The MAGIC Project: From Vision to Reality

Barbara Fuller, Mitretek Systems
Ira Richer, Corporation for National Research Initiatives

Abstract

In the MAGIC project, three major components — an ATM internetwork, a distributed, network-based storage system, and a terrain visualization application — were designed, implemented, and integrated to create a testbed for demonstrating real-time, interactive exchange of data at high speeds among distributed resources. The testbed was developed as a system, with special consideration to how performance was affected by interactions among the components. This article presents an overview of the project, with emphasis on the challenges associated with implementing a complex distributed system, and with coordinating a multi-organization collaborative project that relied on distributed development. System-level design issues and performance measurements are described, as is a tool that was developed for analyzing performance and diagnosing problems in a distributed system. The management challenges that were encountered and some of the lessons learned during the course of the three-year project are discussed, and a brief summary of MAGIC-II, a recently initiated follow-on project, is given.

Gigabit-per-second networks offer the promise of a major advance in computing and communications: high-speed access to remote resources, including archives, time-critical data sources, and processing power. Over the past six years, there have been several efforts to develop gigabit networks and to demonstrate their utility, the most notable being the five testbeds that were supported by ARPA and National Science Foundation (NSF) funding: Aurora, BLANCA, CASA, Nectar, and VISTAnet [1]. Each of these testbeds comprised a mix of applications and networking technology, with some focusing more heavily on applications and others on networking. The groundbreaking work done in these testbeds had a significant impact on the development of high-speed networking technology and on the rapid progress in this area in the 1990s.

It became clear, however, that a new paradigm for application development was needed in order to realize the full benefits of gigabit networks. Specifically, network-based applications and their supporting resources. such as data servers, must be designed explicitly to operate effectively in a high-speed networking environment. For example, an interactive application working with remote storage devices must compensate for network delays. The MAGIC project, which is the subject of this article, is the first high-speed networking testbed that was implemented according to this paradigm. The major components of the testbed were considered to be interdependent parts of a system, and wherever possible they were designed to optimize end-to-end system performance rather than individual component performance.

The objective of the MAGIC (which stands for "Multidimensional Applications and Gigabit Internetwork Consortium") project was to build a testbed that could demonstrate real-time, interactive exchange of data at gigabit-per-second rates among multiple distributed resources. This objective was pursued through a multidisciplinary effort involving concurrent development and subsequent integration of three testbed components:

- An innovative terrain visualization application that requires massive amounts of remotely stored data
- A distributed image server system with performance sufficient to support the terrain visualization application
- A standards-based high-speed internetwork to link the computing resources required for real-time rendering of the terrain

The three-year project began in mid-1992 and involved the participation, support, and close cooperation of many diverse organizations from government, industry, and academia. These organizations had complementary skills and had the foresight to recognize the benefits of collaboration. The principal MAGIC research participants were:

- Earth Resources Observation System Data Center, U.S. Geological Survey (EDC)[1]
- Lawrence Berkeley National Laboratory, U.S. Department of Energy (LBNL)[1]
- Minnesota Supercomputer Center, Inc. (MSCI)[1]
- MITRE Corporation[1]
- Sprint
- SRI International (SRI)[1]

The work reported here was performed while the authors were with the MITRE Corp. in Bedford, MA, and was supported by the Advanced Research Project Agency (ARPA) under contract F19628-94-D-001.

[1] *These organizations were funded by ARPA.*

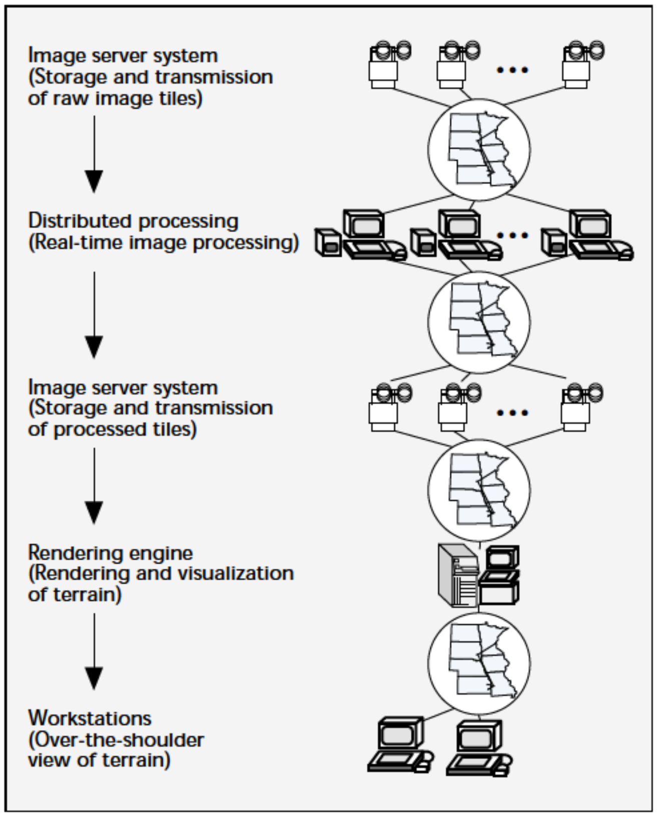


Image server system
(Storage and transmission of raw image tiles)
Distributed processing
(Real-time image processing)
Image server system
(Storage and transmission of processed tiles)
Rendering engine
(Rendering and visualization of terrain)
Workstations
(Over-the-shoulder view of terrain)

■ Figure 1. *Planned functionality of the MAGIC testbed.*

• University of Kansas (KU)[1]
• U S WEST Communications, Inc.

Other MAGIC participants that contributed equipment, facilities, and/or personnel to the effort were:

• Army High-Performance Computing Research Center (AHPCRC)
• Battle Command Battle Laboratory, U.S. Army Combined Arms Command (BCBL)
• Digital Equipment Corporation (DEC)
• Nortel, Inc./Bell Northern Research
• Southwestern Bell Telephone
• Splitrock Telecom

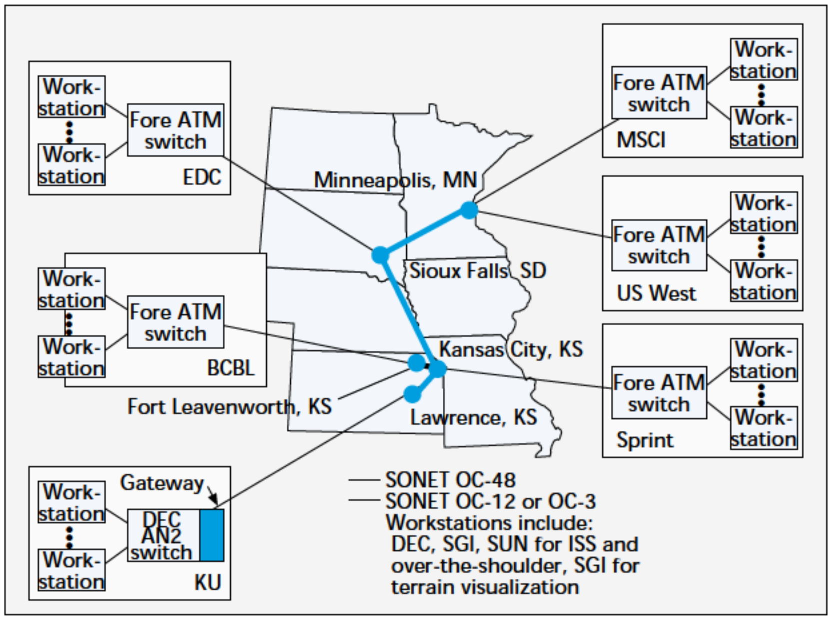
This article presents an overview of the MAGIC project with emphasis on the challenges associated with implementing a complex distributed system. Companion articles [2, 3] focus on a LAN/WAN gateway and a performance analysis tool that were developed for the MAGIC testbed. The article is organized as follows. The following section briefly describes the three major testbed components: the internetwork, the image server system, and the application. The third section discusses some of the system-level considerations that were addressed in designing these components, and the fourth section presents some high-level performance measurements. The fifth (affectionately entitled "Herding Cats") and sixth sections describe how this multi-organizational collaborative project was coordinated, and the technical and managerial lessons learned. Finally, the last section provides a brief summary of MAGIC-II, a follow-on project begun in early 1996.

Overview of the MAGIC Testbed

One of the primary goals of the MAGIC project was to create a testbed to demonstrate advanced capabilities that would not be possible without a very high-speed internetwork. MAGIC accomplished this goal by implementing an interactive terrain visualization application, TerraVision, that relies on a distributed image server system (ISS) to provide it with massive amounts of data in real time. The planned functionality of the MAGIC testbed is depicted in Fig. 1. Currently, TerraVision uses data processed off-line and stored on the ISS. In the future the application will be redesigned to enable real-time image processing as well as real-time terrain visualization (see the last section). Note that the workstations which house the application, the servers of the ISS, and the "over-the-shoulder" tool (see subsection entitled "The Terrain Visualization Application"), as well as those that will perform the on-line image processing, can reside anywhere on the network.

The MAGIC Internetwork

The MAGIC internetwork, depicted in Fig. 2, includes six high-speed local area networks (LANs) interconnected by a wide area network (WAN) backbone. The backbone, which spans a distance of approximately 600 miles, is based on synchronous optical network (SONET) technology and provides OC-48 (2.4 Gb/s) trunks, and OC-3 (155 Mb/s) and OC-12 (622 Mb/s) access ports. The LANs are based on asynchronous transfer mode (ATM) technology. Five of the LANs — those at BCBL in Fort Leavenworth, Kansas, EDC in Sioux Falls, South Dakota, MSCI in Minneapolis, Minnesota, Sprint in Overland Park, Kansas, and U S WEST in Minneapolis, Minnesota — use FORE Systems models ASX-100 and ASX-200 switches with OC-3c and 100 Mb/s TAXI interfaces. The ATM LAN at KU in Lawrence, Kansas, uses a DEC AN2 switch, a precursor to the DEC GigaSwitch/ATM, with OC-3c interfaces. The network uses permanent virtual circuits (PVCs) as well as switched virtual circuits (SVCs) based on both SPANS, a FORE Systems signaling protocol, and the ATM Forum User-Network Interface (UNI) 3.0 Q.2931 signaling stan-



■ Figure 2. *Configuration of the MAGIC ATM internetwork.*

dard. The workstations at the MAGIC sites include models from DEC, SGI, and Sun. As part of MAGIC, an AN2/SONET gateway with an OC-12c interface was developed to link the AN2 LAN at KU to the MAGIC backbone [2].

In addition to implementing the internetwork, a variety of advanced networking technologies were developed and studied under MAGIC. A high-performance parallel interface (HIPPI)/ATM gateway was developed to interface an existing HIPPI network at MSCI to the MAGIC backbone. The gateway is an IP router rather than a network-layer device such as a broadband integrated services digital network (B-ISDN) terminal adapter, and was implemented in software on a high-performance workstation (an SGI Challenge). This architecture provides a programmable platform that can be modified for network research, and in the future can readily take advantage of more powerful workstation hardware. In addition, the platform is general-purpose; that is, it is capable of supporting multiple HIPPI interfaces as well as other interfaces such as fiber distributed data interface (FDDI).

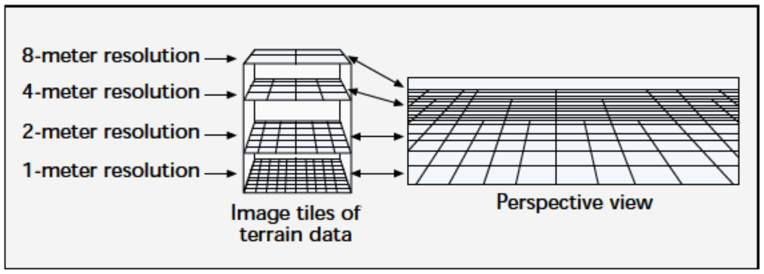
Software was developed to enable UNIX hosts to communicate using Internet Protocol (IP) over an ATM network. This IP/ATM software currently runs on SPARCstations under Sun OS 4.1 and includes a device driver for the FORE SBA series of ATM adapters. It supports PVCs, SPANS, and UNI 3.0 signaling, as well as the "classical" IP and Address Resolution Protocol (ARP) over ATM model [4]. The software should be extensible to other UNIX operating systems, ATM interfaces, and IP/ATM address-resolution and routing strategies, and will facilitate research on issues associated with the integration of ATM networks into IP internets.

In order to enhance network throughput, flow-control schemes were evaluated and applied, and IP/ATM host parameters were tuned. Experiments showed that throughput close to the maximum theoretically possible could be attained on OC-3 links over long distances. To achieve high throughput, both the maximum transmission unit (MTU) and the Transmission Control Protocol (TCP) window must be large, and flow control must be used to ensure fairness and to avoid cell loss if there are interacting traffic patterns [5, 6].

The Terrain Visualization Application

TerraVision allows a user to view and navigate through (i.e., "fly over") a representation of a landscape created from aerial or satellite imagery [7]. The data used by TerraVision are derived from raw imagery and elevation information which have been preprocessed by a companion application known as TerraForm. TerraVision requires very large amounts of data in real time, transferred at both very bursty and high steady rates. Steady traffic occurs when a user moves smoothly through the terrain, whereas bursty traffic occurs when the user jumps ("teleports") to a new position. TerraVision is designed to use imagery data that are located remotely and supplied to the application as needed by means of a high-speed network. This design enables TerraVision to provide high-quality, interactive visualization of very large data sets in real time. TerraVision is of direct interest to a variety of organizations, including the Department of Defense. For example, the ability of a military officer to see a battlefield and to share a common view with others can be very effective for command and control.

Terrain visualization with TerraVision involves two activities: generating the digital data set required by the application, and rendering the image. MAGIC's approach to accomplishing these activities is described below. Enhancements to the application that provide additional features and capabilities are also described.



■ Figure 3. *Relationship between tile resolutions and perspective view. (Source: SRI International)*

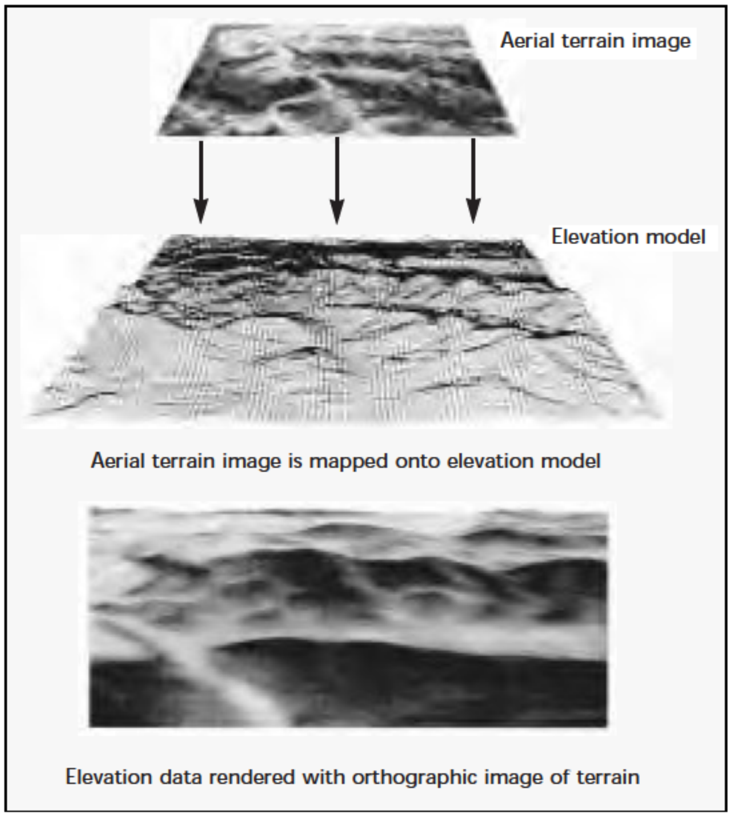
Data Preparation — In order to render an image, TerraVision requires a digital description of the shape and appearance of the subject terrain. The shape of the terrain is represented by a two-dimensional grid of elevation values known as a *digital elevation model* (DEM). The appearance of the terrain is represented by a set of aerial images, known as *orthographic projection images* (ortho-images), that have been specially processed (i.e., ortho-rectified) to eliminate the effects of perspective distortion, and are in precise alignment with the DEM. To facilitate processing, distributed storage, and high-speed retrieval over a network, the DEM and images are divided into small fixed-size units known as *tiles*.

Low-resolution tiles are required for terrain that is distant from the viewpoint, whereas high-resolution tiles are required for close-in terrain. In addition, multiple resolutions are required to achieve perspective. These requirements are addressed by preparing a hierarchy of increasingly lower-resolution representations of the DEM and ortho-image tiles in which each level is at half the resolution of the previous level. The tiled, multiresolution hierarchy and the use of multiple resolutions to achieve perspective are shown in Fig. 3.

Rendering of the terrain on the screen is accomplished by combining the DEM and ortho-image tiles for the selected area at the appropriate resolution. As the user travels over the terrain, the DEM tiles and their corresponding ortho-image tiles are projected onto the screen using a perspective transform whose parameters are determined by factors such as the user's viewpoint and field of view. The mapping of a transformed ortho-image to its DEM and the rendering of that image are shown in Fig. 4.

The data set currently used in MAGIC covers a 1200 km^2 exercise area of the National Training Center at Fort Irwin, California, and is about 1 Gpixel in size. It is derived from aerial photographs obtained from the National Aerial Photography Program archives and DEM data obtained from the U.S. Geological Survey. The images are at approximately 1 m resolution (i.e., the spacing between pixels in the image corresponds to 1 m on the ground). The DEM data are at approximately 30 m resolution (i.e., elevation values in meters are at 30 m intervals).

Software for producing the ortho-images and creating the multiresolution hierarchy of DEM and ortho-image tiles was developed as part of the MAGIC effort. These processes were performed "off-line" on a Thinking Machines Corporation Connection Machine (CM-5) supercomputer owned by the AHPCRC and located at MSCI. The tiles were then stored on the distributed servers of the ISS and used by terrain visualization software residing on rendering engines at several locations.


Aerial terrain image
Elevation model
Aerial terrain image is mapped onto elevation model
Elevation data rendered with orthographic image of terrain

■ **Figure 4.** *Mapping an ortho-image onto its digital elevation model. (Source: SRI International)*

Image Rendering — TerraVision provides for two modes of visualization: two-dimensional (2-D) and three-dimensional (3-D). The 2-D mode allows the user to fly over the terrain, looking only straight down. The user controls the view by means of a 2-D input device such as a mouse. Since virtually no processing is required, the speed at which images are generated is limited by the throughput of the system comprising the ISS, the network, and the rendering engine.

In the 3-D mode, the user controls the visualization by means of an input device that allows six degrees of freedom in movement. The 3-D mode is computationally intensive, and satisfactory visualization requires both high frame rates (i.e., 15–30 frames/s) and low latencies (i.e., no more than 0.1 s between the time the user moves an input device and the time the new frame appears on the screen).

High frame rates are achieved by using a local very-high-speed rendering engine, an SGI Onyx, with a cache of tiles covering not only the area currently visible to the user, but also adjacent areas that are likely to be visible in the near future. A high-speed search algorithm is used to identify the tiles required to render a given view. For example, as noted above, perspective (i.e., 3-D) views require higher-resolution tiles in the foreground and lower-resolution tiles in the background. TerraVision requests the tiles from the ISS, places them in memory, and renders the view. Latency is minimized by separating image rendering from data input/output (I/O) so that the two activities can proceed simultaneously rather than sequentially (see the section entitled "Design Considerations").

Additional Features and Capabilities — TerraVision includes two additional features: superposition of fixed and mobile objects on the terrain, and registration of the user's viewpoint to a map. Both of these features are made possible by precisely aligning the DEM and imagery data with a world coordinate system as well as with each other.

A number of buildings and vehicles have been created and stored on the rendering engine for display as an overlay on the terrain. The locations of vehicles can be updated periodically by transferring vehicle location data, acquired with a global positioning system receiver, to the rendering engine for integration into the terrain visualization displays. Registration of the user's viewpoint to a map enables the user to specify the area he wishes to explore by pointing to it, and it aids the user in orienting himself.

In addition, an over-the-shoulder (OTS) tool was developed to allow a user at a remote workstation to view the terrain as it is rendered. The OTS tool is based on a client/server design and uses XWindow system calls. The user can view the entire image on the SGI screen at low resolution, and can also select a portion of the screen to view at higher resolution. The frame rate varies with the size and resolution of the viewed image, and with the throughput of the workstation.

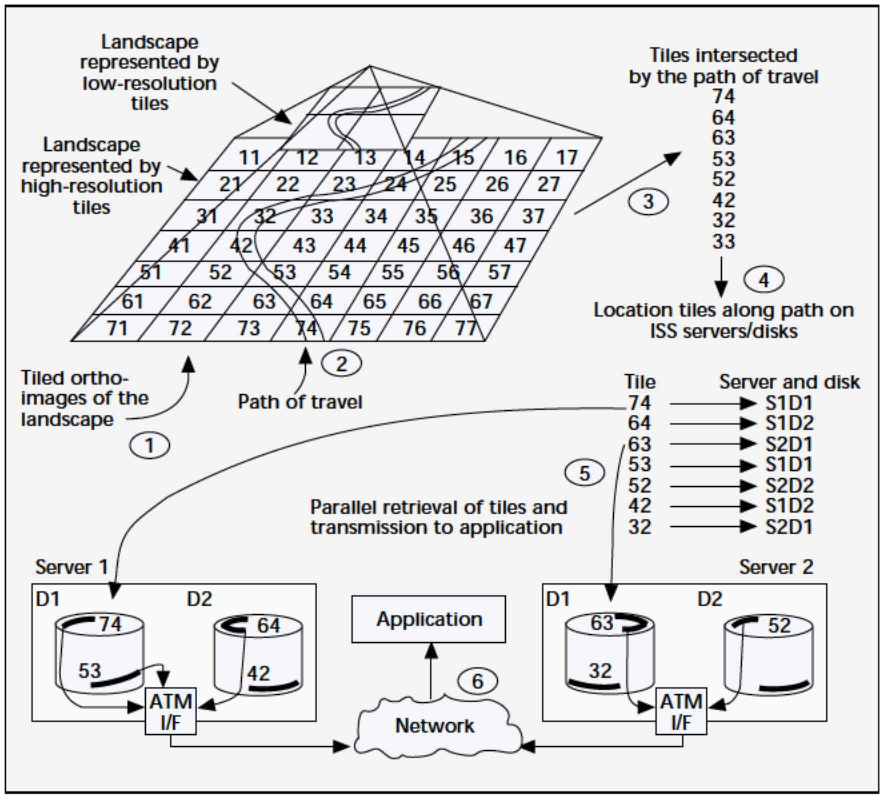
The Image Server System

The ISS stores, organizes, and retrieves the processed imagery and elevation data required by TerraVision for interactive rendering of the terrain. The ISS consists of multiple coordinated workstation-based data servers that operate in parallel and are designed to be distributed around a WAN. This architecture compensates for the performance limitations of current disk technology. A single disk can deliver data at a rate that is about an order of magnitude slower than that needed to support a high-performance application such as TerraVision. By using multiple workstations with multiple disks and a high-speed network, the ISS can deliver data at an aggregate rate sufficient to enable real-time rendering of the terrain. In addition, this architecture permits location-independent access to databases, allows for system scalability, and is low in cost. Although redundant arrays of inexpensive disks (RAID) systems can deliver higher throughput than traditional disks, unlike the ISS they are implemented in hardware and, as such, do not support multiple data layout strategies; furthermore, they are relatively expensive. Such systems are therefore not appropriate for distributed environments with numerous data repositories serving a variety of applications.

The ISS, as currently used in MAGIC, comprises four or five UNIX workstations (including Sun SPARCstations, DEC Alphas, and SGI Indigos), each with four to six fast SCSI disks on two to three SCSI host adapters. Each server is also equipped with either a SONET or a TAXI network interface. The servers, operating in parallel, access the tiles and send them over the network, which delivers the aggregate stream to the host. This process is illustrated in Fig. 5. More details about the design and operation of the ISS can be found in [8].

Design Considerations

In MAGIC, the single most perspicuous criterion of successful operation is that the end user observes satisfactory performance of the interactive TerraVision application. When the user flies over the terrain, the displayed scene must flow smoothly, and when he teleports to an entirely different location, the new scene must appear promptly. Obtaining such performance might be relatively straightforward if the terrain data were collocated with the rendering engine. However, one of the original premises underlying the MAGIC project is that the data set and the application are not collocated. There are several reasons for this, the most important being that the data set could be extremely large, so it might not be feasible to transfer it to the user's site. Moreover, experience has shown that in many cases the "owner" of a data set is also its "curator" and may be reluctant to distribute it, preferring instead to keep the data locally to simplify maintenance and updates. Finally, it was anticipated that future versions of the application might work with a mobile user and with fused data from multiple sources, and neither of these capabilities would be practical with local data. Therefore, since the data will not be local, the MAGIC components must be designed to compensate for possible delays and other degradations in the end-to-end operation of the system.



■ Figure 5. *Schematic representation of the operation of the ISS. (Source: Lawrence Berkeley National Laboratory)*

In order to understand system-level design issues, it is necessary to outline the sequence of events that occurs when the user moves the input device, causing a new scene to be generated. TerraVision first produces a list of new tiles required for the scene. This list is sent to an ISS master, which performs a name translation, mapping the logical address of each tile (the tile identifier) to its physical address (server/disk/location on disk). The master then sends each server an ordered list of the tiles it must retrieve. The server discards the previous list (even if it has not retrieved all the tiles on that list) and begins retrieving the tiles on the new list. Thus, the design for the system comprising TerraVision, the ISS, and the internetwork must address the following questions:

- How can TerraVision compensate for tiles it needs for the next image but have not yet been received?
- How often should TerraVision request tiles from the ISS?
- Where should the ISS master be located?
- How should tiles be distributed among the ISS disks?
- How can cell loss be minimized near the rendering site where the tile traffic becomes aggregated and congestion may occur?

Missing Tiles

Network congestion, an overload at an ISS server, or a component failure could result in the late arrival or loss of tiles that are requested by the application. Several mechanisms were implemented to deal with this problem. First, although the entire set of high-resolution tiles cannot be collocated with the application, it is certainly feasible to store a complete set of lower-resolution tiles. For example, if the entire data set comprises 1 Tbyte of high-resolution tiles, then all of the tiles that are five or more levels coarser would occupy less than 1.5 Mbyte, a readily affordable amount of local storage. If a tile with resolution at, say, level 3 is requested but not delivered in time for the image to be rendered, then, until the missing level-3 tile arrives, the locally available coarser tile from level 5 would be used in place of the 16 level-3 tiles. This substitution manifests itself by the affected portion of the rendered image appearing "fuzzy" for a brief period of time. Temporary substitution of low-resolution tiles for high-resolution tiles is particularly effective for teleporting because that operation requires a large number of new tiles, so it is more likely that one or more will be delayed.

Second, TerraVision attempts to predict the path the user will follow, requesting tiles that *might* soon be needed, and assigning one of three levels of priority to each tile requested. Priority-1 tiles are needed as soon as possible; the ISS retrieves and dispatches these first. This set of tiles is ordered by TerraVision, with the coarsest assigned the highest priority within the set. The reasons are:

- The rendering algorithm needs the coarse tiles before it needs the next-higher-resolution tiles.
- There are fewer tiles at the coarser resolutions, so it is less likely that they will be delayed.

The priority-2 tiles are those that the ISS should retrieve but should transmit only if there are no priority-1 tiles to be transmitted; that is, priority-2 tiles are put on a lower-priority transmit queue in the I/O buffer of each ISS server. (ATM switches would be allowed to drop the cells carrying these tiles.) Priority-3 tiles are those that should be retrieved and cached at the ISS server; these tiles are less likely to be needed by TerraVision. Note that there is a trade-off between "overpredicting" — requesting too many tiles — which would result in poor ISS performance and high network load, and "underpredicting," which would result in poor application performance.

Finally, a tile will continue to be included in TerraVision's request list if it is still needed and has not yet been delivered. Thus, tiles or tile requests that are dropped or otherwise "lost" in the network will likely be delivered in response to a subsequent request from the application.

Frequency of Requests

Another trade-off pertains to the frequency at which TerraVision sends its request list to the ISS. If the interval between requests is too large, then some tiles will not arrive when needed, resulting in a poor-quality display; in addition, the ISS will be idle and hence not used efficiently. On the other hand, if the interval is too short, then the request list might contain tiles that are currently in transit from servers to the application; this would result in poor ISS performance and redundant network traffic. For a typical MAGIC configuration, the interval between requests is currently set at 200 ms, a value that was found empirically to yield satisfactory performance. This value is based roughly on the measured latency of the ISS (about 100 ms) and on the estimated time required for a tile request to travel through the network from the TerraVision host to the ISS master and then to the most distant ISS server, plus the time for the tile itself to travel back to the host (perhaps a total of 50 ms). Additional measurements and analysis are needed to more precisely determine the appropriate request frequency as a function of the performance and location of system components and of network parameters.

For a typical MAGIC configuration, the interval between requests is currently set at 200 ms, a value that was found empirically to yield satisfactory performance.

Location of ISS Master

Since tile requests flow from TerraVision to the ISS master and thence to the servers themselves, the time for delivering the requests to the servers is minimized when the master is collocated with the TerraVision host. However, locating the master with the host is neither desirable nor practical for several reasons. The master is logically part of the ISS; therefore, its location should not be constrained by the application. Also, an ISS may be used with several applications concurrently, by multiple simultaneous users of a particular application, or by a user whose host may be unable to support any ISS functionality (e.g., a mobile user). Moreover, replication of the master would introduce problems associated with maintaining consistency among multiple masters when the ISS is in a read/write environment, as it would be when real-time data are being stored on the servers.

To first order, the delivery time of tile requests is limited by the time τ for a request to travel from TerraVision to the ISS server most distant from the TerraVision host. Hence, if the master is approximately on the path from the TerraVision host to that server, then τ will not be much greater than when the master and host are collocated. Furthermore, in the current MAGIC testbed, τ is much smaller than the sum of the disk latency and the network transit time. In other words, there is considerable freedom in choosing the location of the ISS master. Satisfactory system performance has been demonstrated, for example, with the TerraVision host in Kansas City, the ISS master in Sioux Falls, and servers in Minneapolis and Lawrence. Of course, this conclusion might change if faster servers reduce ISS latency considerably, or the geographic span of the network were substantially larger.

Distribution of Tiles on ISS Servers

The manner in which data are distributed among the servers determines the degree of parallelism and hence the aggregate throughput which can be obtained from the ISS. The data placement strategy depends on the application and is a function of data type and access patterns. For example, the retrieval pattern for a database of video clips would be quite different from that for a database of images. A strategy was developed for a terrain visualization type of application that minimizes the retrieval time for a set of tiles: the tiles assigned to a given disk are as far apart as possible in the terrain in order to maximize parallelism by minimizing the probability that tiles on a request list are on the same disk; and on each disk, tiles that are near each other in the terrain are placed as close as possible to minimize retrieval time. Although this was shown to be an optimal strategy for terrain path-following as in TerraVision [9], it was subsequently shown that ISS performance with random placement of tiles was only slightly worse. This was partly because tile retrieval time is much less than the latency in the ISS servers and network transit time, and is therefore not currently a significant factor in overall performance. Random placement is simpler to implement and is expected to be satisfactory for many other applications. However, as discussed for the location of the ISS master, this conclusion may have to be revisited if the performance or the geographic distribution of system components changes significantly.

Avoiding Cell Loss

When initially implemented, the MAGIC internetwork exhibited very low throughput in certain configurations. One cause of the low throughput was found to be mismatches between the burst rates of components in the communications path. Examples of such rate mismatches were:

- An OC-3 workstation interface transmitting cells at full rate across the network to a 100 Mb/s TAXI interface on another workstation
- Two or more OC-3 input ports at an ATM switch sending data to the same OC-3 output port

A mismatch, coupled with small buffers at the output ports of ATM switches, caused cells to be dropped, which in turn resulted in the retransmission of entire TCP packets, exacerbating the problem. In some cases the measured useful throughput was less than one percent of the capacity of the lower speed line.

Previously it was noted that in many cases a large MTU can increase throughput. However, once again there is a trade-off. As the MTU size is increased, the number of ATM cells needed to carry the MTU increases. The probability that one or more cells from the MTU will be dropped by the network therefore increases, which in turn increases the probability that the MTU will have to be retransmitted, thus possibly decreasing the effective throughput. Flow-control techniques together with large switch buffers and proper choice of protocol parameters did provide satisfactory performance. Nevertheless, the overriding conclusions are that the parameters of the entire end-to-end system, not just those of a single host or switch, must be tuned, each direction of the data path must be evaluated separately, and every component in each direction of the data path must be considered in the evaluation.

Performance

This section presents highlights of system-level performance issues and measurements of the MAGIC testbed. The input data rates that are needed to support the TerraVision application are calculated first, to provide the context for the subsequent discussion. Then the data rates that the network, the ISS, and the application host can actually support are described. Finally, a diagnostic tool that was developed to help analyze system performance is explained. More detailed information about all of the above topics can be found in [3, 5, 6, 10].

TerraVision is used in one of two modes, flyover or teleport, and the characteristics of the data flow for the two modes are quite different. Flyover requires a relatively steady flow over a relatively long period of time (many seconds), whereas teleport requires a large burst of data but occurs relatively infrequently. Quantitative requirements can be estimated as follows. A high-resolution full-screen display comprises about 100 tiles, each tile containing 128 x 128 pixels with 24 bits of color information, or approximately 0.4 Mb. If 10 new tiles are needed for a typical frame update during flyover, then at 30 frames/s, the average data rate is

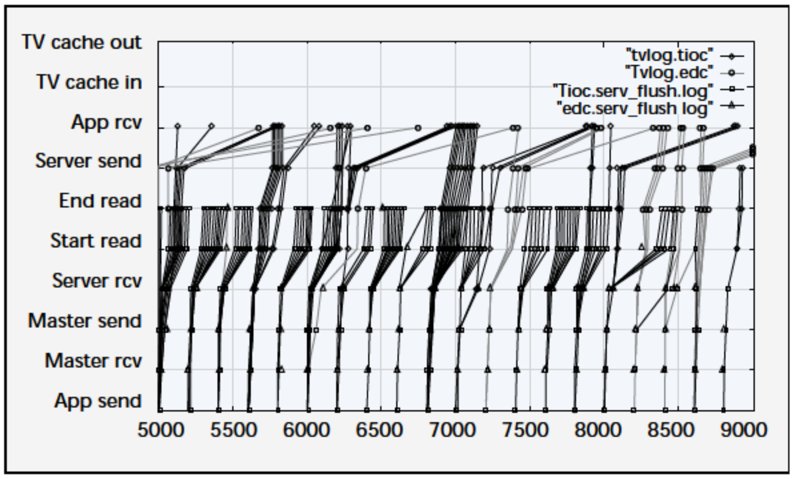
$$(30 \text{ frames/s}) \times (10 \text{ tiles/frame}) \times (0.4 \text{ Mb/tile}) \approx 120 \text{ Mb/s}$$

at the application level in the host. Protocol overhead might add approximately 15 percent to this value, resulting in a line rate of about 140 Mb/s. For a teleport, the burst rate is considerably higher because the entire screen must be repainted within, say, a quarter of a second after the user selects the new location. If the total latency between the instant the user enters the selection and the instant the first bit of the first tile arrives at the TerraVision host is 150 ms, then the full screen of data must be transferred in the remaining (250 – 150) = 100 ms, and the capacity needed to support the transfer is

$$(100 \text{ tiles}) \times (0.4 \text{ Mb/tile})/(0.1\text{s}) \approx 400 \text{ Mb/s}$$

at the application level, or about 450 Mb/s on the transmission line.

The line capacity needed near the TerraVision host site can be determined from these required rates and from the end-to-end throughput that can be attained in the network. Measurements on the MAGIC network showed that if the MTU and the TCP window sizes were large enough, and if flow control were used, then end-to-end TCP rates corresponding to about 80 percent of the line rate could be sustained; this is about 120 Mb/s on an OC-3 line. Thus, a flyover would completely fill a single OC-3 line, so in practice two lines are needed to allow for possible degradations and for variations around the average rate derived above. Similarly, one OC-12 or four OC-3 lines are needed to support a low-response-time teleport. Lower line capacity on the path near the host would degrade the response time (although the degradation would be less than linear because of the additive factor of ISS latency). In summary, the equivalent of two OC-3 lines into the host should give satisfactory flyover performance and a teleport response time less than 0.5 s, but more capacity is needed to reduce the response time and to provide some cushion for contention near the host site.



■ Figure 6. *Timing data from a configuration with two ISS servers. (Source: Laurence Berkeley National Laboratory)*

The next question is, "How many ISS servers are needed to support the application?" Early measurements of a variety of workstations configured as ISS servers showed that a typical SCSI disk delivered data at a steady rate of about 20 Mb/s; a single SCSI adapter with multiple disks could provide about 60 Mb/s; and a workstation with multiple adapters could deliver about 80 Mb/s. Additional disks or adapters did not increase throughput — the bottleneck apparently being memory bandwidth — but did increase the probability that the throughput could be sustained by ensuring that the server was not idle. These data indicate that tiles must be distributed over at least five servers to obtain the 400 Mb/s rate needed for good teleport performance.

The data streams from the ISS servers converge at the TerraVision host, and recent measurements showed that with four servers transmitting to a host with two OC-3 ATM ports, the aggregate application-level throughput was only about 100 Mb/s, and in fact was slightly less than the throughput with a single server. (The peak throughput was about 150 Mb/s, with two input streams.) Cells are apparently being dropped at the ATM interface. This is a serious bottleneck in overall system performance; the host and interface vendors are aware of the problem and are working on a solution.

Clearly, understanding the overall performance of a network-based distributed system such as MAGIC is an appreciably more complex undertaking than simply "concatenating" the standalone performance of the individual components because there are interactions among the components. It is important to be able to measure and correlate these interactions in order to understand and predict the performance of the system as a whole. Stated in concrete terms, a problem observed by a user could have a variety of causes. For example, in MAGIC it would be acceptable if low-resolution tiles are used occasionally in place of high-resolution tiles that are delayed or lost in transit (as described in the previous section), but it would be unacceptable if this occurred frequently. If such observable degradation did occur, the cause could be the application host dropping cells, ATM switches dropping cells, excessive delay somewhere in the ISS, low ISS throughput because of the way tiles are distributed among servers, processing limitations of the TerraVision host, or a combination of these and other phenomena.

To aid in pinpointing potential problems, accurately synchronized clocks were deployed at MAGIC sites, many components were instrumented to log traffic data, and a tool was developed for collecting, processing, and displaying the logged data [3]. The tool's graphical portrayal of measured data gives a readily comprehendible view of the overall operation of the system, permits performance estimates to be calculated easily, and provides an indication of which components may be causing performance problems. This tool, which was developed toward the end of the MAGIC project, has proved to be extremely valuable in diagnosing problems and in providing insight into techniques for improving performance. The tool is applicable to many high-speed distributed systems. A brief description of its use is given below.

Figure 6 displays a representative sample of 4 s of data from a configuration with the application host in Kansas City ("tioc" in the legend), the ISS master in Sioux Falls ("edc"), and one server at each. (The host was not running TerraVision, but an application that emulates TerraVision by sending the identical tile request lists which were sent during a previously recorded TerraVision session.) The diagram traces a time history for each requested tile, showing:

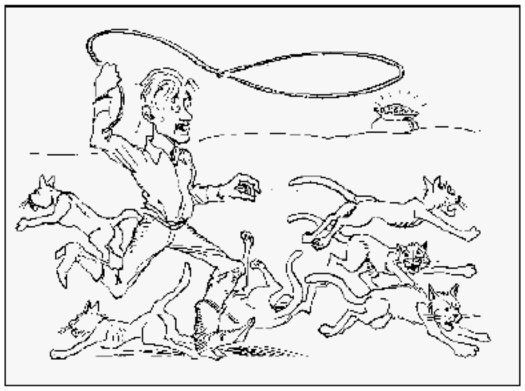
- When the application sends a request list (e.g., at 6800 ms) and when it is received by the ISS master (~6810)[2]
- When the master sends tile lists to the two servers (6820) and when they are received (~6840 and ~6850)
- When the servers start and complete their read operations
- When the tile data are transmitted by the servers and received by the host

In this example, the time between the request list leaving the application host and the first tile arriving at the host is 180 ms. The diagram shows that excluding the server time, the largest component of this 180 ms delay was queuing at the server, a result of TCP retransmissions of previously transmitted tiles that were, in effect, blocking transmission of subsequent tiles. (The shallow-sloped lines between "server send" and "app receive" represent tiles with TCP transmissions.) Rough calculations of throughput at each measurement point can readily be made by counting the number of tiles processed in a selected interval of time; for example, 15 tiles were received by the application between 6980 and 7130 ms, for a throughput of about 40 Mb/s.

Herding Cats

Although the MAGIC project was an ambitious undertaking, it nevertheless was able to achieve most of its goals. The success of the project seems all the more remarkable if one considers the degree of interorganizational collaboration that was required to design, develop, test, and integrate the individual testbed components and to ensure their interoperability. Indeed, fostering this collaboration was one of the most significant nontechnical challenges facing the project — and one of its noteworthy accomplishments.

More than a dozen diverse, geographically dispersed organizations participated in MAGIC, and many of the individuals involved in the project were experienced researchers who were used to working independently. Although five of these organizations were funded by ARPA, each had its own contract and a statement of work that was complementary to the others but theoretically could be executed separately from the rest. In addition, the commercial carriers and other organizations that were expected to be major contributors were not externally funded and therefore were under no obligation to participate actively in the effort. Thus, the situation at the outset was not unlike the metaphorical herding of cats (Fig. 7).



■ Figure 7. *In the beginning, things looked difficult.*

The authors of this article were funded by ARPA to oversee and coordinate the research and development (R & D) efforts of the five ARPA-funded research participants, and to help facilitate their collaboration with the carriers and with the other organizations contributing to the project. This was a challenging assignment because none of these organizations was contractually bound to answer to a third party, so voluntary compliance of all organizations was required. Considering the cast of players and the circumstances of their affiliation, it would have been imprudent to attempt to dictate direction or to impose preferences. Furthermore, to do so not only would have been ineffective but would have been counterproductive because a heavy-handed management style would have stifled the innovation that was critical to the success of the project. In other words, peremptory management might have led to passive obedience (Fig. 8), but the results would have been uninspired [11].

The challenge was to create an environment that facilitated progress and encouraged cooperation while at the same time promoting creativity and initiative. The approach used was to obtain mutual agreement on a common set of goals and related milestones which could not be achieved without the contributions of all of the participants. In this way, the focus of the work shifted from the pursuit of individual goals to the pursuit of common goals, and collaboration was implicitly understood to be essential for success. In retrospect, the reasons why this approach worked well seem obvious. Having a common set of goals engendered an esprit de corps among the participants which gave the sense of a "virtual" organization dedicated to the success of MAGIC.

However, participants soon recognized that while camaraderie and commitment were vital to success, team spirit alone was not sufficient to ensure that success. Differences in work styles, conflicting priorities, geographical dispersion of people and resources, and the sheer magnitude of the interdependencies underscored the need for centralized leadership and for "formal" procedures for coordinating activities. As a result, members of the MAGIC team willingly consented to, and complied with, a set of management practices that they perceived as facilitating the achievement of their technical objectives. The management style was collegial with the authors serving as facilitators for defining and prioritizing project activities, as mediators for resolving disputes, as liaison with the project sponsor (ARPA), and as catalysts for promoting the team interactions required to move forward. Thus, as indicated in Fig. 9, MAGIC took a hybrid approach to managing and coordinating its R & D, with

[2] *These numerical values were obtained from a version of this diagram with an expanded timescale.*

progress toward common goals achieved through high-level consensus-building among the participants.

Three practices stand out as being most critical to the success of the project: demonstrations, planning with flexibility, and ongoing communications.

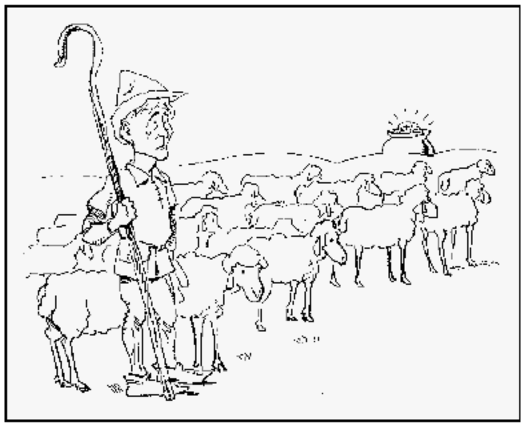


Figure 8. *This wouldn't work either.*

Demonstrations — Although the components of the MAGIC testbed were designed to operate as parts of a system, they were developed independently by organizations that were not collocated. Therefore, interoperability testing and debugging were difficult. To deal with this problem, demonstrations for external observers were scheduled to mark the achievement of milestones. These demonstrations provided a strong incentive for overcoming the logistical obstacles to testing, and for uncovering and finding solutions to tough problems. At first glance, these events appeared to be distractions from the research and a drain on people and resources, and initially they were deemed antithetical to an R & D project. In actuality they were the single most important factor in accelerating progress. Often, it was in the typically frantic last hours before a crucial demonstration that creative solutions to unforeseen problems were conceived.

A number of major demonstrations were scheduled in conjunction with quarterly project meetings or technical symposia. The first, which took place approximately halfway into the three-year project duration, marked the completion of the first phase of the MAGIC testbed: initial versions of TerraVision and the ISS working together over a partially completed backbone. The second, which occurred about six months later, demonstrated improved versions of both TerraVision and the ISS working together over the full internetwork. This demonstration was attended by prospective end users of the system who provided valuable feedback, including suggestions for additional capabilities which were subsequently incorporated into TerraVision, substantially improving the utility of the application.

Planning … with Flexibility — Researchers are notoriously reluctant to document their ideas and approaches in advance for fear of forfeiting their flexibility or limiting their options; however, failure to do so can spell disaster in a collaborative venture involving multiple organizations. Therefore, one of the first priorities of the MAGIC team was to develop a comprehensive research plan for the project. If truth be told, the process of planning was far more valuable than the plan itself. In creating the plan, each organization was forced to clearly define its tasks and milestones, to explore alternative approaches to accomplishing the work, and, most important, to identify interorganizational relationships and dependencies. It was understood that tasks and milestones, as well as technical approaches, would most likely change and evolve over the course of the three-year effort, and the plan was considered a working document to be revised and revisited as appropriate. However, at the conclusion of the project, it was gratifying to discover that the participants had accomplished most of the work they had intended to do within the allotted time and budget constraints.

Ongoing Communications — Ongoing communication was an important factor in maintaining cohesiveness among team members, and was essential for accomplishing the work. Regular interaction was achieved by holding weekly teleconferences and quarterly project meetings to discuss technical issues and interorganizational dependencies, to plan joint activities and events, and to identify and resolve problems. In addition, a variety of mechanisms for exchanging information were established, including multiple mailers, and a project server for storing and retrieving documents such as project plans, papers, and reports. To facilitate collaboration on documents, a common desktop publishing package, which was available for multiple platforms, was adopted by the team very early in the project.

Lessons Learned

The previous section described the challenges of managing the MAGIC project, and discussed some of the factors that promoted cooperation and collaboration among the participants in this multidisciplinary, multi-organizational effort. Below are some additional lessons that were learned — sometimes with pain — during the course of the three-year project.

Technology for R&D Projects

R & D projects such as MAGIC depend on state-of-the-art technology to achieve their goals. There are two alternatives for obtaining this technology: develop it as part of the project, or procure it from vendors or other sources. Where possible, MAGIC opted for the latter alternative, and milestones were planned based on vendors' stated intentions regarding the capabilities of and projected delivery dates for critical hardware and software. As a consequence of this decision, MAGIC researchers learned two important lessons.

Be Prepared to Deal with the Limitations of Vendor Products — Some of the vendor-supplied state-of-the-art products required by MAGIC, for example, the SONET terminals and the ATM switches, were available on schedule and performed satisfactorily. Others, however, were either not available in the time frame expected (e.g., OC-12 cards for the ATM switches) or did not function as anticipated. Specifically, MAGIC researchers had to deal with three types of limitations:

- Product (im)maturity: Early production versions of products required a significant amount of tuning and debugging that would be unacceptable in a mature product. For example, some workstation operating systems initially had hard-coded upper limits on the TCP window size, limiting the achievable throughput across a network having a large bandwidth-delay product.
- Standalone performance vs. system performance: Products did not perform per their standalone specifications when incorporated into a system. For example, the measured rate of a disk on an ISS server was typically less than half the specified rate (perhaps caused by interactions with the SCSI adapter).
- Single-component performance vs. multiple-component

performance: When multiple components were made to operate in parallel, their performance did not scale linearly. For example, the rate at which the TerraVision host could absorb data increased only slightly as the number of ATM interfaces was increased.

Encourage the Active Involvement of Vendors in the R & D Effort — MAGIC depended on products that were under development or "on the horizon" when the project was initiated, and progress often hinged on timely access to early releases or upgrades. In some cases, market pressures on vendors took precedence over research needs, and the products were delayed, or anticipated features were postponed or eliminated. In other cases, products were released but were not robust, and vendor support was difficult to obtain. If equipment vendors had been more actively involved in the R & D effort, the other researchers, as well as ARPA and the carriers, would have been in a better position to influence vendor priorities and development schedules, and would have been more likely to gain the support and assistance they needed to correct shortcomings. Active vendor participation would have been beneficial to the vendors as well, providing them with insight into the strengths and limitations of their products, and helping them identify additional features and performance enhancements that might improve their competitive advantage.

Despite the difficulties associated with relying on vendors for supporting technology, using vendor-supplied products was preferable to developing customized products as part of the project. Such development would have been time- and resource-intensive, and possibly a duplication of effort. In addition, customized technology is expensive to replicate and difficult to transfer to other domains.

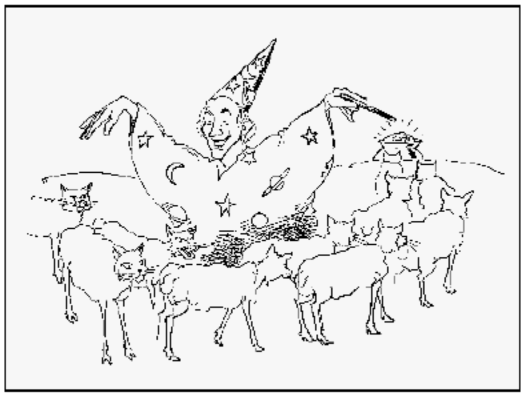


Figure 9. *Heterogeneous collaborative interoperability.*

Support for Demonstrations

As discussed previously, demonstrations were sometimes scheduled to coincide with major project events or milestones. In addition, requests to demonstrate the capabilities of the testbed were occasionally made by ARPA, by the management of the participating organizations, or by prospective end users. While there were significant benefits associated with holding these demonstrations, preparing for them was time-consuming because it was frequently necessary to reconfigure the network and to relocate and assemble the required hardware. The MAGIC team learned two lessons that helped facilitate the conduct of demonstrations during the later stages of the project.

Establish a Reliable Testbed Configuration to Support Demonstrations — Although demonstrations proved to be a significant stimulus to progress, they sometimes conflicted with planned experiments or with development and testing activities. This was particularly troubling when work-in-progress was interrupted or put on hold for a relatively long period of time in order to reconfigure the network (or to test modifications to TerraVision or the ISS) to support a scheduled event. This situation was remedied by implementing stable versions of TerraVision and the ISS and deploying them at selected locations. These versions were used to support demonstrations, performance measurements, and related activities. Updates to the demonstration versions of TerraVision and the ISS were coordinated to ensure their compatibility.

Plan Equipment Logistics Carefully and in Advance — Another problem, pertinent to development as well as demonstrations, is the availability of equipment. Since budgets are finite, choices must be made regarding what equipment to purchase and where this equipment should be located. While it is impossible to foresee all contingencies, equipment needs should be determined as far in advance as possible. Doing so will minimize the disruptions and stress associated with disassembling and transporting hardware over long distances, and acquiring essential components on short notice. It is especially important to develop strategies for supporting off-site demonstrations, particularly those that involve relocating large, cumbersome equipment or require expensive hardware which cannot easily be moved and is difficult to borrow or lease.

One way of helping to ensure that demonstrations can be accommodated without undue disruption is to purchase spares of inexpensive equipment. These spares would be available not only for demonstrations, but for development and experimentation in the event that an original malfunctions. It is less feasible to duplicate expensive equipment; however, if vendors of critical components are actively involved in the project, they might be willing to support demonstrations by providing the necessary hardware.

Support for Development

The MAGIC testbed consists of components that were designed to interoperate but were developed independently by organizations that were geographically separated. In addition, the end users of the system were not research participants in the project. The following lessons were learned regarding how to work more effectively and efficiently under such conditions.

Build Tools to Enable Independent Development of Interoperable Components — Interoperability testing of a given component was challenging because it required that other components possess a level of functionality or performance that was not always available when the tests were ready to be conducted. One way to alleviate this problem was to implement component simulators that enabled interoperability testing. In MAGIC, the implementation of a TerraVision simulator hastened progress on the ISS, whereas the decision not to implement an ISS simulator increased the time needed to complete TerraVision.

Provide High-Speed Network Connections for All Major Participants — Proper testing of TerraVision and the ISS required high-speed interconnectivity. However, SRI and LBNL, the respective developers of these components, did not have such connectivity. As a result, interoperability testing could not be performed locally, and testing at remote sites was both burdensome and inefficient. In the MAGIC project, both of these organizations would have benefited from having high-speed links to the backbone.

Solicit Periodic Input from End Users — Getting input from end users helps to ensure that the final product has useful features, satisfactory performance, and a well-designed interface. Input regarding desired capabilities should be solicited early in the effort and regularly thereafter as development progresses. Although, as noted in the previous section, MAGIC did benefit from such input, the project would have benefited even more if that input had been obtained earlier and more frequently.

Future Work

The MAGIC project has demonstrated a high-speed, wide-area IP/ATM internetwork that supports a real-time terrain visualization application and a distributed storage system. ARPA recently approved funding for a three-year follow-on effort, MAGIC-II, which will build on the technology developed in the original MAGIC project and on the existing MAGIC network facilities. There are two major interrelated goals in MAGIC-II:

- To enhance and upgrade the testbed to demonstrate the utility and capabilities of distributed processing and network-based storage, coupled with high-speed networks, to support a new generation of real-time applications.
- To create a very large internetwork with many end users that will be a realistic test environment for ATM technology and for the above type of application.

The MAGIC-II testbed will demonstrate the scalability of the distributed storage and distributed processing concepts by configuring systems that have a large number of servers and processors on many ATM networks spanning a large geographic area, and have multiple sets of data and multiple simultaneous users.

The MAGIC-II testbed will demonstrate the scalability of the distributed storage and distributed processing concepts by configuring systems that have a large number of servers and processors on many ATM networks spanning a large geographic area, and have multiple sets of data and multiple simultaneous users.

The MAGIC-II testbed is based on a very general paradigm in which high-performance computing, storage, and communications are used to provide rapid access to large amounts of distributed data, including real-time data that must be processed and delivered to an end user on demand. Applications that use this paradigm arise in a variety of situations, including military operations, intelligence imagery analysis, and natural disaster response. The exact type, location, and ownership of the data used by these applications may not be known in advance, and these data may require a large amount of processing to be transformed into useful information. In addition, the processed data may have to be delivered to end users with a range of communications speeds, link qualities, computational powers, and display capabilities. The data, as well as the computing and storage resources required to process them, may reside in multiple administrative domains that have different usage and access control policies.

Specific work to be done in MAGIC-II includes augmenting the MAGIC internetwork with wireless nodes and interconnecting it with other IP/ATM internetworks to create a nation-wide, high-speed, wide-area testbed. This testbed will be used for experimentation with protocols, with routing techniques, and with mobile access to backbone services. A new version of TerraVision that can perform on-the-fly rectification coupled with algorithms for "in-transit" processing will permit near-real-time visualization of raw imagery, enabling data from sensors to be viewed within minutes (rather than hours) after being generated. (Fig. 1.) Data fusion techniques will allow disparate data types to be overlaid. The processing will be performed by sets of distributed devices that are constructed from resources owned by multiple administrative domains. Algorithms that dynamically determine the current state of the network will provide information to the application so that it can adapt to current system performance and to available system resources.

The MAGIC-II project will certainly benefit from the lessons learned in the original MAGIC project. Nevertheless, as with any research effort, new challenges will be encountered, and new lessons, both technical and organizational, will be learned in meeting these challenges. Stay tuned.

References

[1] R. Binder, "Issues in Gigabit Networking," *Proc. IEEE Globecom '92*, Orlando, FL, 12/92. Also see: http://www.CNRI.Reston.VA.US:4000/public/overview.html

[2] J. Evans *et al.*, "A 622 Mb/s LAN/WAN Gateway and Experiences with Wide Area ATM Networking," *IEEE Network*, this issue.

[3] B. Tierney *et al.*, "Performance Analysis in High-Speed Wide Area IP-over-ATM Networks: Top-to-Bottom End-to-End Monitoring," *IEEE Network*, this issue.

[4] M. Laubach, "Classical IP and ARP over ATM," RFC 1577; 20, Jan. 1994.

[5] B.J. Ewy *et al.*, "TCP/ATM Experiences in the MAGIC Testbed," *Proc. 4th IEEE Symp. High Perf. Dist. Comp*, Aug.1995, pp. 87–93.

[6] J. D. Cavanaugh, Minnesota Supercomputer Center, personal communication.

[7] Y. G. Leclerc and S. Q. Lau Jr., "TerraVision: A Terrain Visualization System," Tech. Note 540, SRI International, Menlo Park, CA, Apr. 1994.

[8] B. Tierney *et al.*, "Distributed Parallel Data Storage Systems," *Proc. ACM Multimedia '94*, Oct. 1994. Also available as http://george.lbl.gov/ ISS/papers/ISS-paper.ACM.final.html

[9] L. T. Chen and D. Rotem, "Declustering Objects for Visualization," *Proc. 19th VLDB (Very Large Database) Conf.*, 1993.

[10] B. Tierney *et al.*, "Using High Speed Networks to Enable Distributed Parallel Image Server Systems," *Proc. Supercomputing '94*, Nov. 1994. Also available as: http://george.lbl.gov/ ISS/papers/ISS-paper.SC94.final.html

[11] S. Adams, "Dilbert," *Boston Globe*, Jan. 14, 1996.

Biographies

BARBARA FULLER recently joined Mitretek Systems in McLean, Virginia, as lead staff in the Center for Information Technology Systems. Prior to joining Mitretek, she held a similar position with the MITRE Corporation in Bedford, Massachusetts, where she provided technical, systems engineering, and program planning, coordination, and analysis support to a variety of DoD-sponsored advanced information systems projects, including MAGIC. She also spent 12 years at MITRE managing multidisciplinary projects for U.S. Government agencies dealing with toxic chemicals in the environment. She received her B.A. degree in chemistry from Western Reserve University in Cleveland, Ohio, and her M.S. and Ph.D. degrees in chemistry from New York University.

IRA RICHER is director of networking research at the Corporation for National Research Initiatives (CNRI), Reston, Virginia. His current work includes coordinating the activities of about a dozen companies in a project involving trials of broadband services to residences, and managing the MAGIC project. Prior to joining CNRI, Dr. Richer was with MITRE, where he supervised a small group working on advanced networks and applications. From 1988 to 1991, Dr. Richer was program manager for high-performance networking at ARPA. He initiated ARPA's program in gigabit networks, and he launched ARPA's work in all-optical networking. Dr. Richer received a B.E.E. degree from Rensselaer Polytechnic Institute, and M.S. and Ph.D. degrees from the California Institute of Technology.

INTERNATIONAL TELECOMMUNICATION UNION

CCITT

T.81

(09/92)

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

TERMINAL EQUIPMENT AND PROTOCOLS FOR TELEMATIC SERVICES

INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

Recommendation T.81

Foreword

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The CCITT (the International Telegraph and Telephone Consultative Committee) is a permanent organ of the ITU. Some 166 member countries, 68 telecom operating entities, 163 scientific and industrial organizations and 39 international organizations participate in CCITT which is the body which sets world telecommunications standards (Recommendations).

The approval of Recommendations by the members of CCITT is covered by the procedure laid down in CCITT Resolution No. 2 (Melbourne, 1988). In addition, the Plenary Assembly of CCITT, which meets every four years, approves Recommendations submitted to it and establishes the study programme for the following period.

In some areas of information technology, which fall within CCITT's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC. The text of CCITT Recommendation T.81 was approved on 18th September 1992. The identical text is also published as ISO/IEC International Standard 10918-1.

CCITT NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized private operating agency.

Contents

Introduction

This CCITT Recommendation | ISO/IEC International Standard was prepared by CCITT Study Group VIII and the Joint Photographic Experts Group (JPEG) of ISO/IEC JTC 1/SC 29/WG 10. This Experts Group was formed in 1986 to establish a standard for the sequential progressive encoding of continuous tone grayscale and colour images.

Digital Compression and Coding of Continuous-tone Still images, is published in two parts:

- Requirements and guidelines;
- Compliance testing.

This part, Part 1, sets out requirements and implementation guidelines for continuous-tone still image encoding and decoding processes, and for the coded representation of compressed image data for interchange between applications. These processes and representations are intended to be generic, that is, to be applicable to a broad range of applications for colour and grayscale still images within communications and computer systems. Part 2, sets out tests for determining whether implementations comply with the requirments for the various encoding and decoding processes specified in Part 1.

The user's attention is called to the possibility that – for some of the coding processes specified herein – compliance with this Recommendation | International Standard may require use of an invention covered by patent rights. See Annex L for further information.

The requirements which these processes must satisfy to be useful for specific image communications applications such as facsimile, Videotex and audiographic conferencing are defined in CCITT Recommendation T.80. The intent is that the generic processes of Recommendation T.80 will be incorporated into the various CCITT Recommendations for terminal equipment for these applications.

In addition to the applications addressed by the CCITT and ISO/IEC, the JPEG committee has developped a compression standard to meet the needs of other applications as well, including desktop publishing, graphic arts, medical imaging and scientific imaging.

Annexes A, B, C, D, E, F, G, H and J are normative, and thus form an integral part of this Specification. Annexes K, L and M are informative and thus do not form an integral part of this Specification.

This Specification aims to follow the guidelines of CCITT and ISO/IEC JTC 1 on *Rules for presentation of CCITT | ISO/IEC common text*.

INTERNATIONAL STANDARD

CCITT RECOMMENDATION

INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

1 Scope

This CCITT Recommendation | International Standard is applicable to continuous-tone – grayscale or colour – digital still image data. It is applicable to a wide range of applications which require use of compressed images. It is not applicable to bi-level image data.

This Specification

- specifies processes for converting source image data to compressed image data;
- specifies processes for converting compressed image data to reconstructed image data;
- gives guidance on how to implement these processes in practice;
- specifies coded representations for compressed image data.

NOTE – This Specification does not specify a complete coded image representation. Such representations may include certain parameters, such as aspect ratio, component sample registration, and colour space designation, which are application-dependent.

2 Normative references

The following CCITT Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this CCITT Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this CCITT Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The CCITT Secretariat maintains a list of currently valid CCITT Recommendations.

- *CCITT Recommendation T.80 (1992), Common components for image compression and communication – Basic principles.*

3 Definitions, abbreviations and symbols

3.1 Definitions and abbreviations

For the purposes of this Specification, the following definitions apply.

3.1.1 abbreviated format: A representation of compressed image data which is missing some or all of the table specifications required for decoding, or a representation of table-specification data without frame headers, scan headers, and entropy-coded segments.

3.1.2 AC coefficient: Any DCT coefficient for which the frequency is not zero in at least one dimension.

3.1.3 (adaptive) (binary) arithmetic decoding: An entropy decoding procedure which recovers the sequence of symbols from the sequence of bits produced by the arithmetic encoder.

3.1.4 (adaptive) (binary) arithmetic encoding: An entropy encoding procedure which codes by means of a recursive subdivision of the probability of the sequence of symbols coded up to that point.

3.1.5 application environment: The standards for data representation, communication, or storage which have been established for a particular application.

3.1.6 arithmetic decoder: An embodiment of arithmetic decoding procedure.

3.1.7 arithmetic encoder: An embodiment of arithmetic encoding procedure.

3.1.8 baseline (sequential): A particular sequential DCT-based encoding and decoding process specified in this Specification, and which is required for all DCT-based decoding processes.

3.1.9 binary decision: Choice between two alternatives.

3.1.10 bit stream: Partially encoded or decoded sequence of bits comprising an entropy-coded segment.

3.1.11 block: An 8×8 array of samples or an 8×8 array of DCT coefficient values of one component.

3.1.12 block-row: A sequence of eight contiguous component lines which are partitioned into 8×8 blocks.

3.1.13 byte: A group of 8 bits.

3.1.14 byte stuffing: A procedure in which either the Huffman coder or the arithmetic coder inserts a zero byte into the entropy-coded segment following the generation of an encoded hexadecimal X’FF’ byte.

3.1.15 carry bit: A bit in the arithmetic encoder code register which is set if a carry-over in the code register overflows the eight bits reserved for the output byte.

3.1.16 ceiling function: The mathematical procedure in which the greatest integer value of a real number is obtained by selecting the smallest integer value which is greater than or equal to the real number.

3.1.17 class (of coding process): Lossy or lossless coding processes.

3.1.18 code register: The arithmetic encoder register containing the least significant bits of the partially completed entropy-coded segment. Alternatively, the arithmetic decoder register containing the most significant bits of a partially decoded entropy-coded segment.

3.1.19 coder: An embodiment of a coding process.

3.1.20 coding: Encoding or decoding.

3.1.21 coding model: A procedure used to convert input data into symbols to be coded.

3.1.22 (coding) process: A general term for referring to an encoding process, a decoding process, or both.

3.1.23 colour image: A continuous-tone image that has more than one component.

3.1.24 columns: Samples per line in a component.

3.1.25 component: One of the two-dimensional arrays which comprise an image.

3.1.26 compressed data: Either compressed image data or table specification data or both.

3.1.27 compressed image data: A coded representation of an image, as specified in this Specification.

3.1.28 compression: Reduction in the number of bits used to represent source image data.

3.1.29 conditional exchange: The interchange of MPS and LPS probability intervals whenever the size of the LPS interval is greater than the size of the MPS interval (in arithmetic coding).

3.1.30 (conditional) probability estimate: The probability value assigned to the LPS by the probability estimation state machine (in arithmetic coding).

3.1.31 conditioning table: The set of parameters which select one of the defined relationships between prior coding decisions and the conditional probability estimates used in arithmetic coding.

3.1.32 context: The set of previously coded binary decisions which is used to create the index to the probability estimation state machine (in arithmetic coding).

3.1.33 continuous-tone image: An image whose components have more than one bit per sample.

3.1.34 data unit: An 8×8 block of samples of one component in DCT-based processes; a sample in lossless processes.

3.1.35 DC coefficient: The DCT coefficient for which the frequency is zero in both dimensions.

3.1.36 DC prediction: The procedure used by DCT-based encoders whereby the quantized DC coefficient from the previously encoded 8×8 block of the same component is subtracted from the current quantized DC coefficient.

3.1.37 (DCT) coefficient: The amplitude of a specific cosine basis function – may refer to an original DCT coefficient, to a quantized DCT coefficient, or to a dequantized DCT coefficient.

3.1.38 decoder: An embodiment of a decoding process.

3.1.39 decoding process: A process which takes as its input compressed image data and outputs a continuous-tone image.

3.1.40 default conditioning: The values defined for the arithmetic coding conditioning tables at the beginning of coding of an image.

3.1.41 dequantization: The inverse procedure to quantization by which the decoder recovers a representation of the DCT coefficients.

3.1.42 differential component: The difference between an input component derived from the source image and the corresponding reference component derived from the preceding frame for that component (in hierarchical mode coding).

3.1.43 differential frame: A frame in a hierarchical process in which differential components are either encoded or decoded.

3.1.44 (digital) reconstructed image (data): A continuous-tone image which is the output of any decoder defined in this Specification.

3.1.45 (digital) source image (data): A continuous-tone image used as input to any encoder defined in this Specification.

3.1.46 (digital) (still) image: A set of two-dimensional arrays of integer data.

3.1.47 discrete cosine transform; DCT: Either the forward discrete cosine transform or the inverse discrete cosine transform.

3.1.48 downsampling (filter): A procedure by which the spatial resolution of an image is reduced (in hierarchical mode coding).

3.1.49 encoder: An embodiment of an encoding process.

3.1.50 encoding process: A process which takes as its input a continuous-tone image and outputs compressed image data.

3.1.51 entropy-coded (data) segment: An independently decodable sequence of entropy encoded bytes of compressed image data.

3.1.52 (entropy-coded segment) pointer: The variable which points to the most recently placed (or fetched) byte in the entropy encoded segment.

3.1.53 entropy decoder: An embodiment of an entropy decoding procedure.

3.1.54 entropy decoding: A lossless procedure which recovers the sequence of symbols from the sequence of bits produced by the entropy encoder.

3.1.55 entropy encoder: An embodiment of an entropy encoding procedure.

3.1.56 entropy encoding: A lossless procedure which converts a sequence of input symbols into a sequence of bits such that the average number of bits per symbol approaches the entropy of the input symbols.

3.1.57 extended (DCT-based) process: A descriptive term for DCT-based encoding and decoding processes in which additional capabilities are added to the baseline sequential process.

3.1.58 forward discrete cosine transform; FDCT: A mathematical transformation using cosine basis functions which converts a block of samples into a corresponding block of original DCT coefficients.

3.1.59 frame: A group of one or more scans (all using the same DCT-based or lossless process) through the data of one or more of the components in an image.

3.1.60 frame header: A marker segment that contains a start-of-frame marker and associated frame parameters that are coded at the beginning of a frame.

3.1.61 frequency: A two-dimensional index into the two-dimensional array of DCT coefficients.

3.1.62 (frequency) band: A contiguous group of coefficients from the zig-zag sequence (in progressive mode coding).

3.1.63 full progression: A process which uses both spectral selection and successive approximation (in progressive mode coding).

3.1.64 grayscale image: A continuous-tone image that has only one component.

3.1.65 hierarchical: A mode of operation for coding an image in which the first frame for a given component is followed by frames which code the differences between the source data and the reconstructed data from the previous frame for that component. Resolution changes are allowed between frames.

3.1.66 hierarchical decoder: A sequence of decoder processes in which the first frame for each component is followed by frames which decode an array of differences for each component and adds it to the reconstructed data from the preceding frame for that component.

3.1.67 hierarchical encoder: The mode of operation in which the first frame for each component is followed by frames which encode the array of differences between the source data and the reconstructed data from the preceding frame for that component.

3.1.68 horizontal sampling factor: The relative number of horizontal data units of a particular component with respect to the number of horizontal data units in the other components.

3.1.69 Huffman decoder: An embodiment of a Huffman decoding procedure.

3.1.70 Huffman decoding: An entropy decoding procedure which recovers the symbol from each variable length code produced by the Huffman encoder.

3.1.71 Huffman encoder: An embodiment of a Huffman encoding procedure.

3.1.72 Huffman encoding: An entropy encoding procedure which assigns a variable length code to each input symbol.

3.1.73 Huffman table: The set of variable length codes required in a Huffman encoder and Huffman decoder.

3.1.74 image data: Either source image data or reconstructed image data.

3.1.75 interchange format: The representation of compressed image data for exchange between application environments.

3.1.76 interleaved: The descriptive term applied to the repetitive multiplexing of small groups of data units from each component in a scan in a specific order.

3.1.77 inverse discrete cosine transform; IDCT: A mathematical transformation using cosine basis functions which converts a block of dequantized DCT coefficients into a corresponding block of samples.

3.1.78 Joint Photographic Experts Group; JPEG: The informal name of the committee which created this Specification. The “joint” comes from the CCITT and ISO/IEC collaboration.

3.1.79 latent output: Output of the arithmetic encoder which is held, pending resolution of carry-over (in arithmetic coding).

3.1.80 less probable symbol; LPS: For a binary decision, the decision value which has the smaller probability.

3.1.81 level shift: A procedure used by DCT-based encoders and decoders whereby each input sample is either converted from an unsigned representation to a two’s complement representation or from a two’s complement representation to an unsigned representation.

3.1.82 lossless: A descriptive term for encoding and decoding processes and procedures in which the output of the decoding procedure(s) is identical to the input to the encoding procedure(s).

3.1.83 lossless coding: The mode of operation which refers to any one of the coding processes defined in this Specification in which all of the procedures are lossless (see Annex H).

3.1.84 lossy: A descriptive term for encoding and decoding processes which are not lossless.

3.1.85 marker: A two-byte code in which the first byte is hexadecimal FF (X’FF’) and the second byte is a value between 1 and hexadecimal FE (X’FE’).

3.1.86 marker segment: A marker and associated set of parameters.

3.1.87 MCU-row: The smallest sequence of MCU which contains at least one line of samples or one block-row from every component in the scan.

3.1.88 minimum coded unit; MCU: The smallest group of data units that is coded.

3.1.89 modes (of operation): The four main categories of image coding processes defined in this Specification.

3.1.90 more probable symbol; MPS: For a binary decision, the decision value which has the larger probability.

3.1.91 non-differential frame: The first frame for any components in a hierarchical encoder or decoder. The components are encoded or decoded without subtraction from reference components. The term refers also to any frame in modes other than the hierarchical mode.

3.1.92 non-interleaved: The descriptive term applied to the data unit processing sequence when the scan has only one component.

3.1.93 parameters: Fixed length integers 4, 8 or 16 bits in length, used in the compressed data formats.

3.1.94 point transform: Scaling of a sample or DCT coefficient.

3.1.95 precision: Number of bits allocated to a particular sample or DCT coefficient.

3.1.96 predictor: A linear combination of previously reconstructed values (in lossless mode coding).

3.1.97 probability estimation state machine: An interlinked table of probability values and indices which is used to estimate the probability of the LPS (in arithmetic coding).

3.1.98 probability interval: The probability of a particular sequence of binary decisions within the ordered set of all possible sequences (in arithmetic coding).

3.1.99 (probability) sub-interval: A portion of a probability interval allocated to either of the two possible binary decision values (in arithmetic coding).

3.1.100 procedure: A set of steps which accomplishes one of the tasks which comprise an encoding or decoding process.

3.1.101 process: See coding process.

3.1.102 progressive (coding): One of the DCT-based processes defined in this Specification in which each scan typically improves the quality of the reconstructed image.

3.1.103 progressive DCT-based: The mode of operation which refers to any one of the processes defined in Annex G.

3.1.104 quantization table: The set of 64 quantization values used to quantize the DCT coefficients.

3.1.105 quantization value: An integer value used in the quantization procedure.

3.1.106 quantize: The act of performing the quantization procedure for a DCT coefficient.

3.1.107 reference (reconstructed) component: Reconstructed component data which is used in a subsequent frame of a hierarchical encoder or decoder process (in hierarchical mode coding).

3.1.108 renormalization: The doubling of the probability interval and the code register value until the probability interval exceeds a fixed minimum value (in arithmetic coding).

3.1.109 restart interval: The integer number of MCUs processed as an independent sequence within a scan.

3.1.110 restart marker: The marker that separates two restart intervals in a scan.

3.1.111 run (length): Number of consecutive symbols of the same value.

3.1.112 sample: One element in the two-dimensional array which comprises a component.

3.1.113 sample-interleaved: The descriptive term applied to the repetitive multiplexing of small groups of samples from each component in a scan in a specific order.

3.1.114 scan: A single pass through the data for one or more of the components in an image.

3.1.115 scan header: A marker segment that contains a start-of-scan marker and associated scan parameters that are coded at the beginning of a scan.

3.1.116 sequential (coding): One of the lossless or DCT-based coding processes defined in this Specification in which each component of the image is encoded within a single scan.

3.1.117 sequential DCT-based: The mode of operation which refers to any one of the processes defined in Annex F.

3.1.118 spectral selection: A progressive coding process in which the zig-zag sequence is divided into bands of one or more contiguous coefficients, and each band is coded in one scan.

3.1.119 stack counter: The count of X’FF’ bytes which are held, pending resolution of carry-over in the arithmetic encoder.

3.1.120 statistical conditioning: The selection, based on prior coding decisions, of one estimate out of a set of conditional probability estimates (in arithmetic coding).

3.1.121 statistical model: The assignment of a particular conditional probability estimate to each of the binary arithmetic coding decisions.

3.1.122 statistics area: The array of statistics bins required for a coding process which uses arithmetic coding.

3.1.123 statistics bin: The storage location where an index is stored which identifies the value of the conditional probability estimate used for a particular arithmetic coding binary decision.

3.1.124 successive approximation: A progressive coding process in which the coefficients are coded with reduced precision in the first scan, and precision is increased by one bit with each succeeding scan.

3.1.125 table specification data: The coded representation from which the tables used in the encoder and decoder are generated and their destinations specified.

3.1.126 transcoder: A procedure for converting compressed image data of one encoder process to compressed image data of another encoder process.

3.1.127 (uniform) quantization: The procedure by which DCT coefficients are linearly scaled in order to achieve compression.

3.1.128 upsampling (filter): A procedure by which the spatial resolution of an image is increased (in hierarchical mode coding).

3.1.129 vertical sampling factor: The relative number of vertical data units of a particular component with respect to the number of vertical data units in the other components in the frame.

3.1.130 zero byte: The X’00’ byte.

3.1.131 zig-zag sequence: A specific sequential ordering of the DCT coefficients from (approximately) lowest spatial frequency to highest.

3.1.132 3-sample predictor: A linear combination of the three nearest neighbor reconstructed samples to the left and above (in lossless mode coding).

3.2 Symbols

The symbols used in this Specification are listed below.

A	probability interval
AC	AC DCT coefficient
AC_{ji}	AC coefficient predicted from DC values
Ah	successive approximation bit position, high
Al	successive approximation bit position, low
Ap_i	*i*th 8-bit parameter in APP_n segment
APP_n	marker reserved for application segments
B	current byte in compressed data
B2	next byte in compressed data when B = X’FF’
BE	counter for buffered correction bits for Huffman coding in the successive approximation process
BITS	16-byte list containing number of Huffman codes of each length
BP	pointer to compressed data
BPST	pointer to byte before start of entropy-coded segment
BR	counter for buffered correction bits for Huffman coding in the successive approximation process
Bx	byte modified by a carry-over
C	value of bit stream in code register
C_i	component identifier for frame
C_u	horizontal frequency dependent scaling factor in DCT
C_v	vertical frequency dependent scaling factor in DCT
CE	conditional exchange
C-low	low order 16 bits of the arithmetic decoder code register
Cm_i	*i*th 8-bit parameter in COM segment
CNT	bit counter in NEXTBYTE procedure
CODE	Huffman code value
CODESIZE(V)	code size for symbol V
COM	comment marker
Cs	conditioning table value
Cs_i	component identifier for scan
CT	renormalization shift counter
Cx	high order 16 bits of arithmetic decoder code register
CX	conditional exchange
d_{ji}	data unit from horizontal position i, vertical position j
d_{ji}^k	d_{ji} for component k
D	decision decoded

Da	in DC coding, the DC difference coded for the previous block from the same component; in lossless coding, the difference coded for the sample immediately to the left
DAC	define-arithmetic-coding-conditioning marker
Db	the difference coded for the sample immediately above
DC	DC DCT coefficient
DC_i	DC coefficient for *i*th block in component
DC_k	*k*th DC value used in prediction of AC coefficients
DHP	define hierarchical progression marker
DHT	define-Huffman-tables marker
DIFF	difference between quantized DC and prediction
DNL	define-number-of-lines marker
DQT	define-quantization-tables marker
DRI	define restart interval marker
E	exponent in magnitude category upper bound
EC	event counter
ECS	entropy-coded segment
ECS_i	*i*th entropy-coded segment
Eh	horizontal expansion parameter in EXP segment
EHUFCO	Huffman code table for encoder
EHUFSI	encoder table of Huffman code sizes
EOB	end-of-block for sequential; end-of-band for progressive
EOBn	run length category for EOB runs
EOBx	position of EOB in previous successive approximation scan
EOB0, EOB1, ..., EOB14	run length categories for EOB runs
EOI	end-of-image marker
Ev	vertical expansion parameter in EXP segment
EXP	expand reference components marker
FREQ(V)	frequency of occurrence of symbol V
H_i	horizontal sampling factor for *i*th component
H_{max}	largest horizontal sampling factor
HUFFCODE	list of Huffman codes corresponding to lengths in HUFFSIZE
HUFFSIZE	list of code lengths
HUFFVAL	list of values assigned to each Huffman code
i	subscript index
I	integer variable
Index(S)	index to probability estimation state machine table for context index S
j	subscript index
J	integer variable

JPG	marker reserved for JPEG extensions
JPG_n	marker reserved for JPEG extensions
k	subscript index
K	integer variable
Kmin	index of 1st AC coefficient in band (1 for sequential DCT)
Kx	conditioning parameter for AC arithmetic coding model
L	DC and lossless coding conditioning lower bound parameter
L_i	element in BITS list in DHT segment
$L_i(t)$	element in BITS list in the DHT segment for Huffman table t
La	length of parameters in APP_n segment
LASTK	largest value of K
Lc	length of parameters in COM segment
Ld	length of parameters in DNL segment
Le	length of parameters in EXP segment
Lf	length of frame header parameters
Lh	length of parameters in DHT segment
Lp	length of parameters in DAC segment
LPS	less probable symbol (in arithmetic coding)
Lq	length of parameters in DQT segment
Lr	length of parameters in DRI segment
Ls	length of scan header parameters
LSB	least significant bit
m	modulo 8 counter for RST_m marker
m_t	number of $V_{i,j}$ parameters for Huffman table t
M	bit mask used in coding magnitude of V
Mn	*n*th statistics bin for coding magnitude bit pattern category
MAXCODE	table with maximum value of Huffman code for each code length
MCU	minimum coded unit
MCU_i	*i*th MCU
MCUR	number of MCU required to make up one MCU-row
MINCODE	table with minimum value of Huffman code for each code length
MPS	more probable symbol (in arithmetic coding)
MPS(S)	more probable symbol for context-index S
MSB	most significant bit
M2, M3, M4, ... , M15	designation of context-indices for coding of magnitude bits in the arithmetic coding models
n	integer variable
N	data unit counter for MCU coding
N/A	not applicable

Nb	number of data units in MCU
Next_Index_LPS	new value of Index(S) after a LPS renormalization
Next_Index_MPS	new value of Index(S) after a MPS renormalization
Nf	number of components in frame
NL	number of lines defined in DNL segment
Ns	number of components in scan
OTHERS(V)	index to next symbol in chain
P	sample precision
Pq	quantizer precision parameter in DQT segment
Pq(t)	quantizer precision parameter in DQT segment for quantization table t
PRED	quantized DC coefficient from the most recently coded block of the component
Pt	point transform parameter
Px	calculated value of sample
Q_{ji}	quantizer value for coefficient AC_{ji}
Q_{vu}	quantization value for DCT coefficient S_{vu}
Q_{00}	quantizer value for DC coefficient
QAC_{ji}	quantized AC coefficient predicted from DC values
QDC_k	*k*th quantized DC value used in prediction of AC coefficients
Qe	LPS probability estimate
Qe(S)	LPS probability estimate for context index S
Qk	*k*th element of 64 quantization elements in DQT segment
r_{vu}	reconstructed image sample
R	length of run of zero amplitude AC coefficients
R_{vu}	dequantized DCT coefficient
Ra	reconstructed sample value
Rb	reconstructed sample value
Rc	reconstructed sample value
Rd	rounding in prediction calculation
RES	reserved markers
Ri	restart interval in DRI segment
RRRR	4-bit value of run length of zero AC coefficients
RS	composite value used in Huffman coding of AC coefficients
RST_m	restart marker number m
s_{yx}	reconstructed value from IDCT
S	context index
S_{vu}	DCT coefficient at horizontal frequency u, vertical frequency v

SC	context-index for coding of correction bit in successive approximation coding
Se	end of spectral selection band in zig-zag sequence
SE	context-index for coding of end-of-block or end-of-band
SI	Huffman code size
SIGN	1 if decoded sense of sign is negative and 0 if decoded sense of sign is positive
SIZE	length of a Huffman code
SLL	shift left logical operation
SLL α β	logical shift left of α by β bits
SN	context-index for coding of first magnitude category when V is negative
SOF_0	baseline DCT process frame marker
SOF_1	extended sequential DCT frame marker, Huffman coding
SOF_2	progressive DCT frame marker, Huffman coding
SOF_3	lossless process frame marker, Huffman coding
SOF_5	differential sequential DCT frame marker, Huffman coding
SOF_6	differential progressive DCT frame marker, Huffman coding
SOF_7	differential lossless process frame marker, Huffman coding
SOF_9	sequential DCT frame marker, arithmetic coding
SOF_{10}	progressive DCT frame marker, arithmetic coding
SOF_{11}	lossless process frame marker, arithmetic coding
SOF_{13}	differential sequential DCT frame marker, arithmetic coding
SOF_{14}	differential progressive DCT frame marker, arithmetic coding
SOF_{15}	differential lossless process frame marker, arithmetic coding
SOI	start-of-image marker
SOS	start-of-scan marker
SP	context-index for coding of first magnitude category when V is positive
Sq_{vu}	quantized DCT coefficient
SRL	shift right logical operation
SRL α β	logical shift right of α by β bits
Ss	start of spectral selection band in zig-zag sequence
SS	context-index for coding of sign decision
SSSS	4-bit size category of DC difference or AC coefficient amplitude
ST	stack counter
Switch_MPS	parameter controlling inversion of sense of MPS
Sz	parameter used in coding magnitude of V
S0	context-index for coding of V = 0 decision
t	summation index for parameter limits computation
T	temporary variable

Ta_j	AC entropy table destination selector for *j*th component in scan
Tb	arithmetic conditioning table destination identifier
Tc	Huffman coding or arithmetic coding table class
Td_j	DC entropy table destination selector for *j*th component in scan
TEM	temporary marker
Th	Huffman table destination identifier in DHT segment
Tq	quantization table destination identifier in DQT segment
Tq_i	quantization table destination selector for *i*th component in frame
U	DC and lossless coding conditioning upper bound parameter
V	symbol or value being either encoded or decoded
V_i	vertical sampling factor for *i*th component
$V_{i,j}$	*j*th value for length *i* in HUFFVAL
V_{max}	largest vertical sampling factor
V_t	temporary variable
VALPTR	list of indices for first value in HUFFVAL for each code length
V1	symbol value
V2	symbol value
x_i	number of columns in *i*th component
X	number of samples per line in component with largest horizontal dimension
X_i	*i*th statistics bin for coding magnitude category decision
X1, X2, X3, ... , X15	designation of context-indices for coding of magnitude categories in the arithmetic coding models
XHUFCO	extended Huffman code table
XHUFSI	table of sizes of extended Huffman codes
X’values’	values within the quotes are hexadecimal
y_i	number of lines in *i*th component
Y	number of lines in component with largest vertical dimension
ZRL	value in HUFFVAL assigned to run of 16 zero coefficients
ZZ(K)	*K*th element in zig-zag sequence of quantized DCT coefficients
ZZ(0)	quantized DC coefficient in zig-zag sequence order

4 General

The purpose of this clause is to give an informative overview of the elements specified in this Specification. Another purpose is to introduce many of the terms which are defined in clause 3. These terms are printed in *italics* upon first usage in this clause.

4.1 Elements specified in this Specification

There are three elements specified in this Specification:

a) An *encoder* is an embodiment of an *encoding process*. As shown in Figure 1, an encoder takes as input *digital source image data* and *table specifications*, and by means of a specified set of *procedures* generates as output *compressed image data*.

b) A *decoder* is an embodiment of a *decoding process*. As shown in Figure 2, a decoder takes as input compressed image data and table specifications, and by means of a specified set of procedures generates as output *digital reconstructed image data*.

c) The *interchange format*, shown in Figure 3, is a compressed image data representation which includes all table specifications used in the encoding process. The interchange format is for exchange between *application environments*.

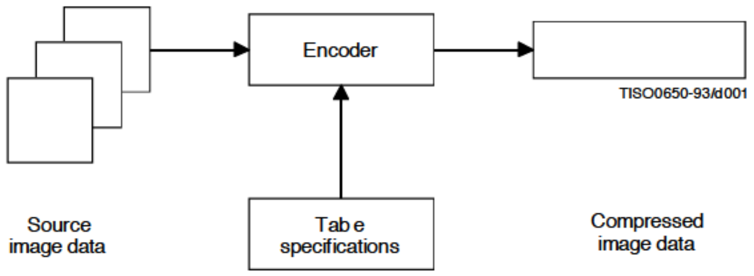


Figure 1 – Encoder

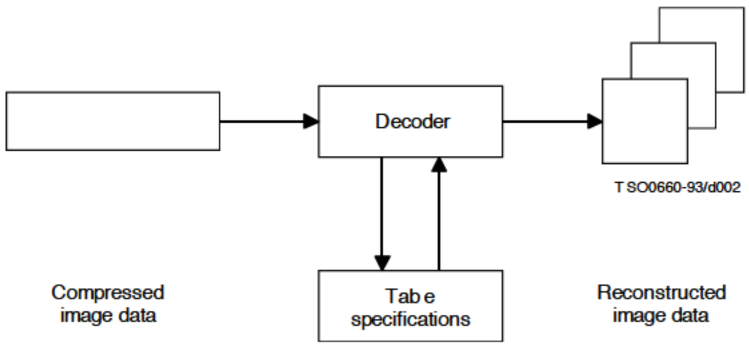


Figure 2 – Decoder

Figures 1 and 2 illustrate the general case for which the *continuous-tone* source and reconstructed image data consist of multiple *components*. (A *colour* image consists of multiple components; a *grayscale* image consists only of a single component.) A significant portion of this Specification is concerned with how to handle multiple-component images in a flexible, application-independent way.

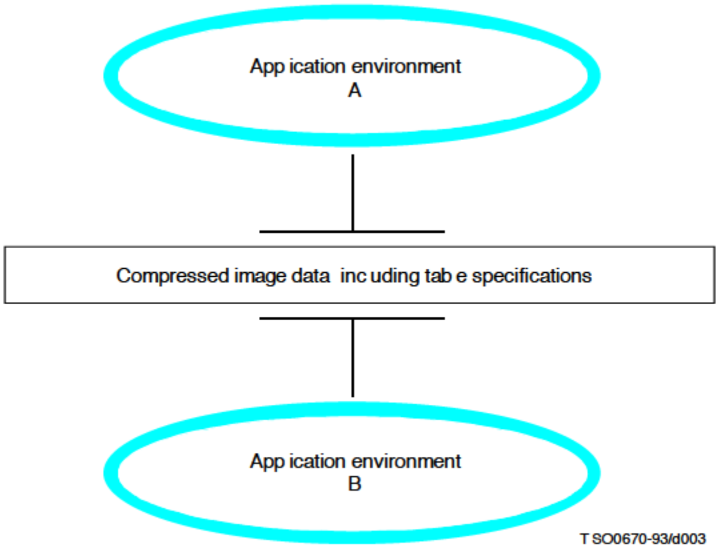

App ication environment A
Compressed image data inc uding tab e specifications
App ication environment B
T SO0670-93/d003

Figure 3 – Interchange format for compressed image data

These figures are also meant to show that the same tables specified for an encoder to use to compress a particular image must be provided to a decoder to reconstruct that image. However, this Specification does not specify how applications should associate tables with compressed image data, nor how they should represent source image data generally within their specific environments.

Consequently, this Specification also specifies the interchange format shown in Figure 3, in which table specifications are included within compressed image data. An image compressed with a specified encoding process within one application environment, A, is passed to a different environment, B, by means of the interchange format. The interchange format does not specify a complete coded image representation. Application-dependent information, e.g. colour space, is outside the scope of this Specification.

4.2 Lossy and lossless compression

This Specification specifies two *classes* of encoding and decoding processes, *lossy* and *lossless* processes. Those based on the *discrete cosine transform* (DCT) are lossy, thereby allowing substantial *compression* to be achieved while producing a reconstructed image with high visual fidelity to the encoder's source image.

The simplest DCT-based *coding process* is referred to as the *baseline sequential* process. It provides a capability which is sufficient for many applications. There are additional DCT-based processes which extend the baseline sequential process to a broader range of applications. In any decoder using *extended DCT-based decoding processes*, the baseline decoding process is required to be present in order to provide a default decoding capability.

The second class of coding processes is not based upon the DCT and is provided to meet the needs of applications requiring lossless compression. These lossless encoding and decoding processes are used independently of any of the DCT-based processes.

A table summarizing the relationship among these lossy and lossless coding processes is included in 4.11.

The amount of compression provided by any of the various processes is dependent on the characteristics of the particular image being compressed, as well as on the picture quality desired by the application and the desired speed of compression and decompression.

4.3 DCT-based coding

Figure 4 shows the main procedures for all encoding processes based on the DCT. It illustrates the special case of a single-component image; this is an appropriate simplification for overview purposes, because all processes specified in this Specification operate on each image component independently.

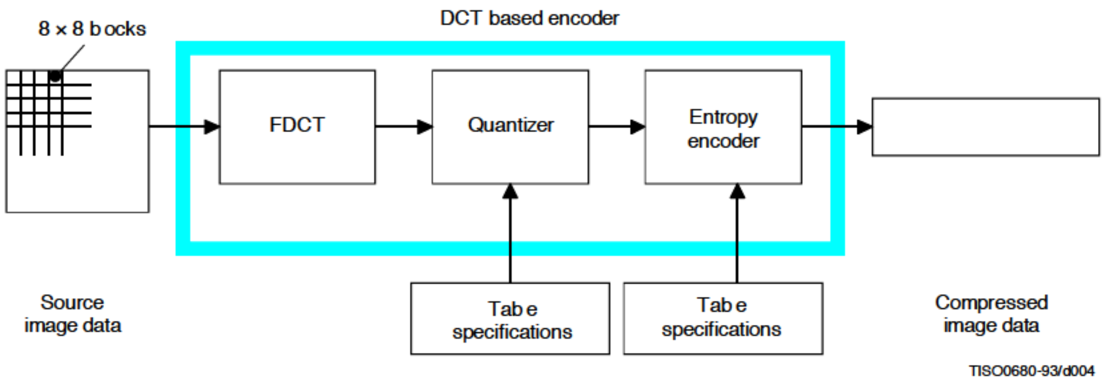


Figure 4 – DCT-based encoder simplified diagram

In the encoding process the input component's *samples* are grouped into 8 × 8 *blocks*, and each block is transformed by the *forward DCT* (FDCT) into a set of 64 values referred to as *DCT coefficients*. One of these values is referred to as the *DC coefficient* and the other 63 as the *AC coefficients*.

Each of the 64 coefficients is then *quantized* using one of 64 corresponding values from a *quantization table* (determined by one of the table specifications shown in Figure 4). No default values for quantization tables are specified in this Specification; applications may specify values which customize picture quality for their particular image characteristics, display devices, and viewing conditions.

After quantization, the DC coefficient and the 63 AC coefficients are prepared for *entropy encoding*, as shown in Figure 5. The previous quantized DC coefficient is used to predict the current quantized DC coefficient, and the difference is encoded. The 63 quantized AC coefficients undergo no such differential encoding, but are converted into a one-dimensional *zig-zag sequence*, as shown in Figure 5.

The quantized coefficients are then passed to an entropy encoding procedure which compresses the data further. One of two entropy coding procedures can be used, as described in 4.6. If *Huffman encoding* is used, *Huffman table* specifications must be provided to the encoder. If *arithmetic encoding* is used, arithmetic coding *conditioning table* specifications may be provided, otherwise the default conditioning table specifications shall be used.

Figure 6 shows the main procedures for all DCT-based decoding processes. Each step shown performs essentially the inverse of its corresponding main procedure within the encoder. The entropy decoder decodes the zig-zag sequence of quantized DCT coefficients. After *dequantization* the DCT coefficients are transformed to an 8 × 8 block of samples by the *inverse DCT* (IDCT).

4.4 Lossless coding

Figure 7 shows the main procedures for the lossless encoding processes. A *predictor* combines the reconstructed values of up to three neighbourhood samples at positions a, b, and c to form a prediction of the sample at position x as shown in Figure 8. This prediction is then subtracted from the actual value of the sample at position x, and the difference is losslessly entropy-coded by either Huffman or arithmetic coding.

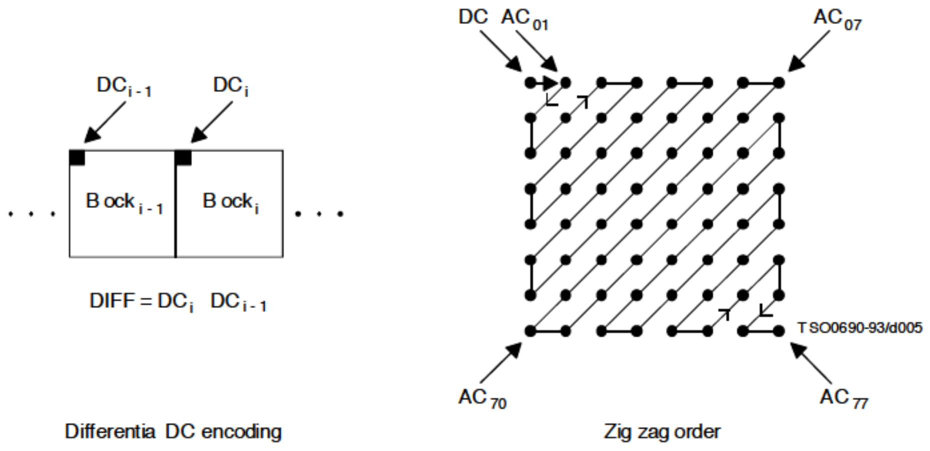


Figure 5 – Preparation of quantized coefficients for entropy encoding

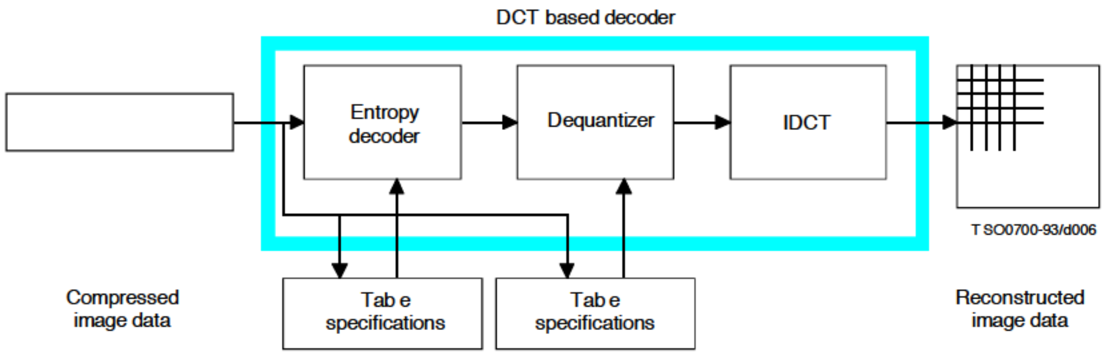


Figure 6 – DCT-based decoder simplified diagram

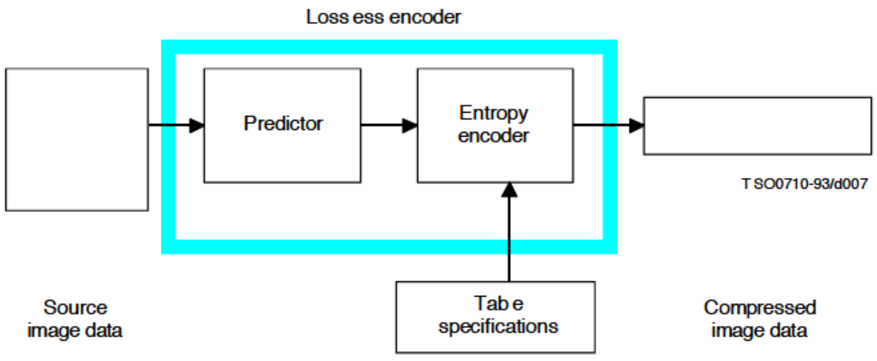


Figure 7 – Lossless encoder simplified diagram

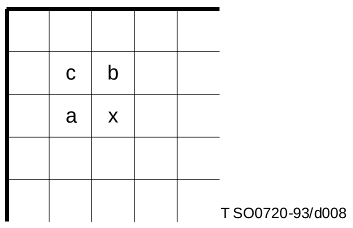


Figure 8 – 3-sample prediction neighbourhood

This encoding process may also be used in a slightly modified way, whereby the *precision* of the input samples is reduced by one or more bits prior to the lossless coding. This achieves higher compression than the lossless process (but lower compression than the DCT-based processes for equivalent visual fidelity), and limits the reconstructed image's worst-case sample error to the amount of input precision reduction.

4.5 Modes of operation

There are four distinct *modes of operation* under which the various coding processes are defined: *sequential DCT-based, progressive DCT-based,* lossless, and *hierarchical.* (Implementations are not required to provide all of these.) The lossless mode of operation was described in 4.4. The other modes of operation are compared as follows.

For the sequential DCT-based mode, 8×8 sample blocks are typically input block by block from left to right, and block-row by block-row from top to bottom. After a block has been transformed by the forward DCT, quantized and prepared for entropy encoding, all 64 of its quantized DCT coefficients can be immediately entropy encoded and output as part of the compressed image data (as was described in 4.3), thereby minimizing coefficient storage requirements.

For the progressive DCT-based mode, 8×8 blocks are also typically encoded in the same order, but in multiple *scans* through the image. This is accomplished by adding an image-sized coefficient memory buffer (not shown in Figure 4) between the quantizer and the entropy encoder. As each block is transformed by the forward DCT and quantized, its coefficients are stored in the buffer. The DCT coefficients in the buffer are then partially encoded in each of multiple scans. The typical sequence of image presentation at the output of the decoder for sequential versus progressive modes of operation is shown in Figure 9.

There are two procedures by which the quantized coefficients in the buffer may be partially encoded within a scan. First, only a specified *band* of coefficients from the zig-zag sequence need be encoded. This procedure is called *spectral selection*, because each band typically contains coefficients which occupy a lower or higher part of the *frequency* spectrum for that 8×8 block. Secondly, the coefficients within the current band need not be encoded to their full (quantized) accuracy within each scan. Upon a coefficient's first encoding, a specified number of most significant bits is encoded first. In subsequent scans, the less significant bits are then encoded. This procedure is called *successive approximation*. Either procedure may be used separately, or they may be mixed in flexible combinations.

In hierarchical mode, an image is encoded as a sequence of *frames*. These frames provide *reference reconstructed components* which are usually needed for prediction in subsequent frames. Except for the first frame for a given component, *differential frames* encode the difference between source components and reference reconstructed components. The coding of the differences may be done using only DCT-based processes, only lossless processes, or DCT-based processes with a final lossless process for each component. *Downsampling* and *upsampling filters* may be used to provide a pyramid of spatial resolutions as shown in Figure 10. Alternatively, the hierarchical mode can be used to improve the quality of the reconstructed components at a given spatial resolution.

Hierarchical mode offers a progressive presentation similar to the progressive DCT-based mode but is useful in environments which have multi-resolution requirements. Hierarchical mode also offers the capability of progressive coding to a final lossless stage.

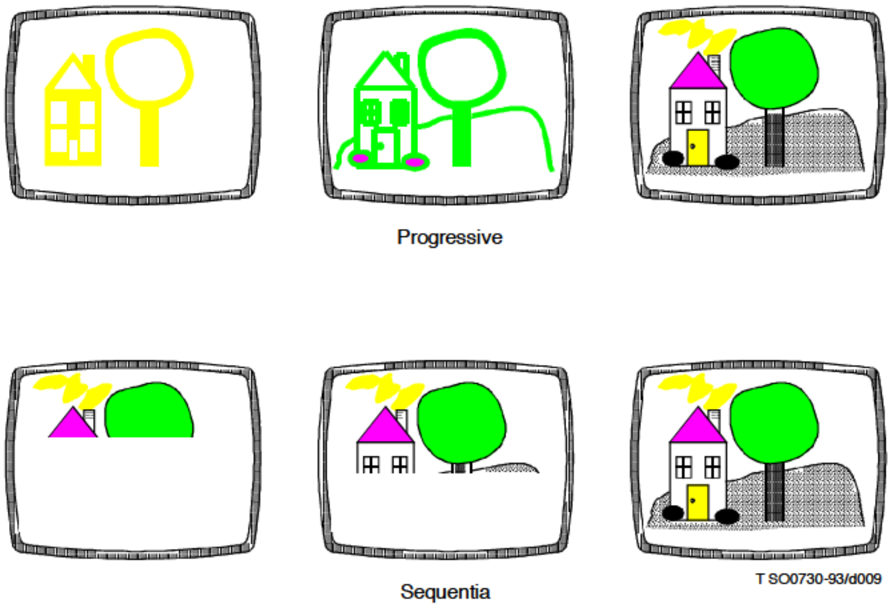


Figure 9 – Progressive versus sequential presentation

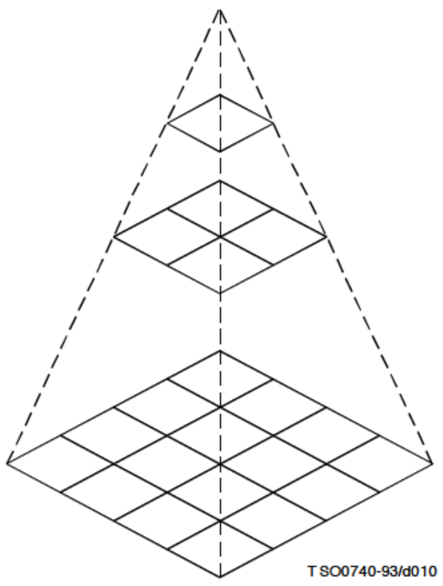


Figure 10 – Hierarchical multi-resolution encoding

4.6 Entropy coding alternatives

Two alternative entropy coding procedures are specified: Huffman coding and arithmetic coding. Huffman coding procedures use Huffman tables, determined by one of the table specifications shown in Figures 1 and 2. Arithmetic coding procedures use arithmetic coding conditioning tables, which may also be determined by a table specification. No default values for Huffman tables are specified, so that applications may choose tables appropriate for their own environments. Default tables are defined for the arithmetic coding conditioning.

The baseline sequential process uses Huffman coding, while the extended DCT-based and lossless processes may use either Huffman or arithmetic coding.

4.7 Sample precision

For DCT-based processes, two alternative sample precisions are specified: either 8 bits or 12 bits per sample. Applications which use samples with other precisions can use either 8-bit or 12-bit precision by shifting their source image samples appropriately. The baseline process uses only 8-bit precision. DCT-based implementations which handle 12-bit source image samples are likely to need greater computational resources than those which handle only 8-bit source images. Consequently in this Specification separate normative requirements are defined for 8-bit and 12-bit DCT-based processes.

For lossless processes the sample precision is specified to be from 2 to 16 bits.

4.8 Multiple-component control

Subclauses 4.3 and 4.4 give an overview of one major part of the encoding and decoding processes – those which operate on the sample values in order to achieve compression. There is another major part as well – the procedures which control the order in which the image data from multiple components are processed to create the compressed data, and which ensure that the proper set of table data is applied to the proper *data units* in the image. (A data unit is a sample for lossless processes and an 8×8 block of samples for DCT-based processes.)

4.8.1 Interleaving multiple components

Figure 11 shows an example of how an encoding process selects between multiple source image components as well as multiple sets of table data, when performing its encoding procedures. The source image in this example consists of the three components A, B and C, and there are two sets of table specifications. (This simplified view does not distinguish between the quantization tables and entropy coding tables.)

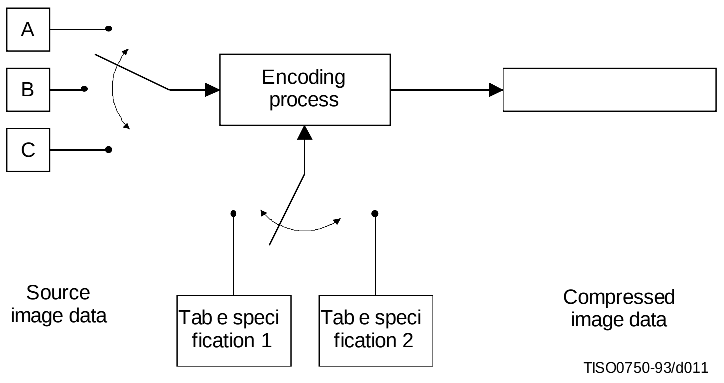


Figure 11 – Component-interleave and table-switching control

In sequential mode, encoding is *non-interleaved* if the encoder compresses all image data units in component A before beginning component B, and then in turn all of B before C. Encoding is *interleaved* if the encoder compresses a data unit from A, a data unit from B, a data unit from C, then back to A, etc. These alternatives are illustrated in Figure 12, which shows a case in which all three image components have identical dimensions: X *columns* by Y lines, for a total of n data units each.

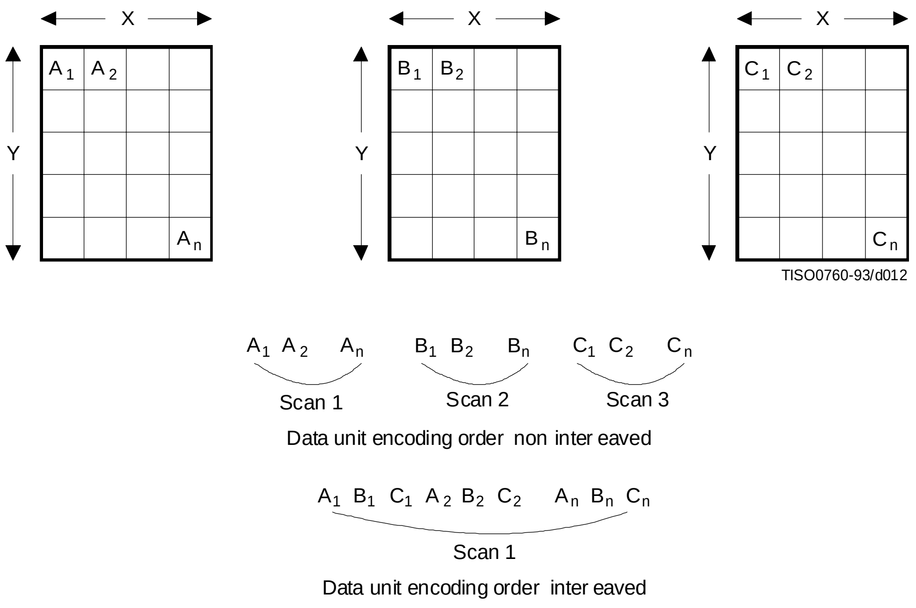


Figure 12 – Interleaved versus non-interleaved encoding order

These control procedures are also able to handle cases in which the source image components have different dimensions. Figure 13 shows a case in which two of the components, B and C, have half the number of horizontal samples relative to component A. In this case, two data units from A are interleaved with one each from B and C. Cases in which components of an image have more complex relationships, such as different horizontal and vertical dimensions, can be handled as well. (See Annex A.)

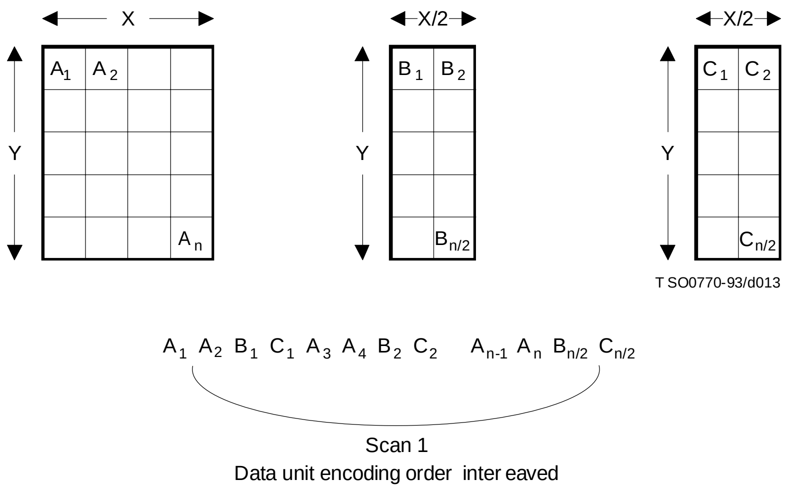

X
X/2
Y
A_1 A_2
A_n
B_1 B_2
$B_{n/2}$
C_1 C_2
$C_{n/2}$
T SO0770-93/d013
A_1 A_2 B_1 C_1 A_3 A_4 B_2 C_2 A_{n-1} A_n $B_{n/2}$ $C_{n/2}$
Scan 1
Data unit encoding order inter eaved

Figure 13 – Interleaved order for components with different dimensions

4.8.2 Minimum coded unit

Related to the concepts of multiple-component interleave is the *minimum coded unit* (MCU). If the compressed image data is non-interleaved, the MCU is defined to be one data unit. For example, in Figure 12 the MCU for the non-interleaved case is a single data unit. If the compressed data is interleaved, the MCU contains one or more data units from each component. For the interleaved case in Figure 12, the (first) MCU consists of the three interleaved data units A_1, B_1, C_1. In the example of Figure 13, the (first) MCU consists of the four data units A_1, A_2 , B_1, C_1.

4.9 Structure of compressed data

Figures 1, 2, and 3 all illustrate slightly different views of compressed image data. Figure 1 shows this data as the output of an encoding process, Figure 2 shows it as the input to a decoding process, and Figure 3 shows compressed image data in the interchange format, at the interface between applications.

Compressed image data are described by a uniform structure and set of *parameters* for both classes of encoding processes (lossy or lossless), and for all modes of operation (sequential, progressive, lossless, and hierarchical). The various parts of the compressed image data are identified by special two-byte codes called *markers*. Some markers are followed by particular sequences of parameters, as in the case of table specifications, *frame header*, or *scan header*. Others are used without parameters for functions such as marking the start-of-image and end-of-image. When a marker is associated with a particular sequence of parameters, the marker and its parameters comprise a *marker segment*.

The data created by the entropy encoder are also segmented, and one particular marker – *the restart marker* – is used to isolate *entropy-coded data segments*. The encoder outputs the restart markers, intermixed with the entropy-coded data, at regular *restart intervals* of the source image data. Restart markers can be identified without having to decode the compressed data to find them. Because they can be independently decoded, they have application-specific uses, such as parallel encoding or decoding, isolation of data corruptions, and semi-random access of entropy-coded segments.

There are three compressed data formats:

a) the interchange format;

b) the *abbreviated format* for compressed image data;

c) the abbreviated format for table-specification data.

4.9.1 Interchange format

In addition to certain required marker segments and the entropy-coded segments, the interchange format shall include the marker segments for all quantization and entropy-coding table specifications needed by the decoding process. This guarantees that a compressed image can cross the boundary between application environments, regardless of how each environment internally associates tables with compressed image data.

4.9.2 Abbreviated format for compressed image data

The abbreviated format for compressed image data is identical to the interchange format, except that it does not include all tables required for decoding. (It may include some of them.) This format is intended for use within applications where alternative mechanisms are available for supplying some or all of the table-specification data needed for decoding.

4.9.3 Abbreviated format for table-specification data

This format contains only table-specification data. It is a means by which the application may install in the decoder the tables required to subsequently reconstruct one or more images.

4.10 Image, frame, and scan

Compressed image data consists of only one image. An image contains only one frame in the cases of sequential and progressive coding processes; an image contains multiple frames for the hierarchical mode.

A frame contains one or more scans. For sequential processes, a scan contains a complete encoding of one or more image components. In Figures 12 and 13, the frame consists of three scans when non-interleaved, and one scan if all three components are interleaved together. The frame could also consist of two scans: one with a non-interleaved component, the other with two components interleaved.

For progressive processes, a scan contains a partial encoding of all data units from one or more image components. Components shall not be interleaved in progressive mode, except for the DC coefficients in the first scan for each component of a progressive frame.

4.11 Summary of coding processes

Table 1 provides a summary of the essential characteristics of the various coding processes specified in this Specification. The full specification of these processes is contained in Annexes F, G, H, and J.

Table 1 – Summary: Essential characteristics of coding processes

Baseline process (required for all DCT-based decoders)
• DCT-based process • Source image: 8-bit samples within each component • Sequential • Huffman coding: 2 AC and 2 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Extended DCT-based processes
• DCT-based process • Source image: 8-bit or 12-bit samples • Sequential or progressive • Huffman or arithmetic coding: 4 AC and 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Lossless processes
• Predictive process (not DCT-based) • Source image: P-bit samples ($2 \leq P \leq 16$) • Sequential • Huffman or arithmetic coding: 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Hierarchical processes
• Multiple frames (non-differential and differential) • Uses extended DCT-based or lossless processes • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

5 Interchange format requirements

The interchange format is the coded representation of compressed image data for exchange between application environments.

The interchange format requirements are that any compressed image data represented in interchange format shall comply with the syntax and code assignments appropriate for the decoding process selected, as specified in Annex B.

Tests for whether compressed image data comply with these requirements are specified in Part 2 of this Specification.

6 Encoder requirements

An encoding process converts source image data to compressed image data. Each of Annexes F, G, H, and J specifies a number of distinct encoding processes for its particular mode of operation.

An encoder is an embodiment of one (or more) of the encoding processes specified in Annexes F, G, H, or J. In order to comply with this Specification, an encoder shall satisfy at least one of the following two requirements.

An encoder shall

a) with appropriate accuracy, convert source image data to compressed image data which comply with the interchange format syntax specified in Annex B for the encoding process(es) embodied by the encoder;

b) with appropriate accuracy, convert source image data to compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex B for the encoding process(es) embodied by the encoder.

For each of the encoding processes specified in Annexes F, G, H, and J, the compliance tests for the above requirements are specified in Part 2 of this Specification.

NOTE – There is **no requirement** in this Specification that any encoder which embodies one of the encoding processes specified in Annexes F, G, H, or J shall be able to operate for all ranges of the parameters which are allowed for that process. An encoder is only required to meet the compliance tests specified in Part 2, and to generate the compressed data format according to Annex B for those parameter values which it does use.

7 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Each of Annexes F, G, H, and J specifies a number of distinct decoding processes for its particular mode of operation.

A decoder is an embodiment of one (or more) of the decoding processes specified in Annexes F, G, H, or J. In order to comply with this Specification, a decoder shall satisfy all three of the following requirements.

A decoder shall

a) with appropriate accuracy, convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which comply with the interchange format syntax specified in Annex B for the decoding process(es) embodied by the decoder;

b) accept and properly store any table-specification data which comply with the abbreviated format for table-specification data syntax specified in Annex B for the decoding process(es) embodied by the decoder;

c) with appropriate accuracy, convert to reconstructed image data any compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex B for the decoding process(es) embodied by the decoder, provided that the table-specification data required for decoding the compressed image data has previously been installed into the decoder.

Additionally, any DCT-based decoder, if it embodies any DCT-based decoding process other than baseline sequential, shall also embody the baseline sequential decoding process.

For each of the decoding processes specified in Annexes F, G, H, and J, the compliance tests for the above requirements are specified in Part 2 of this Specification.

Annex A

Mathematical definitions

(This annex forms an integral part of this Recommendation | International Standard)

A.1 Source image

Source images to which the encoding processes specified in this Specification can be applied are defined in this annex.

A.1.1 Dimensions and sampling factors

As shown in Figure A.1, a source image is defined to consist of Nf components. Each component, with unique identifier C_i, is defined to consist of a rectangular array of samples of x_i columns by y_i lines. The component dimensions are derived from two parameters, X and Y, where X is the maximum of the x_i values and Y is the maximum of the y_i values for all components in the frame. For each component, sampling factors H_i and V_i are defined relating component dimensions x_i and y_i to maximum dimensions X and Y, according to the following expressions:

$$x_i = \left\lceil X \times \frac{H_i}{H_{max}} \right\rceil \text{ and } y_i \left\lceil Y \times \frac{V_i}{V_{max}} \right\rceil,$$

where H_{max} and V_{max} are the maximum sampling factors for all components in the frame, and $\lceil\ \rceil$ is the ceiling function.

As an example, consider an image having 3 components with maximum dimensions of 512 lines and 512 samples per line, and with the following sampling factors:

Component 0	$H_0 = 4, \quad V_0 = 1$
Component 1	$H_1 = 2, \quad V_1 = 2$
Component 2	$H_2 = 1, \quad V_2 = 1$

Then $X = 512$, $Y = 512$, $H_{max} = 4$, $V_{max} = 2$, and x_i and y_i for each component are

Component 0	$x_0 = 512, \quad y_0 = 256$
Component 1	$x_1 = 256, \quad y_1 = 512$
Component 2	$x_2 = 128, \quad y_2 = 256$

NOTE – The X, Y, H_i, and V_i parameters are contained in the frame header of the compressed image data (see B.2.2), whereas the individual component dimensions x_i and y_i are derived by the decoder. Source images with x_i and y_i dimensions which do not satisfy the expressions above cannot be properly reconstructed.

A.1.2 Sample precision

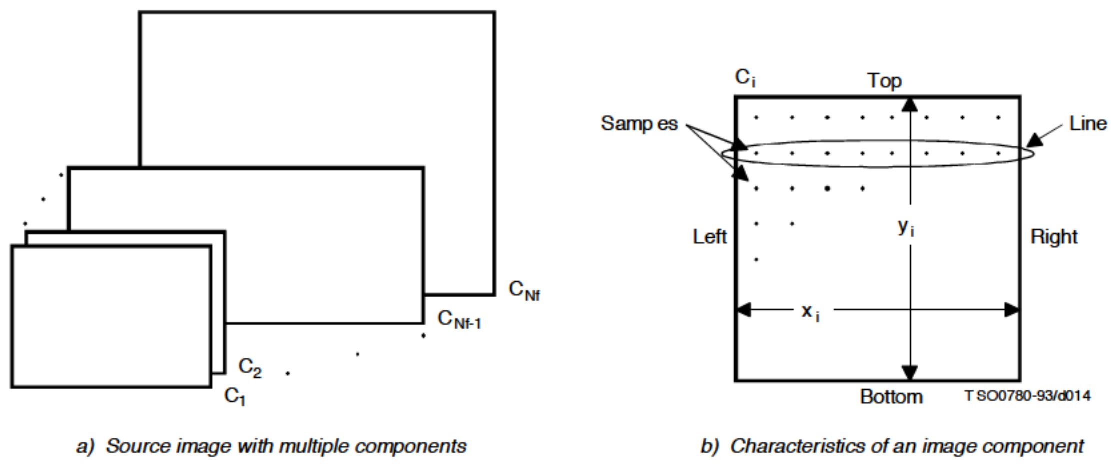
A sample is an integer with precision P bits, with any value in the range 0 through 2^{P-1}. All samples of all components within an image shall have the same precision P. Restrictions on the value of P depend on the mode of operation, as specified in B.2 to B.7.

A.1.3 Data unit

A data unit is a sample in lossless processes and an 8×8 block of contiguous samples in DCT-based processes. The left-most 8 samples of each of the top-most 8 rows in the component shall always be the top-left-most block. With this top-left-most block as the reference, the component is partitioned into contiguous data units to the right and to the bottom (as shown in Figure A.4).

A.1.4 Orientation

Figure A.1 indicates the orientation of an image component by the terms top, bottom, left, and right. The order by which the data units of an image component are input to the compression encoding procedures is defined to be left-to-right and top-to-bottom within the component. (This ordering is precisely defined in A.2.) Applications determine which edges of a source image are defined as top, bottom, left, and right.



C₁
C₂
C_Nf-1
C_Nf
a) Source image with multiple components
Cᵢ
Top
Samp es
Line
Left
Right
yᵢ
xᵢ
Bottom
T SO0780-93/d014
b) Characteristics of an image component

Figure A.1 – Source image characteristics

A.2 Order of source image data encoding

The scan header (see B.2.3) specifies the order by which source image data units shall be encoded and placed within the compressed image data. For a given scan, if the scan header parameter Ns = 1, then data from only one source component – the component specified by parameter Cs_1 – shall be present within the scan. This data is non-interleaved by definition. If Ns > 1, then data from the Ns components Cs_1 through Cs_{Ns} shall be present within the scan. This data shall always be interleaved. The order of components in a scan shall be according to the order specified in the frame header.

The ordering of data units and the construction of minimum coded units (MCU) is defined as follows.

A.2.1 Minimum coded unit (MCU)

For non-interleaved data the MCU is one data unit. For interleaved data the MCU is the sequence of data units defined by the sampling factors of the components in the scan.

A.2.2 Non-interleaved order (Ns = 1)

When Ns = 1 (where Ns is the number of components in a scan), the order of data units within a scan shall be left-to-right and top-to-bottom, as shown in Figure A.2. This ordering applies whenever Ns = 1, regardless of the values of H_1 and V_1.

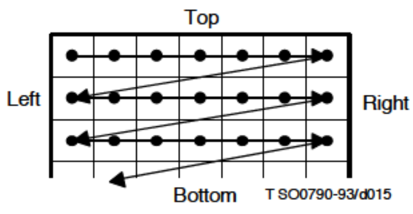
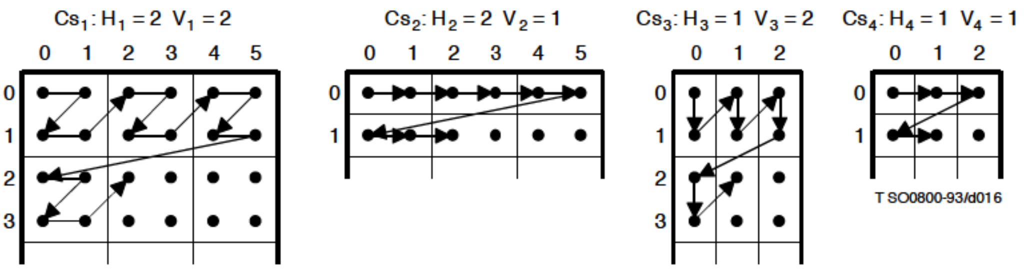


Figure A.2 – Non-interleaved data ordering

A.2.3 Interleaved order (Ns > 1)

When Ns > 1, each scan component Cs_i is partitioned into small rectangular arrays of H_k horizontal data units by V_k vertical data units. The subscripts k indicate that H_k and V_k are from the position in the frame header component-specification for which $C_k = Cs_i$. Within each H_k by V_k array, data units are ordered from left-to-right and top-to-bottom. The arrays in turn are ordered from left-to-right and top-to-bottom within each component.

As shown in the example of Figure A.3, Ns = 4, and MCU_1 consists of data units taken first from the top-left-most region of Cs_1, followed by data units from the corresponding region of Cs_2, then from Cs_3 and then from Cs_4. MCU_2 follows the same ordering for data taken from the next region to the right for the four components.



$$\begin{aligned}
MCU_1 &= d^1_{00}\ d^1_{01}\ d^1_{10}\ d^1_{11}\quad d^2_{00}\ d^2_{01}\quad d^3_{00}\ d^3_{10}\quad d^4_{00}\\
MCU_2 &= d^1_{02}\ d^1_{03}\ d^1_{12}\ d^1_{13}\quad d^2_{02}\ d^2_{03}\quad d^3_{01}\ d^3_{11}\quad d^4_{01}\\
MCU_3 &= d^1_{04}\ d^1_{05}\ d^1_{14}\ d^1_{15}\quad d^2_{04}\ d^2_{05}\quad d^3_{02}\ d^3_{12}\quad d^4_{02}\\
MCU_4 &= \underbrace{d^1_{20}\ d^1_{21}\ d^1_{30}\ d^1_{31}}_{Cs_1 \text{ data units}}\quad \underbrace{d^2_{10}\ d^2_{11}}_{Cs_2}\quad \underbrace{d^3_{20}\ d^3_{30}}_{Cs_3}\quad \underbrace{d^4_{10}}_{Cs_4}
\end{aligned}$$

Figure A.3 – Interleaved data ordering example

A.2.4 Completion of partial MCU

For DCT-based processes the data unit is a block. If x_i is not a multiple of 8, the encoding process shall extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the encoding process shall also extend the number of samples by one or more additional blocks, if necessary, so that the number of blocks is an integer multiple of H_i. Similarly, if y_i is not a multiple of 8, the encoding process shall extend the number of lines to complete the bottom-most block-row. If the component is to be interleaved, the encoding process shall also extend the number of lines by one or more additional block-rows, if necessary, so that the number of block-rows is an integer multiple of V_i.

NOTE – It is recommended that any incomplete MCUs be completed by replication of the right-most column and the bottom line of each component.

For lossless processes the data unit is a sample. If the component is to be interleaved, the encoding process shall extend the number of samples, if necessary, so that the number is a multiple of H_i. Similarly, the encoding process shall extend the number of lines, if necessary, so that the number of lines is a multiple of V_i.

Any sample added by an encoding process to complete partial MCUs shall be removed by the decoding process.

A.3 DCT compression

A.3.1 Level shift

Before a non-differential frame encoding process computes the FDCT for a block of source image samples, the samples shall be level shifted to a signed representation by subtracting 2^{P-1}, where P is the precision parameter specified in B.2.2. Thus, when P = 8, the level shift is by 128; when P = 12, the level shift is by 2048.

After a non-differential frame decoding process computes the IDCT and produces a block of reconstructed image samples, an inverse level shift shall restore the samples to the unsigned representation by adding 2^{P-1} and clamping the results to the range 0 to 2^{P-1}.

A.3.2 Orientation of samples for FDCT computation

Figure A.4 shows an image component which has been partitioned into 8×8 blocks for the FDCT computations. Figure A.4 also defines the orientation of the samples within a block by showing the indices used in the FDCT equation of A.3.3.

The definitions of block partitioning and sample orientation also apply to any DCT decoding process and the output reconstructed image. Any sample added by an encoding process to complete partial MCUs shall be removed by the decoding process.

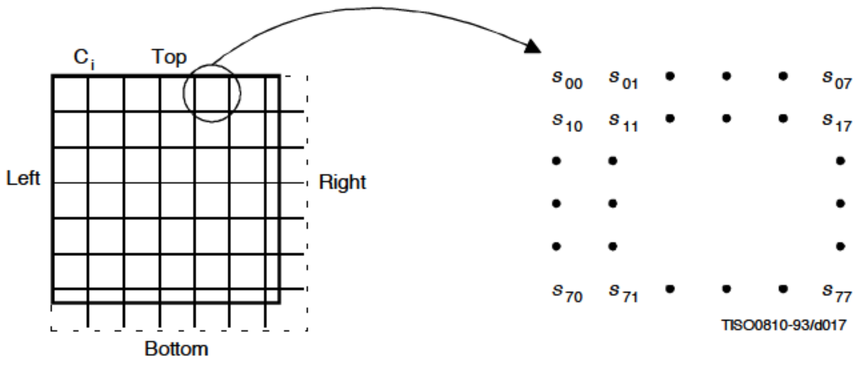

C_i
Top
Left
Right
Bottom
s_{00} s_{01} s_{07}
s_{10} s_{11} s_{17}
s_{70} s_{71} s_{77}
TISO0810-93/d017

Figure A.4 – Partition and orientation of 8 x 8 sample blocks

A.3.3 FDCT and IDCT (informative)

The following equations specify the ideal functional definition of the FDCT and the IDCT.

NOTE – These equations contain terms which cannot be represented with perfect accuracy by any real implementation. The accuracy requirements for the combined FDCT and quantization procedures are specified in Part 2 of this Specification. The accuracy requirements for the combined dequantization and IDCT procedures are also specified in Part 2 of this Specification.

$$\text{FDCT:} \qquad S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

$$\text{IDCT:} \qquad s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{vu} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

where

$$C_u, C_v = 1/\sqrt{2} \text{ for } u, v = 0$$

$$C_u, C_v = 1 \text{ otherwise}$$

otherwise.

A.3.4 DCT coefficient quantization (informative) and dequantization (normative)

After the FDCT is computed for a block, each of the 64 resulting DCT coefficients is quantized by a uniform quantizer. The quantizer step size for each coefficient S_{vu} is the value of the corresponding element Q_{vu} from the quantization table specified by the frame parameter Tq_i (see B.2.2).

The uniform quantizer is defined by the following equation. Rounding is to the nearest integer:

$$Sq_{vu} = round\left(\frac{S_{vu}}{Q_{vu}}\right)$$

Sq_{vu} is the quantized DCT coefficient, normalized by the quantizer step size.

NOTE – This equation contains a term which may not be represented with perfect accuracy by any real implementation. The accuracy requirements for the combined FDCT and quantization procedures are specified in Part 2 of this Specification.

At the decoder, this normalization is removed by the following equation, which defines dequantization:

$$R_{vu} = Sq_{vu} \times Q_{vu}$$

NOTE – Depending on the rounding used in quantization, it is possible that the dequantized coefficient may be outside the expected range.

The relationship among samples, DCT coefficients, and quantization is illustrated in Figure A.5.

A.3.5 Differential DC encoding

After quantization, and in preparation for entropy encoding, the quantized DC coefficient Sq_{00} is treated separately from the 63 quantized AC coefficients. The value that shall be encoded is the difference (DIFF) between the quantized DC coefficient of the current block (DC_i which is also designated as Sq_{00}) and that of the previous block of the same component (PRED):

$$DIFF = DC_i - PRED$$

A.3.6 Zig-zag sequence

After quantization, and in preparation for entropy encoding, the quantized AC coefficients are converted to the zig-zag sequence. The quantized DC coefficient (coefficient zero in the array) is treated separately, as defined in A.3.5. The zig-zag sequence is specified in Figure A.6.

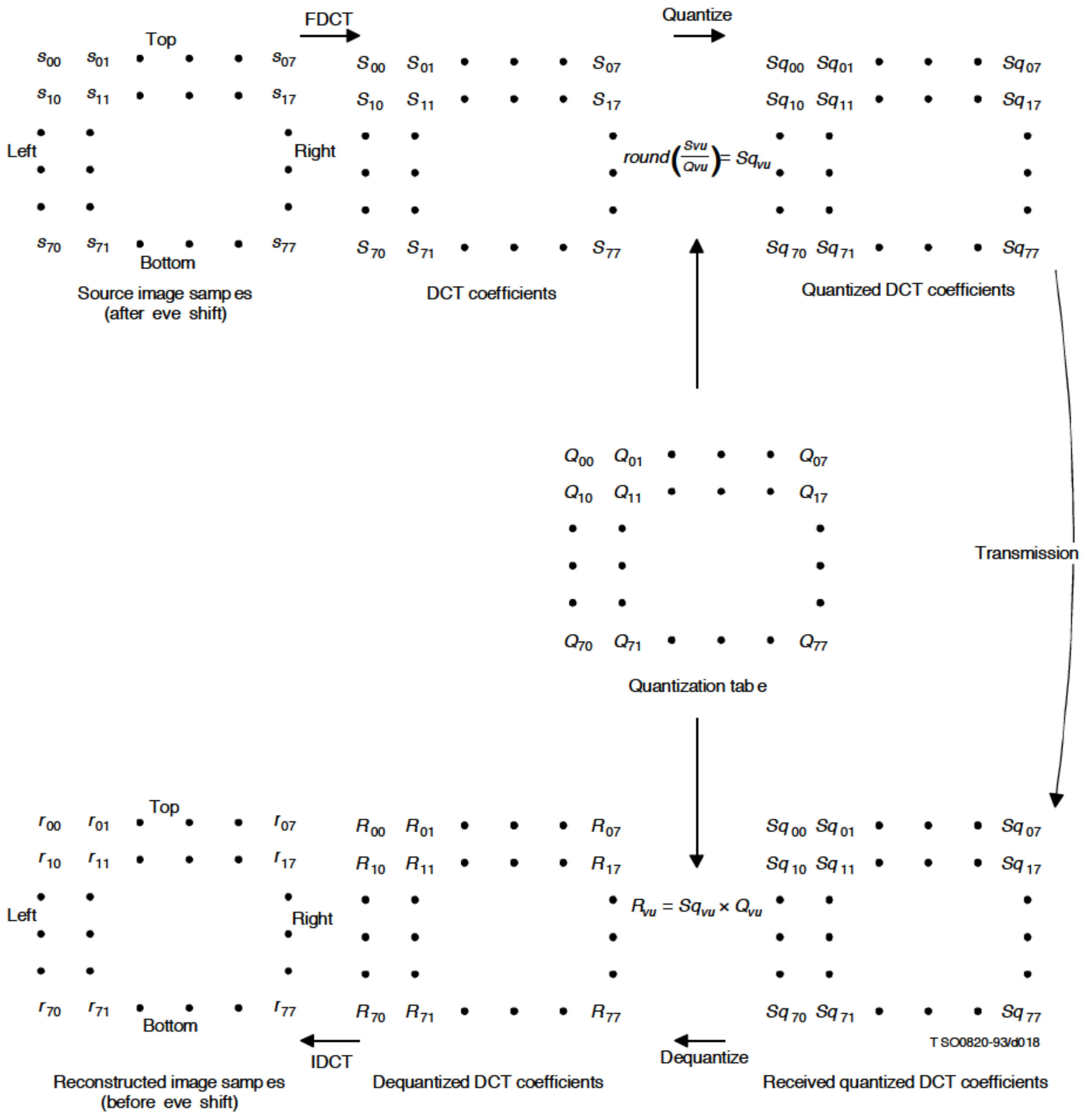
A.4 Point transform

For various procedures data may be optionally divided by a power of 2 by a point transform prior to coding. There are three processes which require a point transform: lossless coding, lossless differential frame coding in the hierarchical mode, and successive approximation coding in the progressive DCT mode.

In the lossless mode of operation the point transform is applied to the input samples. In the difference coding of the hierarchical mode of operation the point transform is applied to the difference between the input component samples and the reference component samples. In both cases the point transform is an integer divide by 2^{Pt}, where Pt is the value of the point transform parameter (see B.2.3).

In successive approximation coding the point transform for the AC coefficients is an integer divide by 2^{Al}, where Al is the successive approximation bit position, low (see B.2.3). The point transform for the DC coefficients is an arithmetic-shift-right by Al bits. This is equivalent to dividing by 2^{Pt} before the level shift (see A.3.1).

The output of the decoder is rescaled by multiplying by 2^{Pt}. An example of the point transform is given in K.10.



FDCT
Quantize
Top
Left
Right
Bottom
Source image samp es (after eve shift)
DCT coefficients
$round\left(\frac{S_{vu}}{Q_{vu}}\right) = Sq_{vu}$
Quantized DCT coefficients
Quantization tab e
Transmission
$R_{vu} = Sq_{vu} \times Q_{vu}$
Reconstructed image samp es (before eve shift)
IDCT
Dequantized DCT coefficients
Dequantize
Received quantized DCT coefficients
T SO0820-93/d018

Figure A.5 – Relationship between 8 × 8-block samples and DCT coefficients

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure A.6 – Zig-zag sequence of quantized DCT coefficients

A.5 Arithmetic procedures in lossless and hierarchical modes of operation

In the lossless mode of operation predictions are calculated with full precision and without clamping of either overflow or underflow beyond the range of values allowed by the precision of the input. However, the division by two which is part of some of the prediction calculations shall be approximated by an arithmetic-shift-right by one bit.

The two’s complement differences which are coded in either the lossless mode of operation or the differential frame coding in the hierarchical mode of operation are calculated modulo 65 536, thereby restricting the precision of these differences to a maximum of 16 bits. The modulo values are calculated by performing the logical AND operation of the two’s complement difference with X’FFFF’. For purposes of coding, the result is still interpreted as a 16 bit two’s complement difference. Modulo 65 536 arithmetic is also used in the decoder in calculating the output from the sum of the prediction and this two’s complement difference.

Annex B

Compressed data formats

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies three compressed data formats:

a) the interchange format, specified in B.2 and B.3;
b) the abbreviated format for compressed image data, specified in B.4;
c) the abbreviated format for table-specification data, specified in B.5.

B.1 describes the constituent parts of these formats. B.1.3 and B.1.4 give the conventions for symbols and figures used in the format specifications.

B.1 General aspects of the compressed data format specifications

Structurally, the compressed data formats consist of an ordered collection of parameters, markers, and entropy-coded data segments. Parameters and markers in turn are often organized into marker segments. Because all of these constituent parts are represented with byte-aligned codes, each compressed data format consists of an ordered sequence of 8-bit bytes. For each byte, a most significant bit (MSB) and a least significant bit (LSB) are defined.

B.1.1 Constituent parts

This subclause gives a general description of each of the constituent parts of the compressed data format.

B.1.1.1 Parameters

Parameters are integers, with values specific to the encoding process, source image characteristics, and other features selectable by the application. Parameters are assigned either 4-bit, 1-byte, or 2-byte codes. Except for certain optional groups of parameters, parameters encode critical information without which the decoding process cannot properly reconstruct the image.

The code assignment for a parameter shall be an unsigned integer of the specified length in bits with the particular value of the parameter.

For parameters which are 2 bytes (16 bits) in length, the most significant byte shall come first in the compressed data's ordered sequence of bytes. Parameters which are 4 bits in length always come in pairs, and the pair shall always be encoded in a single byte. The first 4-bit parameter of the pair shall occupy the most significant 4 bits of the byte. Within any 16-, 8-, or 4-bit parameter, the MSB shall come first and LSB shall come last.

B.1.1.2 Markers

Markers serve to identify the various structural parts of the compressed data formats. Most markers start marker segments containing a related group of parameters; some markers stand alone. All markers are assigned two-byte codes: an X'FF' byte followed by a byte which is not equal to 0 or X'FF' (see Table B.1). Any marker may optionally be preceded by any number of fill bytes, which are bytes assigned code X'FF'.

NOTE – Because of this special code-assignment structure, markers make it possible for a decoder to parse the compressed data and locate its various parts without having to decode other segments of image data.

B.1.1.3 Marker assignments

All markers shall be assigned two-byte codes: a X'FF' byte followed by a second byte which is not equal to 0 or X'FF'. The second byte is specified in Table B.1 for each defined marker. An asterisk (*) indicates a marker which stands alone, that is, which is not the start of a marker segment.

Table B.1 – Marker code assignments

Code Assignment	Symbol	Description
Start Of Frame markers, non-differential, Huffman coding		
X'FFC0'	SOF_0	Baseline DCT
X'FFC1'	SOF_1	Extended sequential DCT
X'FFC2'	SOF_2	Progressive DCT
X'FFC3'	SOF_3	Lossless (sequential)
Start Of Frame markers, differential, Huffman coding		
X'FFC5'	SOF_5	Differential sequential DCT
X'FFC6'	SOF_6	Differential progressive DCT
X'FFC7'	SOF_7	Differential lossless (sequential)
Start Of Frame markers, non-differential, arithmetic coding		
X'FFC8'	JPG	Reserved for JPEG extensions
X'FFC9'	SOF_9	Extended sequential DCT
X'FFCA'	SOF_{10}	Progressive DCT
X'FFCB'	SOF_{11}	Lossless (sequential)
Start Of Frame markers, differential, arithmetic coding		
X'FFCD'	SOF_{13}	Differential sequential DCT
X'FFCE'	SOF_{14}	Differential progressive DCT
X'FFCF'	SOF_{15}	Differential lossless (sequential)
Huffman table specification		
X'FFC4'	DHT	Define Huffman table(s)
Arithmetic coding conditioning specification		
X'FFCC'	DAC	Define arithmetic coding conditioning(s)
Restart interval termination		
X'FFD0' through X'FFD7'	RST_m*	Restart with modulo 8 count "m"
Other markers		
X'FFD8'	SOI*	Start of image
X'FFD9'	EOI*	End of image
X'FFDA'	SOS	Start of scan
X'FFDB'	DQT	Define quantization table(s)
X'FFDC'	DNL	Define number of lines
X'FFDD'	DRI	Define restart interval
X'FFDE'	DHP	Define hierarchical progression
X'FFDF'	EXP	Expand reference component(s)
X'FFE0' through X'FFEF'	APP_n	Reserved for application segments
X'FFF0' through X'FFFD'	JPG_n	Reserved for JPEG extensions
X'FFFE'	COM	Comment
Reserved markers		
X'FF01'	TEM*	For temporary private use in arithmetic coding
X'FF02' through X'FFBF'	RES	Reserved

B.1.1.4 Marker segments

A marker segment consists of a marker followed by a sequence of related parameters. The first parameter in a marker segment is the two-byte length parameter. This length parameter encodes the number of bytes in the marker segment, including the length parameter and excluding the two-byte marker. The marker segments identified by the SOF and SOS marker codes are referred to as headers: the frame header and the scan header respectively.

B.1.1.5 Entropy-coded data segments

An entropy-coded data segment contains the output of an entropy-coding procedure. It consists of an integer number of bytes, whether the entropy-coding procedure used is Huffman or arithmetic.

NOTES

1 Making entropy-coded segments an integer number of bytes is performed as follows: for Huffman coding, 1-bits are used, if necessary, to pad the end of the compressed data to complete the final byte of a segment. For arithmetic coding, byte alignment is performed in the procedure which terminates the entropy-coded segment (see D.1.8).

2 In order to ensure that a marker does not occur within an entropy-coded segment, any X'FF' byte generated by either a Huffman or arithmetic encoder, or an X'FF' byte that was generated by the padding of 1-bits described in NOTE 1 above, is followed by a "stuffed" zero byte (see D.1.6 and F.1.2.3).

B.1.2 Syntax

In B.2 and B.3 the interchange format syntax is specified. For the purposes of this Specification, the syntax specification consists of:

- the required ordering of markers, parameters, and entropy-coded segments;
- identification of optional or conditional constituent parts;
- the name, symbol, and definition of each marker and parameter;
- the allowed values of each parameter;
- any restrictions on the above which are specific to the various coding processes.

The ordering of constituent parts and the identification of which are optional or conditional is specified by the syntax figures in B.2 and B.3. Names, symbols, definitions, allowed values, conditions, and restrictions are specified immediately below each syntax figure.

B.1.3 Conventions for syntax figures

The syntax figures in B.2 and B.3 are a part of the interchange format specification. The following conventions, illustrated in Figure B.1, apply to these figures:

- **parameter/marker indicator:** A thin-lined box encloses either a marker or a single parameter;
- **segment indicator:** A thick-lined box encloses either a marker segment, an entropy-coded data segment, or combinations of these;
- **parameter length indicator:** The width of a thin-lined box is proportional to the parameter length (4, 8, or 16 bits, shown as E, B, and D respectively in Figure B.1) of the marker or parameter it encloses; the width of thick-lined boxes is not meaningful;
- **optional/conditional indicator:** Square brackets indicate that a marker or marker segment is only optionally or conditionally present in the compressed image data;
- **ordering:** In the interchange format a parameter or marker shown in a figure precedes all of those shown to its right, and follows all of those shown to its left;
- **entropy-coded data indicator:** Angled brackets indicate that the entity enclosed has been entropy encoded.

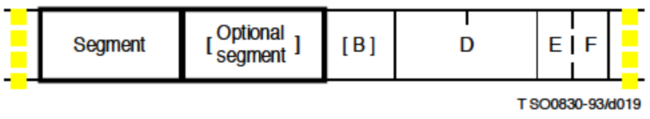


Figure B.1 – Syntax notation conventions

B.1.4 Conventions for symbols, code lengths, and values

Following each syntax figure in B.2 and B.3, the symbol, name, and definition for each marker and parameter shown in the figure are specified. For each parameter, the length and allowed values are also specified in tabular form.

The following conventions apply to symbols for markers and parameters:

- all marker symbols have three upper-case letters, and some also have a subscript. Examples: SOI, SOF_n;
- all parameter symbols have one upper-case letter; some also have one lower-case letter and some have subscripts. Examples: Y, Nf, H_i, Tq_i.

B.2 General sequential and progressive syntax

This clause specifies the interchange format syntax which applies to all coding processes for sequential DCT-based, progressive DCT-based, and lossless modes of operation.

B.2.1 High-level syntax

Figure B.2 specifies the order of the high-level constituent parts of the interchange format for all non-hierarchical encoding processes specified in this Specification.

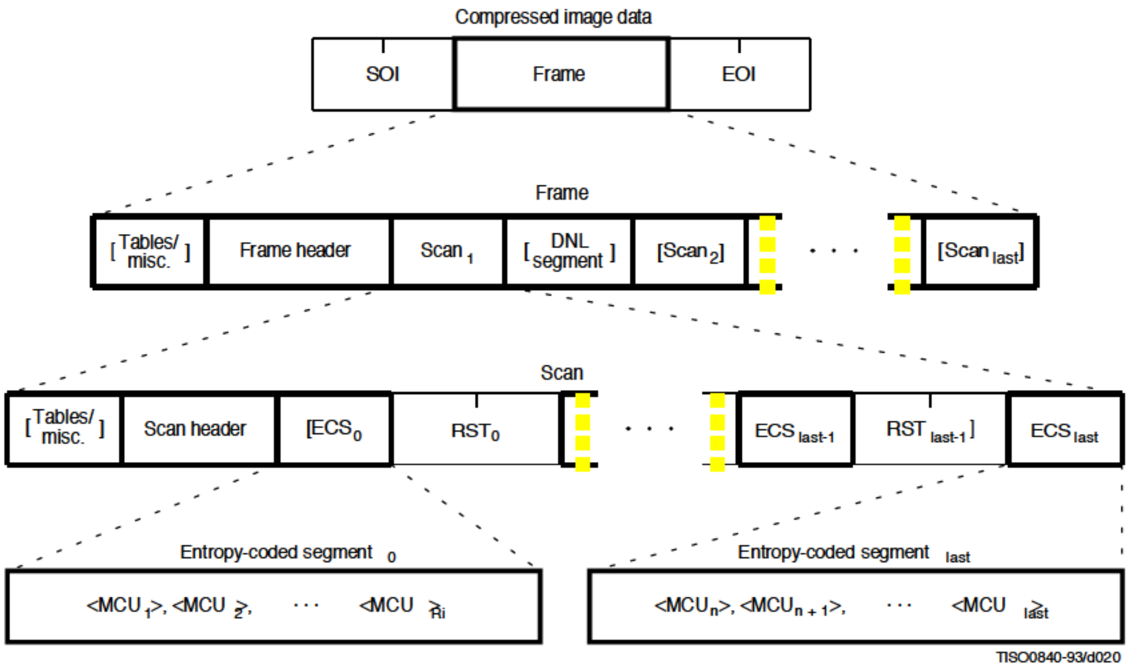


Figure B.2 – Syntax for sequential DCT-based, progressive DCT-based, and lossless modes of operation

The three markers shown in Figure B.2 are defined as follows:

SOI: Start of image marker – Marks the start of a compressed image represented in the interchange format or abbreviated format.

EOI: End of image marker – Marks the end of a compressed image represented in the interchange format or abbreviated format.

RST_m: Restart marker – A conditional marker which is placed between entropy-coded segments only if restart is enabled. There are 8 unique restart markers (m = 0 - 7) which repeat in sequence from 0 to 7, starting with zero for each scan, to provide a modulo 8 restart interval count.

The top level of Figure B.2 specifies that the non-hierarchical interchange format shall begin with an SOI marker, shall contain one frame, and shall end with an EOI marker.

The second level of Figure B.2 specifies that a frame shall begin with a frame header and shall contain one or more scans. A frame header may be preceded by one or more table-specification or miscellaneous marker segments as specified in B.2.4. If a DNL segment (see B.2.5) is present, it shall immediately follow the first scan.

For sequential DCT-based and lossless processes each scan shall contain from one to four image components. If two to four components are contained within a scan, they shall be interleaved within the scan. For progressive DCT-based processes each image component is only partially contained within any one scan. Only the first scan(s) for the components (which contain only DC coefficient data) may be interleaved.

The third level of Figure B.2 specifies that a scan shall begin with a scan header and shall contain one or more entropy-coded data segments. Each scan header may be preceded by one or more table-specification or miscellaneous marker segments. If restart is not enabled, there shall be only one entropy-coded segment (the one labeled “last”), and no restart markers shall be present. If restart is enabled, the number of entropy-coded segments is defined by the size of the image and the defined restart interval. In this case, a restart marker shall follow each entropy-coded segment except the last one.

The fourth level of Figure B.2 specifies that each entropy-coded segment is comprised of a sequence of entropy-coded MCUs. If restart is enabled and the restart interval is defined to be Ri, each entropy-coded segment except the last one shall contain Ri MCUs. The last one shall contain whatever number of MCUs completes the scan.

Figure B.2 specifies the locations where table-specification segments **may** be present. However, this Specification hereby specifies that the interchange format **shall** contain all table-specification data necessary for decoding the compressed image. Consequently, the required table-specification data **shall** be present at one or more of the allowed locations.

B.2.2 Frame header syntax

Figure B.3 specifies the frame header which shall be present at the start of a frame. This header specifies the source image characteristics (see A.1), the components in the frame, and the sampling factors for each component, and specifies the destinations from which the quantized tables to be used with each component are retrieved.

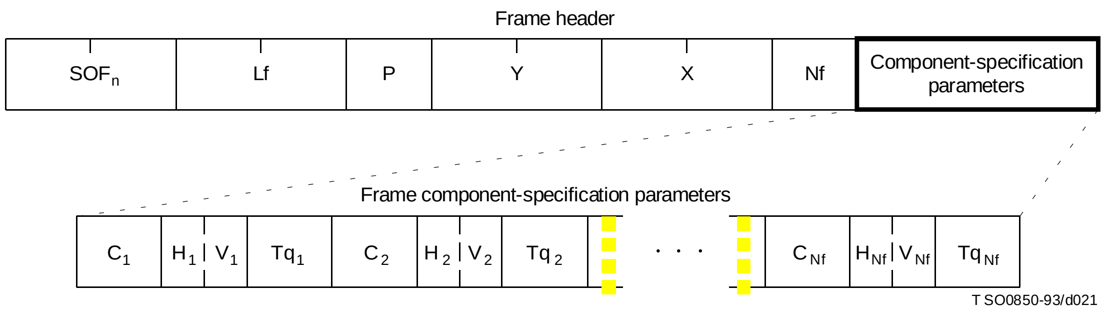


Figure B.3 – Frame header syntax

The markers and parameters shown in Figure B.3 are defined below. The size and allowed values of each parameter are given in Table B.2. In Table B.2 (and similar tables which follow), value choices are separated by commas (e.g. 8, 12) and inclusive bounds are separated by dashes (e.g. 0 - 3).

SOF$_n$: Start of frame marker – Marks the beginning of the frame parameters. The subscript n identifies whether the encoding process is baseline sequential, extended sequential, progressive, or lossless, as well as which entropy encoding procedure is used.

SOF$_0$: Baseline DCT

SOF$_1$: Extended sequential DCT, Huffman coding

SOF$_2$: Progressive DCT, Huffman coding

SOF_3: Lossless (sequential), Huffman coding

SOF_9: Extended sequential DCT, arithmetic coding

SOF_{10}: Progressive DCT, arithmetic coding

SOF_{11}: Lossless (sequential), arithmetic coding

Lf: Frame header length – Specifies the length of the frame header shown in Figure B.3 (see B.1.1.4).

P: Sample precision – Specifies the precision in bits for the samples of the components in the frame.

Y: Number of lines – Specifies the maximum number of lines in the source image. This shall be equal to the number of lines in the component with the maximum number of vertical samples (see A.1.1). Value 0 indicates that the number of lines shall be defined by the DNL marker and parameters at the end of the first scan (see B.2.5).

X: Number of samples per line – Specifies the maximum number of samples per line in the source image. This shall be equal to the number of samples per line in the component with the maximum number of horizontal samples (see A.1.1).

Nf: Number of image components in frame – Specifies the number of source image components in the frame. The value of Nf shall be equal to the number of sets of frame component specification parameters (C_i, H_i, V_i, and Tq_i) present in the frame header.

C_i: Component identifier – Assigns a unique label to the *i*th component in the sequence of frame component specification parameters. These values shall be used in the scan headers to identify the components in the scan. The value of C_i shall be different from the values of C_1 through C_{i-1}.

H_i: Horizontal sampling factor – Specifies the relationship between the component horizontal dimension and maximum image dimension X (see A.1.1); also specifies the number of horizontal data units of component C_i in each MCU, when more than one component is encoded in a scan.

V_i: Vertical sampling factor – Specifies the relationship between the component vertical dimension and maximum image dimension Y (see A.1.1); also specifies the number of vertical data units of component C_i in each MCU, when more than one component is encoded in a scan.

Tq_i: Quantization table destination selector – Specifies one of four possible quantization table destinations from which the quantization table to use for dequantization of DCT coefficients of component C_i is retrieved. If the decoding process uses the dequantization procedure, this table shall have been installed in this destination by the time the decoder is ready to decode the scan(s) containing component C_i. The destination shall not be re-specified, or its contents changed, until all scans containing C_i have been completed.

Table B.2 – Frame header parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lf	16	$8 + 3 \times Nf$			
P	8	8	8, 12	8, 12	2-16
Y	16	0-65 535			
X	16	1-65 535			
Nf	8	1-255	1-255	1-4	1-255
C_i	8	0-255			
H_i	4	1-4			
V_i	4	1-4			
Tq_i	8	0-3	0-3	0-3	0

B.2.3 Scan header syntax

Figure B.4 specifies the scan header which shall be present at the start of a scan. This header specifies which component(s) are contained in the scan, specifies the destinations from which the entropy tables to be used with each component are retrieved, and (for the progressive DCT) which part of the DCT quantized coefficient data is contained in the scan. For lossless processes the scan parameters specify the predictor and the point transform.

NOTE – If there is only one image component present in a scan, that component is, by definition, non-interleaved. If there is more than one image component present in a scan, the components present are, by definition, interleaved.

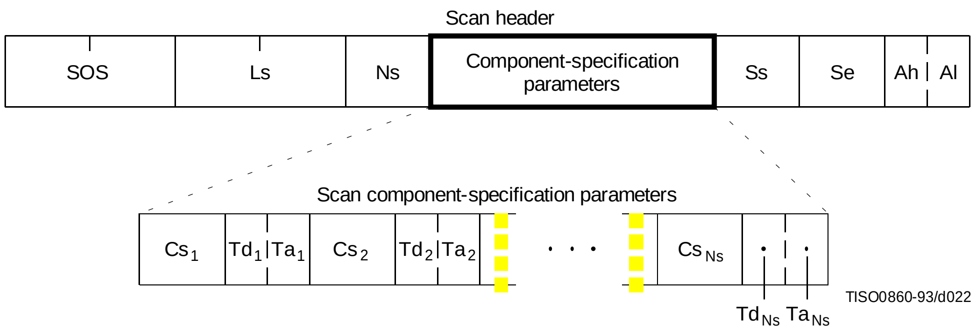


Figure B.4 – Scan header syntax

The marker and parameters shown in Figure B.4 are defined below. The size and allowed values of each parameter are given in Table B.3.

SOS: Start of scan marker – Marks the beginning of the scan parameters.

Ls: Scan header length – Specifies the length of the scan header shown in Figure B.4 (see B.1.1.4).

Ns: Number of image components in scan – Specifies the number of source image components in the scan. The value of Ns shall be equal to the number of sets of scan component specification parameters (Cs_j, Td_j, and Ta_j) present in the scan header.

Cs_j: Scan component selector – Selects which of the Nf image components specified in the frame parameters shall be the *j*th component in the scan. Each Cs_j shall match one of the C_i values specified in the frame header, and the ordering in the scan header shall follow the ordering in the frame header. If Ns > 1, the order of interleaved components in the MCU is Cs_1 first, Cs_2 second, etc. If Ns > 1, the following restriction shall be placed on the image components contained in the scan:

$$\sum_{j=1}^{N_s} H_j \times V_j \le 10,$$

where H_j and V_j are the horizontal and vertical sampling factors for scan component j. These sampling factors are specified in the frame header for component i, where i is the frame component specification index for which frame component identifier C_i matches scan component selector Cs_j.

As an example, consider an image having 3 components with maximum dimensions of 512 lines and 512 samples per line, and with the following sampling factors:

Component 0	$H_0 = 4,$	$V_0 = 1$
Component 1	$H_1 = 1,$	$V_1 = 2$
Component 2	$H_2 = 2$	$V_2 = 2$

Then the summation of $H_j \times V_j$ is $(4 \times 1) + (1 \times 2) + (2 \times 2) = 10$.

The value of Cs_j shall be different from the values of Cs_1 to Cs_{j-1}.

Td_j: DC entropy coding table destination selector – Specifies one of four possible DC entropy coding table destinations from which the entropy table needed for decoding of the DC coefficients of component Cs_j is retrieved. The DC entropy table shall have been installed in this destination (see B.2.4.2 and B.2.4.3) by the time the decoder is ready to decode the current scan. This parameter specifies the entropy coding table destination for the lossless processes.

Ta_j: AC entropy coding table destination selector – Specifies one of four possible AC entropy coding table destinations from which the entropy table needed for decoding of the AC coefficients of component Cs_j is retrieved. The AC entropy table selected shall have been installed in this destination (see B.2.4.2 and B.2.4.3) by the time the decoder is ready to decode the current scan. This parameter is zero for the lossless processes.

Ss: Start of spectral or predictor selection – In the DCT modes of operation, this parameter specifies the first DCT coefficient in each block in zig-zag order which shall be coded in the scan. This parameter shall be set to zero for the sequential DCT processes. In the lossless mode of operations this parameter is used to select the predictor.

Se: End of spectral selection – Specifies the last DCT coefficient in each block in zig-zag order which shall be coded in the scan. This parameter shall be set to 63 for the sequential DCT processes. In the lossless mode of operations this parameter has no meaning. It shall be set to zero.

Ah: Successive approximation bit position high – This parameter specifies the point transform used in the preceding scan (i.e. successive approximation bit position low in the preceding scan) for the band of coefficients specified by Ss and Se. This parameter shall be set to zero for the first scan of each band of coefficients. In the lossless mode of operations this parameter has no meaning. It shall be set to zero.

Al: Successive approximation bit position low or point transform – In the DCT modes of operation this parameter specifies the point transform, i.e. bit position low, used before coding the band of coefficients specified by Ss and Se. This parameter shall be set to zero for the sequential DCT processes. In the lossless mode of operations, this parameter specifies the point transform, Pt.

The entropy coding table destination selectors, Td_j and Ta_j, specify either Huffman tables (in frames using Huffman coding) or arithmetic coding tables (in frames using arithmetic coding). In the latter case the entropy coding table destination selector specifies both an arithmetic coding conditioning table destination and an associated statistics area.

Table B.3 – Scan header parameter size and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Ls	16	$6 + 2 \times Ns$			
Ns	8	1-4			
Cs_j	8	0-255[a)]			
Td_j	4	0-1	0-3	0-3	0-3
Ta_j	4	0-1	0-3	0-3	0
Ss	8	0	0	0-63	1-7[b)]
Se	8	63	63	Ss-63[c)]	0
Ah	4	0	0	0-13	0
Al	4	0	0	0-13	0-15

a) Cs_j shall be a member of the set of C_i specified in the frame header.

b) 0 for lossless differential frames in the hierarchical mode (see B.3).

c) 0 if Ss equals zero.

B.2.4 Table-specification and miscellaneous marker segment syntax

Figure B.5 specifies that, at the places indicated in Figure B.2, any of the table-specification segments or miscellaneous marker segments specified in B.2.4.1 through B.2.4.6 may be present in any order and with no limit on the number of segments.

If any table specification for a particular destination occurs in the compressed image data, it shall replace any previous table specified for this destination, and shall be used whenever this destination is specified in the remaining scans in the frame or subsequent images represented in the abbreviated format for compressed image data. If a table specification for a given destination occurs more than once in the compressed image data, each specification shall replace the previous specification. The quantization table specification shall not be altered between progressive DCT scans of a given component.

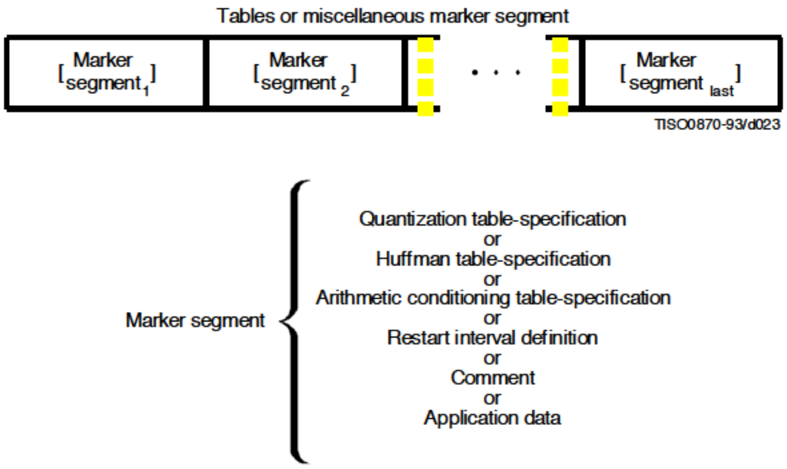

Tables or miscellaneous marker segment
[Marker segment$_1$]
[Marker segment$_2$]
[Marker segment$_{last}$]
TISO0870-93/d023
Marker segment
Quantization table-specification
or
Huffman table-specification
or
Arithmetic conditioning table-specification
or
Restart interval definition
or
Comment
or
Application data

Figure B.5 – Tables/miscellaneous marker segment syntax

B.2.4.1 Quantization table-specification syntax

Figure B.6 specifies the marker segment which defines one or more quantization tables.

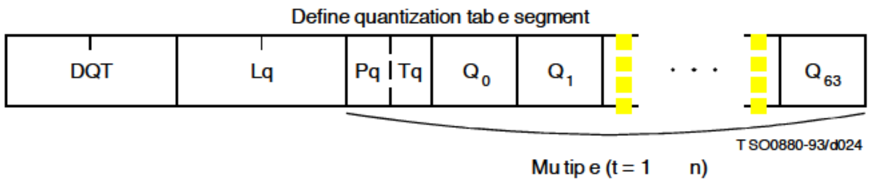


Figure B.6 – Quantization table syntax

The marker and parameters shown in Figure B.6 are defined below. The size and allowed values of each parameter are given in Table B.4.

DQT: Define quantization table marker – Marks the beginning of quantization table-specification parameters.

Lq: Quantization table definition length – Specifies the length of all quantization table parameters shown in Figure B.6 (see B.1.1.4).

Pq: Quantization table element precision – Specifies the precision of the Q_k values. Value 0 indicates 8-bit Q_k values; value 1 indicates 16-bit Q_k values. Pq shall be zero for 8 bit sample precision P (see B.2.2).

Tq: Quantization table destination identifier – Specifies one of four possible destinations at the decoder into which the quantization table shall be installed.

Q_k: Quantization table element – Specifies the *k*th element out of 64 elements, where *k* is the index in the zig-zag ordering of the DCT coefficients. The quantization elements shall be specified in zig-zag scan order.

Table B.4 – Quantization table-specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lq	16	$2 + \sum_{t=1}^{n} (65 + 64 \times Pq(t))$			Undefined
Pq	4	0	0, 1	0, 1	Undefined
Tq	4	0-3			Undefined
Q_k	8, 16	1-255, 1-65 535			Undefined

The value n in Table B.4 is the number of quantization tables specified in the DQT marker segment.

Once a quantization table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used, when referenced, in the remaining scans of the current image and in subsequent images represented in the abbreviated format for compressed image data. If a table has never been defined for a particular destination, then when this destination is specified in a frame header, the results are unpredictable.

An 8-bit DCT-based process shall not use a 16-bit precision quantization table.

B.2.4.2 Huffman table-specification syntax

Figure B.7 specifies the marker segment which defines one or more Huffman table specifications.

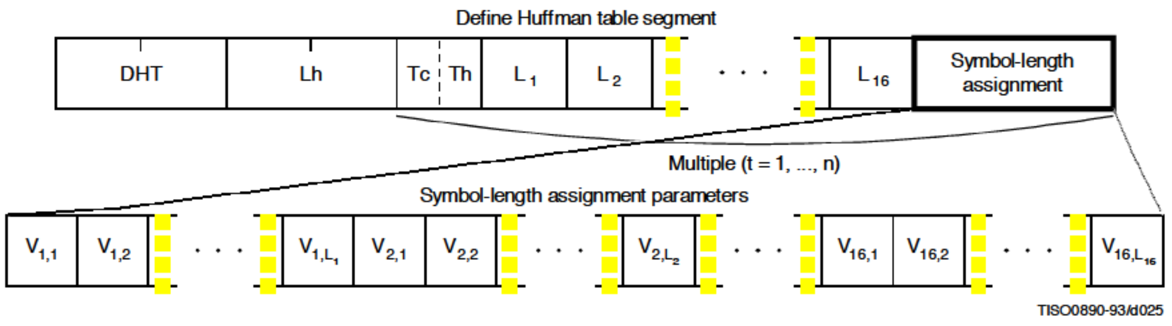


Figure B.7 – Huffman table syntax

The marker and parameters shown in Figure B.7 are defined below. The size and allowed values of each parameter are given in Table B.5.

DHT: Define Huffman table marker – Marks the beginning of Huffman table definition parameters.

Lh: Huffman table definition length – Specifies the length of all Huffman table parameters shown in Figure B.7 (see B.1.1.4).

Tc: Table class – 0 = DC table or lossless table, 1 = AC table.

Th: Huffman table destination identifier – Specifies one of four possible destinations at the decoder into which the Huffman table shall be installed.

L_i: Number of Huffman codes of length i – Specifies the number of Huffman codes for each of the 16 possible lengths allowed by this Specification. L_i's are the elements of the list BITS.

$V_{i,j}$: Value associated with each Huffman code – Specifies, for each i, the value associated with each Huffman code of length i. The meaning of each value is determined by the Huffman coding model. The $V_{i,j}$'s are the elements of the list HUFFVAL.

Table B.5 – Huffman table specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lh	16	$2 + \sum_{t=1}^{n} (17 + m_t)$			
Tc	4	0, 1			0
Th	4	0, 1	0-3		
L_i	8	0-255			
$V_{i\ j}$	8	0-255			

The value n in Table B.5 is the number of Huffman tables specified in the DHT marker segment. The value m_t is the number of parameters which follow the 16 $L_i(t)$ parameters for Huffman table t, and is given by:

$$m_t = \sum_{i=1}^{16} L_i$$

In general, m_t is different for each table.

Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used when referenced, in the remaining scans of the current image and in subsequent images represented in the abbreviated format for compressed image data. If a table has never been defined for a particular destination, then when this destination is specified in a scan header, the results are unpredictable.

B.2.4.3 Arithmetic conditioning table-specification syntax

Figure B.8 specifies the marker segment which defines one or more arithmetic coding conditioning table specifications. These replace the default arithmetic coding conditioning tables established by the SOI marker for arithmetic coding processes. (See F.1.4.4.1.4 and F.1.4.4.2.1.)

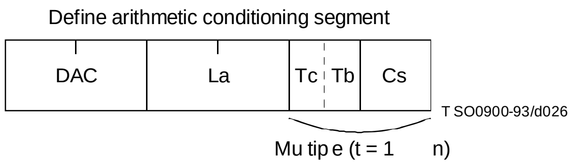


Figure B.8 – Arithmetic conditioning table-specification syntax

The marker and parameters shown in Figure B.8 are defined below. The size and allowed values of each parameter are given in Table B.6.

DAC: Define arithmetic coding conditioning marker – Marks the beginning of the definition of arithmetic coding conditioning parameters.

La: Arithmetic coding conditioning definition length – Specifies the length of all arithmetic coding conditioning parameters shown in Figure B.8 (see B.1.1.4).

Tc: Table class – 0 = DC table or lossless table, 1 = AC table.

Tb: Arithmetic coding conditioning table destination identifier – Specifies one of four possible destinations at the decoder into which the arithmetic coding conditioning table shall be installed.

Cs: Conditioning table value – Value in either the AC or the DC (and lossless) conditioning table. A single value of Cs shall follow each value of Tb. For AC conditioning tables Tc shall be one and Cs shall contain a value of Kx in the range $1 \leq Kx \leq 63$. For DC (and lossless) conditioning tables Tc shall be zero and Cs shall contain two 4-bit parameters, U and L. U and L shall be in the range $0 \leq L \leq U \leq 15$ and the value of Cs shall be $L + 16 \times U$.

The value n in Table B.6 is the number of arithmetic coding conditioning tables specified in the DAC marker segment. The parameters L and U are the lower and upper conditioning bounds used in the arithmetic coding procedures defined for DC coefficient coding and lossless coding. The separate value range 1-63 listed for DCT coding is the Kx conditioning used in AC coefficient coding.

Table B.6 – Arithmetic coding conditioning table-specification parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
La	16	Undefined	2 + 2 × n		
Tc	4	Undefined	0, 1		0
Tb	4	Undefined	0-3		
Cs	8	Undefined	0-255 (Tc = 0), 1-63 (Tc = 1)		0-255

B.2.4.4 Restart interval definition syntax

Figure B.9 specifies the marker segment which defines the restart interval.

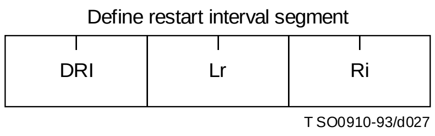


Figure B.9 – Restart interval definition syntax

The marker and parameters shown in Figure B.9 are defined below. The size and allowed values of each parameter are given in Table B.7.

DRI: Define restart interval marker – Marks the beginning of the parameters which define the restart interval.

Lr: Define restart interval segment length – Specifies the length of the parameters in the DRI segment shown in Figure B.9 (see B.1.1.4).

Ri: Restart interval – Specifies the number of MCU in the restart interval.

In Table B.7 the value n is the number of rows of MCU in the restart interval. The value MCUR is the number of MCU required to make up one line of samples of each component in the scan. The SOI marker disables the restart intervals. A DRI marker segment with Ri nonzero shall be present to enable restart interval processing for the following scans. A DRI marker segment with Ri equal to zero shall disable restart intervals for the following scans.

Table B.7 – Define restart interval segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lr	16	4			
Ri	16	0-65 535			n × MCUR

B.2.4.5 Comment syntax

Figure B.10 specifies the marker segment structure for a comment segment.

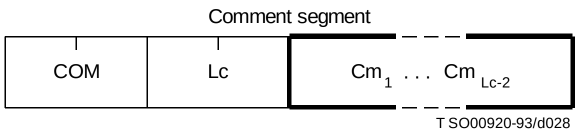


Figure B.10 – Comment segment syntax

The marker and parameters shown in Figure B.10 are defined below. The size and allowed values of each parameter are given in Table B.8.

COM: Comment marker – Marks the beginning of a comment.

Lc: Comment segment length – Specifies the length of the comment segment shown in Figure B.10 (see B.1.1.4).

Cm_i: Comment byte – The interpretation is left to the application.

Table B.8 – Comment segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lc	16	2-65 535			
Cm_i	8	0-255			

B.2.4.6 Application data syntax

Figure B.11 specifies the marker segment structure for an application data segment.

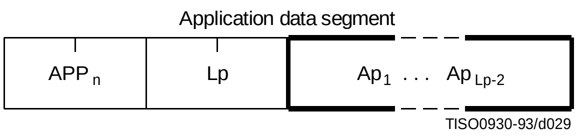


Figure B.11 – Application data syntax

The marker and parameters shown in Figure B.11 are defined below. The size and allowed values of each parameter are given in Table B.9.

APP_n: Application data marker – Marks the beginning of an application data segment.

Lp: Application data segment length – Specifies the length of the application data segment shown in Figure B.11 (see B.1.1.4).

Ap_i: Application data byte – The interpretation is left to the application.

The APP_n (Application) segments are reserved for application use. Since these segments may be defined differently for different applications, they should be removed when the data are exchanged between application environments.

Table B.9 – Application data segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Lp	16	2-65 535			
Ap_i	8	0-255			

B.2.5 Define number of lines syntax

Figure B.12 specifies the marker segment for defining the number of lines. The DNL (Define Number of Lines) segment provides a mechanism for defining or redefining the number of lines in the frame (the Y parameter in the frame header) at the end of the first scan. The value specified shall be consistent with the number of MCU-rows encoded in the first scan. This segment, if used, shall only occur at the end of the first scan, and only after coding of an integer number of MCU-rows. This marker segment is mandatory if the number of lines (Y) specified in the frame header has the value zero.

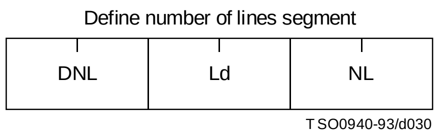


Figure B.12 – Define number of lines syntax

The marker and parameters shown in Figure B.12 are defined below. The size and allowed values of each parameter are given in Table B.10.

DNL: Define number of lines marker – Marks the beginning of the define number of lines segment.

Ld: Define number of lines segment length – Specifies the length of the define number of lines segment shown in Figure B.12 (see B.1.1.4).

NL: Number of lines – Specifies the number of lines in the frame (see definition of Y in B.2.2).

Table B.10 – Define number of lines segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Ld	16	4			
NL	16	1-65 535[a)]			

[a)] The value specified shall be consistent with the number of lines coded at the point where the DNL segment terminates the compressed data segment.

B.3 Hierarchical syntax

B.3.1 High level hierarchical mode syntax

Figure B.13 specifies the order of the high level constituent parts of the interchange format for hierarchical encoding processes.

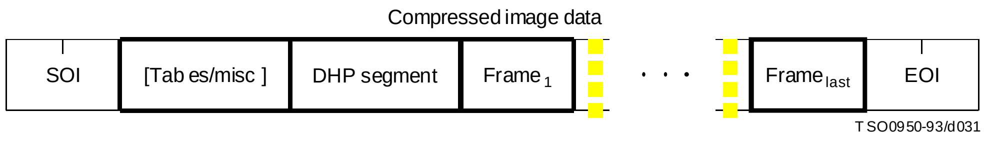


Figure B.13 – Syntax for the hierarchical mode of operation

Hierarchical mode syntax requires a DHP marker segment that appears before the non-differential frame or frames. The hierarchical mode compressed image data may include EXP marker segments and differential frames which shall follow the initial non-differential frame. The frame structure in hierarchical mode is identical to the frame structure in non-hierarchical mode.

The non-differential frames in the hierarchical sequence shall use one of the coding processes specified for SOF_n markers: SOF_0, SOF_1, SOF_2, SOF_3, SOF_9, SOF_{10} and SOF_{11}. The differential frames shall use one of the processes specified for SOF_5, SOF_6, SOF_7, SOF_{13}, SOF_{14} and SOF_{15}. The allowed combinations of SOF markers within one hierarchical sequence are specified in Annex J.

The sample precision (P) shall be constant for all frames and have the identical value as that coded in the DHP marker segment. The number of samples per line (X) for all frames shall not exceed the value coded in the DHP marker segment. If the number of lines (Y) is non-zero in the DHP marker segment, then the number of lines for all frames shall not exceed the value in the DHP marker segment.

B.3.2 DHP segment syntax

The DHP segment defines the image components, size, and sampling factors for the completed hierarchical sequence of frames. The DHP segment shall precede the first frame; a single DHP segment shall occur in the compressed image data.

The DHP segment structure is identical to the frame header syntax, except that the DHP marker is used instead of the SOF_n marker. The figures and description of B.2.2 then apply, except that the quantization table destination selector parameter shall be set to zero in the DHP segment.

B.3.3 EXP segment syntax

Figure B.14 specifies the marker segment structure for the EXP segment. The EXP segment shall be present if (and only if) expansion of the reference components is required either horizontally or vertically. The EXP segment parameters apply only to the next frame (which shall be a differential frame) in the image. If required, the EXP segment shall be one of the table-specification segments or miscellaneous marker segments preceding the frame header; the EXP segment shall not be one of the table-specification segments or miscellaneous marker segments preceding a scan header or a DHP marker segment.

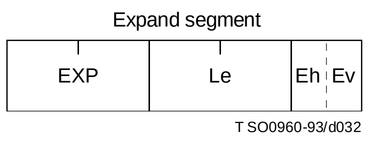


Figure B.14 – Syntax of the expand segment

The marker and parameters shown in Figure B.14 are defined below. The size and allowed values of each parameter are given in Table B.11.

> **EXP:** Expand reference components marker – Marks the beginning of the expand reference components segment.
>
> **Le:** Expand reference components segment length – Specifies the length of the expand reference components segment (see B.1.1.4).
>
> **Eh:** Expand horizontally – If one, the reference components shall be expanded horizontally by a factor of two. If horizontal expansion is not required, the value shall be zero.
>
> **Ev:** Expand vertically – If one, the reference components shall be expanded vertically by a factor of two. If vertical expansion is not required, the value shall be zero.
>
> Both Eh and Ev shall be one if expansion is required both horizontally and vertically.

Table B.11 – Expand segment parameter sizes and values

Parameter	Size (bits)	Values			
		Sequential DCT		Progressive DCT	Lossless
		Baseline	Extended		
Le	16	3			
Eh	4	0, 1			
Ev	4	0, 1			

B.4 Abbreviated format for compressed image data

Figure B.2 shows the high-level constituent parts of the interchange format. This format includes all table specifications required for decoding. If an application environment provides methods for table specification other than by means of the compressed image data, some or all of the table specifications may be omitted. Compressed image data which is missing any table specification data required for decoding has the abbreviated format.

B.5 Abbreviated format for table-specification data

Figure B.2 shows the high-level constituent parts of the interchange format. If no frames are present in the compressed image data, the only purpose of the compressed image data is to convey table specifications or miscellaneous marker segments defined in B.2.4.1, B.2.4.2, B.2.4.5, and B.2.4.6. In this case the compressed image data has the abbreviated format for table specification data (see Figure B.15).

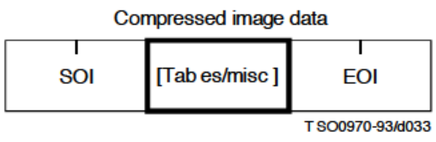


Figure B.15 – Abbreviated format for table-specification data syntax

B.6 Summary

The order of the constituent parts of interchange format and all marker segment structures is summarized in Figures B.16 and B.17. Note that in Figure B.16 double-lined boxes enclose marker segments. In Figures B.16 and B.17 thick-lined boxes enclose only markers.

The EXP segment can be mixed with the other tables/miscellaneous marker segments preceding the frame header but not with the tables/miscellaneous marker segments preceding the DHP segment or the scan header.

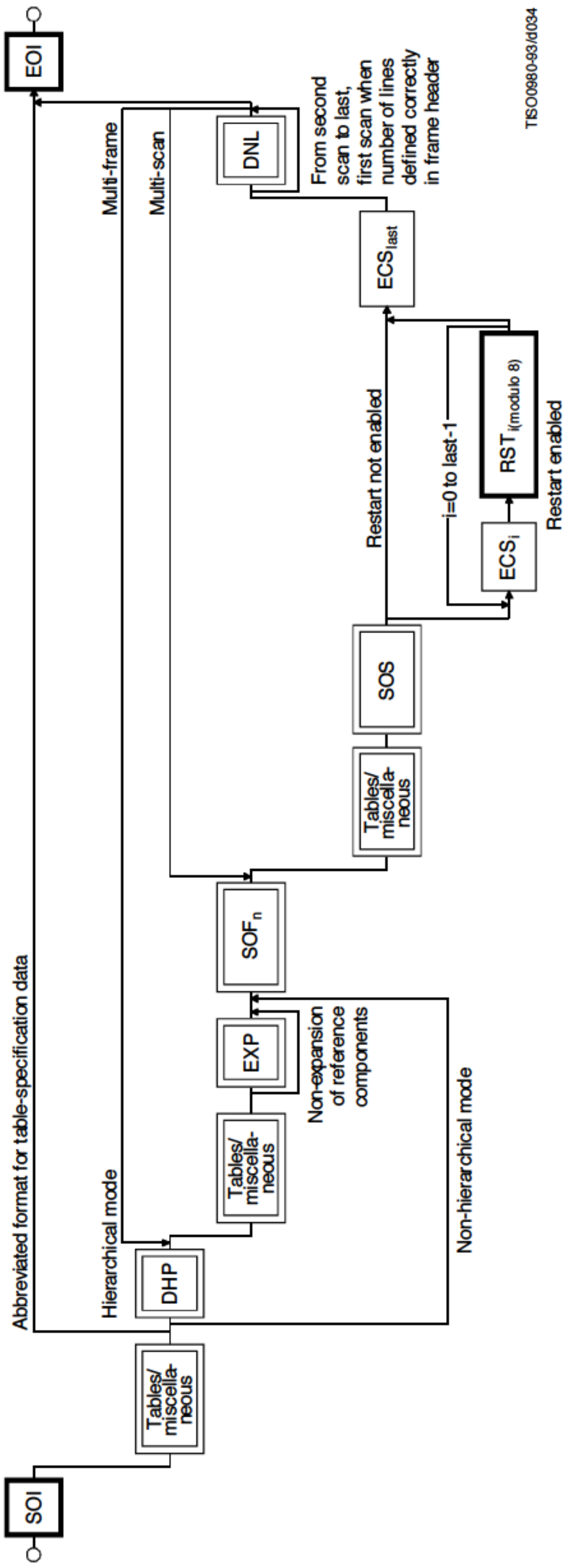


Figure B.16 – Flow of compressed data syntax

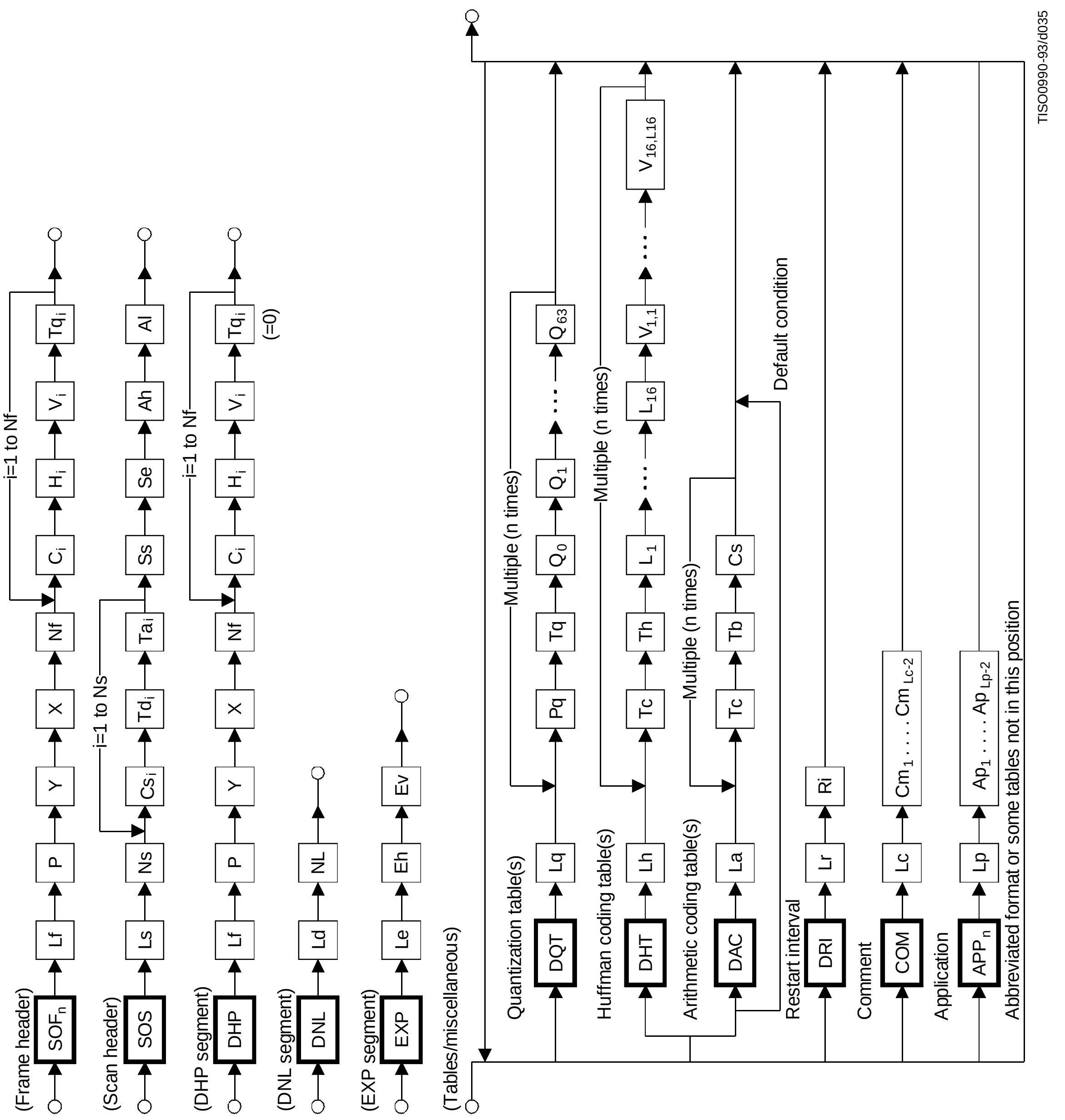


Figure B.17 – Flow of marker segment

Annex C

Huffman table specification

(This annex forms an integral part of this Recommendation | International Standard)

A Huffman coding procedure may be used for entropy coding in any of the coding processes. Coding models for Huffman encoding are defined in Annexes F, G, and H. In this Annex, the Huffman table specification is defined.

Huffman tables are specified in terms of a 16-byte list (BITS) giving the number of codes for each code length from 1 to 16. This is followed by a list of the 8-bit symbol values (HUFFVAL), each of which is assigned a Huffman code. The symbol values are placed in the list in order of increasing code length. Code lengths greater than 16 bits are not allowed. In addition, the codes shall be generated such that the all-1-bits code word of any length is reserved as a prefix for longer code words.

NOTE – The order of the symbol values within HUFFVAL is determined only by code length. Within a given code length the ordering of the symbol values is arbitrary.

This annex specifies the procedure by which the Huffman tables (of Huffman code words and their corresponding 8-bit symbol values) are derived from the two lists (BITS and HUFFVAL) in the interchange format. However, the way in which these lists are generated is not specified. The lists should be generated in a manner which is consistent with the rules for Huffman coding, and it shall observe the constraints discussed in the previous paragraph. Annex K contains an example of a procedure for generating lists of Huffman code lengths and values which are in accord with these rules.

NOTE – There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

C.1 Marker segments for Huffman table specification

The DHT marker identifies the start of Huffman table definitions within the compressed image data. B.2.4.2 specifies the syntax for Huffman table specification.

C.2 Conversion of Huffman table specifications to tables of codes and code lengths

Conversion of Huffman table specifications to tables of codes and code lengths uses three procedures. The first procedure (Figure C.1) generates a table of Huffman code sizes. The second procedure (Figure C.2) generates the Huffman codes from the table built in Figure C.1. The third procedure (Figure C.3) generates the Huffman codes in symbol value order.

Given a list BITS (1 to 16) containing the number of codes of each size, and a list HUFFVAL containing the symbol values to be associated with those codes as described above, two tables are generated. The HUFFSIZE table contains a list of code lengths; the HUFFCODE table contains the Huffman codes corresponding to those lengths.

Note that the variable LASTK is set to the index of the last entry in the table.

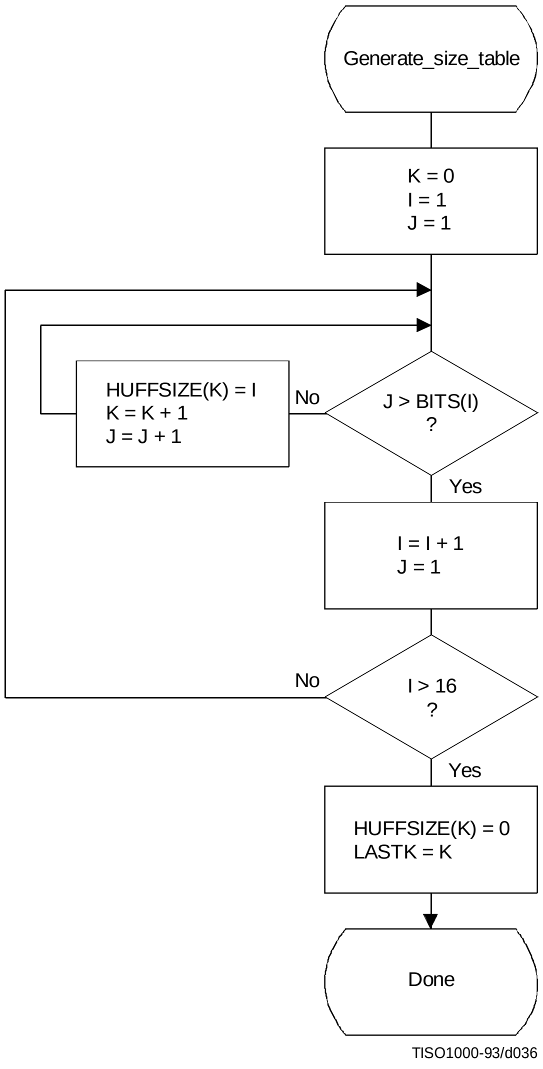


Figure C.1 – Generation of table of Huffman code sizes

A Huffman code table, HUFFCODE, containing a code for each size in HUFFSIZE is generated by the procedure in Figure C.2. The notation "SLL CODE 1" in Figure C.2 indicates a shift-left-logical of CODE by one bit position.

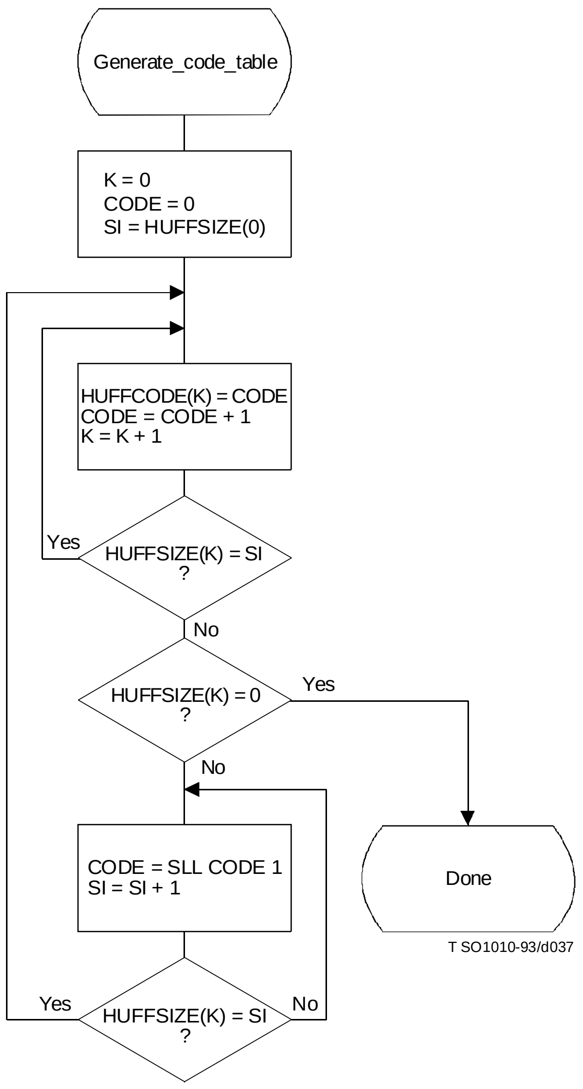


Figure C.2 – Generation of table of Huffman codes

Two tables, HUFFCODE and HUFFSIZE, have now been generated. The entries in the tables are ordered according to increasing Huffman code numeric value and length.

The encoding procedure code tables, EHUFCO and EHUFSI, are created by reordering the codes specified by HUFFCODE and HUFFSIZE according to the symbol values assigned to each code in HUFFVAL.

Figure C.3 illustrates this ordering procedure.

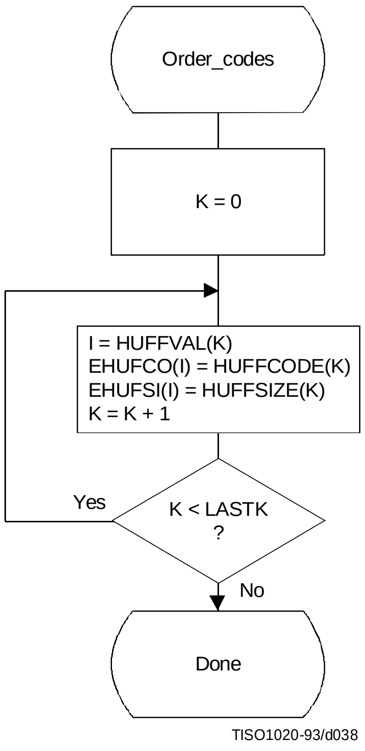


Figure C.3 – Ordering procedure for encoding procedure code tables

C.3 Bit ordering within bytes

The root of a Huffman code is placed toward the MSB (most-significant-bit) of the byte, and successive bits are placed in the direction MSB to LSB (least-significant-bit) of the byte. Remaining bits, if any, go into the next byte following the same rules.

Integers associated with Huffman codes are appended with the MSB adjacent to the LSB of the preceding Huffman code.

Annex D

Arithmetic coding

(This annex forms an integral part of this Recommendation | International Standard)

An adaptive binary arithmetic coding procedure may be used for entropy coding in any of the coding processes except the baseline sequential process. Coding models for adaptive binary arithmetic coding are defined in Annexes F, G, and H. In this annex the arithmetic encoding and decoding procedures used in those models are defined.

In K.4 a simple test example is given which should be helpful in determining if a given implementation is correct.

NOTE – There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

D.1 Arithmetic encoding procedures

Four arithmetic encoding procedures are required in a system with arithmetic coding (see Table D.1).

Table D.1 – Procedures for binary arithmetic encoding

Procedure	Purpose
Code_0(S)	Code a "0" binary decision with context-index S
Code_1(S)	Code a "1" binary decision with context-index S
Initenc	Initialize the encoder
Flush	Terminate entropy-coded segment

The "Code_0(S)"and "Code_1(S)" procedures code the 0-decision and 1-decision respectively; S is a context-index which identifies a particular conditional probability estimate used in coding the binary decision. The "Initenc" procedure initializes the arithmetic coding entropy encoder. The "Flush" procedure terminates the entropy-coded segment in preparation for the marker which follows.

D.1.1 Binary arithmetic encoding principles

The arithmetic coder encodes a series of binary symbols, zeros and ones, each symbol representing one possible result of a binary decision.

Each "binary decision" provides a choice between two alternatives. The binary decision might be between positive and negative signs, a magnitude being zero or nonzero, or a particular bit in a sequence of binary digits being zero or one.

The output bit stream (entropy-coded data segment) represents a binary fraction which increases in precision as bytes are appended by the encoding process.

D.1.1.1 Recursive interval subdivision

Recursive probability interval subdivision is the basis for the binary arithmetic encoding procedures. With each binary decision the current probability interval is subdivided into two sub-intervals, and the bit stream is modified (if necessary) so that it points to the base (the lower bound) of the probability sub-interval assigned to the symbol which occurred.

In the partitioning of the current probability interval into two sub-intervals, the sub-interval for the less probable symbol (LPS) and the sub-interval for the more probable symbol (MPS) are ordered such that usually the MPS sub-interval is closer to zero. Therefore, when the LPS is coded, the MPS sub-interval size is added to the bit stream. This coding convention requires that symbols be recognized as either MPS or LPS rather than 0 or 1. Consequently, the size of the LPS sub-interval and the sense of the MPS for each decision must be known in order to encode that decision.

The subdivision of the current probability interval would ideally require a multiplication of the interval by the probability estimate for the LPS. Because this subdivision is done approximately, it is possible for the LPS sub-interval to be larger than the MPS sub-interval. When that happens a “conditional exchange” interchanges the assignment of the sub-intervals such that the MPS is given the larger sub-interval.

Since the encoding procedure involves addition of binary fractions rather than concatenation of integer code words, the more probable binary decisions can sometimes be coded at a cost of much less than one bit per decision.

D.1.1.2 Conditioning of probability estimates

An adaptive binary arithmetic coder requires a statistical model – a model for selecting conditional probability estimates to be used in the coding of each binary decision. When a given binary decision probability estimate is dependent on a particular feature or features (the context) already coded, it is “conditioned” on that feature. The conditioning of probability estimates on previously coded decisions must be identical in encoder and decoder, and therefore can use only information known to both.

Each conditional probability estimate required by the statistical model is kept in a separate storage location or “bin” identified by a unique context-index S. The arithmetic coder is adaptive, which means that the probability estimates at each context-index are developed and maintained by the arithmetic coding system on the basis of prior coding decisions for that context-index.

D.1.2 Encoding conventions and approximations

The encoding procedures use fixed precision integer arithmetic and an integer representation of fractional values in which X’8000’ can be regarded as the decimal value 0.75. The probability interval, A, is kept in the integer range X’8000’ ≤ A < X’10000’ by doubling it whenever its integer value falls below X’8000’. This is equivalent to keeping A in the decimal range 0.75 ≤ A < 1.5. This doubling procedure is called renormalization.

The code register, C, contains the trailing bits of the bit stream. C is also doubled each time A is doubled. Periodically – to keep C from overflowing – a byte of data is removed from the high order bits of the C-register and placed in the entropy-coded segment.

Carry-over into the entropy-coded segment is limited by delaying X’FF’ output bytes until the carry-over is resolved. Zero bytes are stuffed after each X’FF’ byte in the entropy-coded segment in order to avoid the accidental generation of markers in the entropy-coded segment.

Keeping A in the range $0.75 \leq A < 1.5$ allows a simple arithmetic approximation to be used in the probability interval subdivision. Normally, if the current estimate of the LPS probability for context-index S is Qe(S), precise calculation of the sub-intervals would require:

$Qe(S) \times A$	Probability sub-interval for the LPS;
$A - (Qe(S) \times A)$	Probability sub-interval for the MPS.

Because the decimal value of A is of order unity, these can be approximated by

$Qe(S)$	Probability sub-interval for the LPS;
$A - Qe(S)$	Probability sub-interval for the MPS.

Whenever the LPS is coded, the value of A – Qe(S) is added to the code register and the probability interval is reduced to Qe(S). Whenever the MPS is coded, the code register is left unchanged and the interval is reduced to A – Qe(S). The precision range required for A is then restored, if necessary, by renormalization of both A and C.

With the procedure described above, the approximations in the probability interval subdivision process can sometimes make the LPS sub-interval larger than the MPS sub-interval. If, for example, the value of Qe(S) is 0.5 and A is at the minimum allowed value of 0.75, the approximate scaling gives one-third of the probability interval to the MPS and two-thirds to the LPS. To avoid this size inversion, conditional exchange is used. The probability interval is subdivided using the simple approximation, but the MPS and LPS sub-interval assignments are exchanged whenever the LPS sub-interval is larger than the MPS sub-interval. This MPS/LPS conditional exchange can only occur when a renormalization will be needed.

Each binary decision uses a context. A context is the set of prior coding decisions which determine the context-index, S, identifying the probability estimate used in coding the decision.

Whenever a renormalization occurs, a probability estimation procedure is invoked which determines a new probability estimate for the context currently being coded. No explicit symbol counts are needed for the estimation. The relative probabilities of renormalization after coding of LPS and MPS provide, by means of a table-based probability estimation state machine, a direct estimate of the probabilities.

D.1.3 Encoder code register conventions

The flow charts in this annex assume the register structures for the encoder as shown in Table D.2.

Table D.2 – Encoder register connections

	MSB			LSB
C-register	0000cbbb,	bbbbbsss,	xxxxxxxx,	xxxxxxxx
A-register	00000000,	00000000,	aaaaaaaa,	aaaaaaaa

The "a" bits are the fractional bits in the A-register (the current probability interval value) and the "x" bits are the fractional bits in the code register. The "s" bits are optional spacer bits which provide useful constraints on carry-over, and the "b" bits indicate the bit positions from which the completed bytes of data are removed from the C-register. The "c" bit is a carry bit. Except at the time of initialization, bit 15 of the A-register is always set and bit 16 is always clear (the LSB is bit 0).

These register conventions illustrate one possible implementation. However, any register conventions which allow resolution of carry-over in the encoder and which produce the same entropy-coded segment may be used. The handling of carry-over and the byte stuffing following X'FF' will be described in a later part of this annex.

D.1.4 Code_1(S) and Code_0(S) procedures

When a given binary decision is coded, one of two possibilities occurs – either a 1-decision or a 0-decision is coded. Code_1(S) and Code_0(S) are shown in Figures D.1 and D.2. The Code_1(S) and Code_0(S) procedures use probability estimates with a context-index S. The context-index S is determined by the statistical model and is, in general, a function of the previous coding decisions; each value of S identifies a particular conditional probability estimate which is used in encoding the binary decision.

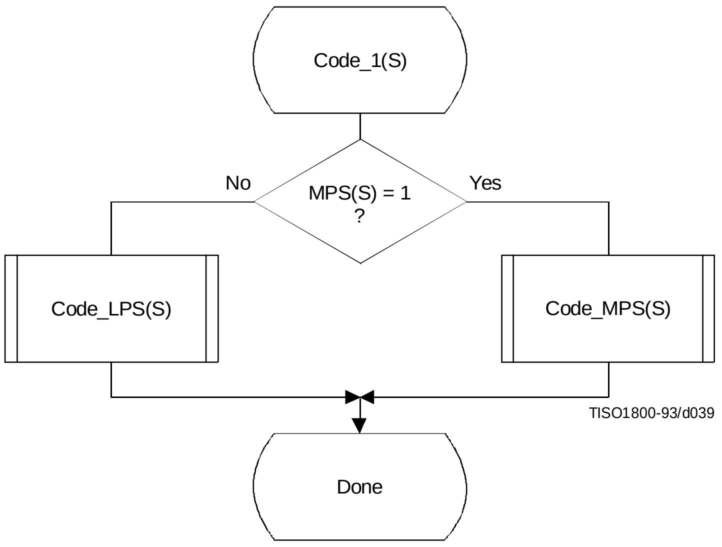


Figure D.1 – Code_1(S) procedure

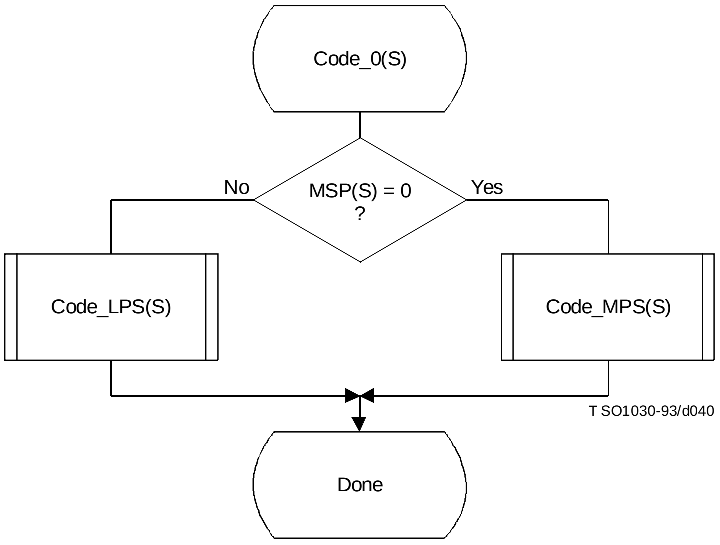


Figure D.2 – Code_0(S) procedure

The context-index S selects a storage location which contains Index(S), an index to the tables which make up the probability estimation state machine. When coding a binary decision, the symbol being coded is either the more probable symbol or the less probable symbol. Therefore, additional information is stored at each context-index identifying the sense of the more probable symbol, MPS(S).

For simplicity, the flow charts in this subclause assume that the context storage for each context-index S has an additional storage field for Qe(S) containing the value of Qe(Index(S)). If only the value of Index(S) and MPS(S) are stored, all references to Qe(S) should be replaced by Qe(Index(S)).

The Code_LPS(S) procedure normally consists of the addition of the MPS sub-interval A – Qe(S) to the bit stream and a scaling of the interval to the sub-interval, Qe(S). It is always followed by the procedures for obtaining a new LPS probability estimate (Estimate_Qe(S)_after_LPS) and renormalization (Renorm_e) (see Figure D.3).

However, in the event that the LPS sub-interval is larger than the MPS sub-interval, the conditional MPS/LPS exchange occurs and the MPS sub-interval is coded.

The Code_MPS(S) procedure normally reduces the size of the probability interval to the MPS sub-interval. However, if the LPS sub-interval is larger than the MPS sub-interval, the conditional exchange occurs and the LPS sub-interval is coded instead. Note that conditional exchange cannot occur unless the procedures for obtaining a new LPS probability estimate (Estimate_Qe(S)_after_MPS) and renormalization (Renorm_e) are required after the coding of the symbol (see Figure D.4).

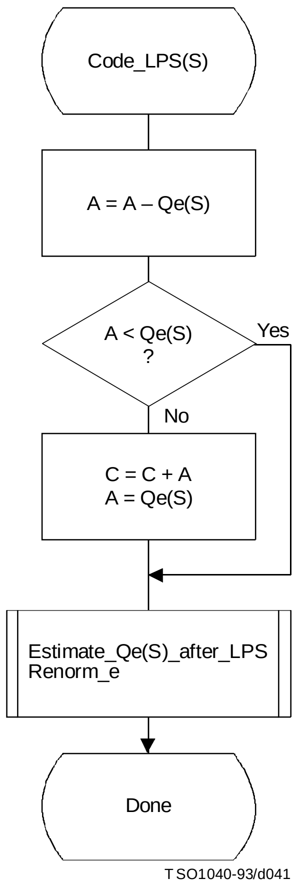


Figure D.3 – Code_LPS(S) procedure with conditional MPS/LPS exchange

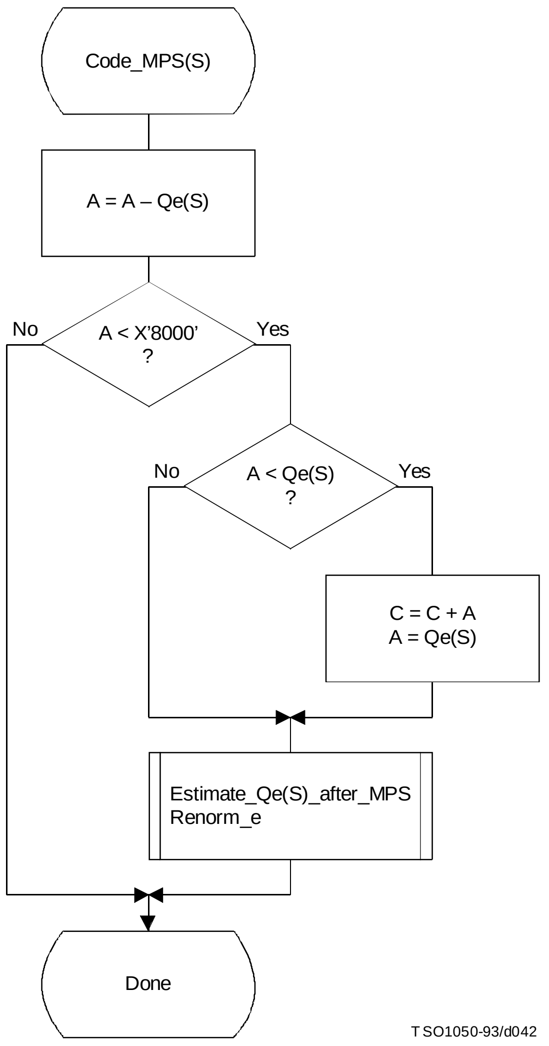


Figure D.4 – Code_MPS(S) procedure with conditional MPS/LPS exchange

D.1.5 Probability estimation in the encoder

D.1.5.1 Probability estimation state machine

The probability estimation state machine consists of a number of sequences of probability estimates. These sequences are interlinked in a manner which provides probability estimates based on approximate symbol counts derived from the arithmetic coder renormalization. Some of these sequences are used during the initial "learning" stages of probability estimation; the rest are used for "steady state" estimation.

Each entry in the probability estimation state machine is assigned an index, and each index has associated with it a Qe value and two Next_Index values. The Next_Index_MPS gives the index to the new probability estimate after an MPS renormalization; the Next_Index_LPS gives the index to the new probability estimate after an LPS renormalization. Note that both the index to the estimation state machine and the sense of the MPS are kept for each context-index S. The sense of the MPS is changed whenever the entry in the Switch_MPS is one.

The probability estimation state machine is given in Table D.3. Initialization of the arithmetic coder is always with an MPS sense of zero and a Qe index of zero in Table D.3.

The Qe values listed in Table D.3 are expressed as hexadecimal integers. To approximately convert the 15-bit integer representation of Qe to a decimal probability, divide the Qe values by (4/3) × (X'8000').

Table D.3 – Qe values and probability estimation state machine

Index	Qe	Next_ Index		Switch
	_Value	_LPS	_MPS	_MPS
0	X'5A1D'	1	1	1
1	X'2586'	14	2	0
2	X'1114'	16	3	0
3	X'080B'	18	4	0
4	X'03D8'	20	5	0
5	X'01DA'	23	6	0
6	X'00E5'	25	7	0
7	X'006F'	28	8	0
8	X'0036'	30	9	0
9	X'001A'	33	10	0
10	X'000D'	35	11	0
11	X'0006'	9	12	0
12	X'0003'	10	13	0
13	X'0001'	12	13	0
14	X'5A7F'	15	15	1
15	X'3F25'	36	16	0
16	X'2CF2'	38	17	0
17	X'207C'	39	18	0
18	X'17B9'	40	19	0
19	X'1182'	42	20	0
20	X'0CEF'	43	21	0
21	X'09A1'	45	22	0
22	X'072F'	46	23	0
23	X'055C'	48	24	0
24	X'0406'	49	25	0
25	X'0303'	51	26	0
26	X'0240'	52	27	0
27	X'01B1'	54	28	0
28	X'0144'	56	29	0
29	X'00F5'	57	30	0
30	X'00B7'	59	31	0
31	X'008A'	60	32	0
32	X'0068'	62	33	0
33	X'004E'	63	34	0
34	X'003B'	32	35	0
35	X'002C'	33	9	0
36	X'5AE1'	37	37	1
37	X'484C'	64	38	0
38	X'3A0D'	65	39	0
39	X'2EF1'	67	40	0
40	X'261F'	68	41	0
41	X'1F33'	69	42	0
42	X'19A8'	70	43	0
43	X'1518'	72	44	0
44	X'1177'	73	45	0
45	X'0E74'	74	46	0
46	X'0BFB'	75	47	0
47	X'09F8'	77	48	0
48	X'0861'	78	49	0
49	X'0706'	79	50	0
50	X'05CD'	48	51	0
51	X'04DE'	50	52	0
52	X'040F'	50	53	0
53	X'0363'	51	54	0
54	X'02D4'	52	55	0
55	X'025C'	53	56	0
56	X'01F8'	54	57	0
57	X'01A4'	55	58	0
58	X'0160'	56	59	0
59	X'0125'	57	60	0
60	X'00F6'	58	61	0
61	X'00CB'	59	62	0
62	X'00AB'	61	63	0
63	X'008F'	61	32	0
64	X'5B12'	65	65	1
65	X'4D04'	80	66	0
66	X'412C'	81	67	0
67	X'37D8'	82	68	0
68	X'2FE8'	83	69	0
69	X'293C'	84	70	0
70	X'2379'	86	71	0
71	X'1EDF'	87	72	0
72	X'1AA9'	87	73	0
73	X'174E'	72	74	0
74	X'1424'	72	75	0
75	X'119C'	74	76	0
76	X'0F6B'	74	77	0
77	X'0D51'	75	78	0
78	X'0BB6'	77	79	0
79	X'0A40'	77	48	0
80	X'5832'	80	81	1
81	X'4D1C'	88	82	0
82	X'438E'	89	83	0
83	X'3BDD'	90	84	0
84	X'34EE'	91	85	0
85	X'2EAE'	92	86	0
86	X'299A'	93	87	0
87	X'2516'	86	71	0
88	X'5570'	88	89	1
89	X'4CA9'	95	90	0
90	X'44D9'	96	91	0
91	X'3E22'	97	92	0
92	X'3824'	99	93	0
93	X'32B4'	99	94	0
94	X'2E17'	93	86	0
95	X'56A8'	95	96	1
96	X'4F46'	101	97	0
97	X'47E5'	102	98	0
98	X'41CF'	103	99	0
99	X'3C3D'	104	100	0
100	X'375E'	99	93	0
101	X'5231'	105	102	0
102	X'4C0F'	106	103	0
103	X'4639'	107	104	0
104	X'415E'	103	99	0
105	X'5627'	105	106	1
106	X'50E7'	108	107	0
107	X'4B85'	109	103	0
108	X'5597'	110	109	0
109	X'504F'	111	107	0
110	X'5A10'	110	111	1
111	X'5522'	112	109	0
112	X'59EB'	112	111	1

D.1.5.2 Renormalization driven estimation

The change in state in Table D.3 occurs only when the arithmetic coder interval register is renormalized. This must always be done after coding an LPS, and whenever the probability interval register is less than X'8000' (0.75 in decimal notation) after coding an MPS.

When the LPS renormalization is required, Next_Index_LPS gives the new index for the LPS probability estimate. When the MPS renormalization is required, Next_Index_MPS gives the new index for the LPS probability estimate. If Switch_MPS is 1 for the old index, the MPS symbol sense must be inverted after an LPS.

D.1.5.3 Estimation following renormalization after MPS

The procedure for estimating the probability on the MPS renormalization path is given in Figure D.5. Index(S) is part of the information stored for context-index S. The new value of Index(S) is obtained from Table D.3 from the column labeled Next_Index_MPS, as that is the next index after an MPS renormalization. This next index is stored as the new value of Index(S) in the context storage at context-index S, and the value of Qe at this new Index(S) becomes the new Qe(S). MPS(S) does not change.

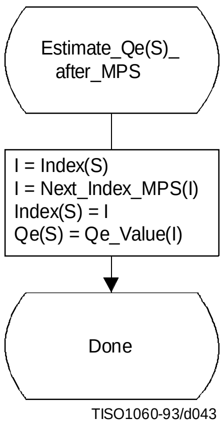


Figure D.5 – Probability estimation on MPS renormalization path

D.1.5.4 Estimation following renormalization after LPS

The procedure for estimating the probability on the LPS renormalization path is shown in Figure D.6. The procedure is similar to that of Figure D.5 except that when Switch_MPS(I) is 1, the sense of MPS(S) must be inverted.

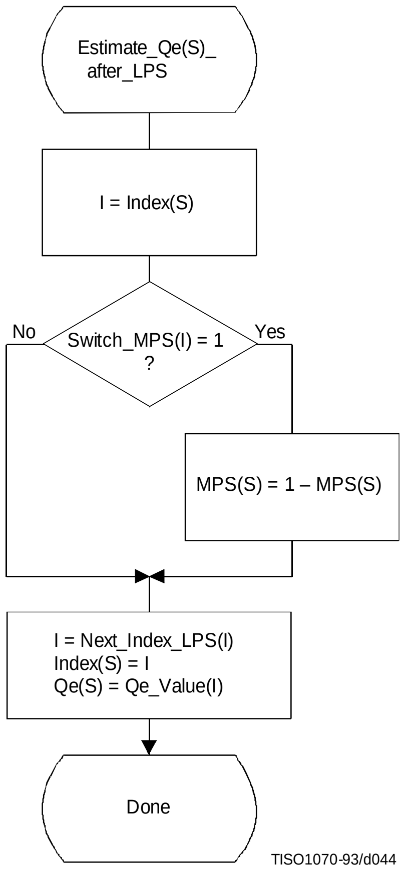


Figure D.6 – Probability estimation on LPS renormalization path

D.1.6 Renormalization in the encoder

The Renorm_e procedure for the encoder renormalization is shown in Figure D.7. Both the probability interval register A and the code register C are shifted, one bit at a time. The number of shifts is counted in the counter CT; when CT is zero, a byte of compressed data is removed from C by the procedure Byte_out and CT is reset to 8. Renormalization continues until A is no longer less than X’8000’.

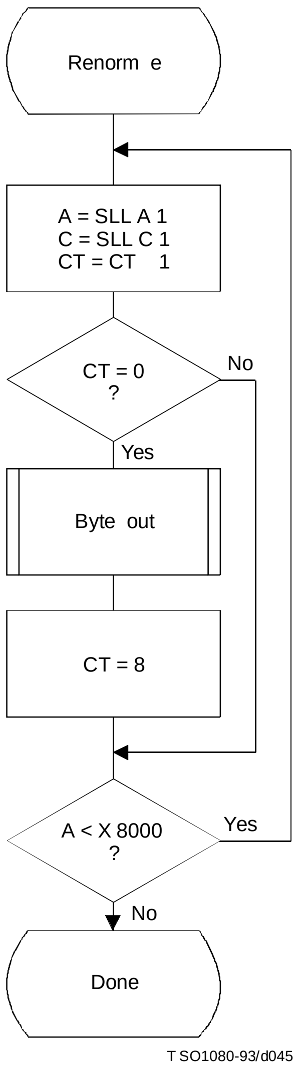


Figure D.7 – Encoder renormalization procedure

The Byte_out procedure used in Renorm_e is shown in Figure D.8. This procedure uses byte-stuffing procedures which prevent accidental generation of markers by the arithmetic encoding procedures. It also includes an example of a procedure for resolving carry-over. For simplicity of exposition, the buffer holding the entropy-coded segment is assumed to be large enough to contain the entire segment.

In Figure D.8 BP is the entropy-coded segment pointer and B is the compressed data byte pointed to by BP. T in Byte_out is a temporary variable which is used to hold the output byte and carry bit. ST is the stack counter which is used to count X’FF’ output bytes until any carry-over through the X’FF’ sequence has been resolved. The value of ST rarely exceeds 3. However, since the upper limit for the value of ST is bounded only by the total entropy-coded segment size, a precision of 32 bits is recommended for ST.

Since large values of ST represent a latent output of compressed data, the following procedure may be needed in high speed synchronous encoding systems for handling the burst of output data which occurs when the carry is resolved.

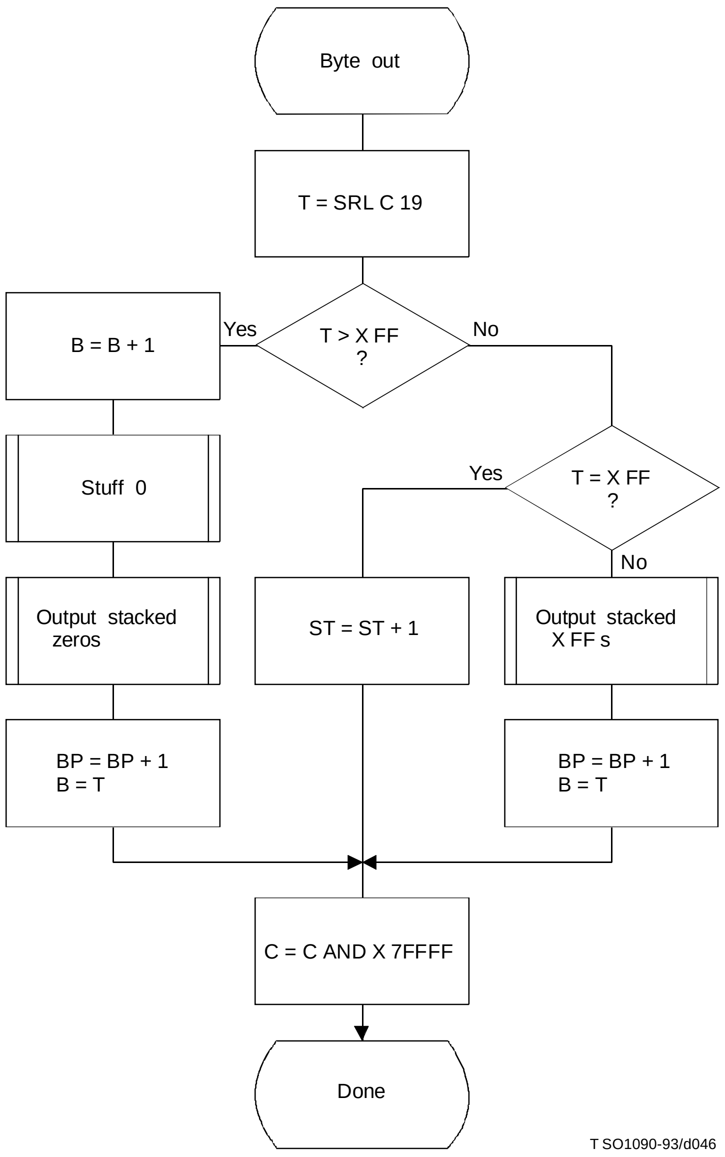


Figure D.8 – Byte_out procedure for encoder

When the stack count reaches an upper bound determined by output channel capacity, the stack is emptied and the stacked X'FF' bytes (and stuffed zero bytes) are added to the compressed data before the carry-over is resolved. If a carry-over then occurs, the carry is added to the final stuffed zero, thereby converting the final X'FF00' sequence to the X'FF01' temporary private marker. The entropy-coded segment must then be post-processed to resolve the carry-over and remove the temporary marker code. For any reasonable bound on ST this post processing is very unlikely.

Referring to Figure D.8, the shift of the code register by 19 bits aligns the output bits with the low order bits of T. The first test then determines if a carry-over has occurred. If so, the carry must be added to the previous output byte before advancing the segment pointer BP. The Stuff_0 procedure stuffs a zero byte whenever the addition of the carry to the data already in the entropy-coded segments creates a X'FF' byte. Any stacked output bytes – converted to zeros by the carry-over – are then placed in the entropy-coded segment. Note that when the output byte is later transferred from T to the entropy-coded segment (to byte B), the carry bit is ignored if it is set.

If a carry has not occurred, the output byte is tested to see if it is X'FF'. If so, the stack count ST is incremented, as the output must be delayed until the carry-over is resolved. If not, the carry-over has been resolved, and any stacked X'FF' bytes must then be placed in the entropy-coded segment. Note that a zero byte is stuffed following each X'FF'.

The procedures used by Byte_out are defined in Figures D.9 through D.11.

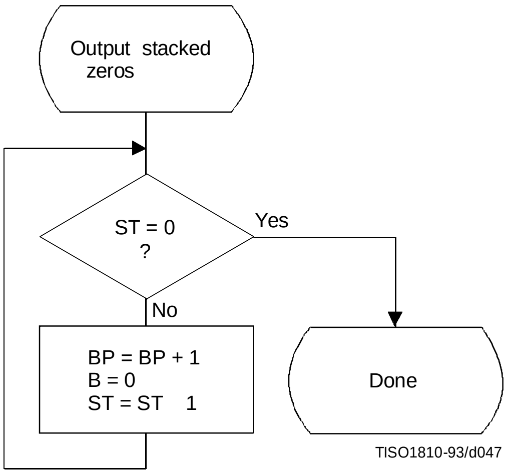


Figure D.9 – Output_stacked_zeros procedure for encoder

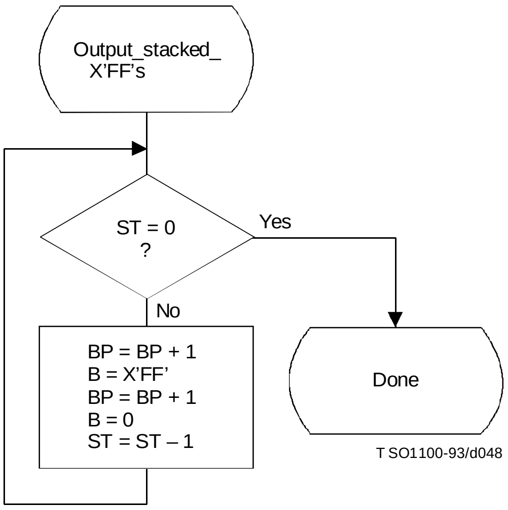


Figure D.10 – Output_stacked_X'FF's procedure for encoder

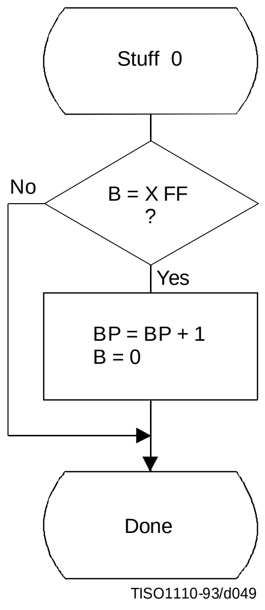


Figure D.11 – Stuff_0 procedure for encoder

D.1.7 Initialization of the encoder

The Initenc procedure is used to start the arithmetic coder. The basic steps are shown in Figure D.12.

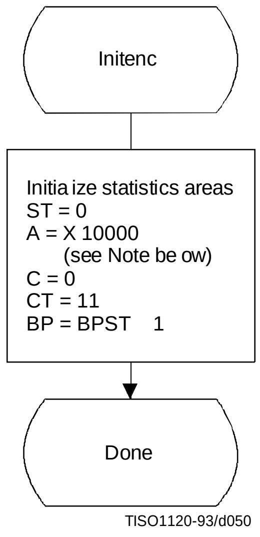


Figure D.12 – Initialization of the encoder

The probability estimation tables are defined by Table D.3. The statistics areas are initialized to an MPS sense of 0 and a Qe index of zero as defined by Table D.3. The stack count (ST) is cleared, the code register (C) is cleared, and the interval register is set to X’10000’. The counter (CT) is set to 11, reflecting the fact that when A is initialized to X’10000’ three spacer bits plus eight output bits in C must be filled before the first byte is removed. Note that BP is initialized to point to the byte before the start of the entropy-coded segment (which is at BPST). Note also that the statistics areas are initialized for all values of context-index S to MPS(S) = 0 and Index(S) = 0.

NOTE – Although the probability interval is initialized to X’10000’ in both Initenc and Initdec, the precision of the probability interval register can still be limited to 16 bits. When the precision of the interval register is 16 bits, it is initialized to zero.

D.1.8 Termination of encoding

The Flush procedure is used to terminate the arithmetic encoding procedures and prepare the entropy-coded segment for the addition of the X’FF’ prefix of the marker which follows the arithmetically coded data. Figure D.13 shows this flush procedure. The first step in the procedure is to set as many low order bits of the code register to zero as possible without pointing outside of the final interval. Then, the output byte is aligned by shifting it left by CT bits; Byte_out then removes it from C. C is then shifted left by 8 bits to align the second output byte and Byte_out is used a second time. The remaining low order bits in C are guaranteed to be zero, and these trailing zero bits shall not be written to the entropy-coded segment.

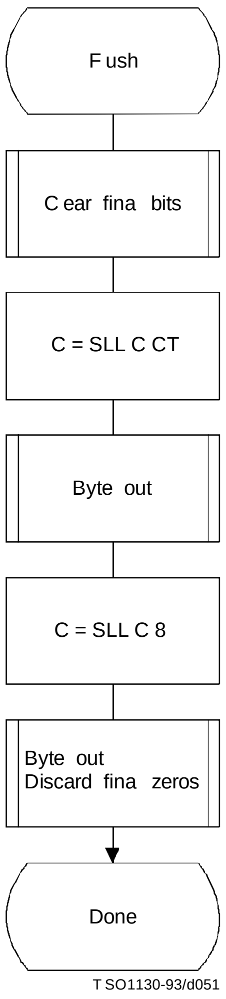


Figure D.13 – Flush procedure

Any trailing zero bytes already written to the entropy-coded segment and not preceded by a X’FF’ may, optionally, be discarded. This is done in the Discard_final_zeros procedure. Stuffed zero bytes shall not be discarded.

Entropy coded segments are always followed by a marker. For this reason, the final zero bits needed to complete decoding shall not be included in the entropy coded segment. Instead, when the decoder encounters a marker, zero bits shall be supplied to the decoding procedure until decoding is complete. This convention guarantees that when a DNL marker is used, the decoder will intercept it in time to correctly terminate the decoding procedure.

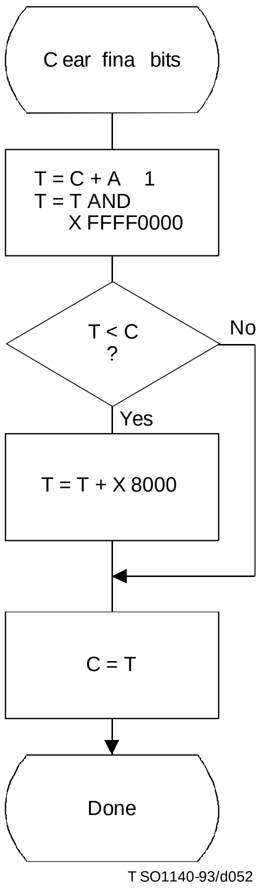


Figure D.14 – Clear_final_bits procedure in Flush

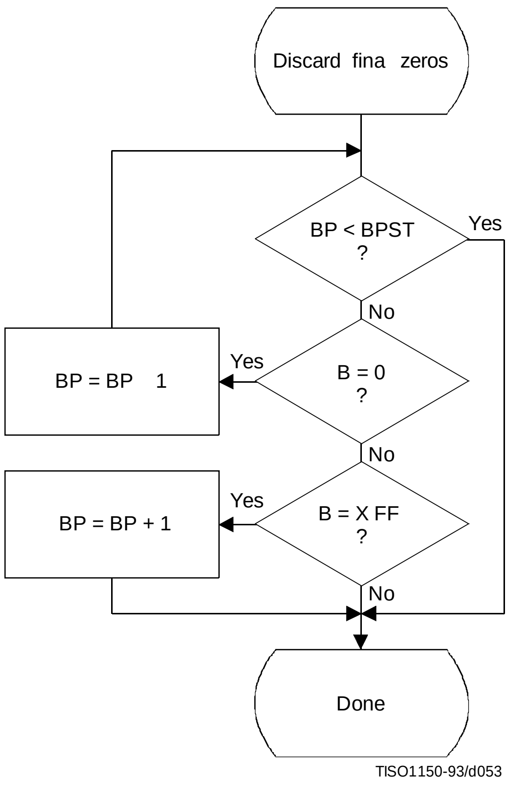


Figure D.15 – Discard_final_zeros procedure in Flush

D.2 Arithmetic decoding procedures

Two arithmetic decoding procedures are used for arithmetic decoding (see Table D.4).

The "Decode(S)" procedure decodes the binary decision for a given context-index S and returns a value of either 0 or 1. It is the inverse of the "Code_0(S)" and "Code_1(S)" procedures described in D.1. "Initdec" initializes the arithmetic coding entropy decoder.

Table D.4 – Procedures for binary arithmetic decoding

Procedure	Purpose
Decode(S)	Decode a binary decision with context-index S
Initdec	Initialize the decoder

D.2.1 Binary arithmetic decoding principles

The probability interval subdivision and sub-interval ordering defined for the arithmetic encoding procedures also apply to the arithmetic decoding procedures.

Since the bit stream always points within the current probability interval, the decoding process is a matter of determining, for each decision, which sub-interval is pointed to by the bit stream. This is done recursively, using the same probability interval sub-division process as in the encoder. Each time a decision is decoded, the decoder subtracts from the bit stream any interval the encoder added to the bit stream. Therefore, the code register in the decoder is a pointer into the current probability interval relative to the base of the interval.

If the size of the sub-interval allocated to the LPS is larger than the sub-interval allocated to the MPS, the encoder invokes the conditional exchange procedure. When the interval sizes are inverted in the decoder, the sense of the symbol decoded must be inverted.

D.2.2 Decoding conventions and approximations

The approximations and integer arithmetic defined for the probability interval subdivision in the encoder must also be used in the decoder. However, where the encoder would have added to the code register, the decoder subtracts from the code register.

D.2.3 Decoder code register conventions

The flow charts given in this section assume the register structures for the decoder as shown in Table D.5:

Table D.5 – Decoder register conventions

	MSB	LSB
Cx register	xxxxxxxx,	xxxxxxxx
C-low	bbbbbbbb,	00000000
A-register	aaaaaaaa,	aaaaaaaa

Cx and C-low can be regarded as one 32-bit C-register, in that renormalization of C shifts a bit of new data from bit 15 of C-low to bit 0 of Cx. However, the decoding comparisons use Cx alone. New data are inserted into the “b” bits of C-low one byte at a time.

NOTE – The comparisons shown in the various procedures use arithmetic comparisons, and therefore assume precisions greater than 16 bits for the variables. Unsigned (logical) comparisons should be used in 16-bit precision implementations.

D.2.4 The decode procedure

The decoder decodes one binary decision at a time. After decoding the decision, the decoder subtracts any amount from the code register that the encoder added. The amount left in the code register is the offset from the base of the current probability interval to the sub-interval allocated to the binary decisions not yet decoded. In the first test in the decode procedure shown in Figure D.16 the code register is compared to the size of the MPS sub-interval. Unless a conditional exchange is needed, this test determines whether the MPS or LPS for context-index S is decoded. Note that the LPS for context-index S is given by 1 – MPS(S).

When a renormalization is needed, the MPS/LPS conditional exchange may also be needed. For the LPS path, the conditional exchange procedure is shown in Figure D.17. Note that the probability estimation in the decoder is identical to the probability estimation in the encoder (Figures D.5 and D.6).

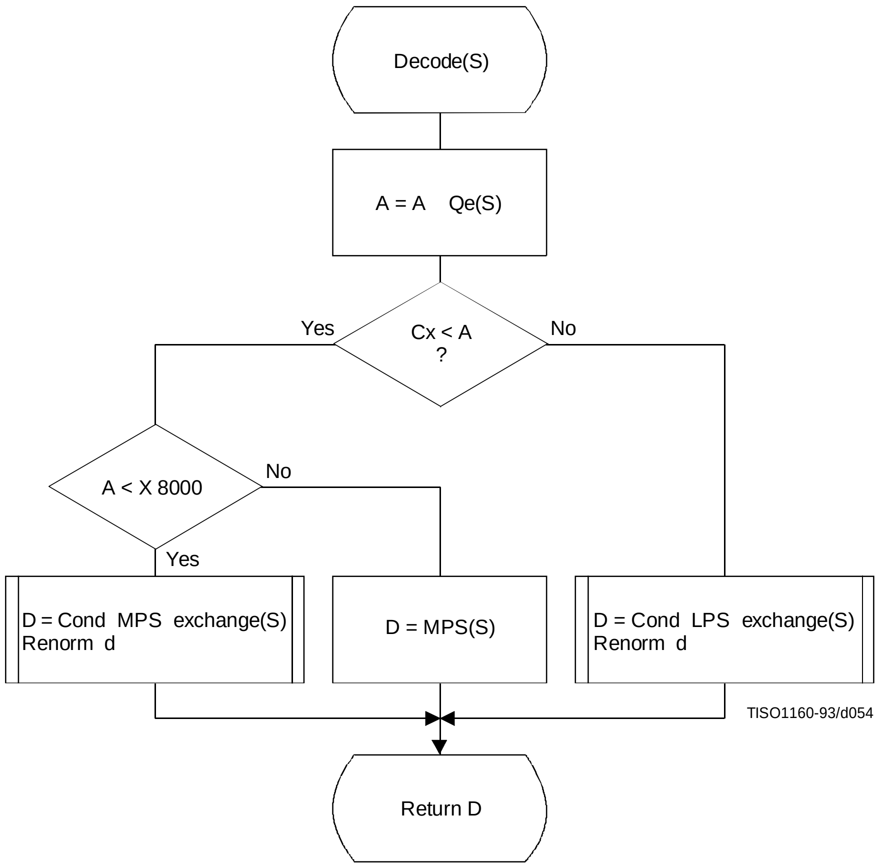


Figure D.16 – Decode(S) procedure

For the MPS path of the decoder the conditional exchange procedure is given in Figure D.18.

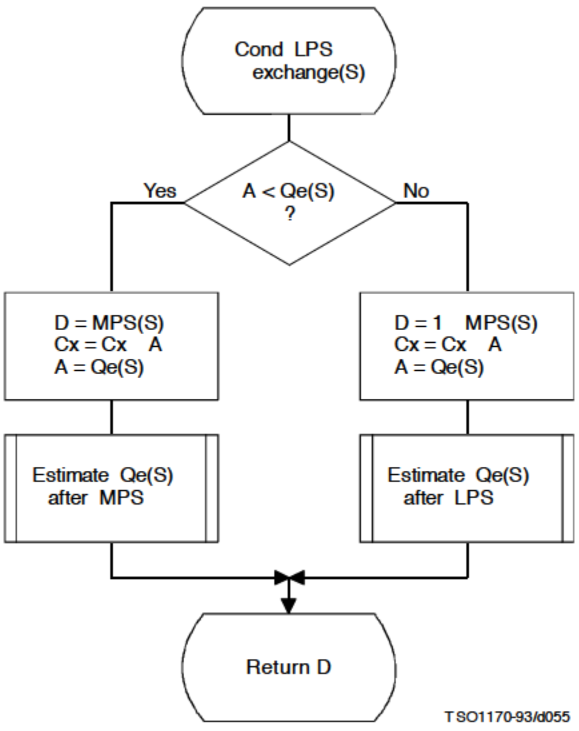


Figure D.17 – Decoder LPS path conditional exchange procedure

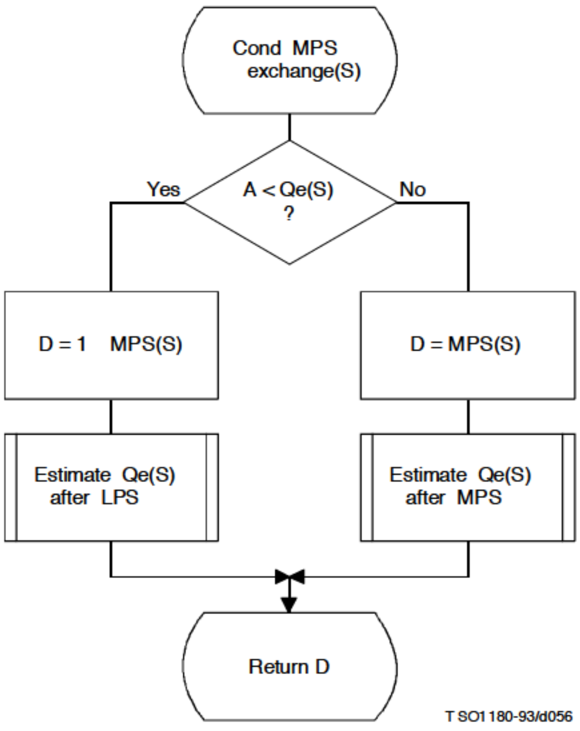


Figure D.18 – Decoder MPS path conditional exchange procedure

D.2.5 Probability estimation in the decoder

The procedures defined for obtaining a new LPS probability estimate in the encoder are also used in the decoder.

D.2.6 Renormalization in the decoder

The Renorm_d procedure for the decoder renormalization is shown in Figure D.19. CT is a counter which keeps track of the number of compressed bits in the C-low section of the C-register. When CT is zero, a new byte is inserted into C-low by the procedure Byte_in and CT is reset to 8.

Both the probability interval register A and the code register C are shifted, one bit at a time, until A is no longer less than X’8000’.

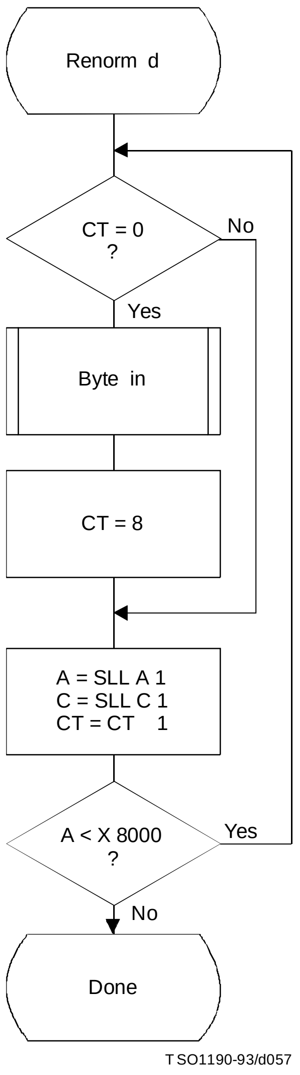


Figure D.19 – Decoder renormalization procedure

The Byte_in procedure used in Renorm_d is shown in Figure D.20. This procedure fetches one byte of data, compensating for the stuffed zero byte which follows any X’FF’ byte. It also detects the marker which must follow the entropy-coded segment. The C-register in this procedure is the concatenation of the Cx and C-low registers. For simplicity of exposition, the buffer holding the entropy-coded segment is assumed to be large enough to contain the entire segment.

B is the byte pointed to by the entropy-coded segment pointer BP. BP is first incremented. If the new value of B is not a X’FF’, it is inserted into the high order 8 bits of C-low.

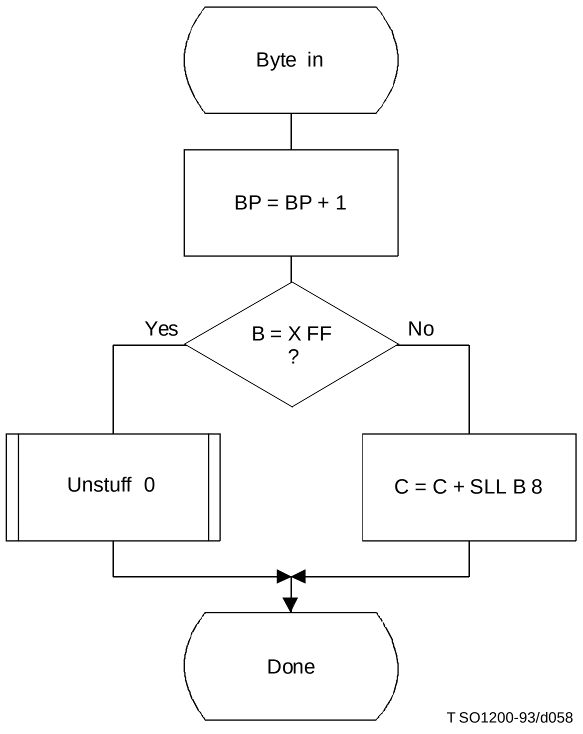


Figure D.20 – Byte_in procedure for decoder

The Unstuff_0 procedure is shown in Figure D.21. If the new value of B is X’FF’, BP is incremented to point to the next byte and this next B is tested to see if it is zero. If so, B contains a stuffed byte which must be skipped. The zero B is ignored, and the X’FF’ B value which preceded it is inserted in the C-register.

If the value of B after a X’FF’ byte is not zero, then a marker has been detected. The marker is interpreted as required and the entropy-coded segment pointer is adjusted (“Adjust BP” in Figure D.21) so that 0-bytes will be fed to the decoder until decoding is complete. One way of accomplishing this is to point BP to the byte preceding the marker which follows the entropy-coded segment.

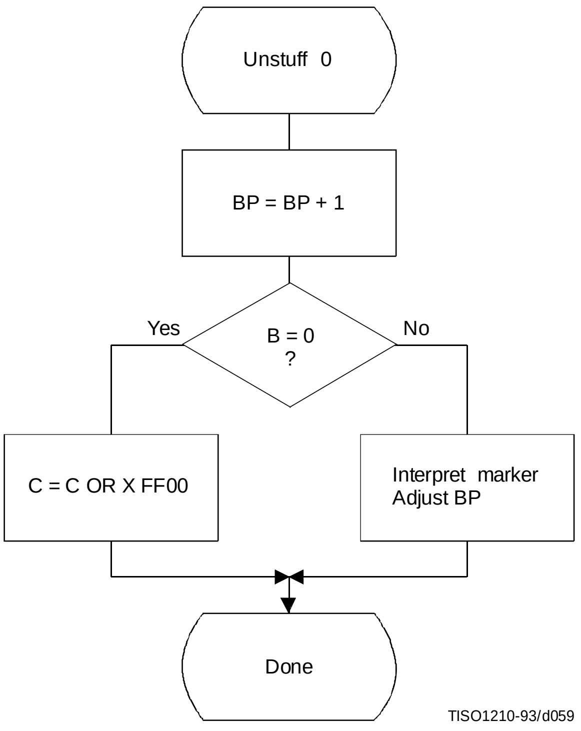


Figure D.21 – Unstuff_0 procedure for decoder

D.2.7 Initialization of the decoder

The Initdec procedure is used to start the arithmetic decoder. The basic steps are shown in Figure D.22.

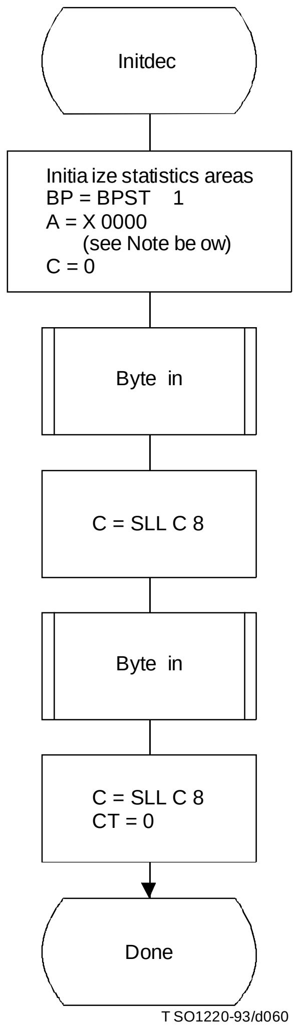


Figure D.22 – Initialization of the decoder

The estimation tables are defined by Table D.3. The statistics areas are initialized to an MPS sense of 0 and a Qe index of zero as defined by Table D.3. BP, the pointer to the entropy-coded segment, is then initialized to point to the byte before the start of the entropy-coded segment at BPST, and the interval register is set to the same starting value as in the encoder. The first byte of compressed data is fetched and shifted into Cx. The second byte is then fetched and shifted into Cx. The count is set to zero, so that a new byte of data will be fetched by Renorm_d.

NOTE – Although the probability interval is initialized to X’10000’ in both Initenc and Initdec, the precision of the probability interval register can still be limited to 16 bits. When the precision of the interval register is 16 bits, it is initialized to zero.

D.3 Bit ordering within bytes

The arithmetically encoded entropy-coded segment is an integer of variable length. Therefore, the ordering of bytes and the bit ordering within bytes is the same as for parameters (see B.1.1.1).

Annex E

Encoder and decoder control procedures

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes the encoder and decoder control procedures for the sequential, progressive, and lossless modes of operation.

The encoding and decoding control procedures for the hierarchical processes are specified in Annex J.

NOTES

1 There is **no requirement** in this Specification that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

2 Implementation-specific setup steps are not indicated in this annex and may be necessary.

E.1 Encoder control procedures

E.1.1 Control procedure for encoding an image

The encoder control procedure for encoding an image is shown in Figure E.1.

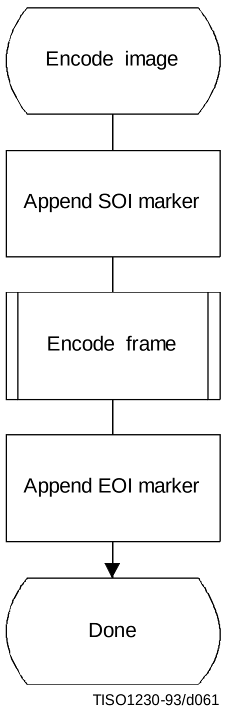


Figure E.1 – Control procedure for encoding an image

E.1.2 Control procedure for encoding a frame

In all cases where markers are appended to the compressed data, optional X'FF' fill bytes may precede the marker.

The control procedure for encoding a frame is oriented around the scans in the frame. The frame header is first appended, and then the scans are coded. Table specifications and other marker segments may precede the SOF_n marker, as indicated by [tables/miscellaneous] in Figure E.2.

Figure E.2 shows the encoding process frame control procedure.

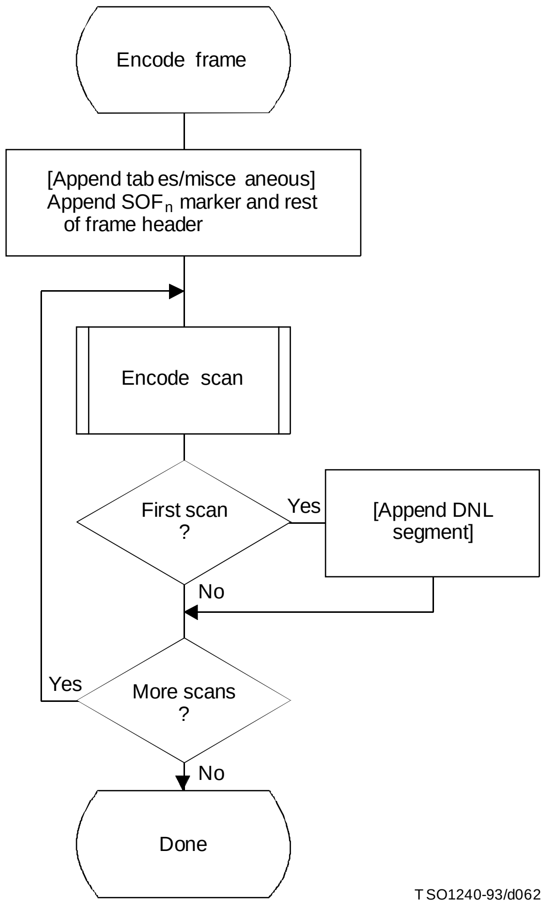


Figure E.2 – Control procedure for encoding a frame

E.1.3 Control procedure for encoding a scan

A scan consists of a single pass through the data of each component in the scan. Table specifications and other marker segments may precede the SOS marker. If more than one component is coded in the scan, the data are interleaved. If restart is enabled, the data are segmented into restart intervals. If restart is enabled, a RST_m marker is placed in the coded data between restart intervals. If restart is disabled, the control procedure is the same, except that the entire scan contains a single restart interval. The compressed image data generated by a scan is always followed by a marker, either the EOI marker or the marker of the next marker segment.

Figure E.3 shows the encoding process scan control procedure. The loop is terminated when the encoding process has coded the number of restart intervals which make up the scan. “m” is the restart interval modulo counter needed for the RST_m marker. The modulo arithmetic for this counter is shown after the “Append RST_m marker” procedure.

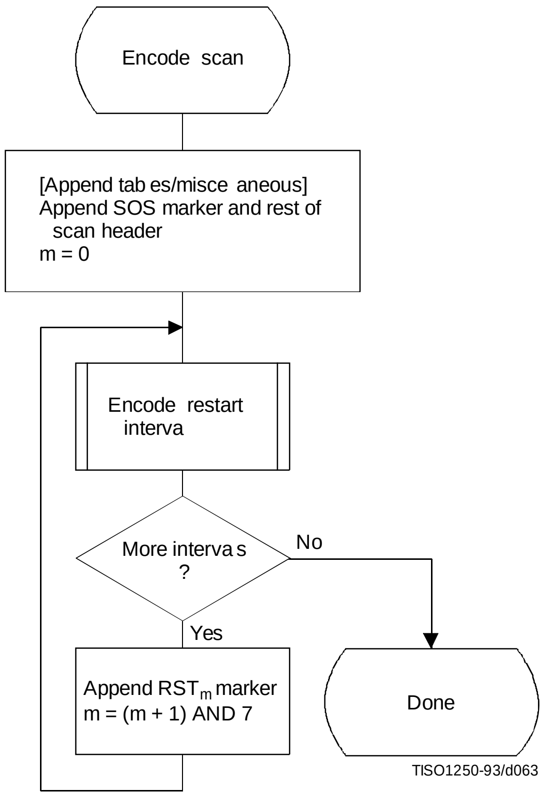


Figure E.3 – Control procedure for encoding a scan

E.1.4 Control procedure for encoding a restart interval

Figure E.4 shows the encoding process control procedure for a restart interval. The loop is terminated either when the encoding process has coded the number of minimum coded units (MCU) in the restart interval or when it has completed the image scan.

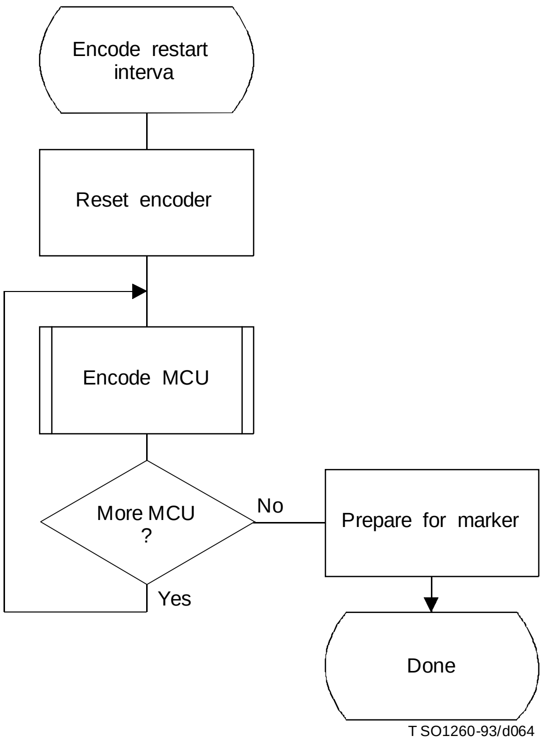


Figure E.4 – Control procedure for encoding a restart interval

The "Reset_encoder" procedure consists at least of the following:

a) if arithmetic coding is used, initialize the arithmetic encoder using the "Initenc" procedure described in D.1.7;

b) for DCT-based processes, set the DC prediction (PRED) to zero for all components in the scan (see F.1.1.5.1);

c) for lossless processes, reset the prediction to a default value for all components in the scan (see H.1.1);

d) do all other implementation-dependent setups that may be necessary.

The procedure "Prepare_for_marker" terminates the entropy-coded segment by:

a) padding a Huffman entropy-coded segment with 1-bits to complete the final byte (and if needed stuffing a zero byte) (see F.1.2.3); or

b) invoking the procedure "Flush" (see D.1.8) to terminate an arithmetic entropy-coded segment.

NOTE – The number of minimum coded units (MCU) in the final restart interval must be adjusted to match the number of MCU in the scan. The number of MCU is calculated from the frame and scan parameters. (See Annex B.)

E.1.5 Control procedure for encoding a minimum coded unit (MCU)

The minimum coded unit is defined in A.2. Within a given MCU the data units are coded in the order in which they occur in the MCU. The control procedure for encoding a MCU is shown in Figure E.5.

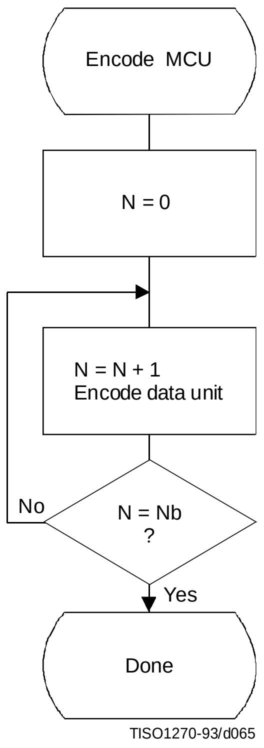


Figure E.5 – Control procedure for encoding a minimum coded unit (MCU)

In Figure E.5, Nb refers to the number of data units in the MCU. The order in which data units occur in the MCU is defined in A.2. The data unit is an 8×8 block for DCT-based processes, and a single sample for lossless processes.

The procedures for encoding a data unit are specified in Annexes F, G, and H.

E.2 Decoder control procedures

E.2.1 Control procedure for decoding compressed image data

Figure E.6 shows the decoding process control for compressed image data.

Decoding control centers around identification of various markers. The first marker must be the SOI (Start Of Image) marker. The "Decoder_setup" procedure resets the restart interval (Ri = 0) and, if the decoder has arithmetic decoding capabilities, sets the conditioning tables for the arithmetic coding to their default values. (See F.1.4.4.1.4 and F.1.4.4.2.1.) The next marker is normally a SOF_n (Start Of Frame) marker; if this is not found, one of the marker segments listed in Table E.1 has been received.

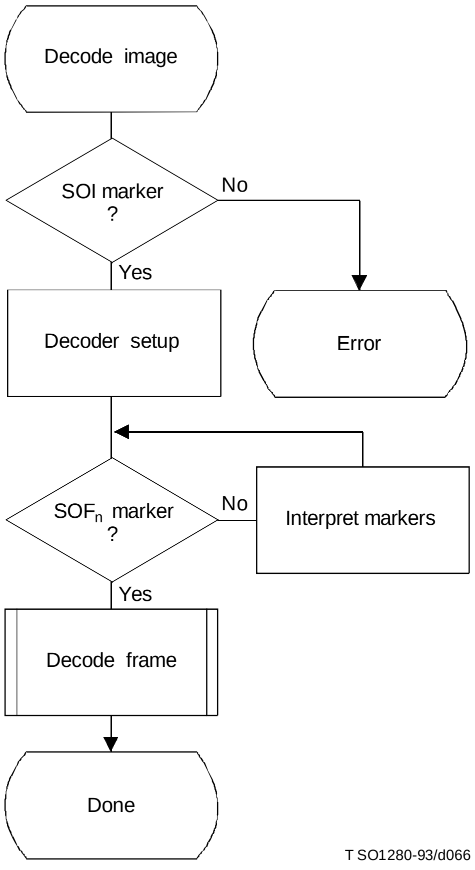


Figure E.6 – Control procedure for decoding compressed image data

Table E.1 – Markers recognized by "Interpret markers"

Marker	Purpose
DHT	Define Huffman Tables
DAC	Define Arithmetic Conditioning
DQT	Define Quantization Tables
DRI	Define Restart Interval
APP$_n$	Application defined marker
COM	Comment

Note that optional X'FF' fill bytes which may precede any marker shall be discarded before determining which marker is present.

The additional logic to interpret these various markers is contained in the box labeled “Interpret markers”. DHT markers shall be interpreted by processes using Huffman coding. DAC markers shall be interpreted by processes using arithmetic coding. DQT markers shall be interpreted by DCT-based decoders. DRI markers shall be interpreted by all decoders. APPn and COM markers shall be interpreted only to the extent that they do not interfere with the decoding.

By definition, the procedures in “Interpret markers” leave the system at the next marker. Note that if the expected SOI marker is missing at the start of the compressed image data, an error condition has occurred. The techniques for detecting and managing error conditions can be as elaborate or as simple as desired.

E.2.2 Control procedure for decoding a frame

Figure E.7 shows the control procedure for the decoding of a frame.

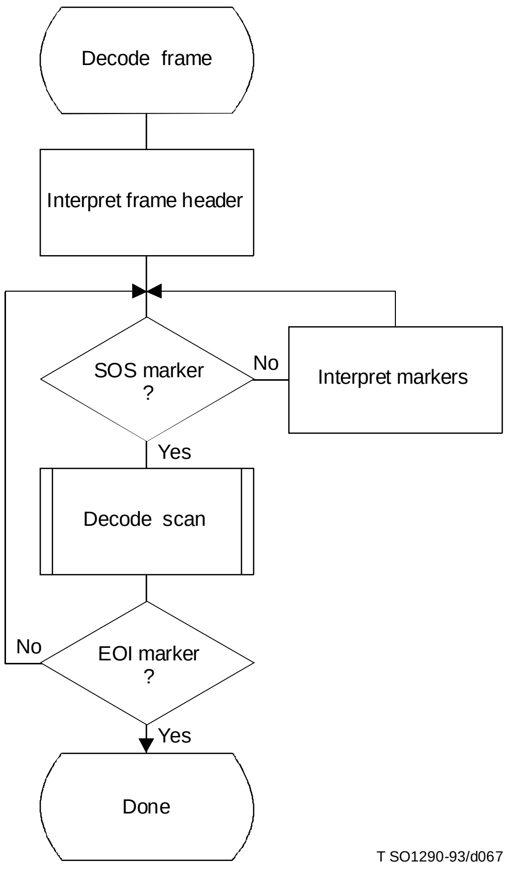


Figure E.7 – Control procedure for decoding a frame

The loop is terminated if the EOI marker is found at the end of the scan.

The markers recognized by “Interpret markers” are listed in Table E.1. Subclause E.2.1 describes the extent to which the various markers shall be interpreted.

E.2.3 Control procedure for decoding a scan

Figure E.8 shows the decoding of a scan.

The loop is terminated when the expected number of restart intervals has been decoded.

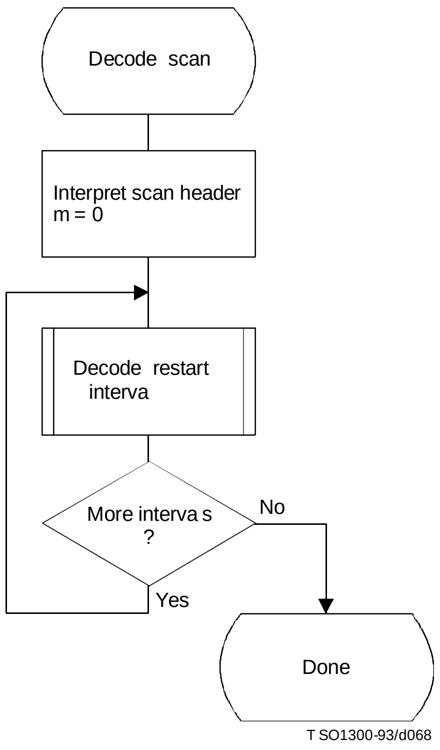


Figure E.8 – Control procedure for decoding a scan

E.2.4 Control procedure for decoding a restart interval

The procedure for decoding a restart interval is shown in Figure E.9. The "Reset_decoder" procedure consists at least of the following:

a) if arithmetic coding is used, initialize the arithmetic decoder using the "Initdec" procedure described in D.2.7;

b) for DCT-based processes, set the DC prediction (PRED) to zero for all components in the scan (see F.2.1.3.1);

c) for lossless process, reset the prediction to a default value for all components in the scan (see H.2.1);

d) do all other implementation-dependent setups that may be necessary.

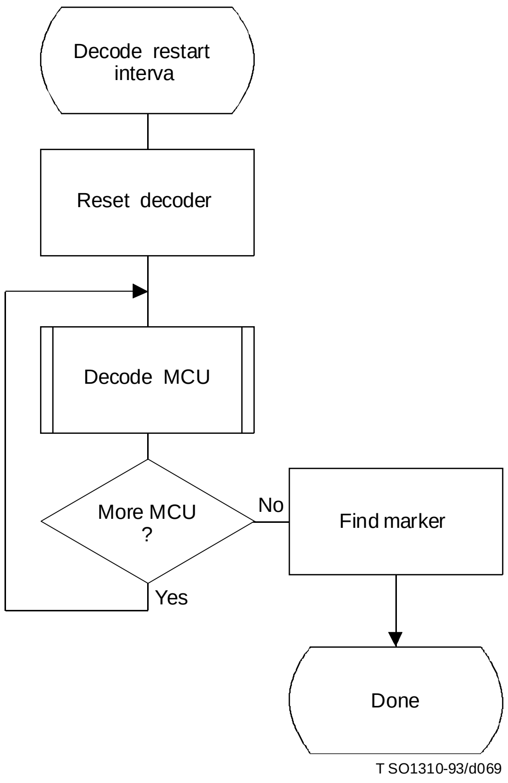


Figure E.9 – Control procedure for decoding a restart interval

At the end of the restart interval, the next marker is located. If a problem is detected in locating this marker, error handling procedures may be invoked. While such procedures are optional, the decoder shall be able to correctly recognize restart markers in the compressed data and reset the decoder when they are encountered. The decoder shall also be able to recognize the DNL marker, set the number of lines defined in the DNL segment, and end the "Decode_restart_interval" procedure.

NOTE – The final restart interval may be smaller than the size specified by the DRI marker segment, as it includes only the number of MCUs remaining in the scan.

E.2.5 Control procedure for decoding a minimum coded unit (MCU)

The procedure for decoding a minimum coded unit (MCU) is shown in Figure E.10.

In Figure E.10 Nb is the number of data units in a MCU.

The procedures for decoding a data unit are specified in Annexes F, G, and H.

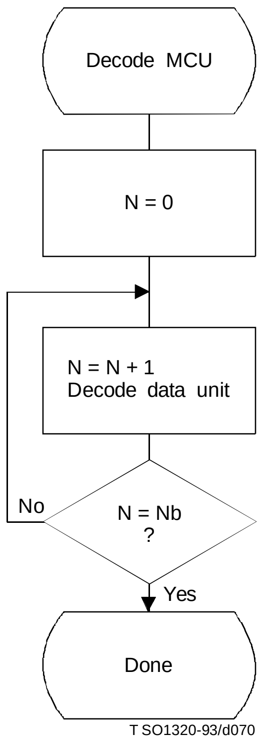


Figure E.10 – Control procedure for decoding a minimum coded unit (MCU)

Annex F

Sequential DCT-based mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the sequential DCT-based mode of operation:

1) baseline sequential;
2) extended sequential, Huffman coding, 8-bit sample precision;
3) extended sequential, arithmetic coding, 8-bit sample precision;
4) extended sequential, Huffman coding, 12-bit sample precision;
5) extended sequential, arithmetic coding, 12-bit sample precision.

For each of these, the encoding process is specified in F.1, and the decoding process is specified in F.2. The functional specification is presented by means of specific flow charts for the various procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

F.1 Sequential DCT-based encoding processes

F.1.1 Sequential DCT-based control procedures and coding models

F.1.1.1 Control procedures for sequential DCT-based encoders

The control procedures for encoding an image and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.1 to E.5. The procedure for encoding a MCU (see Figure E.5) repetitively calls the procedure for encoding a data unit. For DCT-based encoders the data unit is an 8×8 block of samples.

F.1.1.2 Procedure for encoding an 8×8 block data unit

For the sequential DCT-based processes encoding an 8×8 block data unit consists of the following procedures:

a) level shift, calculate forward 8×8 DCT and quantize the resulting coefficients using table destination specified in frame header;
b) encode DC coefficient for 8×8 block using DC table destination specified in scan header;
c) encode AC coefficients for 8×8 block using AC table destination specified in scan header.

F.1.1.3 Level shift and forward DCT (FDCT)

The mathematical definition of the FDCT is given in A.3.3.

Prior to computing the FDCT the input data are level shifted to a signed two’s complement representation as described in A.3.1. For 8-bit input precision the level shift is achieved by subtracting 128. For 12-bit input precision the level shift is achieved by subtracting 2048.

F.1.1.4 Quantization of the FDCT

The uniform quantization procedure described in Annex A is used to quantize the DCT coefficients. One of four quantization tables may be used by the encoder. No default quantization tables are specified in this Specification. However, some typical quantization tables are given in Annex K.

The quantized DCT coefficient values are signed, two’s complement integers with 11-bit precision for 8-bit input precision and 15-bit precision for 12-bit input precision.

F.1.1.5 Encoding models for the sequential DCT procedures

The two dimensional array of quantized DCT coefficients is rearranged in a zig-zag sequence order defined in A.3.6. The zig-zag order coefficients are denoted ZZ (0) through ZZ(63) with:

$$ZZ(0) = Sq_{00}, ZZ(1) = Sq_{01}, ZZ(2) = Sq_{10}, \bullet, \bullet, \bullet, ZZ(63) = Sq_{77}$$

Sq_{vu} are defined in Figure A.6.

Two coding procedures are used, one for the DC coefficient ZZ(0) and the other for the AC coefficients ZZ(1)..ZZ(63). The coefficients are encoded in the order in which they occur in zig-zag sequence order, starting with the DC coefficient. The coefficients are represented as two’s complement integers.

F.1.1.5.1 Encoding model for DC coefficients

The DC coefficients are coded differentially, using a one-dimensional predictor, PRED, which is the quantized DC value from the most recently coded 8×8 block from the same component. The difference, DIFF, is obtained from

$$DIFF = ZZ(0) - PRED$$

At the beginning of the scan and at the beginning of each restart interval, the prediction for the DC coefficient prediction is initialized to 0. (Recall that the input data have been level shifted to two’s complement representation.)

F.1.1.5.2 Encoding model for AC coefficients

Since many coefficients are zero, runs of zeros are identified and coded efficiently. In addition, if the remaining coefficients in the zig-zag sequence order are all zero, this is coded explicitly as an end-of-block (EOB).

F.1.2 Baseline Huffman encoding procedures

The baseline encoding procedure is for 8-bit sample precision. The encoder may employ up to two DC and two AC Huffman tables within one scan.

F.1.2.1 Huffman encoding of DC coefficients

F.1.2.1.1 Structure of DC code table

The DC code table consists of a set of Huffman codes (maximum length 16 bits) and appended additional bits (in most cases) which can code any possible value of DIFF, the difference between the current DC coefficient and the prediction. The Huffman codes for the difference categories are generated in such a way that no code consists entirely of 1-bits (X’FF’ prefix marker code avoided).

The two’s complement difference magnitudes are grouped into 12 categories, SSSS, and a Huffman code is created for each of the 12 difference magnitude categories (see Table F.1).

For each category, except SSSS = 0, an additional bits field is appended to the code word to uniquely identify which difference in that category actually occurred. The number of extra bits is given by SSSS; the extra bits are appended to the LSB of the preceding Huffman code, most significant bit first. When DIFF is positive, the SSSS low order bits of DIFF are appended. When DIFF is negative, the SSSS low order bits of (DIFF – 1) are appended. Note that the most significant bit of the appended bit sequence is 0 for negative differences and 1 for positive differences.

F.1.2.1.2 Defining Huffman tables for the DC coefficients

The syntax for specifying the Huffman tables is given in Annex B. The procedure for creating a code table from this information is described in Annex C. No more than two Huffman tables may be defined for coding of DC coefficients. Two examples of Huffman tables for coding of DC coefficients are provided in Annex K.

Table F.1 – Difference magnitude categories for DC coding

SSSS	DIFF values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047

F.1.2.1.3 Huffman encoding procedures for DC coefficients

The encoding procedure is defined in terms of a set of extended tables, XHUFCO and XHUFSI, which contain the complete set of Huffman codes and sizes for all possible difference values. For full 12-bit precision the tables are relatively large. For the baseline system, however, the precision of the differences may be small enough to make this description practical.

XHUFCO and XHUFSI are generated from the encoder tables EHUFCO and EHUFSI (see Annex C) by appending to the Huffman codes for each difference category the additional bits that completely define the difference. By definition, XHUFCO and XHUFSI have entries for each possible difference value. XHUFCO contains the concatenated bit pattern of the Huffman code and the additional bits field; XHUFSI contains the total length in bits of this concatenated bit pattern. Both are indexed by DIFF, the difference between the DC coefficient and the prediction.

The Huffman encoding procedure for the DC difference, DIFF, is:

SIZE = XHUFSI(DIFF)

CODE = XHUFCO(DIFF)

code SIZE bits of CODE

where DC is the quantized DC coefficient value and PRED is the predicted quantized DC value. The Huffman code (CODE) (including any additional bits) is obtained from XHUFCO and SIZE (length of the code including additional bits) is obtained from XHUFSI, using DIFF as the index to the two tables.

F.1.2.2 Huffman encoding of AC coefficients

F.1.2.2.1 Structure of AC code table

Each non-zero AC coefficient in ZZ is described by a composite 8-bit value, RS, of the form

RS = binary ’RRRRSSSS’

The 4 least significant bits, 'SSSS', define a category for the amplitude of the next non-zero coefficient in ZZ, and the 4 most significant bits, 'RRRR', give the position of the coefficient in ZZ relative to the previous non-zero coefficient (i.e. the run-length of zero coefficients between non-zero coefficients). Since the run length of zero coefficients may exceed 15, the value 'RRRRSSSS' = X'F0' is defined to represent a run length of 15 zero coefficients followed by a coefficient of zero amplitude. (This can be interpreted as a run length of 16 zero coefficients.) In addition, a special value 'RRRRSSSS' = '00000000' is used to code the end-of-block (EOB), when all remaining coefficients in the block are zero.

The general structure of the code table is illustrated in Figure F.1. The entries marked "N/A" are undefined for the baseline procedure.

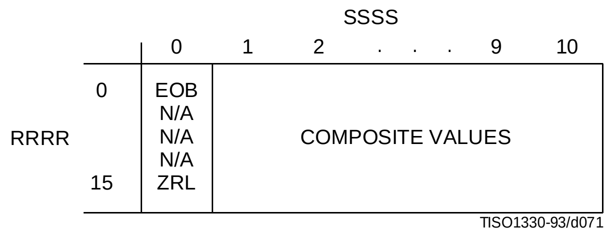


Figure F.1 – Two-dimensional value array for Huffman coding

The magnitude ranges assigned to each value of SSSS are defined in Table F.2.

Table F.2 – Categories assigned to coefficient values

SSSS	AC coefficients
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023

The composite value, RRRRSSSS, is Huffman coded and each Huffman code is followed by additional bits which specify the sign and exact amplitude of the coefficient.

The AC code table consists of one Huffman code (maximum length 16 bits, not including additional bits) for each possible composite value. The Huffman codes for the 8-bit composite values are generated in such a way that no code consists entirely of 1-bits.

The format for the additional bits is the same as in the coding of the DC coefficients. The value of SSSS gives the number of additional bits required to specify the sign and precise amplitude of the coefficient. The additional bits are either the low-order SSSS bits of ZZ(K) when ZZ(K) is positive or the low-order SSSS bits of ZZ(K) – 1 when ZZ(K) is negative. ZZ(K) is the *K*th coefficient in the zig-zag sequence of coefficients being coded.

F.1.2.2.2 Defining Huffman tables for the AC coefficients

The syntax for specifying the Huffman tables is given in Annex B. The procedure for creating a code table from this information is described in Annex C.

In the baseline system no more than two Huffman tables may be defined for coding of AC coefficients. Two examples of Huffman tables for coding of AC coefficients are provided in Annex K.

F.1.2.2.3 Huffman encoding procedures for AC coefficients

As defined in Annex C, the Huffman code table is assumed to be available as a pair of tables, EHUFCO (containing the code bits) and EHUFSI (containing the length of each code in bits), both indexed by the composite value defined above.

The procedure for encoding the AC coefficients in a block is shown in Figures F.2 and F.3. In Figure F.2, K is the index to the zig-zag scan position and R is the run length of zero coefficients.

The procedure “Append EHUFSI(X’F0’) bits of EHUFCO(X’F0’)” codes a run of 16 zero coefficients (ZRL code of Figure F.1). The procedure “Code EHUFSI(0) bits of EHUFCO(0)” codes the end-of-block (EOB code). If the last coefficient (K = 63) is not zero, the EOB code is bypassed.

CSIZE is a procedure which maps an AC coefficient to the SSSS value as defined in Table F.2.

F.1.2.3 Byte stuffing

In order to provide code space for marker codes which can be located in the compressed image data without decoding, byte stuffing is used.

Whenever, in the course of normal encoding, the byte value X’FF’ is created in the code string, a X’00’ byte is stuffed into the code string.

If a X’00’ byte is detected after a X’FF’ byte, the decoder must discard it. If the byte is not zero, a marker has been detected, and shall be interpreted to the extent needed to complete the decoding of the scan.

Byte alignment of markers is achieved by padding incomplete bytes with 1-bits. If padding with 1-bits creates a X’FF’ value, a zero byte is stuffed before adding the marker.

F.1.3 Extended sequential DCT-based Huffman encoding process for 8-bit sample precision

This process is identical to the Baseline encoding process described in F.1.2, with the exception that the number of sets of Huffman table destinations which may be used within the same scan is increased to four. Four DC and four AC Huffman table destinations is the maximum allowed by this Specification.

F.1.4 Extended sequential DCT-based arithmetic encoding process for 8-bit sample precision

This subclause describes the use of arithmetic coding procedures in the sequential DCT-based encoding process.

NOTE – The arithmetic coding procedures in this Specification are defined for the maximum precision to encourage interchangeability.

The arithmetic coding extensions have the same DCT model as the Baseline DCT encoder. Therefore, Annex F.1.1 also applies to arithmetic coding. As with the Huffman coding technique, the binary arithmetic coding technique is lossless. It is possible to transcode between the two systems without either FDCT or IDCT computations, and without modification of the reconstructed image.

The basic principles of adaptive binary arithmetic coding are described in Annex D. Up to four DC and four AC conditioning table destinations and associated statistics areas may be used within one scan.

The arithmetic encoding procedures for encoding binary decisions, initializing the statistics area, initializing the encoder, terminating the code string, and adding restart markers are listed in Table D.1 of Annex D.

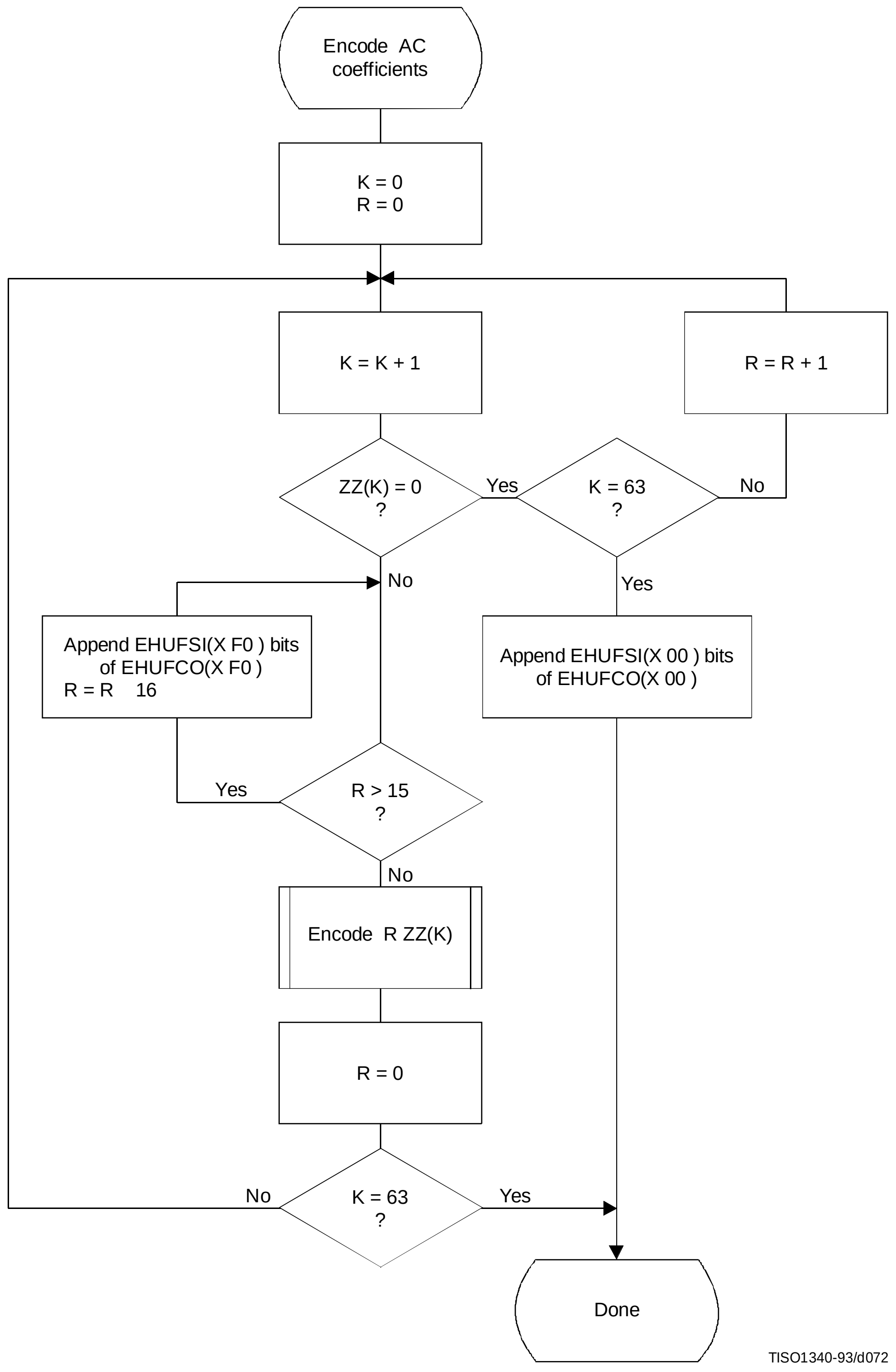


Figure F.2 – Procedure for sequential encoding of AC coefficients with Huffman coding

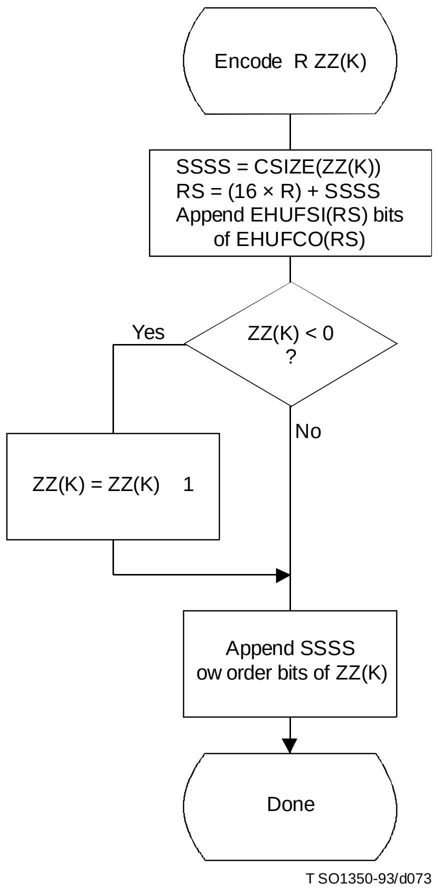


Figure F.3 – Sequential encoding of a non-zero AC coefficient

Some of the procedures in Table D.1 are used in the higher level control structure for scans and restart intervals described in Annex E. At the beginning of scans and restart intervals, the probability estimates used in the arithmetic coder are reset to the standard initial value as part of the Initenc procedure which restarts the arithmetic coder. At the end of scans and restart intervals, the Flush procedure is invoked to empty the code register before the next marker is appended.

F.1.4.1 Arithmetic encoding of DC coefficients

The basic structure of the decision sequence for encoding a DC difference value, DIFF, is shown in Figure F.4.

The context-index S0 and other context-indices used in the DC coding procedures are defined in Table F.4 (see F.1.4.4.1.3). A 0-decision is coded if the difference value is zero and a 1-decision is coded if the difference is not zero. If the difference is not zero, the sign and magnitude are coded using the procedure Encode_V(S0), which is described in F.1.4.3.1.

F.1.4.2 Arithmetic encoding of AC coefficients

The AC coefficients are coded in the order in which they occur in the zig-zag sequence ZZ(1,...,63). An end-of-block (EOB) binary decision is coded before coding the first AC coefficient in ZZ, and after each non-zero coefficient. If the EOB occurs, all remaining coefficients in ZZ are zero. Figure F.5 illustrates the decision sequence. The equivalent procedure for the Huffman coder is found in Figure F.2.

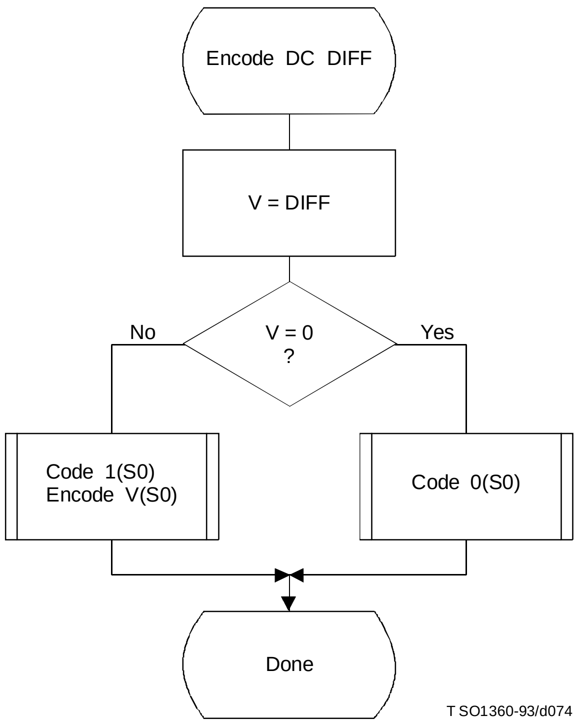


Figure F.4 – Coding model for arithmetic coding of DC difference

The context-indices SE and S0 used in the AC coding procedures are defined in Table F.5 (see F.1.4.4.2). In Figure F.5, K is the index to the zig-zag sequence position. For the sequential scan, Kmin is 1 and Se is 63. The V = 0 decision is part of a loop which codes runs of zero coefficients. Whenever the coefficient is non-zero, "Encode_V(S0)" codes the sign and magnitude of the coefficient. Each time a non-zero coefficient is coded, it is followed by an EOB decision. If the EOB occurs, a 1-decision is coded to indicate that the coding of the block is complete. If the coefficient for K = Se is not zero, the EOB decision is skipped.

F.1.4.3 Encoding the binary decision sequence for non-zero DC differences and AC coefficients

Both the DC difference and the AC coefficients are represented as signed two's complement integer values. The decomposition of these signed integer values into a binary decision tree is done in the same way for both the DC and AC coding models.

Although the binary decision trees for this section of the DC and AC coding models are the same, the statistical models for assigning statistics bins to the binary decisions in the tree are quite different.

F.1.4.3.1 Structure of the encoding decision sequence

The encoding sequence can be separated into three procedures, a procedure which encodes the sign, a second procedure which identifies the magnitude category, and a third procedure which identifies precisely which magnitude occurred within the category identified in the second procedure.

At the point where the binary decision sequence in Encode_V(S0) starts, the coefficient or difference has already been determined to be non-zero. That determination was made in the procedures in Figures F.4 and F.5.

Denoting either DC differences (DIFF) or AC coefficients as V, the non-zero signed integer value of V is encoded by the sequence shown in Figure F.6. This sequence first codes the sign of V. It then (after converting V to a magnitude and decrementing it by 1 to give Sz) codes the magnitude category of Sz (code_log2_Sz), and then codes the low order magnitude bits (code_Sz_bits) to identify the exact magnitude value.

There are two significant differences between this sequence and the similar set of operations described in F.1.2 for Huffman coding. First, the sign is encoded before the magnitude category is identified, and second, the magnitude is decremented by 1 before the magnitude category is identified.

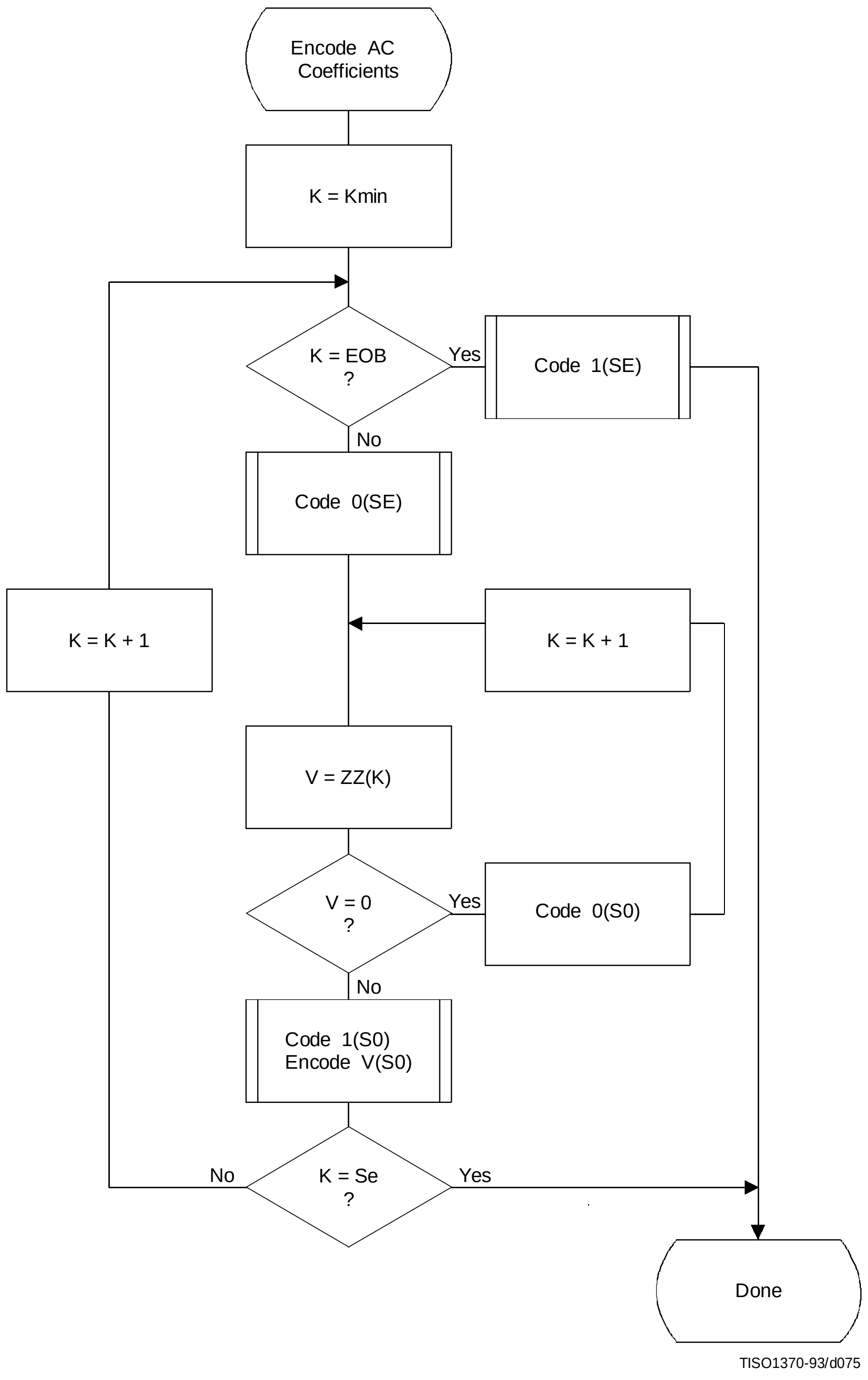


Figure F.5 – AC coding model for arithmetic coding

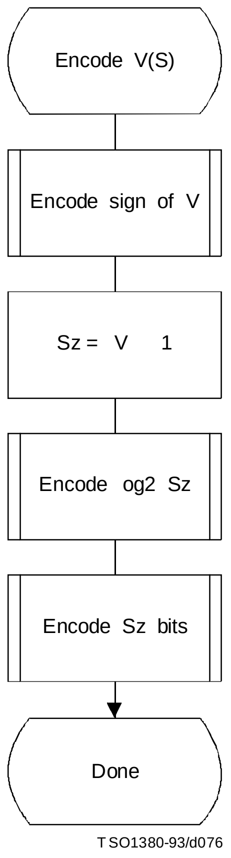


Figure F.6 – Sequence of procedures in encoding non-zero values of V

F.1.4.3.1.1 Encoding the sign

The sign is encoded by coding a 0-decision when the sign is positive and a 1-decision when the sign is negative (see Figure F.7).

The context-indices SS, SN and SP are defined for DC coding in Table F.4 and for AC coding in Table F.5. After the sign is coded, the context-index S is set to either SN or SP, establishing an initial value for Encode_log2_Sz.

F.1.4.3.1.2 Encoding the magnitude category

The magnitude category is determined by a sequence of binary decisions which compares Sz against an exponentially increasing bound (which is a power of 2) in order to determine the position of the leading 1-bit. This establishes the magnitude category in much the same way that the Huffman encoder generates a code for the value associated with the difference category. The flow chart for this procedure is shown in Figure F.8.

The starting value of the context-index S is determined in Encode_sign_of_V, and the context-index values X1 and X2 are defined for DC coding in Table F.4 and for AC coding in Table F.5. In Figure F.8, M is the exclusive upper bound for the magnitude and the abbreviations "SLL" and "SRL" refer to the shift-left-logical and shift-right-logical operations – in this case by one bit position. The SRL operation at the completion of the procedure aligns M with the most significant bit of Sz (see Table F.3).

The highest precision allowed for the DCT is 15 bits. Therefore, the highest precision required for the coding decision tree is 16 bits for the DC coefficient difference and 15 bits for the AC coefficients, including the sign bit.

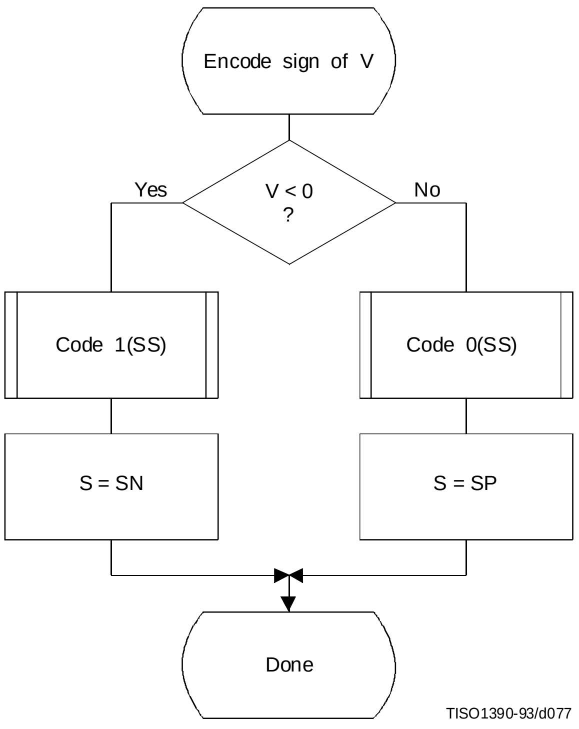


Figure F.7 – Encoding the sign of V

Table F.3 – Categories for each maximum bound

Exclusive upper bound (M)	Sz range	Number of low order magnitude bits
1	0	0
2	1	0
4	2,3	1
8	4,...,7	2
16	8,...,15	3
32	16,...,31	4
64	32,...,63	5
128	64,...,127	6
256	128,...,255	7
512	256,...,511	8
1 024	512,...,1 023	9
2 048	1 024,...,2 047	10
4 096	2 048,...,4 095	11
8 192	4 096,...,8 191	12
16 384	8 192,...,16 383	13
32 768	16 384,...,32 767	14

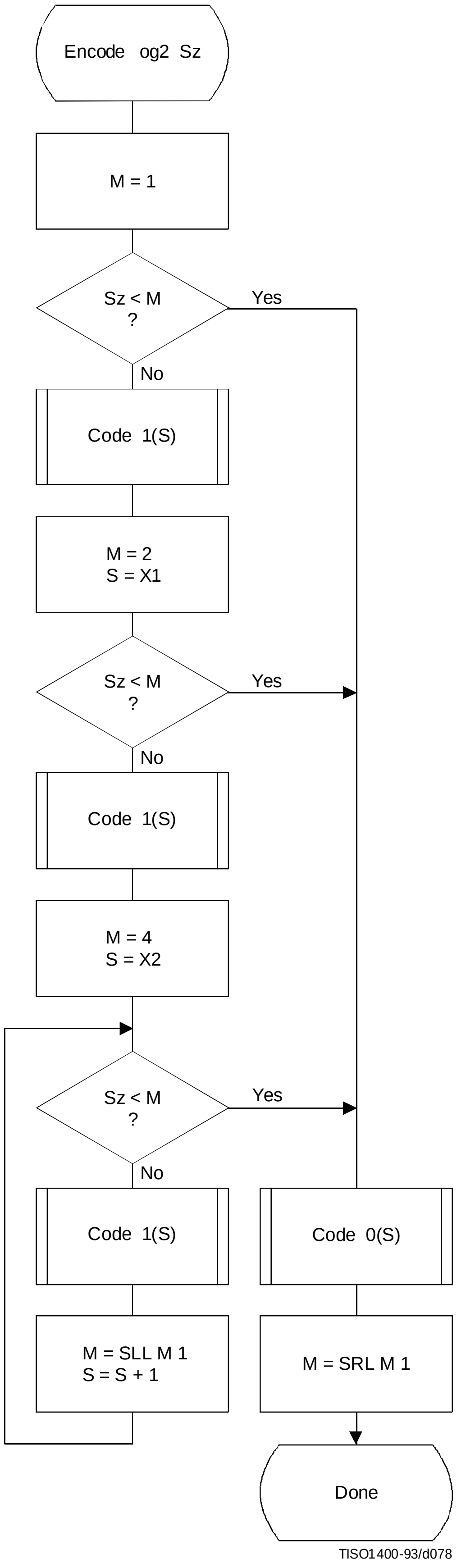


Figure F.8 – Decision sequence to establish the magnitude category

F.1.4.3.1.3 Encoding the exact value of the magnitude

After the magnitude category is encoded, the low order magnitude bits are encoded. These bits are encoded in order of decreasing bit significance. The procedure is shown in Figure F.9. The abbreviation "SRL" indicates the shift-right-logical operation, and M is the exclusive bound established in Figure F.8. Note that M has only one bit set – shifting M right converts it into a bit mask for the logical "AND" operation.

The starting value of the context-index S is determined in Encode_log2_Sz. The increment of S by 14 at the beginning of this procedure sets the context-index to the value required in Tables F.4 and F.5.

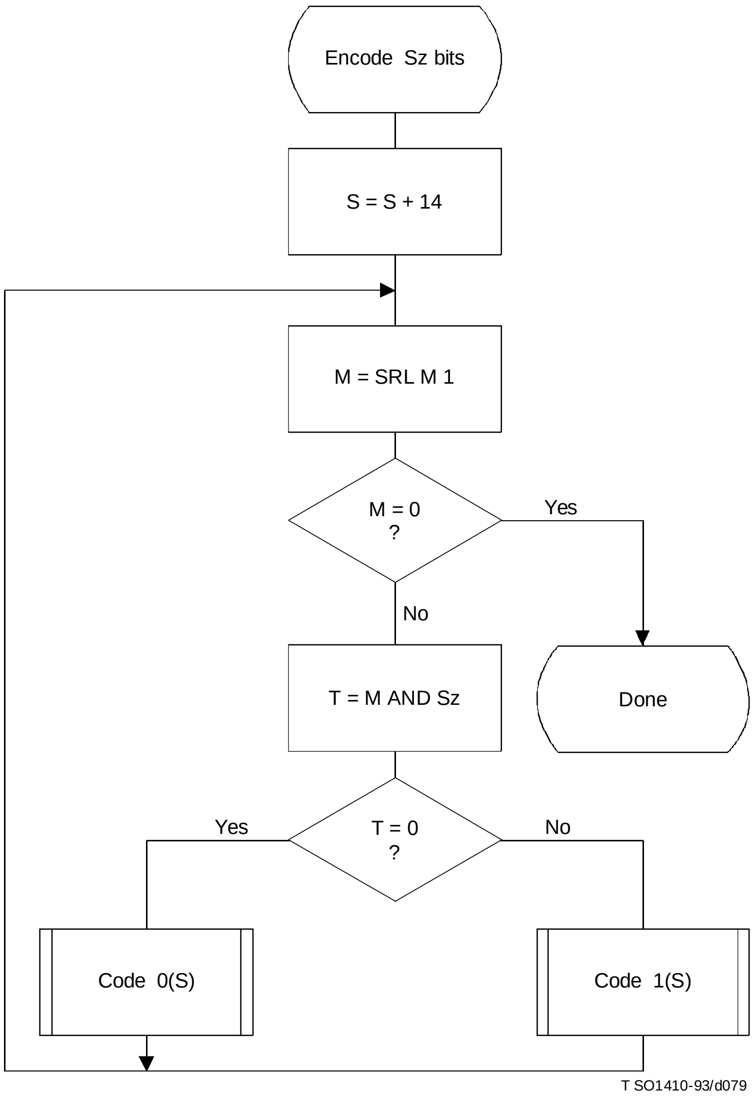


Figure F.9 – Decision sequence to code the magnitude bit pattern

F.1.4.4 Statistical models

An adaptive binary arithmetic coder requires a statistical model. The statistical model defines the contexts which are used to select the conditional probability estimates used in the encoding and decoding procedures.

Each decision in the binary decision trees is associated with one or more contexts. These contexts identify the sense of the MPS and the index in Table D.3 of the conditional probability estimate Qe which is used to encode and decode the binary decision.

The arithmetic coder is adaptive, which means that the probability estimates for each context are developed and maintained by the arithmetic coding system on the basis of prior coding decisions for that context.

F.1.4.4.1 Statistical model for coding DC prediction differences

The statistical model for coding the DC difference conditions some of the probability estimates for the binary decisions on previous DC coding decisions.

F.1.4.4.1.1 Statistical conditioning on sign

In coding the DC coefficients, four separate statistics bins (probability estimates) are used in coding the zero/not-zero ($V = 0$) decision, the sign decision and the first magnitude category decision. Two of these bins are used to code the $V = 0$ decision and the sign decision. The other two bins are used in coding the first magnitude decision, $Sz < 1$; one of these bins is used when the sign is positive, and the other is used when the sign is negative. Thus, the first magnitude decision probability estimate is conditioned on the sign of V.

F.1.4.4.1.2 Statistical conditioning on DC difference in previous block

The probability estimates for these first three decisions are also conditioned on Da, the difference value coded for the previous DCT block of the same component. The differences are classified into five groups: zero, small positive, small negative, large positive and large negative. The relationship between the default classification and the quantization scale is shown in Figure F.10.

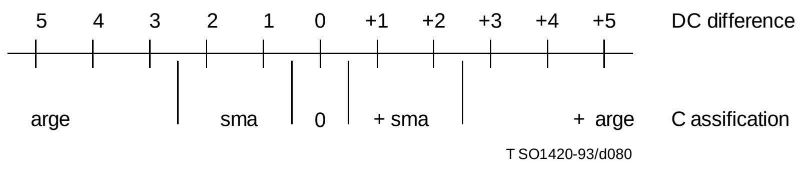


Figure F.10 – Conditioning classification of difference values

The bounds for the "small" difference category determine the classification. Defining L and U as integers in the range 0 to 15 inclusive, the lower bound (exclusive) for difference magnitudes classified as "small" is zero for $L = 0$, and is 2^{L-1} for $L > 0$.

The upper bound (inclusive) for difference magnitudes classified as "small" is 2^{U}.

L shall be less than or equal to U.

These bounds for the conditioning category provide a segmentation which is identical to that listed in Table F.3.

F.1.4.4.1.3 Assignment of statistical bins to the DC binary decision tree

As shown in Table F.4, each statistics area for DC coding consists of a set of 49 statistics bins. In the following explanation, it is assumed that the bins are contiguous. The first 20 bins consist of five sets of four bins selected by a context-index S0. The value of S0 is given by DC_Context(Da), which provides a value of 0, 4, 8, 12 or 16, depending on the difference classification of Da (see F.1.4.4.1.2). The remaining 29 bins, X1,...,X15,M2,...,M15, are used to code magnitude category decisions and magnitude bits.

Table F.4 – Statistical model for DC coefficient coding

Context-index	Value	Coding decision
S0	DC_Context(Da)	V = 0
SS	S0 + 1	Sign of V
SP	S0 + 2	Sz < 1 if V > 0
SN	S0 + 3	Sz < 1 if V < 0
X1	20	Sz < 2
X2	X1 + 1	Sz < 4
X3	X1 + 2	Sz < 8
.	.	.
.	.	.
X15	X1 + 14	$Sz < 2^{15}$
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if $Sz < 2^{15}$

F.1.4.4.1.4 Default conditioning for DC statistical model

The bounds, L and U, for determining the conditioning category have the default values L = 0 and U = 1. Other bounds may be set using the DAC (Define Arithmetic coding Conditioning) marker segment, as described in Annex B.

F.1.4.4.1.5 Initial conditions for DC statistical model

At the start of a scan and at the beginning of each restart interval, the difference for the previous DC value is defined to be zero in determining the conditioning state.

F.1.4.4.2 Statistical model for coding the AC coefficients

As shown in Table F.5, each statistics area for AC coding consists of a contiguous set of 245 statistics bins. Three bins are used for each value of the zig-zag index K, and two sets of 28 additional bins X2,...,X15,M2,...,M15 are used for coding the magnitude category and magnitude bits.

The value of SE (and also S0, SP and SN) is determined by the zig-zag index K. Since K is in the range 1 to 63, the lowest value for SE is 0 and the largest value for SP is 188. SS is not assigned a value in AC coefficient coding, as the signs of the coefficients are coded with a fixed probability value of approximately 0.5 (Qe = X’5A1D’, MPS = 0).

The value of X2 is given by AC_Context(K). This gives X2 = 189 when $K \le Kx$ and X2 = 217 when K > Kx, where Kx is defined using the DAC marker segment (see B.2.4.3).

Note that a X1 statistics bin is not used in this sequence. Instead, the 63×1 array of statistics bins for the magnitude category is used for two decisions. Once the magnitude bound has been determined – at statistics bin Xn, for example – a single statistics bin, Mn, is used to code the magnitude bit sequence for that bound.

F.1.4.4.2.1 Default conditioning for AC coefficient coding

The default value of Kx is 5. This may be modified using the DAC marker segment, as described in Annex B.

F.1.4.4.2.2 Initial conditions for AC statistical model

At the start of a scan and at each restart, all statistics bins are re-initialized to the standard default value described in Annex D.

Table F.5 – Statistical model for AC coefficient coding

Context-index	Value	Coding decision
SE	$3 \times (K - 1)$	K = EOB
S0	SE + 1	V = 0
SS	Fixed estimate	Sign of V
SN,SP	S0 + 1	Sz < 1
X1	S0 + 1	Sz < 2
X2	AC_Context(K)	Sz < 4
X3	X2 + 1	Sz < 8
.	.	.
.	.	.
X15	X2 + 13	$Sz < 2^{15}$
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if $Sz < 2^{15}$

F.1.5 Extended sequential DCT-based Huffman encoding process for 12-bit sample precision

This process is identical to the sequential DCT process for 8-bit precision extended to four Huffman table destinations as documented in F.1.3, with the following changes.

F.1.5.1 Structure of DC code table for 12-bit sample precision

The two's complement difference magnitudes are grouped into 16 categories, SSSS, and a Huffman code is created for each of the 16 difference magnitude categories.

The Huffman table for DC coding (see Table F.1) is extended as shown in Table F.6.

Table F.6 – Difference magnitude categories for DC coding

SSSS	Difference values
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383
15	–32 767..–16 384,16 384..32 767

F.1.5.2 Structure of AC code table for 12-bit sample precision

The general structure of the code table is extended as illustrated in Figure F.11. The Huffman table for AC coding is extended as shown in Table F.7.

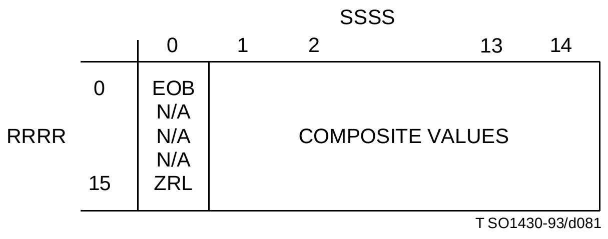


Figure F.11 – Two-dimensional value array for Huffman coding

Table F.7 – Values assigned to coefficient amplitude ranges

SSSS	AC coefficients
11	–2 047..–1 024,1 024..2 047
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383

F.1.6 Extended sequential DCT-based arithmetic encoding process for 12-bit sample precision

The process is identical to the sequential DCT process for 8-bit precision except for changes in the precision of the FDCT computation.

The structure of the encoding procedure is identical to that specified in F.1.4 which was already defined for a 12-bit sample precision.

F.2 Sequential DCT-based decoding processes

F.2.1 Sequential DCT-based control procedures and coding models

F.2.1.1 Control procedures for sequential DCT-based decoders

The control procedures for decoding compressed image data and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.6 to E.10. The procedure for decoding a MCU (Figure E.10) repetitively calls the procedure for decoding a data unit. For DCT-based decoders the data unit is an 8 × 8 block of samples.

F.2.1.2 Procedure for decoding an 8 × 8 block data unit

In the sequential DCT-based decoding process, decoding an 8 × 8 block data unit consists of the following procedures:

a) decode DC coefficient for 8 × 8 block using the DC table destination specified in the scan header;

b) decode AC coefficients for 8 × 8 block using the AC table destination specified in the scan header;

c) dequantize using table destination specified in the frame header and calculate the inverse 8 × 8 DCT.

F.2.1.3 Decoding models for the sequential DCT procedures

Two decoding procedures are used, one for the DC coefficient ZZ(0) and the other for the AC coefficients ZZ(1)...ZZ(63). The coefficients are decoded in the order in which they occur in the zig-zag sequence order, starting with the DC coefficient. The coefficients are represented as two's complement integers.

F.2.1.3.1 Decoding model for DC coefficients

The decoded difference, DIFF, is added to PRED, the DC value from the most recently decoded 8 × 8 block from the same component. Thus ZZ(0) = PRED + DIFF.

At the beginning of the scan and at the beginning of each restart interval, the prediction for the DC coefficient is initialized to zero.

F.2.1.3.2 Decoding model for AC coefficients

The AC coefficients are decoded in the order in which they occur in ZZ. When the EOB is decoded, all remaining coefficients in ZZ are initialized to zero.

F.2.1.4 Dequantization of the quantized DCT coefficients

The dequantization of the quantized DCT coefficients as described in Annex A, is accomplished by multiplying each quantized coefficient value by the quantization table value for that coefficient. The decoder shall be able to use up to four quantization table destinations.

F.2.1.5 Inverse DCT (IDCT)

The mathematical definition of the IDCT is given in A.3.3.

After computation of the IDCT, the signed output samples are level-shifted, as described in Annex A, converting the output to an unsigned representation. For 8-bit precision the level shift is performed by adding 128. For 12-bit precision the level shift is performed by adding 2 048. If necessary, the output samples shall be clamped to stay within the range appropriate for the precision (0 to 255 for 8-bit precision and 0 to 4 095 for 12-bit precision).

F.2.2 Baseline Huffman Decoding procedures

The baseline decoding procedure is for 8-bit sample precision. The decoder shall be capable of using up to two DC and two AC Huffman tables within one scan.

F.2.2.1 Huffman decoding of DC coefficients

The decoding procedure for the DC difference, DIFF, is:

$$T = \text{DECODE}$$

$$\text{DIFF} = \text{RECEIVE}(T)$$

$$\text{DIFF} = \text{EXTEND}(\text{DIFF},T)$$

where DECODE is a procedure which returns the 8-bit value associated with the next Huffman code in the compressed image data (see F.2.2.3) and RECEIVE(T) is a procedure which places the next T bits of the serial bit string into the low order bits of DIFF, MSB first. If T is zero, DIFF is set to zero. EXTEND is a procedure which converts the partially decoded DIFF value of precision T to the full precision difference. EXTEND is shown in Figure F.12.

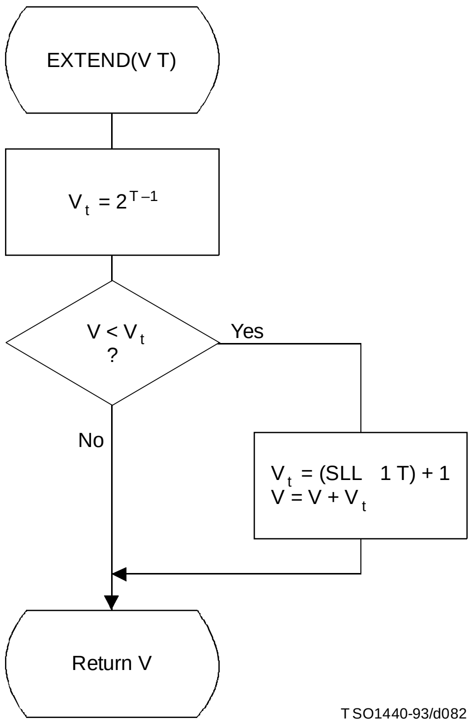


Figure F.12 – Extending the sign bit of a decoded value in V

F.2.2.2 Decoding procedure for AC coefficients

The decoding procedure for AC coefficients is shown in Figures F.13 and F.14.

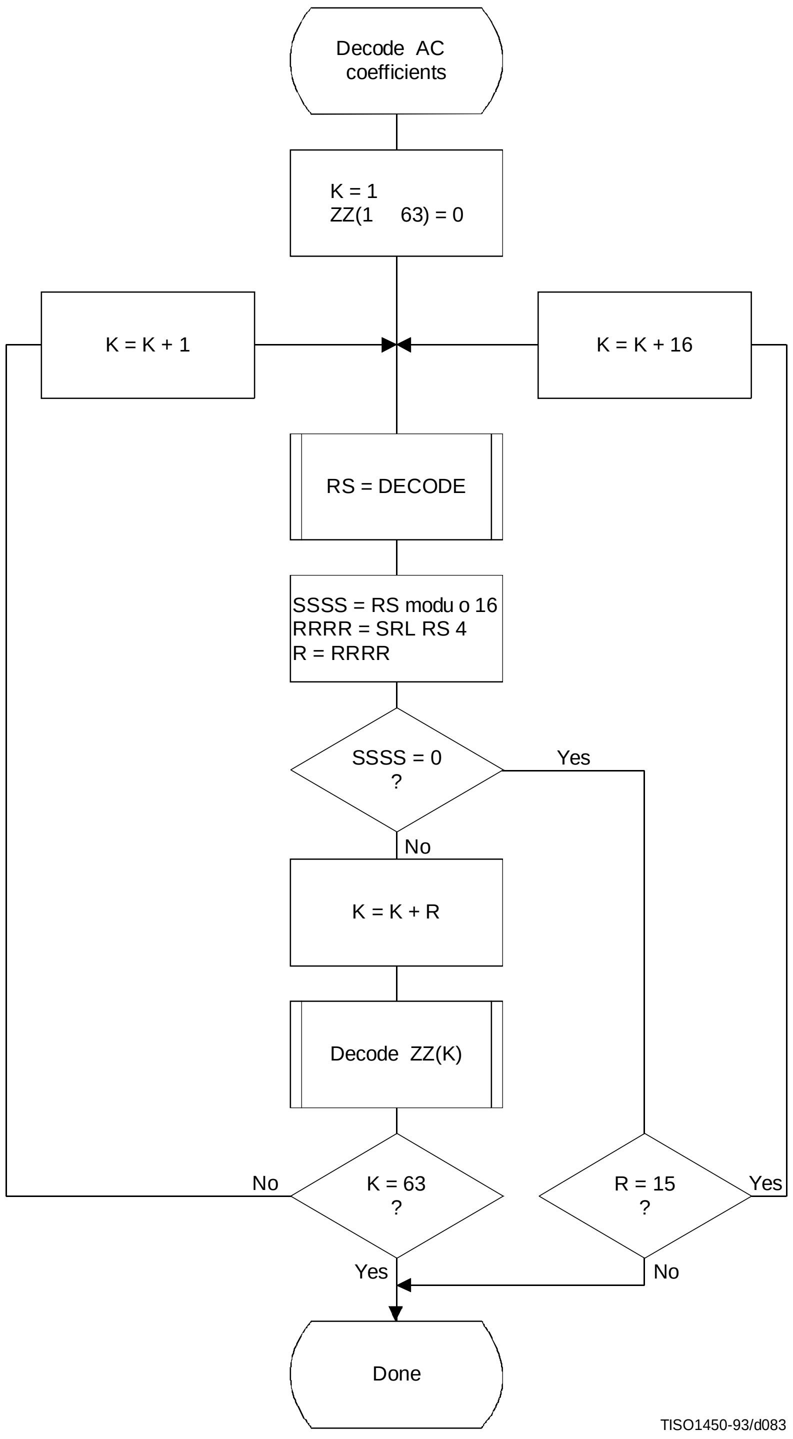


Figure F.13 – Huffman decoding procedure for AC coefficients

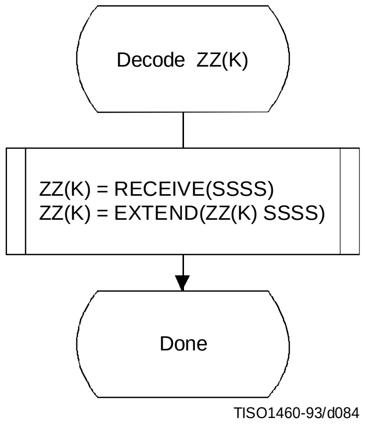


Figure F.14 – Decoding a non-zero AC coefficient

The decoding of the amplitude and sign of the non-zero coefficient is done in the procedure "Decode_ZZ(K)", shown in Figure F.14.

DECODE is a procedure which returns the value, RS, associated with the next Huffman code in the code stream (see F.2.2.3). The values SSSS and R are derived from RS. The value of SSSS is the four low order bits of the composite value and R contains the value of RRRR (the four high order bits of the composite value). The interpretation of these values is described in F.1.2.2. EXTEND is shown in Figure F.12.

F.2.2.3 The DECODE procedure

The DECODE procedure decodes an 8-bit value which, for the DC coefficient, determines the difference magnitude category. For the AC coefficient this 8-bit value determines the zero run length and non-zero coefficient category.

Three tables, HUFFVAL, HUFFCODE, and HUFFSIZE, have been defined in Annex C. This particular implementation of DECODE makes use of the ordering of the Huffman codes in HUFFCODE according to both value and code size. Many other implementations of DECODE are possible.

NOTE – The values in HUFFVAL are assigned to each code in HUFFCODE and HUFFSIZE in sequence. There are no ordering requirements for the values in HUFFVAL which have assigned codes of the same length.

The implementation of DECODE described in this subclause uses three tables, MINCODE, MAXCODE and VALPTR, to decode a pointer to the HUFFVAL table. MINCODE, MAXCODE and VALPTR each have 16 entries, one for each possible code size. MINCODE(I) contains the smallest code value for a given length I, MAXCODE(I) contains the largest code value for a given length I, and VALPTR(I) contains the index to the start of the list of values in HUFFVAL which are decoded by code words of length I. The values in MINCODE and MAXCODE are signed 16-bit integers; therefore, a value of –1 sets all of the bits.

The procedure for generating these tables is shown in Figure F.15. The procedure for DECODE is shown in Figure F.16. Note that the 8-bit "VALUE" is returned to the procedure which invokes DECODE.

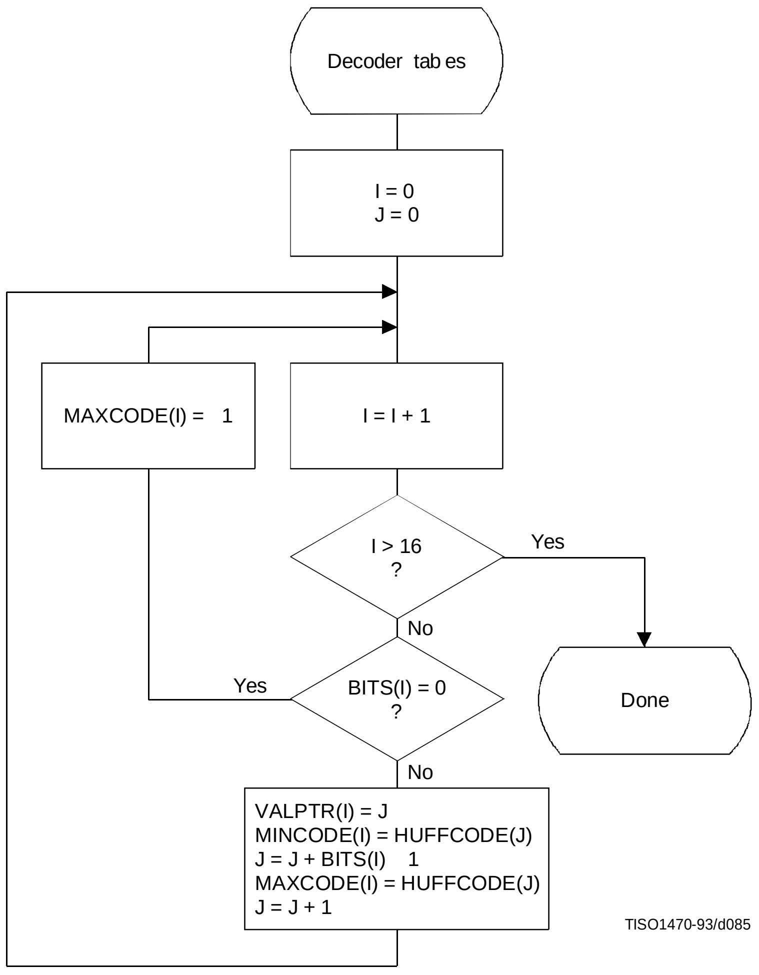


Figure F.15 – Decoder table generation

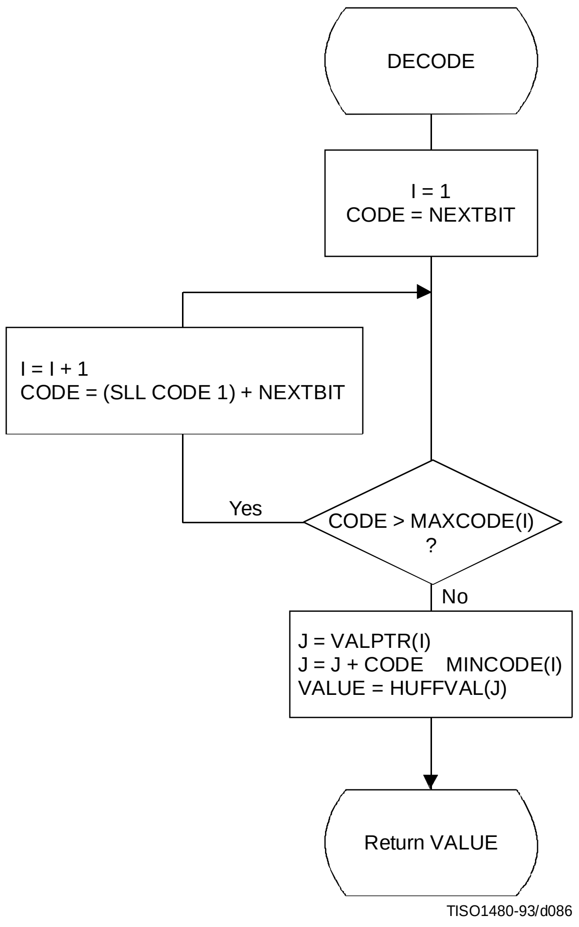


Figure F.16 – Procedure for DECODE

F.2.2.4 The RECEIVE procedure

RECEIVE(SSSS) is a procedure which places the next SSSS bits of the entropy-coded segment into the low order bits of DIFF, MSB first. It calls NEXTBIT and it returns the value of DIFF to the calling procedure (see Figure F.17).

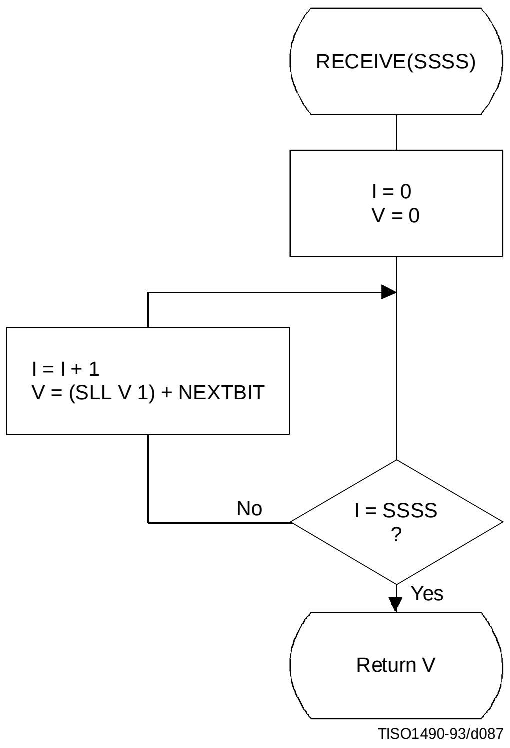


Figure F 17 Procedure for RECEIVE(SSSS)

F.2.2.5 The NEXTBIT procedure

NEXTBIT reads the next bit of compressed data and passes it to higher level routines. It also intercepts and removes stuff bytes and detects markers. NEXTBIT reads the bits of a byte starting with the MSB (see Figure F.18).

Before starting the decoding of a scan, and after processing a RST marker, CNT is cleared. The compressed data are read one byte at a time, using the procedure NEXTBYTE. Each time a byte, B, is read, CNT is set to 8.

The only valid marker which may occur within the Huffman coded data is the RST_m marker. Other than the EOI or markers which may occur at or before the start of a scan, the only marker which can occur at the end of the scan is the DNL (define-number-of-lines).

Normally, the decoder will terminate the decoding at the end of the final restart interval before the terminating marker is intercepted. If the DNL marker is encountered, the current line count is set to the value specified by that marker. Since the DNL marker can only be used at the end of the first scan, the scan decode procedure must be terminated when it is encountered.

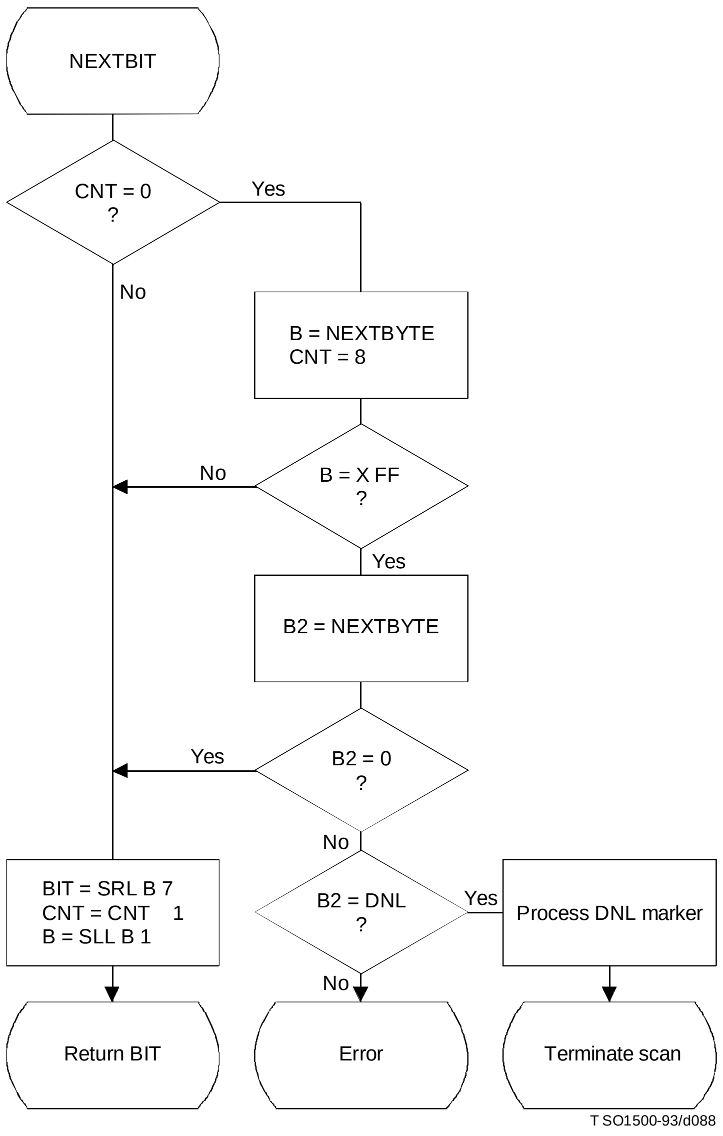


Figure F.18 – Procedure for fetching the next bit of compressed data

F.2.3 Sequential DCT decoding process with 8-bit precision extended to four sets of Huffman tables

This process is identical to the Baseline decoding process described in F.2.2, with the exception that the decoder shall be capable of using up to four DC and four AC Huffman tables within one scan. Four DC and four AC Huffman tables is the maximum allowed by this Specification.

F.2.4 Sequential DCT decoding process with arithmetic coding

This subclause describes the sequential DCT decoding process with arithmetic decoding.

The arithmetic decoding procedures for decoding binary decisions, initializing the statistical model, initializing the decoder, and resynchronizing the decoder are listed in Table D.4 of Annex D.

Some of the procedures in Table D.4 are used in the higher level control structure for scans and restart intervals described in F.2. At the beginning of scans and restart intervals, the probability estimates used in the arithmetic decoder are reset to the standard initial value as part of the Initdec procedure which restarts the arithmetic coder.

The statistical models defined in F.1.4.4 also apply to this decoding process.

The decoder shall be capable of using up to four DC and four AC conditioning tables and associated statistics areas within one scan.

F.2.4.1 Arithmetic decoding of DC coefficients

The basic structure of the decision sequence for decoding a DC difference value, DIFF, is shown in Figure F.19. The equivalent structure for the encoder is found in Figure F.4.

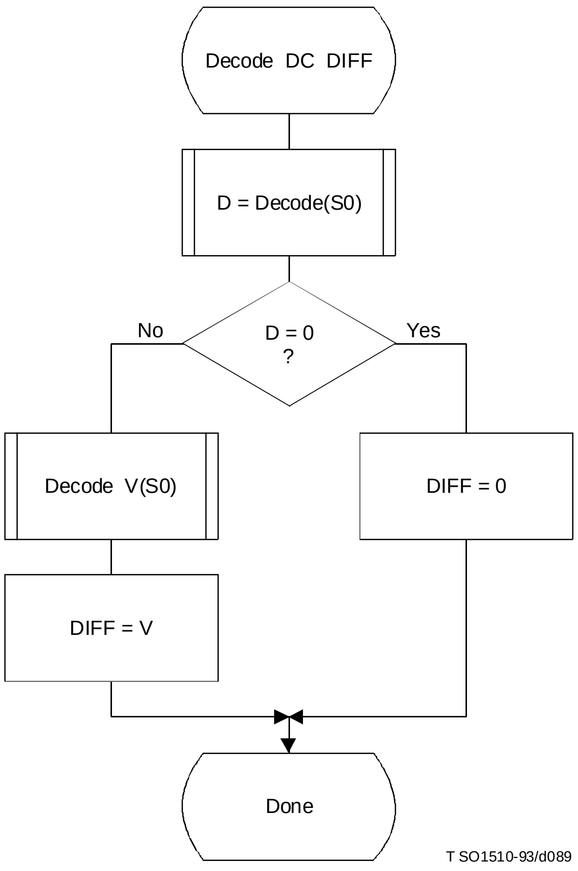


Figure F.19 – Arithmetic decoding of DC difference

The context-indices used in the DC decoding procedures are defined in Table F.4 (see F.1.4.4.1.3).

The “Decode” procedure returns the value “D” of the binary decision. If the value is not zero, the sign and magnitude of the non-zero DIFF must be decoded by the procedure “Decode_V(S0)”.

F.2.4.2 Arithmetic Decoding of AC coefficients

The AC coefficients are decoded in the order that they occur in ZZ(1,...,63). The encoder procedure for the coding process is found in Figure F.5. Figure F.20 illustrates the decoding sequence.

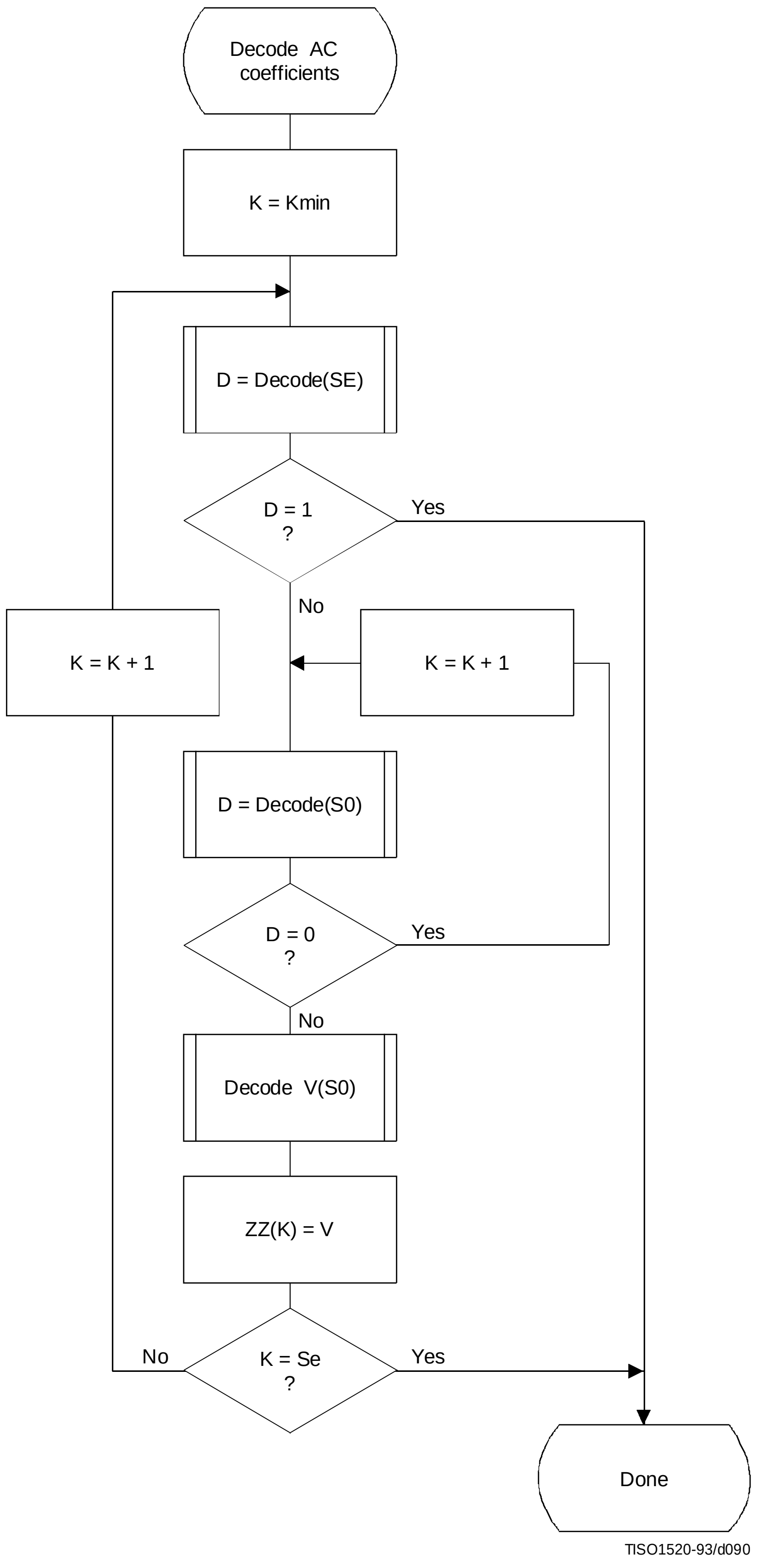


Figure F.20 – Procedure for decoding the AC coefficients

The context-indices used in the AC decoding procedures are defined in Table F.5 (see F.1.4.4.2).

In Figure F.20, K is the index to the zig-zag sequence position. For the sequential scan, Kmin = 1 and Se = 63. The decision at the top of the loop is the EOB decision. If the EOB occurs (D = 1), the remaining coefficients in the block are set to zero. The inner loop just below the EOB decoding decodes runs of zero coefficients. Whenever the coefficient is non-zero, "Decode_V" decodes the sign and magnitude of the coefficient. After each non-zero coefficient is decoded, the EOB decision is again decoded unless K = Se.

F.2.4.3 Decoding the binary decision sequence for non-zero DC differences and AC coefficients

Both the DC difference and the AC coefficients are represented as signed two's complement 16-bit integer values. The decoding decision tree for these signed integer values is the same for both the DC and AC coding models. Note, however, that the statistical models are not the same.

F.2.4.3.1 Arithmetic decoding of non-zero values

Denoting either DC differences or AC coefficients as V, the non-zero signed integer value of V is decoded by the sequence shown in Figure F.21. This sequence first decodes the sign of V. It then decodes the magnitude category of V (Decode_log2_Sz), and then decodes the low order magnitude bits (Decode_Sz_bits). Note that the value decoded for Sz must be incremented by 1 to get the actual coefficient magnitude.

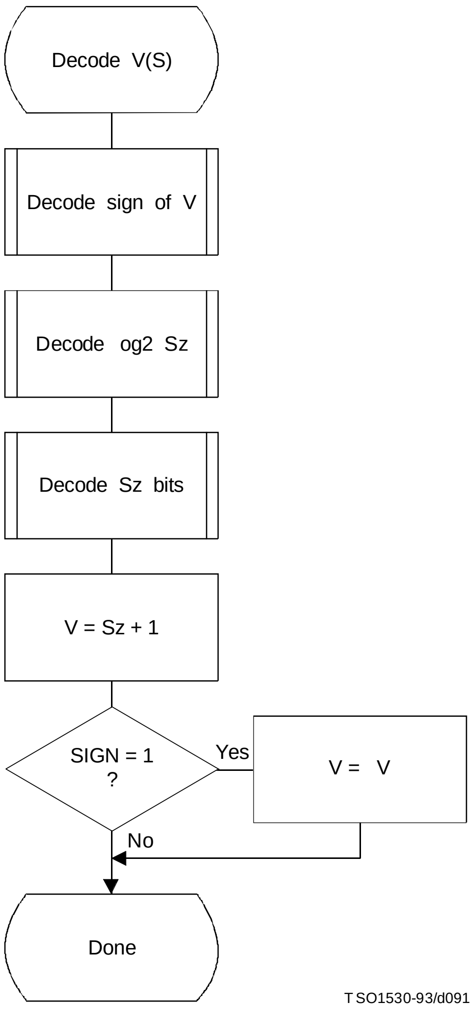


Figure F.21 – Sequence of procedures in decoding non-zero values of V

F.2.4.3.1.1 Decoding the sign

The sign is decoded by the procedure shown in Figure F.22.

The context-indices are defined for DC decoding in Table F.4 and AC decoding in Table F.5.

If SIGN = 0, the sign of the coefficient is positive; if SIGN = 1, the sign of the coefficient is negative.

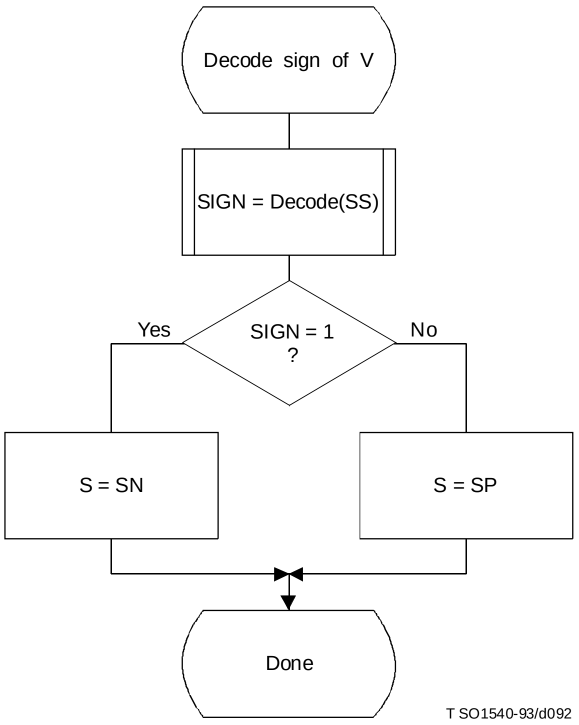


Figure F.22 – Decoding the sign of V

F.2.4.3.1.2 Decoding the magnitude category

The context-index S is set in Decode_sign_of_V and the context-index values X1 and X2 are defined for DC coding in Table F.4 and for AC coding in Table F.5.

In Figure F.23, M is set to the upper bound for the magnitude and shifted left until the decoded decision is zero. It is then shifted right by 1 to become the leading bit of the magnitude of Sz.

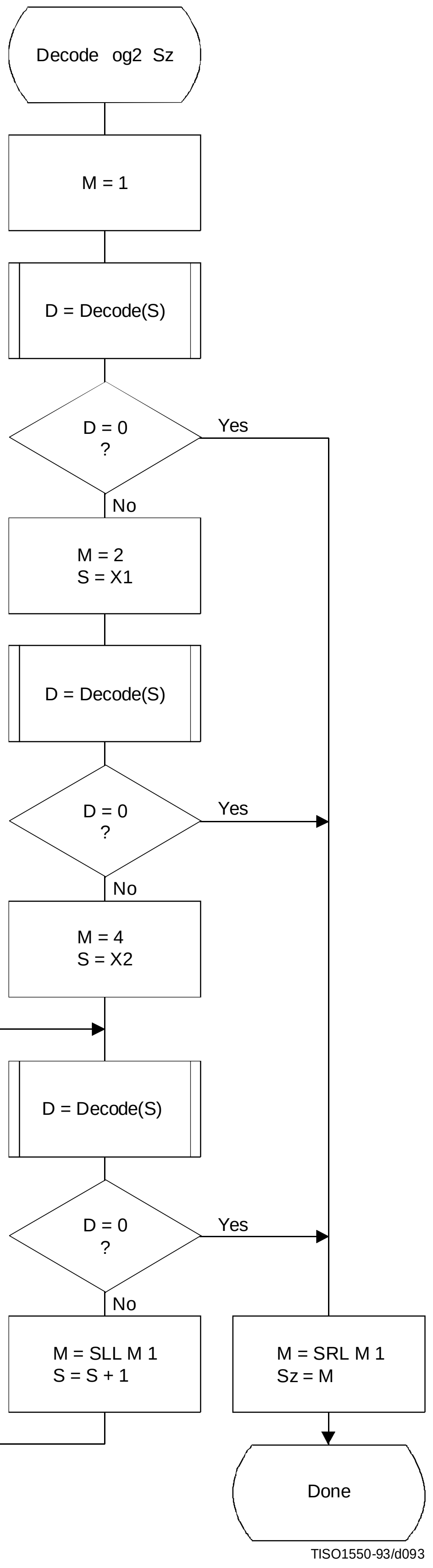


Figure F.23 – Decoding procedure to establish the magnitude category

F.2.4.3.1.3 Decoding the exact value of the magnitude

After the magnitude category is decoded, the low order magnitude bits are decoded. These bits are decoded in order of decreasing bit significance. The procedure is shown in Figure F.24.

The context-index S is set in Decode_log2_Sz.

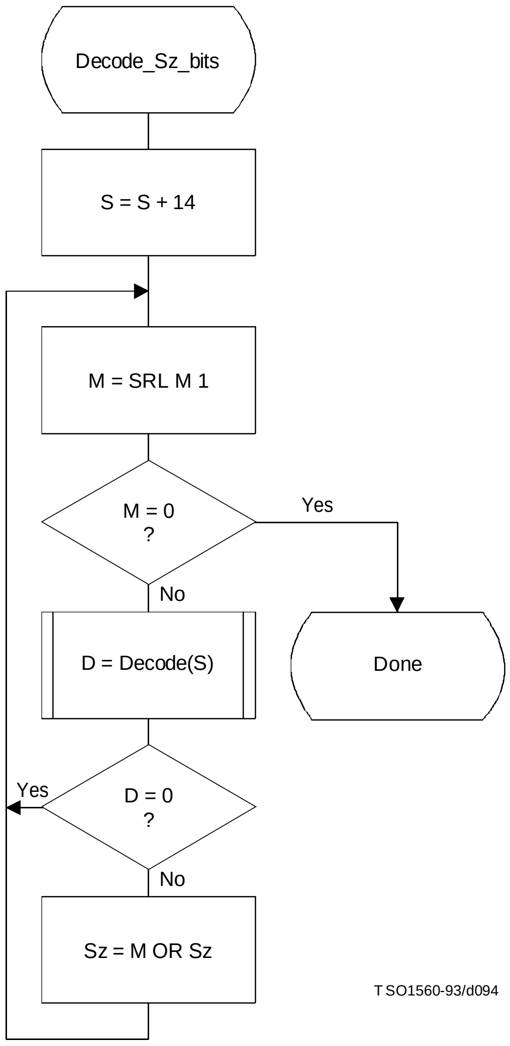


Figure F.24 – Decision sequence to decode the magnitude bit pattern

F.2.4.4 Decoder restart

The RST_m markers which are added to the compressed data between each restart interval have a two byte value which cannot be generated by the coding procedures. These two byte sequences can be located without decoding, and can therefore be used to resynchronize the decoder. RST_m markers can therefore be used for error recovery.

Before error recovery procedures can be invoked, the error condition must first be detected. Errors during decoding can show up in two places:

a) The decoder fails to find the expected marker at the point where it is expecting resynchronization.

b) Physically impossible data are decoded. For example, decoding a magnitude beyond the range of values allowed by the model is quite likely when the compressed data are corrupted by errors. For arithmetic decoders this error condition is extremely important to detect, as otherwise the decoder may reach a condition where it uses the compressed data very slowly.

   NOTE – Some errors will not cause the decoder to lose synchronization. In addition, recovery is not possible for all errors; for example, errors in the headers are likely to be catastrophic. The two error conditions listed above, however, almost always cause the decoder to lose synchronization in a way which permits recovery.

In regaining synchronization, the decoder can make use of the modulo 8 coding restart interval number in the low order bits of the RST_m marker. By comparing the expected restart interval number to the value in the next RST_m marker in the compressed image data, the decoder can usually recover synchronization. It then fills in missing lines in the output data by replication or some other suitable procedure, and continues decoding. Of course, the reconstructed image will usually be highly corrupted for at least a part of the restart interval where the error occurred.

F.2.5 Sequential DCT decoding process with Huffman coding and 12-bit precision

This process is identical to the sequential DCT process defined for 8-bit sample precision and extended to four Huffman tables, as documented in F.2.3, but with the following changes.

F.2.5.1 Structure of DC Huffman decode table

The general structure of the DC Huffman decode table is extended as described in F.1.5.1.

F.2.5.2 Structure of AC Huffman decode table

The general structure of the AC Huffman decode table is extended as described in F.1.5.2.

F.2.6 Sequential DCT decoding process with arithmetic coding and 12-bit precision

The process is identical to the sequential DCT process for 8-bit precision except for changes in the precision of the IDCT computation.

The structure of the decoding procedure in F.2.4 is already defined for a 12-bit input precision.

Annex G

Progressive DCT-based mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the progressive DCT-based mode of operation:

1) spectral selection only, Huffman coding, 8-bit sample precision;
2) spectral selection only, arithmetic coding, 8-bit sample precision;
3) full progression, Huffman coding, 8-bit sample precision;
4) full progression, arithmetic coding, 8-bit sample precision;
5) spectral selection only, Huffman coding, 12-bit sample precision;
6) spectral selection only, arithmetic coding, 12-bit sample precision;
7) full progression, Huffman coding, 12-bit sample precision;
8) full progression, arithmetic coding, 12-bit sample precision.

For each of these, the encoding process is specified in G.1, and the decoding process is specified in G.2. The functional specification is presented by means of specific flow charts for the various procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

The number of Huffman or arithmetic conditioning tables which may be used within the same scan is four.

Two complementary progressive procedures are defined, spectral selection and successive approximation.

In spectral selection the DCT coefficients of each block are segmented into frequency bands. The bands are coded in separate scans.

In successive approximation the DCT coefficients are divided by a power of two before coding. In the decoder the coefficients are multiplied by that same power of two before computing the IDCT. In the succeeding scans the precision of the coefficients is increased by one bit in each scan until full precision is reached.

An encoder or decoder implementing a full progression uses spectral selection within successive approximation. An allowed subset is spectral selection alone.

Figure G.1 illustrates the spectral selection and successive approximation progressive processes.

G.1 Progressive DCT-based encoding processes

G.1.1 Control procedures and coding models for progressive DCT-based procedures

G.1.1.1 Control procedures for progressive DCT-based encoders

The control procedures for encoding an image and its constituent parts – the frame, scan, restart interval and MCU – are given in Figures E.1 through E.5.

The control structure for encoding a frame is the same as for the sequential procedures. However, it is convenient to calculate the FDCT for the entire set of components in a frame before starting the scans. A buffer which is large enough to store all of the DCT coefficients may be used for this progressive mode of operation.

The number of scans is determined by the progression defined; the number of scans may be much larger than the number of components in the frame.

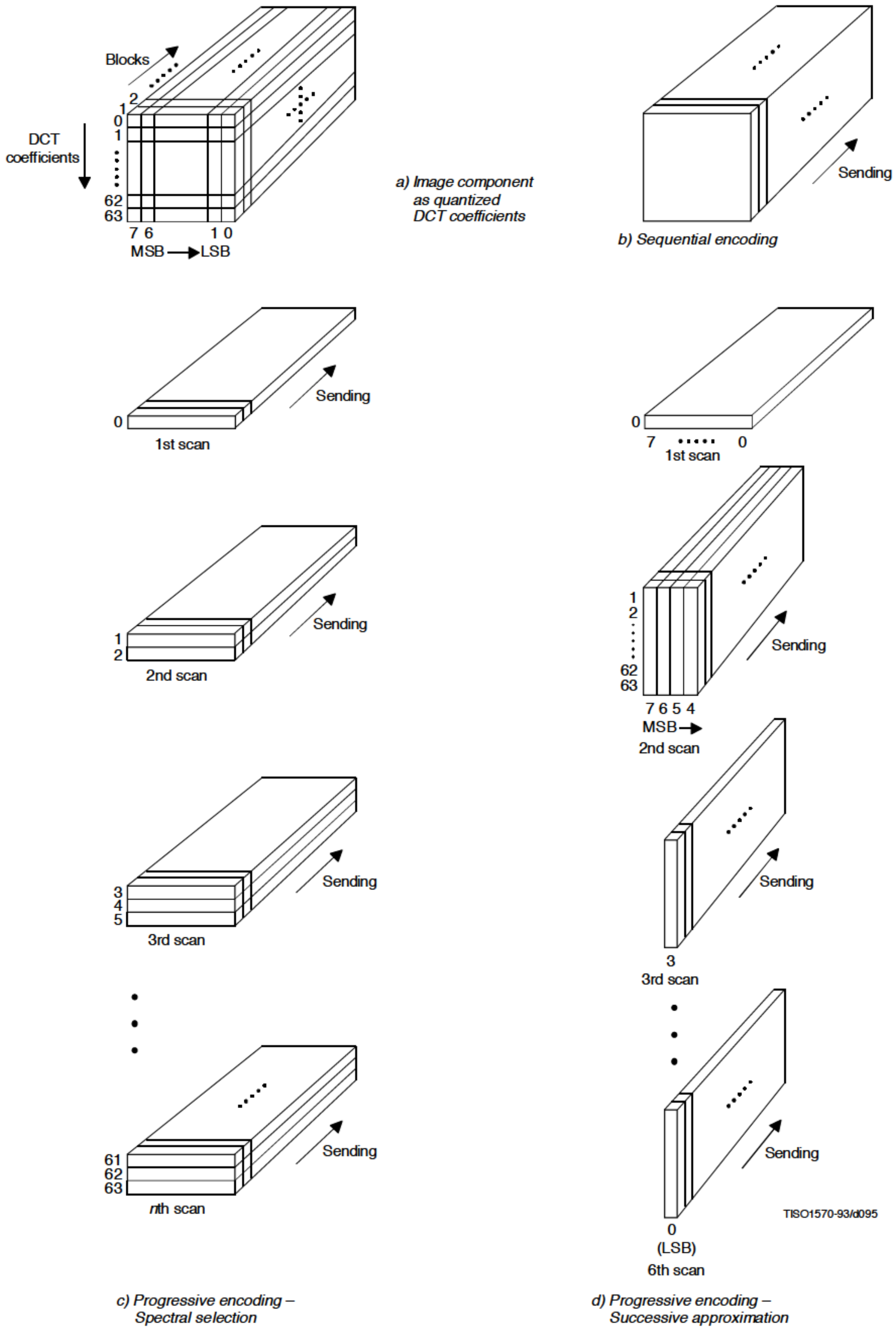


Figure G.1 – Spectral selection and successive approximation progressive processes

The procedure for encoding a MCU (see Figure E.5) repetitively invokes the procedure for coding a data unit. For DCT-based encoders the data unit is an 8×8 block of samples.

Only a portion of each 8×8 block is coded in each scan, the portion being determined by the scan header parameters Ss, Se, Ah, and Al (see B.2.3). The procedures used to code portions of each 8×8 block are described in this annex. Note, however, that where these procedures are identical to those used in the sequential DCT-based mode of operation, the sequential procedures are simply referenced.

G.1.1.1.1 Spectral selection control

In spectral selection the zig-zag sequence of DCT coefficients is segmented into bands. A band is defined in the scan header by specifying the starting and ending indices in the zig-zag sequence. One band is coded in a given scan of the progression. DC coefficients are always coded separately from AC coefficients, and only scans which code DC coefficients may have interleaved blocks from more than one component. All other scans shall have only one component. With the exception of the first DC scans for the components, the sequence of bands defined in the scans need not follow the zig-zag ordering. For each component, a first DC scan shall precede any AC scans.

G.1.1.1.2 Successive approximation control

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform (see A.4) defined in the scan header (see B.2.3). The successive approximation bit position parameter Al specifies the actual point transform, and the high four bits (Ah) – if there are preceding scans for the band – contain the value of the point transform used in those preceding scans. If there are no preceding scans for the band, Ah is zero.

Each scan which follows the first scan for a given band progressively improves the precision of the coefficients by one bit, until full precision is reached.

G.1.1.2 Coding models for progressive DCT-based encoders

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform (see A.4) defined in the scan header (see B.2.3). These models also apply to the progressive DCT-based encoders, but with the following changes.

G.1.1.2.1 Progressive encoding model for DC coefficients

If Al is not zero, the point transform for DC coefficients shall be used to reduce the precision of the DC coefficients. If Ah is zero, the coefficient values (as modified by the point transform) shall be coded, using the procedure described in Annex F. If Ah is not zero, the least significant bit of the point transformed DC coefficients shall be coded, using the procedures described in this annex.

G.1.1.2.2 Progressive encoding model for AC coefficients

If Al is not zero, the point transform for AC coefficients shall be used to reduce the precision of the AC coefficients. If Ah is zero, the coefficient values (as modified by the point transform) shall be coded using modifications of the procedures described in Annex F. These modifications are described in this annex. If Ah is not zero, the precision of the coefficients shall be improved using the procedures described in this annex.

G.1.2 Progressive encoding procedures with Huffman coding

G.1.2.1 Progressive encoding of DC coefficients with Huffman coding

The first scan for a given component shall encode the DC coefficient values using the procedures described in F.1.2.1. If the successive approximation bit position parameter Al is not zero, the coefficient values shall be reduced in precision by the point transform described in Annex A before coding.

In subsequent scans using successive approximation the least significant bits are appended to the compressed bit stream without compression or modification (see G.1.2.3), except for byte stuffing.

G.1.2.2 Progressive encoding of AC coefficients with Huffman coding

In spectral selection and in the first scan of successive approximation for a component, the AC coefficient coding model is similar to that used by the sequential procedures. However, the Huffman code tables are extended to include coding of runs of End-Of-Bands (EOBs). See Table G.1.

Table G.1 – EOBn code run length extensions

EOBn code	Run length
EOB0	1
EOB1	2,3
EOB2	4..7
EOB3	8..15
EOB4	16..31
EOB5	32..63
EOB6	64..127
EOB7	128..255
EOB8	256..511
EOB9	512..1 023
EOB10	1 024..2 047
EOB11	2 048..4 095
EOB12	4 096..8 191
EOB13	8 192..16 383
EOB14	16 384..32 767

The end-of-band run structure allows efficient coding of blocks which have only zero coefficients. An EOB run of length 5 means that the current block and the next four blocks have an end-of-band with no intervening non-zero coefficients. The EOB run length is limited only by the restart interval.

The extension of the code table is illustrated in Figure G.2.

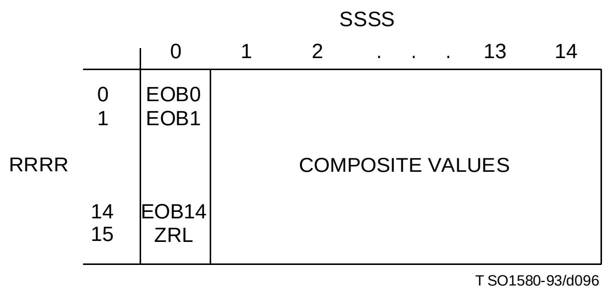


Figure G.2 – Two-dimensional value array for Huffman coding

The EOBn code sequence is defined as follows. Each EOBn code is followed by an extension field similar to the extension field for the coefficient amplitudes (but with positive numbers only). The number of bits appended to the EOBn code is the minimum number required to specify the run length.

If an EOB run is greater than 32 767, it is coded as a sequence of EOB runs of length 32 767 followed by a final EOB run sufficient to complete the run.

At the beginning of each restart interval the EOB run count, EOBRUN, is set to zero. At the end of each restart interval any remaining EOB run is coded.

The Huffman encoding procedure for AC coefficients in spectral selection and in the first scan of successive approximation is illustrated in Figures G.3, G.4, G.5, and G.6.

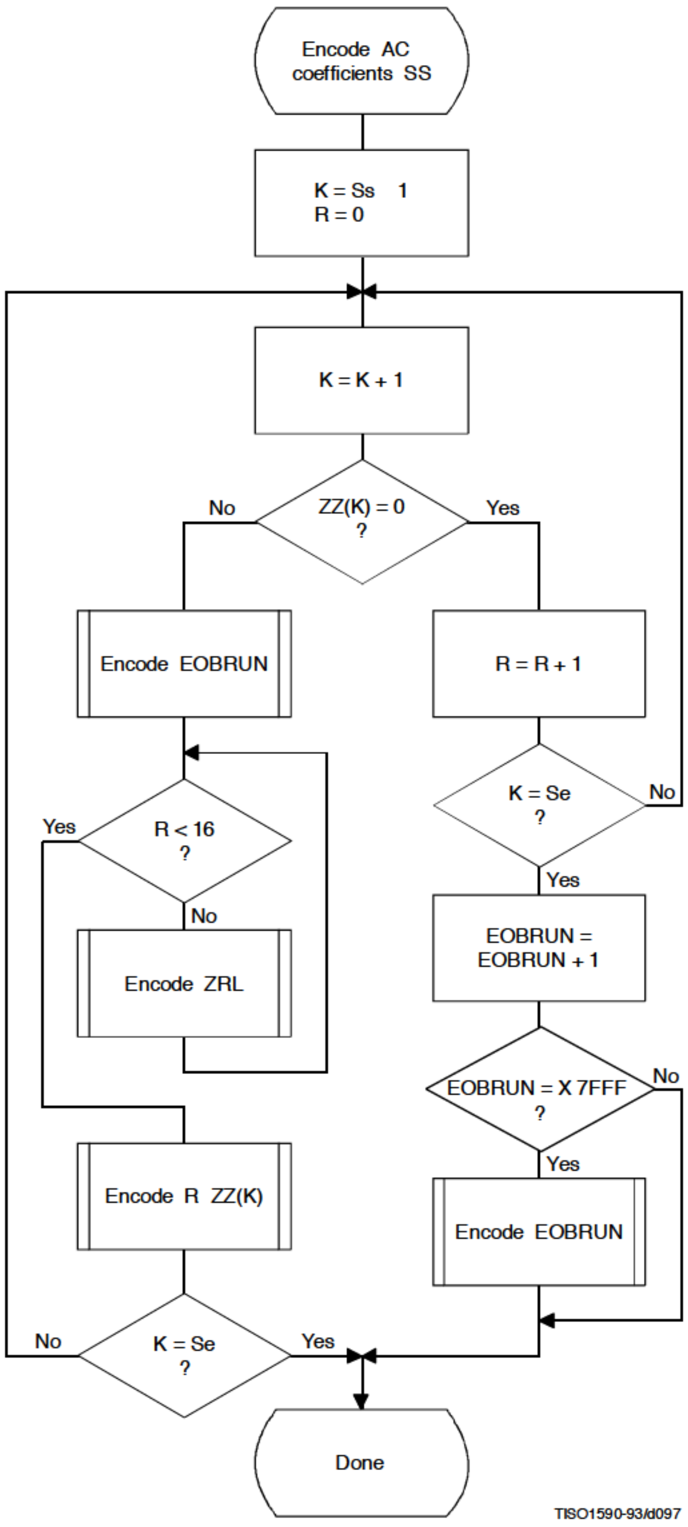


Figure G.3 – Procedure for progressive encoding of AC coefficients with Huffman coding

In Figure G.3, Ss is the start of spectral selection, Se is the end of spectral selection, K is the index into the list of coefficients stored in the zig-zag sequence ZZ, R is the run length of zero coefficients, and EOBRUN is the run length of EOBs. EOBRUN is set to zero at the start of each restart interval.

If the scan header parameter Al (successive approximation bit position low) is not zero, the DCT coefficient values ZZ(K) in Figure G.3 and figures which follow in this annex, including those in the arithmetic coding section, shall be replaced by the point transformed values ZZ'(K), where ZZ'(K) is defined by:

$$ZZ'(K) = \frac{ZZ(K)x}{2^{Al}}$$

EOBSIZE is a procedure which returns the size of the EOB extension field given the EOB run length as input. CSIZE is a procedure which maps an AC coefficient to the SSSS value defined in the subclauses on sequential encoding (see F.1.1 and F.1.3).

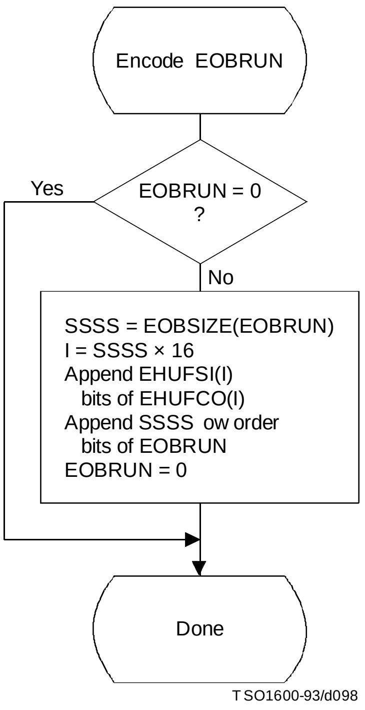


Figure G.4 – Progressive encoding of a non-zero AC coefficient

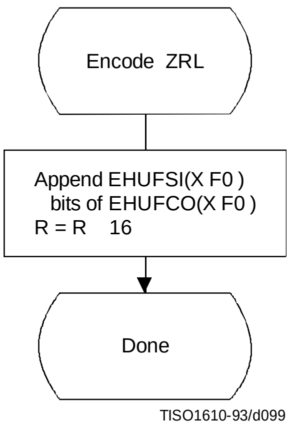


Figure G.5 – Encoding of the run of zero coefficients

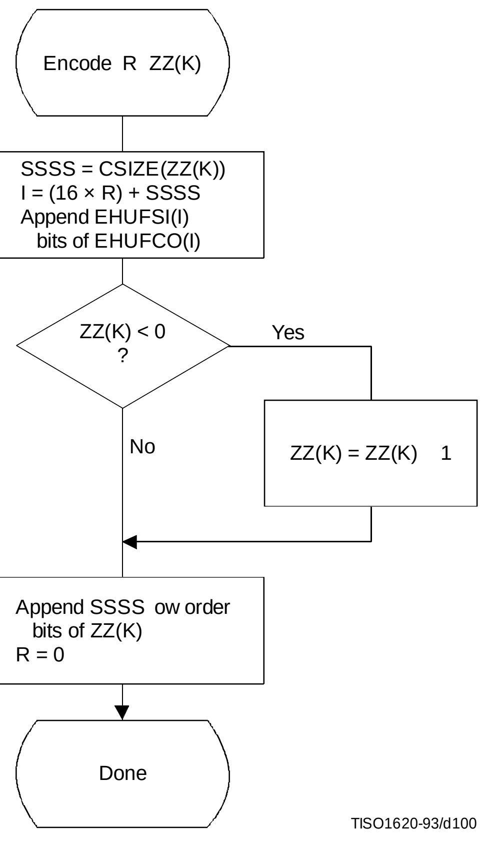


Figure G.6 – Encoding of the zero run and non-zero coefficient

G.1.2.3 Coding model for subsequent scans of successive approximation

The Huffman coding structure of the subsequent scans of successive approximation for a given component is similar to the coding structure of the first scan of that component.

The structure of the AC code table is identical to the structure described in G.1.2.2. Each non-zero point transformed coefficient that has a zero history (i.e. that has a value ± 1, and therefore has not been coded in a previous scan) is defined by a composite 8-bit run length-magnitude value of the form:

RRRRSSSS

The four most significant bits, RRRR, give the number of zero coefficients that are between the current coefficient and the previously coded coefficient (or the start of band). Coefficients with non-zero history (a non-zero value coded in a previous scan) are skipped over when counting the zero coefficients. The four least significant bits, SSSS, provide the magnitude category of the non-zero coefficient; for a given component the value of SSSS can only be one.

The run length-magnitude composite value is Huffman coded and each Huffman code is followed by additional bits:

a) One bit codes the sign of the newly non-zero coefficient. A 0-bit codes a negative sign; a 1-bit codes a positive sign.

b) For each coefficient with a non-zero history, one bit is used to code the correction. A 0-bit means no correction and a 1-bit means that one shall be added to the (scaled) decoded magnitude of the coefficient.

Non-zero coefficients with zero history are coded with a composite code of the form:

HUFFCO(RRRRSSSS) + additional bit (rule a) + correction bits (rule b)

In addition whenever zero runs are coded with ZRL or EOBn codes, correction bits for those coefficients with non-zero history contained within the zero run are appended according to rule b above.

For the Huffman coding version of Encode_AC_Coefficients_SA the EOB is defined to be the position of the last point transformed coefficient of magnitude 1 in the band. If there are no coefficients of magnitude 1, the EOB is defined to be zero.

NOTE – The definition of EOB is different for Huffman and arithmetic coding procedures.

In Figures G.7 and G.8 BE is the count of buffered correction bits at the start of coding of the block. BE is initialized to zero at the start of each restart interval. At the end of each restart interval any remaining buffered bits are appended to the bit stream following the last EOBn Huffman code and associated appended bits.

In Figures G.7 and G.9, BR is the count of buffered correction bits which are appended to the bit stream according to rule b. BR is set to zero at the beginning of each Encode_AC_Coefficients_SA. At the end of each restart interval any remaining buffered bits are appended to the bit stream following the last Huffman code and associated appended bits.

G.1.3 Progressive encoding procedures with arithmetic coding

G.1.3.1 Progressive encoding of DC coefficients with arithmetic coding

The first scan for a given component shall encode the DC coefficient values using the procedures described in F.1.4.1. If the successive approximation bit position parameter is not zero, the coefficient values shall be reduced in precision by the point transform described in Annex A before coding.

In subsequent scans using successive approximation the least significant bits shall be coded as binary decisions using a fixed probability estimate of 0.5 (Qe = X’5A1D’, MPS = 0).

G.1.3.2 Progressive encoding of AC coefficients with arithmetic coding

Except for the point transform scaling of the DCT coefficients and the grouping of the coefficients into bands, the first scan(s) of successive approximation is identical to the sequential encoding procedure described in F.1.4. If Kmin is equated to Ss, the index of the first AC coefficient index in the band, the flow chart shown in Figure F.5 applies. The EOB decision in that figure refers to the “end-of-band” rather than the “end-of-block”. For the arithmetic coding version of Encode_AC_Coefficients_SA (and all other AC coefficient coding procedures) the EOB is defined to be the position following the last non-zero coefficient in the band.

NOTE - The definition of EOB is different for Huffman and arithmetic coding procedures.

The statistical model described in F.1.4 also holds. For this model the default value of Kx is 5. Other values of Kx may be specified using the DAC marker code (Annex B). The following calculation for Kx has proven to give good results for 8-bit precision samples:

$$Kx = Kmin + SRL\ (8 + Se - Kmin)\ 4$$

This expression reduces to the default of Kx = 5 when the band is from index 1 to index 63.

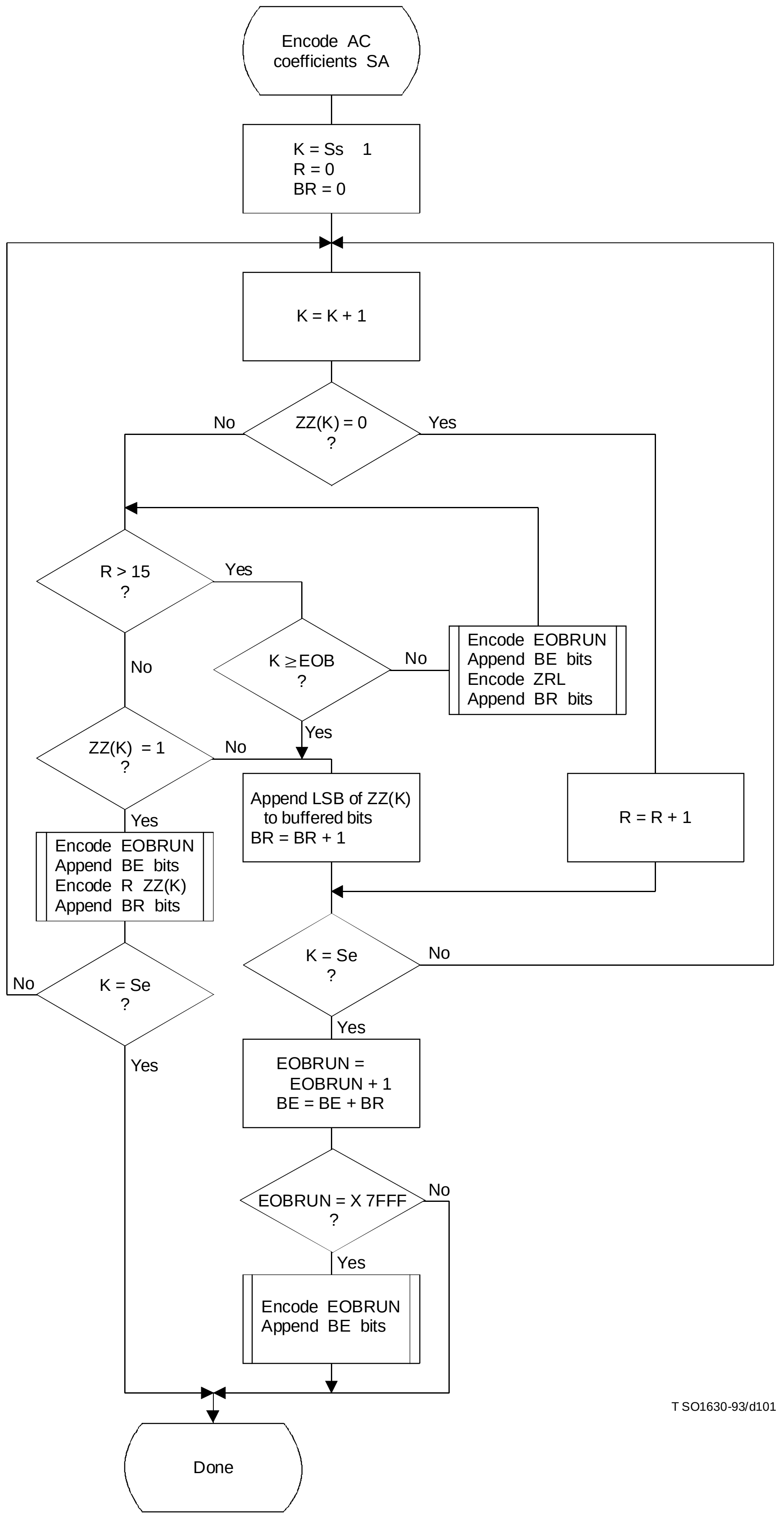


Figure G.7 – Successive approximation coding of AC coefficients using Huffman coding

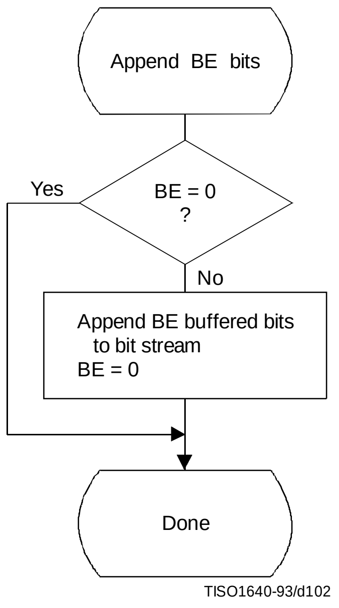


Figure G.8 – Transferring BE buffered bits from buffer to bit stream

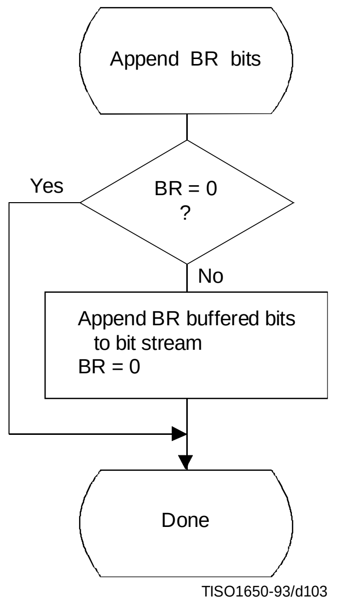


Figure G.9 – Transferring BR buffered bits from buffer to bit stream

G.1.3.3 Coding model for subsequent scans of successive approximation

The procedure “Encode_AC_Coefficient_SA” shown in Figure G.10 increases the precision of the AC coefficient values in the band by one bit.

As in the first scan of successive approximation for a component, an EOB decision is coded at the start of the band and after each non-zero coefficient.

However, since the end-of-band index of the previous successive approximation scan for a given component, EOBx, is known from the data coded in the prior scan of that component, this decision is bypassed whenever the current index, K, is less than EOBx. As in the first scan(s), the EOB decision is also bypassed whenever the last coefficient in the band is not zero. The decision ZZ(K) = 0 decodes runs of zero coefficients. If the decoder is at this step of the procedure, at least one non-zero coefficient remains in the band of the block being coded. If ZZ(K) is not zero, the procedure in Figure G.11 is followed to code the value.

The context-indices in Figures G.10 and G.11 are defined in Table G.2 (see G.1.3.3.1). The signs of coefficients with magnitude of one are coded with a fixed probability value of approximately 0.5 (Qe = X’5A1D’, MPS = 0).

G.1.3.3.1 Statistical model for subsequent successive approximation scans

As shown in Table G.2, each statistics area for subsequent successive approximation scans of AC coefficients consists of a contiguous set of 189 statistics bins. The signs of coefficients with magnitude of one are coded with a fixed probability value of approximately 0.5 (Qe = X’5A1D’, MPS = 0).

G.2 Progressive decoding of the DCT

The description of the computation of the IDCT and the dequantization procedure contained in A.3.3 and A.3.4 apply to the progressive operation.

Progressive decoding processes must be able to decompress compressed image data which requires up to four sets of Huffman or arithmetic coder conditioning tables within a scan.

In order to avoid repetition, detailed flow diagrams of progressive decoder operation are not included. Decoder operation is defined by reversing the function of each step described in the encoder flow charts, and performing the steps in reverse order.

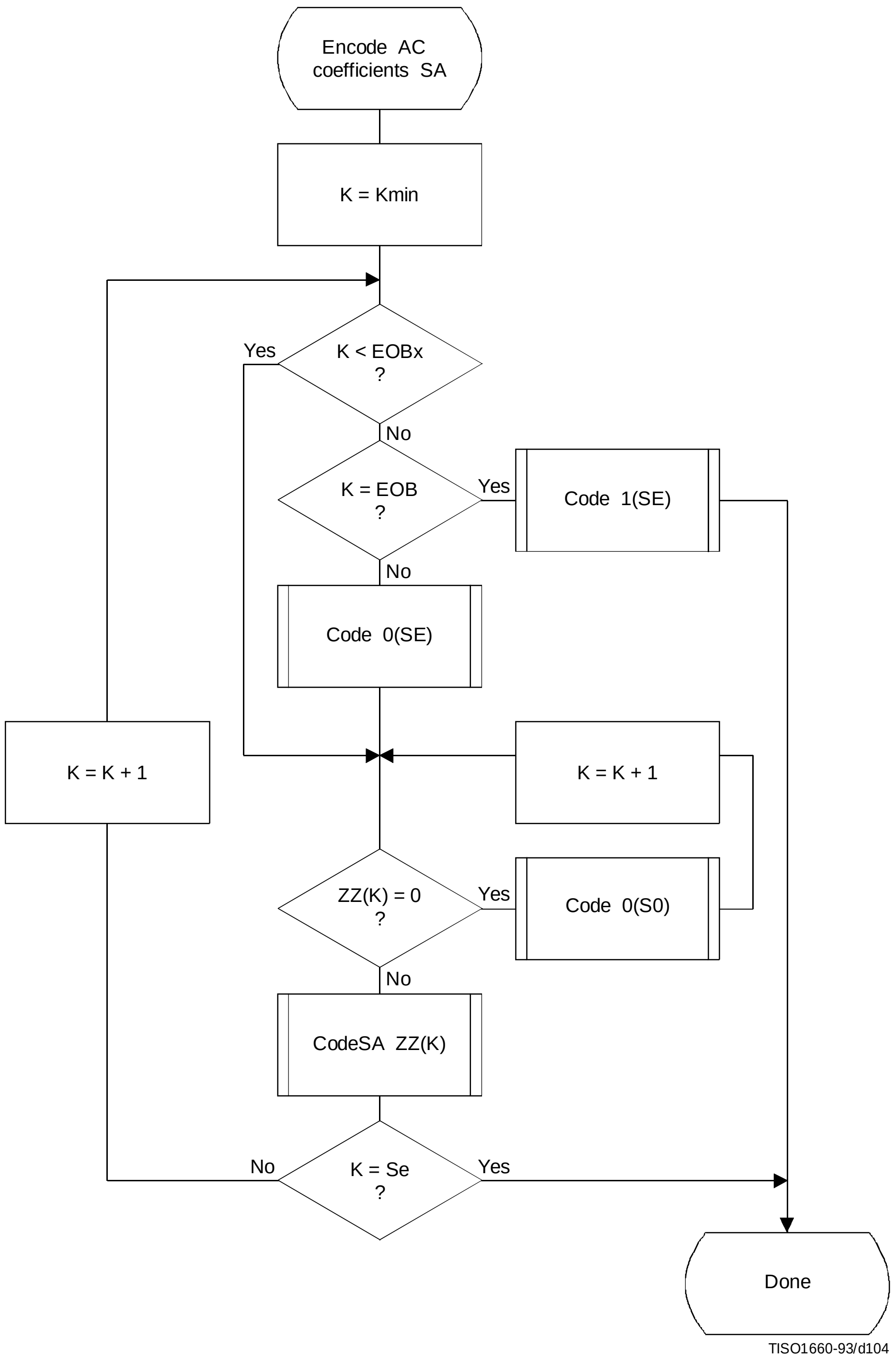


Figure G.10 – Subsequent successive approximation scans for coding of AC coefficients using arithmetic coding

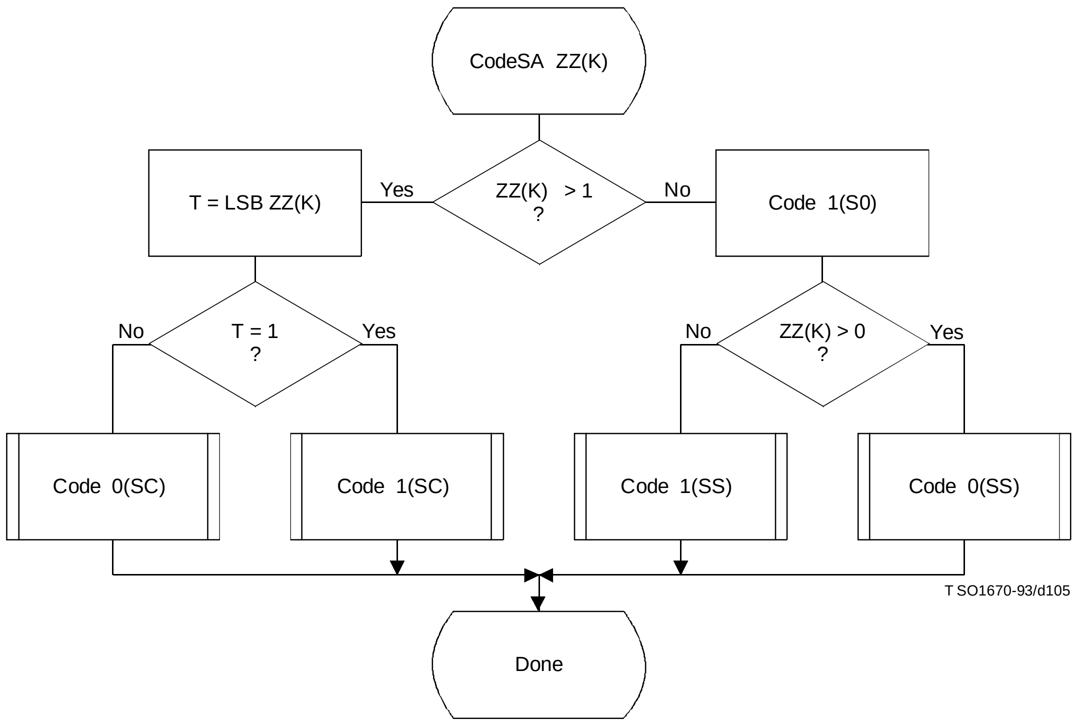


Figure G.11 – Coding non-zero coefficients for subsequent successive approximation scans

Table G.2 – Statistical model for subsequent scans of successive approximation coding of AC coefficient

Context-index	AC coding	Coding decision
SE	3 × (K–1)	K = EOB
S0	SE + 1	V = 0
SS	Fixed estimate	Sign
SC	S0 + 1	LSB ZZ(K) = 1

Annex H

Lossless mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the following coding processes for the lossless mode of operation:

1) lossless processes with Huffman coding;
2) lossless processes with arithmetic coding.

For each of these, the encoding process is specified in H.1, and the decoding process is specified in H.2. The functional specification is presented by means of specific procedures which comprise these coding processes.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

The processes which provide for sequential lossless encoding and decoding are not based on the DCT. The processes used are spatial processes based on the coding model developed for the DC coefficients of the DCT. However, the model is extended by incorporating a set of selectable one- and two-dimensional predictors, and for interleaved data the ordering of samples for the one-dimensional predictor can be different from that used in the DCT-based processes.

Either Huffman coding or arithmetic coding entropy coding may be employed for these lossless encoding and decoding processes. The Huffman code table structure is extended to allow up to 16-bit precision for the input data. The arithmetic coder statistical model is extended to a two-dimensional form.

H.1 Lossless encoder processes

H.1.1 Lossless encoder control procedures

Subclause E.1 contains the encoder control procedures. In applying these procedures to the lossless encoder, the data unit is one sample.

Input data precision may be from 2 to 16 bits/sample. If the input data path has different precision from the input data, the data shall be aligned with the least significant bits of the input data path. Input data is represented as unsigned integers and is not level shifted prior to coding.

When the encoder is reset in the restart interval control procedure (see E.1.4), the prediction is reset to a default value. If arithmetic coding is used, the statistics are also reset.

For the lossless processes the restart interval shall be an integer multiple of the number of MCU in an MCU-row.

H.1.2 Coding model for lossless encoding

The coding model developed for encoding the DC coefficients of the DCT is extended to allow a selection from a set of seven one-dimensional and two-dimensional predictors. The predictor is selected in the scan header (see Annex B). The same predictor is used for all components of the scan. Each component in the scan is modeled independently, using predictions derived from neighbouring samples of that component.

H.1.2.1 Prediction

Figure H.1 shows the relationship between the positions (a, b, c) of the reconstructed neighboring samples used for prediction and the position of x, the sample being coded.

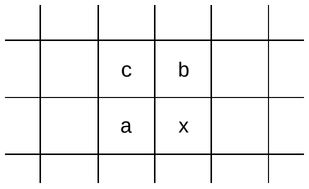


Figure H.1 – Relationship between sample and prediction samples

Define Px to be the prediction and Ra, Rb, and Rc to be the reconstructed samples immediately to the left, immediately above, and diagonally to the left of the current sample. The allowed predictors, one of which is selected in the scan header, are listed in Table H.1.

Table H.1 – Predictors for lossless coding

Selection-value	Prediction
0	No prediction (See Annex J)
1	Px = Ra
2	Px = Rb
3	Px = Rc
4	Px = Ra + Rb – Rc
5	Px = Ra + ((Rb – Rc)/2)[a]
6	Px = Rb + ((Ra – Rc)/2)[a]
7	Px = (Ra + Rb)/2

[a] Shift right arithmetic operation

Selection-value 0 shall only be used for differential coding in the hierarchical mode of operation. Selections 1, 2 and 3 are one-dimensional predictors and selections 4, 5, 6, and 7 are two-dimensional predictors.

The one-dimensional horizontal predictor (prediction sample Ra) is used for the first line of samples at the start of the scan and at the beginning of each restart interval. The selected predictor is used for all other lines. The sample from the line above (prediction sample Rb) is used at the start of each line, except for the first line. At the beginning of the first line and at the beginning of each restart interval the prediction value of 2^{P-1} is used, where P is the input precision.

If the point transformation parameter (see A.4) is non-zero, the prediction value at the beginning of the first lines and the beginning of each restart interval is 2^{P-Pt-1}, where Pt is the value of the point transformation parameter.

Each prediction is calculated with full integer arithmetic precision, and without clamping of either underflow or overflow beyond the input precision bounds. For example, if Ra and Rb are both 16-bit integers, the sum is a 17-bit integer. After dividing the sum by 2 (predictor 7), the prediction is a 16-bit integer.

For simplicity of implementation, the divide by 2 in the prediction selections 5 and 6 of Table H.1 is done by an arithmetic-right-shift of the integer values.

The difference between the prediction value and the input is calculated modulo 2^{16}. In the decoder the difference is decoded and added, modulo 2^{16}, to the prediction.

H.1.2.2 Huffman coding of the modulo difference

The Huffman coding procedures defined in Annex F for coding the DC coefficients are used to code the modulo 2^{16} differences. The table for DC coding contained in Tables F.1 and F.6 is extended by one additional entry. No extra bits are appended after SSSS = 16 is encoded. See Table H.2.

Table H.2 – Difference categories for lossless Huffman coding

SSSS	Difference values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383
15	–32 767..–16 384,16 384..32 767
16	32 768

H.1.2.3 Arithmetic coding of the modulo difference

The statistical model defined for the DC coefficient arithmetic coding model (see F.1.4.4.1) is generalized to a two-dimensional form in which differences coded for the sample to the left and for the line above are used for conditioning.

H.1.2.3.1 Two-dimensional statistical model

The binary decisions are conditioned on the differences coded for the neighbouring samples immediately above and immediately to the left from the same component. As in the coding of the DC coefficients, the differences are classified into 5 categories: zero(0), small positive (+S), small negative (–S), large positive (+L), and large negative (–L). The two independent difference categories combine to give 25 different conditioning states. Figure H.2 shows the two-dimensional array of conditioning indices. For each of the 25 conditioning states probability estimates for four binary decisions are kept.

At the beginning of the scan and each restart interval the conditioning derived from the line above is set to zero for the first line of each component. At the start of each line, the difference to the left is set to zero for the purposes of calculating the conditioning.

Difference above (position b)

Difference to eft (position a)

	0	+S	S	+L	L
0	0	4	8	12	16
+S	20	24	28	32	36
S	40	44	48	52	56
+L	60	64	68	72	76
L	80	84	88	92	96

T SO1690-93/d107

Figure H.2 – 5 × 5 Conditioning array for two-dimensional statistical model

H.1.2.3.2 Assignment of statistical bins to the DC binary decision tree

Each statistics area for lossless coding consists of a contiguous set of 158 statistics bins. The first 100 bins consist of 25 sets of four bins selected by a context-index S0. The value of S0 is given by L_Context(Da,Db), which provides a value of 0, 4,..., 92 or 96, depending on the difference classifications of Da and Db (see H.1.2.3.1). The value for S0 provided by L_Context(Da,Db) is from the array in Figure H.2.

The remaining 58 bins consist of two sets of 29 bins, X1, ..., X15, M2, ..., M15, which are used to code magnitude category decisions and magnitude bits. The value of X1 is given by X1_Context(Db), which provides a value of 100 when Db is in the zero, small positive or small negative categories and a value of 129 when Db is in the large positive or large negative categories.

The assignment of statistical bins to the binary decision tree used for coding the difference is given in Table H.3.

Table H.3 – Statistical model for lossless coding

Context-index	Value	Coding decision
S0	L_Context(Da,Db)	V = 0
SS	S0 + 1	Sign
SP	S0 + 2	Sz < 1 if V > 0
SN	S0 + 3	Sz < 1 if V < 0
X1	X1_Context(Db)	Sz < 2
X2	X1 + 1	Sz < 4
X3	X1 + 2	Sz < 8
.	.	.
.	.	.
X15	X1 + 14	Sz < 2^{15}
M2	X2 + 14	Magnitude bits if Sz < 4
M3	X3 + 14	Magnitude bits if Sz < 8
.	.	.
.	.	.
M15	X15 + 14	Magnitude bits if Sz < 2^{15}

H.1.2.3.3 Default conditioning bounds

The bounds, L and U, for determining the conditioning category have the default values L = 0 and U = 1. Other bounds may be set using the DAC (Define-Arithmetic-Conditioning) marker segment, as described in Annex B.

H.1.2.3.4 Initial conditions for statistical model

At the start of a scan and at each restart, all statistics bins are re-initialized to the standard default value described in Annex D.

H.2 Lossless decoder processes

Lossless decoders may employ either Huffman decoding or arithmetic decoding. They shall be capable of using up to four tables in a scan. Lossless decoders shall be able to decode encoded image source data with any input precision from 2 to 16 bits per sample.

H.2.1 Lossless decoder control procedures

Subclause E.2 contains the decoder control procedures. In applying these procedures to the lossless decoder the data unit is one sample.

When the decoder is reset in the restart interval control procedure (see E.2.4) the prediction is reset to the same value used in the encoder (see H.1.2.1). If arithmetic coding is used, the statistics are also reset.

Restrictions on the restart interval are specified in H.1.1.

H.2.2 Coding model for lossless decoding

The predictor calculations defined in H.1.2 also apply to the lossless decoder processes.

The lossless decoders, decode the differences and add them, modulo 2^{16}, to the predictions to create the output. The lossless decoders shall be able to interpret the point transform parameter, and if non-zero, multiply the output of the lossless decoder by 2^{Pt}.

In order to avoid repetition, detailed flow charts of the lossless decoding procedures are omitted.

Annex J

Hierarchical mode of operation

(This annex forms an integral part of this Recommendation | International Standard)

This annex provides a **functional specification** of the coding processes for the hierarchical mode of operation.

In the hierarchical mode of operation each component is encoded or decoded in a non-differential frame. Such frames may be followed by a sequence of differential frames. A non-differential frame shall be encoded or decoded using the procedures defined in Annexes F, G and H. Differential frame procedures are defined in this annex.

The coding process for a hierarchical encoding containing DCT-based processes is defined as the highest numbered process listed in Table J.1 which is used to code any non-differential DCT-based or differential DCT-based frame in the compressed image data format. The coding process for a hierarchical encoding containing only lossless processes is defined to be the process used for the non-differential frames.

Table J.1 – Coding processes for hierarchical mode

Process	Non-differential frame specification	
1	Extended sequential DCT, Huffman, 8-bit	Annex F, process 2
2	Extended sequential DCT, arithmetic, 8-bit	Annex F, process 3
3	Extended sequential DCT, Huffman, 12-bit	Annex F, process 4
4	Extended sequential DCT, arithmetic, 12-bit	Annex F, process 5
5	Spectral selection only, Huffman, 8-bit	Annex G, process 1
6	Spectral selection only, arithmetic, 8-bit	Annex G, process 2
7	Full progression, Huffman, 8-bit	Annex G, process 3
8	Full progression, arithmetic, 8-bit	Annex G, process 4
9	Spectral selection only, Huffman, 12-bit	Annex G, process 5
10	Spectral selection only, arithmetic, 12-bit	Annex G, process 6
11	Full progression, Huffman, 12-bit	Annex G, process 7
12	Full progression, arithmetic, 12-bit	Annex G, process 8
13	Lossless, Huffman, 2 through 16 bits	Annex H, process 1
14	Lossless, arithmetic, 2 through 16 bits	Annex H, process 2

Hierarchical mode syntax requires a DHP marker segment that appears before the non-differential frame or frames. It may include EXP marker segments and differential frames which shall follow the initial non-differential frame. The frame structure in hierarchical mode is identical to the frame structure in non-hierarchical mode.

Either all non-differential frames within an image shall be coded with DCT-based processes, or all non-differential frames shall be coded with lossless processes. All frames within an image must use the same entropy coding procedure, either Huffman or arithmetic, with the exception that non-differential frames coded with the baseline process may occur in the same image with frames coded with arithmetic coding processes.

If the non-differential frames use DCT-based processes, all differential frames except the final frame for a component shall use DCT-based processes. The final differential frame for each component may use a differential lossless process.

If the non-differential frames use lossless processes, all differential frames shall use differential lossless processes.

For each of the processes listed in Table J.1, the encoding processes are specified in J.1, and decoding processes are specified in J.2.

NOTE – There is **no requirement** in this Specification that any encoder or decoder which embodies one of the above-named processes shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in compliance with this Specification is that it satisfy the requirements given in clause 6 (for encoders) or clause 7 (for decoders), as determined by the compliance tests specified in Part 2.

In the hierarchical mode of operation each component is encoded or decoded in a non-differential frame followed by a sequence of differential frames. A non-differential frame shall use the procedures defined in Annexes F, G, and H. Differential frame procedures are defined in this annex.

J.1 Hierarchical encoding

J.1.1 Hierarchical control procedure for encoding an image

The control structure for encoding of an image using the hierarchical mode is given in Figure J.1.

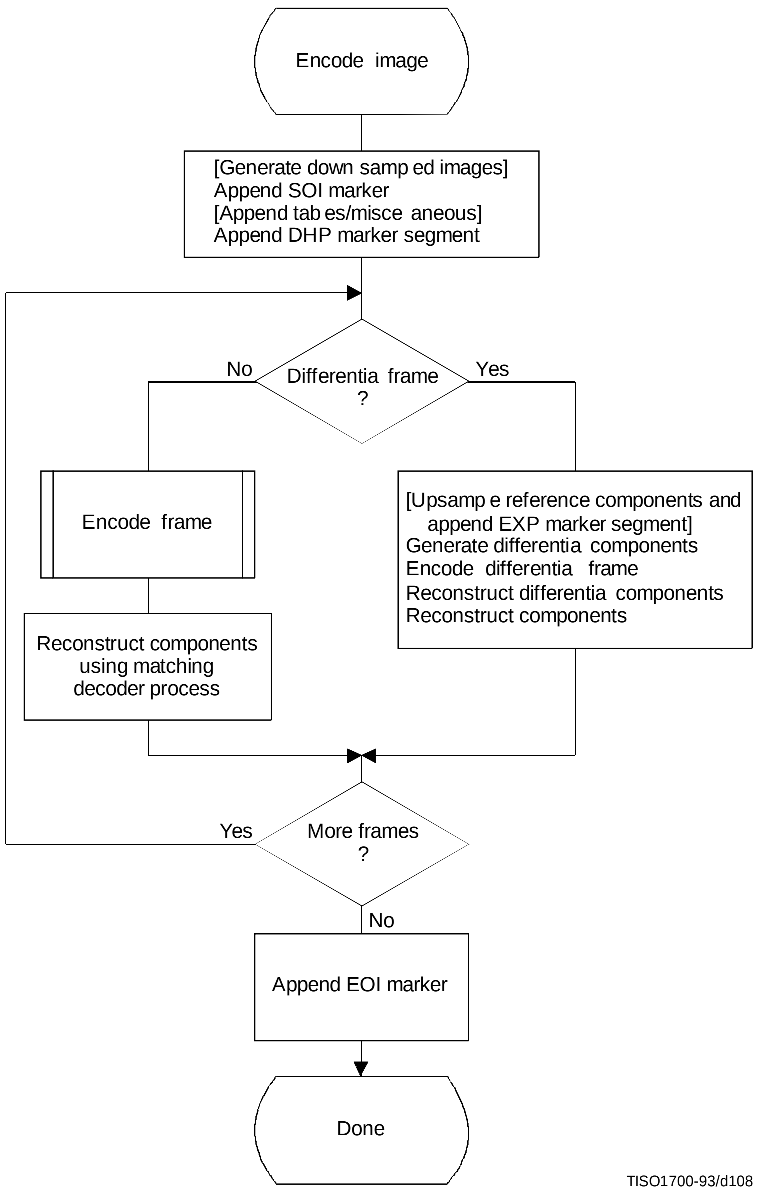

Encode image
[Generate down samp ed images]
Append SOI marker
[Append tab es/misce aneous]
Append DHP marker segment
Differentia frame ?
No
Yes
Encode frame
Reconstruct components using matching decoder process
[Upsamp e reference components and append EXP marker segment]
Generate differentia components
Encode differentia frame
Reconstruct differentia components
Reconstruct components
More frames ?
Yes
No
Append EOI marker
Done
TISO1700-93/d108

Figure J.1 – Hierarchical control procedure for encoding an image

In Figure J.1 procedures in brackets shall be performed whenever the particular hierarchical encoding sequence being followed requires them.

In the hierarchical mode the define-hierarchical-progression (DHP) marker segment shall be placed in the compressed image data before the first start-of-frame. The DHP segment is used to signal the size of the image components of the completed image. The syntax of the DHP segment is specified in Annex B.

The first frame for each component or group of components in a hierarchical process shall be encoded by a non-differential frame. Differential frames shall then be used to encode the two's complement differences between source input components (possibly downsampled) and the reference components (possibly upsampled). The reference components are reconstructed components created by previous frames in the hierarchical process. For either differential or non-differential frames, reconstructions of the components shall be generated if needed as reference components for a subsequent frame in the hierarchical process.

Resolution changes may occur between hierarchical frames in a hierarchical process. These changes occur if downsampling filters are used to reduce the spatial resolution of some or all of the components of the source image. When the resolution of a reference component does not match the resolution of the component input to a differential frame, an upsampling filter shall be used to increase the spatial resolution of the reference component. The EXP marker segment shall be added to the compressed image data before the start-of-frame whenever upsampling of a reference component is required. No more than one EXP marker segment shall precede a given frame.

Any of the marker segments allowed before a start-of-frame for the encoding process selected may be used before either non-differential or differential frames.

For 16-bit input precision (lossless encoder), the differential components which are input to a differential frame are calculated modulo 2^{16}. The reconstructed components calculated from the reconstructed differential components are also calculated modulo 2^{16}.

If a hierarchical encoding process uses a DCT encoding process for the first frame, all frames in the hierarchical process except for the final frame for each component shall use the DCT encoding processes defined in either Annex F or Annex G, or the modified DCT encoding processes defined in this annex. The final frame may use a modified lossless process defined in this annex.

If a hierarchical encoding process uses a lossless encoding process for the first frame, all frames in the hierarchical process shall use a lossless encoding process defined in Annex H, or a modified lossless process defined in this annex.

J.1.1.1 Downsampling filter

The downsampled components are generated using a downsampling filter that is not specified in this Specification. This filter should, however, be consistent with the upsampling filter. An example of a downsampling filter is provided in K.5.

J.1.1.2 Upsampling filter

The upsampling filter increases the spatial resolution by a factor of two horizontally, vertically, or both. Bi-linear interpolation is used for the upsampling filter, as illustrated in Figure J.2.

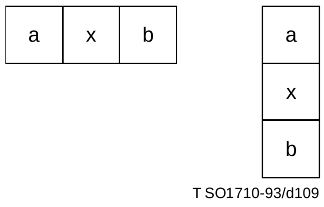


Figure J.2 – Diagram of sample positions for upsampling rules

The rule for calculating the interpolated value is:

$$P_x = (Ra + Rb) / 2$$

where Ra and Rb are sample values from adjacent positions a and b of the lower resolution image and Px is the interpolated value. The division indicates truncation, not rounding. The left-most column of the upsampled image matches the left-most column of the lower resolution image. The top line of the upsampled image matches the top line of the lower resolution image. The right column and the bottom line of the lower resolution image are replicated to provide the values required for the right column edge and bottom line interpolations. The upsampling process always doubles the line length or the number of lines.

If both horizontal and vertical expansions are signalled, they are done in sequence – first the horizontal expansion and then the vertical.

J.1.2 Control procedure for encoding a differential frame

The control procedures in Annex E for frames, scans, restart intervals, and MCU also apply to the encoding of differential frames, and the scans, restart intervals, and MCU from which the differential frame is constructed. The differential frames differ from the frames of Annexes F, G, and H only at the coding model level.

J.1.3 Encoder coding models for differential frames

The coding models defined in Annexes F, G, and H are modified to allow them to be used for coding of two’s complement differences.

J.1.3.1 Modifications to encoder DCT encoding models for differential frames

Two modifications are made to the DCT coding models to allow them to be used in differential frames. First, the FDCT of the differential input is calculated without the level shift. Second, the DC coefficient of the DCT is coded directly – without prediction.

J.1.3.2 Modifications to lossless encoding models for differential frames

One modification is made to the lossless coding models. The difference is coded directly – without prediction. The prediction selection parameter in the scan header shall be set to zero. The point transform which may be applied to the differential inputs is defined in Annex A.

J.1.4 Modifications to the entropy encoders for differential frames

The coding of two’s complement differences requires one extra bit of precision for the Huffman coding of AC coefficients. The extension to Tables F.1 and F.7 is given in Table J.2.

Table J.2 – Modifications to table of AC coefficient amplitude ranges

SSSS	AC coefficients
15	–32 767..–16 384, 16 384..32 767

The arithmetic coding models are already defined for the precision needed in differential frames.

J.2 Hierarchical decoding

J.2.1 Hierarchical control procedure for decoding an image

The control structure for decoding an image using the hierarchical mode is given in Figure J.3.

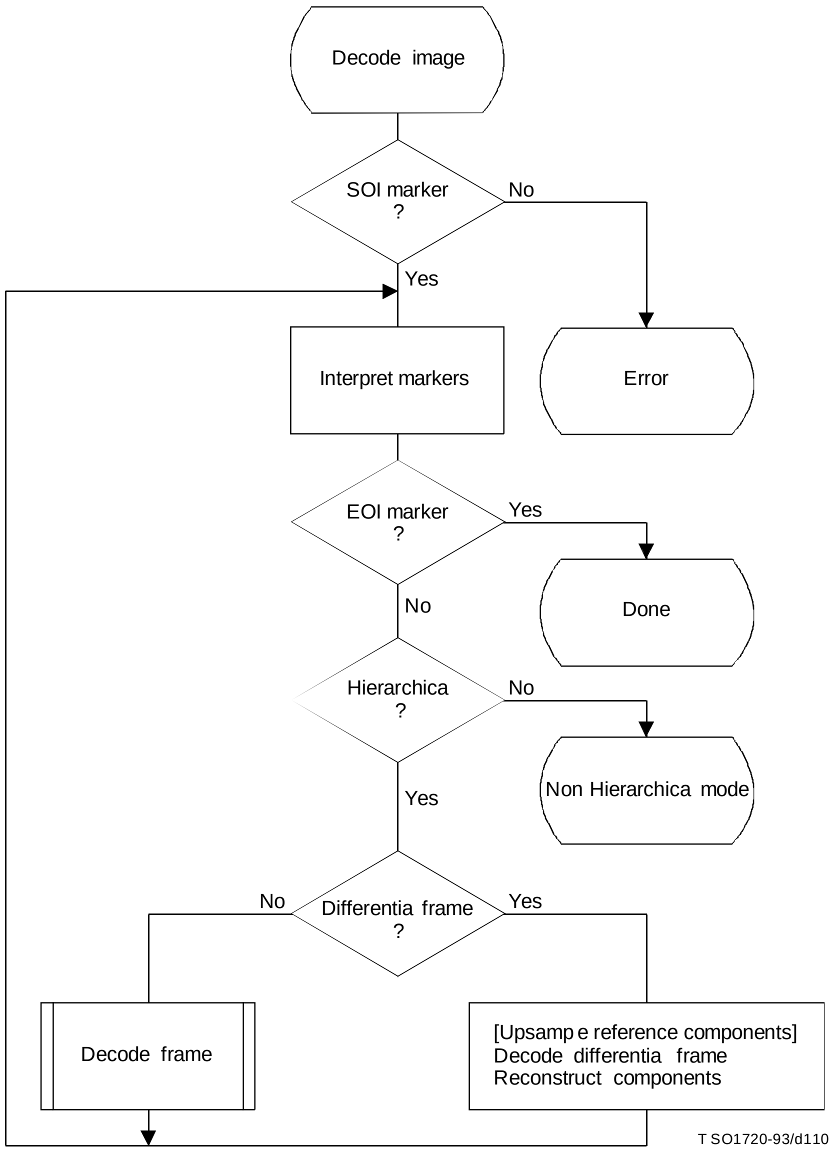


Figure J.3 – Hierarchical control procedure for decoding an image

The Interpret markers procedure shall decode the markers which may precede the SOF marker, continuing this decoding until either a SOF or EOI marker is found. If the DHP marker is encountered before the first frame, a flag is set which selects the hierarchical decoder at the "hierarchical?" decision point. In addition to the DHP marker (which shall precede any SOF) and the EXP marker (which shall precede any differential SOF requiring resolution changes in the reference components), any other markers which may precede a SOF shall be interpreted to the extent required for decoding of the compressed image data.

If a differential SOF marker is found, the differential frame path is followed. If the EXP was encountered in the Interpret markers procedure, the reference components for the frame shall be upsampled as required by the parameters in the EXP segment. The upsampling procedure described in J.1.1.2 shall be followed.

The Decode_differential_frame procedure generates a set of differential components. These differential components shall be added, modulo 2^{16}, to the upsampled reference components in the Reconstruct_components procedure. This creates a new set of reference components which shall be used when required in subsequent frames of the hierarchical process.

J.2.2 Control procedure for decoding a differential frame

The control procedures in Annex E for frames, scans, restart intervals, and MCU also apply to the decoding of differential frames and the scans, restart intervals, and MCU from which the differential frame is constructed. The differential frame differs from the frames of Annexes F, G, and H only at the decoder coding model level.

J.2.3 Decoder coding models for differential frames

The decoding models described in Annexes F, G, and H are modified to allow them to be used for decoding of two's complement differential components.

J.2.3.1 Modifications to the differential frame decoder DCT coding model

Two modifications are made to the decoder DCT coding models to allow them to code differential frames. First, the IDCT of the differential output is calculated without the level shift. Second, the DC coefficient of the DCT is decoded directly – without prediction.

J.2.3.2 Modifications to the differential frame decoder lossless coding model

One modification is made to the lossless decoder coding model. The difference is decoded directly – without prediction. If the point transformation parameter in the scan header is not zero, the point transform, defined in Annex A, shall be applied to the differential output.

J.2.4 Modifications to the entropy decoders for differential frames

The decoding of two's complement differences requires one extra bit of precision in the Huffman code table. This is described in J.1.4. The arithmetic coding models are already defined for the precision needed in differential frames.

Annex K

Examples and guidelines

(This annex does not form an integral part of this Recommendation | International Standard)

This annex provides examples of various tables, procedures, and other guidelines.

K.1 Quantization tables for luminance and chrominance components

Two examples of quantization tables are given in Tables K.1 and K.2. These are based on psychovisual thresholding and are derived empirically using luminance and chrominance and 2:1 horizontal subsampling. These tables are provided as examples only and are not necessarily suitable for any particular application. These quantization values have been used with good results on 8-bit per sample luminance and chrominance images of the format illustrated in Figure 13. Note that these quantization values are appropriate for the DCT normalization defined in A.3.3.

If these quantization values are divided by 2, the resulting reconstructed image is usually nearly indistinguishable from the source image.

Table K.1 – Luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table K.2 – Chrominance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

K.2 A procedure for generating the lists which specify a Huffman code table

A Huffman table is generated from a collection of statistics in two steps. The first step is the generation of the list of lengths and values which are in accord with the rules for generating the Huffman code tables. The second step is the generation of the Huffman code table from the list of lengths and values.

The first step, the topic of this section, is needed only for custom Huffman table generation and is done only in the encoder. In this step the statistics are used to create a table associating each value to be coded with the size (in bits) of the corresponding Huffman code. This table is sorted by code size.

A procedure for creating a Huffman table for a set of up to 256 symbols is shown in Figure K.1. Three vectors are defined for this procedure:

FREQ(V)	Frequency of occurrence of symbol V
CODESIZE(V)	Code size of symbol V
OTHERS(V)	Index to next symbol in chain of all symbols in current branch of code tree

where V goes from 0 to 256.

Before starting the procedure, the values of FREQ are collected for V = 0 to 255 and the FREQ value for V = 256 is set to 1 to reserve one code point. FREQ values for unused symbols are defined to be zero. In addition, the entries in CODESIZE are all set to 0, and the indices in OTHERS are set to –1, the value which terminates a chain of indices. Reserving one code point guarantees that no code word can ever be all “1” bits.

The search for the entry with the least value of FREQ(V) selects the largest value of V with the least value of FREQ(V) greater than zero.

The procedure “Find V1 for least value of FREQ(V1) > 0” always selects the value with the largest value of V1 when more than one V1 with the same frequency occurs. The reserved code point is then guaranteed to be in the longest code word category.

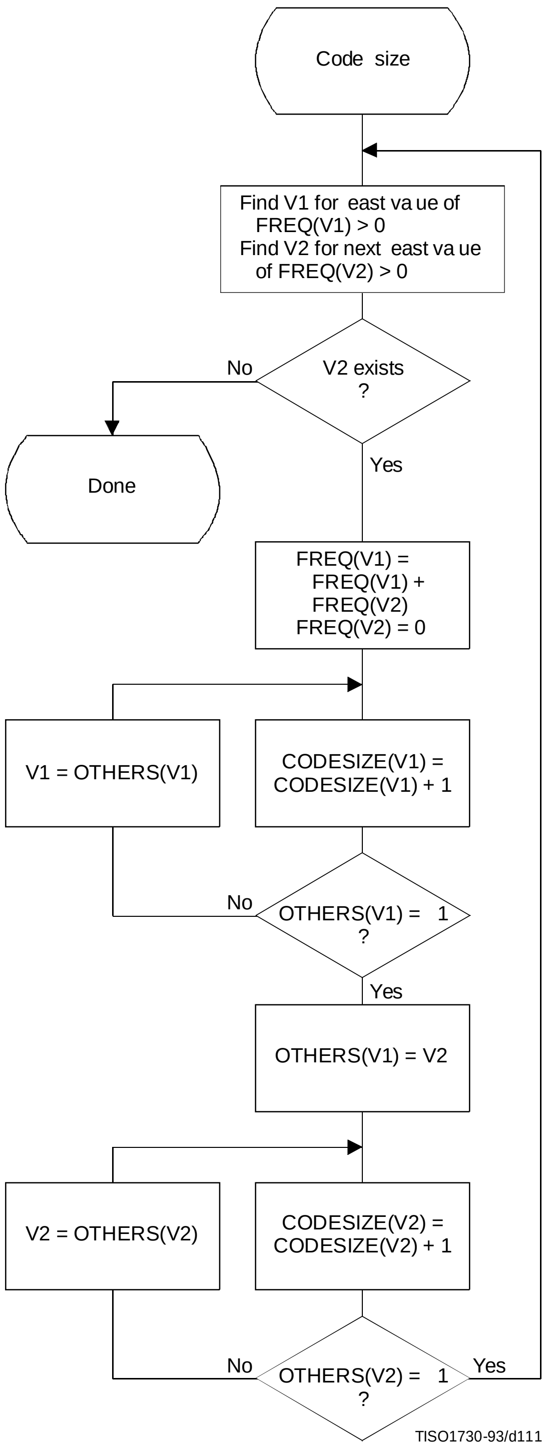

Code size
Find V1 for east va ue of FREQ(V1) > 0
Find V2 for next east va ue of FREQ(V2) > 0
V2 exists ?
No
Done
Yes
FREQ(V1) = FREQ(V1) + FREQ(V2)
FREQ(V2) = 0
V1 = OTHERS(V1)
CODESIZE(V1) = CODESIZE(V1) + 1
No
OTHERS(V1) = 1 ?
Yes
OTHERS(V1) = V2
V2 = OTHERS(V2)
CODESIZE(V2) = CODESIZE(V2) + 1
No
OTHERS(V2) = 1 ?
Yes
TISO1730-93/d111

Figure K.1 – Procedure to find Huffman code sizes

Once the code lengths for each symbol have been obtained, the number of codes of each length is obtained using the procedure in Figure K.2. The count for each size is contained in the list, BITS. The counts in BITS are zero at the start of the procedure. The procedure assumes that the probabilities are large enough that code lengths greater than 32 bits never occur. Note that until the final Adjust_BITS procedure is complete, BITS may have more than the 16 entries required in the table specification (see Annex C).

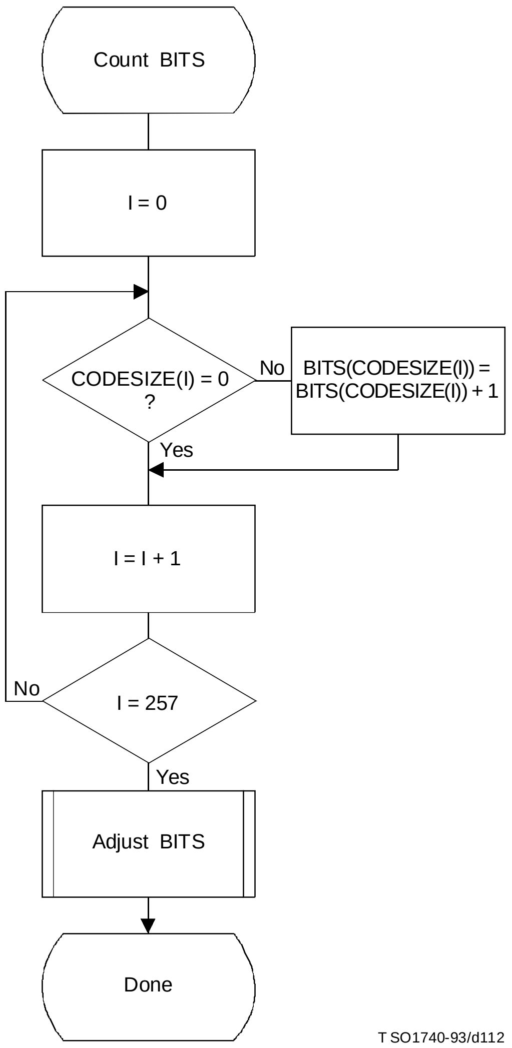


Figure K.2 – Procedure to find the number of codes of each size

Figure K.3 gives the procedure for adjusting the BITS list so that no code is longer than 16 bits. Since symbols are paired for the longest Huffman code, the symbols are removed from this length category two at a time. The prefix for the pair (which is one bit shorter) is allocated to one of the pair; then (skipping the BITS entry for that prefix length) a code word from the next shortest non-zero BITS entry is converted into a prefix for two code words one bit longer. After the BITS list is reduced to a maximum code length of 16 bits, the last step removes the reserved code point from the code length count.

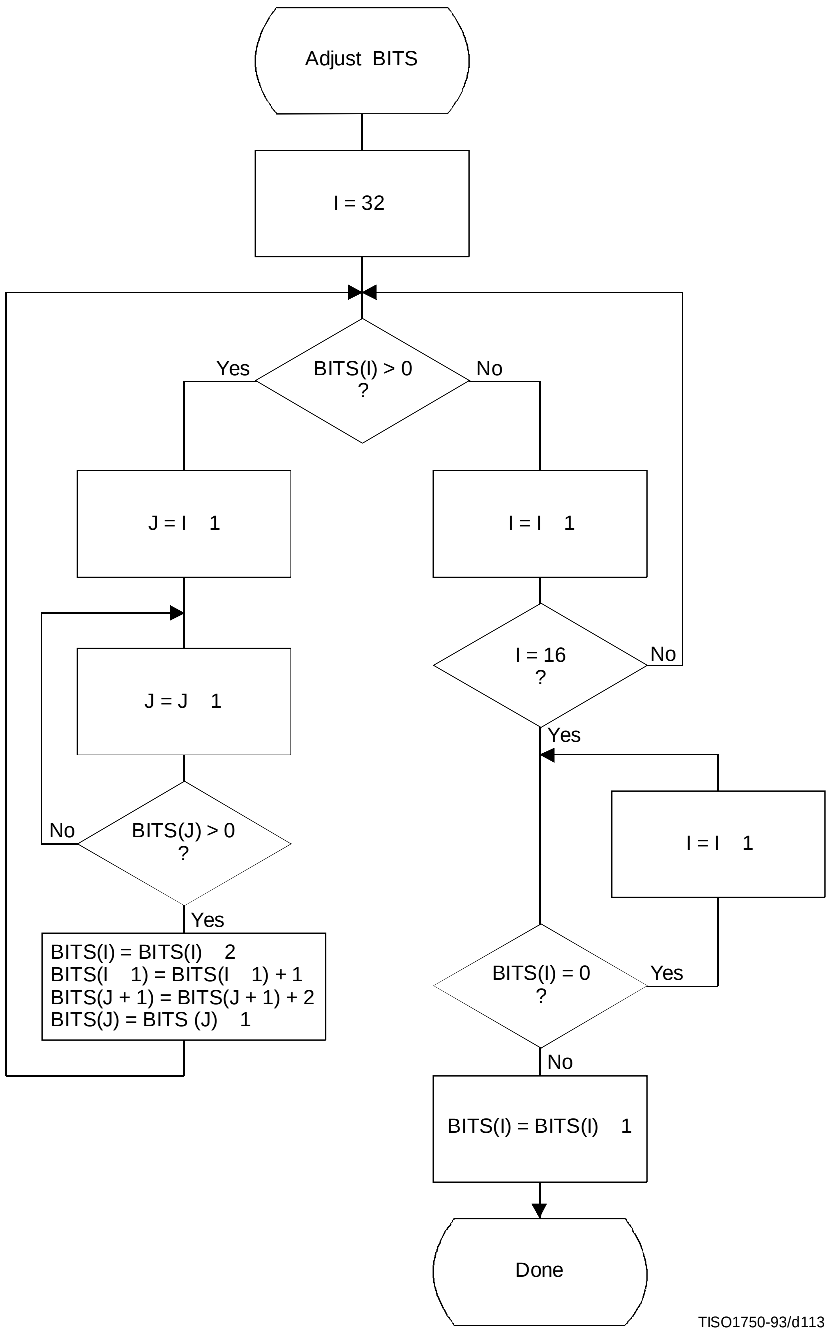


Figure K.3 – Procedure for limiting code lengths to 16 bits

The input values are sorted according to code size as shown in Figure K.4. HUFFVAL is the list containing the input values associated with each code word, in order of increasing code length.

At this point, the list of code lengths (BITS) and the list of values (HUFFVAL) can be used to generate the code tables. These procedures are described in Annex C.

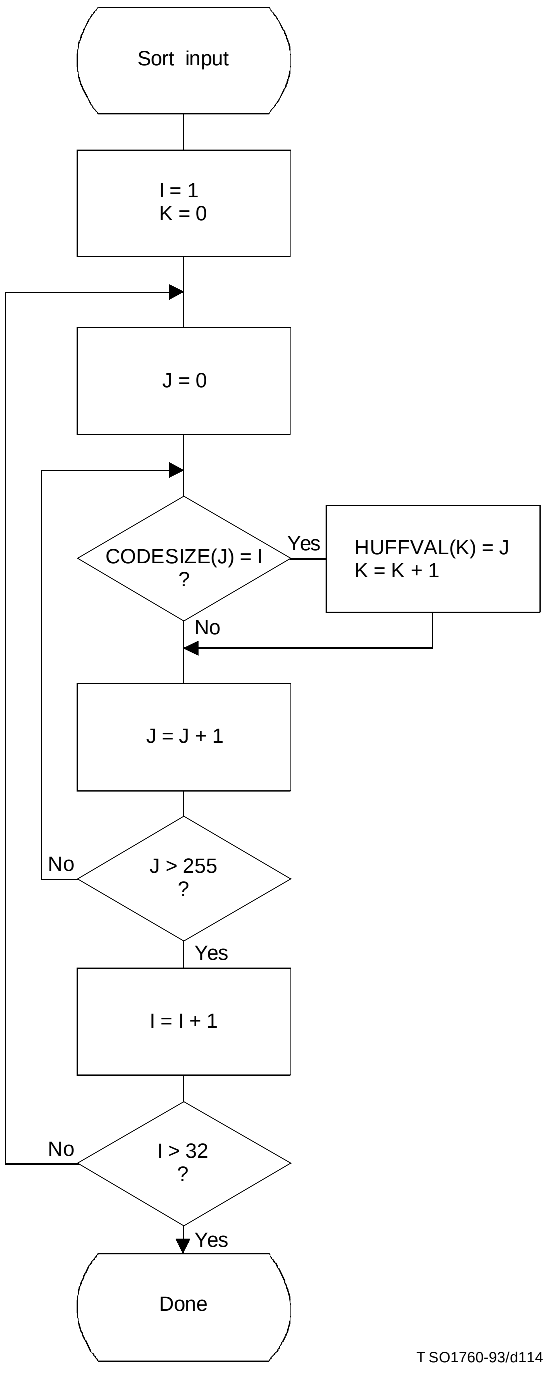


Figure K.4 – Sorting of input values according to code size

K.3 Typical Huffman tables for 8-bit precision luminance and chrominance

Huffman table-specification syntax is specified in B.2.4.2.

K.3.1 Typical Huffman tables for the DC coefficient differences

Tables K.3 and K.4 give Huffman tables for the DC coefficient differences which have been developed from the average statistics of a large set of video images with 8-bit precision. Table K.3 is appropriate for luminance components and Table K.4 is appropriate for chrominance components. Although there are no default tables, these tables may prove to be useful for many applications.

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Table K.4 – Table for chrominance DC coefficient differences

Category	Code length	Code word
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

K.3.2 Typical Huffman tables for the AC coefficients

Tables K.5 and K.6 give Huffman tables for the AC coefficients which have been developed from the average statistics of a large set of images with 8-bit precision. Table K.5 is appropriate for luminance components and Table K.6 is appropriate for chrominance components. Although there are no default tables, these tables may prove to be useful for many applications.

Table K.5 – Table for luminance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Table K.5 (sheet 2 of 4)

Run/Size	Code length	Code word
4/1	6	111011
4/2	10	1111111000
4/3	16	1111111110010110
4/4	16	1111111110010111
4/5	16	1111111110011000
4/6	16	1111111110011001
4/7	16	1111111110011010
4/8	16	1111111110011011
4/9	16	1111111110011100
4/A	16	1111111110011101
5/1	7	1111010
5/2	11	11111110111
5/3	16	1111111110011110
5/4	16	1111111110011111
5/5	16	1111111110100000
5/6	16	1111111110100001
5/7	16	1111111110100010
5/8	16	1111111110100011
5/9	16	1111111110100100
5/A	16	1111111110100101
6/1	7	1111011
6/2	12	111111110110
6/3	16	1111111110100110
6/4	16	1111111110100111
6/5	16	1111111110101000
6/6	16	1111111110101001
6/7	16	1111111110101010
6/8	16	1111111110101011
6/9	16	1111111110101100
6/A	16	1111111110101101
7/1	8	11111010
7/2	12	111111110111
7/3	16	1111111110101110
7/4	16	1111111110101111
7/5	16	1111111110110000
7/6	16	1111111110110001
7/7	16	1111111110110010
7/8	16	1111111110110011
7/9	16	1111111110110100
7/A	16	1111111110110101
8/1	9	111111000
8/2	15	111111111000000

Table K.5 (sheet 3 of 4)

Run/Size	Code length	Code word
8/3	16	1111111110110110
8/4	16	1111111110110111
8/5	16	1111111110111000
8/6	16	1111111110111001
8/7	16	1111111110111010
8/8	16	1111111110111011
8/9	16	1111111110111100
8/A	16	1111111110111101
9/1	9	111111001
9/2	16	1111111110111110
9/3	16	1111111110111111
9/4	16	1111111111000000
9/5	16	1111111111000001
9/6	16	1111111111000010
9/7	16	1111111111000011
9/8	16	1111111111000100
9/9	16	1111111111000101
9/A	16	1111111111000110
A/1	9	111111010
A/2	16	1111111111000111
A/3	16	1111111111001000
A/4	16	1111111111001001
A/5	16	1111111111001010
A/6	16	1111111111001011
A/7	16	1111111111001100
A/8	16	1111111111001101
A/9	16	1111111111001110
A/A	16	1111111111001111
B/1	10	1111111001
B/2	16	1111111111010000
B/3	16	1111111111010001
B/4	16	1111111111010010
B/5	16	1111111111010011
B/6	16	1111111111010100
B/7	16	1111111111010101
B/8	16	1111111111010110
B/9	16	1111111111010111
B/A	16	1111111111011000
C/1	10	1111111010
C/2	16	1111111111011001
C/3	16	1111111111011010
C/4	16	1111111111011011

Table K.5 (sheet 4 of 4)

Run/Size	Code length	Code word
C/5	16	1111111111011100
C/6	16	1111111111011101
C/7	16	1111111111011110
C/8	16	1111111111011111
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
D/2	16	1111111111100010
D/3	16	1111111111100011
D/4	16	1111111111100100
D/5	16	1111111111100101
D/6	16	1111111111100110
D/7	16	1111111111100111
D/8	16	1111111111101000
D/9	16	1111111111101001
D/A	16	1111111111101010
E/1	16	1111111111101011
E/2	16	1111111111101100
E/3	16	1111111111101101
E/4	16	1111111111101110
E/5	16	1111111111101111
E/6	16	1111111111110000
E/7	16	1111111111110001
E/8	16	1111111111110010
E/9	16	1111111111110011
E/A	16	1111111111110100
F/0 (ZRL)	11	11111111001
F/1	16	1111111111110101
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

Table K.6 – Table for chrominance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110
2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010

Table K.6 (sheet 2 of 4)

Run/Size	Code length	Code word
4/2	9	111110110
4/3	16	1111111110010111
4/4	16	1111111110011000
4/5	16	1111111110011001
4/6	16	1111111110011010
4/7	16	1111111110011011
4/8	16	1111111110011100
4/9	16	1111111110011101
4/A	16	1111111110011110
5/1	6	111011
5/2	10	1111111001
5/3	16	1111111110011111
5/4	16	1111111110100000
5/5	16	1111111110100001
5/6	16	1111111110100010
5/7	16	1111111110100011
5/8	16	1111111110100100
5/9	16	1111111110100101
5/A	16	1111111110100110
6/1	7	1111001
6/2	11	11111110111
6/3	16	1111111110100111
6/4	16	1111111110101000
6/5	16	1111111110101001
6/6	16	1111111110101010
6/7	16	1111111110101011
6/8	16	1111111110101100
6/9	16	1111111110101101
6/A	16	1111111110101110
7/1	7	1111010
7/2	11	11111111000
7/3	16	1111111110101111
7/4	16	1111111110110000
7/5	16	1111111110110001
7/6	16	1111111110110010
7/7	16	1111111110110011
7/8	16	1111111110110100
7/9	16	1111111110110101
7/A	16	1111111110110110
8/1	8	11111001
8/2	16	1111111110110111
8/3	16	1111111110111000

Table K.6 (sheet 3 of 4)

Run/Size	Code length	Code word
8/4	16	1111111110111001
8/5	16	1111111110111010
8/6	16	1111111110111011
8/7	16	1111111110111100
8/8	16	1111111110111101
8/9	16	1111111110111110
8/A	16	1111111110111111
9/1	9	111110111
9/2	16	1111111111000000
9/3	16	1111111111000001
9/4	16	1111111111000010
9/5	16	1111111111000011
9/6	16	1111111111000100
9/7	16	1111111111000101
9/8	16	1111111111000110
9/9	16	1111111111000111
9/A	16	1111111111001000
A/1	9	111111000
A/2	16	1111111111001001
A/3	16	1111111111001010
A/4	16	1111111111001011
A/5	16	1111111111001100
A/6	16	1111111111001101
A/7	16	1111111111001110
A/8	16	1111111111001111
A/9	16	1111111111010000
A/A	16	1111111111010001
B/1	9	111111001
B/2	16	1111111111010010
B/3	16	1111111111010011
B/4	16	1111111111010100
B/5	16	1111111111010101
B/6	16	1111111111010110
B/7	16	1111111111010111
B/8	16	1111111111011000
B/9	16	1111111111011001
B/A	16	1111111111011010
C/1	9	111111010
C/2	16	1111111111011011
C/3	16	1111111111011100
C/4	16	1111111111011101
C/5	16	1111111111011110

Table K.6 (sheet 4 of 4)

Run/Size	Code length	Code word
C/6	16	1111111111011111
C/7	16	1111111111100000
C/8	16	1111111111100001
C/9	16	1111111111100010
C/A	16	1111111111100011
D/1	11	11111111001
D/2	16	1111111111100100
D/3	16	1111111111100101
D/4	16	1111111111100110
D/5	16	1111111111100111
D/6	16	1111111111101000
D/7	16	1111111111101001
D/8	16	1111111111101010
D/9	16	1111111111101011
D/A	16	1111111111101100
E/1	14	11111111100000
E/2	16	1111111111101101
E/3	16	1111111111101110
E/4	16	1111111111101111
E/5	16	1111111111110000
E/6	16	1111111111110001
E/7	16	1111111111110010
E/8	16	1111111111110011
E/9	16	1111111111110100
E/A	16	1111111111110101
F/0 (ZRL)	10	1111111010
F/1	15	111111111000011
F/2	16	1111111111110110
F/3	16	1111111111110111
F/4	16	1111111111111000
F/5	16	1111111111111001
F/6	16	1111111111111010
F/7	16	1111111111111011
F/8	16	1111111111111100
F/9	16	1111111111111101
F/A	16	1111111111111110

K.3.3 Huffman table-specification examples

K.3.3.1 Specification of typical tables for DC difference coding

A set of typical tables for DC component coding is given in K.3.1. The specification of these tables is as follows:

For Table K.3 (for luminance DC coefficients), the 16 bytes which specify the list of code lengths for the table are

```
X’00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00’
```

The set of values following this list is

```
X’00 01 02 03 04 05 06 07 08 09 0A 0B’
```

For Table K.4 (for chrominance DC coefficients), the 16 bytes which specify the list of code lengths for the table are

```
X’00 03 01 01 01 01 01 01 01 01 01 00 00 00 00 00’
```

The set of values following this list is

```
X’00 01 02 03 04 05 06 07 08 09 0A 0B’
```

K.3.3.2 Specification of typical tables for AC coefficient coding

A set of typical tables for AC component coding is given in K.3.2. The specification of these tables is as follows:

For Table K.5 (for luminance AC coefficients), the 16 bytes which specify the list of code lengths for the table are

```
X’00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D’
```

The set of values which follows this list is

```
X’01 02 03 00 04 11 05 12 21 31 41 06 13 51 61 07
  22 71 14 32 81 91 A1 08 23 42 B1 C1 15 52 D1 F0
  24 33 62 72 82 09 0A 16 17 18 19 1A 25 26 27 28
  29 2A 34 35 36 37 38 39 3A 43 44 45 46 47 48 49
  4A 53 54 55 56 57 58 59 5A 63 64 65 66 67 68 69
  6A 73 74 75 76 77 78 79 7A 83 84 85 86 87 88 89
  8A 92 93 94 95 96 97 98 99 9A A2 A3 A4 A5 A6 A7
  A8 A9 AA B2 B3 B4 B5 B6 B7 B8 B9 BA C2 C3 C4 C5
  C6 C7 C8 C9 CA D2 D3 D4 D5 D6 D7 D8 D9 DA E1 E2
  E3 E4 E5 E6 E7 E8 E9 EA F1 F2 F3 F4 F5 F6 F7 F8
  F9 FA’
```

For Table K.6 (for chrominance AC coefficients), the 16 bytes which specify the list of code lengths for the table are

X'00 02 01 02 04 04 03 04 07 05 04 04 00 01 02 77'

The set of values which follows this list is:

X'00	01	02	03	11	04	05	21	31	06	12	41	51	07	61	71
13	22	32	81	08	14	42	91	A1	B1	C1	09	23	33	52	F0
15	62	72	D1	0A	16	24	34	E1	25	F1	17	18	19	1A	26
27	28	29	2A	35	36	37	38	39	3A	43	44	45	46	47	48
49	4A	53	54	55	56	57	58	59	5A	63	64	65	66	67	68
69	6A	73	74	75	76	77	78	79	7A	82	83	84	85	86	87
88	89	8A	92	93	94	95	96	97	98	99	9A	A2	A3	A4	A5
A6	A7	A8	A9	AA	B2	B3	B4	B5	B6	B7	B8	B9	BA	C2	C3
C4	C5	C6	C7	C8	C9	CA	D2	D3	D4	D5	D6	D7	D8	D9	DA
E2	E3	E4	E5	E6	E7	E8	E9	EA	F2	F3	F4	F5	F6	F7	F8
F9	FA'														

K.4 Additional information on arithmetic coding

K.4.1 Test sequence for a small data set for the arithmetic coder

The following 256-bit test sequence (in hexadecimal form) is structured to test many of the encoder and decoder paths:

X'00020051 000000C0 0352872A AAAAAAAA 82C02000 FCD79EF6 74EAABF7 697EE74C'

Tables K.7 and K.8 provide a symbol-by-symbol list of the arithmetic encoder and decoder operation. In these tables the event count, EC, is listed first, followed by the value of Qe used in encoding and decoding that event. The decision D to be encoded (and decoded) is listed next. The column labeled MPS contains the sense of the MPS, and if it is followed by a CE (in the "CX" column), the conditional MPS/LPS exchange occurs when encoding and decoding the decision (see Figures D.3, D.4 and D.17). The contents of the A and C registers are the values before the event is encoded and decoded. ST is the number of X'FF' bytes stacked in the encoder waiting for a resolution of the carry-over. Note that the A register is always greater than X'7FFF'. (The starting value has an implied value of X'10000'.)

In the encoder test, the code bytes (B) are listed if they were completed during the coding of the preceding event. If additional bytes follow, they were also completed during the coding of the preceding event. If a byte is listed in the Bx column, the preceding byte in column B was modified by a carry-over.

In the decoder the code bytes are listed if they were placed in the code register just prior to the event EC.

For this file the coded bit count is 240, including the overhead to flush the final data from the C register. When the marker X'FFD9' is appended, a total of 256 bits are output. The actual compressed data sequence for the encoder is (in hexadecimal form)

X'655B5144 F7969D51 7855BFFF 00FC5184 C7CEF939 00287D46 708ECBC0 F6FFD900'

Table K.7 – Encoder test sequence (sheet 1 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
1	0	0		5A1D	0000	00000000	11	0		
2	0	0	CE	5A1D	A5E3	00000000	11	0		
3	0	0		2586	B43A	0000978C	10	0		
4	0	0		2586	8EB4	0000978C	10	0		
5	0	0		1114	D25C	00012F18	9	0		
6	0	0		1114	C148	00012F18	9	0		
7	0	0		1114	B034	00012F18	9	0		
8	0	0		1114	9F20	00012F18	9	0		
9	0	0		1114	8E0C	00012F18	9	0		
10	0	0		080B	F9F0	00025E30	8	0		
11	0	0		080B	F1E5	00025E30	8	0		
12	0	0		080B	E9DA	00025E30	8	0		
13	0	0		080B	E1CF	00025E30	8	0		
14	0	0		080B	D9C4	00025E30	8	0		
15	1	0		080B	D1B9	00025E30	8	0		
16	0	0		17B9	80B0	00327DE0	4	0		
17	0	0		1182	D1EE	0064FBC0	3	0		
18	0	0		1182	C06C	0064FBC0	3	0		
19	0	0		1182	AEEA	0064FBC0	3	0		
20	0	0		1182	9D68	0064FBC0	3	0		
21	0	0		1182	8BE6	0064FBC0	3	0		
22	0	0		0CEF	F4C8	00C9F780	2	0		
23	0	0		0CEF	E7D9	00C9F780	2	0		
24	0	0		0CEF	DAEA	00C9F780	2	0		
25	0	0		0CEF	CDFB	00C9F780	2	0		
26	1	0		0CEF	C10C	00C9F780	2	0		
27	0	0		1518	CEF0	000AB9D0	6	0		65
28	1	0		1518	B9D8	000AB9D0	6	0		
29	0	0		1AA9	A8C0	005AF480	3	0		
30	0	0		1AA9	8E17	005AF480	3	0		
31	0	0		174E	E6DC	00B5E900	2	0		
32	1	0		174E	CF8E	00B5E900	2	0		
33	0	0		1AA9	BA70	00050A00	7	0		5B
34	0	0		1AA9	9FC7	00050A00	7	0		
35	0	0		1AA9	851E	00050A00	7	0		
36	0	0		174E	D4EA	000A1400	6	0		

Table K.7 – Encoder test sequence (sheet 2 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
37	0	0		174E	BD9C	000A1400	6	0		
38	0	0		174E	A64E	000A1400	6	0		
39	0	0		174E	8F00	000A1400	6	0		
40	0	0		1424	EF64	00142800	5	0		
41	0	0		1424	DB40	00142800	5	0		
42	0	0		1424	C71C	00142800	5	0		
43	0	0		1424	B2F8	00142800	5	0		
44	0	0		1424	9ED4	00142800	5	0		
45	0	0		1424	8AB0	00142800	5	0		
46	0	0		119C	ED18	00285000	4	0		
47	0	0		119C	DB7C	00285000	4	0		
48	0	0		119C	C9E0	00285000	4	0		
49	0	0		119C	B844	00285000	4	0		
50	0	0		119C	A6A8	00285000	4	0		
51	0	0		119C	950C	00285000	4	0		
52	0	0		119C	8370	00285000	4	0		
53	0	0		0F6B	E3A8	0050A000	3	0		
54	0	0		0F6B	D43D	0050A000	3	0		
55	0	0		0F6B	C4D2	0050A000	3	0		
56	0	0		0F6B	B567	0050A000	3	0		
57	1	0		0F6B	A5FC	0050A000	3	0		
58	1	0		1424	F6B0	00036910	7	0		51
59	0	0		1AA9	A120	00225CE0	4	0		
60	0	0		1AA9	8677	00225CE0	4	0		
61	0	0		174E	D79C	0044B9C0	3	0		
62	0	0		174E	C04E	0044B9C0	3	0		
63	0	0		174E	A900	0044B9C0	3	0		
64	0	0		174E	91B2	0044B9C0	3	0		
65	0	0		1424	F4C8	00897380	2	0		
66	0	0		1424	E0A4	00897380	2	0		
67	0	0		1424	CC80	00897380	2	0		
68	0	0		1424	B85C	00897380	2	0		
69	0	0		1424	A438	00897380	2	0		
70	0	0		1424	9014	00897380	2	0		
71	1	0		119C	F7E0	0112E700	1	0		
72	1	0		1424	8CE0	001E6A20	6	0		44
73	0	0		1AA9	A120	00F716E0	3	0		

Table K.7 – Encoder test sequence (sheet 3 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
74	1	0		1AA9	8677	00F716E0	3	0		
75	0	0		2516	D548	00041570	8	0		F7
76	1	0		2516	B032	00041570	8	0		
77	0	0		299A	9458	00128230	6	0		
78	0	0		2516	D57C	00250460	5	0		
79	1	0		2516	B066	00250460	5	0		
80	0	0		299A	9458	00963EC0	3	0		
81	1	0		2516	D57C	012C7D80	2	0		
82	0	0		299A	9458	0004B798	8	0		96
83	0	0		2516	D57C	00096F30	7	0		
84	0	0		2516	B066	00096F30	7	0		
85	0	0		2516	8B50	00096F30	7	0		
86	1	0		1EDF	CC74	0012DE60	6	0		
87	1	0		2516	F6F8	009C5FA8	3	0		
88	1	0		299A	9458	0274C628	1	0		
89	0	0		32B4	A668	0004C398	7	0		9D
90	0	0		2E17	E768	00098730	6	0		
91	1	0		2E17	B951	00098730	6	0		
92	0	0		32B4	B85C	002849A8	4	0		
93	1	0		32B4	85A8	002849A8	4	0		
94	0	0		3C3D	CAD0	00A27270	2	0		
95	1	0		3C3D	8E93	00A27270	2	0		
96	0	0		415E	F0F4	00031318	8	0		51
97	1	0		415E	AF96	00031318	8	0		
98	0	0	CE	4639	82BC	000702A0	7	0		
99	1	0		415E	8C72	000E7E46	6	0		
100	0	0	CE	4639	82BC	001D92B4	5	0		
101	1	0		415E	8C72	003B9E6E	4	0		
102	0	0	CE	4639	82BC	0077D304	3	0		
103	1	0		415E	8C72	00F01F0E	2	0		
104	0	0	CE	4639	82BC	01E0D444	1	0		
105	1	0		415E	8C72	0002218E	8	0		78
106	0	0	CE	4639	82BC	0004D944	7	0		
107	1	0		415E	8C72	000A2B8E	6	0		
108	0	0	CE	4639	82BC	0014ED44	5	0		
109	1	0		415E	8C72	002A538E	4	0		
110	0	0	CE	4639	82BC	00553D44	3	0		

Table K.7 – Encoder test sequence (sheet 4 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
111	1	0		415E	8C72	00AAF38E	2	0		
112	0	0	CE	4639	82BC	01567D44	1	0		
113	1	0		415E	8C72	0005738E	8	0		55
114	0	0	CE	4639	82BC	000B7D44	7	0		
115	1	0		415E	8C72	0017738E	6	0		
116	0	0	CE	4639	82BC	002F7D44	5	0		
117	1	0		415E	8C72	005F738E	4	0		
118	0	0	CE	4639	82BC	00BF7D44	3	0		
119	1	0		415E	8C72	017F738E	2	0		
120	0	0	CE	4639	82BC	02FF7D44	1	0		
121	1	0		415E	8C72	0007738E	8	0		BF
122	0	0	CE	4639	82BC	000F7D44	7	0		
123	1	0		415E	8C72	001F738E	6	0		
124	0	0	CE	4639	82BC	003F7D44	5	0		
125	1	0		415E	8C72	007F738E	4	0		
126	0	0	CE	4639	82BC	00FF7D44	3	0		
127	1	0		415E	8C72	01FF738E	2	0		
128	0	0	CE	4639	82BC	03FF7D44	1	0		
129	1	0		415E	8C72	0007738E	8	1		
130	0	0	CE	4639	82BC	000F7D44	7	1		
131	0	0		415E	8C72	001F738E	6	1		
132	0	0		3C3D	9628	003EE71C	5	1		
133	0	0		375E	B3D6	007DCE38	4	1		
134	0	0		32B4	F8F0	00FB9C70	3	1		
135	1	0		32B4	C63C	00FB9C70	3	1		
136	0	0		3C3D	CAD0	03F0BFE0	1	1		
137	1	0		3C3D	8E93	03F0BFE0	1	1		
138	1	0		415E	F0F4	000448D8	7	0		FF00FC
139	0	0	CE	4639	82BC	0009F0DC	6	0		
140	0	0		415E	8C72	00145ABE	5	0		
141	0	0		3C3D	9628	0028B57C	4	0		
142	0	0		375E	B3D6	00516AF8	3	0		
143	0	0		32B4	F8F0	00A2D5F0	2	0		
144	0	0		32B4	C63C	00A2D5F0	2	0		
145	0	0		32B4	9388	00A2D5F0	2	0		
146	0	0		2E17	C1A8	0145ABE0	1	0		

Table K.7 – Encoder test sequence (sheet 5 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
147	1	0		2E17	9391	0145ABE0	1	0		
148	0	0		32B4	B85C	00084568	7	0		51
149	0	0		32B4	85A8	00084568	7	0		
150	0	0		2E17	A5E8	00108AD0	6	0		
151	0	0		299A	EFA2	002115A0	5	0		
152	0	0		299A	C608	002115A0	5	0		
153	0	0		299A	9C6E	002115A0	5	0		
154	0	0		2516	E5A8	00422B40	4	0		
155	0	0		2516	C092	00422B40	4	0		
156	0	0		2516	9B7C	00422B40	4	0		
157	0	0		1EDF	ECCC	00845680	3	0		
158	0	0		1EDF	CDED	00845680	3	0		
159	0	0		1EDF	AF0E	00845680	3	0		
160	0	0		1EDF	902F	00845680	3	0		
161	1	0		1AA9	E2A0	0108AD00	2	0		
162	1	0		2516	D548	000BA7B8	7	0		84
163	1	0		299A	9458	00315FA8	5	0		
164	1	0		32B4	A668	00C72998	3	0		
165	1	0		3C3D	CAD0	031E7530	1	0		
166	1	0		415E	F0F4	000C0F0C	7	0		C7
167	0	0	CE	4639	82BC	00197D44	6	0		
168	0	0		415E	8C72	0033738E	5	0		
169	1	0		3C3D	9628	0066E71C	4	0		
170	1	0		415E	F0F4	019D041C	2	0		
171	0	0	CE	4639	82BC	033B6764	1	0		
172	1	0		415E	8C72	000747CE	8	0		CE
173	0	0	CE	4639	82BC	000F25C4	7	0		
174	1	0		415E	8C72	001EC48E	6	0		
175	1	0	CE	4639	82BC	003E1F44	5	0		
176	1	0		4B85	F20C	00F87D10	3	0		
177	1	0	CE	504F	970A	01F2472E	2	0		
178	0	0	CE	5522	8D76	03E48E5C	1	0		
179	0	0		504F	AA44	00018D60	8	0		F9
180	1	0		4B85	B3EA	00031AC0	7	0		
181	1	0	CE	504F	970A	0007064A	6	0		
182	1	0	CE	5522	8D76	000E0C94	5	0		
183	1	0		59EB	E150	00383250	3	0		

Table K.7 – Encoder test sequence (sheet 6 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
184	0	1		59EB	B3D6	0071736A	2	0		
185	1	0		59EB	B3D6	00E39AAA	1	0		
186	1	1		59EB	B3D6	0007E92A	8	0		38
187	1	1		5522	B3D6	000FD254	7	0		
188	1	1		504F	BD68	001FA4A8	6	0		
189	0	1		4B85	DA32	003F4950	5	0		
190	1	1	CE	504F	970A	007FAFFA	4	0		
191	1	1		4B85	A09E	00FFED6A	3	0		
192	0	1		4639	AA32	01FFDAD4	2	0		
193	0	1	CE	4B85	8C72	04007D9A	1	0		
194	1	1	CE	504F	81DA	0000FB34	8	0	39	00
195	1	1		4B85	A09E	0002597E	7	0		
196	1	1		4639	AA32	0004B2FC	6	0		
197	0	1		415E	C7F2	000965F8	5	0		
198	1	1	CE	4639	82BC	0013D918	4	0		
199	0	1		415E	8C72	00282B36	3	0		
200	0	1	CE	4639	82BC	0050EC94	2	0		
201	1	1		4B85	F20C	0003B250	8	0		28
202	1	1		4B85	A687	0003B250	8	0		
203	1	1		4639	B604	000764A0	7	0		
204	0	1		415E	DF96	000EC940	6	0		
205	1	1	CE	4639	82BC	001ECEF0	5	0		
206	0	1		415E	8C72	003E16E6	4	0		
207	1	1	CE	4639	82BC	007CC3F4	3	0		
208	0	1		415E	8C72	00FA00EE	2	0		
209	1	1	CE	4639	82BC	01F49804	1	0		
210	0	1		415E	8C72	0001A90E	8	0		7D
211	1	1	CE	4639	82BC	0003E844	7	0		
212	0	1		415E	8C72	0008498E	6	0		
213	1	1	CE	4639	82BC	00112944	5	0		
214	0	1		415E	8C72	0022CB8E	4	0		
215	1	1	CE	4639	82BC	00462D44	3	0		
216	1	1		415E	8C72	008CD38E	2	0		
217	1	1		3C3D	9628	0119A71C	1	0		
218	1	1		375E	B3D6	00034E38	8	0		46
219	1	1		32B4	F8F0	00069C70	7	0		
220	1	1		32B4	C63C	00069C70	7	0		

Table K.7 – Encoder test sequence (sheet 7 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	ST	Bx	B
221	0	1		32B4	9388	00069C70	7	0		
222	1	1		3C3D	CAD0	001BF510	5	0		
223	1	1		3C3D	8E93	001BF510	5	0		
224	1	1		375E	A4AC	0037EA20	4	0		
225	0	1		32B4	DA9C	006FD440	3	0		
226	1	1		3C3D	CAD0	01C1F0A0	1	0		
227	1	1		3C3D	8E93	01C1F0A0	1	0		
228	0	1		375E	A4AC	0003E140	8	0		70
229	1	1		3C3D	DD78	00113A38	6	0		
230	0	1		3C3D	A13B	00113A38	6	0		
231	0	1		415E	F0F4	00467CD8	4	0		
232	1	1	CE	4639	82BC	008E58DC	3	0		
233	0	1		415E	8C72	011D2ABE	2	0		
234	1	1	CE	4639	82BC	023AEBA4	1	0		
235	1	1		415E	8C72	0006504E	8	0		8E
236	1	1		3C3D	9628	000CA09C	7	0		
237	1	1		375E	B3D6	00194138	6	0		
238	1	1		32B4	F8F0	00328270	5	0		
239	1	1		32B4	C63C	00328270	5	0		
240	0	1		32B4	9388	00328270	5	0		
241	1	1		3C3D	CAD0	00CB8D10	3	0		
242	1	1		3C3D	8E93	00CB8D10	3	0		
243	1	1		375E	A4AC	01971A20	2	0		
244	0	1		32B4	DA9C	032E3440	1	0		
245	0	1		3C3D	CAD0	000B70A0	7	0		CB
246	1	1		415E	F0F4	002FFCCC	5	0		
247	1	1		415E	AF96	002FFCCC	5	0		
248	1	1		3C3D	DC70	005FF998	4	0		
249	0	1		3C3D	A033	005FF998	4	0		
250	1	1		415E	F0F4	01817638	2	0		
251	0	1		415E	AF96	01817638	2	0		
252	0	1	CE	4639	82BC	0303C8E0	1	0		
253	1	1		4B85	F20C	000F2380	7	0		C0
254	1	1		4B85	A687	000F2380	7	0		
255	0	1		4639	B604	001E4700	6	0		
256	0	1	CE	4B85	8C72	003D6D96	5	0		
Flush:					81DA	007ADB2C	4	0		F6
										FFD9

Table K.8 – Decoder test sequence (sheet 1 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
1	0	0		5A1D	0000	655B0000	0	65 5B
2	0	0	CE	5A1D	A5E3	655B0000	0	
3	0	0		2586	B43A	332AA200	7	51
4	0	0		2586	8EB4	332AA200	7	
5	0	0		1114	D25C	66554400	6	
6	0	0		1114	C148	66554400	6	
7	0	0		1114	B034	66554400	6	
8	0	0		1114	9F20	66554400	6	
9	0	0		1114	8E0C	66554400	6	
10	0	0		080B	F9F0	CCAA8800	5	
11	0	0		080B	F1E5	CCAA8800	5	
12	0	0		080B	E9DA	CCAA8800	5	
13	0	0		080B	E1CF	CCAA8800	5	
14	0	0		080B	D9C4	CCAA8800	5	
15	1	0		080B	D1B9	CCAA8800	5	
16	0	0		17B9	80B0	2FC88000	1	
17	0	0		1182	D1EE	5F910000	0	
18	0	0		1182	C06C	5F910000	0	
19	0	0		1182	AEEA	5F910000	0	
20	0	0		1182	9D68	5F910000	0	
21	0	0		1182	8BE6	5F910000	0	
22	0	0		0CEF	F4C8	BF228800	7	44
23	0	0		0CEF	E7D9	BF228800	7	
24	0	0		0CEF	DAEA	BF228800	7	
25	0	0		0CEF	CDFB	BF228800	7	
26	1	0		0CEF	C10C	BF228800	7	
27	0	0		1518	CEF0	B0588000	3	
28	1	0		1518	B9D8	B0588000	3	
29	0	0		1AA9	A8C0	5CC40000	0	
30	0	0		1AA9	8E17	5CC40000	0	
31	0	0		174E	E6DC	B989EE00	7	F7
32	1	0		174E	CF8E	B989EE00	7	
33	0	0		1AA9	BA70	0A4F7000	4	
34	0	0		1AA9	9FC7	0A4F7000	4	
35	0	0		1AA9	851E	0A4F7000	4	
36	0	0		174E	D4EA	149EE000	3	
37	0	0		174E	BD9C	149EE000	3	
38	0	0		174E	A64E	149EE000	3	
39	0	0		174E	8F00	149EE000	3	
40	0	0		1424	EF64	293DC000	2	

Table K.8 – Decoder test sequence (sheet 2 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
41	0	0		1424	DB40	293DC000	2	
42	0	0		1424	C71C	293DC000	2	
43	0	0		1424	B2F8	293DC000	2	
44	0	0		1424	9ED4	293DC000	2	
45	0	0		1424	8AB0	293DC000	2	
46	0	0		119C	ED18	527B8000	1	
47	0	0		119C	DB7C	527B8000	1	
48	0	0		119C	C9E0	527B8000	1	
49	0	0		119C	B844	527B8000	1	
50	0	0		119C	A6A8	527B8000	1	
51	0	0		119C	950C	527B8000	1	
52	0	0		119C	8370	527B8000	1	
53	0	0		0F6B	E3A8	A4F70000	0	
54	0	0		0F6B	D43D	A4F70000	0	
55	0	0		0F6B	C4D2	A4F70000	0	
56	0	0		0F6B	B567	A4F70000	0	
57	1	0		0F6B	A5FC	A4F70000	0	
58	1	0		1424	F6B0	E6696000	4	96
59	0	0		1AA9	A120	1EEB0000	1	
60	0	0		1AA9	8677	1EEB0000	1	
61	0	0		174E	D79C	3DD60000	0	
62	0	0		174E	C04E	3DD60000	0	
63	0	0		174E	A900	3DD60000	0	
64	0	0		174E	91B2	3DD60000	0	
65	0	0		1424	F4C8	7BAD3A00	7	9D
66	0	0		1424	E0A4	7BAD3A00	7	
67	0	0		1424	CC80	7BAD3A00	7	
68	0	0		1424	B85C	7BAD3A00	7	
69	0	0		1424	A438	7BAD3A00	7	
70	0	0		1424	9014	7BAD3A00	7	
71	1	0		119C	F7E0	F75A7400	6	
72	1	0		1424	8CE0	88B3A000	3	
73	0	0		1AA9	A120	7FBD0000	0	
74	1	0		1AA9	8677	7FBD0000	0	
75	0	0		2516	D548	9F7A8800	5	51
76	1	0		2516	B032	9F7A8800	5	
77	0	0		299A	9458	517A2000	3	
78	0	0		2516	D57C	A2F44000	2	
79	1	0		2516	B066	A2F44000	2	
80	0	0		299A	9458	5E910000	0	

Table K.8 – Decoder test sequence (sheet 3 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
81	1	0		2516	D57C	BD22F000	7	78
82	0	0		299A	9458	32F3C000	5	
83	0	0		2516	D57C	65E78000	4	
84	0	0		2516	B066	65E78000	4	
85	0	0		2516	8B50	65E78000	4	
86	1	0		1EDF	CC74	CBCF0000	3	
87	1	0		2516	F6F8	F1D00000	0	
88	1	0		299A	9458	7FB95400	6	55
89	0	0		32B4	A668	53ED5000	4	
90	0	0		2E17	E768	A7DAA000	3	
91	1	0		2E17	B951	A7DAA000	3	
92	0	0		32B4	B85C	72828000	1	
93	1	0		32B4	85A8	72828000	1	
94	0	0		3C3D	CAD0	7E3B7E00	7	BF
95	1	0		3C3D	8E93	7E3B7E00	7	
96	0	0		415E	F0F4	AF95F800	5	
97	1	0		415E	AF96	AF95F800	5	
98	0	0	CE	4639	82BC	82BBF000	4	
99	1	0		415E	8C72	8C71E000	3	
100	0	0	CE	4639	82BC	82BBC000	2	
101	1	0		415E	8C72	8C718000	1	
102	0	0	CE	4639	82BC	82BB0000	0	
103	1	0		415E	8C72	8C71FE00	7	FF 00
104	0	0	CE	4639	82BC	82BBFC00	6	
105	1	0		415E	8C72	8C71F800	5	
106	0	0	CE	4639	82BC	82BBF000	4	
107	1	0		415E	8C72	8C71E000	3	
108	0	0	CE	4639	82BC	82BBC000	2	
109	1	0		415E	8C72	8C718000	1	
110	0	0	CE	4639	82BC	82BB0000	0	
111	1	0		415E	8C72	8C71F800	7	FC
112	0	0	CE	4639	82BC	82BBF000	6	
113	1	0		415E	8C72	8C71E000	5	
114	0	0	CE	4639	82BC	82BBC000	4	
115	1	0		415E	8C72	8C718000	3	
116	0	0	CE	4639	82BC	82BB0000	2	
117	1	0		415E	8C72	8C700000	1	
118	0	0	CE	4639	82BC	82B80000	0	
119	1	0		415E	8C72	8C6AA200	7	51
120	0	0	CE	4639	82BC	82AD4400	6	

Table K.8 – Decoder test sequence (sheet 4 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
121	1	0		415E	8C72	8C548800	5	
122	0	0	CE	4639	82BC	82811000	4	
123	1	0		415E	8C72	8BFC2000	3	
124	0	0	CE	4639	82BC	81D04000	2	
125	1	0		415E	8C72	8A9A8000	1	
126	0	0	CE	4639	82BC	7F0D0000	0	
127	1	0		415E	8C72	85150800	7	84
128	0	0	CE	4639	82BC	74021000	6	
129	1	0		415E	8C72	6EFE2000	5	
130	0	0	CE	4639	82BC	47D44000	4	
131	0	0		415E	8C72	16A28000	3	
132	0	0		3C3D	9628	2D450000	2	
133	0	0		375E	B3D6	5A8A0000	1	
134	0	0		32B4	F8F0	B5140000	0	
135	1	0		32B4	C63C	B5140000	0	
136	0	0		3C3D	CAD0	86331C00	6	C7
137	1	0		3C3D	8E93	86331C00	6	
138	1	0		415E	F0F4	CF747000	4	
139	0	0	CE	4639	82BC	3FBCE000	3	
140	0	0		415E	8C72	0673C000	2	
141	0	0		3C3D	9628	0CE78000	1	
142	0	0		375E	B3D6	19CF0000	0	
143	0	0		32B4	F8F0	339F9C00	7	CE
144	0	0		32B4	C63C	339F9C00	7	
145	0	0		32B4	9388	339F9C00	7	
146	0	0		2E17	C1A8	673F3800	6	
147	1	0		2E17	9391	673F3800	6	
148	0	0		32B4	B85C	0714E000	4	
149	0	0		32B4	85A8	0714E000	4	
150	0	0		2E17	A5E8	0E29C000	3	
151	0	0		299A	EFA2	1C538000	2	
152	0	0		299A	C608	1C538000	2	
153	0	0		299A	9C6E	1C538000	2	
154	0	0		2516	E5A8	38A70000	1	
155	0	0		2516	C092	38A70000	1	
156	0	0		2516	9B7C	38A70000	1	
157	0	0		1EDF	ECCC	714E0000	0	
158	0	0		1EDF	CDED	714E0000	0	
159	0	0		1EDF	AF0E	714E0000	0	
160	0	0		1EDF	902F	714E0000	0	

Table K.8 – Decoder test sequence (sheet 5 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
161	1	0		1AA9	E2A0	E29DF200	7	F9
162	1	0		2516	D548	D5379000	4	
163	1	0		299A	9458	94164000	2	
164	1	0		32B4	A668	A5610000	0	
165	1	0		3C3D	CAD0	C6B4E400	6	39
166	1	0		415E	F0F4	E0879000	4	
167	0	0	CE	4639	82BC	61E32000	3	
168	0	0		415E	8C72	4AC04000	2	
169	1	0		3C3D	9628	95808000	1	
170	1	0		415E	F0F4	EE560000	7	00
171	0	0	CE	4639	82BC	7D800000	6	
172	1	0		415E	8C72	81FA0000	5	
173	0	0	CE	4639	82BC	6DCC0000	4	
174	1	0		415E	8C72	62920000	3	
175	1	0	CE	4639	82BC	2EFC0000	2	
176	1	0		4B85	F20C	BBF00000	0	
177	1	0	CE	504F	970A	2AD25000	7	28
178	0	0	CE	5522	8D76	55A4A000	6	
179	0	0		504F	AA44	3AA14000	5	
180	1	0		4B85	B3EA	75428000	4	
181	1	0	CE	504F	970A	19BB0000	3	
182	1	0	CE	5522	8D76	33760000	2	
183	1	0		59EB	E150	CDD80000	0	
184	0	1		59EB	B3D6	8CE6FA00	7	7D
185	1	0		59EB	B3D6	65F7F400	6	
186	1	1		59EB	B3D6	1819E800	5	
187	1	1		5522	B3D6	3033D000	4	
188	1	1		504F	BD68	6067A000	3	
189	0	1		4B85	DA32	C0CF4000	2	
190	1	1	CE	504F	970A	64448000	1	
191	1	1		4B85	A09E	3B130000	0	
192	0	1		4639	AA32	76268C00	7	46
193	0	1	CE	4B85	8C72	245B1800	6	
194	1	1	CE	504F	81DA	48B63000	5	
195	1	1		4B85	A09E	2E566000	4	
196	1	1		4639	AA32	5CACC000	3	
197	0	1		415E	C7F2	B9598000	2	
198	1	1	CE	4639	82BC	658B0000	1	
199	0	1		415E	8C72	52100000	0	
200	0	1	CE	4639	82BC	0DF8E000	7	70

Table K.8 – Decoder test sequence (sheet 6 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
201	1	1		4B85	F20C	37E38000	5	
202	1	1		4B85	A687	37E38000	5	
203	1	1		4639	B604	6FC70000	4	
204	0	1		415E	DF96	DF8E0000	3	
205	1	1	CE	4639	82BC	82AC0000	2	
206	0	1		415E	8C72	8C520000	1	
207	1	1	CE	4639	82BC	827C0000	0	
208	0	1		415E	8C72	8BF31C00	7	8E
209	1	1	CE	4639	82BC	81BE3800	6	
210	0	1		415E	8C72	8A767000	5	
211	1	1	CE	4639	82BC	7EC4E000	4	
212	0	1		415E	8C72	8483C000	3	
213	1	1	CE	4639	82BC	72DF8000	2	
214	0	1		415E	8C72	6CB90000	1	
215	1	1	CE	4639	82BC	434A0000	0	
216	1	1		415E	8C72	0D8F9600	7	CB
217	1	1		3C3D	9628	1B1F2C00	6	
218	1	1		375E	B3D6	363E5800	5	
219	1	1		32B4	F8F0	6C7CB000	4	
220	1	1		32B4	C63C	6C7CB000	4	
221	0	1		32B4	9388	6C7CB000	4	
222	1	1		3C3D	CAD0	2EA2C000	2	
223	1	1		3C3D	8E93	2EA2C000	2	
224	1	1		375E	A4AC	5D458000	1	
225	0	1		32B4	DA9C	BA8B0000	0	
226	1	1		3C3D	CAD0	4A8F0000	6	C0
227	1	1		3C3D	8E93	4A8F0000	6	
228	0	1		375E	A4AC	951E0000	5	
229	1	1		3C3D	DD78	9F400000	3	
230	0	1		3C3D	A13B	9F400000	3	
231	0	1		415E	F0F4	E9080000	1	
232	1	1	CE	4639	82BC	72E40000	0	
233	0	1		415E	8C72	6CC3EC00	7	F6
234	1	1	CE	4639	82BC	435FD800	6	
235	1	1		415E	8C72	0DB9B000	5	
236	1	1		3C3D	9628	1B736000	4	
237	1	1		375E	B3D6	36E6C000	3	
238	1	1		32B4	F8F0	6DCD8000	2	
239	1	1		32B4	C63C	6DCD8000	2	
240	0	1		32B4	9388	6DCD8000	2	

Table K.8 – Decoder test sequence (sheet 7 of 7)

EC	D	MPS	CX	Qe (hexadecimal)	A (hexadecimal)	C (hexadecimal)	CT	B
241	1	1		3C3D	CAD0	33E60000	0	
242	1	1		3C3D	8E93	33E60000	0	
Marker detected: zero byte fed to decoder								
243	1	1		375E	A4AC	67CC0000	7	
244	0	1		32B4	DA9C	CF980000	6	
245	0	1		3C3D	CAD0	9EC00000	4	
246	1	1		415E	F0F4	40B40000	2	
247	1	1		415E	AF96	40B40000	2	
248	1	1		3C3D	DC70	81680000	1	
249	0	1		3C3D	A033	81680000	1	
Marker detected: zero byte fed to decoder								
250	1	1		415E	F0F4	75C80000	7	
251	0	1		415E	AF96	75C80000	7	
252	0	1	CE	4639	82BC	0F200000	6	
253	1	1		4B85	F20C	3C800000	4	
254	1	1		4B85	A687	3C800000	4	
255	0	1		4639	B604	79000000	3	
256	0	1	CE	4B85	8C72	126A0000	2	

K.5 Low-pass downsampling filters for hierarchical coding

In this section simple examples are given of downsampling filters which are compatible with the upsampling filter defined in J.1.1.2.

Figure K.5 shows the weighting of neighbouring samples for simple one-dimensional horizontal and vertical low-pass filters. The output of the filter must be normalized by the sum of the neighbourhood weights.

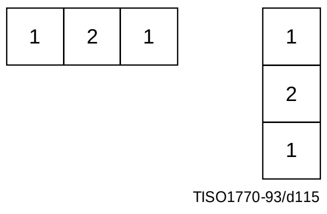


Figure K.5 – Low-pass filter example

The centre sample in Figure K.5 should be aligned with the left column or top line of the high resolution image when calculating the left column or top line of the low resolution image. Sample values which are situated outside of the image boundary are replicated from the sample values at the boundary to provide missing edge values.

If the image being downsampled has an odd width or length, the odd dimension is increased by 1 by sample replication on the right edge or bottom line before downsampling.

K.6 Domain of applicability of DCT and spatial coding techniques

The DCT coder is intended for lossy coding in a range from quite visible loss to distortion well below the threshold for visibility. However in general, DCT-based processes cannot be used for true lossless coding.

The lossless coder is intended for completely lossless coding. The lossless coding process is significantly less effective than the DCT-based processes for distortions near and above the threshold of visibility.

The point transform of the input to the lossless coder permits a very restricted form of lossy coding with the "lossless" coder. (The coder is still lossless after the input point transform.) Since the DCT is intended for lossy coding, there may be some confusion about when this alternative lossy technique should be used.

Lossless coding with a point transformed input is intended for applications which cannot be addressed by DCT coding techniques. Among these are

- true lossless coding to a specified precision;
- lossy coding with precisely defined error bounds;
- hierarchical progression to a truly lossless final stage.

If lossless coding with a point transformed input is used in applications which can be met effectively by DCT coding, the results will be significantly less satisfactory. For example, distortion in the form of visible contours usually appears when precision of the luminance component is reduced to about six bits. For normal image data, this occurs at bit rates well above those for which the DCT gives outputs which are visually indistinguishable from the source.

K.7 Domain of applicability of the progressive coding modes of operation

Two very different progressive coding modes of operation have been defined, progressive coding of the DCT coefficients and hierarchical progression. Progressive coding of the DCT coefficients has two complementary procedures, spectral selection and successive approximation. Because of this diversity of choices, there may be some confusion as to which method of progression to use for a given application.

K.7.1 Progressive coding of the DCT

In progressive coding of the DCT coefficients two complementary procedures are defined for decomposing the 8×8 DCT coefficient array, spectral selection and successive approximation. Spectral selection partitions zig-zag array of DCT coefficients into "bands", one band being coded in each scan. Successive approximation codes the coefficients with reduced precision in the first scan; in each subsequent scan the precision is increased by one bit.

A single forward DCT is calculated for these procedures. When all coefficients are coded to full precision, the DCT is the same as in the sequential mode. Therefore, like the sequential DCT coding, progressive coding of DCT coefficients is intended for applications which need very good compression for a given level of visual distortion.

The simplest progressive coding technique is spectral selection; indeed, because of this simplicity, some applications may choose – despite the limited progression that can be achieved – to use only spectral selection. Note, however, that the absence of high frequency bands typically leads – for a given bit rate – to a significantly lower image quality in the intermediate stages than can be achieved with the more general progressions. The net coding efficiency at the completion of the final stage is typically comparable to or slightly less than that achieved with the sequential DCT.

A much more flexible progressive system is attained at some increase in complexity when successive approximation is added to the spectral selection progression. For a given bit rate, this system typically provides significantly better image quality than spectral selection alone. The net coding efficiency at the completion of the final stage is typically comparable to or slightly better than that achieved with the sequential DCT.

K.7.2 Hierarchical progression

Hierarchical progression permits a sequence of outputs of increasing spatial resolution, and also allows refinement of image quality at a given spatial resolution. Both DCT and spatial versions of the hierarchical progression are allowed, and progressive coding of DCT coefficients may be used in a frame of the DCT hierarchical progression.

The DCT hierarchical progression is intended for applications which need very good compression for a given level of visual distortion; the spatial hierarchical progression is intended for applications which need a simple progression with a truly lossless final stage. Figure K.6 illustrates examples of these two basic hierarchical progressions.

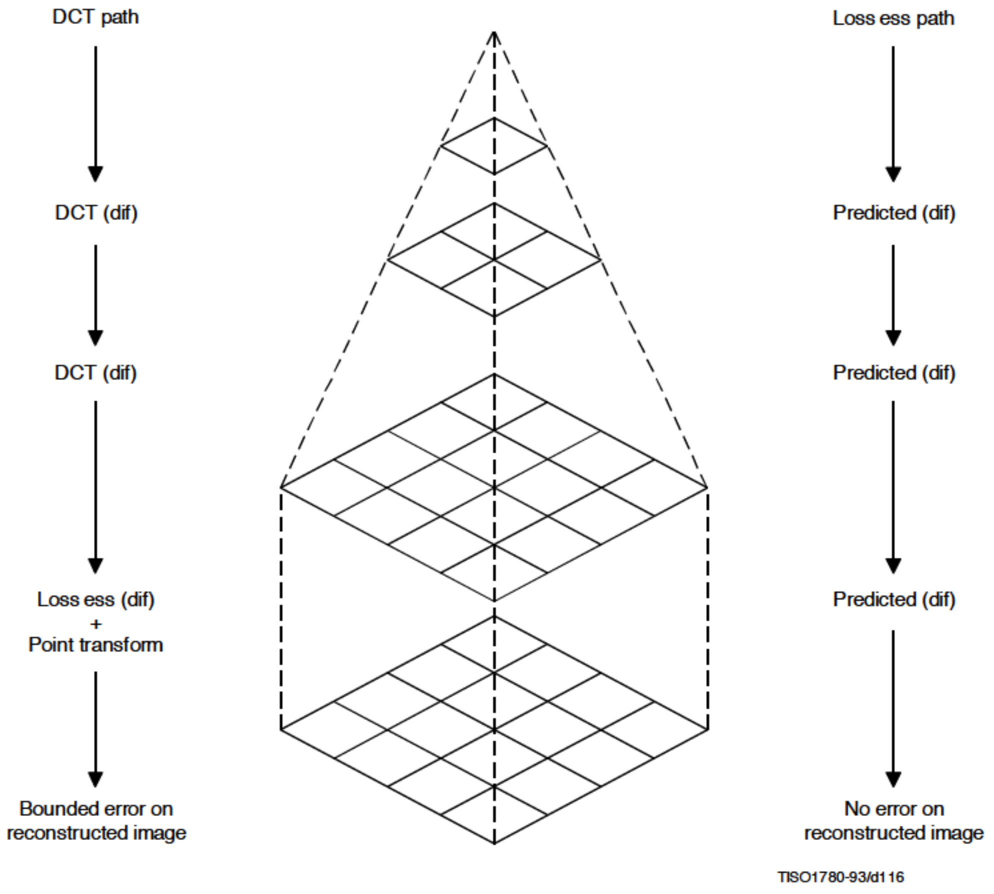


Figure K.6 – Sketch of the basic operations of the hierarchical mode

K.7.2.1 DCT Hierarchical progression

If a DCT hierarchical progression uses reduced spatial resolution, the early stages of the progression can have better image quality for a given bit rate than the early stages of non-hierarchical progressive coding of the DCT coefficients. However, at the point where the distortion between source and output becomes indistinguishable, the coding efficiency achieved with a DCT hierarchical progression is typically significantly lower than the coding efficiency achieved with a non-hierarchical progressive coding of the DCT coefficients.

While the hierarchical DCT progression is intended for lossy progressive coding, a final spatial differential coding stage can be used. When this final stage is used, the output can be almost lossless, limited only by the difference between the encoder and decoder IDCT implementations. Since IDCT implementations can differ significantly, truly lossless coding after a DCT hierarchical progression cannot be guaranteed. An important alternative, therefore, is to use the input point transform of the final lossless differential coding stage to reduce the precision of the differential input. This allows a bounding of the difference between source and output at a significantly lower cost in coded bits than coding of the full precision spatial difference would require.

K.7.2.2 Spatial hierarchical progression

If lossless progression is required, a very simple hierarchical progression may be used in which the spatial lossless coder with point transformed input is used as a first stage. This first stage is followed by one or more spatial differential coding stages. The first stage should be nearly lossless, such that the low order bits which are truncated by the point transform are essentially random – otherwise the compression efficiency will be degraded relative to non-progressive lossless coding.

K.8 Suppression of block-to-block discontinuities in decoded images

A simple technique is available for suppressing the block-to-block discontinuities which can occur in images compressed by DCT techniques.

The first few (five in this example) low frequency DCT coefficients are predicted from the nine DC values of the block and the eight nearest-neighbour blocks, and the predicted values are used to suppress blocking artifacts in smooth areas of the image.

The prediction equations for the first five AC coefficients in the zig-zag sequence are obtained as follows:

K.8.1 AC prediction

The sample field in a 3 by 3 array of blocks (each block containing an 8×8 array of samples) is modeled by a two-dimensional second degree polynomial of the form:

$$P(x,y) = A1(x^2y^2) + A2(x^2y) + A3(xy^2) + A4(x^2) + A5(xy) + A6(y^2) + A7(x) + A8(y) + A9$$

The nine coefficients A1 through A9 are uniquely determined by imposing the constraint that the mean of P(x,y) over each of the nine blocks must yield the correct DC-values.

Applying the DCT to the quadratic field predicting the samples in the central block gives a prediction of the low frequency AC coefficients depicted in Figure K.7.

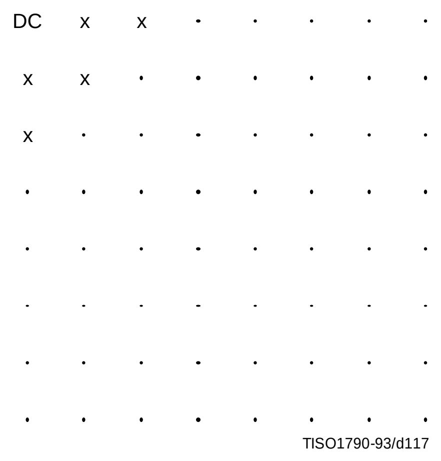


Figure K.7 – DCT array positions of predicted AC coefficients

The prediction equations derived in this manner are as follows:

For the two dimensional array of DC values shown

$$\begin{array}{lll} DC_1 & DC_2 & DC_3 \\ DC_4 & DC_5 & DC_6 \\ DC_7 & DC_8 & DC_9 \end{array}$$

The unquantized prediction equations are

$$AC_{01} = 1{,}13885\ (DC_4 - DC_6)$$
$$AC_{10} = 1{,}13885\ (DC_2 - DC_8)$$
$$AC_{20} = 0{,}27881\ (DC_2 + DC_8 - 2 \times DC_5)$$
$$AC_{11} = 0{,}16213\ ((DC_1 - DC_3) - (DC_7 - DC_9))$$
$$AC_{02} = 0{,}27881\ (DC_4 + DC_6 - 2 \times DC_5)$$

The scaling of the predicted AC coefficients is consistent with the DCT normalization defined in A.3.3.

K.8.2 Quantized AC prediction

The prediction equations can be mapped to a form which uses quantized values of the DC coefficients and which computes quantized AC coefficients using integer arithmetic. The quantized DC coefficients need to be scaled, however, such that the predicted coefficients have fractional bit precision.

First, the prediction equation coefficients are scaled by 32 and rounded to the nearest integer. Thus,

$$1{,}13885 \times 32 = 36$$

$$0{,}27881 \times 32 = \ \ 9$$

$$0{,}16213 \times 32 = \ \ 5$$

The multiplicative factors are then scaled by the ratio of the DC and AC quantization factors and rounded appropriately. The normalization defined for the DCT introduces another factor of 8 in the unquantized DC values. Therefore, in terms of the quantized DC values, the predicted quantized AC coefficients are given by the equations below. Note that if (for example) the DC values are scaled by a factor of 4, the AC predictions will have 2 fractional bits of precision relative to the quantized DCT coefficients.

$$QAC_{01} = (\ (R_d \times Q_{01}) + (36 \times Q_{00} \times (QDC_4 - QDC_6)))/(256 \times Q_{01})$$
$$QAC_{10} = (\ (R_d \times Q_{10}) + (36 \times Q_{00} \times (QDC_2 - QDC_8)))/(256 \times Q_{10})$$
$$QAC_{20} = (\ (R_d \times Q_{20}) + (\ 9 \times Q_{00} \times (QDC_2 + QDC_8 - 2 \times QDC_5)))/(256 \times Q_{20})$$
$$QAC_{11} = (\ (R_d \times Q_{11}) + (\ 5 \times Q_{00} \times ((QDC_1 - QDC_3) - (QDC_7 - QDC_9))))/(256 \times Q_{11})$$
$$QAC_{02} = (\ (R_d \times Q_{02}) + (\ 9 \times Q_{00} \times (QDC_4 + QDC_6 - 2 \times QDC_5)))/(256 \times Q_{02})$$

where QDC_x and QAC_{xy} are the quantized and scaled DC and AC coefficient values. The constant Rd is added to get a correct rounding in the division. Rd is 128 for positive numerators, and –128 for negative numerators.

Predicted values should not override coded values. Therefore, predicted values for coefficients which are already non-zero should be set to zero. Predictions should be clamped if they exceed a value which would be quantized to a non-zero value for the current precision in the successive approximation.

K.9 Modification of dequantization to improve displayed image quality

For a progression where the first stage successive approximation bit, Al, is set to 3, uniform quantization of the DCT gives the following quantization and dequantization levels for a sequence of successive approximation scans, as shown in Figure K.8:

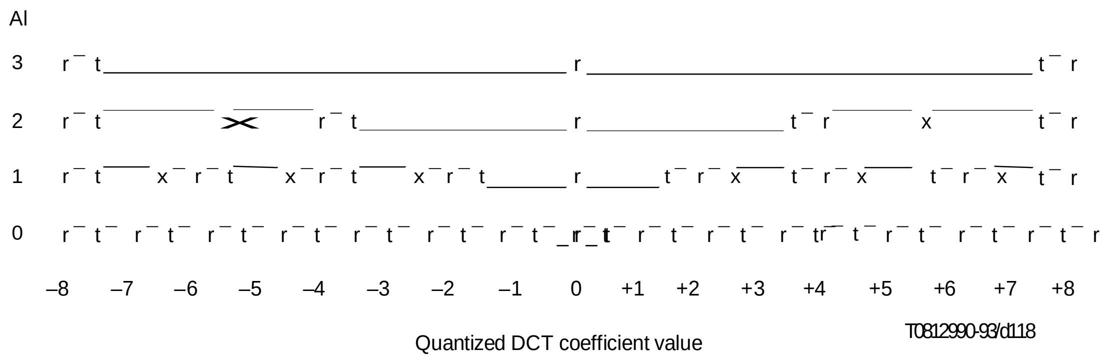


Figure K.8 – Illustration of two reconstruction strategies

The column to the left labelled "Al" gives the bit position specified in the scan header. The quantized DCT coefficient magnitudes are therefore divided by 2^{Al} during that scan.

Referring to the final scan (Al = 0), the points marked with “t” are the threshold values, while the points marked with “r” are the reconstruction values. The unquantized output is obtained by multiplying the horizontal scale in Figure K.8 by the quantization value.

The quantization interval for a coefficient value of zero is indicated by the depressed interval of the line. As the bit position Al is increased, a “fat zero” quantization interval develops around the zero DCT coefficient value. In the limit where the scaling factor is very large, the zero interval is twice as large as the rest of the quantization intervals.

Two different reconstruction strategies are shown. The points marked “r” are the reconstruction obtained using the normal rounding rules for the DCT for the complete full precision output. This rule seems to give better image quality when high bandwidth displays are used. The points marked “x” are an alternative reconstruction which tends to give better images on lower bandwidth displays. “x” and “r” are the same for slice 0. The system designer must determine which strategy is best for the display system being used.

K.10 Example of point transform

The difference between the arithmetic-shift-right by Pt and divide by 2^{Pt} can be seen from the following:

After the level shift the DC has values from +127 to –128. Consider values near zero (after the level shift), and the case where Pt = 1:

Before	Before	After	After
level shift	point transform	divide by 2	shift-right-arithmetic 1
131	+3	+1	+1
130	+2	+1	+1
129	+1	0	0
128	0	0	0
127	–1	0	–1
126	–2	–1	–1
125	–3	–1	–2
124	–4	–2	–2
123	–5	–2	–3

The key difference is in the truncation of precision. The divide truncates the magnitude; the arithmetic shift truncates the LSB. With a divide by 2 we would get non-uniform quantization of the DC values; therefore we use the shift-right-arithmetic operation.

For positive values, the divide by 2 and the shift-right-arithmetic by 1 operations are the same. Therefore, the shift-right-arithmetic by 1 operation effectively is a divide by 2 when the point transform is done before the level shift.

Annex L

Patents

(This annex does not form an integral part of this Recommendation | International Standard)

L.1 Introductory remarks

The user's attention is called to the possibility that – for some of the coding processes specified in Annexes F, G, H, and J – compliance with this Specification may require use of an invention covered by patent rights.

By publication of this Specification, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. However, for each patent listed in this annex, the patent holder has filed with the Information Technology Task Force (ITTF) and the Telecommunication Standardization Bureau (TSB) a statement of willingness to grant a license under these rights on reasonable and non-discriminatory terms and conditions to applicants desiring to obtain such a license.

The criteria for including patents in this annex are:

a) the patent has been identified by someone who is familiar with the technical fields relevant to this Specification, and who believes use of the invention covered by the patent is *required* for implementation of one or more of the coding processes specified in Annexes F, G, H, or J;

b) the patent-holder has written a letter to the ITTF and TSB, stating willingness to grant a license to an unlimited number of applicants throughout the world under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

This list of patents shall be updated, if necessary, upon publication of any revisions to the Recommendation | International Standard.

L.2 List of patents

The following patents may be required for implementation of any one of the processes specified in Annexes F, G, H, and J which uses arithmetic coding:

US 4,633,490, December 30, 1986, IBM, MITCHELL (J.L.) and GOERTZEL (G.): *Symmetrical Adaptive Data Compression/Decompression System.*

US 4,652,856, February 4, 1986, IBM, MOHIUDDIN (K.M.) and RISSANEN (J.J.): *A Multiplication-free Multi-Alphabet Arithmetic Code.*

US 4,369,463, January 18, 1983, IBM, ANASTASSIOU (D.) and MITCHELL (J.L.): *Grey Scale Image Compression with Code Words a Function of Image History*.

US 4,749,983, June 7, 1988, IBM, LANGDON (G.): *Compression of Multilevel Signals*.

US 4,935,882, June 19, 1990, IBM, PENNEBAKER (W.B.) and MITCHELL (J.L.): *Probability Adaptation for Arithmetic Coders*.

US 4,905,297, February 27, 1990, IBM, LANGDON (G.G.), Jr., MITCHELL (J.L.), PENNEBAKER (W.B.), and RISSANEN (J.J.): *Arithmetic Coding Encoder and Decoder System*.

US 4,973,961, November 27, 1990, AT&T, CHAMZAS (C.), DUTTWEILER (D.L.): *Method and Apparatus for Carry-over Control in Arithmetic Entropy Coding*.

US 5,025,258, June 18, 1991, AT&T, DUTTWEILER (D.L): *Adaptive Probability Estimator for Entropy Encoding/Decoding*.

US 5,099,440, March 24, 1992, IBM, PENNEBAKER (W.B.) and MITCHELL (J.L.): *Probability Adaptation for Arithmetic Coders*.

Japanese Patent Application 2-46275, February 26, 1990, MEL ONO (F.), KIMURA (T.), YOSHIDA (M.), and KINO (S.): *Coding System*.

The following patent may be required for implementation of any one of the hierarchical processes specified in Annex H when used with a lossless final frame:

US 4,665,436, May 12, 1987, EI OSBORNE (J.A.) and SEIFFERT (C.): *Narrow Bandwidth Signal Transmission*.

No other patents required for implementation of any of the other processes specified in Annexes F, G, H, or J had been identified at the time of publication of this Specification.

L.3 Contact addresses for patent information

Director, Telecommunication Standardization Bureau (formerly CCITT)
International Telecommunication Union
Place des Nations
CH-1211 Genève 20, Switzerland
Tel. +41 (22) 730 5111
Fax: +41 (22) 730 5853

Information Technology Task Force
International Organization for Standardization
1, rue de Varembé
CH-1211 Genève 20, Switzerland
Tel: +41 (22) 734 0150
Fax: +41 (22) 733 3843

Program Manager, Licensing
Intellectual Property and Licensing Services
IBM Corporation
208 Harbor Drive
P.O. Box 10501
Stamford, Connecticut 08904-2501, USA
Tel: +1 (203) 973 7935
Fax: +1 (203) 973 7981 or +1 (203) 973 7982

Mitsubishi Electric Corp.
Intellectual Property License Department
1-2-3 Morunouchi, Chiyoda-ku
Tokyo 100, Japan
Tel: +81 (3) 3218 3465
Fax: +81 (3) 3215 3842

AT&T Intellectual Property Division Manager
Room 3A21
10 Independence Blvd.
Warren, NJ 07059, USA
Tel: +1 (908) 580 5392
Fax: +1 (908) 580 6355

Senior General Manager
Corporate Intellectual Property and Legal Headquarters
Canon Inc.
30-2 Shimomaruko 3-chome
Ohta-ku Tokyo 146 Japan
Tel: +81 (3) 3758 2111
Fax: +81 (3) 3756 0947

Chief Executive Officer
Electronic Imagery, Inc.
1100 Park Central Boulevard South
Suite 3400
Pompano Beach, FL 33064, USA
Tel: +1 (305) 968 7100
Fax: +1 (305) 968 7319

Annex M

Bibliography

(This annex does not form an integral part of this Recommendation | International Standard)

M.1 General references

LEGER (A.), OMACHI (T.), and WALLACE (G.K.): JPEG Still Picture Compression Algorithm, *Optical Engineering*, Vol. 30, No. 7, pp. 947-954, 1991.

RABBANI (M.) and JONES (P.): Digital Image Compression Techniques, *Tutorial Texts in Optical Engineering*, Vol. TT7, SPIE Press, 1991.

HUDSON (G.), YASUDA (H.) and SEBESTYEN (I.): The International Standardization of a Still Picture Compression Technique, *Proc. of IEEE Global Telecommunications Conference*, pp. 1016-1021, 1988.

LEGER (A.), MITCHELL (J.) and YAMAZAKI (Y.): Still Picture Compression Algorithm Evaluated for International Standardization, *Proc. of the IEEE Global Telecommunications Conference*, pp. 1028-1032, 1988.

WALLACE (G.), VIVIAN (R.) and POULSEN (H.): Subjective Testing Results for Still Picture Compression Algorithms for International Standardization, *Proc. of the IEEE Global Telecommunications Conference*, pp. 1022-1027, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Evolving JPEG Colour Data Compression Standard, *Standards for Electronic Imaging Systems*, M. Nier, M.E. Courtot, Editors, SPIE, Vol. CR37, pp. 68-97, 1991.

WALLACE (G.K.): The JPEG Still Picture Compression Standard, *Communications of the ACM*, Vol. 34, No. 4, pp. 31-44, 1991.

NETRAVALI (A.N.) and HASKELL (B.G.): *Digital Pictures: Representation and Compression*, Plenum Press, New York 1988.

PENNEBAKER (W.B.) and MITCHELL (J.L.): *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, New York 1993.

M.2 DCT references

CHEN (W.), SMITH (C.H.) and FRALICK (S.C.): A Fast Computational Algorithm for the Discrete Cosine Transform, *IEEE Trans. on Communications*, Vol. COM-25, pp. 1004-1009, 1977.

AHMED (N.), NATARAJAN (T.) and RAO (K.R.): Discrete Cosine Transform, *IEEE Trans. on Computers*, Vol. C-23, pp. 90-93, 1974.

NARASINHA (N.J.) and PETERSON (A.M.): On the Computation of the Discrete Cosine Transform, *IEEE Trans. on Communications*, Vol. COM-26, No. 6, pp. 966-968, 1978.

DUHAMEL (P.) and GUILLEMOT (C.): Polynomial Transform Computation of the 2-D DCT, *Proc. IEEE ICASSP-90*, pp. 1515-1518, Albuquerque, New Mexico 1990.

FEIG (E.): A Fast Scaled DCT Algorithm, in *Image Processing Algorithms and Techniques*, Proc. SPIE, Vol. 1244, K.S. Pennington and R. J. Moorhead II, Editors, pp. 2-13, Santa Clara, California, 1990.

HOU (H.S.): A Fast Recursive Algorithm for Computing the Discrete Cosine Transform, *IEEE Trans. Acoust. Speech and Signal Processing*, Vol. ASSP-35, No. 10, pp. 1455-1461.

LEE (B.G.): A New Algorithm to Compute the Discrete Cosine Transform, *IEEE Trans. on Acoust., Speech and Signal Processing*, Vol. ASSP-32, No. 6, pp. 1243-1245, 1984.

LINZER (E.N.) and FEIG (E.): New DCT and Scaled DCT Algorithms for Fused Multiply/Add Architectures, *Proc. IEEE ICASSP-91*, pp. 2201-2204, Toronto, Canada, 1991.

VETTERLI (M.) and NUSSBAUMER (H.J.): Simple FFT and DCT Algorithms with Reduced Number of Operations, *Signal Processing*, 1984.

VETTERLI (M.): Fast 2-D Discrete Cosine Transform, *Proc. IEEE ICASSP-85,* pp. 1538-1541, Tampa, Florida, 1985.

ARAI (Y.), AGUI (T.), and NAKAJIMA (M.): A Fast DCT-SQ Scheme for Images, *Trans. of IEICE,* Vol. E.71, No. 11, pp. 1095-1097, 1988.

SUEHIRO (N.) and HATORI (M.): Fast Algorithms for the DFT and other Sinusoidal Transforms, *IEEE Trans. on Acoust., Speech and Signal Processing,* Vol ASSP-34, No. 3, pp. 642-644, 1986.

M.3 Quantization and human visual model references

CHEN (W.H.) and PRATT (W.K.): Scene adaptive coder, *IEEE Trans. on Communications*, Vol. COM-32, pp. 225-232, 1984.

GRANRATH (D.J.): The role of human visual models in image processing, *Proceedings of the IEEE,* Vol. 67, pp. 552-561, 1981.

LOHSCHELLER (H.): Vision adapted progressive image transmission, *Proceedings of EUSIPCO,* Vol. 83, pp. 191-194, 1983.

LOHSCHELLER (H.) and FRANKE (U.): Colour picture coding – Algorithm optimization and technical realization, *Frequenze,* Vol. 41, pp. 291-299, 1987.

LOHSCHELLER (H.): A subjectively adapted image communication system, *IEEE Trans. on Communications*, Vol. COM-32, pp. 1316-1322, 1984.

PETERSON (H.A.) *et al*: Quantization of colour image components in the DCT domain, *SPIE/IS&T 1991 Symposium on Electronic Imaging Science and Technology*, 1991.

M.4 Arithmetic coding references

LANGDON (G.): An Introduction to Arithmetic Coding, *IBM J. Res. Develop.*, Vol. 28, pp. 135-149, 1984.

PENNEBAKER (W.B.), MITCHELL (J.L.), LANGDON (G.) Jr., and ARPS (R.B.): An Overview of the Basic Principles of the Q-Coder Binary Arithmetic Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 717-726, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder Binary Arithmetic Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 727-736, 1988.

PENNEBAKER (W.B.) and MITCHELL (J.L.): Probability Estimation for the Q-Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 737-752, 1988.

MITCHELL (J.L.) and PENNEBAKER (W.B.): Software Implementations of the Q-Coder, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 753-774, 1988.

ARPS (R.B.), TRUONG (T.K.), LU (D.J.), PASCO (R.C.) and FRIEDMAN (T.D.): A Multi-Purpose VLSI Chip for Adaptive Data Compression of Bilevel Images, *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 775-795, 1988.

ONO (F.), YOSHIDA (M.), KIMURA (T.) and KINO (S.): Subtraction-type Arithmetic Coding with MPS/LPS Conditional Exchange, *Annual Spring Conference of IECED*, Japan, D-288, 1990.

DUTTWEILER (D.) and CHAMZAS (C.): Probability Estimation in Arithmetic and Adaptive-Huffman Entropy Coders, submitted to *IEEE Trans. on Image Processing*.

JONES (C.B.): An Efficient Coding System for Long Source Sequences, *IEEE Trans. Inf. Theory,*Vol. IT-27, pp. 280-291, 1981.

LANGDON (G.): Method for Carry-over Control in a Fifo Arithmetic Code String, *IBM Technical Disclosure Bulletin*, Vol. 23, No.1, pp. 310-312, 1980.

M.5 Huffman coding references

HUFFMAN (D.A.): A Method for the Construction of Minimum Redundancy codes, *Proc. IRE,* Vol. 40, pp. 1098-1101, 1952.