TAPE OPERATIONS
SOFTWARE SUBSYSTEM DESCRIPTION
NO. 3 ELECTRONIC SWITCHING SYSTEM

CONTENTS PAGE

NOTICE
Not for use or disclosure outside the
Bell System except under written agreement

Printed in U.S.A.

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

Figures

CONTENTS PAGE

Tables

1. GENERAL

1.01 This section provides a functional description of the software required to perform tape operations in the No. 3 Electronic Switching System (ESS) office. The areas discussed include:

- Tape format
- Tape handling
- Paging functions
- Tape "client" programs (tape update, etc).

Tape unit diagnostics are discussed in Section 233-153-120, Peripheral Unit Diagnostics.

1.02 This section is being reissued to provide information concerning tape organization relative to the 3E3 generic. Part 2 contains complete coverage of tape operations for the SO-2, Issue 4 or 4A generic. Part 3 contains complete coverage of tape operations for the 3E3 generic. Since this is a general revision, no revision arrows have been used to denote significant changes.

1.03 Part 4 contains a glossary of terms, abbreviations, and definitions necessary for comprehension of the information contained in this document.

1.04 The following sections may be helpful in understanding tape operations:

SECTION	TITLE
233-153-130	Initialization and Processor Fault Recovery, Software Subsystem Description, No. 3 ESS
233-152-125	System Control, Software Subsystem Description, No. 3 ESS
233-151-150	Translations, Software Subsystem Description, No. 3 ESS
233-153-120	Peripheral Unit Diagnostics, Software Subsystem Description, No. 3 ESS
254-300-170	Tape Data Controller Unit, Description and Theory, Common Systems
233-152-135	Traffic and Plant Measurements, Software Subsystem Description, No. 3 ESS
254-340-082	System Utilities, Software Subsystem Description, 3A Processor.

1.05 Information contained in this section will aid in accessing the software listings which contain detailed program functions and coded software instructions. Table A contains the acronyms, names, and program listing numbers of each program referenced in this document as well as the major functions performed by each.

2. TAPE OPERATIONS SUBSYSTEM FUNCTIONAL DESCRIPTION—SO-2 ISSUE 4 OR 4A GENERIC

CARTRIDGE TAPE SYSTEM

2.01 The No. 3 ESS utilizes a magnetic cartridge tape unit to provide a backup memory storage facility for data used to operate and maintain the office. This unit is duplicated for reliability and is referred to as the tape data controller (TDC) unit. The cartridge tape has four tracks. Track 1 of the tape is write-protected while tracks 2, 3, and 4 may be rewritten at field locations.

2.02 The cartridge tape system performs the following functions in a No. 3 ESS:

(a) Provides a backup for system programs and translation data (and any associated overwrites) if memory reloading is required to restore the system to an operational state

(b) Provides additional backdated translation data backups if an error in up-to-date translation data necessitates reload

(c) Stores the line class code translation tables

(d) Stores office records and associated remarks for printing when an office records verify message is received from the TTY

(e) Holds the contents of traffic registers each busy hour for later printout on the traffic TTY

(f) Stores nonresident programs to be paged and executed in the paging buffer.

2.03 The basic element of the tape system is the tape block (Fig. 1) which has a physical length of 258 16-bit words (16 bits each because the two additional parity bits for each word are not stored on tape). Between each block is an interblock gap. The block is made up of a preamble, block address, data area, cyclic redundancy character (CRC), and postamble.

2.04 The preamble and postamble are 16-bit hardware synchronization codes which are added and removed automatically by the TDC when data transfer is in progress between the 3A Central Control (3A CC) and the TDC. The preamble is a prefix while the postamble is a terminator. These two words add to the physical size of a block on the tape but are automatically removed during reading and added during writing by the TDC. They are not accounted for within the processor during the administration of the tape block, thus yielding a logical blocksize of 256 words.

2.05 Any block on the tape may be accessed in a random fashion for either reading (tracks 1 through 4) or writing (tracks 2 through 4 since track 1 is write-protected). The 1-word block header contains an 11-bit block number specifying its physical position relative to the beginning of tape (BOT). This enables the requested block to be located. Therefore, the tape position can be determined by counting the blocks passing over the read head without reading the contents of the blocks. The block header also contains a 2-bit track designation field which denotes the binary track location (0, 1, 2, or 3) of a block.

2.06 A bit-designated end-of-file (EOF) is contained within the block header. This bit is set to one in the last block of each file to signal the termination of the file. The bit also indicates to the tape handling program that a stop be issued when a continuous read has been requested by a tape client. In other words, the last block of each file must contain an EOF bit, but more EOF or file segmentation bits can be contained within a file to further segment the file into subfiles. Certain files cross track boundaries, ie, may be continued from the end of one track to the first block of the next track. A double end-of-file (DEOF) bit is used to flag the tape handler to rewind and adjust all track designation control to the next track, whereby the client of the tape assumes control at the next logical block in its file.

2.07 There are 254 words available in each block for the data area. Formats of data stored vary depending on the file or type of data being stored.

2.08 The last word of a tape block preceding the postamble contains a check character referred to as the CRC. This 16-bit constant is used to validate the data read in or written out by the tape handler.

2.09 A grouping of a sequential number of blocks, under a given name of up to eight characters, is referred to as a file. All four tracks of the cartridge tape are completely formatted. No area between files is left void of data. Empty data areas contain blocks with all zero data areas and the correct block and track designation and CRC words. The area between the beginning-of-tape mark and the first recorded information on a track and the area between the last physical block on a track and the end-of-tape holes must be free of recorded information. An interblock gap level recording exists in those areas of the tape. The List 1 cartridge contains all files required in a No. 3 ESS office; the positions of these files are depicted in Fig. 2. A List 2 cartridge is used to supply an office with a new current program base and patches and contains only those files shown in Fig. 3. Before a List 2 cartridge can be used to reload a system, it must be updated to a List 1 state at the office.

TAPE FILES

A. Directory

2.10 A tape directory (Fig. 4) resides on track 2 of each tape. The directory identifies the physical position of each file on the tape to the tape handling program. The directory starts in the data area of the first block of the directory file and contains a 1-word header defining the number of blocks needed for the directory and the total number of files residing in the directory. The file definitions begin immediately following this header.

2.11 Each file definition entry is made up of six words. The first four words contain the file name of up to eight American Standard Code for Information Interchange (ASCII) characters. Unused character spaces are zero. The fifth word of an entry contains the last physical block number of the file and the track designation for that block. The last word of an entry contains the first physical block number of the file, the track designation for that block, and the tape pair on which the file is located. (The No. 3 ESS uses only one pair of tape units, whereas the tape handling software has the capability to handle more than one pair.) The duplicated bit (DUP) flags the handling program so that when one block of this file is written on a tape, the mate unit is automatically written. This alleviates the necessity for a client program to issue additional orders in keeping both units up-to-date. No file entry is allowed to overflow into another block of the directory. Unused words after the last directory entry are all zero.

B. Bootstrap Files

2.12 Bootstrapping a processor consists of loading memory from the tape and using the processor and its microcode as the controlling mechanism. Layout of information on the tape is engineered for bootstrap operations. Files associated with system recovery are positioned in a sequence permitting unidirectional tape operation (ie, no searching required to access the next file) in case of a massive memory mutilation.

2.13 The bootstrap file, which contains code to perform bootstrap operations, begins in block 0 of track 1. Track 1 is the only track on the tape containing block 0 which is positioned approximately 36 inches from the tape load point. The bootstrap files and generic file are located on track 1 because the two inner tracks (1 and 2) of the tape are more reliably read. Track 1 is also write-protected, meaning it can only be read and not written, thus protecting the system programs. In addition, the bootstrap file is duplicated on track 2 for more reliability. The second block on track 1 (block 1) contains an all-zero data area so that the bootstrap files on tracks 1 and 2 will be synchronized after the first block. The reload process can alternate between tracks 1 and 2 in case an error has been detected. Only an error in the same block on both tracks can block execution.

2.14 The bootstrap file is segmented into two subfiles by EOF marks. The first subfile contains the common initial program loader (CIPL) and the last block of CIPL contains the EOF bit. This bit indicates that the next block on track 1 initiates the system core of programs required to continue initialization. System core programs are extracted from the generic program and are stored in the second subfile of the bootstrap file. System core programs include:

- The first 129 words of store
- CBLM — common base level monitor
- CINIT — common initialization
- CPATCH — common patch area
- CSYSUB — common system subroutines
- MASACS — main store access routines.

The last block of the second subfile also contains an EOF bit. A block containing either program or translation is illustrated in Fig. 5. Initialization and bootstrap functions are described in Section 233-153-130, Initialization and Processor Fault Recovery, Software Subsystem Description, No. 3 ESS.

C. Checksum File

2.15 The checksum file is read during memory reload operations and is updated during patching and recent change operations. It contains checksums used to determine which 4096-word (4K-word) segments of memory are to be reloaded and miscellaneous system parameters needed to perform reload operations. A checksum is a logically derived, data-dependent word which is generated based on the information stored in each 4K-word block. Therefore, during a memory reload, the checksum of each 4K segment of memory beginning on a 4K boundary can be calculated and compared with the checksum stored in the checksum file. If the checksum calculated for a 4K segment of a generic program store does not match the corresponding checksum read from the checksum file, that 4K segment of generic program is reloaded into memory. Although similar checksums exist for translation store, they are not used during a memory reload in the No. 3 ESS.

2.16 The checksum file is adjacent to the backup bootstrap file on track 2 and is accessible for update. The CIPL program accesses the file to recover the system parameters needed to pinpoint other files required for bootstrap operations.

2.17 See Fig. 6 for the format of the checksum file. Word 0 contains the block address, track number, EOF bit, and DEOF bit. Words 1 and 2 contain the field length (number of words comprising the used portion in the block) and the 20-bit value of the symbol CHKSUM. Word 3 contains an update-in-progress bit (which is set when the translation file is being updated) and the starting block number of the generic file. The starting block number of the translation file is in word 4. Word 5 contains the number of 4K translation subfiles on track 2. Word 6 specifies the starting block and track number of the patch file which contains overwrites to the generic program. The number of 4K program store segments containing the generic program and its related temporary memory is in word 7.

2.18 Checksum information related to the generic area of main store follows the header words. Each 2-word entry relates to a 4K-word segment of memory. The checksum field is 24 bits long. The low 16 bits comprise the second word; the high 8 bits occupy the low 8 bits of the first word and are zero. The checksum word is generated by exclusive "OR"ing all the words residing within the 4K-word store segment.

2.19 Every 4K-word segment within the generic area of main store has a 2-word checksum entry reserved for it. If the segment contains temporary storage, no checksum constant is required and the 24-bit field is zero. However, the temporary store flag (TSFLG) in the first word of the related checksum entry is set because no space is reserved in the generic file for temporary store segments. These segments must be indicated to the bootstrap program for the reloading functions. The first word of the last generic checksum entry contains a SHARED bit which is set if translation does not begin on a 4K boundary. This indicates that the last segment of the generic program also contains some translation data. Thus, the last generic checksum must equal the first translation checksum if the SHARED bit is set.

2.20 After the generic checksum entries, a 2-word entry exists of which the first word is unused. The second word contains the number of 4096-word main store segments of translations. Checksum entries for the translations follow. The checksum file is three blocks in length. The length and address fields at the beginning of each block are zero for blocks which do not contain data.

D. Generic File

2.21 The generic file resides on track 1 (which is write-protected). This file contains only the program portion of the generic. No temporary memory is stored on the tape. The file is divided into 4K-word subfiles, ie, each 4K-word program segment starts on a new block and contains an EOF bit in its last block. The first two words in the data area of each block (Fig. 5) specify the loading length and address for each block in memory. Following the length and address are 16-bit words of the generic. Seventeen blocks are required to store one 4K-word segment (16 full blocks and 64 words of the seventeenth block).

2.22 To save main storage, translation is not necessarily begun on a 4K-word boundary and thus shares a segment with the generic in storage. Therefore, the last 4K-word subfile in the generic file on tape contains only the generic program sharing the segment with translation. The last subfile is padded with all-zero data area blocks until the subfile is filled. The last block of the subfile still contains an end-of-file mark. The translation subfile sharing the 4K-word block of main store begins with all-zero data area blocks until the appropriate block for recording the remaining data for that 4K-word block of main store. See Fig. 7 for an illustration of the above arrangement.

2.23 Since the 4K-word block is shared by both generic and translation, the last checksum of the generic area equals the first checksum of the translation area. Only the generic entry carries the SHARED bit designation to indicate this overlap to programs using checksum data.

E. Current Translation File

2.24 The translation (TRNSLN) file begins after the generic file but resides on track 2. It follows the OFR2 file on track 2 and is not marked duplicated in the directory. The general format of a translation block is the same as that for a generic block (Fig. 5). The TRNSLN file is segmented into 17-block subfiles by setting the EOF bit in every seventeenth block. Up to 4K words of translations are stored in each subfile. In a full subfile, therefore, the seventeenth block contains only 64 words of translations. Following the translation data in the last block of the first subfile is a word containing the date (in packed form) of the last update of the file. This date is modified each time an update occurs, even if no data on TRNSLN is actually changed.

2.25 There are 221 blocks in the file which can handle up to 53,248 words of translation. Many offices may not initially fill up all 221 blocks; therefore, the remaining unused file is formatted with all zero data area blocks.

F. Backdated Translation Files (BACKDT1 and BACKDT2)

2.26 If the PAST OFFICE DATA or BACKDT OFFICE DATA key on the system status panel is pressed, an initialization program will load backdated translations from either the BACKDT1 translation file or the BACKDT2 translation file. Figure 8 depicts the layouts of each of these files. Unlike the TRNSLN file, neither of these files is segmented into 17-block subfiles.

2.27 Block 0 of BACKDT1 differs from block 0 of BACKDT2 in that it contains two date words (words 1 and 2). These words contain a packed representation of the date of the last update of BACKDT1 and BACKDT2, respectively. These date words are used by the initialization program to determine which of the two files contains the more recent version of translations. Therefore, when the PAST OFFICE DATA key on the system status panel is pressed, the file containing the more recent backdated version of translations is loaded into memory. When the BACKDT OFFICE DATA key is pressed, the file with the less recent backdated version of translations is loaded. If one of the date words is zero, the file corresponding to the nonzero date word is used to reload memory. The files BACKDT1 and BACKDT2 are marked duplicated in the directory indicating to the system that a block written by a client on one tape should be automatically written on its mate tape if it is in service.

G. Line Class Code Table Files (LCCTBL0, LCCTBL1, LCCTBL2)

2.28 Each assigned directory number in the No. 3 ESS office has a 3-character line class code (LCC) which identifies the line class of service. Examples of line class codes are 1FR (single-party flat rate residential service) and 1FB (single-party business rate service). The line class code is not used in call processing but is provided to make the recent change input and verify output messages more meaningful. A telephone number is associated with a line class code and its related information through the use of an 8-bit line class code index (LCI). Two tables are involved in associating a telephone number with its line class code: a table of 8-bit entries (LCIs) indexed by the packed telephone number, and a table of 3-word LCC entries indexed by the 8-bit LCI. The relationship and layouts of these two tables are depicted in Fig. 9. An 8-bit LCI provides for a maximum of 256 3-word LCC entries (a maximum LCC table size of 768 words). Five bits are needed to represent the office index and ten bits are needed for a binary representation of the last four digits of telephone number; there are 32,768 packed telephone numbers. Since each telephone number has an 8-bit LCI, the telephone number to LCI table is 16,384 words in length.

2.29 These two tables do not reside in translation store for No. 3 ESS because they are prohibitively large and are used only for some recent change and verify operations and not for processing calls. An up-to-date version of these tables is stored on the tape in file LCCTBL0. If memory is reloaded using the PAST or BACKDT OFFICE DATA keys, then line class code information which agrees with the backdated translation store must be used for recent change and verify operations. These backdated line class code tables are stored on files LCCTBL1 (corresponding to file BACKDT1) and LCCTBL2 (corresponding to file BACKDT2).

2.30 Figure 10 shows the layout of the tables within any one of the three LCCTBL files. The 256-entry LCC table is stored in the data area of blocks 0 through 3 of the file, and the telephone number to LCI table occupies the remainder of the file. Block 3 contains only four 3-word entries. In file LCCTBL1, the two words following the last entry in this block are used to store the dates of the last update of LCCTBL1 and LCCTBL2, respectively. The function of these date words is similar to that of the BACKDT1 file date words. If backdated translations are loaded into memory and recent changes or verifies involving line class codes are then performed, the date words are used to determine which file (LCCTBL1 or LCCTBL2) contains the appropriate tables to be used for the recent changes and verifies. If a date word is zero, the file associated with the nonzero date word is used.

2.31 The line class code table files are marked duplicated in the directory.

H. Traffic File

2.32 An operating company can choose up to 23 busy hours for which traffic registers can be saved for later printout. The traffic registers are saved at the end of each busy hour on a tape file, TRAFFIC. At the scheduled daily time, the tape is positioned at the beginning of TRAFFIC, and the register contents for each busy hour are read from the file and printed on the traffic TTY.

2.33 Figure 11 depicts the layout of a block of TRAFFIC data. If word 1 (the header) of the block is 0, then the block starts a new busy hour (data for a new busy hour is always started in a new block). For a new busy hour, the three words following the zeroed header contain information relative to the busy hour. If the header is nonzero, it contains a name field identifying the type of traffic data which follows it and a length field with the number of words of traffic data to follow. A nonzero header is followed immediately by the traffic data. The traffic data itself never crosses a block boundary, ie, every block begins with a header word immediately following the block address. A zero header other than the first header in the block indicates that there is no more traffic data in the block. The TRAFFIC file is marked duplicated in the tape directory. The processing of the traffic file is described in Section 233-152-135, Traffic and Plant Measurements.

I. Patch File

2.34 All patches or revisions to the generic file on track 1 are applied by placing the revisions in the patch file on track 2 immediately following the translations file. This file is also accessed during a reload of the generic. Refer to Fig. 12 while reading the following description of the patch file. The areas described are identified in the figure by the letters.

A. Block address word.

B. Size of the patch file header.

C. Area designated as the patch file header area. These words appear only in the first block of the file.

D. This word is set to octal (177777).

E. A 4-word area reserved for the generic identifier which is a maximum of eight ASCII characters. Unused character space is equal to zero.

F. A 4-word area reserved for the generic issue identifier (a maximum of eight ASCII characters). Unused character space is zero.

G. A length indicator covering all the 16-bit words used to store a given patch. It is a logical index to the next patch.

H. A 14-bit field which contains the patch number assigned by Bell Telephone Laboratories.

I. A 2-bit field equal to 1.

J. Three 16-bit words set to zero.

K. A length field specifying the number of consecutive words patched at the starting address in the M field.

L. A 12-bit field containing the segment number designation for the words patched at location M. Fields L, M, N, O, P, and Q are given in the overwrite input message.

M. The address (20 bits) of the first location involved with this segment of the patch.

N. The low 16 bits of the new data.

O. The high 8 bits of the new data which is always zero for the No. 3 ESS.

P. The high 8 bits of the old data which is always zero for the No. 3 ESS.

Q. The low 16 bits of the old data.

2.35 The N through Q fields are repeated for each consecutive word patched starting at location M. When a new address is required due to a break in the continuity of patched locations, a K entry is started. If no more words exist in the patch, a new patch is started with a G field. The location following the last used Q entry is set to octal (177777) as an internal logical EOF indicator. All unused words after the logical EOF mark and all unused blocks following the block containing this file mark contain all-zero data areas. The last block of the patch file contains the file segmentation, or EOF, bit.

J. LABEL File

2.36 The file designated LABEL (Fig. 13) contains the generic identification. The file resides on track 2 after the directory and is one block in length. The first word of the block contains the normal block address header. The second word indicates the number of ASCII characters in the following words that are used to identify the generic. There are two characters in each word. Following the last word containing the generic identifier is the number of ASCII characters required to identify the issue number of the generic. The words containing the issue number identification characters follow with all remaining unused words in the data area of the block being zero.

K. ROLL File

2.37 The ROLL file resides on track 3 after the office records file. The file contains temporary storage data for multiscan functions. For instance, when a multiscan function of higher priority is permitted to interrupt a multiscan function of lower priority being executed, the lower priority function can save data in the ROLL file for later execution.

L. Paging File

2.38 The paging file contains nonresident program segments for the No. 3 ESS. The file resides on track 4. Segments vary in length but each segment begins in a new block and is terminated with a block containing a file segmentation bit. The program data is stored in blocks of the same format as shown in Fig. 5.

M. Office Record Forms File

2.39 The office record forms file resides on tracks 2 and 3. The file contains office record forms and comments to be used in printing the office records on the TTY. The blocks are formatted in the normal manner with the forms and comments stored in the data area of the blocks.

TAPE HANDLER FUNCTIONS

A. Relationship of Tape Handler to Tape Client Programs

2.40 The tape data controller buffers data between the tape unit and the 3A CC in two switchable hardware buffers so that the processor is not required to engage in continuous communication with the tape unit. Each hardware buffer can store 64 words or 1024 bits. Therefore, four buffers must be filled in order to read or write a full block of 256 words from the tape. The tape unit and processor have access to only one of these buffers at a time. The buffer referred to as the "off-line" buffer is accessed by the processor; while the "on-line" buffer is accessed by the tape unit. The buffer that is on-line becomes off-line and vice versa after the tape unit has serviced that buffer. Since the processor is significantly faster in its ability to service one of these buffers than the slower tape unit, a time interval exists between the processor servicing of one buffer and the tape unit flagging the processor that the next successive buffer is ready. The buffers must be serviced within a specified time to prevent an "overflow" condition; therefore, the buffers are serviced during interrupts generated by a buffer being ready for servicing (loading or unloading).

2.41 There is also a software buffer, referred to as the input/output (I/O) buffer, in main store which has a 256-word storage capacity. The data to be written on the tape is stored in the I/O buffer and then accessed to fill the hardware buffers with the bits to be written onto the tape. In reading, the bits are transferred from the hardware buffers to the I/O buffer in memory to be made available to the using software.

2.42 Programs which read or write tapes use tape macros to supply the tape operation required and gain access to the common tape handler program, CTAPH. The tape macro names and their major functions are shown in Table B. The tape "client" programs, in conjunction with the macros, supply information needed for the tape operation in the tape data control block (TCB), a 16-word block of memory reserved for tape handling software (Fig. 14).

2.43 The tape handler program, CTAPH, initiates the hardware commands to the tape data controllers and buffer units necessary to perform the tape operations requested by the client programs. These hardware commands are described in Section 254-300-170, Tape Data Controller, Description and Theory. The general format of the command word is depicted in Fig. 15. In most cases, CTAPH provides the command word and then calls a subroutine in the common systems subroutines program, CSYSUB, to actually issue the I/O order and monitor the response.

2.44 The tape client programs are scheduled under control of the multiscan function (MSF) controller in the common base level monitor, CBLM.

In essence, a client program seizes control of the TCB (if the TCB is idle) and makes the tape operation request via use of the tape macros. Only one client program may access the tapes at any given time. The identifying number of the client is stored in the TCB as well as the request bits identifying the tape operation requested. The address to which return is to be made after the tape operation is completed by CTAPH is also stored in the TCB.

2.45 The client programs must schedule each tape operation to be performed and access the macros to provide the interface to CTAPH for those operations. For example, if a client is reading blocks at random in a given file, first the OPEN macro is used to seize control of the TCB and to position the tape heads at the beginning of the file needed. The POSITION macro is used for positioning the tape at the block to be read. After the tape is positioned, the client uses the RD1BLK macro for the read operation. Then the tape is positioned at the next block to be read, and so on. The client must use the CLOSE macro to relinquish control of the TCB after all needed blocks have been read.

B. Organization of the Tape Handler Program

2.46 The common tape handler program, CTAPH, is a resident program. One portion of the program is executed during base level, and a second portion is given control during interrupts generated when a hardware buffer of the tape data controller is ready to be loaded or unloaded. Also included in CTAPH are subroutines which process various TTY input messages concerning the tape system. For instance, one routine services the tape unit restore message. This message restores the tape unit to service after checking to see if it passed diagnostics and a tape-to-tape audit (unless the request was unconditional). The remove-tape-from-service message is serviced by another subroutine. There is an emergency TCB cleanup routine, which is accessed with a TTY input message in the event temporary store gets randomly written and the TCB contains erroneous information.

2.47 The base level portion of CTAPH is given control by the common base level monitor, CBLM, once every complete base level loop (approximately once every 200 ms). The base level portion essentially examines the TCB for client requests for tape action and schedules the various functions required to perform these requests. It is the interface between the client and the tape. The interrupt portion is entered at interrupt level for loading and unloading of the tape unit hardware buffers in the reading and writing operations. For instance, in the writing operation, the base level portion issues the write command and fills the first two hardware buffers. When the tape data controller has emptied one of the buffers and has written it on the tape, the buffer ready flag is set. This causes an interrupt with control being given to the interrupt portion of CTAPH. Then CTAPH, during the interrupt, loads the empty buffer and the process is continued until four buffers of information required to write a tape block have been filled and written on the tape. The base level portion of CTAPH monitors the operation until it is complete and then schedules the next function. Figure 16 depicts the client, tape handler, and tape interfaces.

2.48 The requested tape operation of the client may be aborted (halted) prematurely before completion because of various types of trouble encountered. The trouble may include operation time-outs, system initialization, etc. The TCB is cleared and control is returned to the client with an abort return code. If the operation is aborted because of a request from the MSF controller, the client cleans up and returns control to the MSF controller.

C. Base Level Portion of the Common Tape Handler

2.49 Upon entry to the base level portion of the common tape handler, CTAPH, the number of tape units in the office is determined. Also, if CTAPH determines diagnostics are in progress (by examining the proper bit in the TCB), control is returned immediately to CBLM. Otherwise, CTAPH accesses the tape status area in memory (Fig. 17), which contains status and position information for the two tape units, to determine if either unit is out of service. There is one word for each unit.

2.50 The program CTAPH sends a status command to the tape units to further determine the present status of the units. Figure 18 depicts the format of the tape status command response. When trouble is discovered (no response from the tape unit, etc), the tape unit is removed from service. Removal from service includes the printing of a TTY message, stopping tape movement if it is in motion, and setting the status indicator lamp on the system status panel. If the tape unit is also active in the TCB, the TCB is cleared and CTAPH returns control to CBLM since there is only one TCB and, therefore, no more tape work to be performed at that time.

2.51 In addition, a buffer unit status command is sent to check the status of the hardware buffer units. Figure 19 depicts the format of the buffer status command response. When an error is found in the buffer, the tape unit is removed from service as previously described.

2.52 After the status of each tape unit has been determined and updated, CTAPH examines indicators in the TCB to schedule tape functions. These indicators include the request bits, the state word, and the state "flags" word and are used for dispensing tasks.

2.53 When a client program makes a request, the request bits are supplied in the TCB to indicate the major requested operation such as:

a. Read one block

b. Write a block

c. A continuous read

d. Position the tape

e. Close a file

f. Copy a file from one tape to its mate unit

g. An entry caused by initialization.

Upon the first entry to CTAPH after the request is made, control is transferred to the major entry point corresponding to the request. This sets up further control to perpetuate the new task. Scheduling of the functions required to perform the requested tape operation is accomplished by the use of states or substates specified in the TCB which correspond to a section of code, or module. The state word in the TCB can "stack" up to four state indicators (numbers corresponding to the state). The term "stacking" refers to the procedure whereby one state can request another state to take control and can push itself down in the stack so that it will regain control when the previous state has been completed. It can also remove itself so it will not regain control. When CTAPH gets control from CBLM at base level, the value in the first state word location is used to transfer to the state which is in control.

2.54 The various states analyze functions to be performed and may schedule other states or substates to help complete the function. The substate indicators are stored in the state flag word in the TCB and indicate to the state how far the operation has progressed. The flag for a state is stored in four bits of the flag word that corresponds to the bit location of the state in the state word. That is, when a state is pushed up or down in the state word stack, the corresponding substate or flag is also moved. The state, when it gains control, uses the state flag to determine the action to be taken next within the state.

2.55 The states and actions taken by the base level portion of CTAPH are described without consideration for real-time breaks and interrupts.

Monitor State

2.56 The monitor state is entered whenever the state word in the TCB is zero. This can occur when the TCB is not under client control or when a specific client request has been completed. In either case, the TCB is ready to service a new request from the client. The monitor state transfers control, based upon the request bits, to the initial entry point corresponding to the request. The entry routine initializes for the function and supplies the next state to be given control. Request bits equal to 7 indicate the TCB is idle and awaiting a new request.

Timing State

2.57 When more time is required for a function to be completed than can be allowed at that time, a real-time break must be taken. If the time required is less than a base level loop, control is returned to CBLM. The state and substate stored in the TCB will regain control when CTAPH is given control again by CBLM on the next base level loop. However, if more time is needed, the timing state may be used. The state in CTAPH in control can push itself down a level in the state word stack of the TCB and invoke the timing state. The time required is stored in the timing word of the TCB. The timing state then is given control each base level loop until the correct amount of time has elapsed. The timing state will then remove itself from the stack, and the original state and flag will be rotated to the first position and placed in control.

2.58 When a time-out occurs during reading or writing before the operation is complete, an error has occurred and control is given to error recovery routines.

Seizure State

2.59 The OPENTCB subroutine in CTAPH is given control when one of the open macros or the seize macro is used by a tape client. The subroutine determines whether the TCB is idle (that is, no client is in control—client number equals zero). When the TCB is busy, a busy return code is given to the client. Otherwise, the subroutine seizes the TCB for the client and loads the TCB with client request information. The seizure state is loaded into the TCB state word to obtain control. The seizure state performs the following functions:

(a) Determines the tape unit to be accessed (which is the tape unit specified in the macro call or, if one is not specified, the tape unit numerically tied to the on-line processor if available).

(b) Determines the status of the tape units.

(c) Accesses the portion of the directory resident in main store to search for the block containing information on the file of interest.

(d) If the file is not in the resident portion of the directory, invokes the position state to position the tape at the beginning of the tape directory and searches the directory for that file by using the file name.

(e) Uses the position state to position the tape either at the beginning of the client file or at the block index (logical block number) in the file specified by the client.

Copy State

2.60 The copy state is responsible for reading a file off one tape and writing it onto its mate tape. First, the low bit position of the tape unit number to be copied is complemented to provide the number of the mate unit to be written. Then the status of the mate unit is checked. Next, by use of the position state, the mate unit to be written is positioned to the file and block to be copied. Again, the tape unit number is complemented to provide the tape to be copied, and the command to read one block from that tape is issued to the hardware. After timing for the read command, the substate to write the block on the mate unit is invoked. The substate complements the tape number to provide access to the mate unit. The substate also causes the write state to be executed and actually perform the functions and issue the orders required to write the block on the mate unit. (Refer to paragraphs 2.70 through 2.75 for these functions).

2.61 The copy state is used to copy a file from one tape to another tape. Therefore, after each block is written, the copy state determines whether the block just written was the last block in the file to be copied. In this case, control is returned to the client. Each block to be copied is processed as previously described.

Position State

2.62 The position state is used to position the tape at the logical block in the file requested in the TCB. However, if the seize state invoked the position state, the beginning of the directory is the goal.

2.63 The "position-OK" bit in the tape status word (Fig. 17) is examined to determine if the present position of the tape is known. The present position stored in the status word is not dependable if the tape is performing or has just completed a fast search, in which case the tape unit must be stopped and the position verified. The verify state is used to verify the position of the tape and to ensure that the tape is positioned at an interblock gap.

2.64 Once the position of the tape is known, the positioning state determines the distance between the present tape position and the goal. Depending on the distance between the two positions, the position state initiates either a fast search or slow search in the appropriate direction (forward or reverse). A fast search consists of calculating the approximate amount of time required to move the tape the number of blocks needed, storing the time in the TCB, and issuing either the fast reverse or fast forward command. The timing state is given control to perform timing functions. Verification must also be performed by the verify state after each fast search.

2.65 A slow search consists of reading or backspacing one block at a time. The timing for the slow search is also controlled by the timing state. For both fast and slow searches, a bit in the TCB is set to 1 to inhibit the transfer of data to the I/O buffer by the interrupt during reading. Only the block number is stored directly into the present tape position in the tape status word.

2.66 This procedure is continued until the tape is positioned at the block requested. Only one fast search (if needed) is attempted, but that is usually all that is required because the timing algorithm is so accurate. The slow search can then be used to position the tape at the exact block requested. Control is then returned to the client with the tape positioned at the block requested.

Verify State

2.67 The verify state places the tape heads at an interblock gap and verifies the actual tape position. Verification is performed when the position of the tape is unknown, such as after a fast search. The verify state attempts to verify the position of the tape on the track on which the file of interest resides. Verification consists of issuing a backspace command, which is defined as passing through data and stopping at the next interblock gap, and then issuing a read-one-block command. Nothing is stored by the read interrupt in the I/O buffer but the block number is stored directly into the present tape position in the tape status word.

2.68 When a failure occurs in reading, the verify substate will attempt to verify the position of the block on that track twice by backspacing and reading the block. If this procedure fails, an attempt is made to switch to the next track at the same position and verify the position of the block (block number) on that track. This is done for each track until the position is known or all tracks have been accessed. After all tracks have been used and verification has not been successful, the other tape unit will be used if available and if the client permits it.

2.69 The tape operation is aborted if verification is unsuccessful or if verification is successful only after a recovery and the present tape position is equal to the block needed by the client. A TTY message is printed indicating the fatal read error and a repacking of the tapes is scheduled. The client is aborted and the TCB is idled.

Write State

2.70 When a client requests a write operation, the first write entry routine determines from the directory information stored in the TCB whether the file is to be duplicated on its mate unit. In this case, the status of the mate unit is obtained. If the mate unit is in service, the unit is initialized and each block will be written on both tapes. Otherwise, the block is to be written only on the one tape.

2.71 The write state is scheduled to be given control of the writing sequence. The tape must have been positioned at the block to be written by the client, and the position must be known before the write operation is allowed. The client has also loaded the I/O buffer with the information to be written.

2.72 For reliability, track 1 is used as a reference point in the writing operation; the track prevents tape skewing, is write-protected, and never changes. A backspace and a read off of track 1 are performed to align the tape at the proper position for writing.

2.73 If an absolute write request is not made by the client, the write state also formulates in the I/O buffer the block header including the block address. The write state automatically sets the EOF bit if the block being written is the last block of the file. The write state ensures that the block to be written is within the range available to the client file. The CRC value is computed by this state for the block to be written and is stored in the I/O buffer.

2.74 The write state sends the order to prime the TDC buffer to receive words and loads the 64 word off-line buffer from the I/O buffer. The off-line buffer is then switched to on-line and vice versa so that the second buffer can be loaded. After the second TDC buffer is loaded, the order to start writing the block is issued. The remaining two buffers required to write a block are loaded during interrupts generated by the completion of wiring a buffer onto the tape. (See paragraphs 2.98 through 2.101 for the interrupt functions.)

The base level write state waits until the interrupt operations are complete or a time-out occurs. When control of the operation is returned to base level, a check is made for a CRC error. If an error exists, control is given to the maintenance recovery state to process the error. (See paragraphs 2.81 through 2.87.)

2.75 If the file is to be duplicated, the same write state sequence is performed for the mate unit. After the writing is complete, the original tape unit number is placed in the TCB and the TCB reverts to the monitor state (idle) waiting for the client to issue another order.

Read State

2.76 The initial entry code in CTAPH for a read request from a client sends a tape read order to the tape unit defined in the TCB. The order is either a continuous read order, which reads each block until an end-of-file is detected, or a read-one-block order, which reads only the block at which the tape is positioned.

2.77 A delay is initiated to allow the interrupt to gain control once a tape data controller buffer has been filled. The interrupt level code will further delay the read state at base level from gaining control until either the total read operation is complete or an abnormal situation arises. For a continuous read, the operation is complete when an EOF mark is encountered. For a read-one-block command, the operation is complete when the four buffers of data required to store one block have been transferred to the I/O buffer.

2.78 Once the interrupt level code stops delaying the base level, the read state gains control to investigate the success of the interrupt in handling the read operation. The read state interrogates the I/O complete bit in the TCB to assure the read was performed without error. Upon a successful read, the state word in the TCB is emptied so that new client requests can be serviced and control is returned to CBLM.

2.79 If an error occurs in the read, a tape status command is sent. No response causes the tape unit to be removed from service and the client is aborted. Otherwise, a TTY message is printed and the error recovery state is initiated (paragraphs 2.81 through 2.85).

2.80 The read state also has the responsibility to rewind the tape to the beginning-of-tape if a file continues on another track. The interrupt level code detects the double EOF bit set in the last block read and causes the correct substate of the read state to be given control. The rewind command is given and, after tape motion has ceased and the tape is positioned at the beginning-of-tape, the next track is placed in the TCB. If a continuous read was being performed, the read is restarted on the new track; otherwise, the read state is taken out of the state word leaving the TCB in the idle state. Control is returned to CBLM.

Retry State (Error Recovery)

2.81 The retry state is used in error recovery situations. Detection of a CRC error during the reading or writing operation (indicating data was not read or written correctly) stimulates recovery attempts. The maintenance bit and the search failure bit are set in the TCB to indicate the CRC error and that error recovery is in progress. Key control words in the TCB are saved for restoral after error recovery. The retry state is placed in the state word of the TCB.

2.82 The retry state invokes the verify state to attempt to reread the block for detection of any transient errors. The verify state has an error recovery routine which backspaces and attempts to read the block twice. If the error remains after the second attempt, control is returned to the retry state for more error recovery attempts.

2.83 When the error has occurred during a read operation and the two additional attempts to read the block have failed, the retry state increments the traffic fatal read counter and initiates a TTY message indicating the read error. The retry state will attempt to restore the bad block from the other tape unit if possible. However, this recovery attempt is not made when prohibited in the TCB, when the file is not duplicated on the mate unit, when the block to be recovered is on track 1 (track designation in TCB=0), or when the other tape unit is out of service. In addition, an attempt is made to read the block on track 1 immediately preceeding the bad block in order to position the tape at the beginning of the bad block; therefore, if unsuccessful, the block cannot be be recovered.

2.84 Permanent read errors (errors which cannot be corrected) cause a major alarm to be given, a TTY message to be printed, and the read operation to be aborted. The tape unit is not removed from service since other files on the tape may be unharmed.

2.85 When recovery can be attempted from the other tape unit, the position state is used to position the mate unit at the bad block. The block will then be read off the mate tape and (if successful) rewritten onto the original tape having the error. If either of these operations fails, the error cannot be recovered. However, if recovery of the block is successful, the TCB information saved in the maintenance save area is restored to the TCB, the error bits in the TCB are cleared, and the client read request is reinitiated. Another read error can occur on that block, in which case the error recovery process is repeated. Therefore, the client program times the operation requested to prevent an error loop from occurring.

2.86 If the error was incurred during a write operation, the retry state will attempt to read the block written twice and then will attempt to rewrite the block if the error persists. If the rewrite fails, the error cannot be recovered. A TTY message is printed and the client is aborted. In addition, the traffic fatal write peg count for that tape unit is incremented.

2.87 If the write recovery is successful, a TTY message is printed indicating the success and the TCB information is restored from the maintenance save area so that normal operation can continue. Control is then returned to CBLM.

Close State

2.88 When tape operations are finished, the client must make a CLOSE request to relinquish control of the TCB. This results in the close state being given control. In addition, the tape handler, CTAPH, uses the close state code to abort some tape operations when the operation cannot be completed. The close state clears the TCB but saves the bit indicating repack requests pending. The close state then indicates the TCB is idle by placing the value 7 in the request bits and restores the repacks request bit to the TCB. A pending repack request will be processed the next time CTAPH is given control at base level by CBLM to perform tape tasks and the TCB is idle. (There is no client number in the TCB and there is a 7 in the request bits.) Control is then returned to CBLM.

Pack Tape State

2.89 Repacking of the tape is performed automatically once a day or upon request. The packing process realigns the tape in a flat horizontal plane. Repacking is done only when the TCB is idle (ie, client number=0 and request bits=7) and the repack request bit in the TCB is set to 1. All client requests are inhibited during the repacking process by setting the "request stop" bit in the TCB.

2.90 Repacking the tapes is accomplished by the pack tape state. Repacking consists of initializing the tape units and fast forwarding and fast reversing of the tapes so that both ends of the tapes are reached, ie, the beginning-of-tape and the end-of-tape. A TTY message is printed indicating the completion of repacking the tapes. The close state is then scheduled to idle the TCB and to allow more requests.

Initialization Entry (MRF State)

2.91 A zero in the request bits of the TCB means an initialization of sufficient magnitude has resulted in the clearing of the call store TCB or a TTY request has resulted in the initialization of the tape system (an idle TCB has the value 7 in its request bit). The tape units are initialized and rewound to the beginning of the tapes to prepare for a bootstrap initialization, if needed. The maintenance reset function (MRF) initialization state performs and monitors the rewinding of the tape units and places the tape system in a known state ready for client use. A tape unit indicating trouble is removed from service.

D. Interrupt Portion of the Tape Handler

General

2.92 An interrupt is generated by the tape unit when one of the two tape buffers is ready to be loaded or unloaded by the processor. The buffers must be serviced within a short interval to prevent an overflow condition and thus require servicing during interrupts. The interrupt portion of the tape handler (CTAPH) is given control. A check is first made to determine whether the interrupt was generated by an in-service tape unit and thus is valid. The buffer status is then tested to determine whether the tape unit is reading or writing so that control can be given to the proper routines. If the buffer is in the unload state, a reading operation is in progress; if it is in the load state, a writing operation is in progress.

Interrupt Reading Functions

2.93 In a reading operation, the interrupt is generated after the TDC has completed loading one of the tape buffers from the tape. The buffer is then ready for unloading by the processor. The interrupt reading routine in CTAPH must determine which buffer is being unloaded (four buffers are required for reading a complete block). A counter is, therefore, maintained to identify the number of buffers serviced for a particular block.

2.94 An interrupt is generated each time a TDC buffer must be serviced by the processor. The servicing of a TDC buffer consists of unloading the buffer and storing the words in the proper area of the I/O buffer, incrementing the buffer counter, and timing to detect possible hardware errors in the unloading process. (If an error is encountered in an interrupt command, a TTY error message is printed and the close state is scheduled to obtain control at base level to abort the client.) In addition, the interrupt routine resets the timer in the TCB to extend the real-time break for base level so that the base level tape handler routine will not see a time-out until the complete operation has been performed (a whole block has been read). For a continuous read, the timer is continuously extended until all the blocks are read and the EOF bit is encountered. Each interrupt is terminated by passing control to the application base level monitor (BLMMA) which branches to a routine in the common system subroutines program (CSYSUB) to end the interrupt. The system (registers, I/O channel, etc) is restored to the preinterrupt state and return is made to the program being executed at the time of the interrupt.

2.95 When a buffer is the first buffer to be unloaded for a block, the tape status word (Fig. 17) is updated with the present tape position (block number). The position-known bit is not set at this time since it is not known whether the block will be read correctly. Also, the STPST bit in the TCB is examined to determine if the data read is to be transferred into the I/O buffer since some operations, such as a search, do not require the transfer of the data from the TDC buffers into the I/O buffer. Only the block address information is transferred to the tape status word (present tape position) from the first hardware buffer encountered for the block. The remaining information in that buffer and the other three buffers for that block are ignored. No information is transferred to the I/O buffer.

2.96 When the buffer is the last buffer to be serviced for a block, a CRC test is made to determine if the block was read correctly. The interrupt routine schedules the error recovery state (paragraphs 2.81 through 2.85) in the base level portion of the tape handler to obtain control at base level when a CRC failure is detected. Otherwise, the position-known bit in the tape status word is set indicating that the tape position is known and the block has been read correctly. A sequence check is made to detect block sequence errors. In addition, a range check is made to prevent a client from accessing a block in a file other than the file that is open. If the block just read contains a double end-of-file bit equal to 1, the tape is initialized so it can be rewound to the beginning of the tape at base level because the file is continued on the next track. The proper substate of the base level read state is scheduled in the TCB to process the double end-of-file condition detected. A continuous read operation is terminated when an EOF bit is encountered by the interrupt read routine.

2.97 When each complete block has been read, control is returned briefly during the interrupt to an address supplied by the tape client program in the TCB for interrupt work. The client program can perform a small amount of work (setting a flag, examining a bit in the I/O buffer, or moving the block from the I/O buffer to a save area) before returning control to the interrupt routine in CTAPH. This capability is often used in a continuous read operation. Once the interrupt routine in CTAPH regains control from the client, it sets the I/O complete bit (IOCOMP) in the TCB to indicate the tape operation is complete and transfers control to BLMMA. To end the interrupt, BLMMA then branches to a routine in CSYSUB. The system is restored to the preinterrupt state and control is returned to the location being executed at the time the interrupt occurred.

Interrupt Writing Functions

2.98 For writing operations, the base level portion of CTAPH loads the first two TDC buffers for a block to be written from the I/O buffer (paragraphs 2.70 through 2.75). After one of the hardware buffers has been written on the tape by the TDC, an interrupt is generated so that the processor can load that buffer (the third buffer). The interrupt writing routine in CTAPH loads the TDC buffer from the I/O buffer and increments the buffer counter. The interrupt is then terminated. Control is transferred to BLMMA which then branches to a routine in CSYSUB and ends the interrupt. The system (registers, I/O channel, etc) is restored to the preinterrupt state and control is returned to the location being executed at the time the interrupt occurred.

2.99 After the second TDC buffer is written by the TDC on the tape, it is ready for loading by the processor (thus becoming the fourth buffer to be loaded). Again, an interrupt is generated to load the buffer, the buffer counter is incremented by one, and the interrupt is terminated as described in paragraph 2.98.

2.100 After the third buffer is written by the TDC on the tape (the fourth buffer was loaded while the third was being written), another interrupt is generated. The purpose of the interrupt is to issue the write-stop command to the tape unit. This command primes the tape transport to stop after the fourth buffer (and thus a complete block) has been written. The buffer counter is cleared and the I/O complete flag (IOCOMP) in the TCB is set to indicate the write operation is complete. The interrupt is then terminated as described in paragraph 2.98.

2.101 If an error is detected during the write operation, a TTY error message is printed and control is returned to base level with the I/O complete flag in the TCB not set. The error is thus handled by the base level portion of the CTAPH.

TAPE CLIENTS

2.102 There are many tape clients, ie, programs which access the tape files. Most of the client programs are described in the software subsystem description on the functional area for which the program is used, such as Traffic and Plant Measurements, Translations, etc. However, some of the client functions are described in the following paragraphs.

A. Paging

2.103 Certain programs (referred to as nonresident programs) and data tables do not reside in memory but reside on tape and are brought or paged into memory only when needed. Typical programs which are paged are not real-time sensitive, do not usually execute simultaneously, and are written in such a manner to be easily separated into small segments, each segment having the ability to execute without the need of the next part.

2.104 The paging buffer is the area in memory into which nonresident programs are paged. The buffer is approximately 4,000 words in length out of which many words of program can execute, thus saving a significant amount of storage. The tape file containing segments to be paged is called the paging file (see paragraph 2.38).

2.105 Inherently, all nonresident programs are executed under the control of the multiscan function controller in the common base level monitor, CBLM. Thus, the request for a nonresident program is a request for a multiscan function and is scheduled through the multiscan function controller.

2.106 Programs are identified to the program loader as either resident or nonresident. In addition, nonresident programs are identified in such a way as to form a "tree" of programs (or segments) which identifies dependency on other segments in the tree. Programs must be laid out so that nonresident supporting programs which are needed by a program can be brought into memory at the same time as the program needing support. Refer to Fig. 20 for a typical tree structure. According to the tree structure pictured, segment 3 requires segments 2 and 1 to execute; therefore, 2 and 1 must exist in the paging buffer with 3. However, 3 is independent of 4, 5, 6, and 7. After 3 has completed execution, it may logically progress to one of the other segments, such as 4, at which time that segment and its supporting segments must be paged in.

2.107 After interpreting the supplied layout information, the program loader creates (at the time of loading) a segment table (SEGTAB) in memory. This segment table (Fig. 21) depicts the tree structure and there is a 2-word entry for each segment on the tree. The segment number is used as an index into the segment table. Each entry contains a right brother number and a logical index of the segment in the paging file on tape. The right brother number reveals the relative dependency of the segments of the tree.

2.108 The loader also automatically creates a linkage to the paging monitor, CPAGM, when it detects a transfer to a symbol in a nonresident segment. The transfer cannot be made directly since the program is not in memory and may require being paged in. This linkage is transparent to the calling and called programs. For each transfer to a nonresident program by a resident program, the loader creates the linkage and calls an ENTAB entry (Fig. 22) which resides in main store. The loader places in the transfer instruction a pointer to the location of the ENTAB entry instead of the paging buffer location value assigned to the symbol that will receive control. The ENTAB entry contains the segment number of the program containing the symbol. The ENTAB entry also contains the address of the symbol location in the paging buffer which is accessed by the paging monitor. The ENTAB entry has a BSAI (branch and save address) instruction which gives control to the paging monitor, thus providing the linkage.

2.109 In addition, the loader creates ENTAB linkages for nonresident program transfers to other nonresident programs. These ENTAB entries are linked to the nonresident segments and are also stored on the tape (thus nonresident). The ENTAB entries for each program segment are, therefore, paged into the paging buffer along with the program segment.

2.110 As already stated, the paging monitor CPAGM gains control via the call instruction (BSAI) in the ENTAB entry. A temporary area of store called the paging control block (PCB) is used by CPAGM to save registers and to store the address in the paging buffer of the symbol to which transfer is to be made. The symbol address and the segment number of the segment containing the symbol from the ENTAB entry are retrieved by CPAGM.

2.111 Using the segment number, CPAGM indexes into the paging map to determine whether the segment needed is already resident in the paging buffer. The paging map is a 16-word block of memory forming a 16 by 16 matrix that contains bits representing the status of each segment. When the bit is set, the segment is resident in the buffer; however, a zero bit implies the segment must be paged if needed. If the segment needed is already in the buffer, CPAGM gives control to the location of the symbol in the paging buffer for execution.

2.112 When the segment is not already in the paging buffer, CPAGM zeroes all the bits in the paging map after the bit for the needed segment because any segments in the paging buffer will likely to be overlayed. The segment table is then examined by CPAGM to determine any supporting segments which must also be paged along with the needed segment. All segments on which a segment is dependent appear above that segment in the segment table (that is, the segment numbers are less than that of the dependent segment). Also, the right brother numbers of any segments required by another segment are greater than or equal to the right brother number of that segment. Therefore, CPAGM compares the right brother number of each segment in the segment table above the needed segment with that of the needed segment and pages into the paging buffer all segments that are required. Also, CPAGM updates the bits in the paging map to indicate whether each segment is loaded or not loaded.

2.113 To accomplish the paging operation, CPAGM requests that the paging file be opened by the tape handler (CTAPH). It then requests the tape to be positioned at the beginning of the segment to be paged into memory (identified by the tape logical index in the segment table entry for that segment). A continuous read request is initiated to the tape handler which will read all blocks of the segment. (The last block of the segment has an end-of-file designation which terminates the read.) After each block is read, control is returned briefly at interrupt level to CPAGM which moves the block from the I/O buffer into the paging buffer at the location identified in the loading address in the tape block. This procedure is performed for each segment which must be paged. If CPAGM must take a real-time break during the paging operation, control is returned to the MSF controller.

2.114 After the segment containing the original destination and all supporting segments have been loaded into the paging buffer, CPAGM gives control to the location specified in the ENTAB entry. Normal execution will now continue. Figure 23 depicts the general flow of the paging operation.

2.115 The program CPAGM is not only entered indirectly through the linkage created by the loader but also can be explicitly entered. A client needing a nonresident data table from the tape calls a special entry point in CPAGM. The client provides the address of a 2-word data table (Fig. 24) which defines the segment number and address where the nonresident table is to be located in memory. The ENTAB entry is linked to the segment by the loader. The segment containing the data table is paged in the same manner as previously described except that control is not transferred to the data table but is returned to the client or calling program.

B. Verify Dates on Translation Files and Line Class Code Table Files

2.116 The program VRDATE is a tape client which, upon a TTY request, can print the dates of the current translation file, the two backdated translation files, or the backdated line class code table files. Program VRDATE determines the request made and issues the tape macro call to seize control of the tape control block (TCB) and to open the appropriate file at the block containing the date(s). After the file is opened and the tape is positioned at the correct block, VRDATE initiates the reading of the block (performed by CTAPH). The date(s) retrieved are then printed and the file is closed.

2.117 Any errors encountered cause the operation to be aborted and a TTY message to be printed indicating the operation was not completed.

C. Translation File Updates

2.118 The task of the office data file update program, OPDATA, is to update the backup files containing the translations for the No. 3 ESS office. The program is nonresident and is executed as a multiscan function. Program OPDATA consists of three parts:

(1) The updating of the most recent copy of translations contained in the file TRNSLN (paragraphs 2.24 and 2.25)

(2) The updating of the backdated translations in file BACKDT1 and BACKDT2 (paragraphs 2.26 and 2.27)

(3) The updating of a List 2 cartridge to a List 1 cartridge status with translations (paragraphs 2.24 and 2.25).

Update of Current Translation File

2.119 In most cases when a memory reload of translation data is required, reloading the most recent copy of translations is sufficient to restore the system. Program OPDATA is invoked automatically every 24 hours to update the current translation file, TRNSLN, with the current translations in memory and also the associated data in the CHECKSUM file. This task is initiated after the daily traffic printouts have been completed, at which time the traffic multiscan function (TRAFIC) passes control to OPDATA. This function can also be invoked by a TTY request (OP:DATA;CURR!). Program OPDATA uses a temporary storage area called a task control block in order to store information needed for the performance of the function. This task control block also contains an internal progress mark which is used after real-time breaks to determine the next routine to be executed. Real-time breaks are initiated when a function requires more time than can be allowed at a given time. The WAIT subroutine in CBLM is called to cause a base level loop delay before returning control to OPDATA for further processing.

2.120 A test is made to ensure that both tape units are in service. If either is out-of-service or in a read only mode, an error message is printed on the TTY and the update is not performed. If both tape units are in service, the next step is to examine the segment status table in memory to determine whether any recent changes have been made since the last update of TRNSLN. Whenever any part of a 4K segment of translations is changed, the bit representing that segment in the segment status table is set to 0. If no recent changes have been made, no update is necessary and the only change made to the file is to change the date word within the file to the current date. This requires formatting the date word in the I/O buffer, opening the TRNSLN file on unit 0, writing the date on the file positioned in the proper block, and closing the file. The same procedure is repeated to change the data in the file on unit 1 and a TTY message is printed to indicate that the operation has been completed.

2.121 If recent changes have occurred, both the TRNSLN file and the checksum file require updating. It is possible that the updating process could abort due to an I/O error or a request from the multiscan function controller to abort. This could leave the office with a partially updated translation file, and a subsequent memory reload would load this invalid data into main store. Two precautions are taken to prevent this from occurring:

(1) The update process is performed on one tape at a time.

(2) Before the update of TRNSLN begins, the update-in-progress bit in the first block of the checksum file is set.

Then, if the update fails to complete successfully and a memory reload becomes necessary before the problem is corrected, the bit will indicate to the system initialization routines that the other tape should be used for memory reload. Thus, the checksum file is opened on tape unit 0, the first block is read and rewritten with the update-in-progress bit set, and the file is then closed. In addition, the checksum block read is stored in a temporary save area for further reference.

2.122 The TRNSLN file is next opened on tape unit 0. The updates are done in 4K-segment increments and only those segments which have been changed are rewritten on the tape. Each 4K-segment is stored on tape beginning in a new block. Seventeen tape blocks are required to store a complete segment (16 full blocks plus 64 words of the seventeenth block). The remaining portions of blocks which are not full are zeroed. The EOF bit is set in the last block of each segment.

2.123 The first segment in memory may not begin on a 4K boundary, in which case the segment on tape will not be completely filled with translation data. Those blocks which contain no data are zeroed for the segment.

2.124 The program OPDATA accesses the translation segment status word in memory to determine whether a translation segment has been changed and must be updated. The changed segment is accessed in memory and the tape is positioned to the first block for that segment. Program OPDATA formats each block of data in the I/O buffer and formats for each block the new loading length and loading address, which identify the number of words of translation data in the block and the loading location in memory. The write operation is then initiated for the block (indicating that the EOF bit is to be set if the last block of the segment). The seventeenth block of the first segment contains the date word for the file; therefore, OPDATA must also format the present date in the I/O buffer for that block before initiating the write.

2.125 The checksum for each updated translation segment is calculated by OPDATA as each segment is written on the tape and is stored in the temporary save area for the checksum block. When the first translation segment does not begin on a 4K boundary and shares a 4K segment in memory with the generic, the checksum for that segment must be stored both in the proper generic checksum area as well as the translation checksum area. After all changed 4K-segments have been outputted to the tape, a comparison is made between the number of translation segments which existed on the tape before the update (determined from the checksum save area) and the number of segments now in translation data. When the current number of segments is less than the previous number, the excess segments on the tape are zeroed. The TRNSLN file is then closed.

2.126 The checksum file on tape unit 0 is reopened. The new checksums for the changed segments have already been calculated and saved in the checksum save area. The new segment count is stored in the proper word in the checksum block if different from the previous count. Also, the new loading length of the checksum block is calculated and stored in the block and any checksums corresponding to excess translation segments are zeroed. The update-in-progress bit is zeroed and the entire block is rewritten onto the tape. The checksum file is then closed.

2.127 The entire procedure is repeated for tape unit 1. The segment status word is then reinitialized to all 1s to indicate no recent changes have been made since the last update and a TTY message is printed indicating the update has been completed. For TTY requests, control is passed to the administrative control program, ADMCON. When OPDATA is invoked automatically, control is passed to the diagnostic control program DCON which is the next multiscan function to be performed.

2.128 When an error occurs or when the multiscan function controller requests the update operation to be aborted, OPDATA closes the currently open file. Also, an abort message is printed and the update is terminated by passing control to the appropriate location depending on whether the update request was a TTY request or automatic.

Update of Backdated Translation Files

2.129 Since it is possible that an error can occur in the up-to-date translation data, older versions of translation data are maintained in files BACKDT1 and BACKDT2 (Fig. 8) should it become necessary to reload memory. The second section of OPDATA does the updating of these files and is invoked manually by a TTY input message (OP:DATA;OLD!). No checksums are associated with the data on these files. The update is done simultaneously on both tapes because, if one of the backdated files contains invalid (partially updated) data, the other file can be used in a memory reload. The files are marked duplicated in the tape directory; therefore, any operation performed by the tape handler on one tape is automatically performed on the other tape.

2.130 The program OPDATA uses a temporary area of memory as a task control block for storing information needed for performing the update operation. In the task control block is an internal progress mark which is used after real-time breaks to indicate the next routine to be given control.

2.131 The file BACKDT1 is opened at the first block which contains the two date words for the two files. An "in-progress" response is given on the TTY if the open is successful. If unsuccessful, a "repeat-later" response is given. Program OPDATA reads and examines the date words to determine which file is the oldest. The file with a zero date word or the oldest file is to be updated with the translations currently stored in the on-line memory. The date word for the file to be updated is zeroed to indicate that an update is in progress. If the update is interrupted, a zeroed date word indicates to the system initialization programs during a reload that the file should not be used for the reload because it contains erroneous data.

2.132 When BACKDT2 is to be updated, a backspace is issued and the first block of BACKDT1 is rewritten containing the zeroed date word. BACKDT1 is then closed and BACKDT2 is opened for updating. Otherwise, BACKDT1 remains open for updating and the date word will be written when the first block is updated.

2.133 Unlike the TRNSLN file, the backdated files are not segmented into 4K segments. The translation data is transferred a block at a time into the I/O buffer from memory. In addition, all translation segments in memory are copied to tape, not just the segments which have been changed. For each block, the program OPDATA formats in the I/O buffer the loading length and loading address fields, fills the remainder of the block with translation data, and initiates the writing of the block on the tape. The last block is written with the EOF bit set.

2.134 After the last block has been written, the date word must be updated. If BACKDT2 was being updated, it is closed and BACKDT1 is opened. Otherwise, BACKDT1 is positioned to the block containing the date word. The current date is formatted in the proper word of the block and the block is rewritten on the tape. The BACKDT1 file is then closed. A TTY message is printed indicating the update operation is complete and control is given to the administrative control program ADMCON.

Update of New List 2 Cartridge With Translations

2.135 A List 2 cartridge is used to supply a current program base and patches and contains only those files shown in Fig. 3. Before a List 2 cartridge can be used to reload a system, it must be updated to a List 1 state. Update of the translation file must be requested by a TTY message (OP:DATA;NEW!), and OPDATA is given control to perform the function.

2.136 Before performing the update, OPDATA tests the status of the tape units. Detected problems cause the function to be aborted and a TTY error message to be printed. Because it is possible for the updating process to be aborted before completion, the checksum file is opened, the update-in-programs bit is set, and the file is then closed. The checksum block read is also stored in a temporary save area for later reference.

2.137 The generic file is opened next and the block address of the last block of the file is computed. The tape is positioned to the last block in the file which is read into the I/O buffer. The memory loading address for the last block of the generic is retrieved. The address of the beginning of translations is retrieved from the TDATA area of memory, and the difference between the two addresses is then computed for a relocation factor.

2.138 The translation file area on the tape is opened next. The segment status table in memory is set to all zeros so that all segments of translations will appear to have been changed and, therefore, will be copied into the translation file on tape. Program OPDATA formats the I/O buffer word-by-word with translation data from memory for writing on the tape. When a word of translations is in the proper range to be part of the master table index (MTI) and contains the address of a translation element, the relocation factor is accessed and the address is adjusted accordingly.

2.139 Each 4K segment is stored on tape beginning in a new block. Seventeen tape blocks are required to store a complete segment (16 full blocks plus 64 words of the seventeenth block). The remaining portions of blocks which are not full are zeroed. Program OPDATA formats the new memory loading length and loading address for each block. The write operation is initiated for each block and the EOF bit is set if the block is the last block of a segment. The seventeenth block of the first segment contains the date word for the file; therefore, OPDATA must also format the present date in the I/O buffer for that block before writing it onto the tape.

2.140 As each segment is written on the tape, the checksum for the segment is calculated by OPDATA and stored in the temporary save area for the checksum block. After all translation segments have been written onto tape, the translation file is closed and the checksum file is reopened. The new segment count is stored in the proper word of the checksum block. In addition, the new loading length of the checksum block is calculated and stored in the block and any checksums corresponding to excess translation segments are zeroed. The update-in-progress bit is zeroed and the entire block is rewritten onto the tape. The checksum file is then closed and the updating procedure is complete.

D. Line Class Code Table Files Updates

2.141 The COPYLC program is a tape client which copies one line class code table file to another line class code table file (Fig. 10). The file contains nonresident translation tables. The request is made by one of the following TTY input messages:

```
COPY:LCCTBL PAST:CURR!

COPY:LCCTBL BACKDT:CURR!

COPY:LCCTBL CURR:OLD!
```

The first message results in the most recent of the backdated files (LCCTBL1 or LCCTBL2) being copied into the current file (LCCTBL0). The second form of the message causes the oldest of the two backdated files to be copied into the current file. The third message results in the current file being copied into the oldest backdated file.

2.142 The updating of the line class code table file (nonresident translation tables) must accompany the updating of the resident translation files. Therefore, when it is necessary to load backdated translations into memory, the corresponding backdated line class code tables must be moved to the current file for access.

2.143 The COPYLC program is nonresident and is executed as a multiscan function request from the common base level monitor, CBLM. Both tape units must be in service and capable of writing for the update to be allowed. A TTY message is printed to indicate when any operation is aborted. Program COPYLC uses a temporary storage area for saving information and flags for the update. An internal progress mark is used to indicate the next function to be performed after a real-time break.

2.144 The LCCTBL1 file is opened at the block containing the two dates of the backdated files in order to determine the file to be used in the update process. When the task is the copying of the current file into a backdated file, the proper date word in LCCTBL1 is zeroed to indicate an update is in progress. Therefore, if the update is incomplete, the zeroed date indicates the file contains erroneous data which should not be used. Because LCCTBL1 contains the date words, the corresponding words in LCCTBL0 and LCCTBL2 are zero. When the LCCTBL0 file is being copied into the LCCTBL1 file, the date of LCCTBL2 is saved so that it is not destroyed and can be rewritten in the block on LCCTBL1 containing the date words.

2.145 The copying of one file to another is accomplished by the following steps.

(1) Open the file to be copied on tape unit 0.

(2) Position the tape to the block to be read.

(3) Read N blocks.

(4) Close the file to be copied.

(5) Open the file to be written or updated on tape unit 1.

(6) Format and write N blocks on the file to be updated.

(7) Close the file being updated.

(8) Repeat 1 through 7 until all blocks have been copied from one file to the other.

(9) Then the updated file is copied from tape unit 1 to the same file on tape unit 0.

2.146 When the current file (LCCTBL0) is being copied to one of the backdated files, LCCTBL1 is again opened at the block containing the date words. The correct date is formatted and written in the appropriate date word for the updated file. The file is then closed.

2.147 A TTY output message is printed when the update operation is complete. Control is then passed to the administrative control program ADMCON.

E. Initialization Use of the Tape

2.148 The program CIPL (common initial program loader) must access the tape in order to reload memory with the generic programs. Therefore, CIPL contains the tape access routines needed to restore the generic since CTAPH is not available in memory at the time. These routines are described in Section 233-153-130, Initialization and Processor Fault Recovery, Software Subsystem Description, No. 3 ESS.

F. Tape Utilities

2.149 Tape utilities for the No. 3 ESS are nonresident and are incorporated in the common nonresident utilities program CNRUTL. The ALW:TAPEUTIL! input message is used to request the utilities to be made available, ie, initiates paging of the program into the paging buffer in resident memory. Tape utilities use the seize macro to gain access to the whole tape; whereas, other tape clients use the open macro. Tape utilities perform such functions as dumping any block on the tape to the TTY, copying one tape to another (except track 1), auditing a tape against a standard, emergency writing onto tape from the TTY, reading a block, and storing the generic file and translation file in off-line store. In addition, the capability to change the generic is available in CNRUTL and is designated the overwrite multiscan function. Tape utilities are described in more detail in Section 254-340-082.

3. TAPE OPERATIONS SUBSYSTEM FUNCTIONAL DESCRIPTION—3E3 GENERIC

CARTRIDGE TAPE SYSTEM

3.01 The No. 3 ESS utilizes a magnetic cartridge tape unit to provide a backup memory storage facility for data used to operate and maintain the office. This unit is duplicated for reliability and is referred to as the tape data controller (TDC) unit. The cartridge tape has four tracks. Track 1 of the tape is write-protected while tracks 2, 3, and 4 may be rewritten at field locations.

3.02 The cartridge tape system performs the following functions in a No. 3 ESS:

(a) Provides a backup for system programs and translation data (and any associated overwrites) if memory reloading is required to restore the system to an operational state

(b) Provides additional backdated translation data backups if an error in up-to-date translation data necessitates reload

(c) Stores office records and associated remarks for printing when an office records verify message is received from the TTY

(d) Holds the contents of traffic registers each busy hour for later printout on the traffic TTY

(e) Stores nonresident programs to be paged and executed in the paging buffer.

3.03 The basic element of the tape system is the tape block (Fig. 25) which has a physical length of 834 16-bit words (16 bits each because the two additional parity bits for each word are not stored on tape). Between each block is an interblock gap. The block is made up of a preamble, block address, data area, cyclic redundancy character (CRC), and postamble.

3.04 The preamble and postamble are 16-bit hardware synchronization codes which are added and removed automatically by the TDC when data transfer is in progress between the 3A Central Control (3A CC) and the TDC. The preamble is a prefix while the postamble is a terminator. These two words add to the physical size of a block on the tape but are automatically removed during reading and added during writing by the TDC. They are not accounted for within the processor during the administration of the tape block, thus yielding a logical block size of 832 words.

3.05 Any block on the tape may be accessed in a random fashion for either reading (tracks 1 through 4) or writing (tracks 2 through 4 since track 1 is write-protected). The 1-word block header contains an 11-bit block number specifying its physical position relative to the beginning of tape (BOT). This enables the requested block to be located. Therefore, the tape position can be determined by counting the blocks passing over the read head without reading contents of the blocks. The block header also contains a 2-bit track designation field which denotes the binary track location (0, 1, 2, or 3) of a block.

3.06 A bit-designated end-of-file (EOF) is contained within the block header. This bit is set to one in the last block of each file to signal the termination of the file. The bit also indicates to the tape handling program that a stop be issued when a continuous read has been requested by a tape client. In other words, the last block of each file must contain an EOF bit, but more EOF or file segmentation bits can be contained within a file to further segment the file into subfiles. Certain files cross track boundaries, ie, may be continued from the end of one track to the first block of the next track. A double end-of-file (DEOF) bit is used to flag the tape handler to rewind and adjust all track designation control to the next track, whereby the client of the tape assumes control at the next logical block in its file.

3.07 There are 830 words available in each block for the data area. Formats of data stored vary depending on the file or type of data being stored.

3.08 The last word of a tape block preceding the postamble contains a check character referred to as the CRC. This 16-bit constant is used to validate the data read in or written out by the tape handler.

3.09 A grouping of a sequential number of blocks, under a given name of up to eight characters, is referred to as a file. All four tracks of the cartridge tape are completely formatted. No area between files is left void of data. Empty data areas contain blocks with all zero data areas and the correct block and track designation and CRC words. The area between the beginning-of-tape mark and the first recorded information on a track and the area between the last physical block on a track and the end-of-tape holes must be free of recorded information. An interblock gap level recording exists in those areas of the tape. The List 1 cartridge contains all files required in a No. 3 ESS office; the positions of these files are depicted in Fig. 26. A List 2 cartridge is used to supply an office with a new current program base independent of office translation files. The cartridge contains only those files shown in Fig. 27.

TAPE FILES

A. Directory

3.10 A tape directory (Fig. 4) resides on track 2 of each tape. The directory identifies the physical position of each file on the tape to the tape handling program. The directory starts in the data area of the first block of the directory file and contains a 1-word header defining the number of blocks needed for the directory and the total number of files residing in the directory. The file definitions begin immediately following this header.

3.11 Each file definition entry is made up of six words. The first four words contain the file name of up to eight American Standard Code for Information Interchange (ASCII) characters. Unused character spaces are zero. The fifth word of an entry contains the last physical block number of the file and the track designation for that block. The last word of an entry contains the first physical block number of the file, the track designation for that block, and the tape pair on which the file is located. (The No. 3 ESS uses only one pair of tape units, whereas the tape handling software has the capability to handle more than one pair.) The duplicated bit (DUP) flags the handling program so that when one block of this file is written on a tape, the mate unit is automatically written. This alleviates the necessity for a client program to issue additional orders in keeping both units up-to-date. No file entry is allowed to overflow into another block of the directory. Unused words after the last directory entry are all zero.

B. Bootstrap Files

3.12 Bootstrapping a processor consists of loading memory from the tape and using the processor and its microcode as the controlling mechanism. Layout of information on the tape is engineered for bootstrap operations. Files associated with system recovery are positioned in a sequence permitting unidirectional tape operation (ie, no searching or backspacing is required to access the next file) in case of a massive memory mutilation.

3.13 The bootstrap file, which contains code to perform bootstrap operations, begins in block 0 of track 1. Track 1 is the only track on the tape containing block 0 which is positioned approximately 36 inches from the tape load point. The bootstrap files and generic file are located on track 1 because the two inner tracks (1 and 2) of the tape are more reliably read. Track 1 is also write-protected, meaning it can only be read and not written, thus protecting the system programs. In addition, the bootstrap file is duplicated on track 2 for more reliability. The second block on track 1 (block 1) contains an all-zero data area so that the bootstrap files on tracks 1 and 2 will be synchronized after the first block. The reload process can alternate between tracks 1 and 2 in case an error has been detected. Only an error in the same block on both tracks can block execution.

3.14 The bootstrap file is segmented into two subfiles by EOF marks. The first subfile contains the common initial program loader (CIPL) and the last block of CIPL contains the EOF bit. This bit indicates that the next block on track 1 initiates the system core of programs required to continue initialization. System core programs are extracted from the generic program and are stored in the second subfile of the bootstrap file. System core programs include:

- The first 129 words of store
- CBLM — common base level monitor
- CINIT — common initialization
- CPATCH — common patch area
- CSYSUB — common system subroutines
- MASACS — main store access routines.

The last block of the second subfile also contains an EOF bit. A block containing either program or translation is illustrated in Fig. 5. Initialization and bootstrap functions are described in Section 233-153-130, Initialization and Processor Fault Recovery, Software Subsystem Description, No. 3 ESS.

C. Checksum File

3.15 The checksum file is read during memory reload operations and is updated during patching and recent change operations. It contains miscellaneous system parameters needed to perform reload operations and hashsums used to determine which 4096-word (4K-word) segments of memory are to be reloaded. A hashsum is a logically derived (from an algorithm), data-dependent word whose generation is based on the information stored in each 4K-word block. Therefore, during a memory reload, the hashsum of each 4K segment of memory beginning on a 4K boundary can be calculated and compared with the hashsum stored in the checksum file. If the hashsum calculated for a 4K segment of a generic program store does not match the corresponding hashsum read from the checksum file, that 4K segment of generic program is reloaded into memory. Although similar hashsums exist for translation store, they are not used during a memory reload in the No. 3 ESS.

3.16 The checksum file is adjacent to the backup bootstrap file on track 2 and is accessible for update. The CIPL program accesses the file to recover the system parameters needed to pinpoint other files required for bootstrap operations.

3.17 See Fig. 28 for the format of the checksum file. Word 0 contains the block address, track number, EOF bit, and DEOF bit. Words 1 and 2 contain the field length (number of words comprising the used portion in the block) and the 20-bit value of the symbol CHECKSUM. Word 3 contains an update-in-progress bit (which is set when the translation file is being updated) and the starting block number of the generic file. The starting block number of the translation file is in word 4. Word 5 contains the number of hashsum entries for the translations stored on track 2. Word 6 specifies the starting block and track number of the patch file which contains overwrites to the generic program. The number of 4K program store segments containing the generic program and its related temporary memory is in word 7.

3.18 Following the header words is a 2-word entry containing hashsum information related to the generic area of main store. The first word relates to the number of MAS words for the hashsum entry. Each hashsum entry is associated with 4096 words of generic or translation text. Hashsums are only relevant for these two types of main store data. The second word contains the hashsum value that is generated by the hashsum algorithm.

3.19 Every 4K-word segment within the generic area of main store has a 2-word hashsum entry reserved for it. If the segment contains temporary storage, the hashsum value is zero. However, the temporary store flag (TSFLG) in the first word of the related hashsum entry is set because no space is reserved in the generic file for temporary store segments. These segments must be indicated to the bootstrap program for the reloading functions.

3.20 After the generic hashsum entries, a 2-word entry exists of which the first word is unused. The second word contains the number of 4096-word main store segments of translations. Hashsum entries for the translations follow. The checksum file is one block in length.

D. Generic File

3.21 The generic file resides on track 1 (which is write-protected). This file contains only the program portion of the generic. No temporary memory is stored on the tape. The generic text is stored in the manner described in Fig. 5. The file is segmented into 4K word segments. Each segment is five blocks in length. The first four blocks contain 828 words and the fifth block contains 784 words and the segmentation bit. The first two words in the data area of each block (Fig. 5) specify the loading length and address for each block in memory. Following the length and address are 16-bit words of the generic.

E. Current Translation File

3.22 The translation (TRNSLN) file begins after the generic file but resides on track 2. It is not marked duplicated in the directory. The general format of a translation block is the same as that for a generic block (Fig. 5). The data is formatted the same way as the generic file (ie, five block logical files using the segmentation bit as the dividing mechanism). Translations will always start with the first word of a 4096-word block of main store (4K boundary).

3.23 There are 150 blocks in the file which can handle up to 122,880 words of translation. Many offices may not initially fill up all 150 blocks; therefore, the remaining unused file is formatted with all zero data area blocks. Also, the last block must contain a file segmentation bit.

F. Backdated Translation Files (BACKDT1 and BACKDT2)

3.24 If the PAST OFFICE DATA or BACKDT OFFICE DATA key on the system status panel is pressed, an initialization program will load backdated translations from either the BACKDT1 translation file or the BACKDT2 translation file. Figure 8 depicts the layouts of each of these files.

3.25 Block 0 of BACKDT1 differs from block 0 of BACKDT2 in that it contains two date words (words 1 and 2). These words contain a packed representation of the date of the last update of BACKDT1 and BACKDT2, respectively. These date words are used by the initialization program to determine which of the two files contains the more recent version of translations. Therefore, when the PAST OFFICE DATA key on the system status panel is pressed, the file containing the more recent backdated version of translations is loaded into memory. When the BACKDT OFFICE DATA key is pressed, the file with the less recent backdated version of translations is loaded. If one of the date words is zero, the file corresponding to the nonzero date word is used to reload memory. The files BACKDT1 and BACKDT2 are marked duplicated in the directory indicating to the system that a block written by a client on one tape should be automatically written on its mate tape if it is in service.

G. Traffic File

3.26 An operating company can choose up to 23 busy hours for which traffic registers can be saved for later printout. The traffic registers are saved at the end of each busy hour on a tape file, TRAFFIC. At the scheduled daily time, the tape is positioned at the beginning of TRAFFIC, and the register contents for each busy hour are read from the file and printed on the traffic TTY.

3.27 Figure 11 depicts the layout of a block of TRAFFIC data. If word 1 (the header) of the block is 0, then the block starts a new busy hour (data for a new busy hour is always started in a new block). For a new busy hour, the three words following the zeroed header contain information relative to the busy hour. If the header is nonzero, it contains a name field identifying the type of traffic data which follows it and a length field with the number of words of traffic data to follow. A nonzero header is followed immediately by the traffic data. The traffic data itself never crosses a block boundary, ie, every block begins with a header word immediately following the block address. A zero header other than the first header in the block indicates that there is no more traffic data in the block. The TRAFFIC file is marked duplicated in the tape directory. The processing of the traffic file is described in Section 233-152-135, Traffic and Plant Measurements.

H. Patch File

3.28 All patches or revisions to the generic file on track 1 are applied by placing the revisions in the patch file on track 2 immediately following the translations file. This file is also accessed during a reload of the generic. Refer to Fig. 12 while reading the following description of the patch file. The areas described are identified in the figure by the letters.

A. Block address word.

B. Size of the patch file header.

C. Area designated as the patch file header area. These words appear only in the first block of the file.

D. This word is set to octal (177777).

E. An 8-byte area reserved for the generic identifier which is a maximum of eight ASCII characters. Unused character space is equal to zero.

F. An 8-byte area reserved for the generic issue identifier (a maximum of eight ASCII characters). Unused character space is zero.

G. A length indicator covering all the 16-bit words used to store a given patch. It is a logical index to the next patch.

H. A 14-bit field which contains the patch number assigned by Bell Telephone Laboratories.

I. A 2-bit field equal to 1.

J. Three 16-bit words set to zero.

K. A length field specifying the number of consecutive words patched at the starting address in the M field.

L. A 12-bit field containing the segment number designation for the words patched at location M. Fields L, M, N, O, P, and Q are given in the overwrite input message.

M. The address (20 bits) of the first location involved with this segment of the patch.

N. The low 16 bits of the new data.

O. The high 8 bits of the new data which is always zero for the No. 3 ESS.

P. The high 8 bits of the old data which is always zero for the No. 3 ESS.

Q. The low 16 bits of the old data.

3.29 The N through Q fields are repeated for each consecutive word patched starting at location M. When a new address is required due to a break in the continuity of patched locations, a K entry is started. If no more words exist in the patch, a new patch is started with a G field. The location following the last used Q entry is set to octal (177777) as an internal logical EOF indicator. All unused words after the logical EOF mark and all unused blocks following the block containing this file mark contain all-zero data areas. The last block of the patch file contains the file segmentation, or EOF, bit.

I. LABEL File

3.30 The file designated LABEL (Fig. 13) contains the generic identification. The file resides on track 2 after the directory and is one block in length. The first word of the block contains the normal block address header. The second word indicates the number of ASCII characters in the following words that are used to identify the generic. There are two characters in each word. Following the last word containing the generic identifier is the number of ASCII characters required to identify the issue number of the generic. The words containing the issue number identification characters follow with all remaining unused words in the data area of the block being zero.

J. Paging File

3.31 The paging file contains nonresident program segments for the No. 3 ESS. The file resides on track 4. Segments vary in length but each segment begins in a new block and is terminated with a block containing a file segmentation bit. The program data is stored in blocks of the same format as shown in Fig. 5.

K. Office Record Forms File

3.32 The office record forms file resides on track 3. The file contains office record forms and comments to be used in printing the office records on the TTY. The blocks are formatted in the normal manner with the forms and comments stored in the data area of the blocks.

TAPE HANDLER FUNCTIONS

A. Relationship of Tape Handler to Tape Client Programs

3.33 The tape data controller buffers data between the tape unit and the 3A CC in two switchable hardware buffers so that the processor is not required to engage in continuous communication with the tape unit. Each hardware buffer can store 64 words or 1024 bits. Therefore, 13 buffers must be filled in order to read or write a full block of 832 words from the tape. The tape unit and processor have access to only one of these buffers at a time. The buffer referred to as the "off-line" buffer is accessed by the processor; while the "on-line" buffer is accessed by the tape unit. The buffer that is on-line becomes off-line and vice versa after the tape unit has serviced the buffer. Since the processor is significantly faster in its ability to service one of these buffers than the slower tape unit, a time interval exists between the processor servicing of one buffer and the tape unit flagging the processor that the next successive buffer is ready. The buffers must be serviced within a specified time to prevent an "overflow" condition; therefore, the buffers are serviced during interrupts generated by a buffer being ready for servicing (loading or unloading).

3.34 There is also a software buffer, referred to as the input/output (I/O) buffer, in main store which has a 832-word storage capacity. The data to be written on the tape is stored in the I/O buffer and then accessed to fill the hardware buffers with the bits to be written onto the tape. In reading, the bits are transferred from the hardware buffers to the I/O buffer in memory to be made available to the using software.

3.35 Programs which read or write tapes use tape macros to supply the tape operation required and gain access to the common tape handler program, CTAPH. The tape macro names and their major functions are shown in Table B. The tape "client" programs, in conjunction with the macros, supply information needed for the tape operation in the tape data control block (TCB), a 16-word block of memory reserved for tape handling software (Fig. 14).

3.36 The tape handler program, CTAPH, initiates the hardware commands to the tape data controllers and buffer units necessary to perform the tape operations requested by the client programs. These hardware commands are described in Section 254-300-170, Tape Data Controller, Description and Theory. The general format of the command word is depicted in Fig. 15. In most cases, CTAPH provides the command word and then calls a subroutine in the common systems subroutines program, CSYSUB, to actually issue the I/O order and monitor the response.

3.37 The tape client programs are scheduled under control of the multiscan function (MSF) controller in the common base level monitor, CBLM. In essence, a client program seizes control of the TCB (if the TCB is idle) and makes the tape operation request via use of the tape macros. Only one client program may access the tapes at any given time. The identifying number of the client is stored in the TCB as well as the request bits identifying the tape operation requested. The address to which return is to be made after the tape operation is completed by CTAPH is also stored in the TCB.

3.38 The client programs must schedule each tape operation to be performed and access the macros to provide the interface to CTAPH for those operations. For example, if a client is reading blocks at random in a given file, first the OPEN macro is used to seize control of the TCB and to position the tape heads at the beginning of the file needed. The POSITION macro is used for positioning the tape at the block to be read. After the tape is positioned, the client uses the RD1BLK macro for the read operation. Then the tape is positioned at the next block to be read, and so on. The client must use the CLOSE macro to relinquish control of the TCB after all needed blocks have been read.

B. Organization of the Tape Handler Program

3.39 The common tape handler program, CTAPH, is a resident program. One portion of the program is executed during base level, and a second portion is given control during interrupts generated when a hardware buffer of the tape data controller is ready to be loaded or unloaded. Also included in CTAPH are subroutines which process various TTY input messages concerning the tape system. For instance, one routine services the tape unit restore message. This message restores the tape unit to service after checking to see if it passed diagnostics and a tape-to-tape audit (unless the request was unconditional). The remove-tape-from-service message is serviced by another subroutine. There is an emergency TCB cleanup routine, which is accessed with a TTY input message in the event temporary store gets randomly written and the TCB contains erroneous information.

3.40 The base level portion of CTAPH is given control by the common base level monitor, CBLM, once every complete base level loop (approximately once every 200 ms). The base level portion essentially examines the TCB for client requests for tape action and schedules the various functions required to perform these requests. It is the interface between the client and the tape. The interrupt portion is entered at interrupt level for loading and unloading of the tape unit hardware buffers in the reading and writing operations. For instance, in the writing operation, the base level portion issues the write command and fills the first two hardware buffers. When the tape data controller has emptied one of the buffers and has written it on the tape, the buffer ready flag is set. This causes an interrupt with control being given to the interrupt portion of CTAPH. Then CTAPH, during the interrupt, loads the empty buffer and the process is continued until 13 buffers of information required to write a tape block have been filled and written on the tape. The base level portion of CTAPH monitors the operation until it is complete and then schedules the next function. Figure 16 depicts the client, tape handler, and tape interfaces.

3.41 The requested tape operation of the client may be aborted (halted) prematurely before completion because of various types of trouble encountered. The trouble may include operation time-outs, system initialization, etc. The TCB is cleared and control is returned to the client with an abort return code. If the operation is aborted because of a request from the MSF controller, the client cleans up and returns control to the MSF controller.

C. Base Level Portion of the Common Tape Handler

3.42 Upon entry to the base level portion of the common tape handler, CTAPH, the number of tape units in the office is determined. Also, if CTAPH determines diagnostics are in progress (by examining the proper bit in the TCB), control is returned immediately to CBLM. Otherwise, CTAPH accesses the tape status area in memory (Fig. 17), which contains status and position information for the two tape units, to determine if either unit is out of service. There is one word for each unit.

3.43 The program CTAPH sends a status command to the tape units to further determine the present status of the units. Figure 18 depicts the format of the tape status command response. When trouble is discovered (no response from the tape unit, etc), the tape unit is removed from service. Removal from service includes the printing of a TTY message, stopping tape movement if it is in motion, and setting the status indicator lamp on the system status panel. If the tape unit is also active in the TCB, the TCB is cleared and CTAPH returns control to CBLM since there is only one TCB and, therefore, no more tape work to be performed at that time.

3.44 In addition, a buffer unit status command is sent to check the status of the hardware buffer units. Figure 19 depicts the format of the buffer status command response. When an error is found in the buffer, the tape unit is removed from service as previously described.

3.45 After the status of each tape unit has been determined and updated, CTAPH examines indicators in the TCB to schedule tape functions. These indicators include the request bits, the state word, and the state "flags" word and are used for dispensing tasks.

3.46 When a client program makes a request, the request bits are supplied in the TCB to indicate the major requested operation such as:

a. Read one block

b. Write a block

c. A continuous read

d. Position the tape

e. Close a file

f. Copy a file from one tape to its mate unit

g. An entry caused by initialization.

Upon the first entry to CTAPH after the request is made, control is transferred to the major entry point corresponding to the request. This sets up further control to perpetuate the new task. Scheduling of the functions required to perform the requested tape operation is accomplished by the use of states or substates specified in the TCB which correspond to a section of code, or module. The state word in the TCB can "stack" up to four state indicators (numbers corresponding to the state). The term "stacking" refers to the procedure whereby one state can request another state to take control and can push itself down in the stack so that it will regain control when the previous state has been completed. It can also remove itself so it will not regain control. When CTAPH gets control from CBLM at base level, the value in the first state word location is used to transfer to the state which is in control.

3.47 The various states analyze functions to be performed and may schedule other states or substates to help complete the function. The substate indicators are stored in the state flag word in the TCB and indicate to the state how far the operation has progressed. The flag for a state is stored in four bits of the flag word that corresponds to the bit location of the state in the state word. That is, when a state is pushed up or down in the state word stack, the corresponding substate or flag is also moved. The state, when it gains control, uses the state flag to determine the action to be taken next within the state.

3.48 The states and actions taken by the base level portion of CTAPH are described without consideration for real-time breaks and interrupts.

Monitor State

3.49 The monitor state is entered whenever the state word in the TCB is zero. This can occur when the TCB is not under client control or when a specific client request has been completed. In either case, the TCB is ready to service a new request from the client. The monitor state transfers control, based upon the request bits, to the initial entry point corresponding to the request. The entry routine initializes for the function and supplies the next state to be given control. Request bits equal to 7 indicate the TCB is idle and awaiting a new request.

Timing State

3.50 When more time is required for a function to be completed than can be allowed at that time, a real-time break must be taken. If the time required is less than a base level loop, control is returned to CBLM. The state and substate stored in the TCB will regain control when CTAPH is given control again by CBLM on the next base level loop. However, if more time is needed, the timing state may be used. The state in CTAPH in control can push itself down a level in the state word stack of the TCB and invoke the timing state. The time required is stored in the timing word of the TCB. The timing state then is given control each base level loop until the correct amount of time has elapsed. The timing state will then remove itself from the stack, and the original state and flag will be rotated to the first position and placed in control.

3.51 When a time-out occurs during reading or writing before the operation is complete, an error has occurred and control is given to error recovery routines.

Seizure State

3.52 The OPENTCB subroutine in CTAPH is given control when one of the open macros or the seize macro is used by a tape client. The subroutine determines whether the TCB is idle (that is, no client is in control—client number equals zero). When the TCB is busy, a busy return code is given to the client. Otherwise, the subroutine seizes the TCB for the client and loads the TCB with client request information. The seizure state is loaded into the TCB state word to obtain control. The seizure state performs the following functions:

(a) Determines the tape unit to be accessed (which is the tape unit specified in the macro call or, if one is not specified, the tape unit numerically tied to the on-line processor if available).

(b) Determines the status of the tape units.

(c) Accesses the portion of the directory resident in main store to search for the block containing information on the file of interest.

(d) If the file is not in the resident portion of the directory, invokes the position state to position the tape at the beginning of the tape directory and searches the directory for that file by using the file name.

(e) Uses the position state to position the tape either at the beginning of the client file or at the block index (logical block number) in the file specified by the client. Aborting of the client will be made if the seizure state locates a fatal error.

Copy State

3.53 The copy state is responsible for reading a file off one tape and writing it onto its mate tape. First, the low bit position of the tape unit number to be copied is complemented to provide the number of the mate unit to be written. Then the status of the mate unit is checked. Next, by use of the position state, the mate unit to be written is positioned to the file and block to be copied. Again, the tape unit number is complemented to provide the tape to be copied, and the command to read one block from that tape is issued to the hardware. After timing for the read command, the substate to write the block on the mate unit is invoked. The substate complements the tape number to provide access to the mate unit. The substate also causes the write state to be executed and actually perform the functions and issue the orders required to write the block on the mate unit. (Refer to paragraphs 3.63 through 3.68 for these functions).

3.54 The copy state is used to copy a file from one tape to another tape. Therefore, after each block is written, the copy state determines whether the block just written was the last block in the file to be copied. In this case, control is returned to the client. Each block to be copied is processed as previously described.

Position State

3.55 The position state is used to position the tape at the logical block in the file requested in the TCB. However, if the seize state invoked the position state, the beginning of the directory is the goal.

3.56 The "position-OK" bit in the tape status word (Fig. 17) is examined to determine if the present position of the tape is known. The present position stored in the status word is not dependable if the tape is performing or has just completed a fast search, in which case the tape unit must be stopped and the position verified. The verify state is used to verify the position of the tape and to ensure that the tape is positioned at an interblock gap.

3.57 Once the position of the tape is known, the positioning state determines the distance between the present tape position and the goal. Depending on the distance between the two positions, the position state initiates either a fast search or slow search in the appropriate direction (forward or reverse). A fast search consists of calculating the approximate amount of time required to move the tape the number of blocks needed, storing the time in the TCB, and issuing either the fast reverse or fast forward command. The timing state is given control to perform timing functions. Verification must also be performed by the verify state after each fast search.

3.58 A slow search consists of reading or backspacing one block at a time. The timing for the slow search is also controlled by the timing state. For both fast and slow searches, a bit in the TCB is set to 1 to inhibit the transfer of data to the I/O buffer by the interrupt during reading. Only the block number is stored directly into the present tape position in the tape status word.

3.59 This procedure is continued until the tape is positioned at the block requested. Only one fast search (if needed) is attempted, but that is usually all that is required because the timing algorithm is so accurate. The slow search can then be used to position the tape at the exact block requested. Control is then returned to the client with the tape positioned at the block requested.

Verify State

3.60 The verify state places the tape heads at an interblock gap and verifies the actual tape position. Verification is performed when the position of the tape is unknown, such as after a fast search. The verify state attempts to verify the position of the tape on the track on which the file of interest resides. Verification consists of issuing a backspace command, which is defined as passing through data and stopping at the next interblock gap, and then issuing a read-one-block command. Nothing is stored by the read interrupt in the I/O buffer but the block number is stored directly into the present tape position in the tape status word.

3.61 When a failure occurs in reading, the verify substate will attempt to verify the position of the block on that track twice by backspacing and reading the block. If this procedure fails, an attempt is made to switch to the next track at the same position and verify the position of the block (block number) on that track. This is done for each track until the position is known or all tracks have been accessed. After all tracks have been used and verification has not been successful, the other tape unit will be used if available and if the client permits it.

3.62 The tape operation is aborted if verification is unsuccessful or if verification is successful only after a recovery and the present tape position is equal to the block needed by the client. A TTY message is printed indicating the fatal read error and a repacking of the tapes is scheduled. The client is aborted and the TCB is idled.

Write State

3.63 When a client requests a write operation, the first write entry routine determines from the directory information stored in the TCB whether the file is to be duplicated on its mate unit. In this case, the status of the mate unit is obtained. If the mate unit is in service, the unit is initialized and each block will be written on both tapes. Otherwise, the block is to be written only on the one tape.

3.64 The write state is scheduled to be given control of the writing sequence. The tape must have been positioned at the block to be written by the client, and the position must be known before the write operation is allowed. The client has also loaded the I/O buffer with the information to be written.

3.65 For reliability, track 1 is used as a reference point in the writing operation; the track prevents tape skewing, is write-protected, and never changes. A backspace and a read off of track 1 are performed to align the tape at the proper position for writing.

3.66 If an absolute write request is not made by the client, the write state also formulates in the I/O buffer the block header including the block address. It automatically sets the EOF bit if the block being written is the last block of the file. The write state ensures that the block to be written is within the range available to the client file. The CRC value is computed by this state for the block to be written and is stored in the I/O buffer.

3.67 The write state sends the order to prime the TDC buffer to receive words and loads the 64-word off-line buffer from the I/O buffer. The off-line buffer is then switched to on-line and vice versa so that the second buffer can be loaded. After the second TDC buffer is loaded, the order to start writing the block is issued. The remaining buffers required to write a block are loaded during interrupts generated by the completion of writing a buffer onto the tape. (See paragraphs 3.91 through 3.94 for the interrupt functions.) The base level write state waits until the interrupt operations are complete or a time-out occurs. When control of the operation is returned to base level, a check is made for a CRC error. If an error exists, control is given to the maintenance recovery state to process the error. (See paragraphs 3.74 through 3.80.)

3.68 If the file is to be duplicated, the same write state sequence is performed for the mate unit. After the writing is complete, the original tape unit number is placed in the TCB and the TCB reverts to the monitor state (idle) waiting for the client to issue another order.

Read State

3.69 The initial entry code in CTAPH for a read request from a client sends a tape read order to the tape unit defined in the TCB. The order is either a continuous read order, which reads each block until an end-of-file is detected, or a read-one-block order, which reads only the block at which the tape is positioned.

3.70 A delay is initiated to allow the interrupt to gain control once a tape data controller buffer has been filled. The interrupt level code will further delay the read state at base level from gaining control until either the total read operation is complete or an abnormal situation arises. For a continuous read, the operation is complete when an EOF mark is encountered. For a read-one-block command, the operation is complete when the 13 buffers of data required to store one block have been transferred to the I/O buffer.

3.71 Once the interrupt level code stops delaying the base level, the read state gains control to investigate the success of the interrupt in handling the read operation. The read state interrogates the I/O complete bit in the TCB to assure the read was performed without error. Upon a successful read, the state word in the TCB is emptied so that new client requests can be serviced and control is returned to CBLM.

3.72 If an error occurs in the read, a tape status command is sent. No response causes the tape unit to be removed from service and the client is aborted. Otherwise, a TTY message is printed and the error recovery state is initiated (paragraphs 3.74 through 3.78).

3.73 The read state also has the responsibility to rewind the tape to the beginning-of-tape if a file continues on another track. The interrupt level code detects the double EOF bit set in the last block read and causes the correct substate of the read state to be given control. The rewind command is given and, after tape motion has ceased and the tape is positioned at the beginning-of-tape, the next track is placed in the TCB. If a continuous read was being performed, the read is restarted on the new track; otherwise, the read state is taken out of the state word leaving the TCB in the idle state. Control is returned to CBLM.

Retry State (Error Recovery)

3.74 The retry state is used in error recovery situations. Detection of a CRC error during the reading or writing operation (indicating data was not read or written correctly) stimulates recovery attempts. The maintenance bit and the search failure bit are set in the TCB to indicate the CRC error and that error recovery is in progress. Key control words in the TCB are saved for restoral after error recovery. The retry state is placed in the state word of the TCB.

3.75 The retry state invokes the verify state to attempt to reread the block for detection of any transient errors. The verify state has an error recovery routine which backspaces and attempts to read the block twice. If the error remains after the second attempt, control is returned to the retry state for more error recovery attempts.

3.76 When the error has occurred during a read operation and the two additional attempts to read the block have failed, the retry state increments the traffic fatal read counter and initiates a TTY message indicating the read error. The retry state will attempt to restore the bad block from the other tape unit if possible. However, this recovery attempt is not made when prohibited in the TCB, when the file is not duplicated on the mate unit, when the block to be recovered is on track 1 (track designation in TCB=0), or when the other tape unit is out of service. In addition, an attempt is made to read the block on track 1 immediately preceeding the bad block in order to position the tape at the beginning of the bad block; therefore, if unsuccessful, the block cannot be recovered.

3.77 Permanent read errors (errors which cannot be corrected) cause a major alarm to be given, a TTY message to be printed, and the read operation to be aborted. The tape unit is not removed from service since other files on the tape may be unharmed.

3.78 When recovery can be attempted from the other tape unit, the position state is used to position the mate unit at the bad block. The block will then be read off the mate tape and (if successful) rewritten onto the original tape having the error. If either of these operations fails, the error cannot be recovered. However, if recovery of the block is successful, the TCB information saved in the maintenance save area is restored to the TCB, the error bits in the TCB are cleared, and the client read request is reinitiated. Another read error can occur on that block, in which case the error reovery process is repeated. Therefore, the client program times the operation requested to prevent an error loop from occurring.

3.79 If the error was incurred during a write operation, the retry state will attempt to read the block written twice and then will attempt to rewrite the block if the error persists. If the rewrite fails, the error cannot be recovered. A TTY message is printed and the client is aborted. In addition, the traffic fatal write peg count for that tape unit is incremented.

3.80 If the write recovery is successful, a TTY message is printed indicating the success and the TCB information is restored from the maintenance save area so that normal operation can continue. Control is then returned to CBLM.

Close State

3.81 When tape operations are finished, the client must make a CLOSE request to relinquish control of the TCB. This results in the close state being given control. In addition, the tape handler, CTAPH, uses the close state code to abort some tape operations when the operation cannot be completed. The close state clears the TCB but saves the bit indicating repack requests pending. The close state then indicates the TCB is idle by placing the value 7 in the request bits and restores the repack request bit to the TCB. A pending repack request will be processed the next time CTAPH is given control at base level by CBLM to perform tape tasks and the TCB is idle. (There is no client number in the TCB and there is a 7 in the request bits.) Control is then returned to CBLM.

Pack Tape State

3.82 Repacking of the tape is performed automatically once a day or upon request. The packing process realigns the tape in a flat horizontal plane. Repacking is done only when the TCB is idle (ie, client number=0 and request bits=7) and the repack request bit in the TCB is set to 1. All client requests are inhibited during the repacking process by setting the "request stop" bit in the TCB.

3.83 Repacking the tapes is accomplished by the pack tape state. Repacking consists of initializing the tape units and fast forwarding and fast reversing of the tapes so that both ends of the tapes are reached, ie, the beginning-of-tape and the end-of-tape. A TTY message is printed indicating the completion of repacking the tapes. The close state is then scheduled to idle the TCB and to allow more requests.

Initialization Entry (MRF State)

3.84 A zero in the request bits of the TCB means an initialization of sufficient magnitude has resulted in the clearing of the call store TCB or a TTY request has resulted in the initialization of the tape system (an idle TCB has the value 7 in its request bit). The tape units are initialized and rewound to the beginning of the tapes to prepare for a bootstrap initialization, if needed. The maintenance reset function (MRF) initialization state performs and monitors the rewinding of the tape units and places the tape system in a known state ready for client use. A tape unit indicating trouble is removed irom service.

D. Interrupt Portion of the Tape Handler

General

3.85 An interrupt is generated by the tape unit when one of the two tape buffers is ready to be loaded or unloaded by the processor. The buffers must be serviced within a short interval to prevent an overflow condition and thus require servicing during interrupts. The interrupt portion of the tape handler (CTAPH) is given control. A check is first made to determine whether the interrupt was generated by an in-service tape unit and thus is valid. The buffer status is then tested to determine whether the tape unit is reading or writing so that control can be given to the proper routines. If the buffer is in the unload state, a reading operation is in progress; if it is in the load state, a writing operation is in progress.

Interrupt Reading Functions

3.86 In a reading operation, the interrupt is generated after the TDC has completed loading one of the tape buffers from the tape. The buffer is then ready for unloading by the processor. The interrupt reading routine in CTAPH must determine which buffer is being unloaded (13 buffers are required for reading a complete block). A counter is, therefore, maintained to identify the number of buffers serviced for a particular block.

3.87 An interrupt is generated each time a TDC buffer must be serviced by the processor. The servicing of a TDC buffer consists of unloading the buffer and storing the words in the proper area of the I/O buffer, incrementing the buffer counter, and timing to detect possible hardware errors in the unloading process. (If an error is encountered in an interrupt command, a TTY error message is printed and the close state is scheduled to obtain control at base level to abort the client.) In addition, the interrupt routine resets the timer in the TCB to extend the real-time break for base level so that the base level tape handler routine will not see a time-out until the complete operation has been performed (a whole block has been read). For a continuous read, the timer is continuously extended until all the blocks are read and the EOF bit is encountered. Each interrupt is terminated by passing control to the application base level monitor (BLMMA) which branches to a routine in the common system subroutines program (CSYSUB) to end the interrupt. The system (registers, I/O channel, etc) is restored to the preinterrupt state and return is made to the program being executed at the time of the interrupt.

3.88 When a buffer is the first buffer to be unloaded for a block, the tape status word (Fig. 17) is updated with the present tape position (block number). The position-known bit is not set at this time since it is not known whether the block will be read correctly. Also, the STPST bit in the TCB is examined to determine if the data read is to be transferred into the I/O buffer since some operations, such as a search, do not require the transfer of the data from the TDC buffers into the I/O buffer. Only the block address information is transferred to the tape status word (present tape position) from the first hardware buffer encountered for the block. The remaining information in that buffer and the other buffers for that block are ignored. No information is transferred to the I/O buffer.

3.89 When the buffer is the last buffer to be serviced for a block, a CRC test is made to determine if the block was read correctly. The interrupt routine schedules the error recovery state (paragraphs 3.74 through 3.78) in the base level portion of the tape handler to obtain control at base level when a CRC failure is detected. Otherwise, the position-known bit in the tape status word is set indicating that the tape position is known and the block has been read correctly. A sequence check is made to detect block sequence errors. In addition, a range check is made to prevent a client from accessing a block in a file other than the file that is open. If the block just read contains a double end-of-file bit equal to 1, the tape is initialized so it can be rewound to the beginning of the tape at base level because the file is continued on the next track. The proper substate of the base level read state is scheduled in the TCB to process the double end-of-file condition detected. A continuous read operation is terminated when an EOF bit is encountered by the interrupt read routine.

3.90 When each complete block has been read, control is returned briefly during the interrupt to an address supplied by the tape client program in the TCB for interrupt work. The client program can perform a small amount of work (setting a flag, examining a bit in the I/O buffer, or moving the block from the I/O buffer to a save area) before returning control to the interrupt routine in CTAPH. This capability is often used in a continuous read operation. Once the interrupt routine in CTAPH regains control from the client, it sets the I/O complete bit (IOCOMP) in the TCB to indicate the tape operation is complete and transfers control to BLMMA. To end the interrupt, BLMMA then branches to a routine in CSYSUB. The system is restored to the preinterrupt state and control is returned to the location being executed at the time the interrupt occurred.

Interrupt Writing Functions

3.91 For writing operations, the base level portion of CTAPH loads the first two TDC buffers for a block to be written from the I/O buffer (paragraphs 3.63 through 3.68). After one of the hardware buffers has been written on the tape by the TDC, an interrupt is generated so that the processor can load that buffer. The interrupt writing routine in CTAPH loads the TDC buffer from the I/O buffer and increments the buffer counter. The interrupt is then terminated. Control is transferred to BLMMA which then branches to a routine in CSYSUB and ends the interrupt. The system (registers, I/O channel, etc) is restored to the preinterrupt state and control is returned to the location being executed at the time the interrupt occurred.

3.92 After the second TDC buffer is written by the TDC on the tape, it is ready for loading by the processor (thus becoming the fourth buffer to be loaded). Again, an interrupt is generated to load the buffer, the buffer counter is incremented by one, and the interrupt is terminated as described in paragraph 3.91.

3.93 After the twelfth buffer is written by the TDC on the tape (the thirteenth buffer was loaded while the twelfth was being written), another interrupt is generated. The purpose of the interrupt is to issue the write-stop command to the tape unit. This command primes the tape transport to stop after the thirteenth buffer (and thus a complete block) has been written. The buffer counter is cleared and the I/O complete flag (IOCOMP) in the TCB is set to indicate the write operation is complete. The interrupt is then terminated as described in paragraph 3.91.

3.94 If an error is detected during the write operation, a TTY error message is printed and control is returned to base level with the I/O complete flag in the TCB not set. The error is thus handled by the base level portion of the CTAPH.

TAPE CLIENTS

3.95 There are many tape clients, ie, programs which access the tape files. Most of the client programs are described in the software subsystem description on the functional area for which the program is used, such as Traffic and Plant Measurements, Translations, etc. However, some of the client functions are described in the following paragraphs.

A. Paging

3.96 Certain programs (referred to as nonresident programs) and data tables do not reside in memory but reside on tape and are brought or paged into memory only when needed. Typical programs which are paged are not real-time sensitive, do not usually execute simultaneously, and are written in such a manner to be easily separated into small segments, each segment having the ability to execute without the need of the next part.

3.97 The paging buffer is the area in memory into which nonresident programs are paged. The buffer is approximately 12,000 words in length out of which many words of program can execute, thus saving a significant amount of storage. The tape file containing segments to be paged is called the paging file (see paragraph 3.31).

3.98 Inherently, all nonresident programs are executed under the control of the multiscan function controller in the common base level monitor, CBLM. Thus, the request for a nonresident program is a request for a multiscan function and is scheduled through the multiscan function controller.

3.99 Programs are identified to the program loader as either resident or nonresident. In addition, nonresident programs are identified in such a way as to form a "tree" of programs (or segments) which identifies dependency on other segments in the tree. Programs must be laid out so that nonresident supporting programs which are needed by a program can be brought into memory at the same time as the program needing support. Refer to Fig. 20 for a typical tree structure. According to the tree structure pictured, segment 3 requires segments 2 and 1 to execute; therefore, 2 and 1 must exist in the paging buffer with 3. However, 3 is independent of 4, 5, 6, and 7. After 3 has completed execution, it may logically progress to one of the other segments, such as 4, at which time that segment and its supporting segments must be paged in.

3.100 After interpreting the supplied layout information, the program loader creates (at the time of loading) a segment table (SEGTAB) in memory. This segment table (Fig. 21) depicts the tree structure and there is a 2-word entry for each segment on the tree. The segment number is used as an index into the segment table. Each entry contains a right brother number and a logical index of the segment in the paging file on tape. The right brother number reveals the relative dependency of the segments of the tree.

3.101 The loader also automatically creates a linkage to the paging monitor, CPAGM, when it detects a transfer to a symbol in a nonresident segment. The transfer cannot be made directly since the program is not in memory and may require being paged in. This linkage is transparent to the calling and called programs. For each transfer to a nonresident program by a resident program, the loader creates the linkage and calls an ENTAB entry (Fig. 22) which resides in main store. The loader places in the transfer instruction a pointer to the location of the ENTAB entry instead of the paging buffer location value assigned to the symbol that will receive control. The ENTAB entry contains the segment number of the program containing the symbol. It also contains the address of the symbol location in the paging buffer which is accessed by the paging monitor. The ENTAB entry also has a branch and save address (BSAI) instruction which gives control to the paging monitor, thus providing the linkage.

3.102 In addition, the loader creates ENTAB linkages for nonresident program transfers to other nonresident programs. These ENTAB entries are linked to the nonresident segments and are also stored on the tape (thus nonresident). The ENTAB entries for each program segment are, therefore, paged into the paging buffer along with the program segment.

3.103 As already stated, the paging monitor CPAGM gains control via the call instruction (BSAI) in the ENTAB entry. A temporary area of store called the paging control block (PCB) is used by CPAGM to save registers and to store the address in the paging buffer of the symbol to which transfer is to be made. The symbol address and the segment number of the segment containing the symbol from the ENTAB entry are retrieved by CPAGM.

3.104 Using the segment number, CPAGM indexes into the paging map to determine whether the segment needed is already resident in the paging buffer. The paging map is a 16-word block of memory forming a 16 by 16 matrix that contains bits representing the status of each segment. When the bit is set, the segment is resident in the buffer; however, a zero bit implies the segment must be paged if needed. If the segment needed is already in the buffer, CPAGM gives control to the location of the symbol in the paging buffer for execution.

3.105 When the segment is not already in the paging buffer, CPAGM zeroes all the bits in the paging map after the bit for the needed segment because any segments in the paging buffer will likely be overlayed. The segment table is then examined by CPAGM to determine any supporting segments which must also be paged along with the needed segment. All segments on which a segment is dependent appear above that segment in the segment table (that is, the segment numbers are less than that of the dependent segment). Also, the right brother numbers of any segments required by another segment are greater than or equal to the right brother number of that segment. Therefore, CPAGM compares the right brother number of each segment in the segment table above the needed segment with that of the needed segment and pages into the paging buffer all segments that are required. Also, CPAGM updates the bits in the paging map to indicate whether each segment is loaded or not loaded.

3.106 To accomplish the paging operation, CPAGM requests that the paging file be opened by the tape handler (CTAPH). It then requests the tape to be positioned at the beginning of the segment to be paged into memory (identified by the tape logical index in the segment table entry for that segment). A continuous read request is initiated to the tape handler which will read all blocks of the segment. (The last block of the segment has an end-of-file designation which terminates the read.) After each block is read, control is returned briefly at interrupt level to CPAGM which moves the block from the I/O buffer into the paging buffer at the location identified in the loading address in the tape block. This procedure is performed for each segment which must be paged. If CPAGM must take a real-time break during the paging operation, control is returned to the multiscan function controller.

3.107 After the segment containing the original destination and all supporting segments have been loaded into the paging buffer, CPAGM gives control to the location specified in the ENTAB entry. Normal execution will now continue. Figure 23 depicts the general flow of the paging operation.

3.108 In case of a tape or paging error, an abort will return control to the CBLM. A TTY error message is printed indicating the error.

3.109 The program CPAGM is not only entered indirectly through the linkage created by the loader but also can be explicitly entered. A client needing a nonresident data table from the tape calls a special entry point in CPAGM. The client provides the address of a 2-word data table (Fig. 24) which defines the segment number and address where the nonresident table is to be located in memory. The ENTAB entry is linked to the segment by the loader. The segment containing the data table is paged in the same manner as previously described except that control is not transferred to the data table but is returned to the client or calling program.

B. Verify Dates on Translation Files

3.110 The program VRDATE is a tape client which can print the dates of the current translation file or the two backdated translation files upon a TTY request. Program VRDATE determines the request made and issues the tape macro call to seize control of the tape control block (TCB) and to open the appropriate file at the block containing the date(s). After the file is opened and the tape is positioned at the correct block, VRDATE initiates the reading of the block (performed by CTAPH). The date(s) retrieved are then printed and the file is closed.

3.111 Any errors encountered cause the operation to be aborted and a TTY message to be printed indicating the operation was not completed.

C. Translation File Updates

3.112 The task of the office data file update program, OPDATA, is to update the backup files containing the translations for the No. 3 ESS office. The program is nonresident and is executed as a multiscan function. Program OPDATA consists of two parts:

(1) The updating of the most recent copy of translations contained in the file TRNSLN (paragraphs 3.22 and 3.23)

(2) The updating of the backdated translations in file BACKDT1 and BACKDT2 (paragraphs 3.24 and 3.25).

Update of Current Translation File

3.113 In most cases when a memory reload of translation data is required, reloading the most recent copy of translations is sufficient to restore the system. Program OPDATA is invoked automatically every 24 hours to update the current translation file, TRNSLN, with the current translations in memory and also the associated data in the CHECKSUM file. This task is initiated after the daily traffic printouts have been completed, at which time the traffic multiscan function (TRAFIC) passes control to OPDATA. This function can also be invoked by a TTY request (OP:DATA;CURR!). Program OPDATA uses a temporary storage area called a task control block in order to store information needed for the performance of the function. This task control block also contains an internal progress mark which is used after real-time breaks to determine the next routine to be executed. Real-time breaks are initiated when a function requires more time than can be allowed at a given time. The WAIT subroutine in CBLM is called to cause a base level loop delay before returning control to OPDATA for further processing.

3.114 A test is made to ensure that both tape units are in service. If either is out-of-service or in a read only mode, an error message is printed on the TTY and the update is not performed. If both tape units are in service, various checks are made to determine whether any recent changes have been made since the last update of TRNSLN, ie, the number of translation hashsums calculated is equal to zero, the number of translation entries in the checksum file is equal to zero, etc. If no recent changes have been made, no update is necessary and the only change made to the file is to change the date word within the file to the current date. This requires formatting the date word in the I/O buffer, opening the TRNSLN file on unit 0, writing the date on the file positioned in the proper block, and closing the file. The same procedure is repeated to change the data in the file on unit 1 and a TTY message is printed to indicate that the operation has been completed.

3.115 If recent changes have occurred, both the TRNSLN file and the checksum file require updating. It is possible that the updating process could abort due to an I/O error or a request from the multiscan function controller to abort. This could leave the office with a partially updated translation file, and a subsequent memory reload would load this invalid data into main store. Two precautions are taken to prevent this from occurring:

(1) The update process is performed on one tape at a time.

(2) Before the update of TRNSLN begins, the update-in-progress bit in the first block of the checksum file is set.

Then, if the update fails to complete successfully and a memory reload becomes necessary before the problem is corrected, the bit will indicate to the system initialization routines that the other tape should be used for memory reload. Thus, the checksum file is opened on tape unit 0, the first block is read and rewritten with the update-in-progress bit set, and the file is then closed. In addition, the hashsum block read is stored in a temporary save area for further reference.

3.116 The TRNSLN file is next opened on tape unit 0. The updates are done in 4K-segment increments and only those segments which have been changed are rewritten on the tape. Each 4K segment is stored on tape beginning in a new block. In all translations, the first segment in memory will always begin on a 4K boundary.

3.117 The changed segment is accessed in memory and the tape is positioned to the first block for that segment. Each block contains a loading length and loading address which tell how many words of translation data are contained in the block and where they are to be loaded in translation store. These parameters are used by the bootstrap program in reloading translation store. Program OPDATA formats and stores these parameters in each block in addition to the translation data.

3.118 After all changed 4K segments have been outputted to the tape, a comparison is made between the number of translation segments which existed on the tape before the update (determined from the hashsum save area) and the number of segments now in translation data. When the current number of segments is less than the previous number, the excess segments on the tape are zeroed. The TRNSLN file is then closed.

3.119 The checksum file on tape unit 0 is reopened. The new hashsums for the changed segments are calculated. The new hashsum calculation is stored in the proper word in the hashsum block if different from the previous hashsum. Also, the new loading length of the hashsum block is calculated and stored in the block and any hashsums corresponding to excess translation segments are zeroed. The update-in-progress bit is zeroed and the entire block is rewritten onto the tape. The checksum file is then closed.

3.120 The entire procedure is repeated for tape unit 1. Then checks are made to verify the update and a TTY message is printed indicating the update has been completed. For TTY requests, control is passed to the administrative control program, ADMCON. When OPDATA is invoked automatically, control is passed to the diagnostic control program DCON which is the next multiscan function to be performed.

3.121 When an error occurs or when the multiscan function controller requests the update operation to be aborted, OPDATA closes the currently open file. Also, an abort message is printed and the update is terminated by passing control to the appropriate location depending on whether the update request was a TTY request or automatic.

Update of Backdated Translation Files

3.122 Since it is possible that an error can occur in the up-to-date translation data, older versions of translation data are maintained in files BACKDT1 and BACKDT2 (Fig. 8) should it become necessary to reload memory. The second section of OPDATA does the updating of these files and is invoked manually by a TTY input message (OP:DATA;OLD!). No hashsums are associated with the data on these files. The update is done simultaneously on both tapes because, if one of the backdated files contains invalid (partially updated) data, the other file can be used in a memory reload. The files are marked duplicated in the tape directory; therefore, any operation performed by the tape handler on one tape is automatically performed on the other tape.

3.123 The file BACKDT1 is opened at the first block which contains the two date words for the two files. An "in-progress" response is given on the TTY if the open is successful. If unsuccessful, a "repeat-later" response is given. Program OPDATA reads and examines the date words to determine which file is the oldest. The file with a zero date word or the oldest file is to be updated with the translations currently stored in the on-line memory. The date word for the file to be updated is zeroed to indicate that an update is in progress. If the update is interrupted, a zeroed date word indicates to the system initialization programs during a reload that the file should not be used for the reload because it contains erroneous data.

3.124 When BACKDT2 is to be updated, a backspace is issued and the first block of BACKDT1 is rewritten containing the zeroed date word. BACKDT1 is then closed and BACKDT2 is opened for updating. Otherwise, BACKDT1 remains open for updating and the date word will be written when the first block is updated.

3.125 The translation data is transferred into the I/O buffer from memory a block at a time. In addition, all translation segments in memory are copied to tape, not just the segments which have been changed. For each block, the program OPDATA formats in the I/O buffer the loading length and loading address fields, fills the remainder of the block with translation data, and initiates the writing of the block on the tape. The last block is written with the EOF bit set.

3.126 After the last block has been written, the date word must be updated. If BACKDT2 is being updated, it is closed and BACKDT1 is opened. Otherwise, BACKDT1 is positioned to the block containing the date word. The current date is formatted in the proper word of the block and the block is rewritten on the tape. The BACKDT1 file is then closed. A TTY message is printed indicating the update operation is complete and control is given to the administrative control program ADMCON.

D. Initialization Use of the Tape

3.127 The program CIPL (common initial program loader) must access the tape in order to reload memory with the generic programs. Therefore, CIPL contains the tape access routines needed to restore the generic since CTAPH is not available in memory at the time. These routines are described in Section 233-153-130, Initialization and Processor Fault Recovery, Software Subsystem Description, No. 3 ESS.

E. Tape Utilities

3.128 Tape utilities for the No. 3 ESS are nonresident and are incorporated in the common nonresident utilities program CNRUTL. The ALW:TAPEUTIL! input message is used to request the utilities to be made available, ie, initiates paging of the program into the paging buffer in resident memory. Tape utilities use the seize macro to gain access to the whole tape; whereas, other tape clients use the open macro. Tape utilities perform such functions as copying one tape to another (except track 1), auditing a tape against a standard, emergency writing onto tape from the TTY, reading a block, and storing the generic file and translation file in off-line store. In addition, the utilities have the capability to dump (1) a block of tape to the TTY, (2) a segment of tape to the TTY, (3) the tape label file from tape to the TTY, or (4) the tape directory file from tape to the TTY. Also, the capability to change the generic is available in CNRUTL and is designated the overwrite multiscan function. Tape utilities are described in more detail in Section 254-340-082.

4. GLOSSARY

4.01 The following terms, abbreviations, and definitions are used in this section to describe the tape operation functions.

Address—A combination of bits that identify a location in a storage device or equipment unit.

ASCII—American Standards Committee for Information Interchange.

Base level—Major software loop including all functions not done at interrupt level.

Bit—The binary unit of information which is represented by one of two possible conditions; such as the digits 0 and 1, on or off.

BOT—Beginning of tape.

Clear—To restore a storage device to the "zero" state.

CRC—Cyclic redundancy character.

DEOF—Double end-of-file.

DUP—Duplicated bit.

EOF—End-of-file.

ESS—Electronic Switching System.

Interrupt—An in-process program is interrupted so that a task can be performed that is required immediately. When the task is completed, the interrupted program continues to execute.

I/O buffer—Input/output buffer.

K—1024.

Logical index—Number of the tape block with reference to the beginning of the file.

Macro—A sequence of operations called by an abbreviated notation. A macro can generate different sequences of code depending on the parameters supplied in the macro call.

MRF—Maintenance reset function.

MSF—Multiscan function.

Paged—Method by which nonresident programs or tables are brought into memory from tape.

PCB—Paging control block.

Physical block number—Number of the tape block with reference to the beginning-of-tape mark.

TCB—Tape control block.

TDC—Tape data controller.

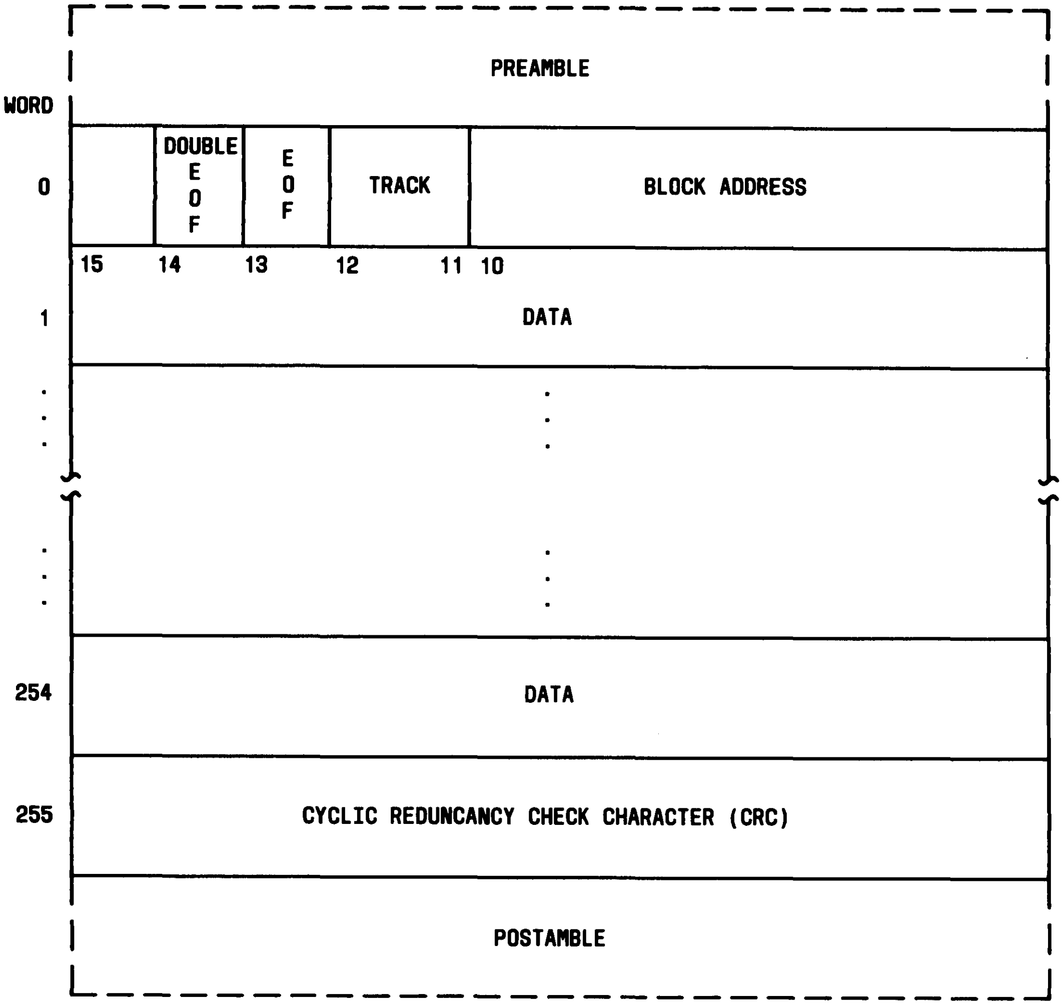
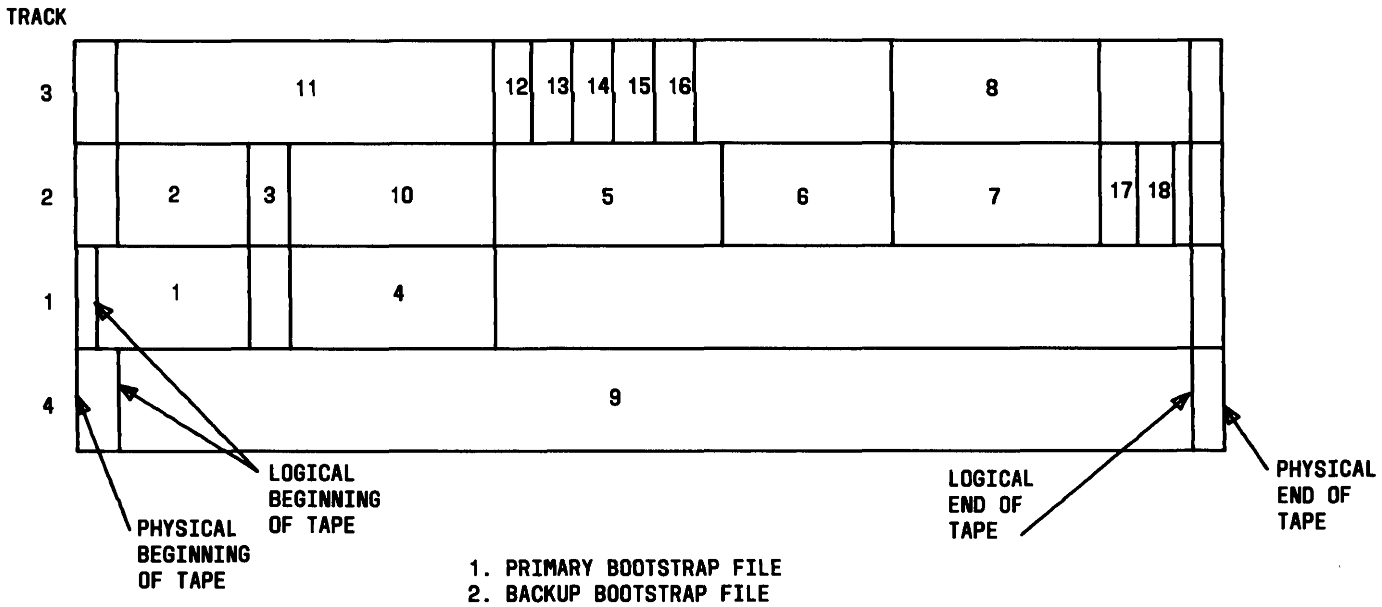


Fig. 1—Tape Block General Format—SO-2 Issue 4 or 4A Generic



1. PRIMARY BOOTSTRAP FILE
2. BACKUP BOOTSTRAP FILE
3. PRIMARY CHECKSUM FILE
4. GENERIC FILE
5. TRANSLATION FILE
6. PATCH FILE
7. BACKDATED TRANSLATION FILE (BACKDT1)
8. BACKDATED TRANSLATION FILE (BACKDT2)
9. PAGED PROGRAM FILE
10. OFFICE RECORD FORM FILE
11. OFFICE RECORD FORM FILE
12. MULTISCAN FUNCTION OVERLAY SAVE FILE (ROLL)
13. LINE CLASS CODE TABLE FILES (LCCTBL0)
14. LINE CLASS CODE TABLE FILES (LCCTBL1)
15. LINE CLASS CODE TABLE FILES (LCCTBL2)
16. TRAFFIC FILE
17. DIRECTORY
18. LABEL FILE

Fig. 2—No. 3 ESS List 1 Tape Cartridge Format—SO-2 Issue 4 or 4A Generic

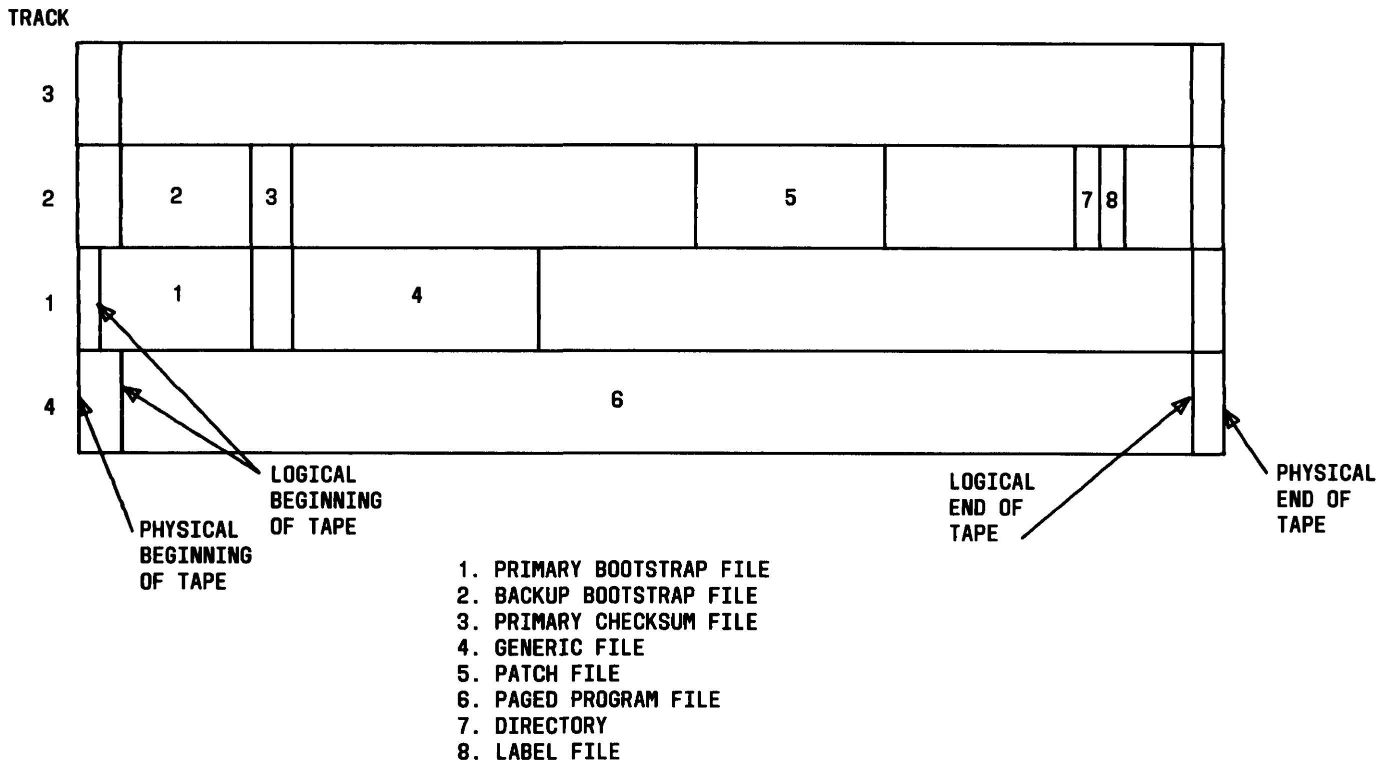


Fig. 3—List 2 Tape Cartridge Format—SO-2 Issue 4 or 4A Generic

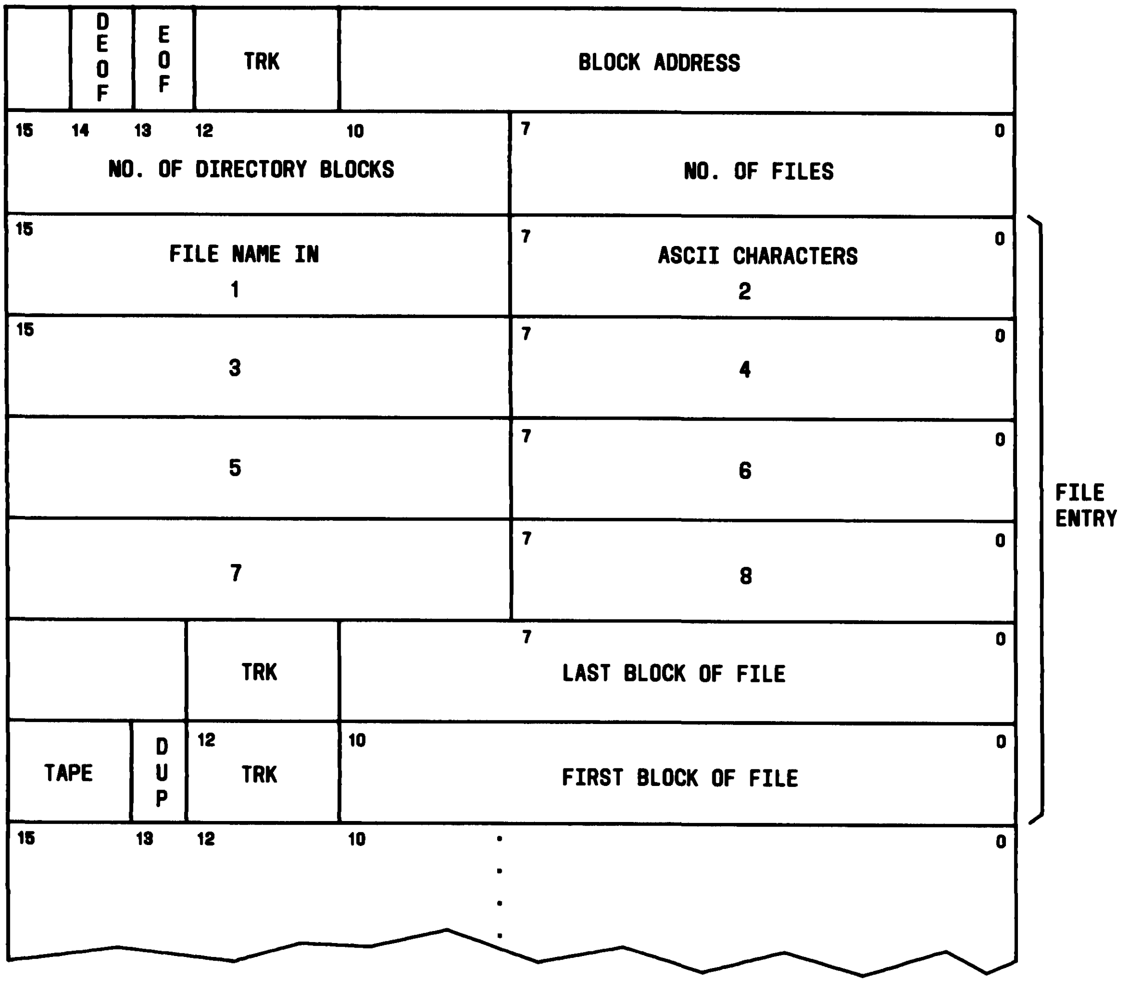


Fig. 4—Tape Directory

<table>
<tr><td></td><td>D
E
O
F</td><td>E
O
F</td><td>TRACK</td><td colspan="2">BLOCK ADDRESS</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12 11</td><td>10</td><td>3 0</td></tr>
<tr><td colspan="5">LOADING LENGTH</td><td>HIGH 4 BITS OF
LOADING ADDRESS</td></tr>
<tr><td colspan="5">15</td><td>3 0</td></tr>
<tr><td colspan="6">LOW 16 BITS OF LOADING ADDRESS</td></tr>
<tr><td colspan="5">15</td><td>0</td></tr>
<tr><td colspan="6">PROGRAM OR DATA WORD</td></tr>
<tr><td colspan="5">15</td><td>0</td></tr>
<tr><td colspan="6">PROGRAM OR DATA WORD</td></tr>
<tr><td colspan="5">15</td><td>0</td></tr>
<tr><td colspan="6">.
.
.
.
.
.</td></tr>
<tr><td colspan="6">CRC</td></tr>
</table>

Fig. 5—Block of Program or Translation Data

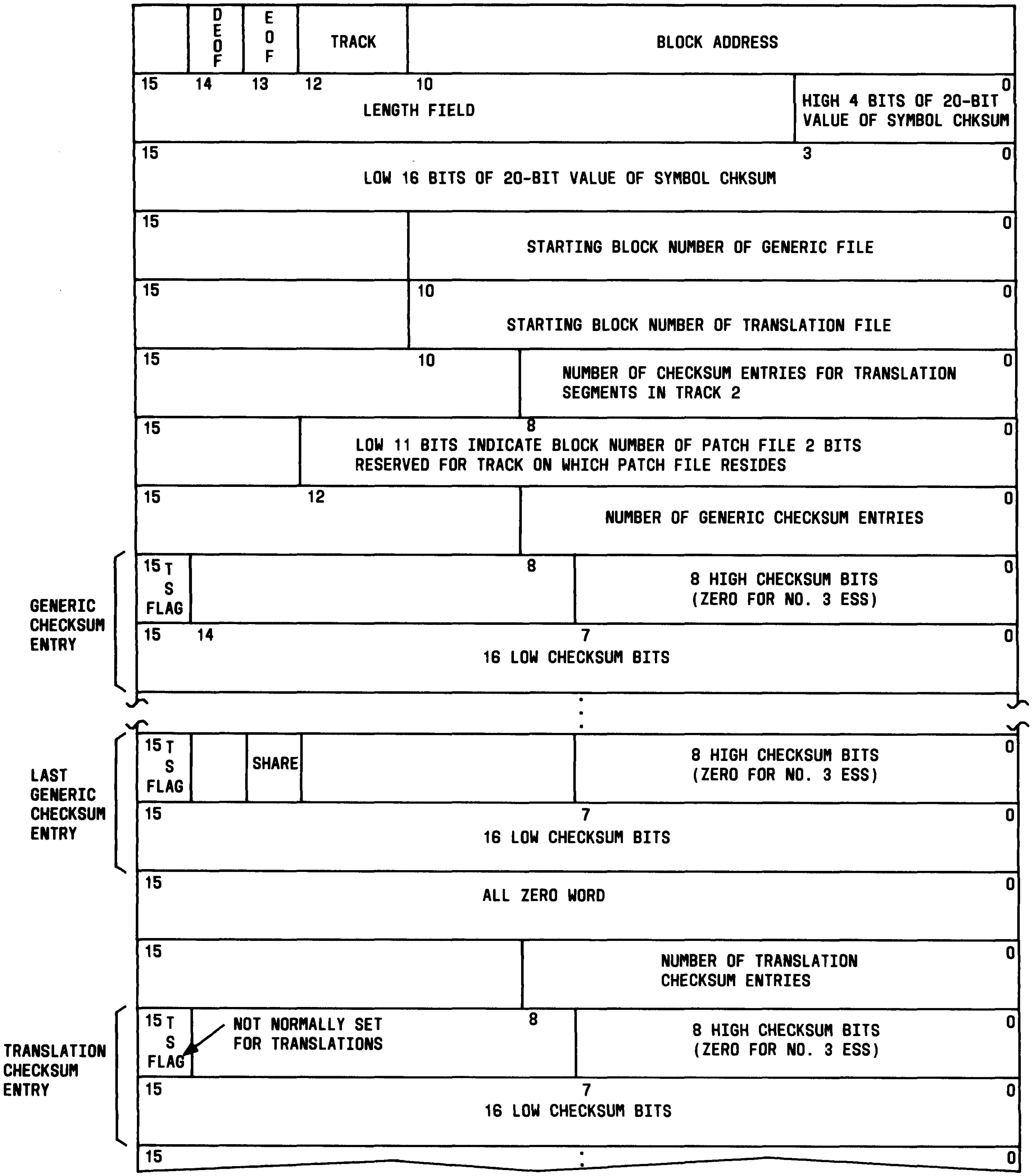

D E O F
E O F
TRACK
BLOCK ADDRESS
15 14 13 12 10 0
LENGTH FIELD
HIGH 4 BITS OF 20-BIT VALUE OF SYMBOL CHKSUM
15 3 0
LOW 16 BITS OF 20-BIT VALUE OF SYMBOL CHKSUM
15 0
STARTING BLOCK NUMBER OF GENERIC FILE
15 10 0
STARTING BLOCK NUMBER OF TRANSLATION FILE
15 10 0
NUMBER OF CHECKSUM ENTRIES FOR TRANSLATION SEGMENTS IN TRACK 2
15 8 0
LOW 11 BITS INDICATE BLOCK NUMBER OF PATCH FILE 2 BITS RESERVED FOR TRACK ON WHICH PATCH FILE RESIDES
15 12 0
NUMBER OF GENERIC CHECKSUM ENTRIES
GENERIC CHECKSUM ENTRY
15 T S FLAG
8 0
8 HIGH CHECKSUM BITS (ZERO FOR NO. 3 ESS)
15 14 7 0
16 LOW CHECKSUM BITS
LAST GENERIC CHECKSUM ENTRY
15 T S FLAG
SHARE
0
8 HIGH CHECKSUM BITS (ZERO FOR NO. 3 ESS)
15 7 0
16 LOW CHECKSUM BITS
15 0
ALL ZERO WORD
15 0
NUMBER OF TRANSLATION CHECKSUM ENTRIES
TRANSLATION CHECKSUM ENTRY
15 T S FLAG
NOT NORMALLY SET FOR TRANSLATIONS
8 0
8 HIGH CHECKSUM BITS (ZERO FOR NO. 3 ESS)
15 7 0
16 LOW CHECKSUM BITS
15 0

Fig. 6—Checksum File—SO-2 Issue 4 or 4A Generic

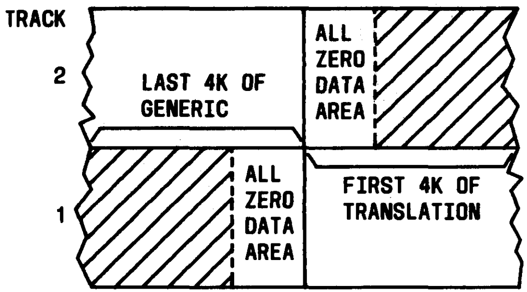
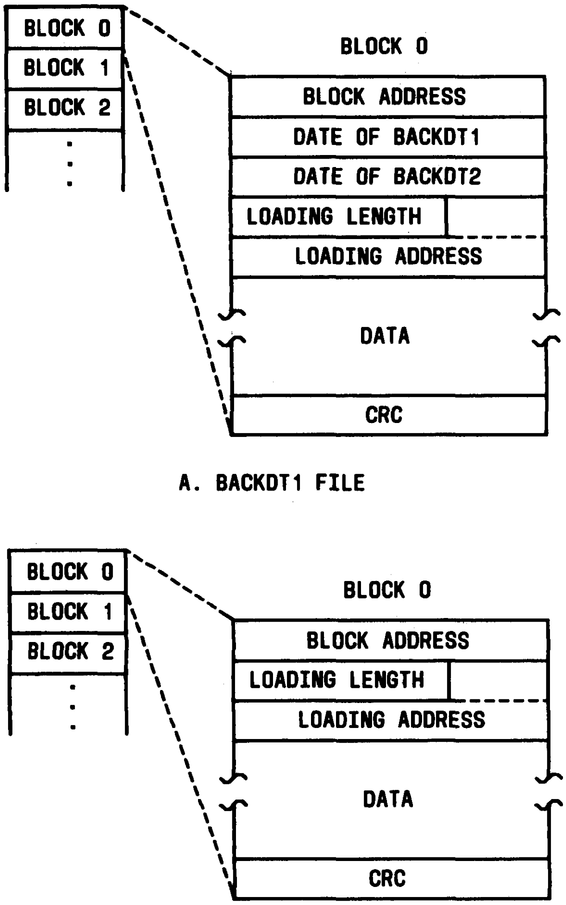


Fig. 7—Provision for Overlap Arrangement of Generic and Translation Areas



A. BACKDT1 FILE

B. BACKDT2 FILE

Fig. 8—Backdated Translation Files

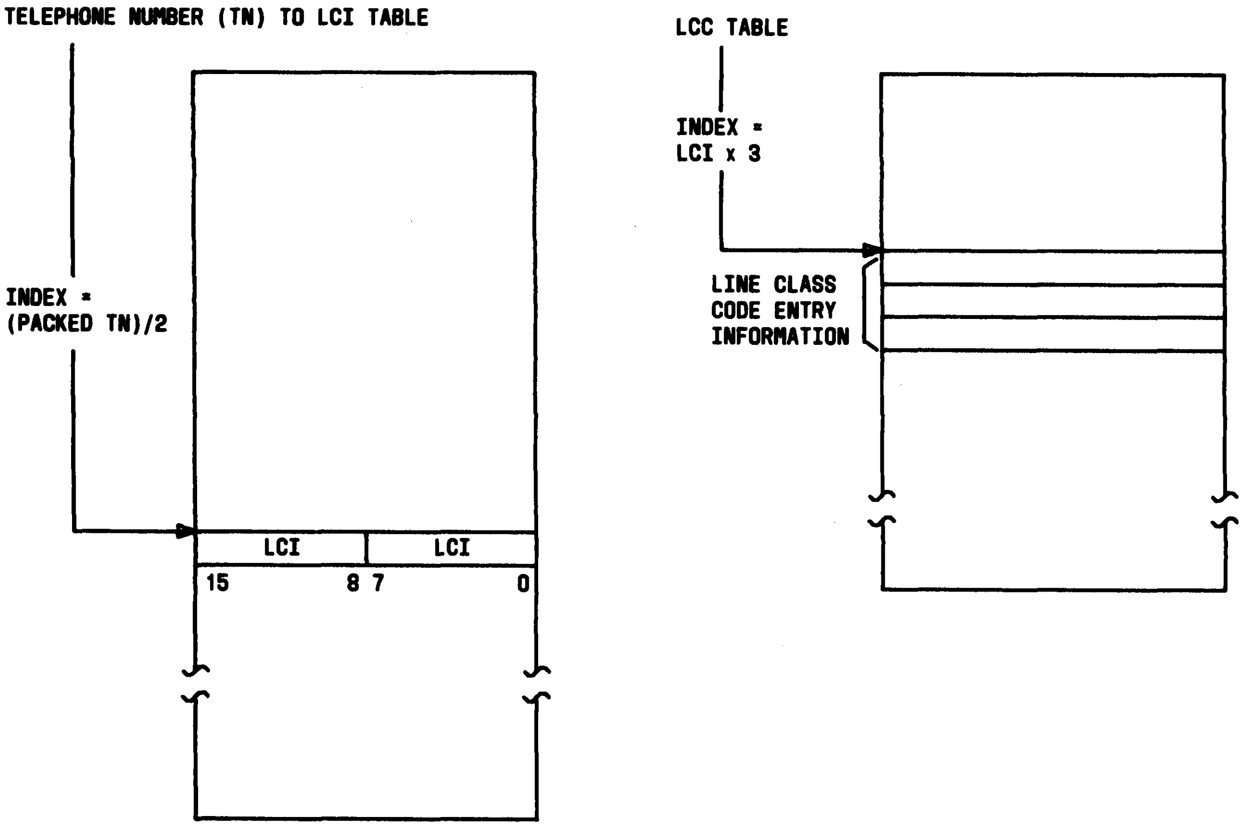


Fig. 9—Line Class Code Tables

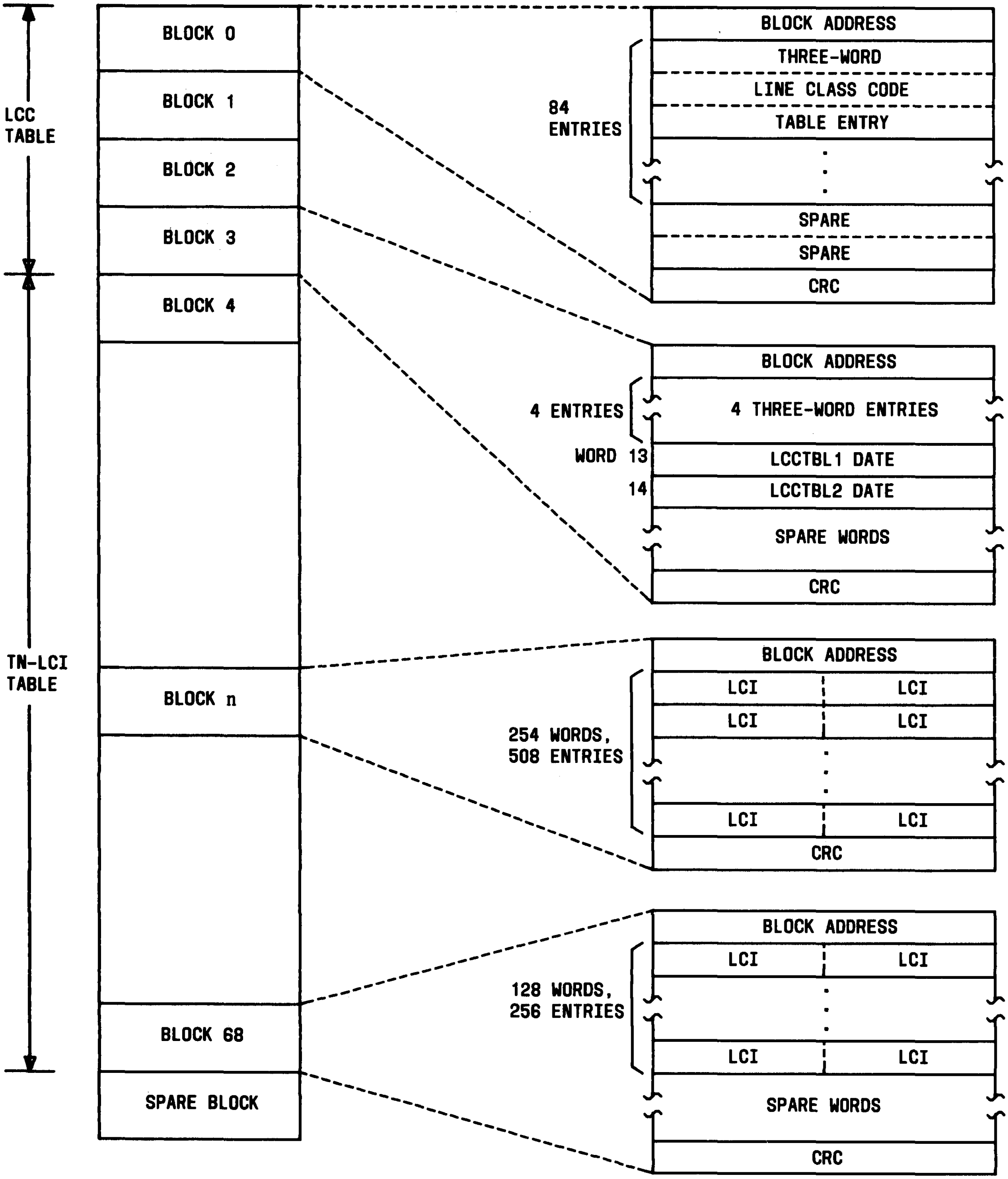
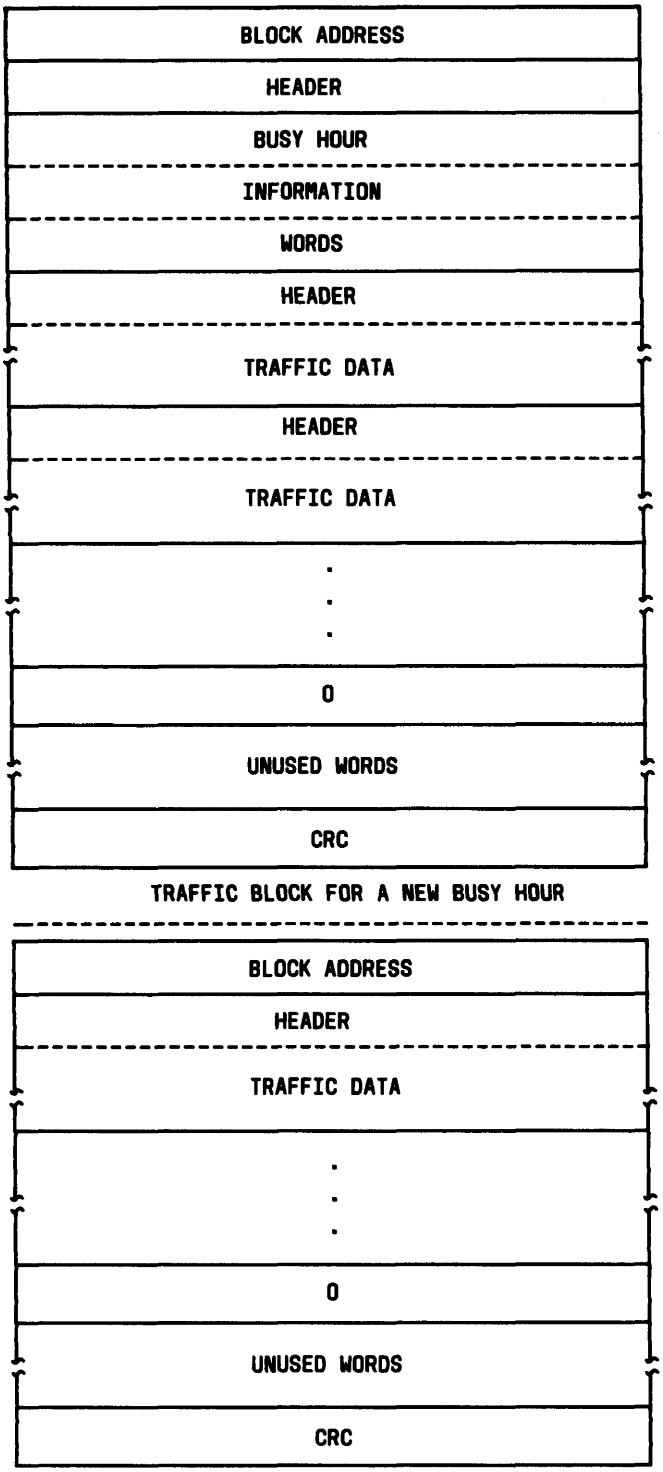


Fig. 10—Line Class Code Table Files



TRAFFIC BLOCK FOR A NEW BUSY HOUR

TRAFFIC BLOCK CONTINUING PRESENT BUSY HOUR

Fig. 11—Traffic File

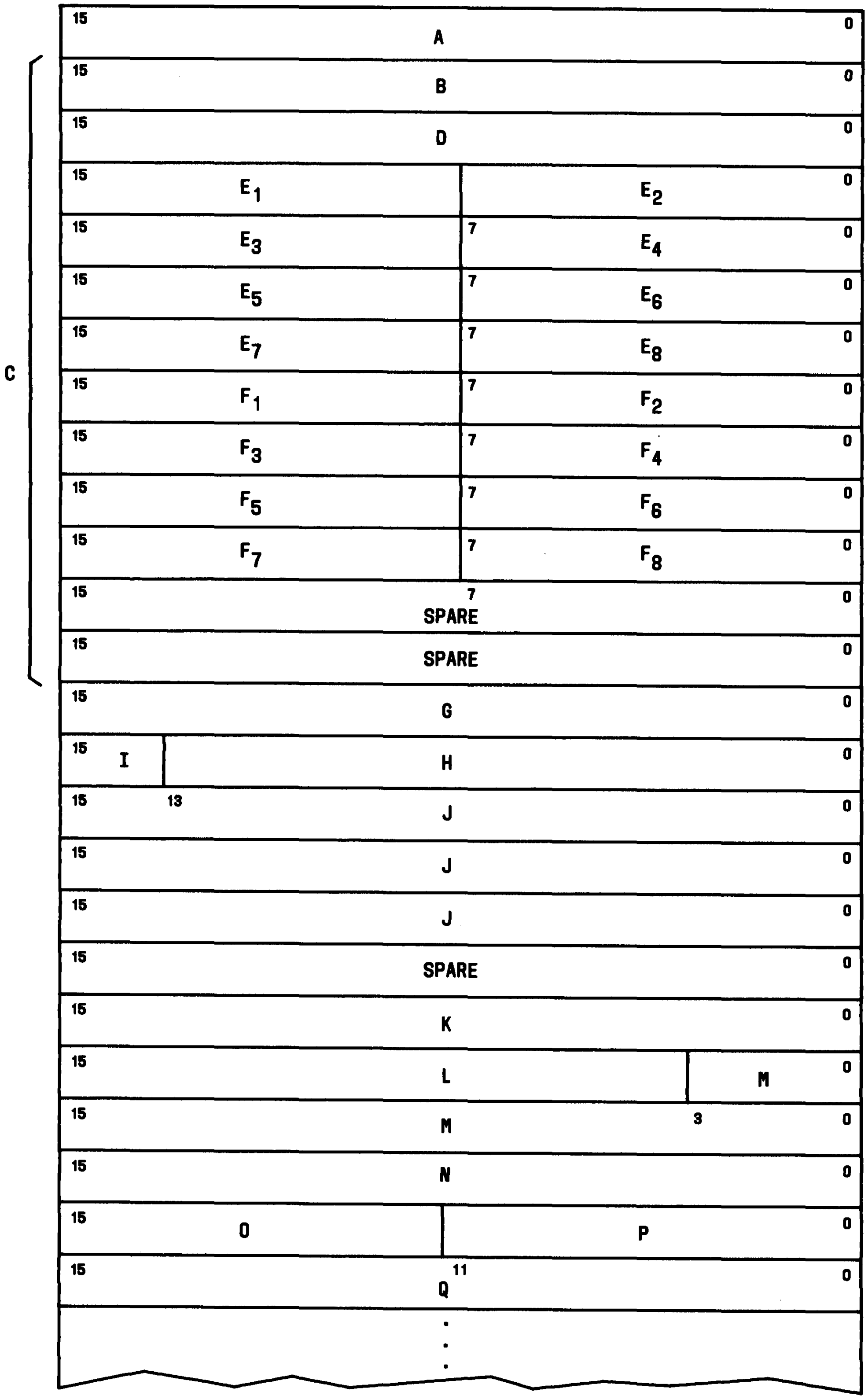


Fig. 12—Patch File

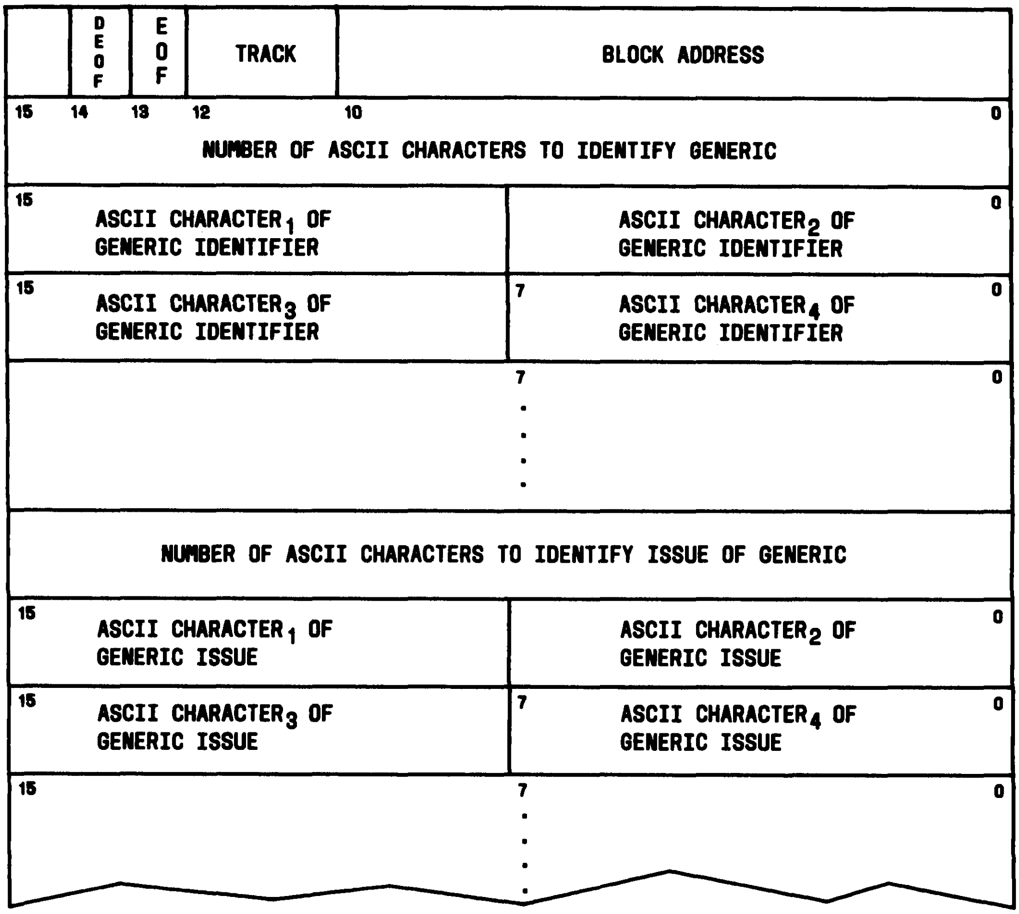

DEOF
EOF
TRACK
BLOCK ADDRESS
15
14
13
12
10
0
NUMBER OF ASCII CHARACTERS TO IDENTIFY GENERIC
15
ASCII CHARACTER$_1$ OF GENERIC IDENTIFIER
ASCII CHARACTER$_2$ OF GENERIC IDENTIFIER
0
15
ASCII CHARACTER$_3$ OF GENERIC IDENTIFIER
7
ASCII CHARACTER$_4$ OF GENERIC IDENTIFIER
0
7
0
NUMBER OF ASCII CHARACTERS TO IDENTIFY ISSUE OF GENERIC
15
ASCII CHARACTER$_1$ OF GENERIC ISSUE
ASCII CHARACTER$_2$ OF GENERIC ISSUE
0
15
ASCII CHARACTER$_3$ OF GENERIC ISSUE
7
ASCII CHARACTER$_4$ OF GENERIC ISSUE
0
15
7
0

Fig. 13—LABEL File

<table>
<tr><th>WORD</th><th colspan="16"></th></tr>
<tr><td>00</td><td colspan="5">CLIENT</td><td>STPST</td><td>SRFAL</td><td>IOCMP</td><td>ABWRT</td><td>WSI</td><td>MAINT</td><td>REQSTP</td><td colspan="4">REQ</td></tr>
<tr><td>01</td><td colspan="12"></td><td colspan="4">HOFILNAM</td></tr>
<tr><td>02</td><td colspan="16">FILNAM</td></tr>
<tr><td>03</td><td colspan="12"></td><td colspan="4">HOCONPT</td></tr>
<tr><td>04</td><td colspan="16">CONTLPT</td></tr>
<tr><td>05</td><td>SEEZ</td><td>ERNA</td><td>OTHERU</td><td colspan="2">VERTRK</td><td colspan="11">LOGBLK</td></tr>
<tr><td>06</td><td>EVOD</td><td>EXPLIC</td><td>DUP</td><td colspan="2">TRACK</td><td colspan="11">PHYBLK</td></tr>
<tr><td>07</td><td colspan="2">DIAGBITS</td><td>REPK</td><td colspan="2">ENDTRK</td><td colspan="11">LASTBLK</td></tr>
<tr><td>08</td><td colspan="14"></td><td colspan="2">TAPNO</td></tr>
<tr><td>09</td><td colspan="12"></td><td colspan="4">BUFCT</td></tr>
<tr><td>10</td><td colspan="16">STATEWRD</td></tr>
<tr><td>11</td><td colspan="16">STATEFLG</td></tr>
<tr><td>12</td><td colspan="16">TIMEK</td></tr>
<tr><td>13</td><td colspan="16">LSTCOM</td></tr>
<tr><td>14</td><td>5MRTN</td><td>SNER</td><td>TORLOK</td><td colspan="13">ORGIN</td></tr>
<tr><td>15</td><td colspan="3"></td><td colspan="2"></td><td colspan="11">LSTGOOD</td></tr>
<tr><td></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

REQ	REQUEST BITS	DUP	DUPLICATED FILE
REQSTP	STOP REQUEST BIT	EXPLIC	EXPLICIT TAPE UNIT REQUEST
MAINT	MAINTENANCE REQUEST BIT (ERROR RECOVERY STATE IN PROGRESS)	EVOD	EVEN OR ODD TU FOR EXPLICIT TAPE REQUESTS
WSI	WRITE STOP INTERRUPT EXPECTED	LASTBLK	LAST BLOCK IN FILE
ABWRT	ABSOLUTE WRITE REQUEST	ENDTRK	ENDING TRACK OF FILE
IOCMP	I/O SUCCESS FLAG	REPK	REPACK TAPE REQUEST
SRFAL	SEARCH FAIL (INTERRUPT DETECTED FAILURE)	DIAGBITS	DIAGNOSTIC IN PROGRESS
STPST	STOP DATA STORAGE (DATA NOT READ INTO I/O BUFFER)	TAPNO	TAPE NUMBER
		BUFCT	BUFFER COUNT
CLIENT	CLIENT NUMBER	STATEWRD	STATE STACK WORD (4 BITS PER STATE)
HOFILNAM	HIGH BITS OF FILE NAME ADDRESS		
FILNAM	FILE NAME ADDRESS LOW ORDER BITS	STATEFLG	STATE FLAGS (4 BITS PER STATE)
HOCONPT	HIGH BITS OF CLIENT CONTROL ADDRESS	TIMEK	TIME CONSTANT STORAGE
CONTLPT	LOW BITS OF CLIENT CONTROL ROUTINE	LSTCOM	LAST MOTION ORDER SENT TO TDC AT BASE LEVEL
LOGBLK	CLIENT LOGICAL INDEX INTO FILE		
VERTRK	TRACK VERIFY WILL BE DONE ON	ORGIN	ORIGIN OF FILE, BLOCK, AND TRACK
OTHERU	OTHER UNIT ALREADY USED IN RECOVERY	TORLOK	TAPE OFF REEL FLAG
ERNA	ERROR RECOVERY NOT ALLOWED BIT	SNER	SEEN TAPE OFF REEL
SEEZ	SEIZE FUNCTION IN PROGRESS	5MRTN	CLIENT WISHES INTERRUPT CONTROL IF BR FLAG EQUALS ZERO
PHYBLK	PHYSICAL LOCATION OF FILE		
TRACK	TAPE TRACK	LSTGOOD	LAST GOOD BLOCK READ IN

Fig. 14—TCB

		TRACK CODES		COMMAND CODE					MANUAL ENABLE	DEVICE CODE					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Fig. 15—Command Word

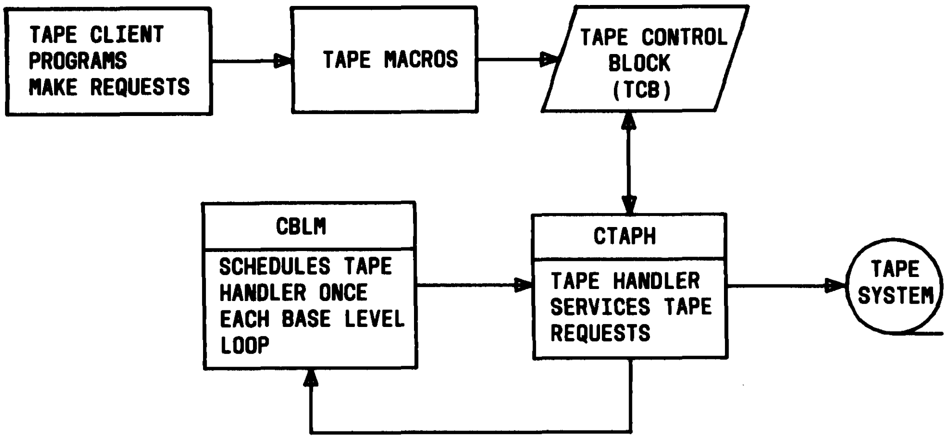

TAPE CLIENT PROGRAMS MAKE REQUESTS
TAPE MACROS
TAPE CONTROL BLOCK (TCB)
CBLM
SCHEDULES TAPE HANDLER ONCE EACH BASE LEVEL LOOP
CTAPH
TAPE HANDLER SERVICES TAPE REQUESTS
TAPE SYSTEM

Fig. 16—Tape Handler Interfaces

SQERR	REDON	POSOK	COMPROK	TSTATE	PTP - PRESENT TAPE POSITION										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

PTP	- PRESENT TAPE POSITION
TSTATE	- OUT-OF-SERVICE INDICATOR
COMPROK	- COMPARE BIT
POSOK	- POSITION VERIFIED (POSITION OK)
REDON	- TAPE IN READ ONLY STATE
SQERR	- BLOCK READ IN IS OUT OF ORDER

Fig. 17—Tape Status Word

MOP	BOT	REWND	MTCE	DDT	TIM	TOR	EOT	TUR	WENBL	DEVICODE					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

DEVICODE	- DEVICE CODE FOR TAPE
WENBL	- WRITE ENABLE
TUR	- TAPE UNIT READY
EOT	- END OF TAPE FLAG
TOR	- TAPE OFF REEL
TIM	- TAPE IN MOTION
DDT	- DATA DETECT
MTCE	- MAINTENANCE ACTIVE
REWND	- TAPE REWINDING
BOT	- BEGINNING OF TAPE FLAG
MOP	- MANUAL OPERATION ACTIVE

Fig. 18—Tape Status Command Response

FILL	BOFL	ACTIV	STUFF	UNST	LODST	INTOFF	RDY	BR	PPE	DEVICE CODE					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

DEVICE CODE	- DEVICE CODE FOR BUFFER
PPE	- PARALLEL PARITY ERROR
BR	- BUFFER READY FLAG
RDY	- BUFFER UNIT READY
INTOFF	- INTERRUPT OFF
LODST	- LOAD STATE SET UP
UNST	- UNLOAD STATE SET UP
STUFF	- STUFF ACTIVE
ACTIV	- ACTIVE TAPE BUFFER
BOFL	- BUFFER OVERFLOW SET
FILL	- FILL OCCURRING

Fig. 19—Buffer Status Command Response

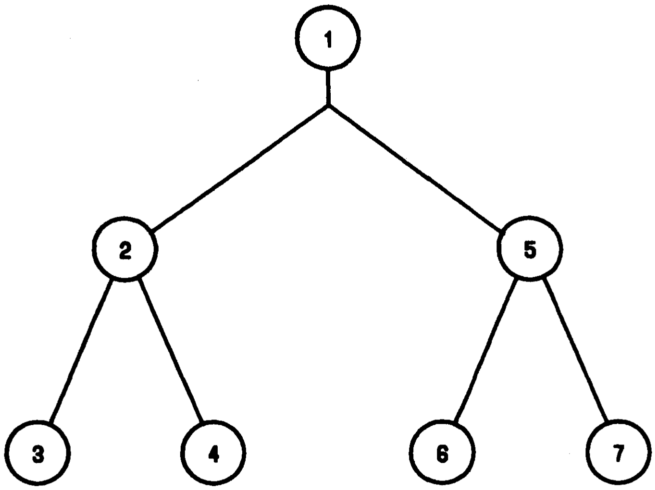


Fig. 20—Typical Tree Structure

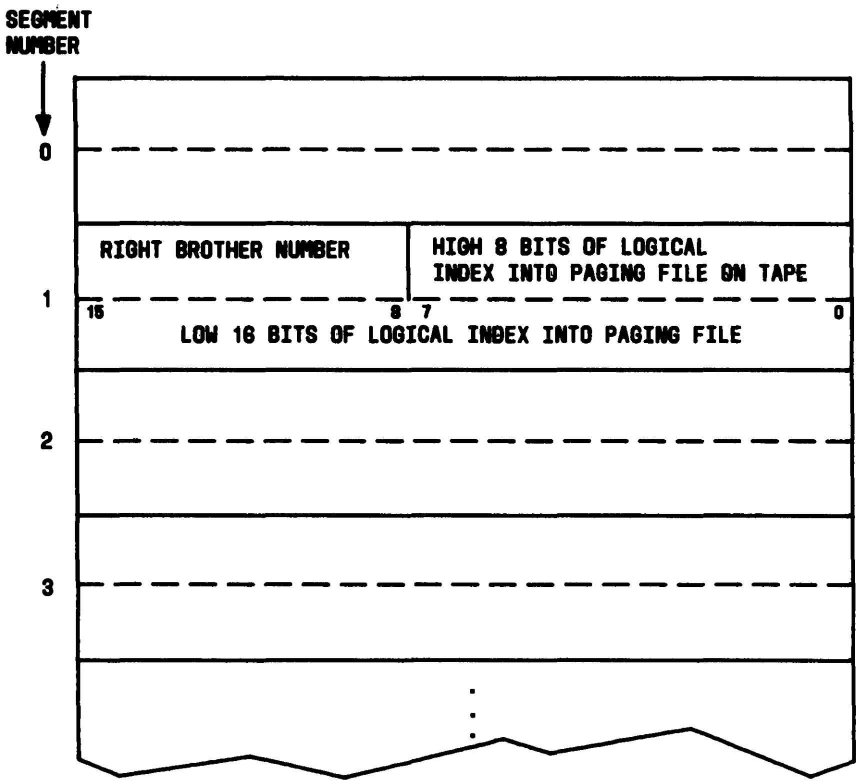

SEGMENT NUMBER
0
1
2
3
RIGHT BROTHER NUMBER
HIGH 8 BITS OF LOGICAL INDEX INTO PAGING FILE ON TAPE
15
8
7
0
LOW 16 BITS OF LOGICAL INDEX INTO PAGING FILE

Fig. 21—Segment Table

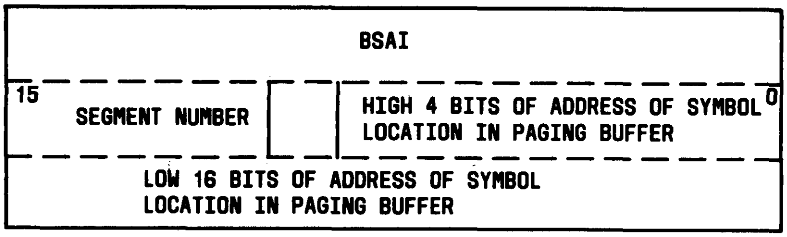

BSAI
15
SEGMENT NUMBER
HIGH 4 BITS OF ADDRESS OF SYMBOL LOCATION IN PAGING BUFFER
0
LOW 16 BITS OF ADDRESS OF SYMBOL LOCATION IN PAGING BUFFER

Fig. 22—ENTAB Entry

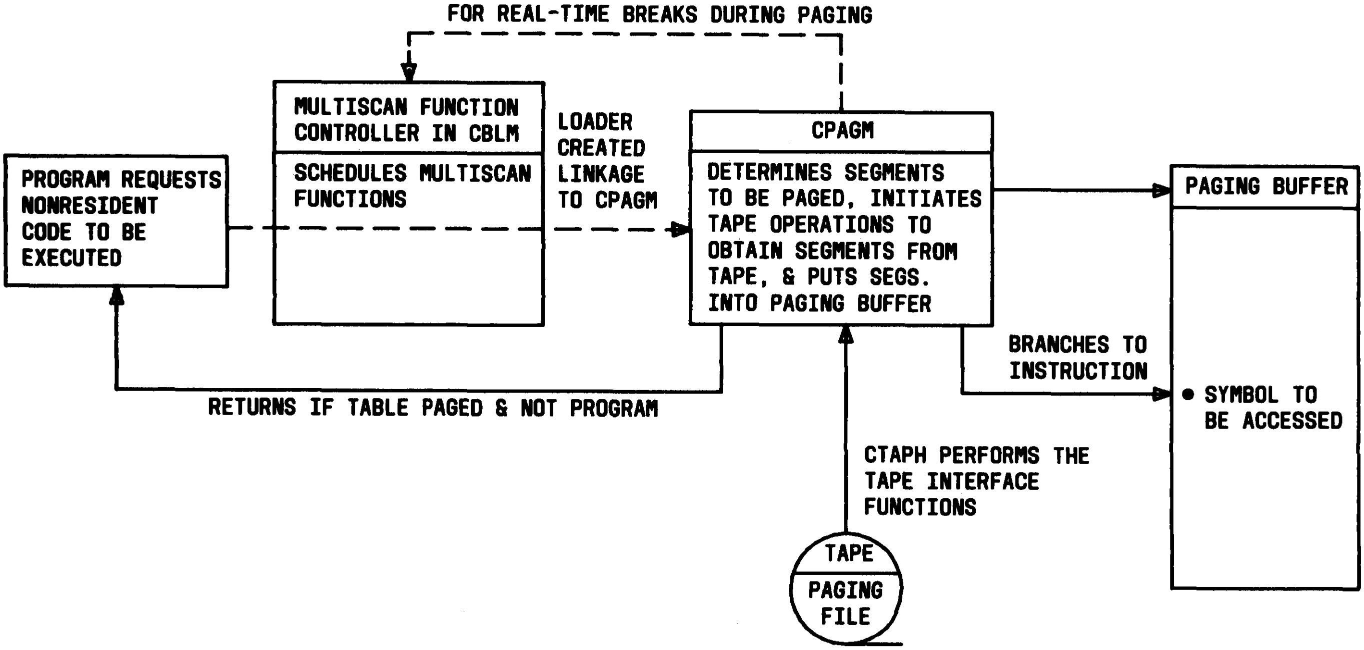

FOR REAL-TIME BREAKS DURING PAGING
PROGRAM REQUESTS NONRESIDENT CODE TO BE EXECUTED
MULTISCAN FUNCTION CONTROLLER IN CBLM
SCHEDULES MULTISCAN FUNCTIONS
LOADER CREATED LINKAGE TO CPAGM
CPAGM
DETERMINES SEGMENTS TO BE PAGED, INITIATES TAPE OPERATIONS TO OBTAIN SEGMENTS FROM TAPE, & PUTS SEGS. INTO PAGING BUFFER
PAGING BUFFER
BRANCHES TO INSTRUCTION
● SYMBOL TO BE ACCESSED
RETURNS IF TABLE PAGED & NOT PROGRAM
CTAPH PERFORMS THE TAPE INTERFACE FUNCTIONS
TAPE
PAGING FILE

Fig. 23—Paging Functions

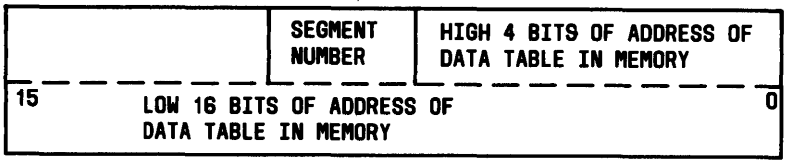
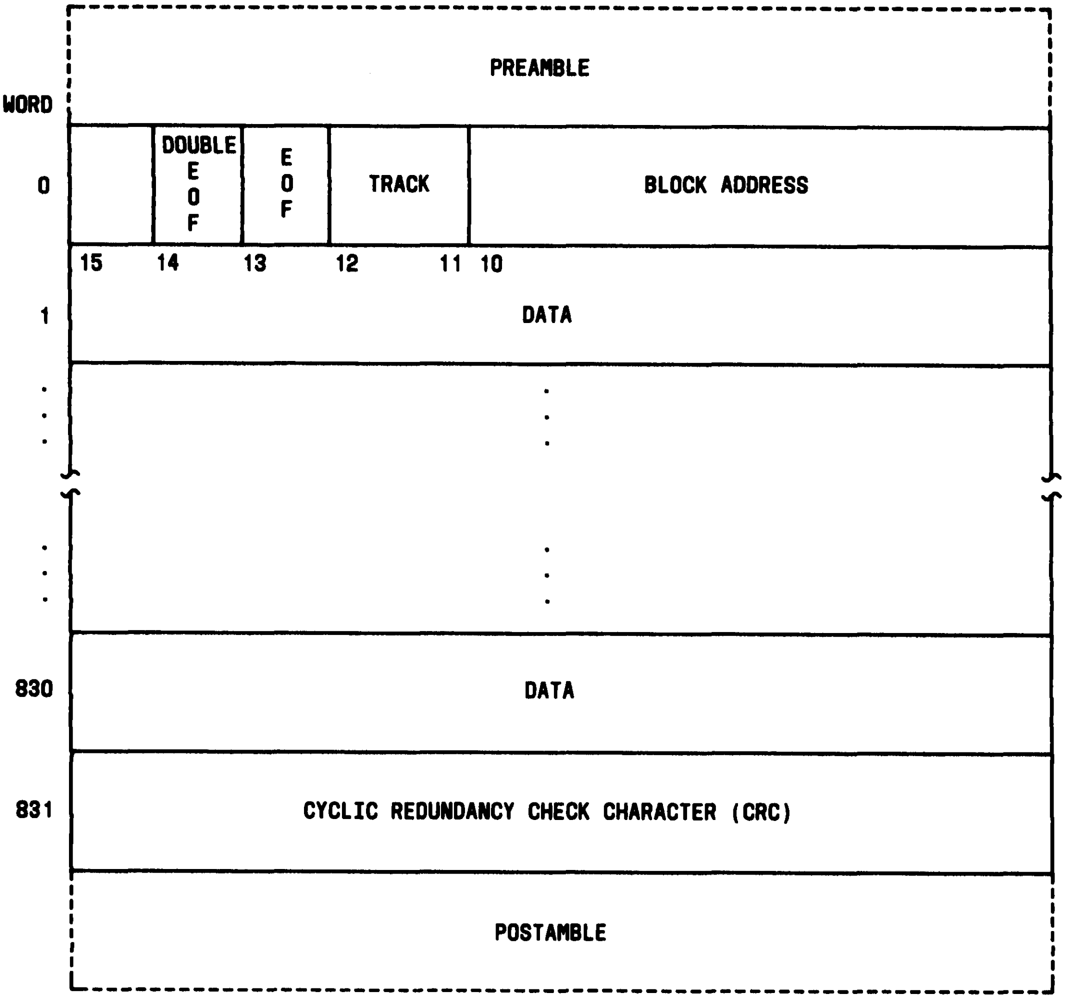
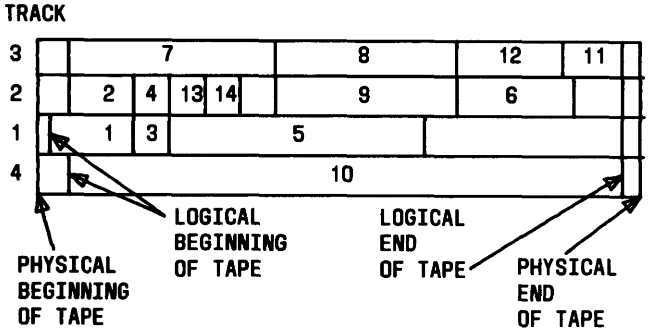

SEGMENT NUMBER
HIGH 4 BITS OF ADDRESS OF DATA TABLE IN MEMORY
15
LOW 16 BITS OF ADDRESS OF DATA TABLE IN MEMORY
0

Fig. 24—Table Entry for Explicit Paging



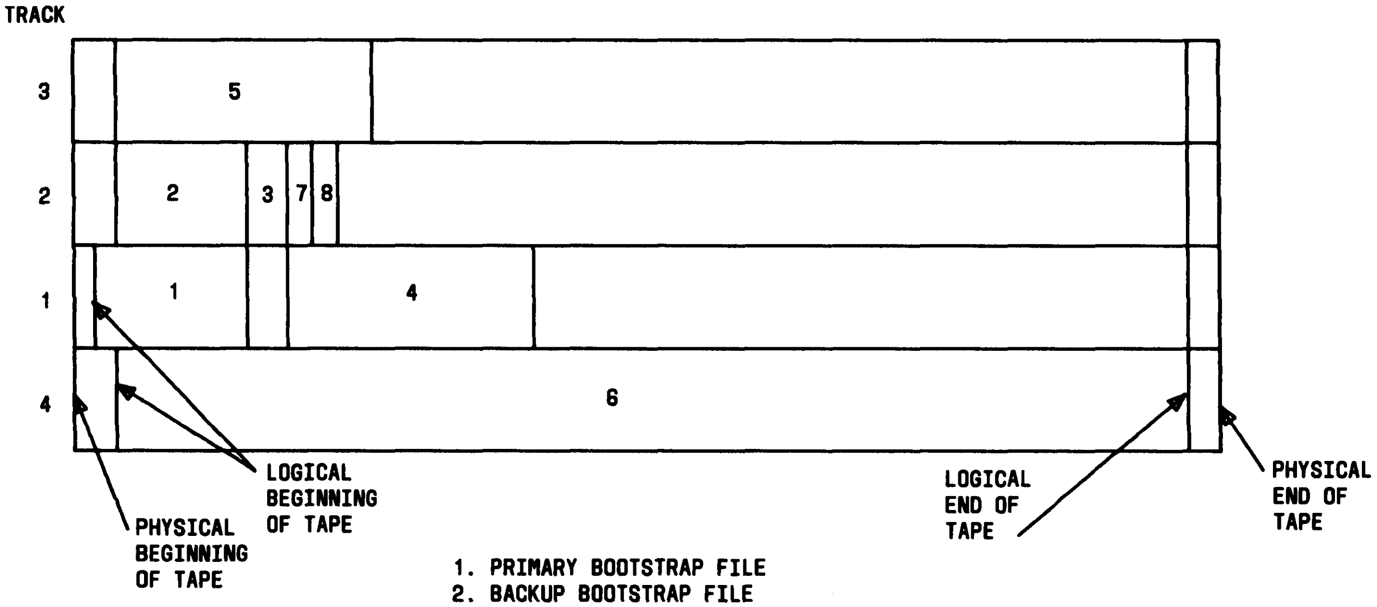
WORD					
	PREAMBLE				
0		DOUBLE E O F	E O F	TRACK	BLOCK ADDRESS
	15	14	13	12 11	10
1	DATA				
. . .	. . .				
. . .	. . .				
830	DATA				
831	CYCLIC REDUNDANCY CHECK CHARACTER (CRC)				
	POSTAMBLE				

Fig. 25—Tape Block General Format—3E3 Generic



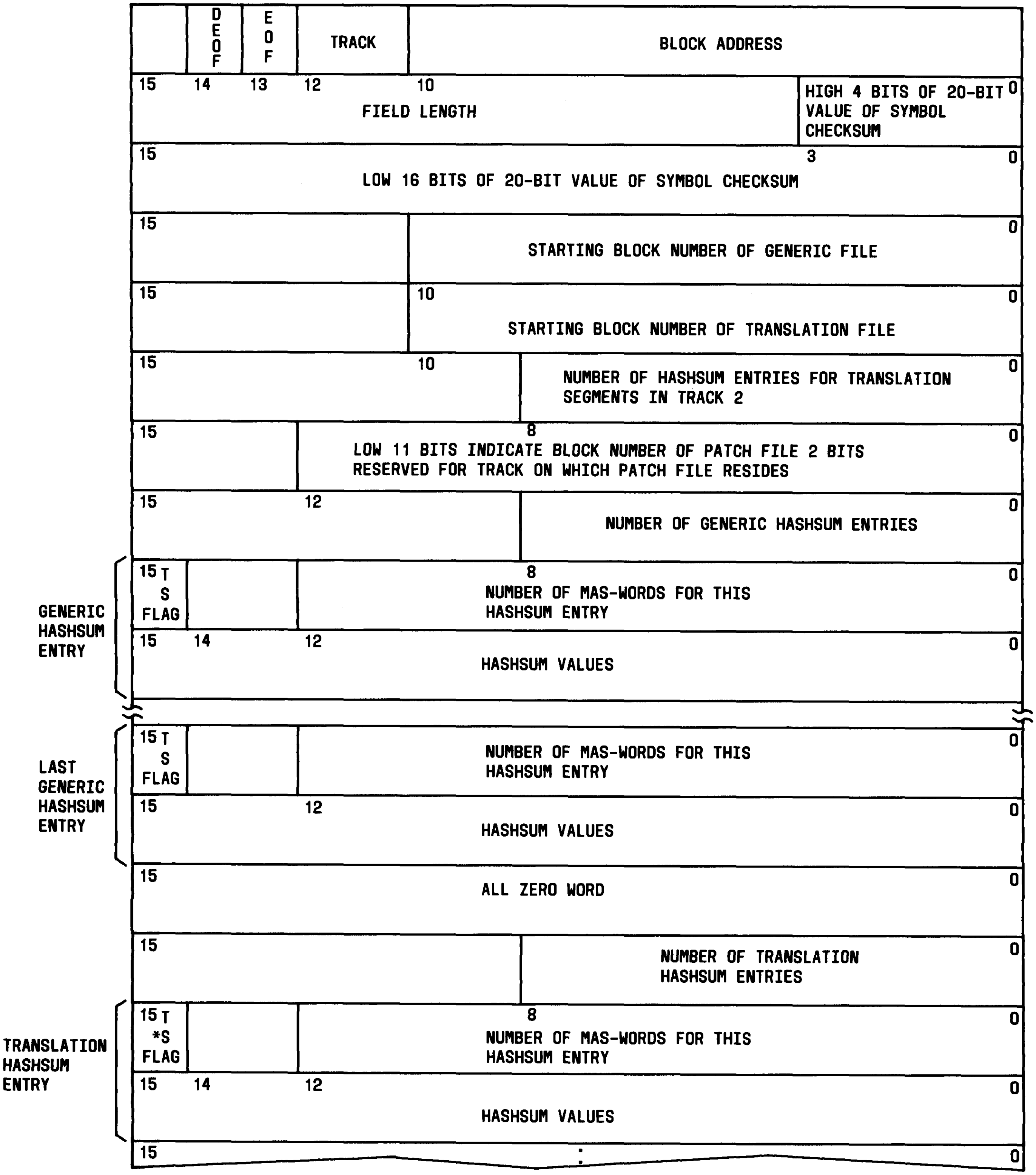
1. PRIMARY BOOTSTRAP FILE
2. BACKUP BOOTSTRAP FILE
3. AREA FORMATTED WITH ALL ZEROS
4. CHECKSUM FILE
5. GENERIC FILE
6. TRANSLATION FILE
7. PATCH FILE
8. BACKDATE FILE 1
9. BACKDATE FILE 2
10. PAGED PROGRAM FILE
11. TRAFFIC FILE
12. OFFICE RECORD FORMS FILE
13. DIRECTORY
14. LABEL FILE

Fig. 26—No. 3 ESS List 1 Tape Cartridge Format—3E3 Generic



1. PRIMARY BOOTSTRAP FILE
2. BACKUP BOOTSTRAP FILE
3. PRIMARY CHECKSUM FILE
4. GENERIC FILE
5. PATCH FILE
6. PAGED PROGRAM FILE
7. DIRECTORY
8. LABEL FILE

Fig. 27—List 2 Tape Cartridge Format—3E3 Generic


DEOF
EOF
TRACK
BLOCK ADDRESS
15
14
13
12
10
FIELD LENGTH
HIGH 4 BITS OF 20-BIT VALUE OF SYMBOL CHECKSUM
0
15
3
0
LOW 16 BITS OF 20-BIT VALUE OF SYMBOL CHECKSUM
15
0
STARTING BLOCK NUMBER OF GENERIC FILE
15
10
0
STARTING BLOCK NUMBER OF TRANSLATION FILE
15
10
0
NUMBER OF HASHSUM ENTRIES FOR TRANSLATION SEGMENTS IN TRACK 2
15
8
0
LOW 11 BITS INDICATE BLOCK NUMBER OF PATCH FILE 2 BITS RESERVED FOR TRACK ON WHICH PATCH FILE RESIDES
15
12
0
NUMBER OF GENERIC HASHSUM ENTRIES
GENERIC HASHSUM ENTRY
15 TS FLAG
8
0
NUMBER OF MAS-WORDS FOR THIS HASHSUM ENTRY
15
14
12
0
HASHSUM VALUES
LAST GENERIC HASHSUM ENTRY
15 TS FLAG
0
NUMBER OF MAS-WORDS FOR THIS HASHSUM ENTRY
15
12
0
HASHSUM VALUES
15
0
ALL ZERO WORD
15
0
NUMBER OF TRANSLATION HASHSUM ENTRIES
TRANSLATION HASHSUM ENTRY
15 T *S FLAG
8
0
NUMBER OF MAS-WORDS FOR THIS HASHSUM ENTRY
15
14
12
0
HASHSUM VALUES
15
0

* – TS FLAG IS NOT NORMALLY SET FOR TRANSLATIONS

Fig. 28—Checksum File—3E3 Generic

TABLE A

PROGRAM IDENTIFICATION TABLE

PROGRAM NAMES	PROGRAM TITLES	PROGRAM NUMBERS	MAJOR FUNCTIONS
BLMMA	Application Portion of the Base Level Monitor	PR-3H004	Provides tables that are used by the common systems base level monitor (eg, multiscan function tables) and invokes routines executed once per base level loop. Also contains an interface to the tape handler during interrupt processing. The program is resident and is executed at base level.
CBLM	Common Base Level Monitor	PR-1C950	CBLM is the focal point of the base level system since it determines the sequencing of all programs. Contains four of the major monitors (including the multiscan function controller) and calls the remaining monitors (including the tape handler - CTAPH). It is resident and is executed at base level.
CINIT	Common Initialization	PR-1C952	The program CINIT is the primary control program used for initialization.
CIPL	Common Initial Program Loader	PR-1C953	This program is used as a means of bootstrapping the memory.
CNRUTL	Common Nonresident Utilities	PR-1C954	This program consists of utility routines which may be nonresident. Included in these routines is the tape utility multiscan function which provides the capability to manipulate individual physical blocks of data on the tape. The contents of a particular block can be printed on the TTY, read into main store, or changed by being overwritten by the contents of a dedicated tape buffer (I-OBUF). In addition, the overwrite multiscan function utility which is contained in CNRUTL provides a means of making changes to the generic program.
*COPYLC	Line Class Code Translation Update Program	PR-3H022	This program is used to copy current nonresident translation files (line class code tables file) to backup (backdated) files, or vice versa. This program is nonresident.
CPAGM	Common Tape Paging Monitor	PR-1C955	The paging monitor loads the required nonresident program segments from tape into the paging buffer in main store for execution. The paging monitor is resident.
CSYSUB	Common System Subroutines	PR-1C956	CSYSUB contains a collection of common system subroutines. Each subroutine is used by more than one program.

*Relates only to SO-2 Issue 4 or 4A generic.

TABLE A (Contd)

PROGRAM IDENTIFICATION TABLE

PROGRAM NAMES	PROGRAM TITLES	PROGRAM NUMBERS	MAJOR FUNCTIONS
CTAPH	Common Tape Handler	PR-1C957	The tape handler performs various tape operations requested by client programs. These operations include such functions as opening and closing files, read and write operations, and positioning of the tape. The tape handler is resident and portions are executed at base level, while other portions are executed at interrupt level.
CTAPM	Common Tape Maintenance	PR-1C958	This maintenance program diagnoses the tape data controller and minirecorder.
INITA	Application Portion of the Initialization Program	PR-3H006	INITA is used with the common system program CINIT to make CINIT applicable to the No. 3 ESS.
*LCSUB	Line Class Code Table Access Routines	PR-3H025	The program LCSUB is a collection of nonresident subroutines used to access the line class code tables which are nonresident translation tables and reside in files on tape.
MASACS	Main Store Access Routines	PR-1C965	The main store access routines are used to interface between the tape format and main memory during memory reload.
OPDATA	Office Data File Update Program	PR-3H080	The nonresident program OPDATA updates the current translation data file (TRNSLN) on each tape from the translation area of main store automatically once daily. In addition, an update of the current or one of the backdated translation data files can be initiated by a TTY request.
VRDATE	Verify Date of Translation Files	PR-3H068	The program VRDATE initiates the printing of the date for the current translation file (TRNSLN) and its backdate file. For SO-2 Issue 4 or 4A generic only, VRDATE also initiates the printing of the line class code table file (LCCTBL) and its backdate file. This program is, therefore, a client of the tape handler and is nonresident.

* Relates only to SO-2 Issue 4 or 4A generic.

TABLE B

TAPE MACROS

MACRO	FUNCTION
SEIZE	Gains access to all files on the tape and seizes control of TCB.
OPEN	Gains access to a specified file (and control of the TCB) and positions the tape at the beginning of the file or at a specified (optional) logical index into the file. Accesses tape numerically tied to the on-line processor if possible; otherwise, accesses the other tape.
OPENE (open explicit)	Gains access to a specified file on a specified tape unit (and control of the TCB) and positions the tape at the beginning of the file or at a specified (optional) logical index into the file.
OPENM	Allows the client to specify in the macro call the address of a data block containing the ASCII characters of the file name and the client number in the open operation rather than providing the information in the macro call. This gives the client flexibility in specifying the file to be opened.
DOPEN	Provides a data block table of files to be opened without actually opening the file. The client can then direct the tape handler to the file in the table to be opened.
READ	Causes a continuous read until an end-of-file is encountered. Control is transferred briefly to a specified address after each block is read for minimal work.
RD1BLK	Reads one block and transfers control back to specified client address.
WRITE	Writes a block onto tape. The tape handler determines tape position and formulates in the I/O buffer the block number and other control information associated with the first word in the block.
WRTAB (write absolute)	Writes contents for I/O buffer onto tape regardless of tape position. The tape handler does no checking of the position or block number determination.
POSITION	Positions the tape to any specified logical (index) block number in the file relative to the beginning of the file.
BACKSP	Moves tape backward one block at the time.
COPY	Copies file which is open to the same file on the other tape unit.
STATST	Determines if function requested has been completed (that is, if the TCB is idle).
CLOSE	Relinquishes access to a file and the TCB.