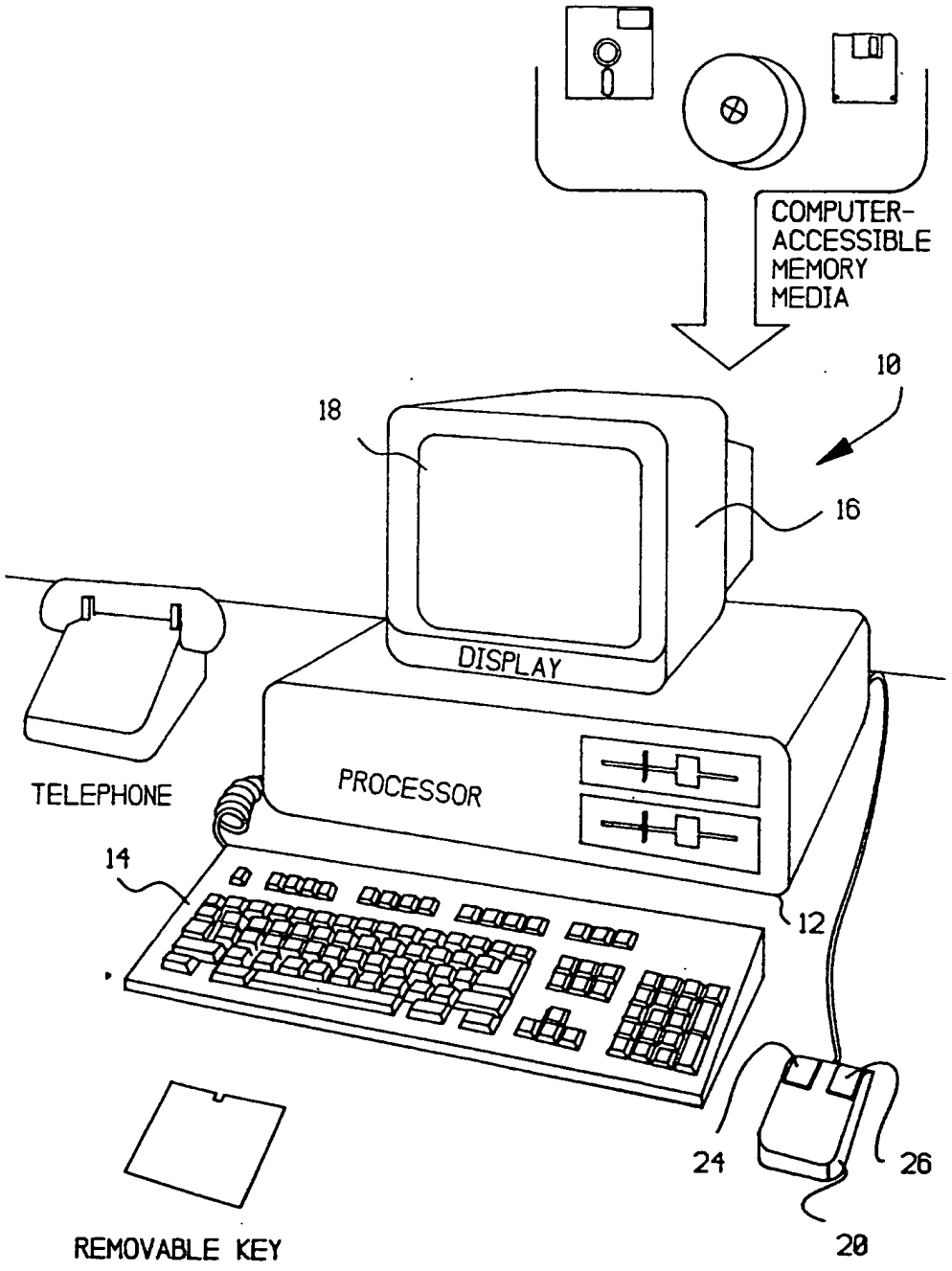
6. A method of passing encrypted files between data processing systems, comprising:
at a source computer providing at least one file which is encrypted with a key which is at least partially derived from at least one unique source computer system attribute;
providing a transfer memory medium;
initiating 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.

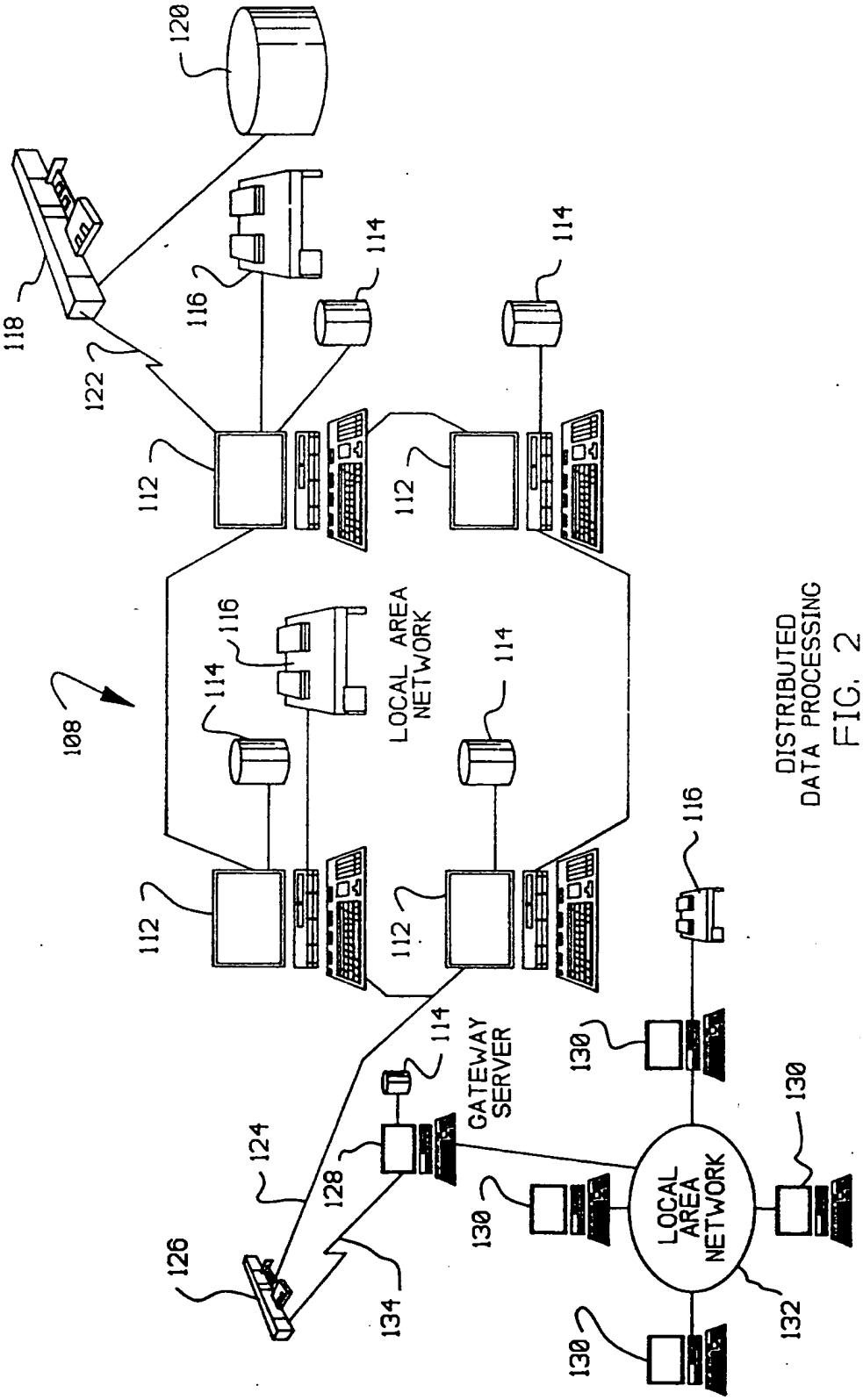
8. A method of passing encrypted files according to Claim 6 or 7, further comprising:
at said target computer, updating said at least one file to provide an identification of said source computer.

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

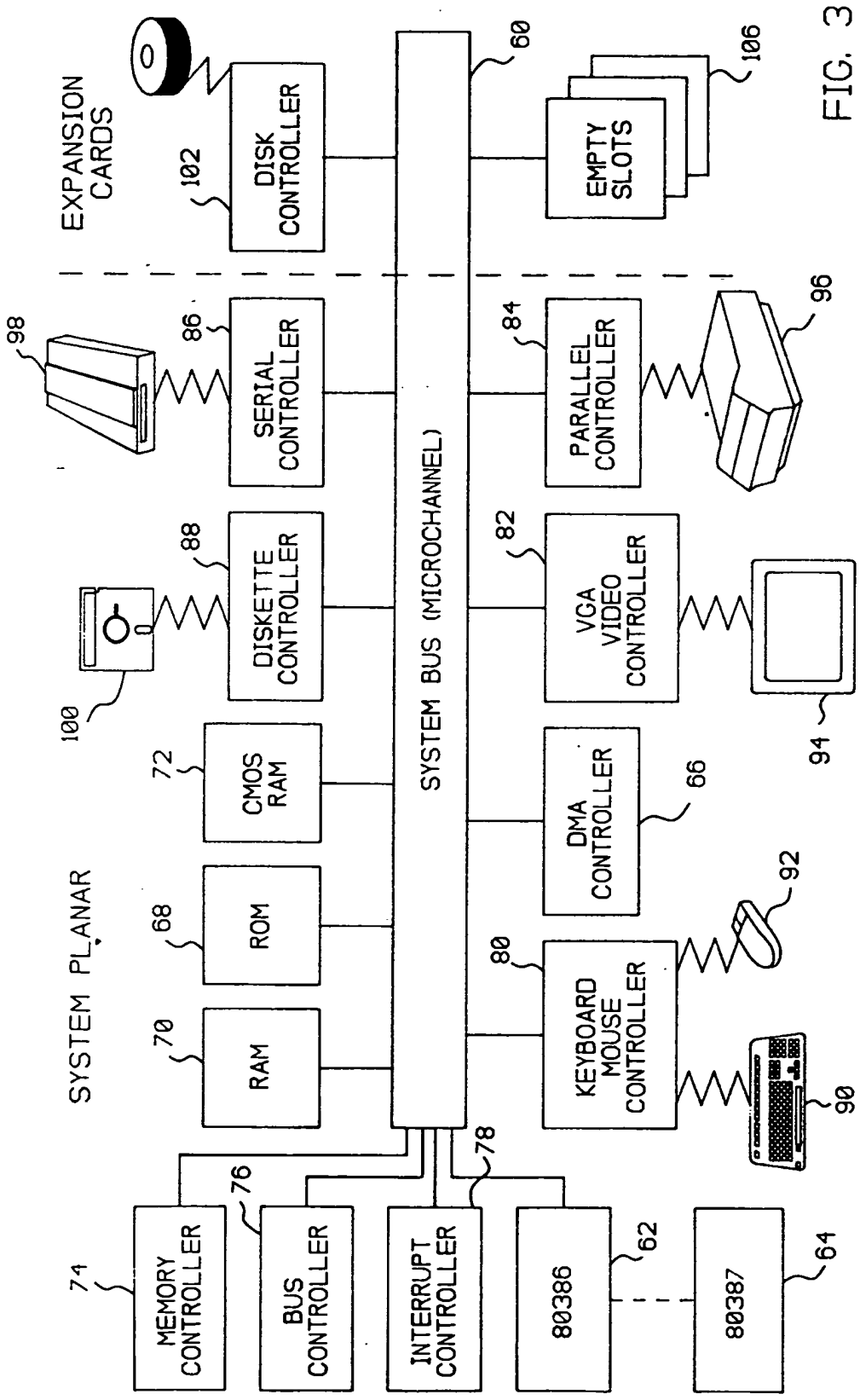


FIG. 3

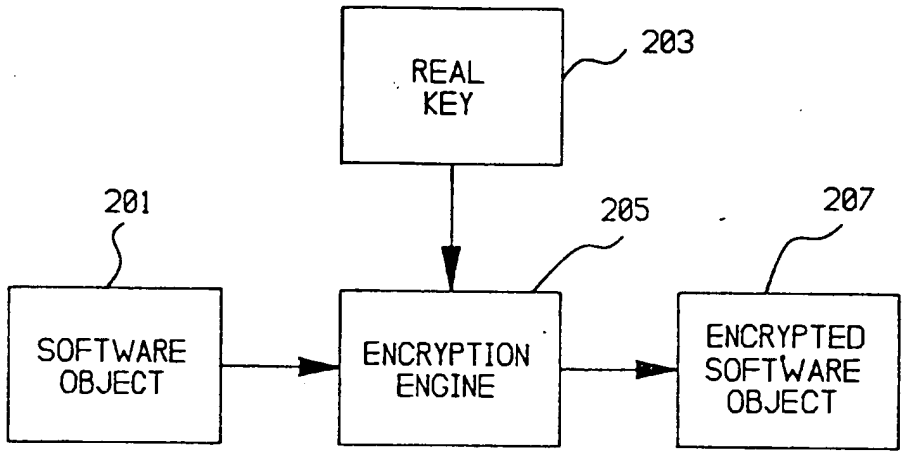


FIG. 4

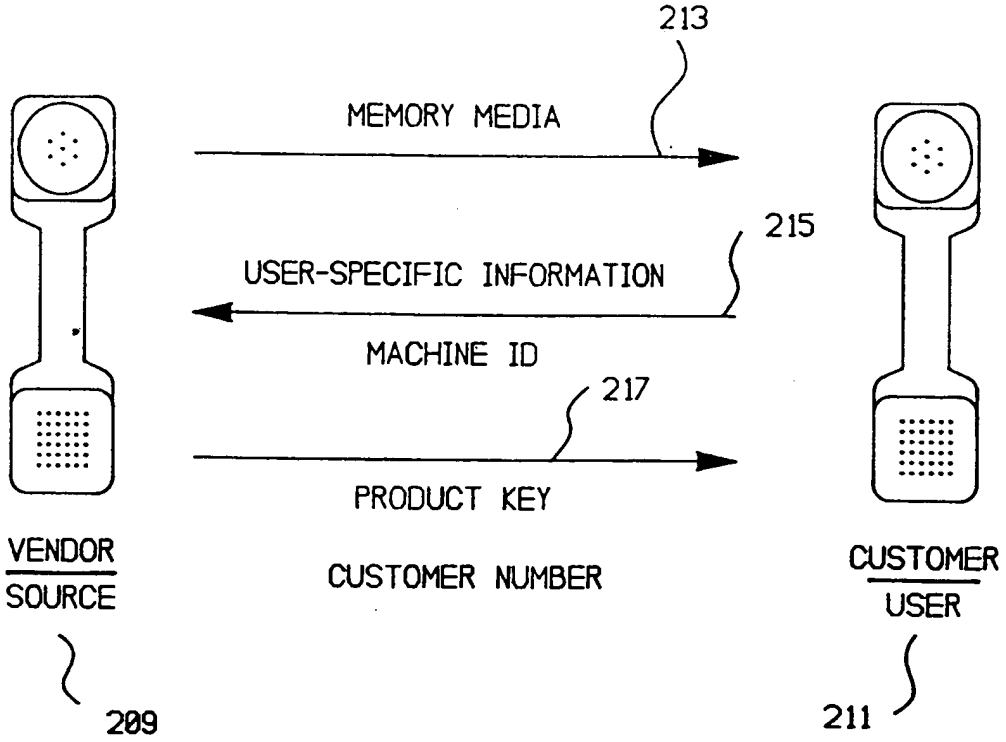
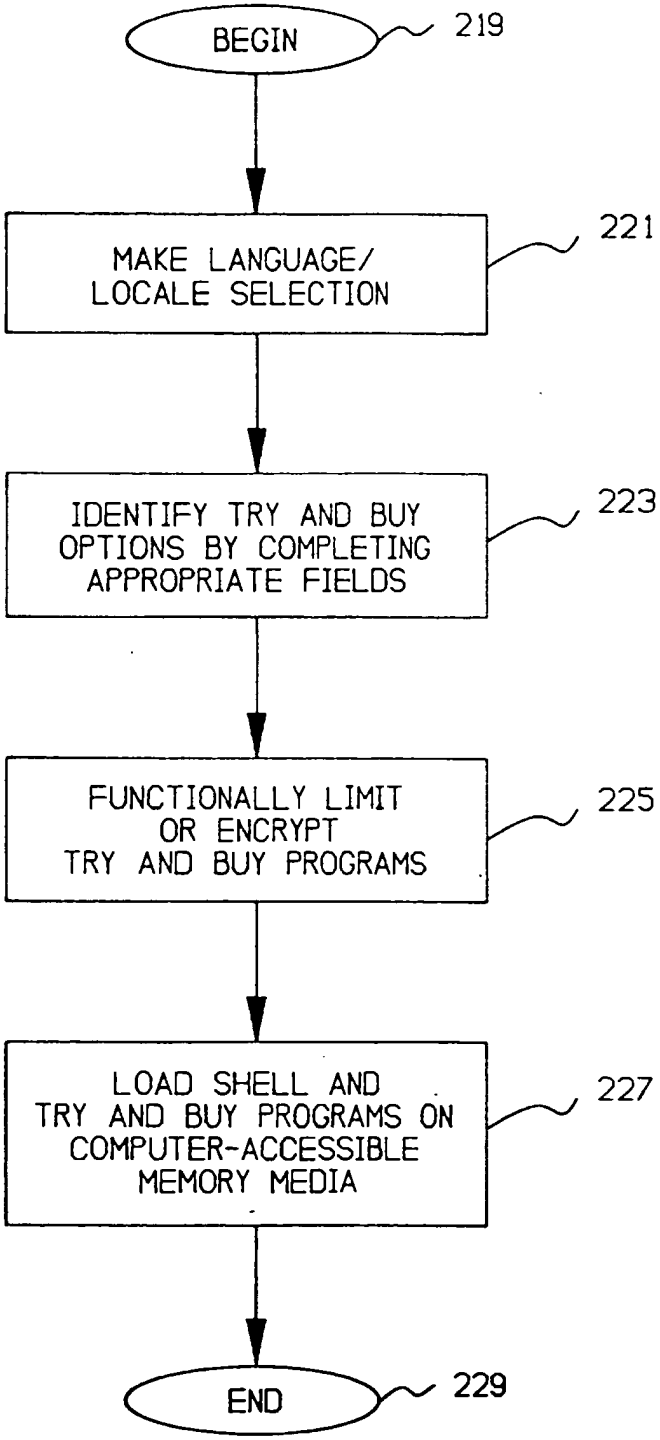


FIG. 5



BUILDING THE SHELL

FIG. 6

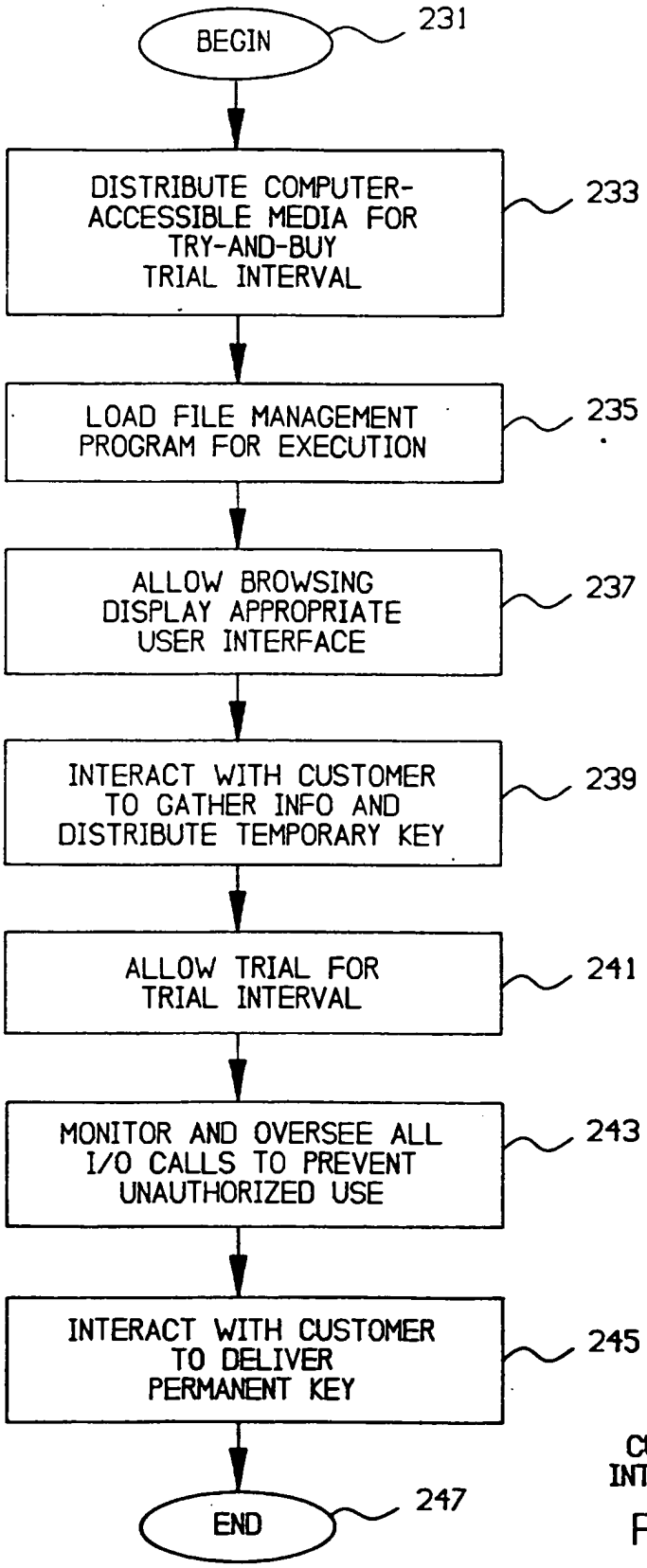


FIG. 7

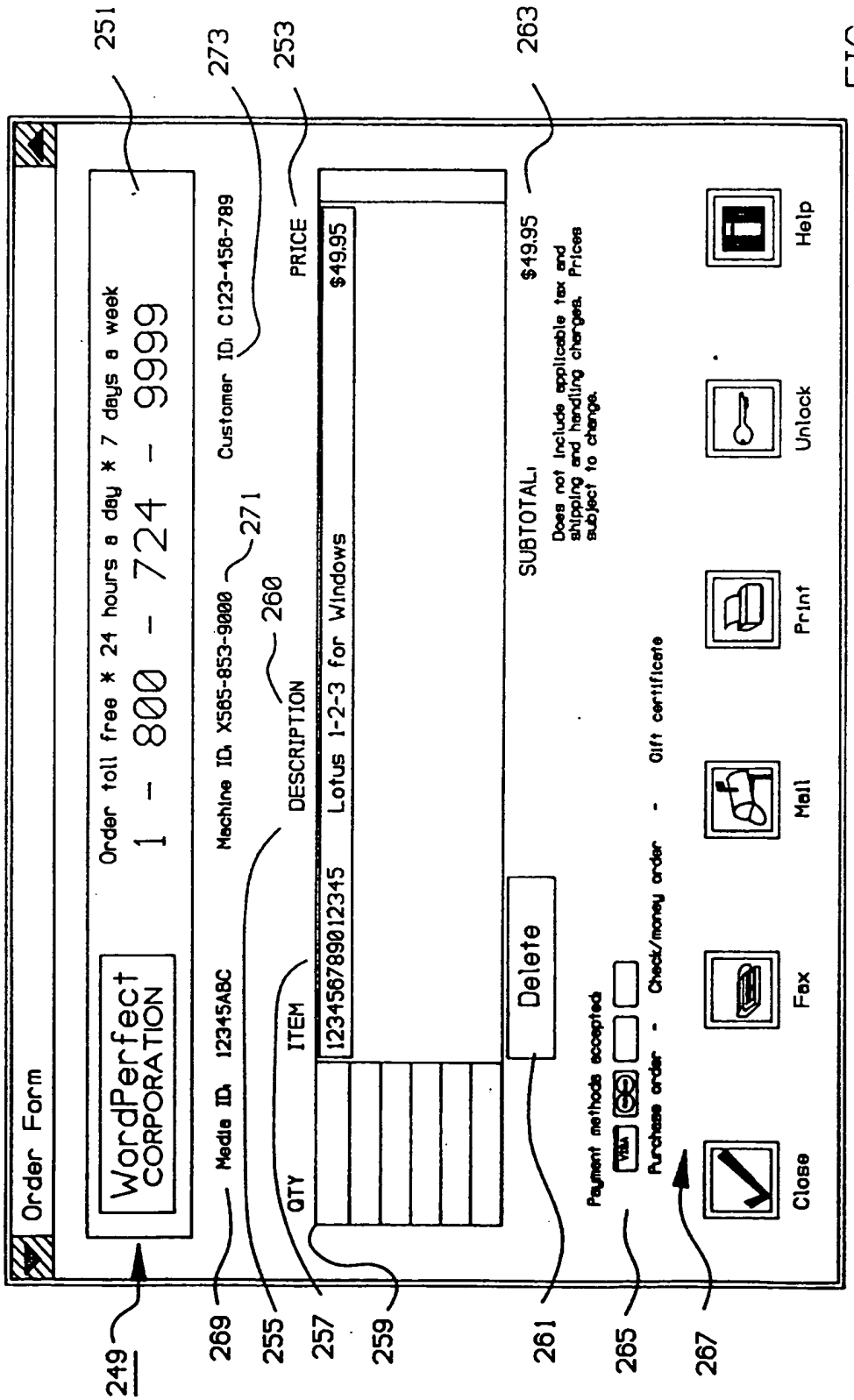


FIG. 8

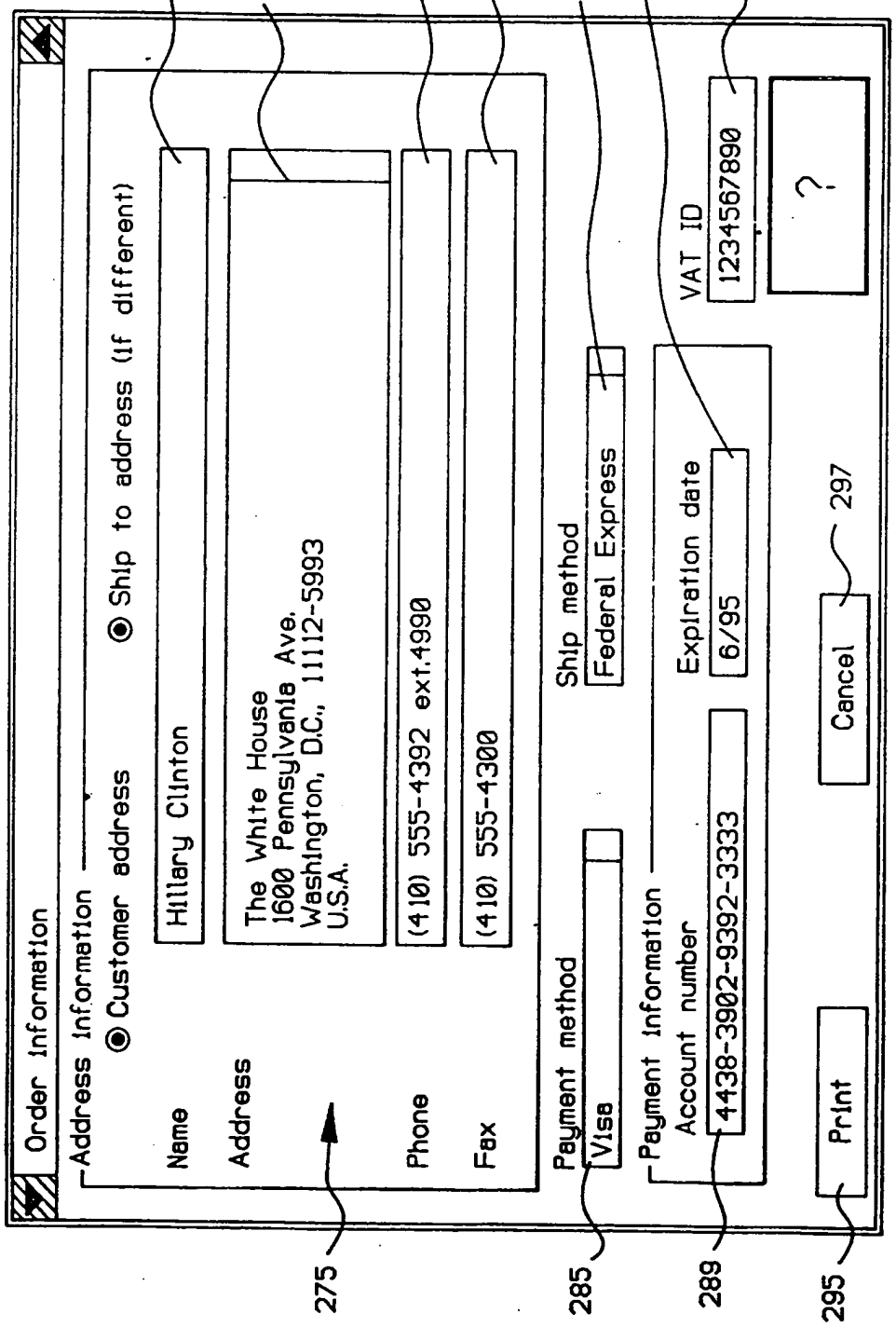


FIG. 9

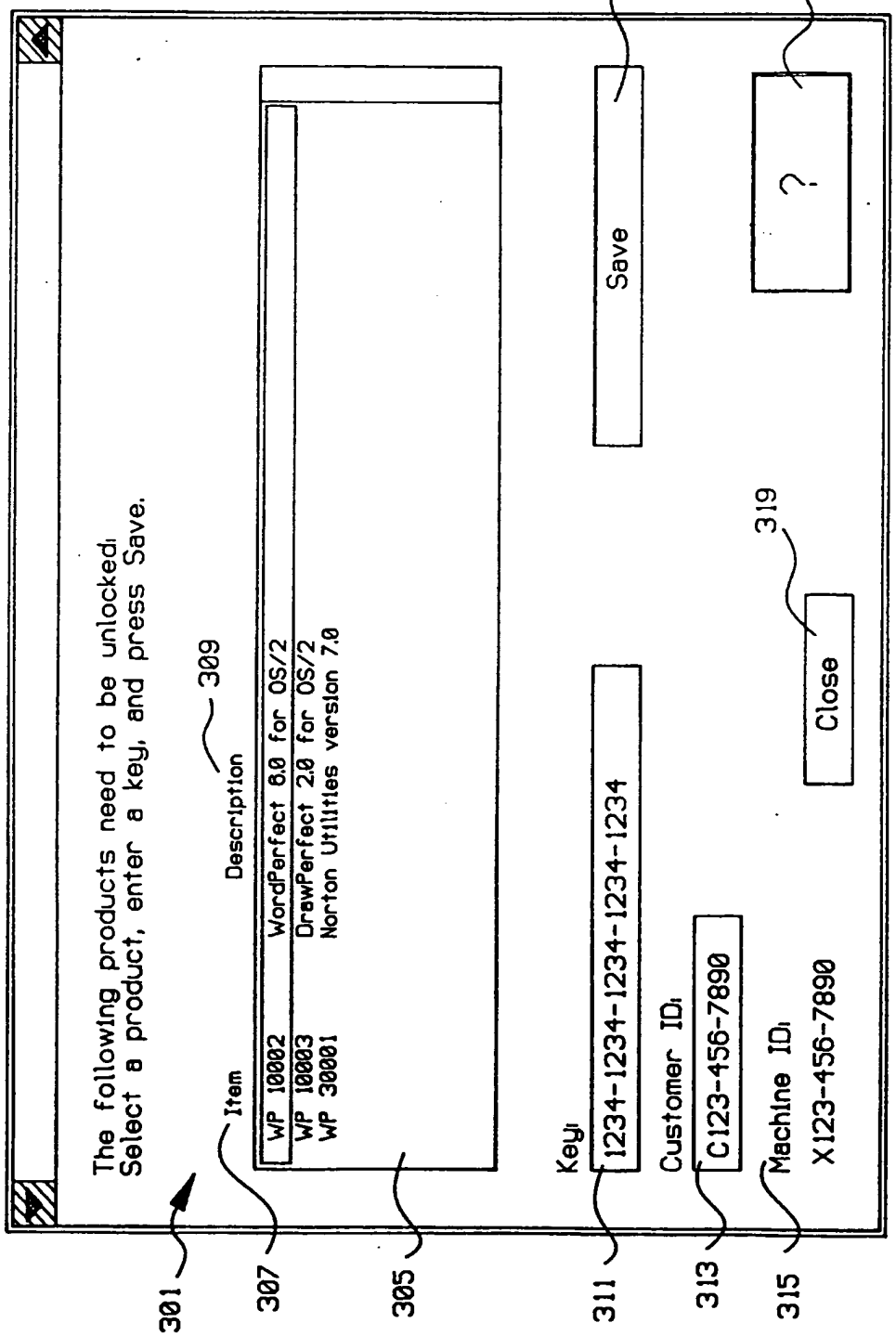


FIG. 10A

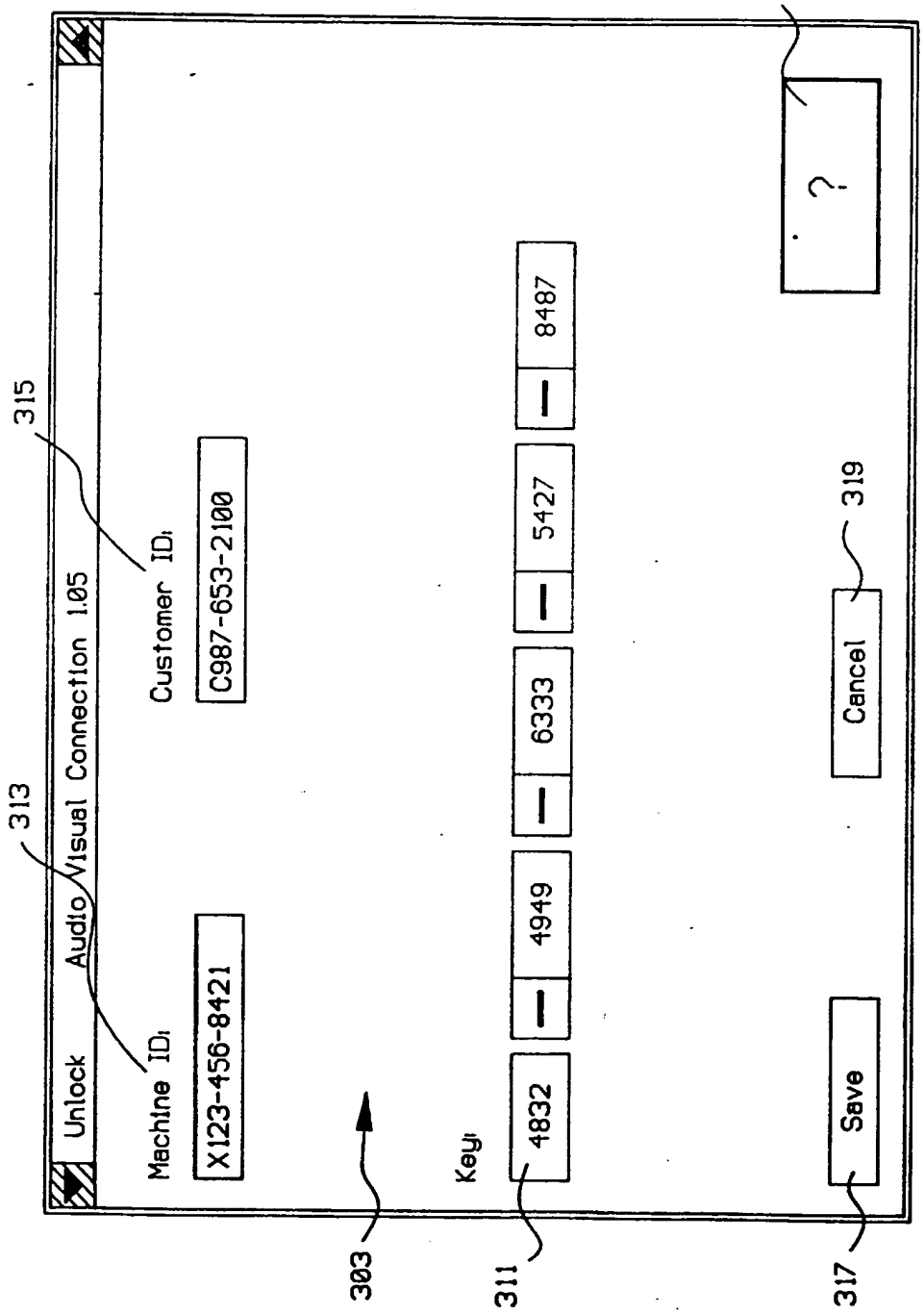


FIG. 10B

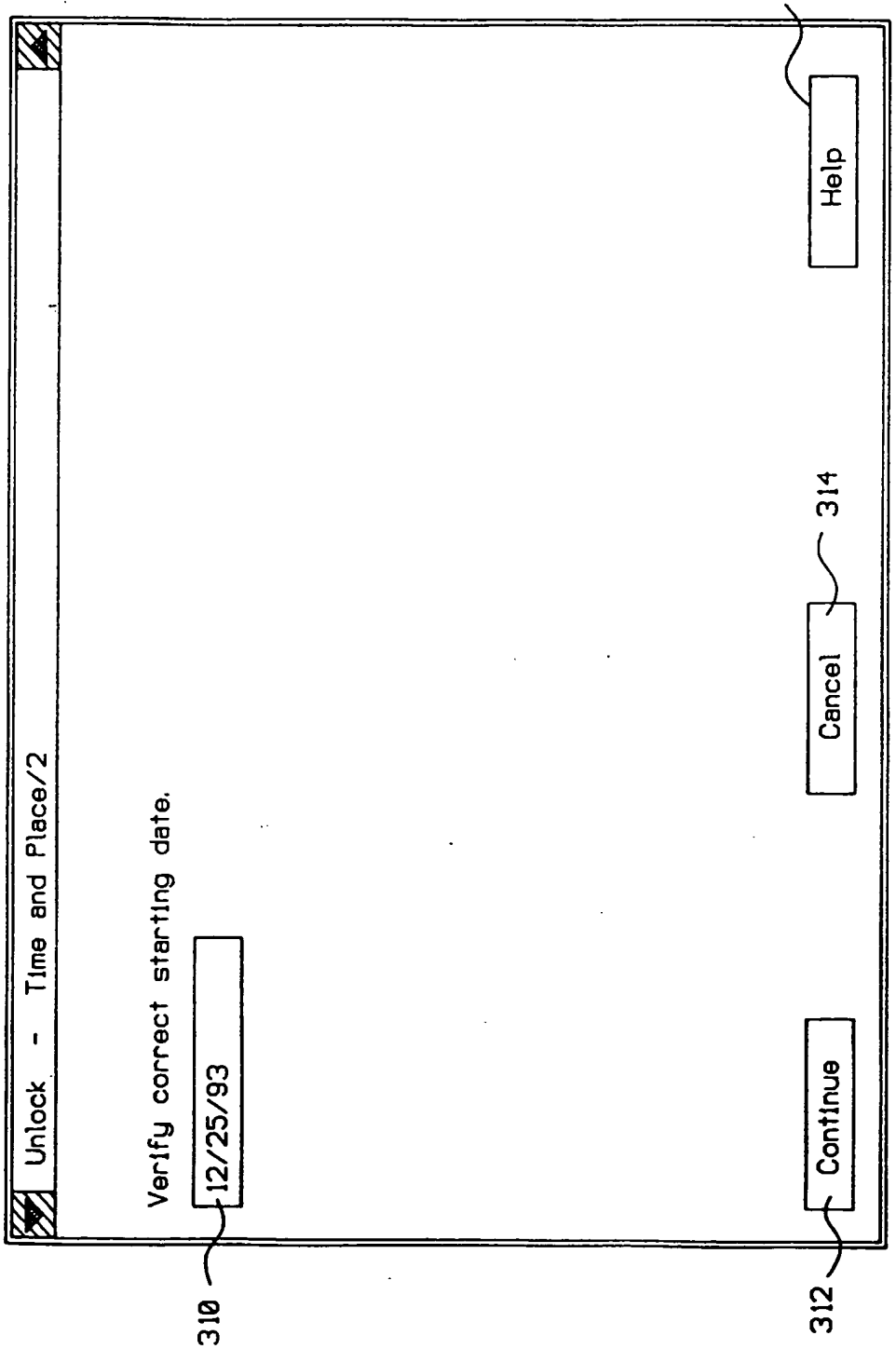


FIG. 11

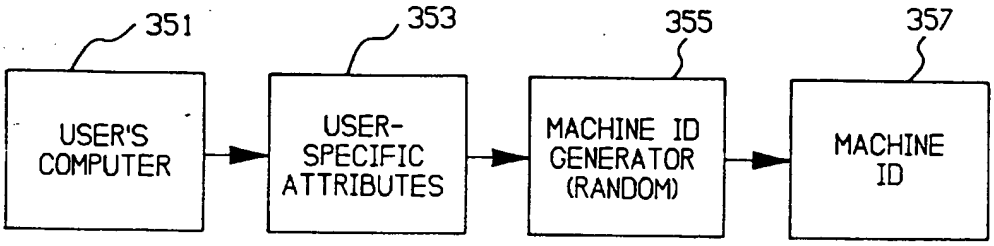
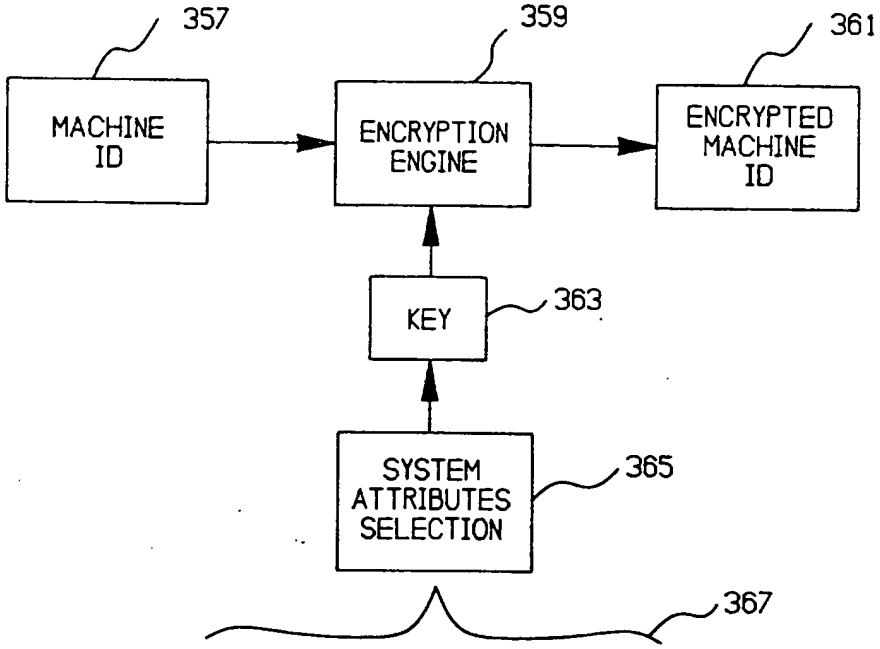
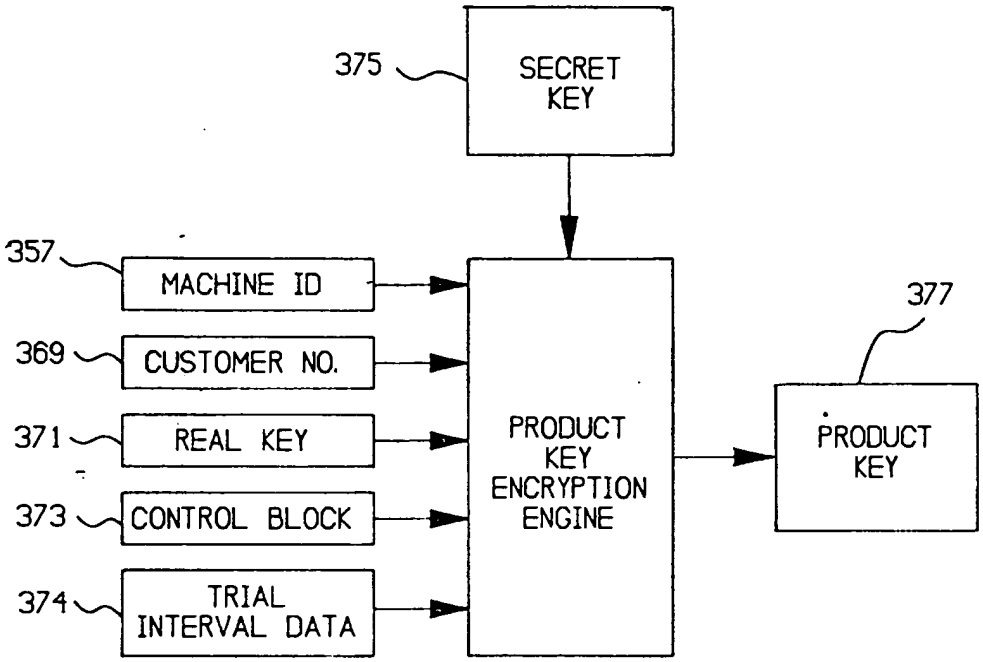


FIG. 12

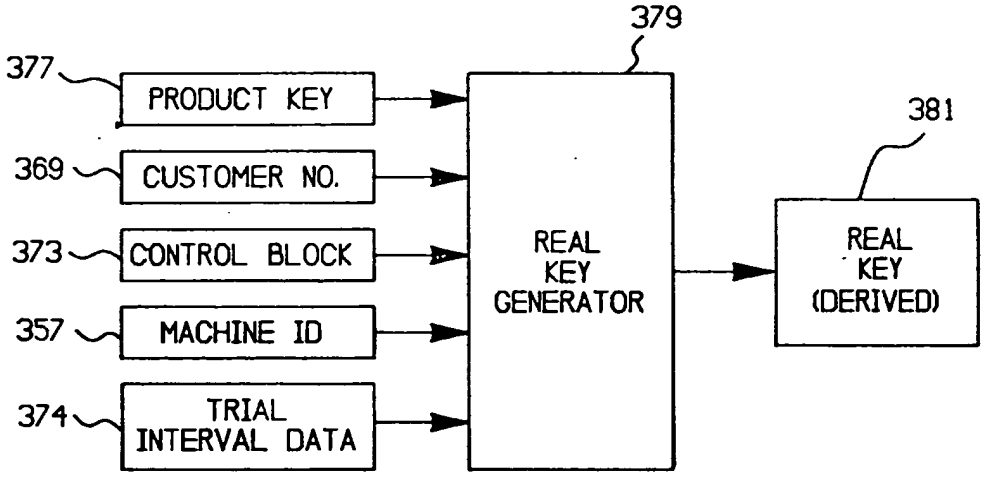


1. HARD DISK SERIAL NO.
2. SIZE OF HARD DISK
3. FORMAT OF HARD DISK
4. SYSTEM MODEL NO.
5. HARDWARE INTERFACE CARD
6. HARDWARE SERIAL NO.
7. CONFIGURATION PARAMETERS

FIG. 13



GENERATION OF PRODUCT KEY
FIG. 14



VALIDATION OF PRODUCT KEY
FIG. 15

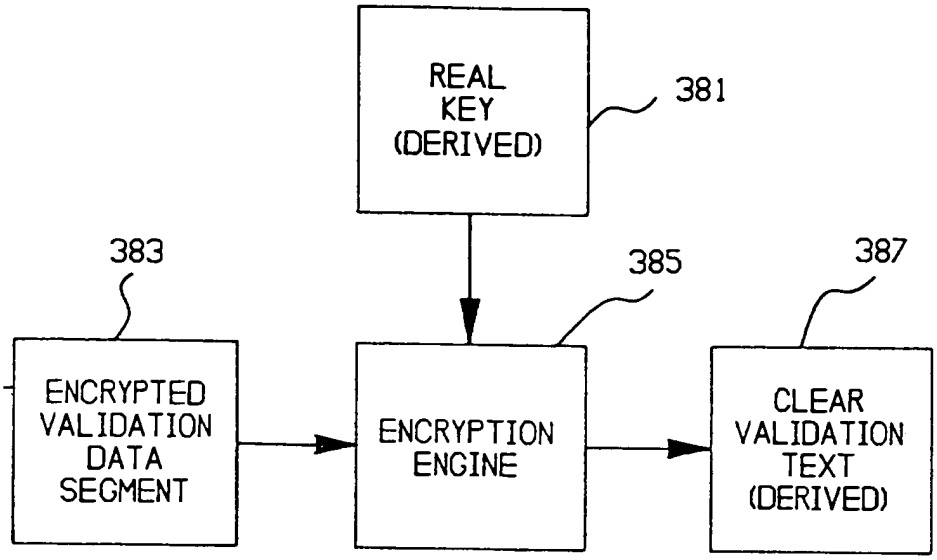


FIG. 16

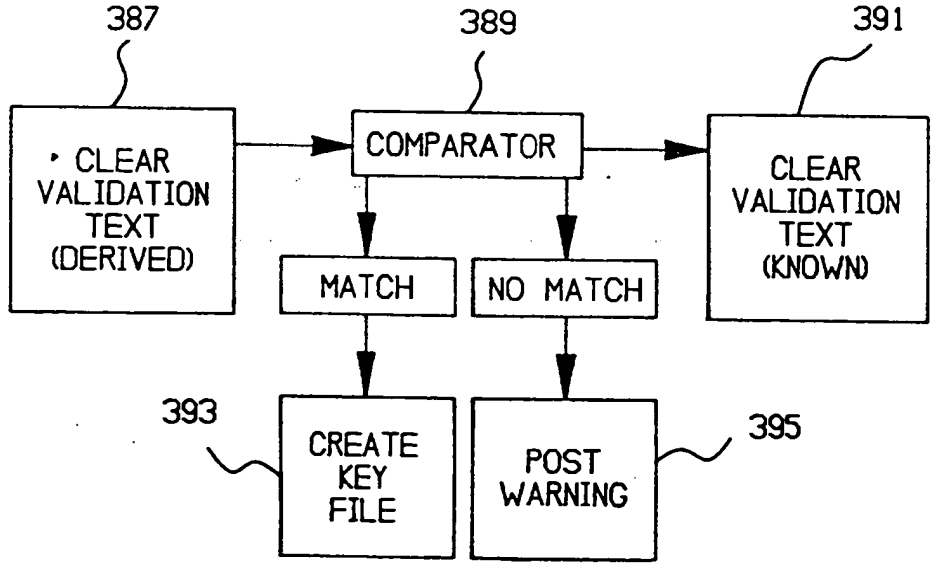


FIG. 17

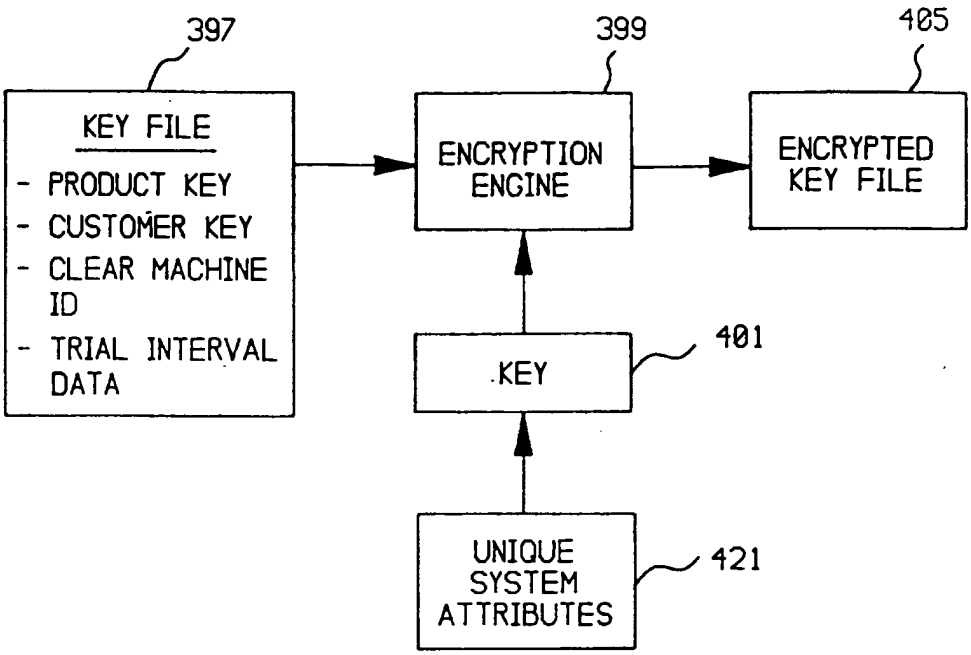


FIG. 18

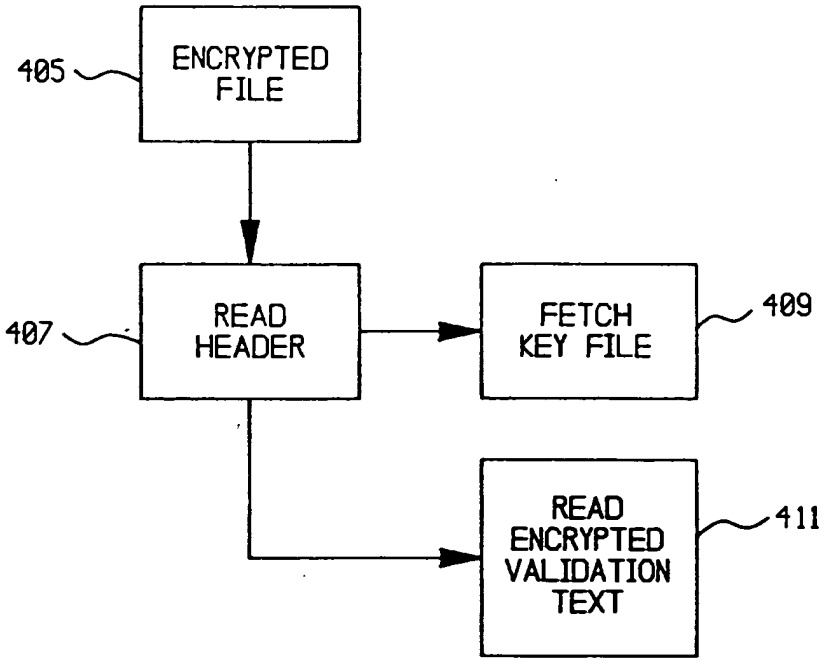


FIG. 19

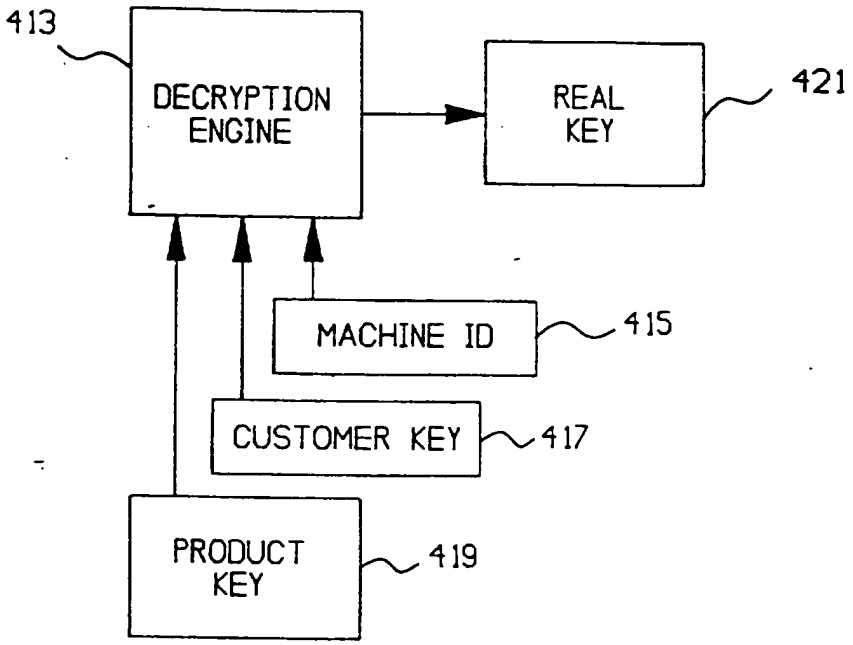


FIG. 20

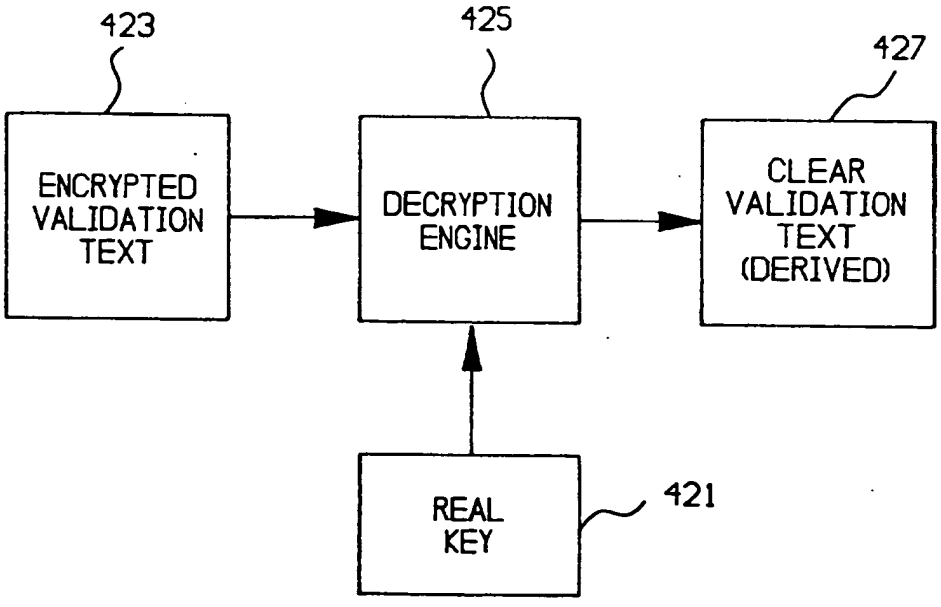


FIG. 21

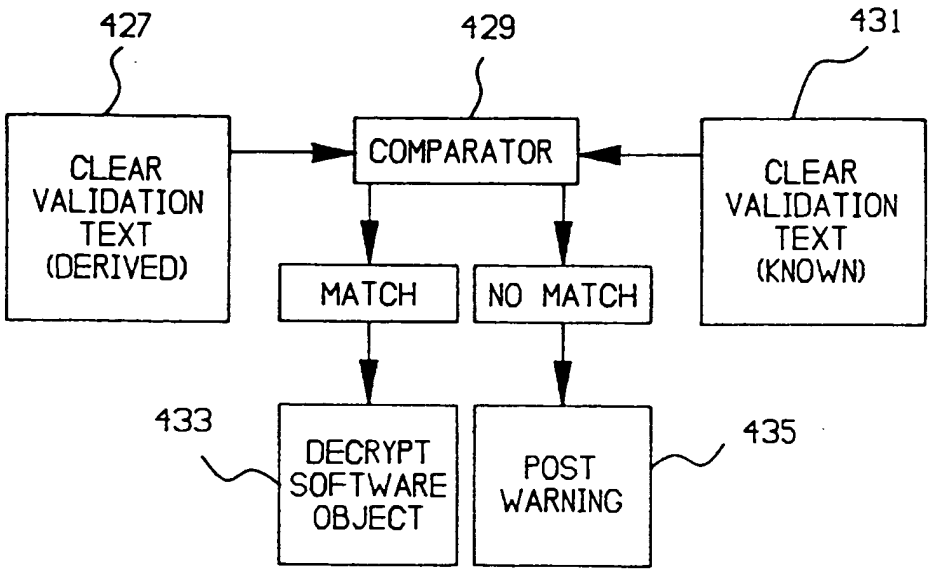


FIG. 22

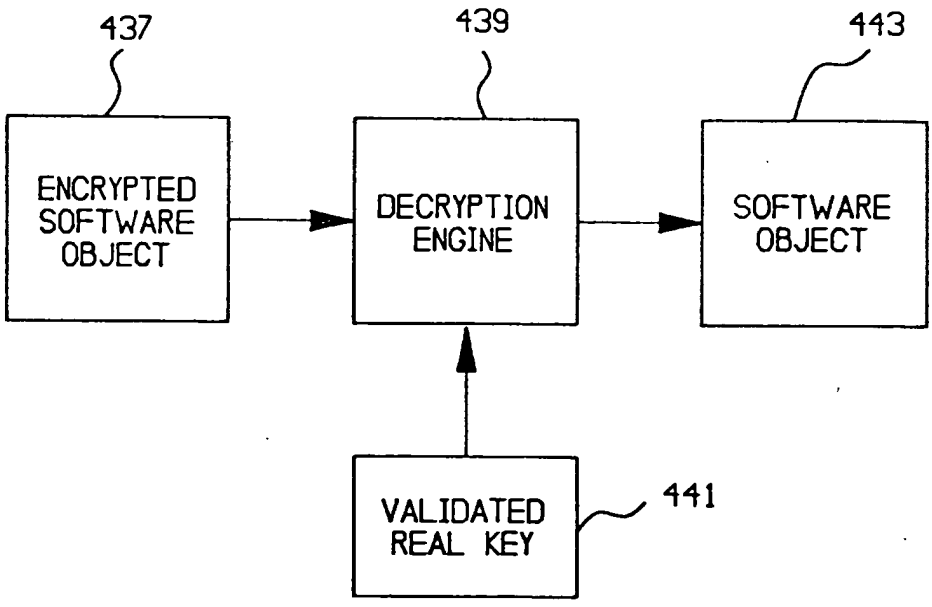
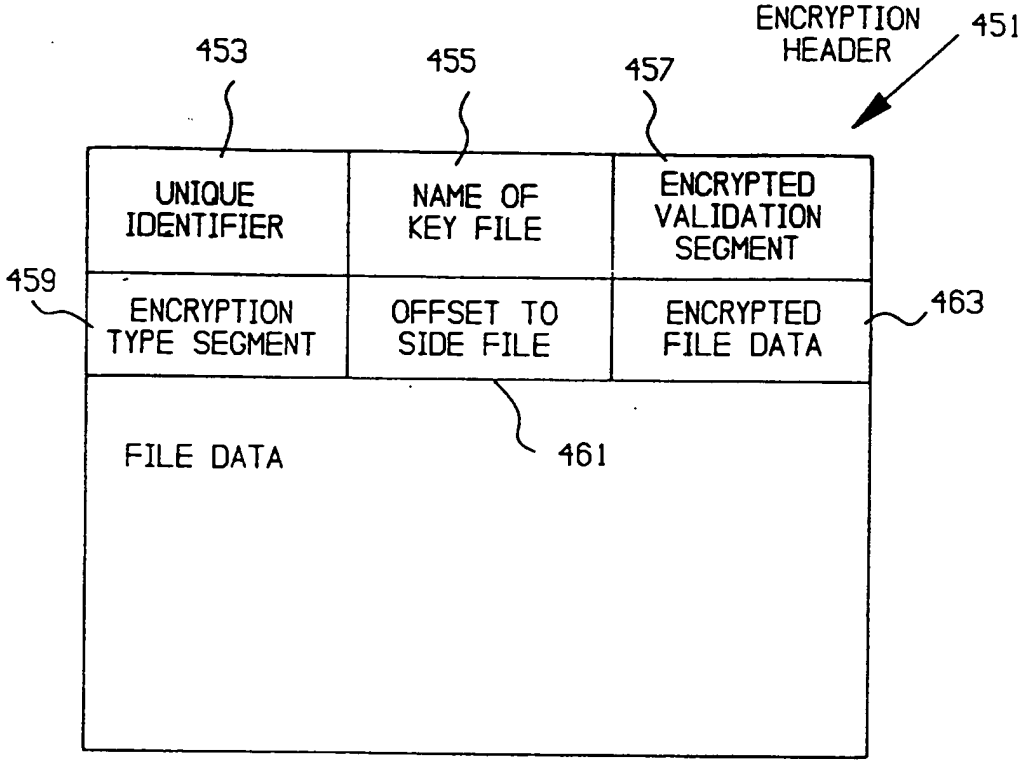


FIG. 23



ENCRYPTED FILE

FIG. 24

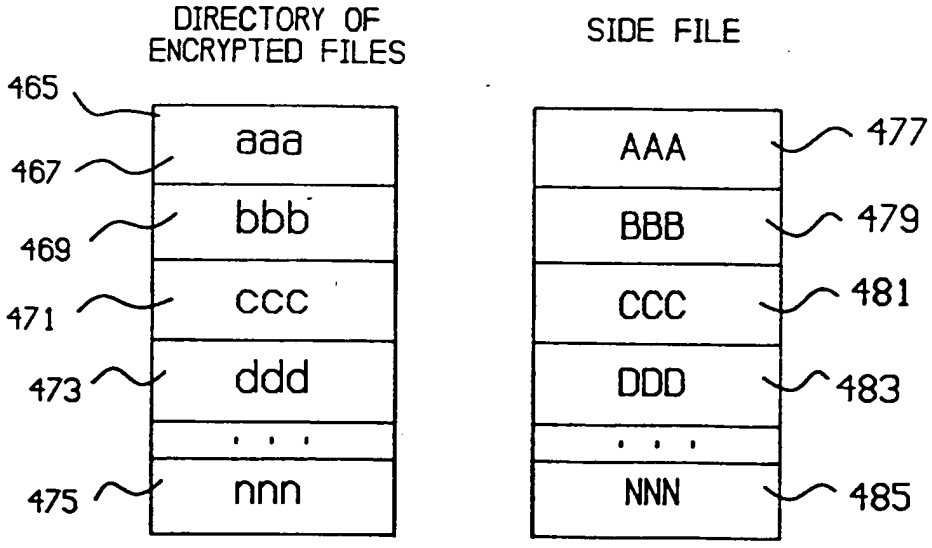


FIG. 25

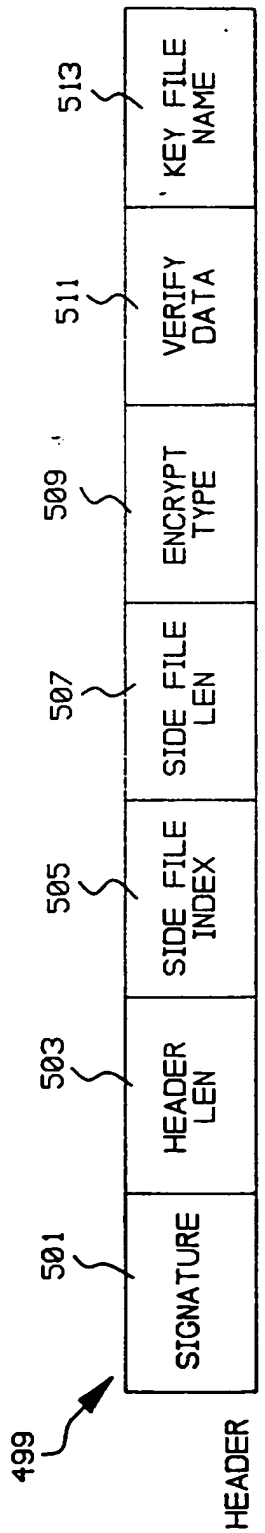


FIG. 26

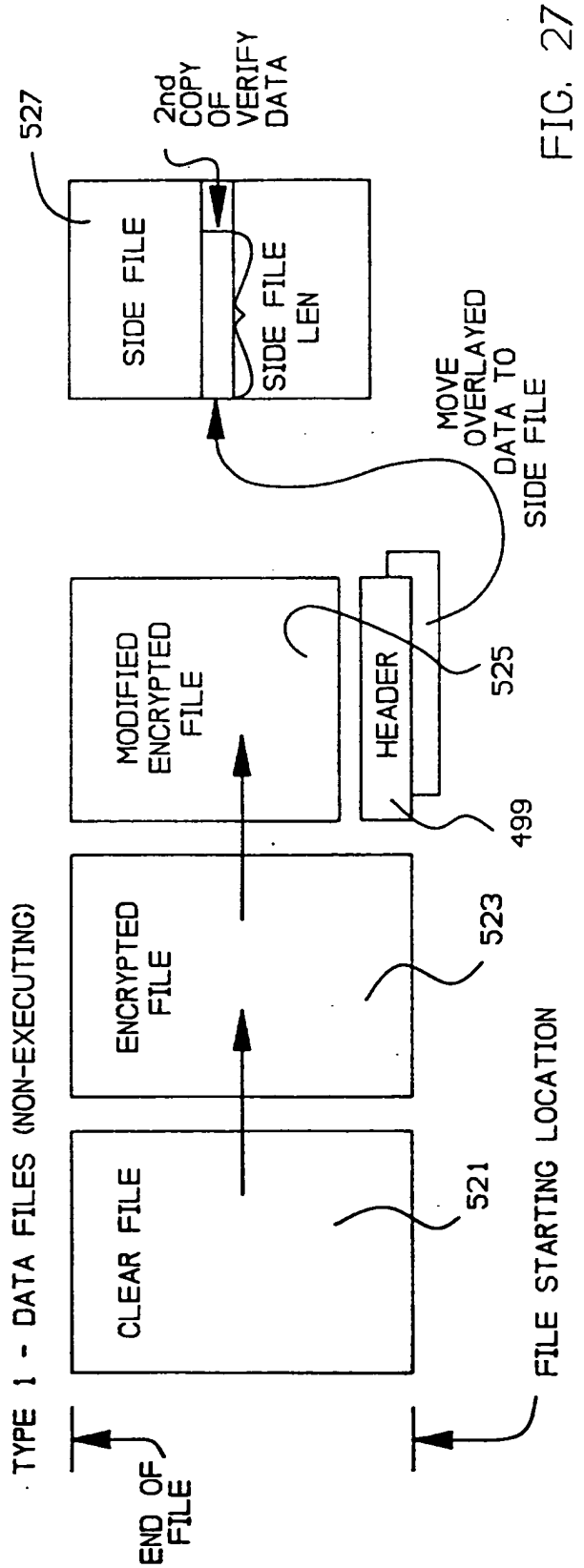


FIG. 27

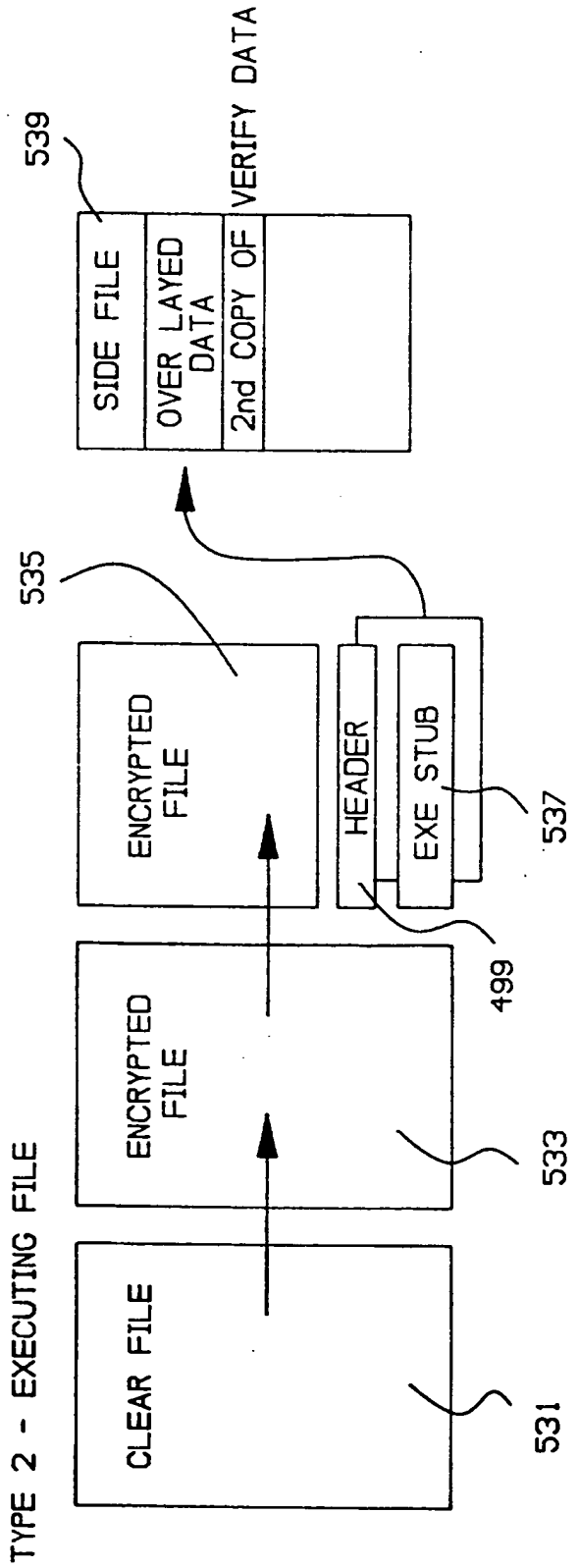


FIG. 28

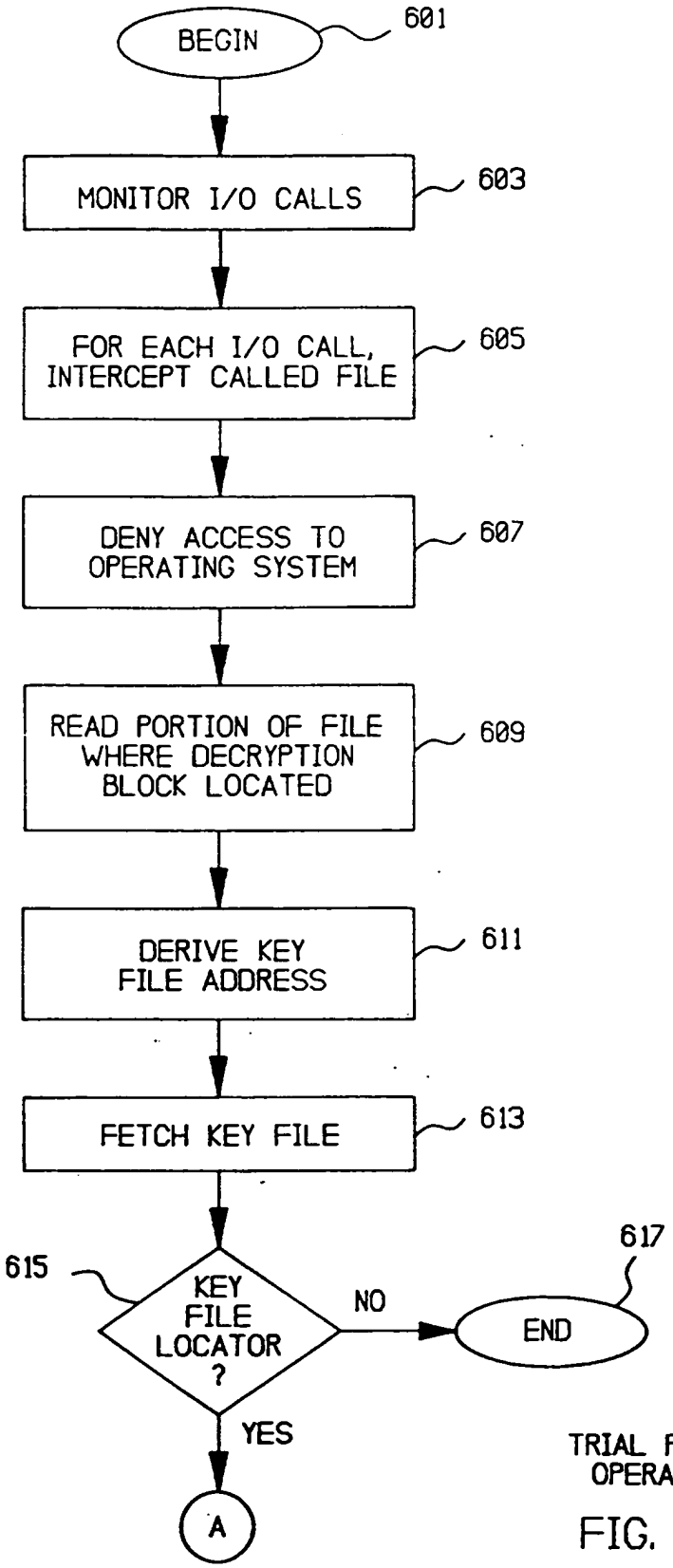
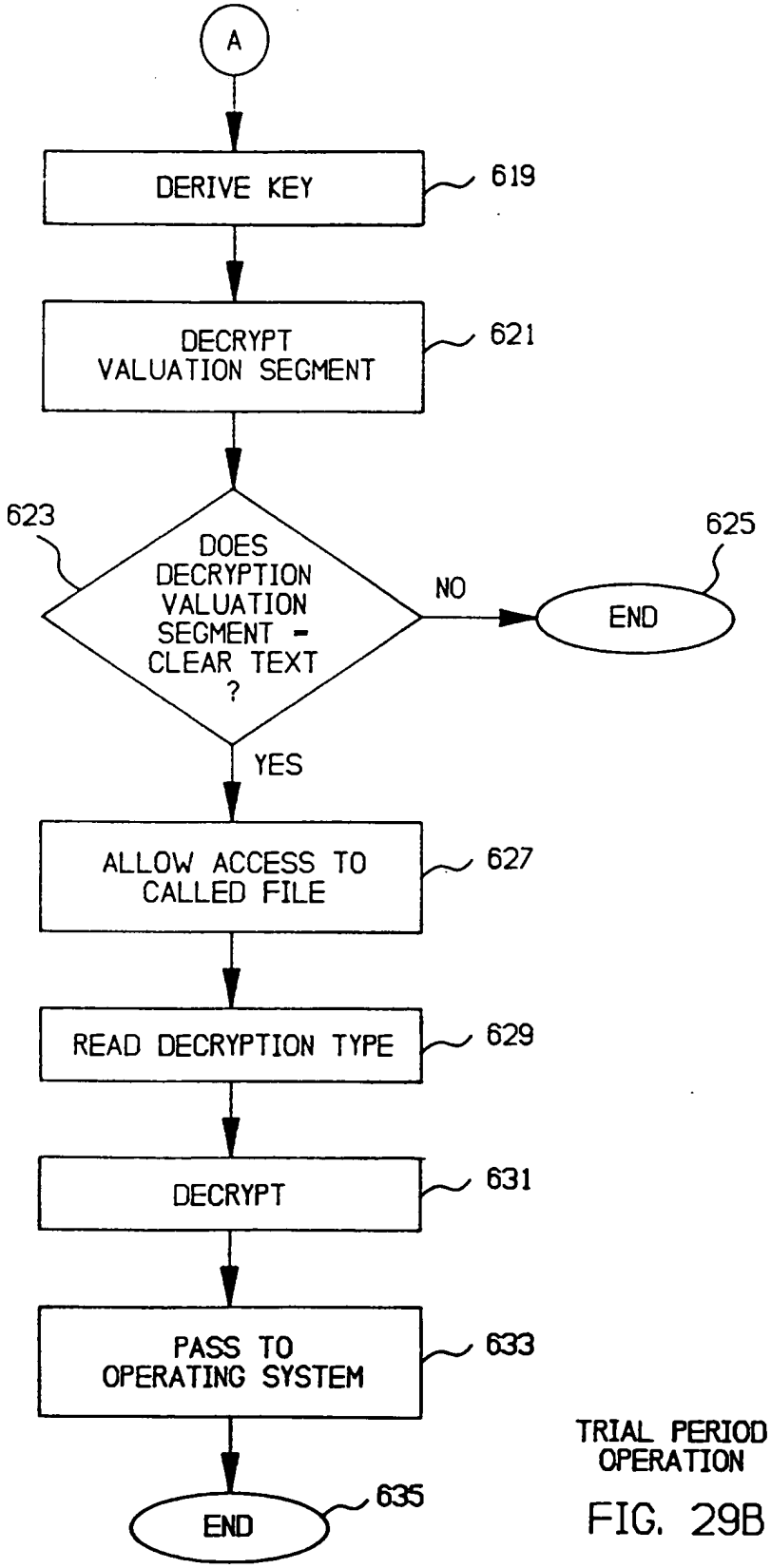
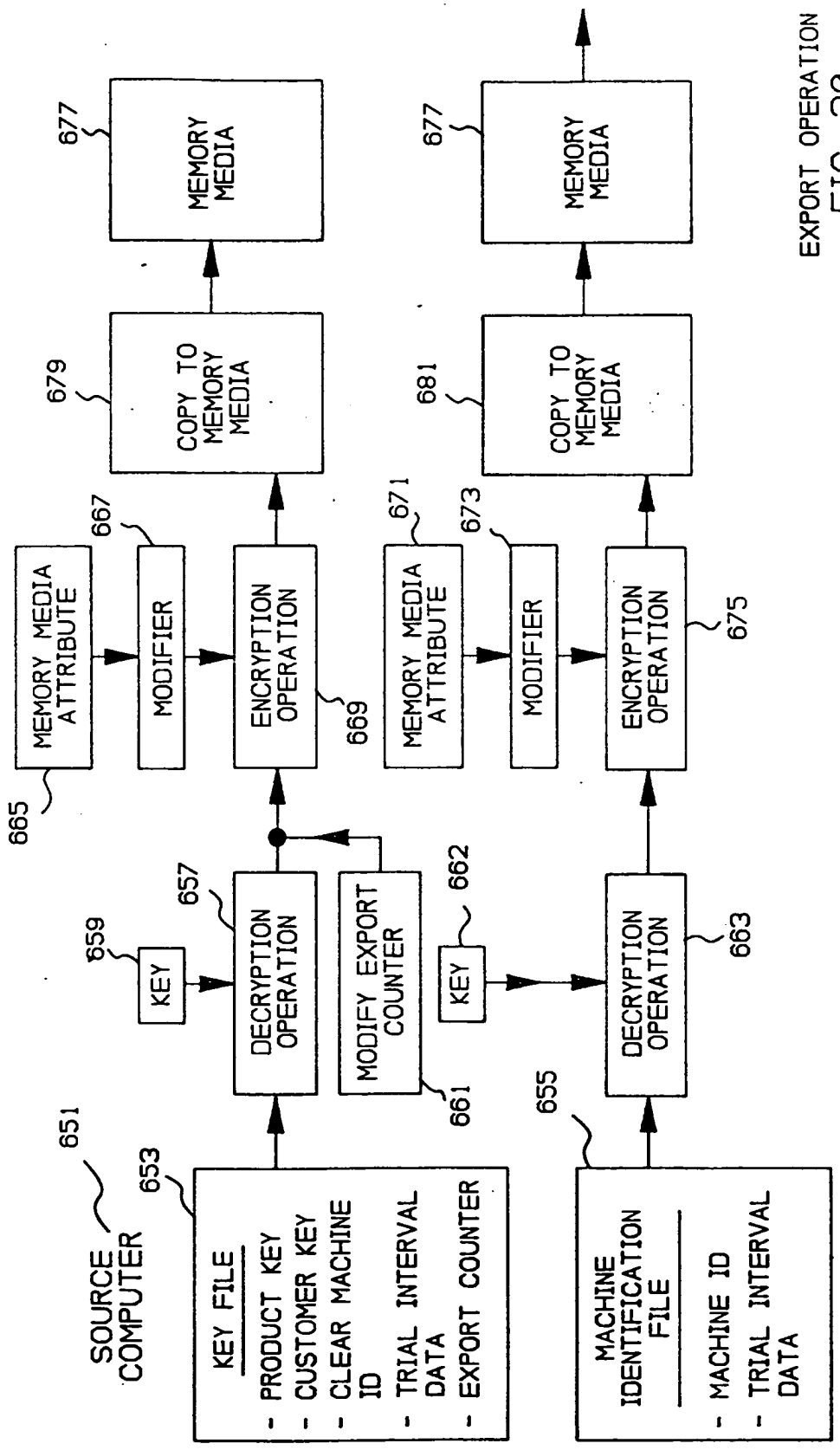


FIG. 29A



TRIAL PERIOD OPERATION

FIG. 29B



EXPORT OPERATION
FIG. 30

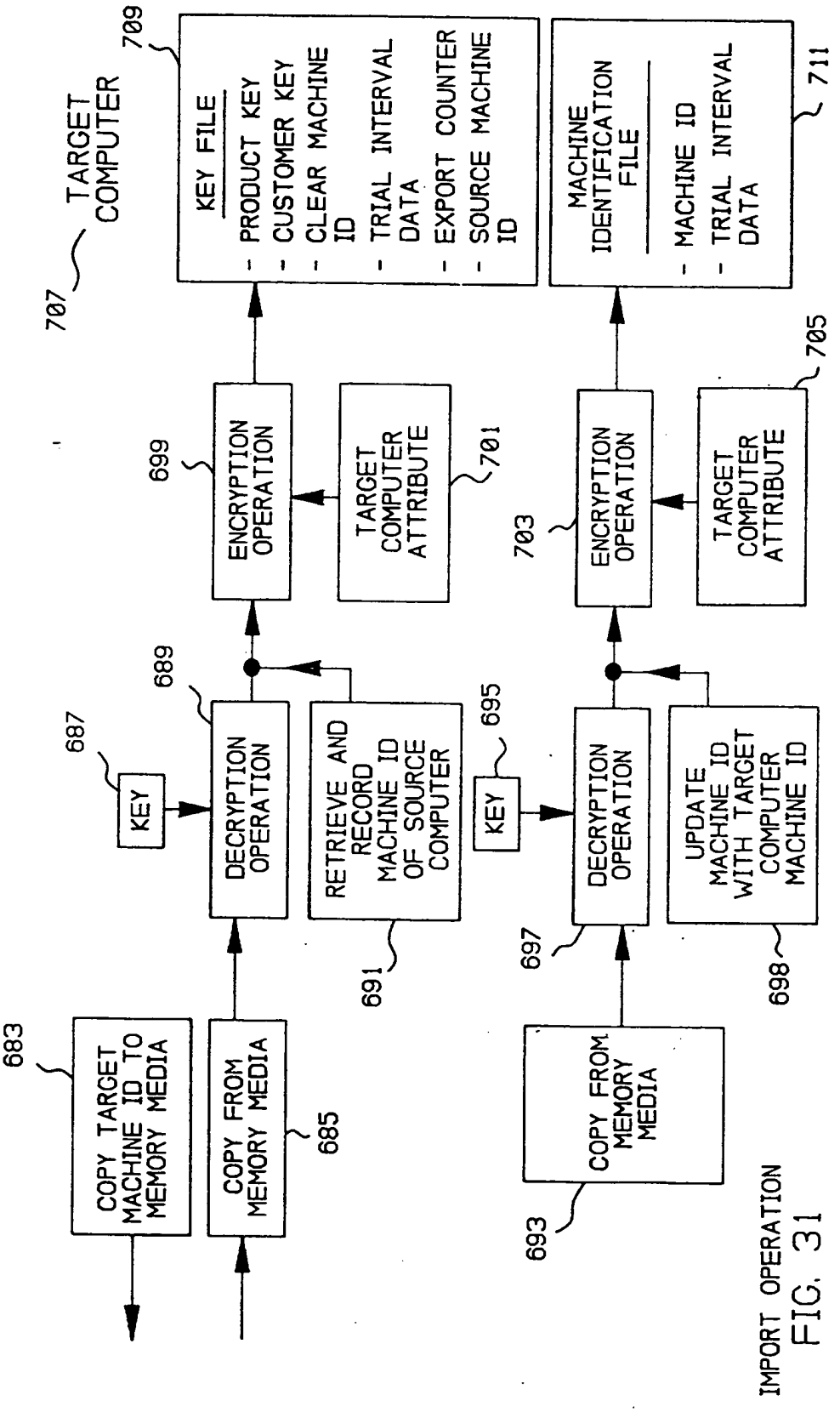


FIG. 31