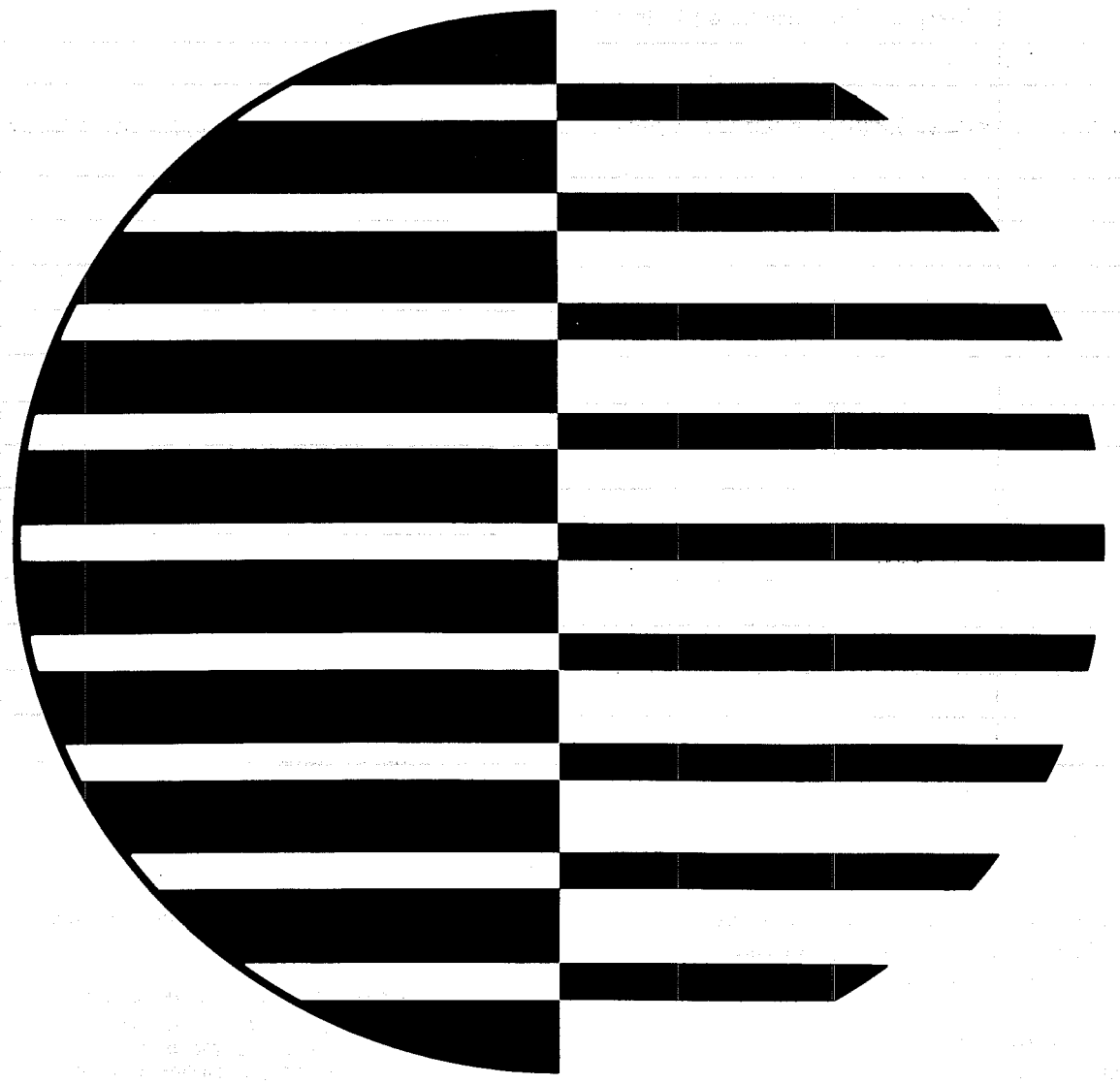
CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
SCOPE 3.1 Reference Manual

missing p 3-1, 3-2



60189400A	**REVISION RECORD**
REVISION	**NOTES**
1	Chapter 11 added; Contents adjusted accordingly.
(2-29-68)	
B	SCOPE 3 changes in Chapters 1, 2, 3, 9 and Appendix C. SCOPE 4 will retain
(3-28-68)	these changes except in certain cases noted as applicable only to SCOPE 3 and
	designated by brackets [ ] in this revision.

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Pub. No. 60189400A
February 1968

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Software Documentation
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA 94304

or use comment sheet in the back of this manual.

INTRODUCTION

The 6400/6500/6600 SCOPE 3.1 Operating System provides the user with easy access to and control of all the advanced capabilities of the CONTROL DATA® 6400/6500/6600 computers.

The basic computer includes one or two central processors, 10 peripheral processors, and 12 channels to which I/O devices can be connected. (The hardware is described in detail in the Computer Systems Reference Manual, Publication No. 60100000.)

The operating system is in constant control of all jobs, handling storage allocation, job scheduling, accounting, input/output control, and operator communication. When used on the dual processor 6500, the system allows automatic scheduling of dual central processors within the multi-programming environment.

CONTENTS

SYSTEM DESCRIPTION 1

1.1 HARDWARE/ SOFTWARE INTEGRATION

When the computer is deadstarted, all the peripheral processors (PP's) are forced to read, and can be filled with programs from a system tape. Once this has been done, each of the PP's is completely sovereign; other components of the machine cannot force PP activity; they can neither put information into its memory nor read information out of its memory. A PP can be requested to receive, transmit, or process information only. (Every PP must contain a program for receiving and responding to requests.)

A central processor (CP), though a main component from the user's viewpoint, is completely in the power of every PP at all times. Any PP, by executing one of its own instructions, can alter all the registers of a CP, write new information into central memory (CM), or read information out of CM. A CP, on the other hand, cannot directly affect a PP in any way.

One PP is in permanent, supreme control of the system; the other PP's cannot perform any function not approved in advance by the controlling PP. A communication area in central memory is assigned to each PP. The first word of each communication area is the input register of the associated PP; the second word is the output register, and the remainder is the message buffer.

PP number 0 contains the monitor program (MTR) that oversees or controls all other activities. PP number 9, under the supervision of MTR, is permanently assigned to the console typewriter and display scopes. The other PP's, 1 to 8, are initially assigned to read their input registers over and over. The monitor makes a request of a PP by putting a significant word into the input register of that PP. Upon finding the request, the PP obeys it (or determines that it cannot do so), indicates to the monitor via its output register that it has finished, and returns to its idling state of continually reading its input register. Thus all requests to a PP other than the monitor are communicated through the input register of that PP.

Each PP (other than MTR) uses its output register for requests to the monitor and for completion status of the requests. The monitor periodically reads the other PP output registers in turn, looking for requests, and zeros them whenever the requests have been satisfied.

Although the primary task of a PP is to act on request from MTR, on occasion a PP must request the cooperation of other PP's. Such requests are routed through the monitor. Furthermore, a PP must request permission from the monitor before using an I/O channel. Since every PP is capable of connecting itself to any channel, it is essential in preserving order that only one PP at a time try to use any one channel. To avoid an attempt by two PP's to use the same channel (which would disrupt both PP's and the channel), the monitor maintains a list of channels and their status. Before a PP can use a channel, it must request the monitor to assign that channel for its exclusive use. When finished with the channel, the PP requests the monitor to note that the channel is free.

1.2 MULTI-PROGRAMMING

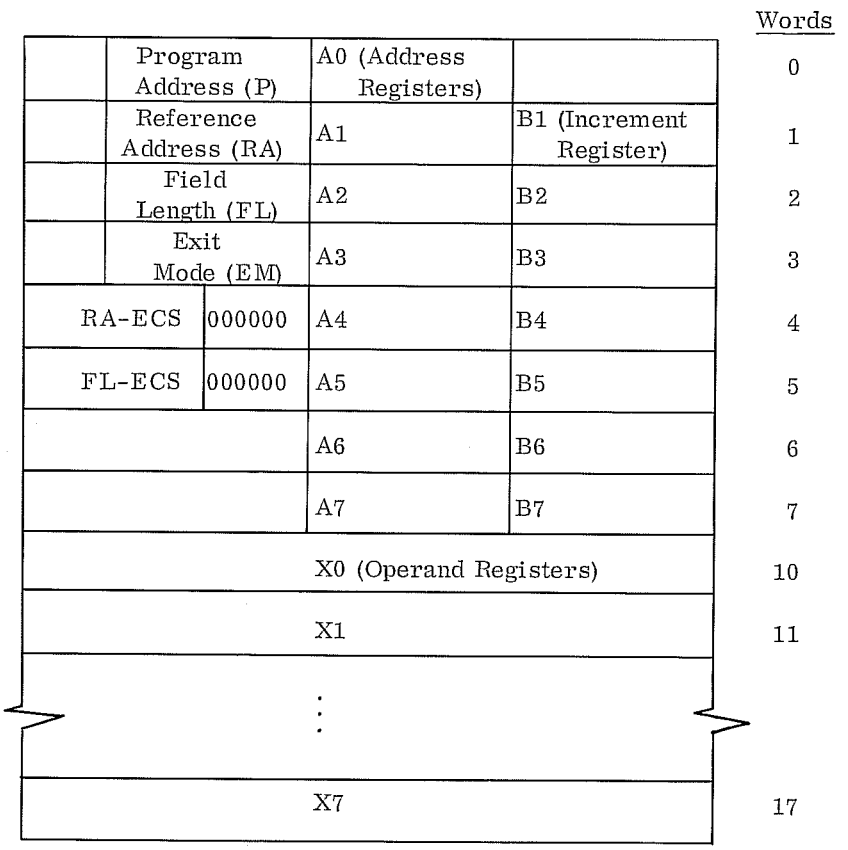
1.2.1 CENTRAL MEMORY USAGE

CM low core called the Central Memory Resident is reserved for various system tables and is never accessible to a user's CP program. The remainder of CM is allocated by the monitor to user jobs as they are selected on a priority basis for execution. SCOPE can supervise as many as seven separate CP jobs.

1.2.2 CONTROL POINTS

Eight areas, numbered 0 to 7, are designated as control points within Central Memory Resident (CMR). Every CP program is assigned to a control point; control point 0 is used for system functions.

When a job is in CM, the control point area to which it is assigned includes the following information: job name, length, starting address in CM, time used so far, I/O equipment assigned to job, and its control statements. The control point area also contains an exchange package, a 16-word section consisting of the contents of all CP registers used in executing a program. This information is necessary to start or resume a program. The format of the exchange package follows:



A central memory program can be easily relocated by moving the program in memory and resetting the reference address (RA) in the exchange jump area. All central processor reference addresses to central memory instructions or data are relative to the reference address. The RA and field length (FL) define the central memory limits of a program (RA plus FL). Field length is the total program length. The program address register (P) defines the location of a program step. Each reference to memory is made to the address specified by P + RA. In starting a program for the first time, the monitor provides the values for RA, FL, and P in the exchange area.

To make CP execute a program, the monitor causes the program to be stored in central memory, and then executes an Exchange Jump instruction. This PP instruction exchanges the information in the CP registers with the contents of the exchange package. When the instruction is issued, the initial address of the exchange package is contained in the PP's A-register.

To illustrate multiprogramming on the 6400/6500/6600 a simple situation is described below in which the monitor causes the CP to execute two programs alternately, for one second at a time. All numbers are octal.

Program One is stored in cells 1000 through 1777 of CM, and Program Two in cells 2000 through 2777. Each program is to begin at its first location; at this point the contents of the A, B, and X registers are not significant. The exchange areas are cells 0100 to 0117 for Program One; cells 0300 to 0317 for Program Two. To start Program One, the monitor stores:

xx000000xxxxxxxxxxxx in cell 0100

xx001000xxxxxxxxxxxx in cell 0101

xx001000xxxxxxxxxxxx in cell 0102

Then the monitor puts 0100 in its own A register and executes an exchange jump. This puts the current contents (which are irrelevant) of the P, RA, and FL registers into cells 0100, 0101, and 0102, respectively, and sets the P register to 000000, the RA register to 1000, and the FL register to 1000. The central processor begins to function as a computer with a memory of 1000 words (set by FL register), with location 0 at address 1000 in central memory (set by RA register) and 0 initially in the program counter (set by P register). The 1000 CM cells that any PP can access, with addresses 1000 to 1777, are now the only memory that CP can access; it numbers them 0000 to 0777.

The monitor prepares to start Program Two by storing:

xx000000xxxxxxxxxxxx in cell 0300

xx002000xxxxxxxxxxxx in cell 0301

xx001000xxxxxxxxxxxx in cell 0302

It waits until Program One has run for one second (according to the hardware clock) and then puts 0300 in its own A register and executes an exchange jump.

When an exchange jump interrupts the central processor, several hardware steps insure that the interrupted program is left in a state for re-entry:

1. Instructions cease after all instructions from the current instruction word in the stack are issued.
2. P is set to the address of the next instruction word to be executed.
3. Issued instructions are executed.
4. The two programs are exchanged.

CP register contents are relevant now because they will be needed to resume running Program One a second later. The exchange jump stores the current contents of all CP registers in CM cells 0300 to 0317, and in exchange puts 000000 in the P register, 2000 in the RA register, and 1000 in the FL register of the central processor. The CP begins to function as a computer with a memory of 1000 words, with location 0 at address 2000 in central memory and 0 initially in the program counter. The 1000 cells that any PP can access, with addresses 2000 to 2777, are now the only memory that the CP can access; it numbers them 0000 to 0777.

After another second has elapsed, the monitor has only to put 0300 in its A register and execute an exchange jump to restore the CP to exactly the condition it was in when the running of Program One was discontinued at the end of the first second. Program One is resumed as if there had been no interruption. At the same time, the registers, as they stand at the end of the first second of processing Program Two, are stored in CM cells 0300 to 0317. One second later, the monitor can again interrupt Program One and resume Program Two.

In the control point system, a 3-bit number is enough to identify a job in process. For instance, a user program may request the reading of magnetic tape which gives rise to several requests passed back and forth among the PP's. Each of these requests needs only 3 bits to connect it with the user program. When information about that program is needed during the processing of any of these requests, it is located easily.

1.3 FILES

1.3.1 ACTIVE FILES

SCOPE is a file-oriented system: all information contained within the system is considered to be either a file or part of a file. Active files — those immediately available to the system at any moment — are defined to be any of the following:

All jobs (each job is a file) waiting to be run. This set of files is called the job stack or input queue.

Output files from jobs which have been run and are waiting to be disposed of by printing, punching, etc.

Jobs (files) presently in some state of execution.

Files currently being used by the jobs in execution.

Common files, which maintain active status by specific request.

The monitor keeps a list of active files in the file name/status table which is part of Central Memory Resident. This table stores the following information: the name of a file, the control point to which it is assigned if in use, its type, the kind of I/O equipment to which it is assigned, and information about the progress of reading or writing the file.

The four types of active files are: input, local, output, and common. As a job progresses, the job file goes through several type changes.

When a job file is read from the card reader, it is copied onto disk storage and becomes an input file; it is not assigned to any control point. The file name is that name given on the job control card. The file name/status table contains a priority (from the control card) for the file which becomes the priority for the job.

When the job is assigned to a control point, the input file becomes a local file; and its name is changed to INPUT. The original name of the input file is saved in a word of the control point as the name of the job. New local files named OUTPUT, PUNCH, and PUNCHB will be established, if referenced, and given disposition codes of print, punch coded and punch binary, respectively.

INPUT, OUTPUT, PUNCH and PUNCHB are all local files in disk storage. They are the immediate source of card input and the immediate destination of printer output and coded and binary card output. Because several jobs may run concurrently at different control points, several local files called INPUT, several called OUTPUT, and several called PUNCHB are in the file name/status table simultaneously. When a local file is sought in the table, both the name and the control point number are used to identify it.

When a job terminates, the local file called INPUT for the assigned control point is released. Entries in the file name/status table for the local files called OUTPUT, PUNCH, and PUNCHB for that control point are altered so that their names are changed to the name of the job itself, which is found in the control point area. The control point is then released.

Other local files can be created by the job. For instance, the first time a job references a file called RASP, the system consults the file name/status table entries for a local file of that name assigned to the job's control point. If one does not exist, a file is immediately created, initially consisting only of an end-of-information mark. This file is named RASP and entered into the file name/status table as a local file assigned to that control point, and the job references the RASP file. When the job terminates, all local files created in this manner are completely eliminated from the system.

The fourth type of active file — the common file — is a local file for which active status is maintained by a control card request, so that the file does not disappear when the job originating it is terminated.

Example:

A job contains the control statement:

COMMON RASP.

If the control statement generates a local file called RASP, that file does not disappear when the job terminates. The entry in the file name/status table for the local file RASP is altered so that it no longer belongs to any control point, and its type will be common.

It is assumed that the file name/status table did not already contain an entry for a common file called RASP. However, if it did contain such an entry, when a job is processed that contains the control statement COMMON RASP., file RASP would be assigned to the control point of that job. RASP would then be available to that job just as if it were a local file.

If a third job contained the control statement COMMON RASP. and if, when this card was processed, it was found that the common file RASP had been assigned to the control point of a running job, the earlier job would have to terminate and file RASP be released from its control point before RASP would be available to the latest job.

To eliminate a common file like RASP from the system, a job must contain the control statement COMMON RASP. and a later control statement:

RELEASE RASP.

When the latter control statement is processed, RASP is converted from a common file to a local file, but not otherwise altered. When the job is terminated, the local file RASP is destroyed.

1.3.2 LOGICAL RECORDS

All files within the SCOPE system, regardless of type, are organized into logical records: for input files, through the ordering of control cards; for output files, through the language translator or other program producing the output; otherwise, logical record generation is up to the user.

Since the logical record concept is defined for all devices, files may be transferred between devices without losing their structure. The physical format of a logical record is determined by the device on which the file resides. The physical record unit size (PRU) is the smallest amount of information that may be transferred during a single physical read or write operation for each device within the system. Logical records are written as one or more PRU's, the last of which is short or zero-length. A zero-length PRU is written if the logical record is an even multiple of the PRU size or if a write operation was requested with no data in the buffer. A zero-length PRU contains fewer bits than a CM word.

Coded files on 1/2-inch magnetic tape receive special treatment. Within the SCOPE system, all coded information is carried in display code; therefore, a conversion to external BCD must be made before writing on the tape. Translation is character-for-character.

The display code end-of-line mark (12-bit zero byte) is converted to the external BCD characters 1632_8. The display code end-of-line mark is recognized only when it appears in the lower 12 bits of a central memory word.

PRU Sizes (decimal)

6638 disk	64 CM words	SCOPE 3 only
6603 disk	64 CM words	
863 and 865 drum	64 CM words	SCOPE 3 only
854 disk pack	64 CM words	SCOPE 3 only
coded 1/2 inch tapes	128 CM words	
all other tapes	512 CM words	

LEVEL NUMBERS

Related logical records within a file may be grouped by the user into an organized hierarchy. The level number (0-17_8)† of a logical record is contained in the short or zero-length PRU which terminates the record. This PRU is the level mark. The level number is declared in the write request. If no number is specified, a level of 0 is assigned. If, when no data is in the buffer, a level number is specified in a write request, a zero-length PRU containing the level number is written. A write end-of-file request causes a zero-length PRU of level 17 (logical end-of-file mark) to be written. The level mark appended to each logical record is not placed in the circular buffer when the file is read; but it is returned as part of the status information.

†Level number 16 should not be used for a job which includes a request for a checkpoint dump as this level number is used in a unique way by the checkpoint dump program.

The lowest level within a file is associated with a single logical record. A higher level defines a set of records consisting of the logical record at that level plus all preceding records at a lower level.

For instance, a file might be regarded as a multi-volume book; level 0 would be equivalent to a page, level 1 to a chapter, and level 2 to a volume. In the following example, the lowest level 0 is associated with a single logical record called a page; level 1 marks delimit a group of pages called chapters; chapters are grouped by level 2 marks into volumes. A reference to a logical record of level 1 includes all information between the referenced level 1 mark and the succeeding one. Included, therefore, will be several logical records as shown in the diagram.

Logical Record	Level Mark	Page	Chapter	Volume
1	0	1		
2	1	2	1	
3	0	3		I
4	0	4	2	
5	2	5		
6	0	6		
7	0	7	3	
8	1	8		
9	0	9		
10	0	10		
11	0	11	4	II
12	1	12		
13	0	13		
14	2	14	5	
15	0	15		
16	1	16	6	
17	0	17		III
18	0	18	7	
19	2	19		
End of Information				

The format of the level mark varies depending on the device type on which file resides, as follows:

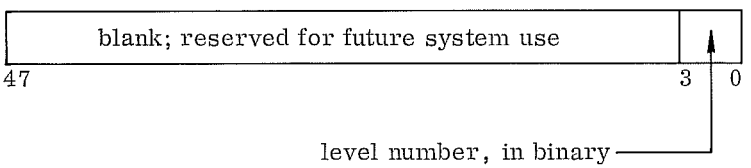
Card Files

Each logical record is terminated by a card with 7, 8, 9 punches in column 1. Columns 2 and 3 may have an octal integer, 00-17, to denote level number. Level zero is assumed in the absence of punches in columns 2 and 3.

The end of information is signaled by a card with 6, 7, 8, 9 punches in column 1.

1-inch Magnetic Tape Files, Disk Files, and Binary Mode 1/2-inch Magnetic Files

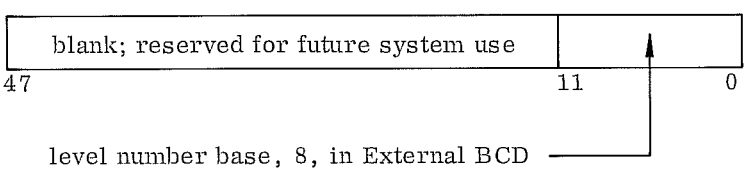
Each logical record is terminated by 8 characters (48 bits) as follows:



If the last information in the logical record does not fit exactly into a physical record unit, the 8-character marker is appended to the last written PRU; otherwise, the marker is written as a single PRU of zero length.

Coded Mode 1/2-inch Magnetic Tape Files

Each logical record is terminated by 8 characters as follows:



If the level number is zero, an external BCD blank is used as the value. If the last information in the logical record does not fit exactly into a physical record unit, the 8-character marker is appended to the last written PRU; otherwise, the marker is written as a single PRU of zero-length.

1.4 RANDOM ACCESS

Random access files can be created: Records within a disk file can be addressed directly. A disk address refers to pointers to system tables. When random file processing is requested, the disk address is returned when a logical record is written. A disk address is accepted from the user when a logical record is read.

Generally, the disk addresses returned when the file is written are gathered into an index. SCOPE 3.1 provides a routine (IORANDM) which automatically formats one of two types of indexes containing either named or numbered records. In either case, the first word is an indicator of the index type: +1 or -1. If the file contains only numbered records, sequencing of disk addresses in the index corresponds to the record numbering; the first address belongs to record one, the second to record two, and so on. The index need be only n + 1 words in length, where n is the maximum number of logical records in the file. The first word of such an index is set to +1 the first time the file is written.

If the file includes named records, the index contains a two-word entry for each record. The length of this index is 2n + 1 words. The first word of the index entry contains the record name, one to seven display code characters, left justified with zero fill. The second word of the entry contains the disk address of that record. If a given record has no name, the first word of its index entry contains zero, and the record must be accessed by its sequence number. The first word of a name index is set to -1 the first time the file is written.

When a record is written with a name that already appears in the index, the new record is substituted for the existing record. When a record is written with a name that does not appear in the index, IORANDM places the new name in the index at the lowest unoccupied position and assigns the number of that position to the record. When there is no space in the index for a new name, the request is rejected and index full status is returned.

When a record is written by number and the records for that file can be named, the name is not disturbed.

Other forms of indexes may be defined with a central processor subroutine which sets fields in the file environment table (FET) and locates the records within the index.

When files contain many logical records, multiple levels of indexes can be defined to conserve central memory space. When a multi-index file is written, logical record disk addresses are directed to a subindex buffer. When the buffer becomes full, the subindex itself can be written as a logical record in the file: the subindex disk address is directed to a main, or primary, index. The forms of the primary and subindexes can be that supplied by IORANDM or by a user-supplied routine. They need not be the same type.

Random files which are to endure between runs in a job or between jobs should be declared as common files. At the end of a run which creates such a file, the user should close the file with an unload option. The system then automatically appends the contents of the index buffer specified in the FET to the end of the file. When the file is to be read, the user must initially open the file with an index area specified in the FET. The system then reads the index record into the specified area.

1.5 FILE LABELS

SCOPE system file labels are defined for files recorded on 1/2-inch magnetic tape only [For SCOPE 3 only, a system device label is defined for private disk packs.] The labels are described and designed to conform to the Proposed USA Standard, Working Paper Magnetic Tape Labels and File Structure for Information Interchange, produced by ISO, the International Organization for Standardization, Technical Committee ISO/TC 97, Sub-Committee 2.

Tapes containing a system label are recognized as labeled tape. All other tapes are considered unlabeled. Label processing is not provided for non-standard labeled (non-labeled) tapes. Tape labels are recorded at 556 bpi. All system labels are 80 characters. Labeled tapes are checked by the system for file name, reel number, creation date, expiration date, and edition number. Labeled tapes are protected from accidental destruction by checking the creation and release dates in the file header label. This label is delivered to the circular buffer for an input file, so that the program may check it further as required. Unless the UP bit is set in the FET, reel swapping for a multi-reel tape file is automatic. The system executes two function calls: CLOSER, lfn, UNLOAD and OPEN, lfn, REEL. Since these calls are issued by the system, the file header label is delivered for the first reel only.

The following terms are defined in conjunction with the SCOPE system file labels.

Volume: Synonymous with reel of magnetic tape.

Volume Set: A collection of related volumes in which one or more files are recorded. A volume set may consist of:

- A single volume containing one file
- A single volume containing several files
- Several consecutive volumes containing one file
- Several consecutive volumes containing several files

Tape Mark: A one-character record, 17_8, plus check character recorded in even parity. The tape mark separates label information for file information.

The first four characters of labels identify the type.

Type	Identifier
Volume header label	VOL1
Volume trailer label	EOV1
File header label	HDR1
File trailer label	EOF1
Device header label	DEV1

Label formats are described in Appendix C.

1.5.1 TAPE FILE STRUCTURE

SCOPE standard system labels and tape marks establish the tape file structure according to the following rules. Required labels are indicated by a 4-character identifier, and tape marks are indicated by asterisks.

Single-Reel File

VOL1 HDR1*...Data Blocks...* EOF1**

Multi-Reel File

VOL1 HDR1*...First Volume Data...* EOV1**

VOL1 HDR1*...Last Volume Data...* EOF1**

Multi-File Reel

VOL1 HDR1*...File A...*EOF1 * HDR1*...File B...*...EOF1**

Multi-Reel Multi-File

VOL1 HDR1*...File A...* EOF1 * HDR1*...File B...* EOV1**

VOL1 HDR1*...Continuation of File B.....* EOV1**

VOL1 HDR1*...Last of File B...* EOF1 * HDR1*...File C...* EOF1**

Volume Header Label

The first PRU in the volume must be a volume header label; it may not appear elsewhere.

File Header Label

Every file must be preceded by a file header label and every file header must be preceded by a tape mark. When a volume ends within a file, the continuation of that file in the next volume must also be preceded by a file header.

File Trailer Label

A file trailer label is required as the last block of every file. A file trailer must be followed by a tape mark, and if it is the last file trailer in the volume, two tape marks are required.

Volume Trailer Label

When a volume ends within a file, the last PRU of the file in that volume must be followed by a volume trailer label which must be preceded and followed by tape marks.

When end-of-volume and end-of-file coincide the labeling configuration is one of the following:

```
                                        ...File A...* EOV1* *

VOL1 HDR1* * EOF1 * HDR1*...File B...
       (A)       (A)      (B)

            ...File A...* EOF1 * HDR1* * EOV1* *

VOL1 HDR1*...File B...
       (B)
```

SCOPE 3 only

1.5.2 PRIVATE DISK PACK FILE STRUCTURE

Disk packs privided for the 854 disk drives, are designated as shared system devices or as privately assigned, dismountable devices.

Shared packs are dedicated to the system as long as any active files are assigned partially or totally to them. Recording and retrieval of data from these files is handled entirely by the system. Shared packs containing active files may not be removed from the system; an operator-controlled function is defined to protect them.†

† The structure of shared system files is defined in Section 9.2.

For SCOPE 3 only

A single file, sequential or random, is assigned to each private disk pack by a user. File overflow to a second pack is not possible; the entire file must be contained on a single pack. A private pack cannot be removed while its file is active to the system. A private pack is associated with a single job. Access is not shared with jobs active at control points other than the assigned control point. At unloading, the recorded data of a private pack consists of four parts if it is sequential, or five parts if it is random:

Device label

Copy of record block reservation (RBR) table

Copy of record block assignment chain, taken from record block table (RBT)

Recorded data of the file

Index of the file (random file only)

When the file is mounted and made active, the recorded data is restored in central memory tables. Copies defined on the packs are rewritten prior to removal of the pack.

Defined on a disk pack are 2000_{10} record blocks of five PRU's each. Each record block is recorded on a single track of the disk pack; two revolutions are required to read or write the entire record block. PRU sector assignments for a full track are shown below; sector 15 of each track is unused.

Record Block Structure (854)

PRU	Sectors
0	0, 1, and 2
1	6, 7, and 8
2	12, 13, and 14
3	3, 4, and 5
4	9, 10, and 11

The first two record blocks of private packs record the device label, the RBR copy, and pointers to the RBT chain. The 12-bit byte contents of the system RB's for the first two record blocks are shown below:

For SCOPE 3 only

RB	PRU	Bytes	Contents
0	0	0-39	Device label
0	0	40-229	RBR table copy
0	0	230-318	Zeros - reserved for future use by the system
0	0	319	Check-sum
0	1	0-9	First RB address chain entry
0	1	10-19	Second RB address chain entry

Sequentially ordered RB address chain entries define record blocks in the order data was recorded. An RB address entry has the format: 10 * cylinder + head group. The first RBT entry defines a minimum of four record blocks; the remaining entries define at least eight record blocks. There are 32 entries per PRU; the nine remaining PRU's are sufficient to define the entire disk pack surface.

Data recording begins physically (not necessarily logically) at RB 2. When an index is included, it is stored at the logical end of data.

JOB PROCESSING 2

A job consists of one file of punch cards or card images. The first logical record of a job file consists of the control cards which identify the programs and data files and control the sequence of program executions (runs). Control cards specify how the job is to be processed; they determine all operations performed on subsequent logical records of the job file.

2.1 JOB FLOW

SCOPE begins processing by reading the job card. It copies the job file on disk storage and adds the name of the job to the list of input files.

When a control point is available for the job and the required amount of memory is free, the job is brought to the control point through the following steps:

- Memory is allocated; jobs already in the computer may be moved forward or backward in central memory.
- The first record (or part thereof) of the job deck (control card record) is copied into the control statement buffer of the control point, and a pointer is initialized to indicate the first control card.
- All data following the control card record in the job deck is made available to the control point as a file called INPUT.
- The first control statement in the buffer is executed, and the pointer moves to the next control statement. This begins the first run within the job. When that run is completed, the next control statement is executed, beginning the second run. When control statements are depleted or an EXIT card occurs, the job is terminated.

Each job must begin with a job card and end with a file separator card. All control cards must appear between the job card and the first record separator. The end of the control cards is signified by a 7, 8, 9 punch card (end-of-record) or a 6, 7, 8, 9 punch card (end-of-information) if the job consists of control cards only.

2.2 CONTROL CARDS

Control cards have two fields. The first contains the flag word, beginning in column 1. Flag words described in this section are reserved for the system and may not be used as program call names. The second field is optional; it may contain one or more parameters, separated by commas. The two fields are separated by a comma or left parenthesis; a blank separator is also recognized on the REQUEST, COMMON, RELEASE, SWITCH, and MODE cards. The parameter field is terminated by a period or right parenthesis, and a terminator must be present even when no parameters are specified.

JOB CARD

n, Tt, CMfl, ECfl, Pp.†

The first control card of a job must indicate the job name, priority, central processor time limit, and memory requirements. Fields are separated by commas and the last field is terminated by a period. Blanks are ignored in a job card. Fields other than n may appear in any order as they are identified by leading characters indicated above by capital letters.

n — Alphanumeric job name (1-7 characters); must begin with a letter. To assure unique job names, SCOPE replaces the last two characters with a system generated value. If only a job name is specified, installation-declared values are assumed for the remaining fields.

Tt — t = central processor time limit for the job in seconds; a maximum of 5 octal digits. Value may not exceed 32767_{10}. Time limit must suffice for the whole job including all compilation and execution.

CMfl — fl = total central memory field length of the job; a maximum of 6 octal digits. The field length (storage requirement) is rounded up to a multiple of 100_8 by the system. Length cannot exceed:

$360,000_8$ on a 131K machine
$163,000_8$ on a 65K machine
$61,000_8$ on a 32K machine

ECfl — fl = total extended core storage field length given as the number of 1000_8-word blocks required. Value may not exceed 7777_8.

Pp — p = priority level, in octal, at which job enters the system. $1 \leq p \leq 2^k-1$; k is an installation option ≤ 8; 1 is the lowest priority.

---

† Compatibility with job card formats used by previous systems may be obtained at installation option.

SWITCH CARD SWITCH n.

Pseudo sense switches 1-6 may be set for reference by a subsequent program. Settings are preserved at the control point and copied to RA for use by the central program. Switches may be changed by console commands, OFFSW and ONSW.

MODE CARD MODE n.

MODE is used to select exit or stop conditions for a central processor program. The exit selections (n) are loaded into the exchange jump package. Upon an exchange jump, the selections are stored in the central processor and the exit occurs as soon as the selected condition is sensed. The exit mode is set to 7, if not otherwise specified.

n value	Exit Condition
0	Disable exit mode - no selections made
1	Address is out of range because of: Attempt was made to reference central memory or extended core storage outside established limits Word count in extended core storage communication instruction is negative Attempt was made to reference last 60-bit word (word 7) in relative address FL ECS.
2	Operand out of range, floating point arithmetic unit received an infinite operand.
3	Address or operand is out of range.
4	Indefinite operand, floating point arithmetic unit attempted to use an indefinite operand.
5	Indefinite operand or address is out of range.
6	Indefinite operand or operand is out of range.
7	Indefinite operand or operand is out of range or address is out of range.

Example:

MODE 3. Selects address out of range and operand out of range as stop conditions.

RESTART CARD

RESTART, pckp, CLEAR, *

When a job which has taken checkpoint dumps terminates abnormally, it may be restarted from the last checkpoint dump with this control card. pckp is the unique file name included on the REQUEST card for the run. CLEAR is included when no restart is required. The * is acknowledged only when CLEAR is given. The * indicates that although the original job is not to be restarted, all files associated with the job should be processed according to their disposition code. If only certain files are to be so processed, they may be listed in place of the *, for example:

RESTART, pckp, CLEAR, lfn, lfn_2, lfn_3, . . .

COMMENT CARD

COMMENT. comments

The period must appear in this card. Characters following the period through column 80 are entered into the dayfile and displayed.

EXIT CARD

EXIT.

The EXIT card can be used to separate the control cards for normal execution from a group of control cards to be executed in event of error exit as listed below:

Error Flag	Condition	
1	Time expired	Job has used all CP time it requested. Further attempts to use the CP in any way will cause termination.
2	Arithmetic error	CP error exit has occurred.
3	PPU abort	PP has encountered an illegal request such as illegal file name or request to write outside the job field length.
4	CPU abort	Central program has requested that the job be terminated.
5	PP call error	Monitor has encountered a PP call entered in RA+1 by a central program.
6	Operator drop	Operator requested job to be dropped.

Conditions 3 and 5 can occur if a program accidentally writes in RA+1.

When the above conditions occur, an error flag is set at the control point. For flags 1, 2, 5 and 6, a dayfile message is issued; for 3, the fault-finding PP issues a message.

When an error flag is set, a search is made for the next EXIT control card. If none is found, the job is terminated. If an EXIT card is found, the error flag is cleared and succeeding control cards are processed. If an EXIT card is encountered when no error flag is set, the job is terminated normally at that point.

Example:

MYJOB,P1,T400,CM50000.	Job card
REQUEST TAPE5, WT.	Request scratch tape
RUN.	Compile and execute
EXIT.	
DMP.	Dump exchange package
DMP,1000.	Dump first 1000_8 words of storage
7_{8_9}	Record separator
(program)	
7_{8_9}	Record separator
(data)	
$6_{7_{8_9}}$	End of information

Dumps are made only when an error condition occurs.

2.3 PROGRAM EXECUTION

These control cards are used to load and execute files. The SCOPE control card format described below pertains to the EXECUTE and program call cards. All numbers used are decimal. The card may be a unit record of up to 80 characters including freely interspersed blanks. The general SCOPE format is:

Name List Comment

Name and list are required fields; comment is optional. Name is a string of one to seven alphanumeric characters beginning with a letter. Comment is a string of Hollerith characters composed from the set defined in Appendix A.

List contains parameters to be used by the program being loaded. The contents of list can vary greatly. If parameters are not required, list is simply a period. Parameters may be enclosed in parentheses or preceded by a comma and concluded by a period. The list may contain as many parameters as fit on a card expressed in one of three forms: P, P = 0, or P = Q where P and Q are strings of one to seven alphanumeric characters. P may specify one to four types of information flow: input, BCD output, binary output, or special information. Q specifies the name of the file involved. In all cases, the program to which P only is transmitted will take the action defined for P; the program to which P = 0 is transmitted will negate the action defined for P; and the program to which P = Q is transmitted will take the action defined for P using the file named Q in place of an otherwise assumed file.

LOAD CARD

LOAD (lfn)

This card directs the system to load the file named lfn into central memory. If lfn is INPUT, loading begins from the current position of the file. All other files are rewound by the system prior to loading. Loading terminates when the end-of-information or an empty record is encountered. All loader directives must appear in the named file before any subprogram. These directives specify whether overlay, segment, and section processing is required. Overlays, segments, and relocatable binary decks may be loaded with the LOAD control card. The first record of the file lfn specifies the kind of loading operations to be performed.

If subprograms are to be loaded from more than one file, more than one LOAD card is needed; but the first record of the first file always determines the kind of loading for all subsequent LOAD cards.

EXECUTE CARD

EXECUTE (name, $p_1, p_2, \ldots, p_n$)

Name is the entry point of the program to be executed once loading is completed. If name is absent, the last transfer address (XFER, appendix D) encountered is used. The parameters p_i are passed to the program to be executed.

The EXECUTE card causes completion of loading. This process includes filling out all unsatisfied references with entry points from the system library except where inhibited by segment parameters.

For segment or overlay operations, program execution begins in the first segment or the main overlay. Subsequent segments or overlays must be loaded by user calls from these programs.

PROGRAM CALL CARD

name $(p_1, p_2, \ldots, p_n)$

Initially, the file name/status table is searched for this name. If found, subprograms are loaded from the named file, bypassing with a message on OUTPUT, any routines already loaded by LOAD cards. The file is rewound before loading. If name does not appear in FNT/FST, the system library is searched and matching subprograms are loaded. Loading is completed; if no fatal errors are found, execution begins at the specified name. The parameters p_i are passed to the program to execute.

Example: LOAD (BTGN)
EXECUTE. } = BTGN.

To replace one subprogram with a subprogram of the same name from another file, a possible sequence is:

```
LOAD (HOO45)
LAU36.
```

The subprograms will be loaded from the file HOO45 and those of the same name on LAU36 will be bypassed.

NOGO CARD

NOGO.

When NOGO is encountered, the loader processes the loaded program in the same manner as for an EXECUTE card; however, the program is not executed. This card permits mapping a program, bypassing execution, and continuing other portions of the job.

2.24 EQUIPMENT ASSIGNMENT

If a file is not specifically assigned by a REQUEST card or REQUEST function, the system assigns that file to disk storage. A job need not assign the card reader, printer or punch for normal input/output from compilations, assemblies, etc., as this is done automatically by the system. In addition, any file named OUTPUT, PUNCH or PUNCHB will always be printed, punched, or punched binary, respectively, by the system when the job is completed.

A REQUEST card or function must be given to assign a file directly to a private device. The device assigned to the requesting control point becomes the private source or destination of files for that job. As job control cards are processed in order, required private equipment assignments must precede any reference to the corresponding private file.

REQUEST CARD REQUEST,lfn,dt,dc,x.

This card requests the operator at the system display console to assign a peripheral unit to a file and declares properties of the file and unit. The job waits for operator action before proceeding. With the exception of a disk, any equipment to be used must be specified on a REQUEST card or in a REQUEST function.

Because the control cards of a job are processed in order, equipment assignments must be made before the file is referenced. The parameters after the first may appear in any order. Successive blanks, commas, periods, left or right parentheses are ignored. If a parameter is listed more than once or is in error, a message is issued and the job is terminated.

lfn Logical file name (1-7 digits or letters) must begin with a letter. This is the name of the file to which equipment is to be assigned and the name by which the user refers to the file within his program. A REQUEST card must have at least one parameter, and the first parameter is assumed to be lfn.

dt Designates the type of device to which the file is to be assigned. If the dt parameter is absent, any equipment may be assigned by the operator. If dc is specified, the operator must assign the proper type of device. dt is written as yxx. xx is the mnemonic for the equipment type as listed below:

CP	card punch
LP	line printer
MT	1/2" magnetic tape
LO	1/2" magnetic tape at density 200 bpi
HI	1/2" magnetic tape at density 556 bpi
HY	1/2" magnetic tape at density 800 bpi
WT	1" magnetic tape
Dnnnn	disk; nnnn may be one of the codes under device type field in the FET (section 3.1.1)
CR	card reader

When the equipment type is MT and the tape has SCOPE system labels, the volume header label is set to indicate density as follows:

Input tape is read at density specified in volume header label.

Output tape is written at density specified by an installation parameter.

Code and Status (CS) (18 bits)

The CS field is used for communication of requested functions and resulting status between the central processor program and the peripheral processor input/output routines. This field is set to the request code by CPC when a request is encountered for this file. The request codes are defined in the file action request descriptions. The code and status bits have the following significance:

Bits 14-17 Record level number. On skip and write record requests, this subfield is set by CPC as part of the function code. On read requests, it is set by CIO as part of the status when an end-of-record is read. Initially the level subfield is set to zero when the FET is generated.

Bits 9-13 Status information upon request completion. Zero indicates normal completion. Non-zero indicates an abnormal condition, not necessarily an error; an OWNCODE routine, if present, will be executed. Status codes are described under OWNCODE routines. Initially, this subfield is set to zero when the FET is generated.

Bits 0-8 Used primarily to pass function codes to a peripheral processor. Function codes are even numbers (bit 0 has a zero value). When the request has been processed, bit 0 is set to one. When the FET is generated, bit 0 must be set to one to indicate that the file is not busy. Bit 1 specifies the mode of the file (0 = coded, 1 = binary). Bit 1 is not altered by CPC when a request is issued.

Bits 2-8 are used to pass function codes to a peripheral processor (file action requests).

Bits 3 and 4 may be altered by the peripheral processor routine when the request is completed if an end-of-record (10_2) or end-of-file was read (11_2).

The initial value of bits 2-17 is irrelevant.

Device Type (DT) (12 bits)

The device type field may be used in one of two ways:

The file may be assigned to a specific type of allocatable device when an OPEN function is given. Such an assignment is effective only if no prior reference to the file has been made.

The hardware type portion of the field will be set by SCOPE upon return from any other file action request, if the FET is more than five words long (the field length in word 2 of the FET is nonzero).

The device type field contains two 6-bit fields; the left 6 bits specify a hardware device and the right 6 bits declare a type within the device. When the code is 00, SCOPE selects the most easily accessible allocatable device. Other codes are shown below in octal:

Hardware	Type (record block size)	
01 6603 disk	00	alternate sector half-track both zones
	01	alternate sector half-track, inner zone only
	02	alternate sector half-track, outer zone only
	03	reserved for system
	†04	sequential sector, full-track; both zones
	†05	sequential sector, full-track; inner zone only
	†06	sequential sector, full-track; outer zone only
	07	reserved for system
	10	eight sector allocation
	11-77	reserved for system
02 6638 disk	00	alternate sector, half-track
	01-07	reserved for system
	10	eight sector allocation
	11-77	reserved for system
03 data cell	00-77	reserved for system
04 6603 with field option 10124 installed	xx	same assignments as unmodified 6603
05-06 reserved for system		

†SCOPE provides only the method for allocation; not the drivers.

Hardware	Type (record block size)	
07 3234/854 disk pack	00	normal system allocation
	†01	private user allocation
	02-77	reserved for system
10 reserved for system		
11 3637B/863 drum	00	[For SCOPE 3 only, standard allocation; 64 words per PRU, 5 PRU's per record block, 8 records per track]
	01-77	reserved for system
12 3637/B 865 drum	xx	same as 863 drum; 1 PRU = 3 sectors
13-77 reserved for system		
†40 1/2" magnetic tape	00	high density
	01	low density
	02	hyper density
	03-77	reserved for system
41 1" magnetic tape	00-77	reserved for system
42-49 reserved for system		
50 line printer	00-77	reserved for system
50-57 reserved for system		
60 card reader	00-77	reserved for system
61-67 reserved for system		
70 card punch	00-77	reserved for system
71 6612 display console	00-77	reserved for system
72-77 reserved for system		

Random Access (r) (1 bit)

The r field is set to one if the RFILEB or RFILEC macro is used; otherwise, r is zero. This field indicates a random access file and that record position information should be returned. If the file does not reside on a random access device, the r field is set to zero when the first reference is made to it.

† Device codes of 40 or greater may not be assigned from an FET, these devices require a REQUEST card or function. A private disk pack also requires a REQUEST card or function.

Release Bit (n) (1 bit)

A release bit set to one when a file action request is issued has the following effects for read and skip operations; it is meaningless on any other operation.

After a read or a skip forward operation, record blocks will be released.

After a skip backward operation, record blocks subsequent to the position of the file will be released.

User Processing (UP) (1 bit)

The UP bit is set to one when the calling program is to be notified when an end-of-reel condition is encountered during a 1/2" magnetic tape operation. If the field is set to zero, tape switching proceeds automatically without notification to the calling program; the function in process when end-of-reel is detected will be completed on a subsequent reel of tape.

When the UP field is set to one and an end-of-reel is detected on 1/2" magnetic tape, the end-of-reel status is set, 02_8 in bits 9-13 of the code and status field. This is the only point at which the end-of-reel status is returned.

All functions that do not transfer data from the circular buffer will be completed; those which transfer data may be re-issued as indicated by examination of the buffer pointers. CPC detects the end-of-reel status and transfers to the EOI OWNCODE routine, if present. At this juncture, the calling program may perform any action subject to the following restrictions:

CLOSER and OPEN, REEL functions must eventually be issued for the file in that order.

No file action requests other than CLOSER and OPEN, REEL may be issued for a labeled tape file.

The following decision table indicates action taken by the system and permitted in the CP program.

End-of-Reel Detected

Labeled Tape	Y	N	Y	N
Up bit Set	N	N	Y	Y
	1	2	3	4

1. Automatic switching of tapes with SCOPE labels. CP program is not aware of the operation. Control returns to the CP program after the request obstructed by the end-of-reel condition has been completed on the new tape.

2. Automatic switching of tapes without SCOPE labels; otherwise as in 1.

3. OWNCODE routine entered, if present. Only CLOSER and OPEN,REEL requests may be issued, in that order. These requests should be issued with recall to simplify processing. When the OPEN,REEL request is issued for an input tape, the system will deliver the file header label for the new reel to the circular buffer.

4. OWNCODE routine entered, if present. Any file action request is honored. Thus, the user may effectively put his own labels at the beginning or end of the tapes. Eventually a CLOSER function must be issued for the current reel of tape to terminate processing. Also, eventually an OPEN,REEL request must be issued for the subsequent reel of tape to restore the system to its proper status. If data is written prior to issuing the OPEN,REEL function for the new reel of tape the OPEN,REELNR option should be used so that this data is not overwritten.

   The OPEN function delivers an input label only if labels corresponding to the SCOPE Standard are declared on the REQUEST card or function.

   Routines which should be executed before and after the first volume file header label and the first volume trailer label may be written before and after the OPEN function or the file. Routines which should be executed before and after the last volume file trailer label may be written before and after the CLOSE function for an output tape. For an input tape such routines may be written in conjunction with the OWNCODE routine which processes the end of information status 01_8 in bits 9-13 of the code and status field.

Error Processing (EP) (1 bit)

The EP bit is set when the calling program wants to be notified of error conditions. If EP=0, the job will be aborted.

Disposition Code (dc) (12 bits)

The value in this field indicates the disposition to be made of the file when the job is terminated or the file is closed.

Value (Octal)	Disposition
0000	No special action required
0001	Checkpoint file
0002	Multi-file tape
0003-0007	Reserved for system
0010	File is punched on Hollerith cards when job is complete. If the disposition code of a PUNCH file is initially 0000, the file is assigned code 0010 before job completion
0011	Reserved for system
0012	File is punched on standard binary cards when job is complete. If the disposition code of a PUNCH B file is initially 0000, the file is assigned code 0012 before job completion.
0013	Reserved for system
0014	File is punched on binary cards when job is complete; sixteen 10-character words per card (80 columns)
0015-0017	Reserved for system
0020	File is sent to a filming device when job is complete. If the disposition code of a FILMPR file is initially 0000, the file is assigned code 0020 before job completion.
0021	Reserved for system
0022	File is sent to a filming device when job is complete. If the disposition code of a FILMPL file is initially 0000, the file is assigned code 0022 before job completion.
0023-0027	Reserved for system
0030	File is sent to a plotting device when job is complete. If the disposition code of a PLOT file is initially 0000, the file is assigned code 0030 before job completion.
0031-0037	Reserved for system

Value (Octal)	Disposition
0040	File is printed when job is complete. If the disposition code of an OUTPUT file is initially 0000, the file is assigned code 0040 before job completion.
0041-1777	Reserved for system
2000-7777	A one in the leftmost bit indicates that the file is being processed by EXPORT/IMPORT; a one in the next bit means that the file is being processed by RESPOND; therefore, codes 2000-7777 are not used in normal SCOPE processing.

Length of FET (ℓ) (6 bits)

The system FET length is determined as follows: FET first word address + 5 + ℓ = last word address + 1. The minimum FET length is five words (ℓ= 0). If the minimum FET is used, only the logical file name, code and status field, FIRST, IN, OUT, and LIMIT are relevant. No other field will be set or checked by SCOPE. A length of six words (ℓ= 1) is used if a working storage area is needed for blocking/deblocking. A length of eight words (ℓ= 3) is used if the r bit is set, indicating an indexed file. Length is nine words (ℓ= 4), if OWNCODE routines are declared. The maximum system FET length is 13 words (ℓ = 8). The maximum size is used if a labeled tape file is declared.

FNT Pointer (12 bits)

The FNT pointer is set by SCOPE, upon return from a file action request, to the location of the file in the FNT/FST. The pointer is placed in the FET to minimize table search time and does not affect the program. The pointer will not be set if a minimum FET is used.

Physical Record Unit Size (PRU) (12 bits)

The physical record unit size of the device to which the file is assigned is returned in this field at OPEN time. It is given as the number of central memory words. The PRU size is used by CPC to determine when to issue a physical read or write. PRU size will not be returned if a minimum FET is used.

Record Block Size (15 bits)

If the file resides on an allocatable device, the size of the device record block is returned in this field at OPEN time. It is given as the number of physical record units in a record block. If the number of PRU's is not defined or is variable, the field is set to zero. Record block size is not returned if a minimum FET is used.

FIRST, IN, OUT, LIMIT

Data is transmitted in physical record units, the size of which is determined by the hardware device. For example, the 6603 disk has an inherent PRU size of 64 CM words; binary mode magnetic tape files are assigned a PRU size of 512 words.

For each file, the user must provide one buffer, which can be any length greater than a PRU size. This is called a circular buffer because it is filled and emptied as if it were a cylindrical surface in which the highest addressed location is immediately followed by the lowest.

The FET fields FIRST, IN, OUT and LIMIT control movement of data to and from the circular buffer.

FIRST and LIMIT never vary; they permanently indicate buffer limits to the user and to SCOPE. During reading, SCOPE varies IN as it fills the buffer, and the user varies OUT as he removes data from the buffer. During writing, the user varies IN as he fills the buffer with data, and the system varies OUT as it removes data from the buffer and writes it out — the program that puts data into the buffer varies IN, and the program that takes it out varies OUT. The user cannot vary IN or OUT automatically except when using READIN and WRITOUT functions; he must do this within the program by inserting a new value into lfn + 2 (IN) or lfn + 3 (OUT). For the user's as well as for the system's convenience, the words containing IN and OUT contain no other items; this eliminates the need for masking operation.

The system dynamically checks the values of IN and OUT during data transfers, making continuous read or write possible.

If IN = OUT, the buffer is empty; this is the initial condition. If IN > OUT, the area from OUT to IN - 1 contains available data. If OUT > IN, the area from OUT to LIMIT - 1 contains the first part of the available data, and the area from FIRST to IN - 1 contains the balance.

To begin buffering, a READ function may be issued. SCOPE will put one or more PRU's of data into the buffer beginning at IN, resetting IN to one more than the address of the last word filled after each PRU is read. Data may be processed from the buffer beginning with the word at OUT, and going as far as desirable, but not beyond IN - 1. The user must then set OUT to one more than the address of the last word taken from the buffer. He sets OUT = IN to indicate that the buffer is empty.

When a READ request is issued, if the buffer is dormant (no physical read occurring), CPC determines how much free space the buffer contains. If OUT > IN, OUT - IN words are free. If IN > OUT, (LIMIT - IN) + (OUT - FIRST) words are free. The system subtracts 1 from the number of free words, because it must never fill the last word; this would result in IN = OUT, which would falsely indicate an empty buffer. If the number of free words, minus 1, is less than the PRU size, CPC does not issue a physical read request; control is returned normally.

The example below illustrates the way IN and OUT pointers are used. Speed of operation is not considered and simultaneous processing and physical I/O is not attempted.

The initial buffer pointer position is:

FIRST = BCBUF

IN = BCBUF

OUT = BCBUF

LIMIT = BCBUF+500

The user issues a READ with recall request.

Neglecting the possibilities of an end-of-record or end-of-file, the system reads as many PRU's as possible (if PRU size is 64 words, 7 x 64 = 448 words) and leaves the pointers:

FIRST = BCBUF

IN = BCBUF+448

OUT = BCBUF

LIMIT = BCBUF+500

The user is processing items of 110 words. He takes four items from the buffer, leaving the pointers:

FIRST = BCBUF

IN = BCBUF+448

OUT = BCBUF+440

LIMIT = BCBUF+500

The user issues another READ request, since he knows the buffer does not contain a complete item. The system is aware that IN > OUT, so that the vacant space amounts to LIMIT - IN + OUT - FIRST = 492 words; since it must not fill the last word, it must read fewer than 492 words.

The nearest lower multiple of 64 is 7 x 64 = 448, so it reads 52 words into IN through LIMIT - 1, and then 396 more words into FIRST through FIRST + 395. It then resets IN so that the pointers look like:

FIRST = BCBUF

IN = BCBUF+396

OUT = BCBUF+440

LIMIT = BCBUF+500

The system has just used the circular feature of the buffer; now the user must do so. The next time he wants an item, he takes the first 60 words from OUT through LIMIT - 1, and the remaining 50 from FIRST through FIRST + 49. Then he resets OUT, making the pointers:

FIRST = BCBUF

IN = BCBUF+396

OUT = BCBUF+50

LIMIT = BCBUF+500

On input, this can continue indefinitely, with OUT following IN, around the buffer. The system stops on encountering an end-of-record, and sets the code and status bits accordingly. The system may, or may not, have read data before the end-of-record, so it is up to the user to examine the pointers before taking end-of-record action.

In writing, the process is similar, but the roles are reversed. The user puts information into the buffer and resets IN; and when he calls the system, it removes information from the buffer and resets OUT. For writing, the system removes data in physical record units and empties the buffer if possible. The user must be careful not to overfill the buffer — IN must not become equal to OUT. During the process of emptying the buffer, SCOPE resets OUT after each PRU has been written and checked for errors.

Working Storage Area

The two fields in word 6 of the FET specify the first word address (fwa) and last word address + 1 (lwa + 1) of a working storage area within the program field length. Logical records may be deblocked into or blocked from this area into the circular buffer. (See READIN and WRITOUT.)

File Indexing Fields

The file indexing fields (record request/return information, record number index length and index address) are used for communication between the indexing functions (READIN and WRITOUT) and the peripheral processor input/output routines. Index address and index length fields are declared when the FET is generated; they specify the fwa of the index buffer within the program field length and length of the index buffer. The record request/return information field is set to zero when the FET is generated; both the indexing functions and the peripheral processor input/output routines set the field during random file processing.

For an indexing method other than that offered by SCOPE, the following information is pertinent. When writing of a new logical record begins, if the random access bit and the record request/return information field are both nonzero, the latter is assumed to contain the address of a location within an index. The PP routine inserts into that location (in bits 0-24), the record block and PRU address of the logical record. To read the record again, the random access bit should be set to nonzero and the record block and PRU address should be entered in the FET in the record request/return information field.

OWNCODE Routines

Addresses of user-supplied routines may be given in the FET. These routines are executed by CPC as indicated below. A zero value indicates that no routine is supplied.

An OWNCODE routine should be set up like a closed subroutine with execution beginning in the second word of the routine. CPC calls an OWNCODE routine by copying the exit word of CPC into the first word of the OWNCODE routine, putting the contents of the first word of the FET into X1, and branching to the second word of the OWNCODE routine.

Termination of an OWNCODE routine by a branch to its first word causes a branch to the point in the program to which CPC would have returned if the OWNCODE routine had not been called. The A, B, and X registers may have been changed by CPC before control gets back to the routine that called CPC. Therefore, an OWNCODE routine which is terminated by a branch to its first word should not rely on passing information to the main program in the registers.

EOI Address Field

CPC enters the end-of-information (EOI) routine under the following circumstances:

Bits 9-13 of Code and Status:

01_8 End-of-information encountered after forward read

02_8 End-of-reel reached during magnetic tape forward operation

Just before entering an end-of-information OWNCODE routine, CPC zeros bits 9 and 10 of the first word of the FET. However, as the routine is entered, X1 still contains the first word of the FET as it appeared before those two bits were zeroed.

Error Address Field

This field specifies an address to receive control if an error condition occurs after a file action request. The FET code and status field will reflect the error condition. If processing can continue, the error routine should exit through its entry point; otherwise, an ABORT request may be issued.

If the error address field is zero, the run continues normally. The FET code and status bits reflect the error condition upon normal return to the program.

Bits 9-13 of Code and Status:

- 04_8 Unrecoverable parity error on last operation
- 10_8 Device capacity exceeded on write, or tape physical record size greater than circular buffer on read, or unrecoverable parity error on last write (lost data)
- 20_8 OPEN function redundant
- 21_8 CLOSE function redundant
- 22_8 Illegal function
- 23_8 Index full
- 24_8 FNT full
- 25_8 An attempt was made to read or write record number n of a random file, but the index of the file is full.
- 26_8 An attempt was made to read a named record from a random file, but the name does not appear in the index.
- 27_8 An attempt was made to write a named record on a random file, but the name does not appear in the index, and there is no room to add a new name.

In the event that both EOI and error routine execution are needed, the error routine is executed.

Just before entering an error OWNCODE routine, CPC zeros bits 11-13 of the first word of the FET. However, as the routine is entered, X1 contains the first word of the FET as it appeared before those bits were zeroed.

3.2 LABELED TAPE FILES

File Label Name (17 characters)

This field contains from 1 to 17 alphanumeric display-coded characters, starting with a letter, left justified; if less than 17 characters are declared, the unused characters are binary zero-filled. The file label name appears in field 3 of the file label header (appendix C). It ensures that the correct file is being referenced. Checking is accomplished when the file is opened providing a labeled tape was specified on a REQUEST card or by a REQUEST function.

Edition Number (2 characters)

The two-character edition number of the file is stored in field 8 of the file header label (appendix C) of an output tape and verified against this field for an input tape. If this parameter is omitted, 01 is written in the label of an output tape and in the FET, and no checking is done for an input tape.

Reel Number (4 characters)

The reel number of the file is stored in field 5 of the file header label (appendix C) of an output tape and verified against this field for an input tape. If this parameter is omitted, 0001 is written in the label of an output tape and in the FET. For each reel, this field is increased by one at the conclusion of the processing of the file trailer label and the tape mark is written on the tape. When the file is closed, this field is set to 0001.

Creation Date (5 characters)

The first two characters specify the year and the remaining three the Julian day within the year. This data is stored in field 10 of the file header label for an output tape and verified against this field for an input tape. If this field is omitted, today's date as stored in the SCOPE system is written in this format in the label of an output tape and in the FET. For an input tape, this field is read from the label and stored in the FET.

Retention Cycle (3 characters)

Three digits specify the number of days a tape is to be protected from accidental destruction. This field is added to the value of the creation date field to obtain the expiration date which is written on an output tape label or verified against an input tape label. Field 12 of the file header label is used (appendix C).

3.2.1 MULTI-FILE TAPES

Multi-file Name (6 characters)

This field contains from one to six alphanumeric display-coded characters, starting with a letter, left justified; unused characters are binary zero-filled. This field identifies all files of a multi-file volume (appendix C) and must be the same for all files on a particular volume. If this field is omitted, only a single file may be generated or read in a volume set. A multi-file tape must be so declared by stating its disposition on a REQUEST card/function.

In such a REQUEST, the multi-file name is given as the lfn, for example:

REQUEST, mfn,dt,MF,x.

Only one file on a multi-file tape may be open at any given time:

Position Number (3 digits)

This field is ignored unless a multi-file name is specified. If this parameter is absent for a multi-file output tape, the file is assigned in the sequence in which it is written and this position number will be returned to the FET. Overwriting a file on a multi-file volume set destroys the remaining files it contains. If this parameter is absent for a multi-file input tape, the value determined in the search for the file will be stored in the FET.

3.3 FET CREATION MACROS

System macros in the COMPASS language facilitate generation of the system FET, as follows:

Coded File - Sequential

lfn FILEC fwa, f, (WSA = $addr_w$, 1_w), (OWN = eoi, err), LBL, DTY = dt, DSC = dc, UPR, EPR

Binary File - Sequential

lfn FILEB fwa, f, (WSA = $addr_w$, 1_w), (OWN = eoi, err), LBL, DTY = dt, DSC = dc, UPR, EPR

Coded File - Random

lfn RFILEC fwa, f, (WSA = $addr_w$, 1_w), (IND = $addr_i$, 1_i), (OWN = eoi, err), LBL, DTY = dt, DSC = dc, UPR, EPR

Binary File - Random

lfn RFILEB fwa, f, (WSA = $addr_w$, 1_w), (IND = $addr_i$, 1_i), (OWN = eoi, err), LBL, DTY = dt, DSC = dc, UPR, EPR

The last seven subfields (WSA, DTY, DSC, UPR, IND, OWN, LBL) are order-independent; within the subfield, order is fixed. Upper case characters designate actual subfield content, lower case characters indicate parameters to be supplied by the user. All parameters except lfn, fwa, and f are optional.

	lfn	file name
	fwa	substituted in FIRST, IN, and OUT
	f	fwa + f substituted in LIMIT
WSA	Working storage area parameters	
	$addr_w$	first word address of working storage area
	l_w	$addr_w + l_w$ = last word address + 1 of working storage area
IND	Index buffer parameters	
	$addr_i$	first word address of index buffer
	l_i	length of index buffer
OWN	OWNCODE routines	
	eoi	end-of-information address
	error	error address
DTY	Device type parameter	
	dt	a 12-bit code described in FET field descriptions
DSC	Disposition code parameter	
	dc	a 12-bit code described in FET field descriptions
UPR	User desires processing at end-of-reel	
LBL	Label information will follow. The LABEL macro which provides label information, must be written immediately following the FILE macro to which it pertains. It has the form:	
		lfn LABEL lfn, ed, ret, create, reel, mfn, pos
EPR	User desires to handle error conditions	

Examples:

To create a minimum FET for the standard INPUT file:

```
INPUT FILEC BUFFER, LBUFFER
```

To create an FET for a binary random file:

```
FILEABC RFILEB BUFFER, LBUFFER, (IND = INDEX, LINDEX)
```

To create an FET for a labeled tape file with user processing at end-of-reel condition. OWNCODE routine is supplied:

```
TAPE1 FILEB BUFA, LBUFA, LBL, UPR, (OWN=PROCEOR)

TAPE1 LABEL SORTINPUTTAPE, 32, 90
```

To create an FET for a list file. OWNCODE routines are supplied and the working storage area is used:

```
PRINT FILEC BUFB, LBUFB, (WSA=LINE, 14), DSC=40B,
                                (OWN=ENDING, ERRORS)
```

To create an FET for a file to be written on a 6603 Disk, using only inner zones:

```
FILE1 FILEB BUFD, LBUFD, DTY=0101B
```

3.4 CENTRAL PROGRAM CONTROL SUBROUTINE (CPC)

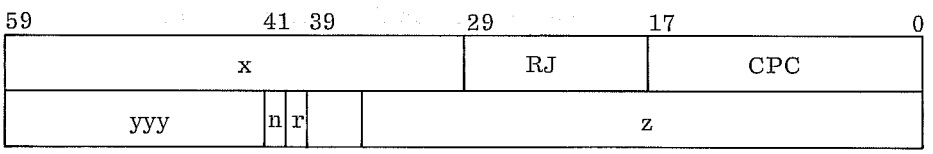
The central program control subroutine (CPC) provides the linkage between user programs and the SCOPE system. All file action requests and system action requests are processed by the CPC library subroutine which is loaded with the user program within the field length of the job. The program communicates with CPC through macro requests and the file environment table (FET). Communication with SCOPE is handled by CPC setting and checking RA + 1.

CPC may also cause the execution of one or more user subroutines for which addresses are specified in the FET. Such a subroutine is entered at the address given in the FET + 1. The exit from the CPC is stored at the OWNCODE routine given in the FET; (X1) = the first word of the FET.

A normal exit from CPC returns control to the object program at the point following the macro request. A normal exit is made if the request is honored and no error conditions occur. X1 contains word 1 of the FET upon exit if the status is other than request completed. CPC does not save registers.

3.4.1 CALLING SEQUENCE

Format of the calling sequence to the central program control subroutine:



RJ Return jump instruction

CPC Entry point to the CPC subroutine

n = 0 File action request

yyy Display-coded name of the PP program to be inserted by CPC in RA + 1 or one of the following:

000001 if only a file RECALL is wanted
000007 for CLOSE or EVICT
000004 for OPEN
000002 for READ or WRITE (without end-of-record)
000003 for other functions

x SA1 <base address of FET>

z Request code

n = 1 System action request

yyy Display-coded name of the called PP program

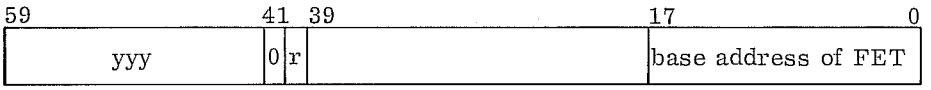
x not relevant

z parameters as required

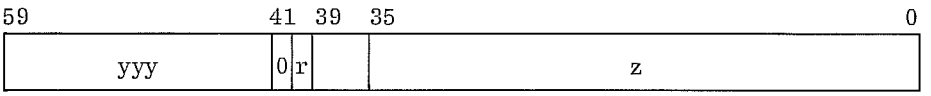
r = 1 Issue request and enter RECALL

r = 0 Issue request and return control to the program

A file action request to the SCOPE monitor is formatted by CPC in RA + 1 as follows:



A system action request to the SCOPE monitor is formatted in RA + 1 as follows:



z appears in the buffer code and status field of the FET.

Bits not specified in the calling sequence are reserved for future system use.

3.5 SYSTEM COMMUNICATION MACROS

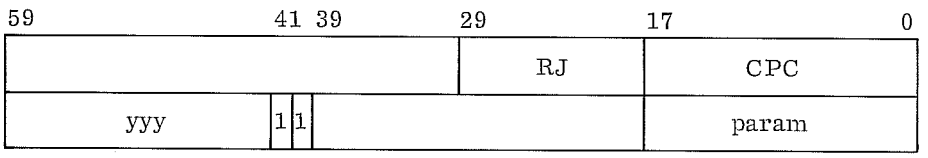
In the following descriptions the system macro is followed by the macro expansions.

3.5.1 FILE ACTION REQUESTS

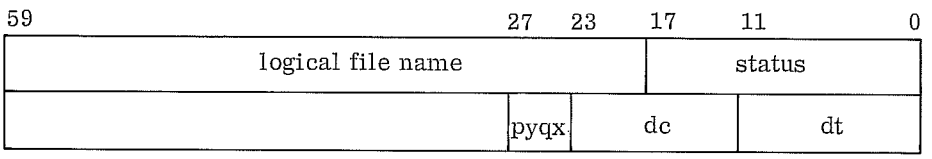
File action requests result in a return jump to the central program control subroutine. Subsequent actions depend on the state of the file. An OWNCODE routine may be executed and/or a request to SCOPE may be posted. In either case, control returns to the calling program after SCOPE accepts the request if the recall bit, r, is equal to zero, or after SCOPE completes the request if r is equal to one. In macros specifying the optional final parameter recall, r is set to one when recall is present.

READ lfn, recall

REQUEST REQUEST addr



With the REQUEST function, a CP program can assign equipment during execution without requiring a REQUEST control card. param is the first word address of a two-word list of parameters, as shown below.



The values for dc and dt are given in section 3.1.1 (Basic File Environment Table). The 4-bit parameter pyqx applies only when dt specifies 1/2-inch magnetic tape. pyqx is interpreted as follows:

p = 1 External tape

p = 0 SCOPE tape

y = 1 2 tapes

y = 0 1 tape

q = 1 SCOPE system labels for this file

q = 0 Unlabeled

x = 1 Existing file

x = 0 New file

If the lfn designated in REQUEST parameters is already associated with a file, the REQUEST function is ignored and control is returned normally. Therefore, the REQUEST function should be issued prior to any reference to the logical file name, since a reference to a nonexistent lfn will cause the name to be associated with an empty file.

The status field should contain zero when the REQUEST function is issued. Bit 0 is set to one when the function is completed. In addition, the following values may be returned in bits 9-13:

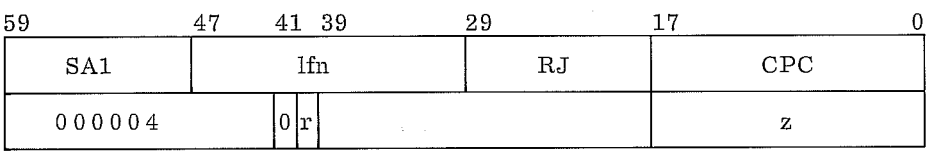
22_8 Illegal function; REQUEST function was issued without the recall bit.

24_8 FNT full

25_8 No equipment logically available. All equipment of the requested type is assigned; at least one unit is assigned to some other control point.

26_8 No equipment available. Either the requested equipment does not exist in the configuration or all equipment of this type is already assigned to this control point.

OPEN OPEN lfn, x, RECALL



The OPEN function readies the file for processing. The x parameter specifies the operation to be performed: READ, WRITE, READNR, WRITENR, ALTER, REEL or REELNR. The OPEN function causes information to be returned to the user via the FET. (See File Environment Table, p 3-1.) Unless the no rewind option is specified (READNR, WRITENR, REELNR), the file is rewound and the buffer pointers (IN and OUT) are set equal to FIRST.

This function is optional except in the following cases:

Indexed file: The OPEN function is required to read the index into the index buffer.

File recorded on 1/2-inch magnetic tape with standard SCOPE system labels: The OPEN function is required to process the label and position the tape. The label is delivered to the circular buffer for an input file.

Device type other than 0000 is to be assigned to the file.

x = READ (Z = 140)

If the file has a system label, it is read into the circular buffer and positioned at the first data record. If the file does not exist, an end-of-information status is returned. The file may be read only until it is closed.

x = WRITE (Z = 144)

If the file has a system label, it is written using the parameters in the FET. The file remains positioned after the tape mark following the tape label. The file may be written only until it is closed.

x = ALTER (Z = 160)

Checking insures that the file resides on a random access device. If it does not, the random access bit (r) in the FET is set to zero.

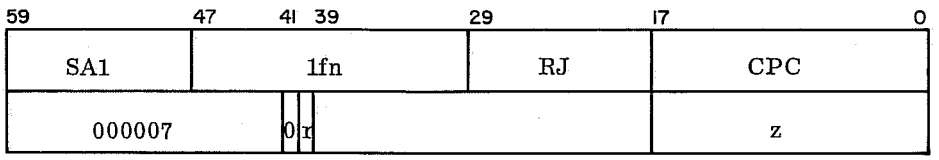
A file is normally rewound when the OPEN function is issued. If it is not to be rewound, options of x may be issued:

x = READNR (Z = 100)	Open as in READ; do not rewind.
x = ALTERNR (Z = 120)	Open as in READNR; do not rewind. Security code is SET OPEN.
x = WRITENR (Z = 104)	Open as in WRITE; do not rewind.

Swapping for multi-reel, labeled tapes may be controlled by setting the UP bit in the FET and using the following option:

x = REEL (Z = 340)	Tape is rewound; for labeled tape beginning label is processed. Reel is initialized.
x = REELNR (Z = 300)	Tape is not rewound. Reel is initialized.

CLOSE



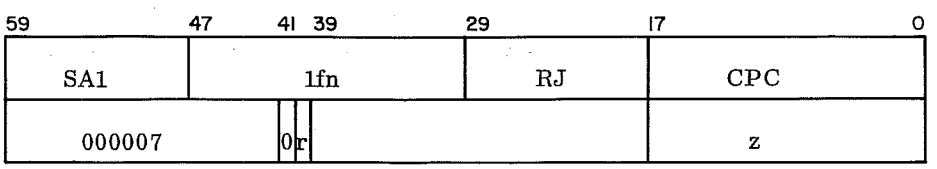
The CLOSE function sets the file to closed status. The x parameter specifies additional action to be performed. An end-of-information mark is written on an output file. If the file resides on 1/2-inch magnetic tape and standard SCOPE system labels are used, ending label procedures are performed.

Unless the no rewind option (NR) is used, the file is rewound and the buffer pointers (IN and OUT) are set equal to FIRST.

x is absent (Z = 150)	File is set to beginning-of-information or beginning of current reel.
x = NR (Z = 130)	File is not rewound.
x = UNLOAD (Z = 170)	Job termination procedures for the file are executed using disposition code in FET. Local files are dropped. Common files are dissociated from job. If present, index is written as suffix to output file. A tape file is rewound and unloaded.

For a local file on a private disk pack, device label, RBR record, and RBT chain are written on the pack. The unit assignment is dropped from control point. The disk pack remains in a private mode until unloaded by operator and a new pack is added.

CLOSER CLOSER lfn,x,recall

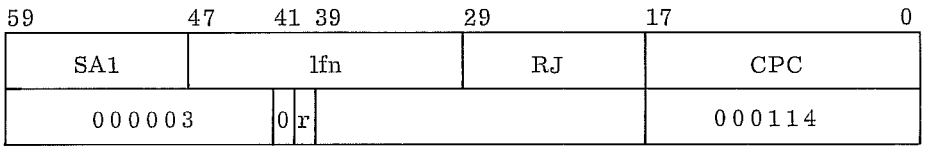


The CLOSER function is used for files on 1/2-inch magnetic tape to terminate processing prematurely on a given reel of a multi-reel tape or to control labeling. If standard SCOPE system labels have been used, ending label procedures are performed for the reel.

The x parameter specifies file position after CLOSER action.

x is absent (Z = 350)	Current reel is rewound.
x = NR (Z =330)	Reel is not rewound.
x = UNLOAD (Z = 370)	Tape is rewound and unloaded.

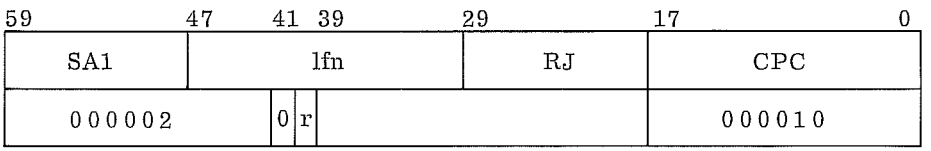
EVICT EVICT lfn, recall



EVICT releases to the system all space occupied by a file on disk and makes it available for use by either the releasing program or other programs. The logical file name is retained.

3.5.2 DATA FUNCTIONS

READ READ lfn, recall



This function reads information into the circular buffer if the specified file is open. If there is room in the circular buffer for at least one physical record unit, reading is initiated and continues until:

Buffer is full

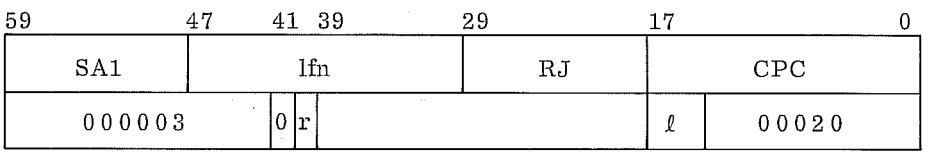
End-of-record or end-of-file is encountered

End-of-information is encountered

The mode is determined by bit 1 in the first word of the FET.

If the end-of-record (bit 4) is set upon entry to CPC, no operation is performed.

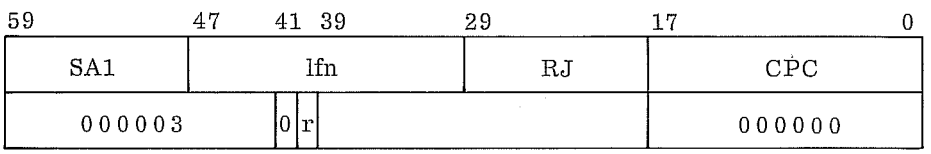
READSKP READSKP lfn,ℓ, recall



READSKP functions as a READ, except that if the buffer is filled prior to an end-of-record, information is bypassed until the end-of-record is read. The file is positioned ready to read the next logical record. If a level parameter (ℓ) is specified, information is skipped until an end-of-record with a level number occurs that is greater than or equal to that specified.

Executing a READSKP sets the end-of-record (bit 4) to 1, since an end-of-record is encountered. If the next operation on the file is READ, the EOR bit must first be zeroed by the calling program.

RPHR RPHR lfn, recall



The RPHR function causes any information already in the buffer to be discarded by setting the OUT pointer equal to the IN pointer; then the next physical record on the input device, which must be magnetic tape, will be read into the buffer. The mode is determined by bit 1 in the first word of the FET. Only conversion from external to internal BCD is performed. If the data read does not exactly fill an integral number of CM words, the last word is filled with zeros.

If the physical record is too big for the circular buffer, or if it is longer than 512_{10} words, as many words as possible are put into the buffer, and the error response is 10_8 for device capacity exceeded.

An end-of-file response is given if the physical record is an end-of-file mark, and the buffer will be empty. An end-of-record response is never given.

A RPHR function is ignored for any device other than 1/2-inch magnetic tape, and an illegal function status (22_8) is returned.

READIN

With this function reading depends on file mode and the presence or absence of a file index, a working storage area, and the x parameter. In the following paragraphs n represents the number of words in the working storage area.

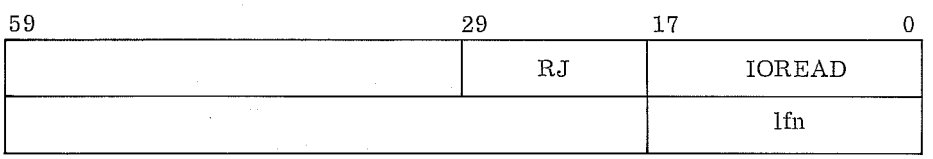
READIN takes the next n words from the circular buffer of file lfn and stores them in the working storage area; a READ request is issued if the buffer is empty. If the file is binary mode, READIN attempts to fill the working storage area until end-of-record or end-of-information is encountered. For a coded file, information is moved to the working storage area until a zero byte (end-of-line) is encountered or until the working storage area is full. When a zero byte is encountered, two blanks are substituted and the remainder of the working storage area is filled with blanks. If a zero byte is not encountered before the working storage area is full, the remainder of the line is skipped and a subsequent READIN request reads the next line.

The status of the request is returned in X1 as follows:

+0	Requested number of words was read and the function completed normally.
positive nonzero	Fewer than n words remained in the logical record when the request was issued. When control is returned to the user program, X1 contains the last address + 1 of the data transferred to working storage or first word address if no data was transferred. For coded files, this is always the first word address.
negative nonzero	If end-of-information is encountered, X1 contains a negative number. No information is transferred into the working storage area.

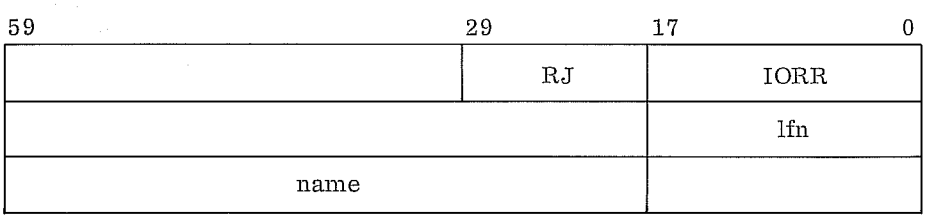
If a working storage area is not specified, a READIN request has no effect and no error indication is given unless it addresses a file with a name or number index. In that case, the effect of the request will be to terminate any previous action on the file, locate the specified logical record, and set up the pointers so that the next request to continue reading the current record on that file will begin with the first word of the specified record.

x is absent: READIN lfn



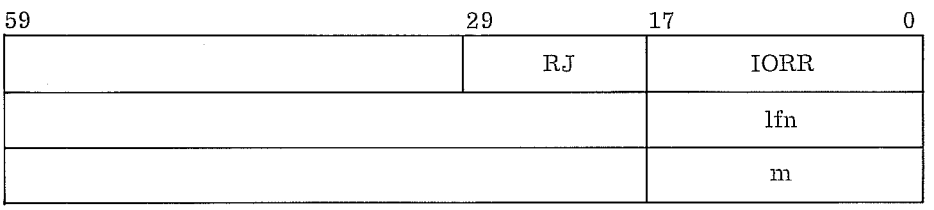
This form of the READIN request transfers data to the working storage area.

x is of the form /name/: READIN lfn,/name/



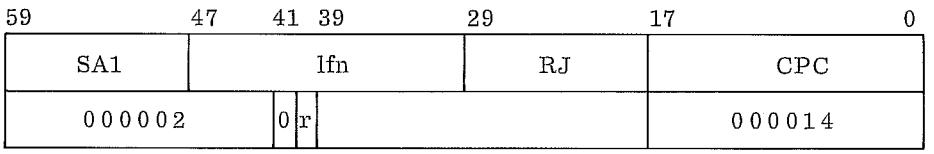
This form of the READIN request causes logical record /name/ on the file named lfn to be read into the circular buffer. n words are transferred to the working storage as described above. The file must have a name index.

x is of the form m where m is a logical record number: READIN lfn, m



This form of the READIN request causes logical record number m of the file named lfn to be read into the circular buffer. n words are transferred to the working storage area as described above. The file must be indexed by name or number. If m is zero, the next record is read, where next record is defined to be the first logical record of the file if this is the first request, or the last logical record read + 1 if it is the second or subsequent READIN.

WRITE WRITE lfn, recall



This function writes information from the circular buffer if there is sufficient information to fill one or more physical record units. Writing continues until the buffer is empty, or there is not enough data in the buffer to fill a PRU.

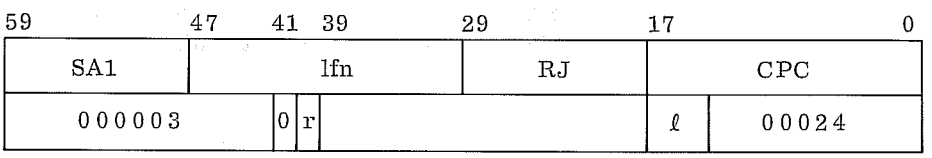
The mode is determined by bit 1 in the first word of the FET. If the device type is MT and the tape is at load point, a check is made before writing to insure that the tape does not contain a SCOPE system label with an unexpired expiration date. If it does, a message to the operator at the system console indicates the condition. The operator then has three options:

Request another check either on the same type or a new one.

Drop the job.

Indicate that checking is to be ignored, i.e., the mounted tape may be written upon.

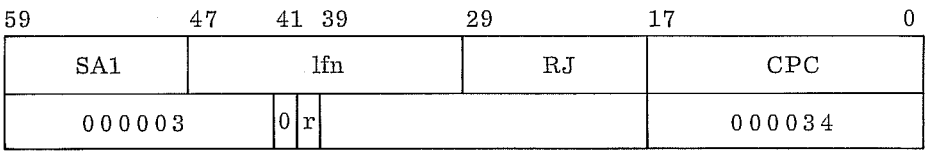
WRITER WRITER lfn, ℓ, recall



This function is processed the same as WRITE, with the following exceptions. Data in the circular buffer is written out and terminated by a short or zero-length PRU to indicate end-of-record. If no information is in the buffer, a zero-length PRU is written.

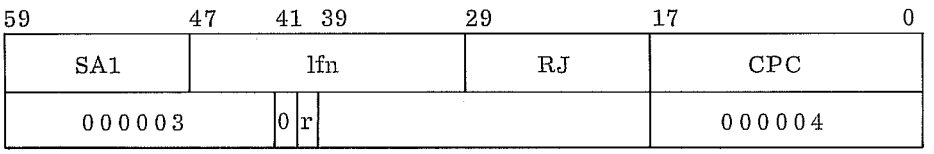
If the level parameter (*l*) is present, the short or zero-length PRU will reflect the level number. In the absence of the level parameter, the *l* field is set to 0 and level zero is assumed.

WRITEF WRITEF lfn, recall



The WRITEF function causes a logical end-of-file mark to be written. A logical end-of-file is written as a zero-length physical record unit of level 17_8. When a WRITEF function is issued, any data present in the buffer is first written and terminated with a level zero end-of-record.

WPHR WPHR lfn, recall



The WPHR function causes the information in the circular buffer to be written as a single physical record on the output device, which must be magnetic tape. The mode is determined by bit 1 in the first word of the FET.

If the buffer contains less than 512 words the IN and OUT pointers in the FET are set equal to indicate an empty buffer when writing is completed. Only conversion from external to internal BCD is performed.

If the buffer contains more than 512_{10} words when the request is issued, the first 512 words are written out and the IN and OUT pointers are set to show that words remain in the buffer. The device-capacity-exceeded status (10_8) is returned.

A WPHR function issued for any device other than 1/2-inch magnetic tape is ignored and an illegal function status (22_8) is returned. Labels are checked as for WRITE.

WRITOUT

With this function, writing depends on file mode and the presence or absence of a file index, a working storage area, and the x parameter. In the following paragraphs, n represents the number of words in the working storage area.

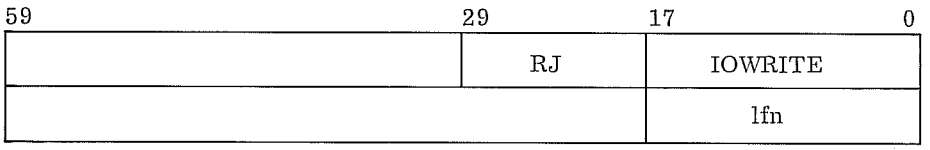
WRITOUT takes n words from the working storage area and transfers them to the circular buffer, thereby adding them to the logical record currently being constructed. If there is no current record, they become the first words of a new logical record. If the file is indexed, however, such a request is rejected, because the system has no way of knowing which record of the file is being addressed. A WRITE request is issued automatically when the buffer is full.

If the file is in binary mode, the entire working storage area is transferred to the circular buffer. If the file is in coded mode, trailing blanks are removed and a zero byte (end-of-line) is inserted as the data is transferred to the buffer.

The WRITER function may be requested to terminate writing of a record. If the file is indexed, and no record is currently being written, a WRITER request is rejected.

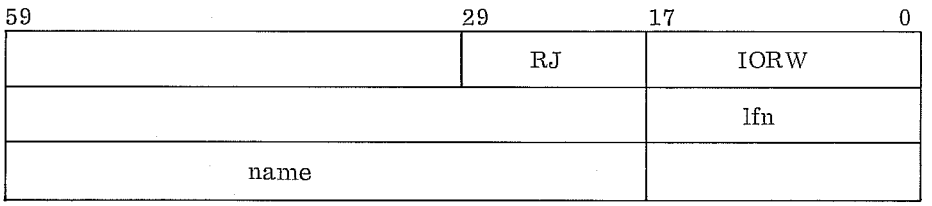
If a working storage area is not specified, execution of a WRITOUT request has no effect and gives no error indication, unless it addresses a file with a name or number index. In that case, the effect of the request will be to terminate any previous action on the file, locate the specified logical record, and set up the pointers so that the next request to continue writing the current record on that file will begin with the first word of the specified record.

x is absent: WRITOUT lfn



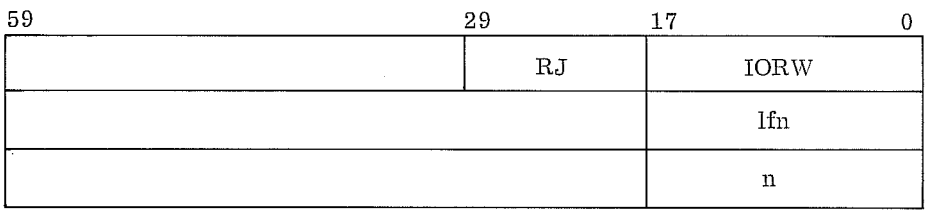
This form of the WRITOUT request transfers data from the working storage area.

x is of the form /name/: WRITOUT lfn,/name/



This form of the WRITOUT request begins writing the /name/ record on the file named lfn, using the words in the working storage area as the first n words of the record. The file must have a name index; i.e., on the first occasion that anything was written on the file, the record was addressed by name.

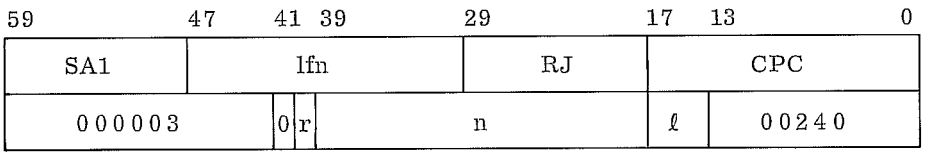
x is of the form m, where m is a logical record number: WRITOUT lfn,m



This form of the WRITOUT request begins writing logical record number m on the file named lfn using the words in the working storage area as the first words of the record. The file must be indexed, either by name or by number. If m = 0, the request will address the record with a number one higher than that of the record most recently addressed, or record number 1 if the file has not been addressed. The first record of an indexed file is number 1; there is no record number 0.

3.5.3
POSITION FUNCTIONS

SKIPF SKIPF lfn, n, ℓ, recall



SKIPF causes one or more logical records to be bypassed in a forward direction. The request may be initiated at any point in a logical record. The number of logical records or record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. The maximum value of n is 777777_8; if n = 777777_8, the file is positioned at the end-of-information.

If the level parameter (ℓ) appears, logical records are skipped until an end-of-record with a level number greater than or equal to the requested level is reached; the file is positioned immediately following the end-of-record mark. This positioning process will be performed n times. For example, a SKIPF lfn, 2, 1 issued while positioned at page 6 would cause repositioning to the beginning of chapter 5 (Level Numbers, 1-3).

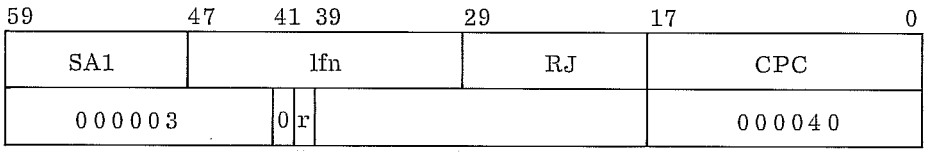
If the level parameter is absent, the ℓ field is set to zero and the file is positioned forward n logical records (or partial logical records if the SKIPF is issued in the middle of a logical record).

If the end-of-information is encountered before an end-of-record with the specified level is found, the end-of-information status bit will be set. Parity errors encountered during a SKIPF operation are ignored.

Backspace Functions

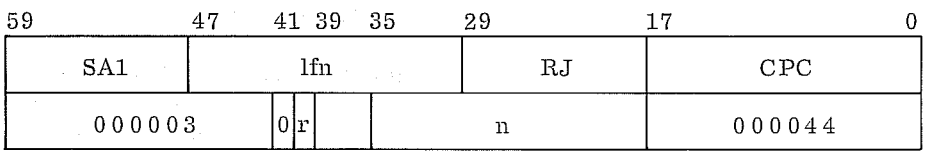
Backspace functions will not go beyond the beginning of the current reel of magnetic tape.

BKSP BKSP lfn, recall



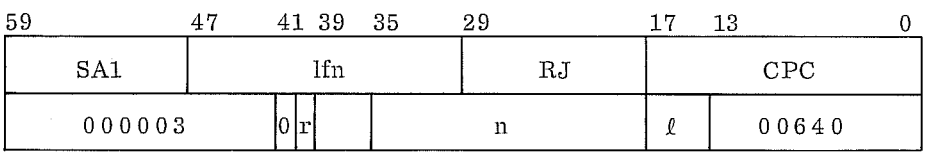
The BKSP function causes one logical record to be bypassed in a reverse direction. The request may be issued at any point in a logical record. This function is a subset of SKIPB; it is included for compatibility with previous systems.

BKSPRU BKSPRU lfn, n, recall



One or more PRU's are bypassed in a reverse direction. The request may be issued at any point in a logical record. If n appears, n PRU's are bypassed. If n does not appear one PRU is bypassed. Parity errors encountered during a BKSPRU are ignored.

SKIPB SKIPB lfn, n, ℓ, recall



SKIPB causes one or more logical records to be bypassed in a reverse direction. The request may be initiated at any point in a logical record. The number of logical records or logical record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. The maximum value of n is 777777_8; if n = 777777_8, the file is rewound.

If the level parameter is used, logical records are read backwards until a short PRU of the specified level has been read. A forward read is issued, leaving the file positioned after this short PRU. If the file is positioned initially between logical records, the level number immediately preceding the current position is ignored in searching for a logical record of the specified level. This positioning process is performed n times.

Consecutive logical records within a file may be organized into a group by using level numbers. The file will be composed of one or more groups of logical records. This may be done by choosing a minimum level number $\ell \neq 0$ and assigning a level number greater than or equal to ℓ to the last logical record of each group, and a level number less than ℓ to all other logical records.

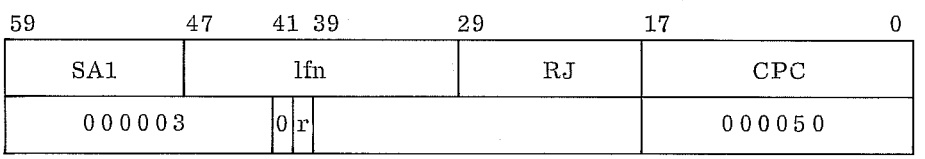
Then SKIPB lfn,,ℓ will skip the file backward to the beginning of the logical record group which immediately follows a logical record of level ℓ. In the example of level numbers shown in section 1.3.2, the minimum level number was 1; a SKIPB lfn,2,1 issued while positioned at page 14 would cause repositioning to the beginning of chapter 4.

If the level parameter is absent, the ℓ field is set to zero and the file is positioned backward n logical records (or partial logical records if the SKIPB is issued in the middle of a logical record).

If the beginning-of-information is encountered before the requested level number is found, the beginning-of-information status bit is set. Parity errors encountered during a SKIPB operation are ignored.

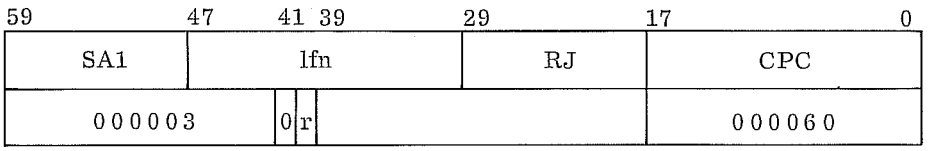
REWIND

REWIND lfn, recall



REWIND positions the file at the beginning of the first data record or at the beginning of the current reel. A REWIND function on a file already rewound has no effect. A REWIND function issued for a device not capable of being repositioned causes an illegal function status (22_8) to be returned.

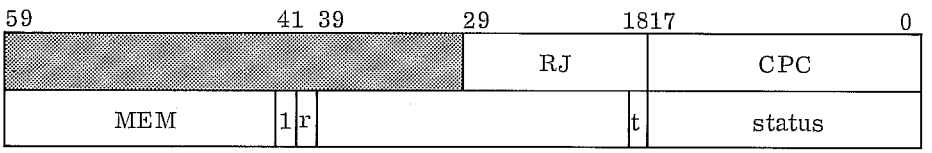
UNLOAD UNLOAD lfn, recall



UNLOAD functions the same as REWIND. If the file resides on magnetic tape, the tape is unloaded.

3.5.4 SYSTEM ACTION REQUESTS

MEMORY MEMORY t, status, recall



The field length assigned to a job may be obtained or changed by the MEMORY request. Control will not be returned until the request is complete.

t = 0 CM If central memory field length is to be referenced.

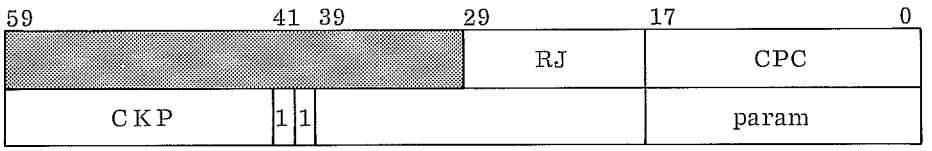
t = 1 ECS If extended core storage field length is to be referenced.

If the location addressed by status initially contains zero, no field length is altered: the current field length is returned in the upper half of the location and bit 0 is set to one.

If the upper half of the location initially contains a number, the field length is altered to equal the value of the number and bit 0 is set to one.

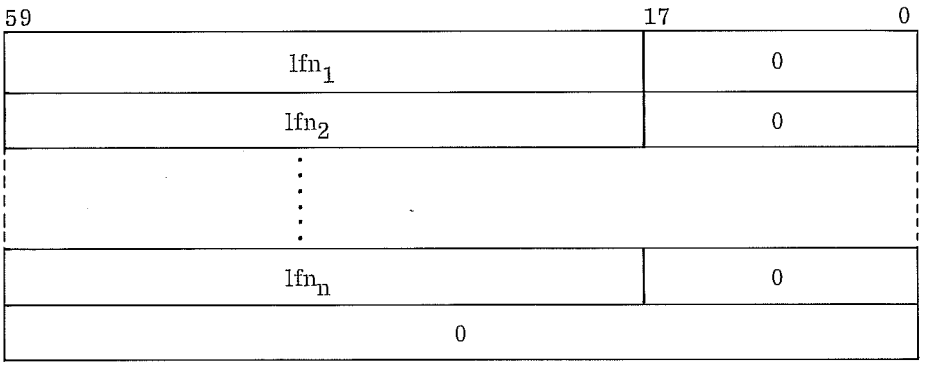
Bits 0-29 of the location addressed by status should initially contain zero in either case.

CHECKPOINT CHECKPT



A checkpoint dump may be requested from an executing program. A checkpoint dump is taken when this function is issued. The object program must have checked for conditions conducive to a checkpoint dump, such as end of reel, x logical records processed, etc. Checkpoint requests may appear more than once in an object program.

Param is the address of a parameter list, in the following format:

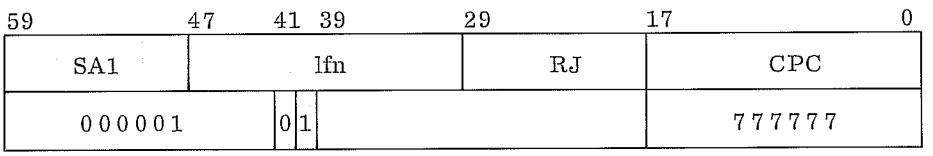


lfn_1 to lfn_n specify names of files from which contents are to be written on the checkpoint file. The list is terminated by a zero word; if the list is empty, the first word contains zero.

When the request is completed, bit 0 of the first word is set to one. If lfn_i cannot be found in the FNT/FST, bits 9-13 will be set to 22_8 (illegal function) upon return.

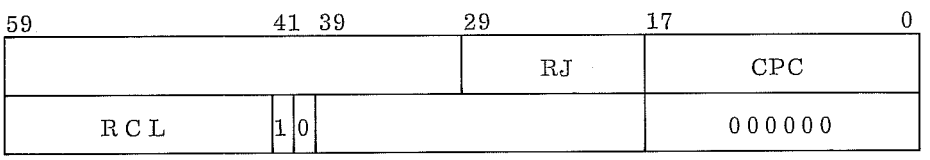
RECALL The RECALL request generates one of two calling sequences depending on the presence or absence of the lfn parameter. Execution of either function causes the job to relinquish the central processor.

RECALL lfn



lfn is the base address of a file environment table. Control is not returned to the program until bit 0 of the code and status field becomes a one, indicating that an input/output request has been completed for that file. Error checking is performed and an OWNCODE routine executed, if necessary, before control is returned. Since recall may be entered when the operation is initiated if the recall parameter is used, RECALL is needed only in the event that some useful processing can be done between initiating and completing an input/output operation.

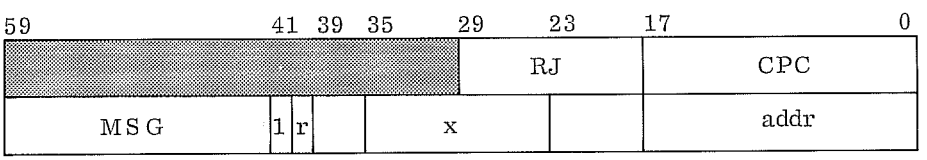
RECALL



If a RECALL request is issued without the lfn parameter, the central processor will be relinquished only until the next time around the monitor loop. The user must determine whether the condition that required a recall is still present.

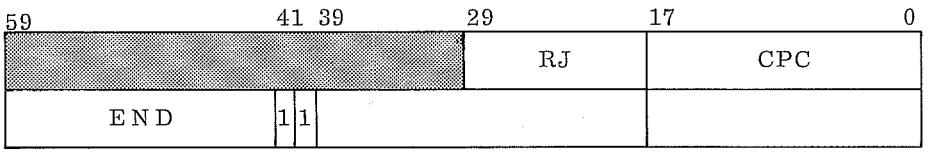
MESSAGE

MESSAGE addr, x, recall



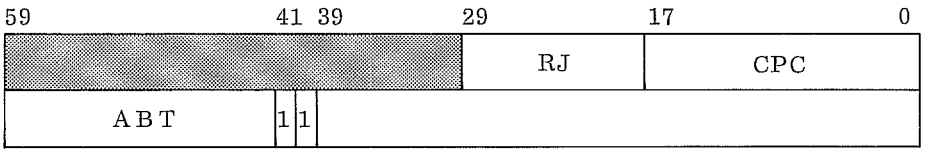
Execution of the MESSAGE function puts a message, which has been stored in display code beginning at location addr, into the job dayfile. Maximum message length is 60_{10} characters. SCOPE considers the message to end either at the first word with all zeros in the rightmost 12 bits or at the 60th character, whichever comes first. If the x parameter is non-blank, the message is displayed but not entered into the dayfile.

ENDRUN ENDRUN



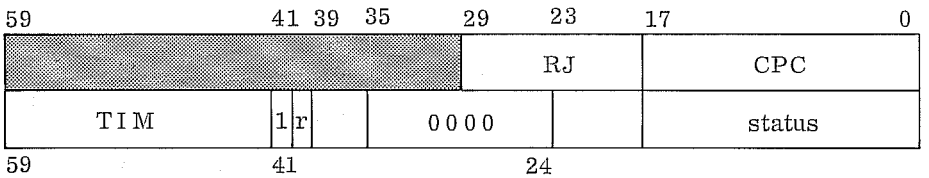
Execution of the ENDRUN function is the normal way of ending a run. SCOPE examines the control card record of the job deck, and begins execution with the next unused control card. If there are no more control cards or if the next card is an EXIT card, the job is terminated.

ABORT ABORT

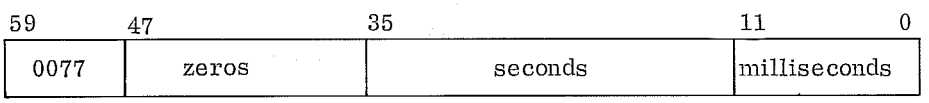


Execution of this function causes the monitor to terminate the job, just as if an error, such as out-of-bounds memory reference, had occurred. If the control card section of the job deck contains an EXIT card, the system continues processing the job with the next control card after the EXIT card.

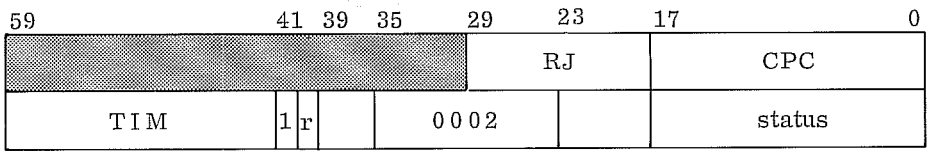
TIME TIME status, recall



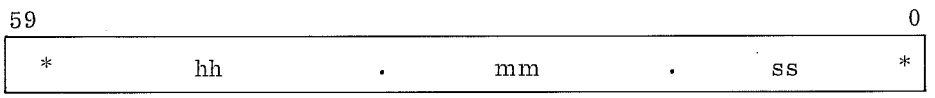
With this function, the central processor time used by the job is returned in status, in the following format:



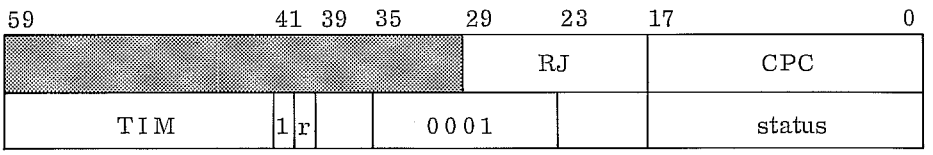
CLOCK



With this function, the current reading of the system clock is returned in status, in the following format:

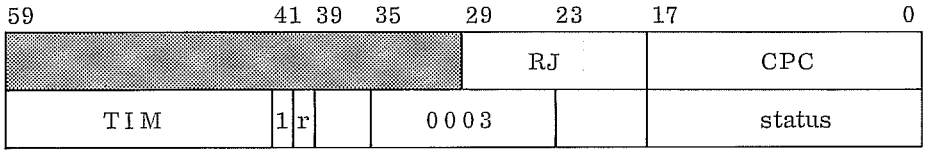


DATE DATE status, recall

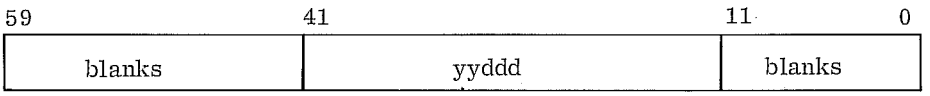


With this function the current date, as typed by the operator, with one leading blank and one trailing blank, is returned in status.

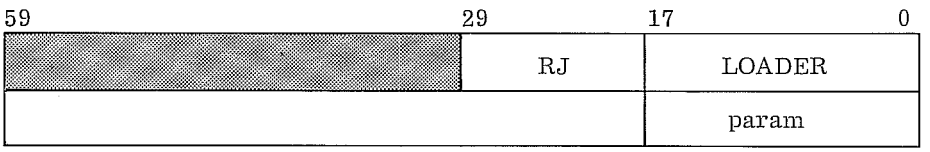
JDATE



With this function, the Julian date is returned in status, in the following format; yyddd is in display code:



LOADER LOADER param

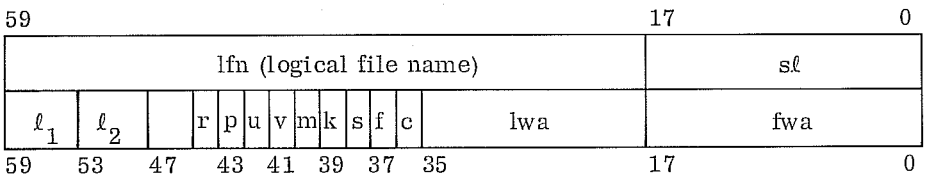


A program may request service from the loader with this function. Param is the location at which the user has established a parameter list for the load sequence. Only the parameters are described below. The loader is described in Chapter 4.

When a job area is initially loaded with program material, a small resident is placed within the user's field length. LOADER is an external symbol which is satisfied by the loader and which will ultimately reference an entry point in this resident.

Unlike control card requests for LOADER activity, user requests do not cause the specified file to be rewound. Instead it is the user's responsibility to position all files properly before issuing a user request. Upon a request for a full file load, LOADER loads programs only to the end-of-file. In all other cases, the file is searched for the specified programs end-around. If all programs are located, the file will be positioned immediately following the last program loaded. If not all programs are located, the file will be positioned at its original starting point and a fatal error flag returned to the user.

The load sequence parameter list begins at address param. The list consists of one or more 2-word entries, the last of which is followed by a full word of zeros. The format of an entry is shown below.



lfn One of the following:

Name of the file from which programs will be loaded

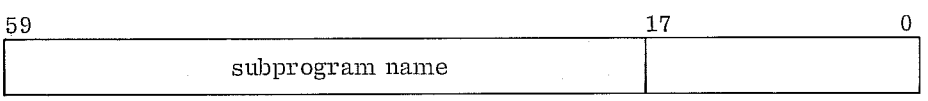
Name of an entry point in a program

Subprogram name

Zero

sℓ Location of a list of sections, a segment, or a list of subprograms to be loaded as a segment; or if segment loading is not requested, a list of subprograms to be loaded from file lfn. Names may not exceed seven characters. The list may be empty. It is terminated by a zero word.

Each entry in the sℓ list has the following format:



ℓ_1 Segment level (0-63) if s = 0; v = 0. Overlay level if s = 0; v ≠ 0. ℓ_1 is the primary overlay level and the secondary overlay level.

r Reset bit. If r ≠ 0, all loader tables are cleared before loading; normal loading only.

p Partial map bit. If p ≠ 0, an on-line partial core map is given.

u Library flag. When u ≠ 0, sℓ refers to a list of externals that are to be satisfied by loading from the system library. When v = 0 and u ≠ 0, an overlay will be loaded from the system library.

v Overlay flag. If v ≠ 0, an overlay load operation is requested.

m NOMAP flag. If m ≠ 0, all maps of segment or overlay load are suppressed. Otherwise a map is written on the OUTPUT file.

k Search key. If k ≠ 0, lfn is the name of an entry point. The search key is used to find the address of a previously loaded entry point, and no loading is performed.

s Segment flag. If s ≠ 0, a segment loading operation is requested.

f Fill flag. If f ≠ 0, unsatisfied external symbols are filled with out-of-bounds references.

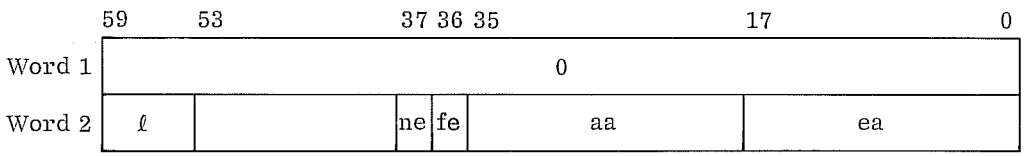
c Complete flag. If c ≠ 0, loading is to be completed by loading necessary subroutines from the system library. The origin and length of blank common will be established. Until loading has been completed, the length may vary between subprograms.

lwa Last location, relative to RA, available for the loading operation. If lwa = 0, the limit of program loading is the first word of LOADER tables stored in core descending addresses starting at fwa LOADER. For blank common declarations, lwa is designated as RA + fl-20_8.

fwa Initial location, relative to RA, at which to begin loading. If fwa = 0, loading begins at the next available location as determined by the current state of the loading operation.

Reply from LOADER

When LOADER has completed the requested operation (loading is not necessarily complete) LOADER signals the caller by setting the parameter list as follows:



ℓ Level at which segment was loaded. ℓ = 0 if segment loading not requested.

ne Non-fatal error flag. ne ≠ 0 if the following loading errors are detected by LDR:

Unsatisfied externals if c = 1.

Duplicate occurrences of a named program; second and subsequent occurrences are ignored.

fe Fatal error flag. fe ≠ 0 if the following loader errors are detected by LOADER.

Improper deck structure

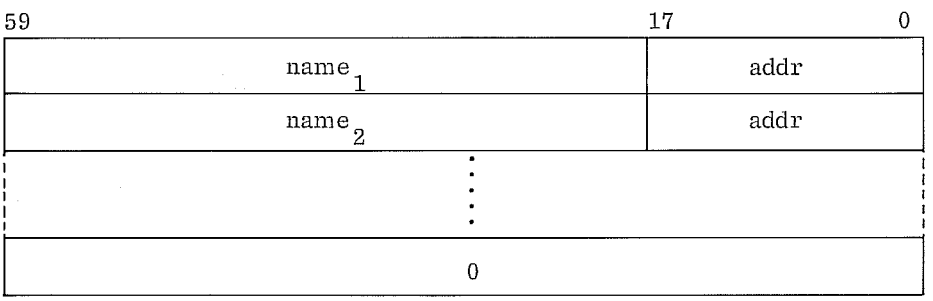
Improper parameter specification

Requested file name, program name, or entry point not found

aa Entry address. aa = 0 if less than two named XFER's were encountered. aa = address of next to last name if more than one named XFER was encountered.

ea Entry address. If k = 0, ea is the location (relative to RA) of last encountered named entry specified in a XFER table. If more than one named XFER is encountered, the last one is in ea, and the earlier entry in aa. If k = 1, ea is the location (relative to RA) of the named entry fn. If v ≠ 0, ea is the entry point to the overlay. If ea = 0, no name was found.

If sℓ ≠ 0, the list of entry points and/or subroutines to be loaded from the library contains the address at which each name is loaded. If the name was not loaded, the address is zero. The list then has the form:



User Request Processing

Examples of parameter lists to be processed by the loader are given below:

Load from File:

lfn = name of file

sℓ = 0

v = 0

k = 0

s = 0

Subprograms will be loaded from lfn until end-of-information is encountered. ℓ is ignored. If $c \neq 0$, loading will be completed.

Load Named Entry:

lfn = name of entry point in a subprogram or the name of a subprogram.

$s\ell = 0$

v = 0

k = 0

s = 0

The named routine will be loaded from the system library. ℓ is ignored. If $c \neq 0$, loading will be completed.

Load Segment from File:

The segment defined by the list at $s\ell$ will be loaded from lfn at level ℓ. If $\ell >$ current segment level, the segment will be loaded at the current level + 1. If $\ell \leq$ current level, segments at a higher level will be removed. If a subprogram specified in the segment list is not located on lfn before the entire file has been searched, the fatal error flag will be set. lfn is not rewound prior to loading.

lfn = name of file

$s\ell$ = address of list, contains a segment name or section names and subprogram names only

ℓ = desired level

v = 0

k = 0

s = 1

If $c = 0$, loading will be completed in that the origin and length of blank common will be established. If $c \neq 0$, loading will be completed normally. $f \neq 0$ will cause unsatisfied external references to be set to out-of-bounds references.

Load Named Subprograms from File:

The list of subprograms specified by the list at sℓ will be loaded from lfn and the System library.

lfn = name of file

sℓ = address of list

ℓ is ignored

v = 0

k = 0

s = 0

If c ≠ 0, loading will be completed.

Load Overlays:

lfn = name of file

sℓ = 0

ℓ1 = primary level

ℓ2 = secondary level

v = 1

s, f, c and k are ignored

The overlay file (built during the initial load from overlay cards and binary text) is searched for the unique identifier ℓ1, ℓ2. The overlay is then loaded into its absolute locations. The absence of such an overlay will cause the loader to set the fatal error flag.

LOADER OPERATION 4

6400/6500/6600 SCOPE version 3.1 includes a relocatable loader which provides high speed transfer from input or storage devices to central memory. Initially the loader is called by SCOPE control cards; later it may be called from the text of an object program.

Subprograms assembled or compiled independently, in absolute or relocatable binary, may be loaded and linked to one another or to library subprograms by the loader. The loader issues diagnostic messages on the dayfile and prints memory maps when requested.

A number of subprograms may be grouped together as a segment to be loaded, linked and, upon request, later delinked and removed as a unit by the loader. The loader can also generate overlays which are written out to a specified file in absolute format. These overlays may then be loaded by a smaller, faster version of the loader.

These features are governed by control cards, loader requests from object programs, and a standard relocatable subprogram format.

4.1 LOADING SEQUENCE

The loading operation proceeds in the following general manner:

1. The loader may be called as a result of control card requests for loading or by the user program.
2. The initial control card results in loading of the PPU routines, LOD, LDR, and a CPU routine, LOADER.
3. LDR handles all loader input/output and relocation to central memory; LOADER handles all bookkeeping, routine linking and delinking.
4. The combination of these two routines, hereinafter referred to as the loader, processes specific requests in the parameter list on the control card or in the user program. These may be:
   a. Build segment definition tables
   b. Build section tables
   c. Prepare overlays and write them out to a defined file
   d. Load absolute programs

e. Load relocatable program texts

f. Load segments

g. Load overlays

(d and e are subset functions of g and f, respectively.)

5. Programs can be loaded from more than one file (including the system library) for a single job.

6. During loading, all external reference points and entry points are linked together.

7. At completion of load and at the user's discretion, all unsatisfied references are filled from the system library or by out-of-bounds references.

8. During loading, a memory map is created for all programs other than main programs loaded from the system library (such as FORTRAN, etc.).

9. Loading is completed upon appearance of an EXECUTE or NOGO card. NOGO inhibits program execution and is used primarily to provide a map of the program. Subsequent loading following the NOGO will begin as if no programs had been loaded prior to NOGO. An EXECUTE card causes control to transfer directly to the loaded programs.

10. Loading of OVERLAYS by normal or segment jobs, or the converse, is not prohibited by LOADER. However, extreme care must be exercised in the allocation of core and communication between component programs. The loading of absolutely "original" code is also permitted but can result in complex situations when accomplished in overlays and segments.

4.2 SEGMENTATION

A segment is a group of relocatable subprograms which are to be treated as a unit by the loader. Segmentation allows the user to add programs as they are required and to eliminate those no longer required during the execution of his job. The user defines the subprograms to be included in a segment either with control cards or with parameters included in the object program loader call. To facilitate reference to groups of programs, a segment definition may contain both program names and section names. A section is a convenience in the loader scheme to reduce the number of program names appearing in segment calls.

Segments are loaded at levels ranging from $0\text{-}77_8$. Level zero is reserved for the initial, or main, segment. Segment zero must be the first segment defined; thereafter segments may be defined and loaded at any level.

When a segment is loaded, external references within the segment are linked to entry points in segments previously loaded at a lower level. Unsatisfied external references may be linked to entry points in segments loaded subsequently. Optionally, the user may specify that unsatisfied external references be satisfied, if possible, from the system library, thereby nominally including certain library subprograms within a given segment. If the level requested for loading a segment is less than or equal to the level of the last loaded segment, the loader performs a delinking operation. All segments previously loaded at a level equal to or greater than the presently requested level are removed and all linking of external references to entry points within these segments is eliminated, causing the external references involved to become unsatisfied again. Once delinking is complete, the segment is loaded at the requested level.

Ordinarily, only one occurrence of a given subprogram or entry point is loaded since all segments are linked to that subprogram. However, a user may force subsequent loading of an already loaded subprogram by explicitly naming it in another segment to be loaded at a higher level. Thereafter, all external references in higher level segments would be linked to the last loaded subprogram.

Example:

> The SINE routine is loaded in a segment at level 1. To try an experimental version of SINE, the user loads a new segment containing SINE at level 3. Now, although any references to SINE occurring at level 2 will be linked to the entry point in level 1, all segments loaded at level 4 or higher will be linked to SINE at level 3. This will occur until level 3 is delinked and removed as described above or until yet another SINE is loaded at a higher level.

Labeled common block references are established between programs in a given segment <u>but not between segments</u>. Therefore, delinking is not required. Blank common references are established between programs within a segment and also <u>between segments</u>. The origin and maximum blank common length is established in the first segment which declares blank common. If this segment is ever delinked, blank common will be re-established in the next segment loaded which declares blank common. The following diagram shows the storage allocation in core resulting from the loading of several segments:

Loading Order	Segment Level Loaded	Contents of User's Area After the Segment is Loaded				
1	0	0				
2	2	0	2			
3	4	0	2	4		
4	7	0	2	4	7	
5	2	0	2_1			
6	1	0	1			
7	2	0	1	2_2		
8	5	0	1	2_2	5	
9	7	0	1	2_2	7_1	

4.3 OVERLAYS

The loader provides the facility to subdivide a large task into portions, called overlays, and write them out in absolute form. These overlays can then be loaded at execution time without a relocatable loading operation. The resident loader for overlays is substantially reduced in size and may be easily retained with the job for subsequent loading. Overlays are generated through control cards processed directly by this loader (loader directives).

Each overlay is identified by an ordered pair of octal numbers, 0-77. The first number denotes the primary level; the second denotes the secondary level. A secondary overlay (non-zero secondary level) is associated with a subordinate to the primary which has the same primary level and a zero secondary level. Overlays (1,1), (1,2) and (1,3) are secondary overlays of the primary (1,0).

The initial, or main overlay, must be primary with level 0,0. It cannot have any associated secondary overlays; overlays numbered 0,1; 0,2; etc., are illegal. The main overlay remains in memory throughout the job. For any given program execution, all overlays must have unique identifiers.

Primary overlays all begin at the same point immediately following the main overlay (0,0). The loading of any primary overlay will destroy any other primary overlay. For this reason, LOADER will not return CP control to the instruction following the LOADER call. Instead, control will be transferred to the entry point of that overlay.

The origin of secondary overlays immediately follows their associated primary overlay, and they may be loaded only by their primary overlay or by the main overlay. The loading of a secondary overlay destroys any previously loaded secondary overlay. No more than three overlays are available to the user at one time: the main overlay, one primary, and one secondary.

When the loader detects illegal overlays during preparation, because of erroneous identification or size, an abort flag is set which causes the system to bypass the next EXECUTE or NOGO card.

The following example shows the storage allocation in core during an overlay loading operation:

Loading Order	Primary Level of Overlay	Secondary Level of Overlay	Contents of User's Area After this Overlay has been Loaded		
1	0	0	(0,0)	Must be first loaded overlay	
2	1	0	(0,0)	(1,0)	
3	1	1	(0,0)	(1,0)	(1,1)
4	1	3	(0,0)	(1,0)	(1,3)
5	2	0	(0,0)	(2,0)	
6	2	2	(0,0)	(2,0)	(2,2)
7	2	1	(0,0)	(2,0)	(2,1)
8	4	0	(0,0)	(4,0)	

4.4 LOADER DIRECTIVES

The following control cards are interpreted by the loader as directives for the loader execution. They may be interspersed with tables but may not be interspersed with cards making up a table.

4.4.1 OVERLAYS

OVERLAY (lfn, l1, l2, Cnnnnnn)

lfn is the file name on which the overlay is to be written; the first overlay card must have a named lfn. Subsequent cards may omit it, indicating that the overlays are related and are to be written in the same lfn. A different lfn on subsequent cards results in generation of overlays to the new lfn. l1 is the primary level number in octal; l2 is the secondary level number in octal. l1, l2 for the first overlay card must be 0,0.

Cnnnnnn is an optional parameter consisting of the letter C and a six-digit octal number. If this parameter is present, the overlay will be loaded nnnnnn words from the start of blank common. This provides the programmer with a method of changing the size of blank common at execution time. Cnnnnnn cannot be included on the overlay 0,0 loader directive. If this parameter is omitted, the overlay is loaded in the normal manner.

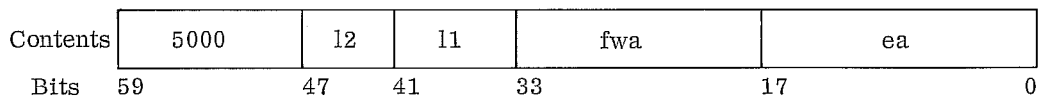
OVERLAY DECKS

The data, relocatable binary decks immediately following OVERLAY up to the next OVERLAY control card or an end-of-file, comprise the overlay deck. When the overlay deck has been loaded, loading is completed by satisfying undefined external references from the system library. The overlay and its identification are written as the next logical record in the file.

Each overlay has a unique entry which is the last transfer address (XFER) encountered in the overlay subprograms during preparation. External references which cannot be satisfied, even by the system library, result in job termination after loading is completed and maps are produced for all overlays. References to entry points in the main overlay may be made from primary and secondary overlays. References to entry points in a primary overlay may be made only from an associated secondary overlay.

OVERLAY FORMAT

Each overlay consists of a logical record in absolute format. The first word is an identification. Words 2 through end of logical record are data words.



l1	primary overlay level
l2	secondary overlay level
fwa	first word address of overlay (overlay is loaded at FWA).
ea	entry point to the overlay

4.4.2 SECTIONS

This card defines a section within a segment. Segments are loaded by user calls during execution or by MTR during initial load.

SECTION (sname, pn_1, pn_2, . . . , pn_n)

sname is the name of the section (7 characters maximum). pn_i are names of subprograms belonging to the section. If more than one card is necessary to define a section, additional cards with the same sname may follow consecutively.

All subprograms within a section are loaded whenever the named section is loaded. All section cards must appear prior to the SEGMENT cards which refer to the named sections.

4.4.3 SEGMENTS

All programs requiring segmentation loading must contain a SEGZERO card prior to any of the binary text.

SEGZERO (sn, pn_1, pn_2, . . . , pn_n)

sn is the segment name and pn_i are names of subprograms or section names which make up the main or zero level segment. Defining other segments in a similar manner reduces the list of subprograms in the loader call.

SEGMENT (sn, pn_1, pn_2, . . . , pn_n)

The parameters are defined as in SEGZERO. In a segment, all programs must reside on the same file. A segment defined in the user's program need not be defined by a SEGMENT card; however, a SEGZERO card is always required.

4.5 MEMORY ALLOCATION

4.5.1 SYSTEM USAGE

Storage areas are allocated within the user's declared field length in contiguous memory locations. The first 100_8 locations of the area are automatically assigned as follows:

RA+0		Reserved for use during execution
RA+1		
RA+2 : : RA+63		Parameters from the program call card upon entry. Available to user during execution for file names or other usage.
RA+64	0-17	Number of parameters
RA+65	0-17	Next available core location
RA+66	36-53	FWA of object program Pointer to first word of loader tables
RA+67	0-17	Address of first word of LOADER
RA+70 : : RA+77		Upon initial entry from a named routine call or an EXECUTE card, these locations will contain the card image, in display codes, of the card which called for execution.

If a SECTION card appears prior to an initial loading operation, a section definition table (SDT) is originated at RA+100_8 otherwise the user's first loaded subprogram is originated at RA+100. The origin of the user's area can be found in bits 0-17 of RA+66.

The system establishes loader tables at the high end of the user's field length area. The user must provide space for the loader and the loader tables in his field length declaration. Blank common may overlay the loader and its tables. Conversely, if the loader is called again it may overlay blank common. It is the user's responsibility to assure that his field length is long enough to accommodate the loader, its tables, and blank common if he is concerned with preservation of data.

There are no guarantees that the programmer cannot destroy the loader or loader tables. Both areas are checksummed and the checksum is verified upon initial entry into LOADER. If this initial verification routine is destroyed the results of RJ LOADER are meaningless.

The PPU portion of the loader (LOD) contains a further safeguard. Before LOD will honor any LOADER requests for actual input, key LOADER areas are checksummed. If the user program accidentally enters LOADER, the job can be terminated before any more data is loaded into the job area, thereby destroying clues as to the program bug.

4.5.2 USER ALLOCATIONS

The subprogram and associated labeled common blocks are assigned memory as they are encountered. Blank common relocation information is preserved until loading is completed, at which time it is allocated following the last loaded program and/or labeled common block.

The initial declaration of a labeled common block establishes the maximum length for that data block. Length declarations in subsequent programs must be less than or equal to the original declaration. A diagnostic occurs if this rule is violated.

Declarations of blank common may vary between subprograms, and the largest declaration determines the memory allocation.

4.5.3 SEGMENT ALLOCATIONS

After a segment is loaded, the current loader tables are moved to a point immediately following the last loaded subprogram/common block.

The user must allow for the space consumed by the loader table within his field length definition.

4.6 MEMORY MAP

Following completion of loading, an optional map of the user's area may be produced in the OUTPUT file. The map includes:

- Names, lengths, and locations of entry points with a sublist of all programs referencing the entry point
- Names and locations of common blocks
- Total length of all loaded programs and common blocks
- Length of the loader and its tables
- Unsatisfied external references

During execution of a segmented or overlay job, a record of a new segment or overlay load is provided each time a call is made to LOADER. This map can be suppressed by setting the NOMAP bit in the LOADER parameters to 1.

SYSTEM LIBRARY PREPARATION AND MAINTENANCE 5

The SCOPE system library includes system, library maintenance, and the SORT/MERGE programs, language processors such as COMPASS, FORTRAN, and COBOL, and coded records.

The EDITLIB program creates and maintains the SCOPE library. Modifications may be applicable to only the currently operating environment or they may be permanent changes to a system library deadstart load tape.

The library material may be in three forms:

- System library tape, which may be deadstart loaded.
- File called SYSTEM on disk storage, which is virtually a copy of a system library tape after deadstart loading is completed. Parts of SYSTEM will also have been copied into central memory. In particular, library programs that are to be available from CM will have been copied there from the disk file SYSTEM.
- Files containing one or more logical records in a form suitable for insertion into SYSTEM or a system library file.

A system library tape is written as three groups of records.

- System Records: The first five records on a system library tape may be altered only through a deadstart load. These records of a currently operating system are as follows:

PLR	preloader
STL	system dead start loader
CMR	central memory resident
MTR	system monitor
DSD	system display

- Directory: The library directory is written as two records.
- Library Records: The remainder of the library consists of the compilers, assemblers, PP programs, etc.

5.1 EDITLIB CALL CARD

EDITLIB is called into operation when an EDITLIB card appears in the control card record of a job. This card may assume one of two forms which affects only the initial processing (thereafter processing is identical for the two forms).

```
EDITLIB.
```

With this form, the current system directory is saved on a common file called SSSSSST.

```
EDITLIB (RESTORE)
```

With this form, the current system directory is replaced by the contents of the common file called SSSSSST.

5.2 EDITLIB FUNCTION CARDS

The EDITLIB program initially reads one record from file INPUT and copies it to a scratch file. This file contains the function cards to control the run initiated by the EDITLIB call card. The first card image read informs EDITLIB of an action to be performed. When the action is complete, the next card image is read, and the action it calls for is performed. When the record is exhausted, EDITLIB returns control to SCOPE.

A function card begins with the name of the function, which may start in any column. Parameters may be separated by blanks or any characters other than period, dollar sign, right parenthesis, minus sign, or asterisk. A period or right parenthesis terminates the card, and the minus sign and asterisk have special meanings for the ADD function. In a word that begins with a letter, a dollar sign is treated as a letter; elsewhere, a dollar sign acts as a separator.

ADD (A, B, CM) and ADD A+B, , , CM. have the same meaning.

A function card requesting a condition that already exists is ignored. Errors in function cards result in a message. For some errors the function card is ignored, for others the job is terminated.

The following notations are used in the descriptions of the function cards:

- s — Source file; name of a file from which data is to be read. The name SYSTEM is reserved for the current operating system.
- d — Destination file; the name of a file onto which data is to be written. The name SYSTEM is reserved for the current operating system.

p Name of record to be processed.

r Residence. The device from which a particular record is to be loaded when that record becomes part of SYSTEM. All records are written on disk (DS). In addition, some records may be kept in central memory (CM) for faster access. In the latter case, the residence is central memory.

e Edition. A number, 0-63, attached to a record by the ADD function.

5.2.1 SYSTEM MODIFICATION FUNCTIONS

The following functions may be requested without a preliminary READY function.

MOVE

MOVE (p,r)

The residence of a record is changed in the SYSTEM directory. If a record is moved into or out of central memory, the body of the record is added to or removed from the directory. Storage is moved accordingly.

DELETE

DELETE (p)

Record p is to be deleted from the SYSTEM directory. If record p was resident in central memory, the body of the record is removed and storage is moved accordingly. The disk copy of the record is not affected.

LIST

LIST (s)

List causes file s to be rewound, the first five records are skipped and the sixth and seventh records are read on the assumption that they are the entry point table and program name table for the system which the file constitutes. The name of the file is written on OUTPUT, and the two tables are used to write, on OUTPUT, a list of the records in the file. For each record the following information is given:

Name

Description (PP PROG, CP PROG, OVERLAY or COS)

Residence (CM or DS)

Edition number (2 decimal digits)

Length (5 octal digits); length is in CM words, excludes the prefix of the record.

If the record is a program with entry points, they are listed immediately below, indented.

5.2.2 LIBRARY REVISION FUNCTIONS

READY

In response to the READY function, EDITLIB prepares to create a system library on file d. A READY function is required for any modification more complex than deleting or changing residence of one or more programs from SYSTEM.

```
READY (d)
```

If d ≠ SYSTEM, a model of an empty directory and an empty scratch file is prepared. Subsequent function cards cause information to be written into the directory and scratch file.

```
READY (SYSTEM)
```

The directory of the presently operating SYSTEM is to be manipulated. The current directory is copied within the field length of the EDITLIB program for subsequent modifications.

```
READY (SYSTEM,*)
```

An empty directory is created. This form is used when SYSTEM is to be completely replaced from sources other than the currently operating SYSTEM.

With the above forms, any records added are written into a common file called SSSSSSU. This file is not rewound between one dead start and the next. It is used as a scratch file of indefinite length. Directory entries for programs added since dead start will point to SSSSSSU.

TRANSFER

PLR, STL, CMR, MTR and DSD records are not indexed in a system file directory. This function is used to add such records to a system file.

```
TRANSFER (s,n)
```

If n is a number, the next n records are copied from file s to the new file. If any records have a prefix, the run is aborted.

If n is not a number, the prefix of the next record name on file s is verified against n; the prefix is discarded and the remainder of the record is copied to the new file. If the name in the prefix is different from n or if the record has no prefix, the run is terminated.

TRANSFER (s) and TRANSFER (s,1) are equivalent statements.

TRANSFER (s,n,2)

In this form of the TRANSFER statement, n is a name. The next record on file s is checked for a prefix containing name n. The prefix is discarded and the rest of the record is copied to the new file. The file record is finished by adding, without checking prefix, all of the next record on file s.

TRANSFER (s,n,2) is used to put the PLR and STL records on the beginning of a system file. Both records include PP and CP programs which must be assembled separately. TRANSFER combines the PP and CP programs unless PLR or STL are being copied from one system file to another.

ADD ADD (p,s,r,e)

Record p from file s is added to the system library currently under construction. The record is assigned residence r. If r is absent, and file s is a system library, the residence p is taken from file s; otherwise, disk residence is assumed. The edition number attached to each record added is e, 0-63. If e is absent and s is a system library, the record is transferred with its same edition number. If e is absent and s is not a system library, the edition number will be 0. Since r is never a number, and e is always a number, they cannot be confused, and any of the following is acceptable:

ADD (p,s)	ADD (p,s,e)	ADD (p,s,r,e)
ADD (p,s,r)	ADD (p,s,e,r)	

If s ≠ SYSTEM, s must be positioned at the beginning of record p; if file s is not properly positioned, a diagnostic is given and the function card skipped. If s = SYSTEM, pre-positioning is not necessary.

In addition to a single record name, the p parameter may assume the following forms:

$p_1 - p_2$	Records p_1 through p_2 are added to the system library under construction. p_2 must appear after p_1 in the file s.
p - *	All records on file s from record p to the end of the file are added to the system library under construction.
*	All records on file s from the present position to the end of the file are added. If s = SYSTEM, all records listed in the directory are added.

The following two functions are needed only to add special records from a file other than a system library.

ADDBCD

ADDBCD (p, s, r, e)

Parameters have the same meaning as for ADD; however, p is the name of a coded record. An overlay is created and written on the new system library. The first card of a coded record to be added to a system library must contain the name p, beginning in column 1. p must be the name of a single record; the formats $p_1 - p_2$, $p_1 - *$, and * are not allowed.

ADDTEXT

ADDTEXT (p, s, r, e)

This is like ADDBCD except that file s is presumed to be a coded file formatted as the COMPILE file output by EDITSYM. Even though there is a serial number after column 80 on each line, ADDTEXT discards everything after column 72 to save space in the overlay. p must be the name of a single record; formats $p_1 - p_2$, $p - *$, and * are not allowed.

ADDCOS

ADDCOS (p, s, r, e)

Parameters have the same meaning as for ADD; however, p is the name of a record without a prefix. ADDCOS is used to add the Chippewa Operating System RUN compiler and object routines to a SCOPE system library.

DELETE

DELETE (p)

This function used after a READY function causes record p to be deleted from the new directory and system library under construction.

COMPLETE

COMPLETE

This function causes the file d initialized by a READY function to be completed. A COMPLETE function without a preceding READY is meaningless and the job will be terminated.

If d = SYSTEM, the revised directory prepared since READY was executed replaces the current system directory.

If d ≠ SYSTEM, the new directory and scratch file is written on file d.

LENGTH LENGTH (p)

If p is a digit between 4 and 9, this function causes EDITLIB to request a field length sufficient to accommodate a directory model of p times 10000_8 words. Before any function cards are obeyed, EDITLIB must have obtained a field length for a directory model of a minimum of 30000_8 words.

The directory models to be accommodated are the directory of the running system as EDITLIB finds it initially and any modifications of it that EDITLIB may make between obeying a READY (SYSTEM) card and the next following COMPLETE. card. The length of the directory is 2+2p+e+r, where p is the number of programs and overlays in the library, e is the number of entry points in the CP programs in the library, and r is the number of words in the bodies of all the CM-resident programs. Normally a directory of more than 30000_8 words is not required; however, a larger field length can be established by using an appropriate LENGTH (p) card as the first function card read by EDITLIB.

5.2.3 POSITION FUNCTIONS

A position function may appear anywhere within an EDITLIB deck.

REWIND REWIND (s)

File s is rewound.

SKIPB SKIPB (s,n)

n logical records are backspaced on file s. n may be 1 to $2^{17}-1$.

SKIPF SKIPF (s,n)

If n is numeric (1 to $2^{17}-1$), n logical records on file s are skipped in a forward direction. If n is a name, the skip is forward to the end of logical record n. If the end-of-information is reached before n can be satisfied, the job is terminated.

5.3 EDITLIB EXAMPLES

The running system includes program FORTRAN, a resident on disk only.

- Bring into central memory residence, run a batch of compilations, and return the program FORTRAN to disk residence:

```
job card
EDITLIB.                    Control card begins an EDITLIB run.
7
8
9
MOVE (FORTRAN,CM)           Function card read by EDITLIB.
6
7
8
9                           Ends EDITLIB run and job.
job card                    Begins batch of compilations.
  :                         Cards for the compilation jobs.
6
7
8
9
job card
EDITLIB.                    Control card begins an EDITLIB run.
7
8
9
MOVE (FORTRAN,DS)           Function card read by EDITLIB.
6
7
8
9                           Ends EDITLIB run and job.
```

- Construct a file called NEWSYS on tape, which duplicates the system now running:

```
job card
REQUEST NEWSYS.             Operator will assign a tape.
EDITLIB.                    Save current directory and begin EDITLIB run.
7
8
9
REWIND (SYSTEM)
READY (NEWSYS)
TRANSFER (SYSTEM,5)         Copy the first five records of the system.
```

ADD(*, SYSTEM)	Copies to a scratch file all programs found in directory.
COMPLETE.	Copy tables and scratch file to NEWSYS.
$^{6}_{7}{}_{8}{}_{9}$	Ends EDITLIB run and job.

- Same as preceding example, except that the program in the present system called 2TS is to be replaced in NEWSYS by another program called 2TS, which is at hand as a deck of cards; 2TS is to have disk residence:

job card	
REQUEST NEWSYS.	Operator will assign a tape.
EDITLIB.	Save current directory and begin EDITLIB run.
$^{7}_{8}{}_{9}$	
REWIND (SYSTEM)	
READY (NEWSYS)	
TRANSFER (SYSTEM, 5)	Transfer the five system records.
ADD (*, SYSTEM)	Copy to a scratch file all the programs found in the directory.
DELETE (2TS)	Remove program called 2TS from new directory.
ADD (2TS, INPUT, DS)	Add a new 2TS from INPUT.
COMPLETE.	
$^{7}_{8}{}_{9}$	
the new 2TS binary deck	
$^{7}_{8}{}_{9}$	
$^{6}_{7}{}_{8}{}_{9}$	

- Replace, temporarily, the program called 2TS; try it out by runs within the same job; and finally restore the original system:

job card

EDITLIB. Saves present directory and, therefore, system.

Control cards for runs that will test new 2TS

EDITLIB (RESTORE) Restores what was saved by EDITLIB.

7_{8_9}

READY (SYSTEM) Function cards for EDITLIB.

DELETE (2TS)

ADD (2TS, INPUT, DS)

COMPLETE.

7_{8_9}

New 2TS binary deck

7_{8_9}

Input cards for runs that will test new 2TS

$6_{7_{8_9}}$

- Produce a new system file called MYTAPE which is a copy of the presently running system except that MTR is to be replaced:

job card

REQUEST MYTAPE.

EDITLIB.

7_{8_9}

REWIND (SYSTEM)

READY (MYTAPE)

TRANSFER (SYSTEM, 3) Transfer PLR, STL, and CMR to MYTAPE.

TRANSFER (INPUT, MTR) Transfer the new MTR to MYTAPE.

```
SKIPF (SYSTEM,1)          Skip over the old MTR.
TRANSFER (SYSTEM,1)       Transfer DSD to MYTAPE.
ADD (*,SYSTEM)
COMPLETE.
7
8
9
New MTR binary deck
6
 7
  8
   9
```

- Simulate the dead start loading of a system tape called MAYBE, assuming its first five records to be the same as on file SYSTEM:

```
job card
REQUEST MAYBE.
EDITLIB.
7
8
9
SKIPF (MAYBE,7)           Skip over the 5 system records and the 2
                          directory records.
READY (SYSTEM,*)
ADD (*,MAYBE)
COMPLETE.
6
 7
  8
   9
```

Any following jobs, provided they do not call EDITLIB, will use the system from MAYBE. Then the original system can be restored by:

```
job card
EDITLIB (RESTORE)
6
 7
  8
   9
```

The UPDATE program enables the user to organize a collection of programs into a file and operate upon them with updating facilities. The primary feature in UPDATE which differs from other symbolic updating programs is the invariance of card identification: a given card receives a sequence identification which never changes and which is used as a reference when making modifications. Thus, barring overlapping corrections, symbolic corrections to the program library can be keyed to any version of the file.

6.1 CALLING UPDATE

UPDATE exists as a program within the SCOPE Operating System. The UPDATE call card contains information which directs UPDATE to specific files and modifies the operation of the program. The parameter specifications listed below may occur in any order. Each parameter must be specified by a character string of the form:

```
ident=value
```

ident is any character string not containing a comma, period, or parenthesis. The first letter is one of those specified for parameter names. value is a string of characters following an equal sign (blanks ignored) which specifies a file name and is subject to file name format rules. If no value is specified, a default value is assumed.

PARAMETERS ON THE UPDATE CALL CARDS

Parameter	Value	Default Value
P = OLDPL	File containing old program library. Meaningful only for corrections.	OLDPL is assumed.
N = NEWPL	File containing new program library.	No new program library is written.
I = INPUT	File containing control cards.	INPUT is assumed.
L = OUTPUT †	Listable output file.	OUTPUT is assumed.
C = COMPILE††	File onto which is written the card images to be assembled.	COMPILE is assumed.
S = SOURCE	File on which a copy of the source deck is written. Contains a copy of the cards necessary to regenerate the existing library minus the inactive cards.	No source deck file written.
Special Options		
F	Selective assembly feature is to be ignored. All decks are to be written onto compile file. †††	Only modification decks and those cards containing modifications are written.
Q	Speeds updating process by specifying information on routines which will be updated; modifies the updating operation. (Section 6.3.6)	††††

---

† The L option does not list the card images of a program deck and their sequence identifiers. To obtain such a listing, the COMPILE file must be copied onto OUTPUT.

†† COMPILE output depends upon the mode of updating. If C=0, no compile file is written.

††† In effect, during initial program library creation the F parameter is always on.

†††† This option is effective only if correction runs are being performed.

6.2 STRUCTURE OF PROGRAM LIBRARIES

Program libraries, usually stored on magnetic tape, contain symbolic cards of the programs being maintained. They are divided into decks by the insertion of *DECK and *COMDECK cards into the text stream. Although the deck division is arbitrary, the specific division has ramifications in the selective assembly feature when updating and in the original serialization when creating a program.

The program library file consists of three sections:

- Directory — List of all identifiers used
- Deck list — List of all decks on the file including those no longer current
- Text stream — Card images and control information known as correction history bytes (CHB's)

The first word of the program library file is composed of two counters: the count of the identifiers contained in the first section (bits 35-18), and the count of the deck names contained in the second section (bits 17-00).

The identifiers, each left justified, follow the first word. The 42 high-order bits of an identifier are filled with trailing zeros; miscellaneous data is stored in the low-order 18 bits.

Deck list words are structured in a similar manner.

In the text stream, each card image is preceded by at least one word of control information arrayed as follows:

Bit	
59	Activity bit; contains a 1 if the text card, which immediately follows the control information, was active at the time the program library is written.
58-54	Unused (zero value).
53-36	Number of words used by the compressed card image. This information speeds up input operations.
35-18	Primary CHB; identifies the deck or correction set under which this card was introduced and thus gives the card its alphanumeric name. Secondary CHBs, recorded in bits 17-00 of the first word, continue in subsequent 18-bit bytes in the low order 54 bits of each word and terminate with a zero-value CHB.

CHBs record the effect of a correction set on a given card. The primary CHB in the control word assigns an alphanumeric name to the card by identifying the deck or correction set under which the card is introduced. The 18-bit CHB field is divided as follows:

Bit	
17	Not used.
16	Activity bit; contains a 1 if the correction set activated the cards.
15-00	Identify the correction set which performed the action. Provide the ordinal into the identifier table.

The compressed card image begins in the word subsequent to the one which contains the zero-value CHB. 00 characters represent consecutive blank columns. A two-character field of 00xx represents xx+1 blank columns; the value 0000 represents end-of-card. The value of 55 is retained for one blank column.

CARD IDENTIFICATION

Card identifiers are composed of the following elements:

Alphanumeric identifier	Maximum length of seven characters. The first CHB on the card corresponds to this identifier. It is defined on the *DECK and *COMDECK card for program library creation or on the *IDENT cards in the case of correction runs.
Period	Separates the two elements of an identification.
Sequence number	Obtained by counting the number of cards with the same identification.

Once assigned, an identification is permanent: it cannot be removed or changed.

Example:

SC12.42 refers to the 42nd card (sequence number) with the identifier SC12. In this example, the identifier is composed of a reference symbol (SC) which is combined with a reference number (12).

6.3 UPDATE CONTROL CARDS

Control and data cards (processed by UPDATE from the input file unless otherwise specified) are grouped into the following categories:

File manipulation cards:

*REWIND
*SKIP
*READ
*LABEL

Creation directive control cards:

*DECK
*COMDECK
*END

Correction directive control cards:

*IDENT
*PURGE
*DELETE
*RESTORE
*INSERT
*YANK
*/
*ADDFILE

Assembly directive control card:

*COMPILE

Output directive control cards:

*DECK
*COMDECK
*WEOR
*CALL

Text cards:

Any card not included above.

All control cards are punched on an 80-column card. The control word is signified by an asterisk in column one; the word begins in column 2, and terminates with a blank or comma. Parameters may begin in any column after the control word with any number of intervening blanks. No embedded

blanks are permitted in the parameter string (except for the *LABEL card). All identifiers, both for decks and correction identifiers, are one to seven alphanumeric characters. All numeric fields contain decimal numbers.

6.3.1 FILE MANIPULATION CARDS

The following control cards may be inserted at any point in an input record. File manipulation allows text and other control cards to be read from files other than the main input file. Except for *LABEL, they may not appear on other than the main input file. File manipulation cards may not reference any of the files UPDATE is controlling.

REWIND

*REWIND fname

This card causes the file indicated by fname to be rewound.

SKIP

*SKIP fname,rent

The SKIP card causes the named file, fname, to be spaced forward the number of records specified in the numeric field, rent. If the record count is zero or absent, the file is spaced forward one logical record.

READ

*READ fname

UPDATE derives input data from the named file. UPDATE reads from this file until an end-of-record is encountered, at which point it returns to the main input file. Any cards, except SKIP, REWIND, ADDFILE and READ, may occur on the fname file.

LABEL

*LABEL label name

The LABEL card is meaningful only in a creation operation. It precedes the first DECK or COMDECK card; it specifies the 20-character label name which will be given to the program library being created. The UPDATE program automatically updates the edition number and labels program libraries generated as a result of corrections. The label must be punched in columns 11-30.

6.3.2 CREATION DIRECTIVE CONTROL CARDS

The following control cards are used to create a new program library.

DECK

*DECK dname

dname Name of deck being introduced

This name must be different from that of any previously introduced deck or correction identifier. A deck is terminated by the first occurrence of another *DECK, *COMDECK or end-of-record. All intervening cards comprise text. They are identified with the deck name and numerically sequenced starting at 00001 for the *DECK card itself. File manipulation control cards may be embedded within the data cards, but they are not included in the numbering scheme. Cards introduced as a result of *READ, however, are included in the numbering scheme. Output directive control cards other than *DECK and *COMDECK may also be introduced and numbered.

COMDECK

*COMDECK dname

The COMDECK card introduces a common deck. This card is subject to the same rules as the DECK card concerning naming and numbering; however, it is not output in the same manner (section 6.3.5).

END

*END

The *END card is provided for EDITSYM compatibility; it is ignored and suppressed. If the S option is selected, UPDATE regenerates *END cards.

DECK GROUPING

In the usual application, a *DECK or *COMDECK card would precede the definition of each deck or common deck in a system. However, more than one subprogram may be included in a deck.

Example:

```
DECK      FIRST
          IDENT     FIRST
          .....
          END
          IDENT     SECOND
          .....
          END
```

Deck grouping is chiefly a function of the output section of UPDATE. Normally, two decks are grouped together if modification of one requires reassembly of both decks. Deletion of a DECK card, however, removes the deck division and groups two or more decks together. Similarly, insertion of a DECK card in a later updating run will introduce a division. If, as in the above example, two subprograms are joined into one deck, all cards are identified by the first deck name regardless of later insertion of a DECK card for the second deck.

6.3.3 CORRECTION DIRECTIVE CONTROL CARDS

The following control cards are used for correcting and updating program libraries. In the descriptions:

a and b represent identifiers, normally of a card or correction set.

n and m represent decimal numbers corresponding to card numbers.

IDENT

*IDENT idnam

The IDENT card introduces a correction set. All corrective operations except PURGE and ADDFILE must occur after an IDENT. All cards introduced in this correction set are identified by idnam, which is subject to the rules for identifiers. This identifier must not be the same as any identifier currently in effect, such as the original deck names and any later identifier names. Presumably, the identifier would be a correction set number, such as SC12, or some other unique name. This identifier holds until the next IDENT or PURGE card.

PURGE

*PURGE idnam

The PURGE card may appear in the place of an IDENT card. It causes the complete purging of any reference to the correction set idnam. Idnam must, therefore, reference an identifier already present in the correction set. All cards introduced under the specified identification are physically removed from the new program library; all corrections performed in the named correction set are removed by squeezing out the correction history bytes of that identification. Since permanent sequencing information is affected, the PURGE card must be used with care.

PURGE simulates the effect of YANK described below; however, the results of PURGE cannot be undone, the results of YANK can be altered. PURGE can be used to remove completely a deck of cards. Assume that routine CIO is initially introduced with the deck name CIO. All cards of CIO (and only those cards) have identifier CIO. To remove CIO completely from the

deck, two modifications are required:

```
*IDENT      xxxxxx
*DELETE     CIO.1,CIO.978      (indicates 978 cards)
*PURGE      CIO
```

The first statement deactivates all inserts made to CIO and the second removes all CIO cards.

A purged identifier may no longer be used or referenced.

```
*PURGE idnam,*
```

This alternate form of the PURGE statement is used to purge all correction sets introduced on or after the introduction of the one identified by idnam. Since all modifications introduced are recorded in the library, this form is order-dependent and may be used to return a program library to an earlier level. Purged items cannot be restored, however.

DELETE

```
*DELETE     a.n.
*DELETE     a.n,b.m
```

Cards may be deactivated with the DELETE card. The first form of the statement specifies one card; the second, an inclusive range of cards. This range may include cards already deleted, which are deleted again by appending a correction history byte. In the second form the b.m card must occur after the a.n card; in both forms all referenced cards must exist. Text cards may be inserted after the last card deleted.

RESTORE

```
*RESTORE    a.n
*RESTORE    a.n,b.m
```

Specified cards are reactivated with the RESTORE card. The first form of the statement specifies one card; the second an inclusive range of cards. Cards are restored by appending a correction history byte. Text cards may be inserted after the last card restored.

INSERT

```
*INSERT     a.n
```

Cards may be inserted with this card. a.n specifies the card after which the insertion is to be made.

YANK *YANK a

All effects of a correction set may be removed with the YANK card. This correction set, identified by a, must be extant at the time the YANK card is encountered. YANK restores the library to a form it had before the correction set occurred. In effect, all CHB's referring to the correction set are ignored. If a YANK card was included in the correction set, its effect is nullified. Insertions may not follow the YANK card.

/(slash) */ any comments

Comments to be listed with the correction set may be included with this card. It has an asterisk in column 1, a slash in column 2, and a comma or blank in column 3. This card is ignored by UPDATE; it is simply copied onto the report.†

ADDFILE *ADDFILE fname,a.n

When ADDFILE is encountered, UPDATE reads creation directive control cards and text card data from the named file, fname, and inserts this information after the a.n card on the old program library. The first card on fname must be *DECK or *COMDECK. UPDATE reads from this file until an end of record, which returns UPDATE to the main file. The ADDFILE card can appear only on the main input file. The ADDFILE card cannot appear when the UPDATE (Q) option is being used.

6.3.4 ASSEMBLY DIRECTIVE CONTROL CARDS

The selective assembly feature is handled by the control cards described below. Normally, only modified decks are written onto the assembly output file. To control this process at the deck level, *COMPILE cards are introduced following all correction directives. After the first such card, only *COMPILE cards may appear until the beginning of the next correction set or end-of-record.

†If this card occurs within an ADDFILE it will not be recognized.

COMPILE

```
*COMPILE    a,b,c...,d
*COMPILE    a.d
```

The COMPILE card causes the specified decks to be assembled. With the first form shown above, all decks are specially mentioned. In the second form, decks to be assembled include the two specified as well as all decks listed between them in the list of deck names produced by UPDATE. All decks affected by these statements are assembled regardless of the existence of any modifications within them.

If the full assembly feature is selected, the *COMPILE cards have no effect, although such cards are legal and are processed.

6.3.5 OUTPUT DIRECTIVE CONTROL CARDS

Output directive control cards control the compile file output. The DECK and COMDECK cards have additional functions during program library creation. These control cards are included in the program library.

DECK

```
*DECK dname
```

DECK delimits a deck for compile output. This division is meaningful during a correction run when the selective assembly feature is employed. Under this mode of operation, the only decks included on the assembly files are those in which one or more cards have been changed. In selecting the cards to be assembled, the UPDATE program compares card activities in the current run with those which existed when the program library was created. If any common deck called within a deck has been changed, that deck is considered to be changed.

COMDECK

```
*COMDECK dname
```

This card introduces a common deck. Common decks are not written onto the compile file but are saved by UPDATE and introduced into the compile file as a result of a CALL statement. The common deck must precede the call.

WEOR

```
*WEOR n
```

This statement writes an end-of-record of level n onto the COMPILE file. If n is greater than or equal to 15, an end-of-file is written. The WEOR card does not appear on the compile file. WEOR must be inserted into a deck on the program library and is effective only when that deck is written onto the compile file.

CALL *CALL dname

The CALL card is used to insert the text of a previously encountered common deck, dname, into the compile file. Common code, such as system symbol definitions, may be declared in the common deck and used in subsequent decks (or assemblies) without replicating the data cards. The CALL card does not appear on the assembly output file. The contents of the common deck, excluding the COMDECK card, follow immediately. The CALL card cannot occur within a common deck.

6.3.6 Q OPTION EFFECTS

The Q-option must be specified on the UPDATE call card and has effect only if corrections are being performed. If the Q option is used incorrectly, the UPDATE operations result in job termination.

The following modifications of the updating operation take effect under this option:

1. Only routines named within *COMPILE statements are written onto the COMPILE file. The set of names in the *COMPILE statements must include all routines to be modified.
2. If corrections are to be made to common decks, UPDATE will not automatically include onto *COMPILE the decks which call the common deck. Propagation of these modifications is left to the user.
3. If corrections are specified to routines not included in the *COMPILE statements, UPDATE will abort because of unprocessed corrections.
4. At the end of the updating pass, positioning of the old program library is unpredictable.
5. The selective assembly feature has no effect.
6. A new program library cannot be produced; if so specified, it is ignored.
7. If a source file is requested (S), all the common decks as well as those decks specified only on the *COMPILE cards will be produced.

6.4 LISTABLE OUTPUT FROM UPDATE

When a new program library is created, the list file contains a copy of all file manipulation and creation directive cards, as well as diagnostics.

During a correction pass, the listings are more detailed. The first listing is a printout of the correction sets as encountered. Each IDENT (or PURGE) appears on a titled page. A printout of each card on the input file follows. All cards read in by the READ command are included; they are identified on the right by the file from which they were read.

The second set of listings, a continuous commentary of all effective changes introduced to the file, includes all purged cards as well as cards for which the activity status changed since they were placed on the program library. Cards inserted by the ADDFILE statement are not listed.

Diagnostics are listed as they occur. Whether or not the updating process is successfully accomplished, an appropriate dayfile message appears.

6.5 OVERLAPPING CORRECTIONS

A correction set can be re-modified by a later correction set. Corrections which modify a card more than once in one correction set are marked in the output listing by the word OVERLAP. Modifications for each correction set are performed by UPDATE in the order in which the sets were introduced. Provided that no correction is dependent on another, the order in which they are specified is irrelevant. If a dependent relationship exists, however, the order is crucial.

For example, if a numbered insertion appears in a correction set which is subsequently deleted, the insertion card is present but inactive. If, however, the deletion occurs first, by specifying a range of cards and then indicating the point at which the insertion is to be made, the numbered insertion will be active.

An INSERT statement can be used to verify that an earlier correction is present on the file. In such a case, the insertion will be empty.

6.6 FILES PROCESSED BY UPDATE

Several files are processed by UPDATE, some of which may be assigned to tape or positioned at the conclusion of the updating process.

FILES PROCESSED BY UPDATE

File	Control Card Identification	Function of File	Type of File	Status After Update Call
Input	I	Provides control information	Coded	Unpositioned. Left at end of record terminating update control card stream†
Output	L	Produces listings	Coded	Appended to current position of output file
Compile	C	Produces images for assembly or compilation	Coded	Rewound before and after UPDATE operation
Old Program Library	P	Contains old program library	Binary Coded	Rewound before UPDATE operation; left at end-of-record terminating program library
New Program Library	N	Contains new program library	Binary Coded	

FILES USED BY UPDATE IN PROCESSING

File Name	Functions	Comments
CMPSCR	Scratch file. During library creation, holds entire symbolic source. During correction runs, holds programs as they are updated. If decks are to be written on the COMPILE file, COMPSCR is used to copy the decks to the compile file.††	Binary file. Rewound prior to and after updating. Not used if the full assembly mode is selected although it is address by UPDATE.
COMDKS	Used by UPDATE internally	Must be disk files. Files are evicted before and after the updating operation.
FTEXT	Used by UPDATE internally	

Files mentioned in *READ, *REWIND and *SKIP operations are left as defined by the latest directive encountered on the input stream.

---

† If UPDATE aborts, the input file is at an unpredictable location.

†† To avoid disk conflicts, tape assignment is often desirable if large volume files are involved.

6.7 UPDATE EXAMPLES

(1) Generating a program library by calling UPDATE:

File A contains:

```
            CARDA1
            CARDA2
*COMDECK    SET2
            CARDA3
            CARDA4
```

File B contains:

```
            *LABEL    EXAMPLE
            *DECK,SET1
                      CARDB1
                      CARDB2
            *READ,A
                      CARDB3
            *END
            *DECK SET3
            *WEOR
                      CARDB4
```

the call UPDATE(I=B,N) results in the library EXAMPLE with the identifiers (deck names):

```
            SET1   SET2   SET3
```

Active cards in EXAMPLE are:

```
*DECK,SET1                SET1      00001
            CARDB1        SET1      00002
            CARDB2        SET1      00003
            CARDA1        SET1      00004
            CARDA2        SET1      00005
*COMDECK SET2             SET2      00001
            CARDA3        SET2      00002
            CARDA4        SET2      00003
            CARDB3        SET2      00004
*DECK SET3                SET3      00001
*WEOR                     SET3      00002
            CARDB4        SET3      00003
```

(2) Creating a new edition of an established library.

Apply the following correction set to EXAMPLE by the call UPDATE(N):

```
*IDENT, CORR1
*INSERT, SET3.1
*CALL, SET2
          CORR11
*DELETE, SET1.3
*DECK, SET1A
          CORR12
*IDENT, CORR2
*DELETE, SET2.2, SET2.3
*IDENT, CORR3
*DELETE, SET1.2
*INSERT, SET1.3
          CORR31
          CORR32
```

COMPILE file contains decks:

SET1 SET1A SET3

With contents:

```
CORR31      CORR3     00001
CORR32      CORR3     00002
CORR12      CORR1     00002
CARDA1      SET1      00004
CARDA2      SET1      00005
CARDB3      SET2      00004
CORR11      CORR1     00004
CARDB4      SET3      00003
```

with an EOR after CORR11.

Identifiers for this edition of EXAMPLE:

SET1 SET2 SET3 CORR1 CORR2 CORR3

Active cards in the new edition of EXAMPLE:

```
*DECK,SET1                SET1    00001
          CORR31          CORR3   00001
          CORR32          CORR3   00002
*DECK,SET1A               CORR1   00001
          CORR12          CORR1   00002
          CARDA1          SET1    00004
          CARDA2          SET1    00005
*COMDECK SET2             SET2    00001
          CARDB3          SET2    00004
*DECK,SET3                SET3    00001
*CALL,SET2                CORR1   00003
          CORR11          CORR1   00004
*WEOR                     SET3    00002
          CARDB4          SET3    00003
```

(3) Modifying a corrected library by calling UPDATE

Apply the following correction set to the edition of the EXAMPLE tape produced by the call UPDATE(N):

```
*IDENT,CORR4
*RESTORE,SET1.3,SET2.2
          CORR41
YANK,CORR3
```

COMPILE file contains the decks:

```
SET1   SET3   (SET1A is not present as it was not altered.
               SET3 is present as SET2 was altered.)
```

With contents:

```
CARDB1          SET1    00002
CARDB2          SET1    00003
CARDA3          SET2    00002
CARDB3          SET2    00004
CORR11          CORR1   00004
CARDB4          SET3    00003
```

with an EOR after CORR11.

Identifiers for this edition of EXAMPLE:

```
SET1   SET2   SET3   CORR1   CORR2   CORR3   CORR4
```

Active cards in new edition of EXAMPLE:

```
*DECK,SET1                SET1      00001
          CARDB1          SET1      00002
          CARDB2          SET1      00003
*DECK,SET1A               CORR1     00001
          CORR12          CORR1     00002
          CARDA1          SET1      00004
          CARDA2          SET1      00005
*COMDECK SET2             SET2      00001
          CARDA3          SET2      00002
          CARDB3          SET2      00004
*DECK,SET3                SET3      00001
*CALL,SET2                CORR1     00003
          CORR11          CORR1     00004
*WEOR                     SET3      00002
          CARDB4          SET3      00003
```

(4) Modifying routines by calling UPDATE

```
JOB,CM50000,T60000.
REQUEST OLDPL.                OLD PROGRAM LIBRARY
UPDATE.                       UPDATE 2TS AND COMPASS
COMPASS (I=COMPILE,S=SCPTEXT,L=0)
7
8
9
*IDENT,SC17
*DELETE,2TS.454,2TS.457
EXI1  LDD    D.CPAD
      ADN    W.CPERT
      CRD    CM
      LDN    P.ZERO
*IDENT,CP21
*DELETE,COMPASS.1535,COMPASS1538
      BX6    -X1*X0
      IX2    X6-X0
      ZR     X2,SCLIST
      SX7    B1
6
7
8
9
```

Corrective code for SC17

Corrective code for CP21

(5) Constructing a new program library from the old program library and adding a new routine, a new common deck, and a new SYSTEXT deck by calling UPDATE

```
JOB
REQUEST    OLDPL.
REQUEST    NEWPL.
UPDATE(N,F)
7
8
9
*IDENT,AAA          AAA=same name as deck to be added.
*INSERT,BBB.nnn     BBB.nnn=identifier of card to precede added deck
                    (last card of deck to precede new deck)
*COMDECK,AAA
    text cards for common deck AAA
*IDENT,DDD          DDD=name of the new system text.
*INSERT,EEE.nnn     EEE.nnn deck=last card of deck to precede DDD
*DECK,DDD           deck.
*WEOR
DDD  Name card necessary for EDITLIB text cards for system text
*IDENT,FFF          FFF=name of new routine
*INSERT,GGG.nnn
*DECK,FFF
    text cards for routine FFF
7
8
9
6
7
8
9
```

6.8 UPDATE MESSAGES

Listing Messages

***ADDFILE INVALID FROM *READ FILE

***ADDFILE INVALID WITH Q-OPTION

***DECK NAMES SEPARATED BY PERIOD IN WRONG ORDER

***DUPLICATE DECK NAME

***DUPLICATE IDENT NAME

***FIRST CARD MUST BE *DECK OR *COMDECK

***INVALID NUMERIC FIELD

***NO SUCH COMMON DECK

***NOT ALL MODS WERE PROCESSED

***ONE OF THE ABOVE DECKS DOES NOT EXIST

***PREMATURE END OF RECORD ON OLD PROGRAM LIBRARY

***RESERVED FILE NAME

***TERMINAL LIMIT EXCEEDS FIRST LIMIT ON DELETE OR RESTORE CARD

***UNKNOWN IDENTIFIER

Display Messages

CORE OVERFLOW

DECK STRUCTURE CHANGED

ONE OR MORE OVERLAPPING CORRECTIONS

UPDATE ERRORS, JOB ABORTED

UPDATING

UPDATING FINISHED

IMPROPER UPDATE PARAMETER, UPDATE ABORTED

The EDITSYM program enables the user to organize symbolic information into program libraries and to make symbolic corrections or alterations to existing program libraries. Data in a program library may be source cards for a compiler or assembler run, data cards, line images for a document, or any other symbolic information desired. Once the symbolic material has been put into the program library format, EDITSYM provides a two-level editing capability.

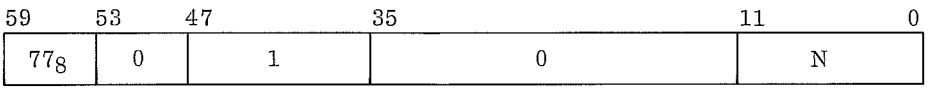
Primary edit operations result in permanent alterations to the program library; secondary edit operations allow the user to keep track of changes. All primary level editing operations result in physical rearrangement of the program library and resequencing of the primary sequence numbers. Secondary level operations do not result in resequencing; they do not alter the arrangement of cards with primary sequence numbers. Primary and secondary levels are associated with two sets of decimal sequence numbers. Primary sequence numbers are decimal integers 1-n. Secondary sequence numbers are decimal numbers in the form j.k where j is the primary sequence number of the preceding card and k is a secondary sequence number.

For example, in a program library containing a deck of cards numbered one to five, a primary level edit operation would be used to delete card three. The new file would not contain card three, and the sequence numbers would be changed to one through four. If, however, a secondary level edit is performed to delete card three, the card image would still exist on the new file but it would be marked as cancelled. Similarly, new cards added in a primary edit would be inserted where requested and given appropriate primary sequence numbers. If new cards were added in a secondary edit, for instance, after card three, they would be assigned the numbers, 3.1, 3.2, 3.3, etc., and the primary numbers would remain as they were.

7.1 PROGRAM LIBRARY FORMAT

A program library may consist of two sections: common and text, but it need not contain both. Each section may contain one or more logical records. In both sections the first two words of a logical record, the prefix, contain deck name identification information. The format of the prefix is as follows:

Word 1



N 2-digit display code number

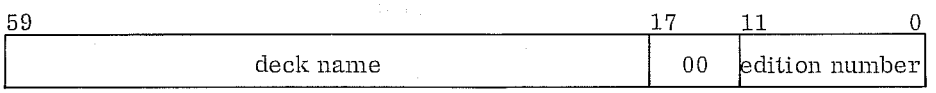
= 02 common section

= 03 text sections

.
.
.

= 99

Word 2 of common and text sections



The deck name may not exceed seven characters. The edition number is a display coded integer which is increased by one each time a new program library is requested.

7.1.1 COMMON DECKS

The first section of a program library contains the common decks. Each common deck consists of one logical record with a two-word prefix. The remainder of the record contains the packed images of the deck. The last card image is *END. If there are no common decks, there is no common section.

7.1.2 TEXT DECKS

The second section of a program library contains an arbitrary number of text decks. Each consists of one logical record identical in format to a common section record, except that in the first prefix word N is equal to or greater than 3.

7.1.3 COMPRESSED DECK FORM

Both common and text decks exist in the program library in compressed form with blank characters removed. Blanks are replaced by the character 55 followed by a 6-bit count. Thus, the character pair 5500 represents one blank, 5501 represents two blanks, etc. If more than 64 blanks are to be represented, two character pairs are needed. 55775502 as a consecutive character string represents 67 blanks. The end of a card is recorded by a 00 character followed by a 00, 01, or 02 depending upon the editing status of the card.

00	indicates card to be used.
01	indicates card logically deleted by secondary editing.
02	indicates card added by secondary editing.

7.2 COMPILE OUTPUT

The main function of EDITSYM is to produce, from selected portions of a program library, a file of information in a format that can be processed by FORTRAN, COMPASS, or other processor. When a compile file is requested, the procedure is as follows:

Card marked as cancelled by secondary editing is not written on the compile file.

EDITSYM control cards are not written on the compile file, with one exception. Control card *CALL,dn, when encountered within the text of a deck on the program library, is retained as a comment card. The common deck named dn is found and written immediately after the *CALL card.

Text decks are written into the compile file as a single logical record until or unless a *WEOR card appears in the EDITSYM control card deck. Cards are represented by 90 display code characters, terminated by a zero byte (end-of-line), as follows:

Columns	
1-72	Supplied on the source
73-79	Deck name
80-84	Primary sequence number
85	Period for a secondary text card
86-90	Secondary sequence number

7.3 CONTROL CARDS

7.3.1 EDITSYM CALL CARD

The call for EDITSYM is as follows:

```
EDITSYM(I=input file, C=compile file, L=list file,
        OPL=old program library, NPL=new program library)
```

Parameters may appear in any order.

Input	absent	corrections on INPUT
	I	corrections on INPUT
	INPUT	corrections on INPUT
	I=lfn	corrections on lfn
	INPUT=lfn	corrections on lfn
Compile	absent	no compile output
	C	compile output on COMPILE
	COMPILE	compile output on COMPILE
	C=0	no compile output
	COMPILE=0	no compile output
	C=lfn	compile output on lfn
	COMPILE=lfn	compile output on lfn
List	absent	no list
	L	list on OUTPUT
	LIST=L	list on OUTPUT
	LIST	list on OUTPUT
	L=0	no list
	LIST=0	no list
	L=lfn	list on lfn
	LIST=lfn	list on lfn
Old Program Library	absent	no old program library
	OPL=0	no old program library
	OPL	old program library on OPL
	OPL=lfn	old program library on lfn
New Program Library	absent	no new program library
	NPL=0	no new program library
	NPL	new program library on NPL
	NPL=lfn	new program library on lfn

The following cards control the addition and deletion of entire decks and local corrections within decks. All control cards referencing common decks must precede any control cards referencing text decks. This is necessary because all of common must exist in its final form before any text processing is done to insure correct processing of common references within text decks.

7.3.2 NEW DECKS

New decks may be introduced by placing

```
*COMDECK,dn
```

or

```
*DECK,dn,n
```

in front of the deck, and

```
*END
```

at the end of the deck.

The deck name to appear in the prefix is dn. All cards introduced in this way are considered primary cards and are given a 00 editing status terminator.

*COMDECK card specifies the introduction of a common deck.

If common decks are to be introduced, *COMDECK cards must precede any control cards which introduce or reference text decks.

*DECK identifies the subsequent cards as a deck belonging to the text section of the new program library.

The n parameter specifies the value of N to be used in the prefix. n may be 3 to 99; the value 3 is assumed if n is absent. Values 4-99 may be used as special flags for other routines which process program libraries.

7.3.3 DECK SEQUENCE CONTROL

```
*COPY,dn1,dn2
```

When this card is encountered, EDITSYM copies the entire text decks from the old program library to the new one and/or to the compile file. Copying begins at deckname dn_1 and continues up to and including the deck name dn_2. If dn_2 is absent, only dn_1 is copied. If dn_1 is * copying begins at the

present position and continues through dn_2. If dn_2 is * copying begins at dn_1 and continues through the end of the program library.

*WEOR

The WEOR card causes EDITSYM to terminate the logical record being written on the compile file.

*CATALOG,lfn

Common and text deck names from the program library, lfn, are listed on OUTPUT.

*COMPILE, lfn

A compile file will be written on lfn. A *COMPILE card overrides the compile parameter on the EDITSYM call card; the *COMPILE card applies only to the deck specified on the following *EDIT, *DECK, or *COPY card. Once a *COMPILE card has been encountered, compile files for any remaining text decks must be requested by a *COMPILE card.

7.3.4 EDIT CONTROL

The user can make corrections to a program library deck with edit control cards. Primary and secondary numbers specify cards in the deck to be altered or after which new cards are to be entered. Sequence numbers are not contiguous from one program library deck to another; therefore, the name of the deck must also be specified.

*EDIT,dn

dn is the name of the deck to be edited. This card must terminate the set of edit control cards which modify the deck dn.

Primary Level Edit Control

*INSERT,n

Corrections are inserted following card n. The corrections terminate with the EDIT control card. All text introduced is considered primary text and thus will cause resequencing if a new program library is requested. n must be an integer.

*DELETE,m,n

Cards m through n, inclusive, are deleted. If n is omitted, only card m is deleted. Source cards may follow the *DELETE control card and are inserted following the last deleted card. The cards deleted are removed from the new program library. Any cards inserted are primary corrections. n and m must be integers.

*RESTORE,m,n

This card restores to its original state a portion (m through n inclusive) of a deck altered by secondary editing. All primary text cards cancelled as a result of a secondary editing operation (all cards with a 01 editing status terminator), are restored as normal primary text cards. All added secondary text cards appearing within this range are removed.

Secondary Level Edit Control

*CANCEL,m,n

Either m or n may be of form j.k where j is a primary number, and k is a secondary sequence number. This card will cause cancellation of all cards from m to n inclusive. Primary cards are not removed; they are marked as cancelled; however, secondary cards are removed. Source cards may follow the *CANCEL control card and are inserted following the last cancelled card. The insertions are marked as secondary text. Cancellation does not cause resequencing of the primary cards when a new program library is requested.

*ADD,n

n may be of the form j.k as defined above. The ensuing cards are inserted as secondary text. Addition does not resequence primary cards when a new program library is requested.

7.4 EDITSYM EXAMPLES

Create a program library.

```
.
.
.
REQUEST NPL.
REWIND (NPL)
EDITSYM (NPL, L)
.
.
.
7
8
9
*COMDECK, MACROS
.
.
.                                          macro definitions
*END
*COMDECK, DIMENS
.
.
.                                          dimension statements
*END
*DECK, PROGA
.                                          COMPASS subprogram
.                                          containing *CALL, MACROS
*END
*DECK, PROGB
.                                          COMPASS subprogram
.                                          containing *CALL, MACROS
*END
*DECK, FTNPROG
.                                          FORTRAN program con-
.                                          taining *CALL, DIMENS
*END
*DECK, FTNSUBR
.                                          FORTRAN subroutine
.                                          containing *CALL, DIMENS
*END
7
8
9
.
.
.
6
7
8
9
```

Modify and assemble PROGB from the program library created in the preceding example.

```
:
REQUEST OPL.
REWIND (OPL)
EDITSYM (OPL, C)
COMPASS (I=COMPILE, B=PUNCHB)
:
7
8
9
*INSERT, 300
:
*DELETE, 2, 3
:
*EDIT, PROGB
7
8
9
:
6
7
8
9
```

Modify and assemble/compile the program library in the preceding examples; also add two decks and create a new program library. Catalogue the new program library.

```
:
REQUEST, OPL.
REQUEST, NPL.
REWIND, NPL.
REWIND, OPL.
EDITSYM (NPL, OPL)
COMPASS (I=ASSEM)
RUN(S, , , COMPL)
:
7
8
9
*COMPILE, COMPL
    Correction deck
```

```
*EDIT, FTNPROG
*COMPILE, COMPL
   Correction deck
*EDIT, FTNSUBR
*COMPILE, ASSEM
   Correction deck
*EDIT, PROGA
*COMPILE, ASSEM
   Correction deck
*EDIT, PROGB
*DECK, PROGC
   Source deck
*END
*DECK, PROGD
   Source deck
*END
*CATALOG, NPL
7
8
9
:
:
:
6
7
8
9
```

CHECKPOINT/RESTART 8

The programmer may take checkpoint dumps during job execution. When a checkpoint dump is taken at some point in the program, a file is written containing all information needed to restart the job at that point. This procedure allows the programmer to restart his job from the point of the last checkpoint dump rather than from the beginning of the job in case of machine malfunction or operator error. The checkpoint file is made available to the programmer only in event of abnormal job termination.

8.1 CHECKPOINT REQUEST

Every central processor program which uses the CHECKPT function must provide a REQUEST control card or function to identify the name and storage device of the checkpoint file as follows:

REQUEST, cname, dt, CK, n

cname	Unique name for the checkpoint file
dt	MT, LO, HI, HY, WT, Dxxxx
CK	Disposition code identifying cname as the checkpoint file
n	File declaration code indicating checkpoint on 1/2" magnetic tape should be labeled and is initially an output tape. This parameter should be included if the checkpoint tape is to be labeled. Such a tape will have cname as the file label name. The creation date field in the label will be the current date as stored within CM resident. The retention cycle is an installation option.

CHECKPT function requests are honored only if the checkpoint file is defined by a REQUEST. Position information is saved for all files. File contents are saved if the file name is given in the checkpoint function parameter list.

CHECKPOINT PROCEDURE

Checkpoint is an unconditional request which results in a checkpoint dump when the request is encountered. The checkpoint program (CKP) is loaded and a dayfile message is written acknowledging the request. A delay allows all input/output requests in progress to be completed. The FNT/FST is scanned for the checkpoint file, cname. If a checkpoint file is not found, CKP will issue a dayfile message to indicate an invalid function and then release its PP, allowing the job to continue execution.

If the checkpoint file is located, an identification record is constructed and written. This record contains information from all central memory resident tables associated with the job.

A second record is written containing the central memory area assigned to the job (defined by RA and fl). Record groups are written, one group of logical records for each file named in the checkpoint function parameter list. Each group contains the contents of a logical file specified in the parameter list. Also each group of records is terminated by a zero length PRU of level 16. Level 16 should not be used in any files, designated in the checkpoint parameter list, to be copied. The checkpoint file is rewound, the central processor is requested for the job, and the PP containing program CKP is released, allowing the job to continue execution.

Upon abnormal job termination, all uncopied associated local disk files are placed on locked status to protect the disk records from being overwritten.

8.2 RESTART REQUEST

A job may be restarted from a checkpoint dump by the RESTART control card:

```
RESTART, cname
```

cname is the unique file name included on the REQUEST card for the run in which the checkpoint request was made.

When the job is restarted, file disposition is the same as when the checkpoint was taken. Thus, for example, a file with CK disposition in the checkpoint run will still have CK disposition for the restart run.

After reading the checkpoint file, the restart program leaves it as a local, unlocked file, so that it will be replaced if a checkpoint is taken after the job is restarted. Some consequences are:

If a restarted job writes no new checkpoint files and terminates normally, the old checkpoint file is released.

If a restarted job writes no new checkpoint files and terminates abnormally, the old checkpoint file is retained.

If a restarted job writes new checkpoint files and terminates normally, the old checkpoint file is lost because the current one was written over it, and the current checkpoint file is released.

If a restarted job writes new checkpoint files and terminates abnormally, the current checkpoint file is retained and, since it was written over the old one, the old checkpoint file is lost.

When a restarted job terminates abnormally, at least three possibilities must be considered:

The program may have an error that caused both the original and the restarted jobs to terminate abnormally. Then, generally, nothing is gained by another restart.

The checkpoint dump may not be restartable.

A machine error may have caused abnormal termination of the restarted job.

If the decision is made not to restart from a checkpoint file, the user must release all locked disk files, by submitting a RESTART control card with the following format:

```
RESTART, cname, CLEAR,*
```

The FNT/FST is searched for the checkpoint file. If this file is not found, a dayfile message is issued and the job is terminated. If the file resides on magnetic tape, the operator is requested to provide the checkpoint file as tape input. The checkpoint records are read and used to restore the control point area and to restore and reposition files. For each tape in use by the program when the checkpoint dump was taken, a message is issued to the operator to mount the tape. The central memory field length (FL) is obtained from the control point area and used in the MTR request storage function to obtain FL words of central memory. The central processor program is read into this area. A dayfile message is written to indicate

that the restart is completed. The MTR function request for central processor is issued and the PP containing the restart program is released.

8.3 UNRESTARTABLE CHECKPOINT DUMPS

A checkpoint dump may not be restartable in the following cases:

A file necessary for restarting the program was overwritten after the checkpoint dump was taken.

A checkpoint file was on a disk involved in a subsequent machine failure.

A machine error propagated bad results but did not cause abnormal termination until after another checkpoint dump.

Disk files necessary for restarting the program were closed after the checkpoint dump was taken. This action releases the allocation for these records, allowing overwrite.

The first and last cases may be guarded against by listing all such files as parameters to the CKP function request. In addition, CKP functions may be issued after each file is closed.

8.4 ROLL-OUT/ ROLL-IN

The operator may request that a job which has control of the central processor at a specified control point be rolled out so that all of the memory assigned to the job except its control point area is released. Neither the control point nor the equipment assigned to the job will be relinquished. Later, the operator can request that the rolled out job be restored and its execution resumed.

9.1 PROCESSING MODES

After deadstart loading is completed, no activity will be displayed at control points 1 to 7. A keyboard message at the console is needed to initiate the mode of job processing. The SCOPE operating system provides two modes of operation: automatic and manual. The operator may override the automatic mode and call individual programs to each control point.

9.1.1 AUTOMATIC MODE

After deadstart loading, the system awaits input from the operator. A keyboard entry AUTO. selects the mode for automatic processing. Under this mode, input from the card reader is assumed. The read package is assigned to the first available PP (assigned to control point 1), and then assigns output to the next available PP (assigned to control point 2). The read package reads, converts the information to display code, and forms input files on the disk until cards are depleted.

The NEXT package is assigned to all available control points, other than 7. Input files are executed according to the priority designated on the job card. As each job produces output, the records are packed and sent to the disk. When a job is completed, output is disposed of by the output package in the order of job priorities.

In normal operation, job processing is initiated with the type-in AUTO., which brings the following activities to control points.

Control Point Number	Job Name	Activity
1	READ	Load jobs from card reader and store on disk.
2	OUTPUT	Transfer job output from disk to printer or punch.
3	NEXT	Search FNT for a job to process at this control point.
4	NEXT	
5	NEXT	
6	NEXT	
7	---	No activity.

The jobs brought to control points are system jobs which process various phases of object jobs.

Example:

AUTO. is equivalent to keying:

1. READ.
2. OUTPUT.
3. NEXT.
4. NEXT.
5. NEXT.
6. NEXT.

Activities may also be dropped.

Example:

2. DROP. Printing ceases and output accumulates on the disk until output or dump is introduced.

AUTO. operation leaves control point 7 free, but an activity may be initiated at that point.

Example:

7. NEXT. Enables control point 7 for object jobs.
7. LOAD. Brings to control point 7 a package for loading jobs from magnetic tape to the disk.

9.1.2 MANUAL MODE

Manual operation is similar to automatic except that each operation must be requested by the operator. For example, after the deadstart load has been executed, the operator may enter on the keyboard the control point number and the read statement, n. READ. This will load the read package in the next available PP and assign it to the designated control point, n.

If an input job stack is prepared on magnetic tape, the operator enters on the keyboard the control point number and the load statement, n. LOAD; the load package is loaded into the next available PP; the package requests the tape unit number and continues.

To execute, the operator types n.NEXT. for each control point to be assigned a job. As each job is executed, output files are entered on the disk. No output processing results until the operator enters the control point number and the output statement, n.OUTPUT.

An alternate method is to enter n.DUMP. on the keyboard; the accumulated output files are dumped on magnetic tape in the order of job priorities for printing off-line. The operator may enter n.DROP. on the keyboard, which drops the job at the designated control point. After all input is read, the read or load routine may be dropped to allow another job to use the control point. This operation may be used in both automatic and manual mode.

9.2 CONSOLE AND DISPLAY SCOPES

The operator and the system communicate through the console keyboard entries and two or more console display scopes. The operator may introduce jobs, change priorities, and so forth. SCOPE also allows the operator to examine selected portions of memory and keeps a permanent record of job history which can be called at any time.

The system communicates with the operator through the two console display scopes. Data is assembled and disassembled by individual routines in SCOPE. At the request of the operator, all portions of job or system status may be displayed. Data entered at the console keyboard is also displayed on one of the scopes. A permanent record of system/console communication is retained in the dayfile and ultimately printed at operator request.

The operator communicates with the system through the console keyboard. As they are being typed, keyboard type-in messages are shown in the lower left-hand corner of the left display scope. The operator may check entries prior to initiating execution.

Each keyboard-initiated command to DSD or DIS is a single line. Backspacing causes the last character keyed to be blanked. Each command should end with a period followed by a carriage return, at which point the message is interpreted. If the command is acceptable, it is acted upon and the line on the screen is cleared; if not acceptable, an error message appears above the erroneous line. A blank type-in can be used to clear an entire command. These messages or diagnostics can appear above the line of keyboard input:

FORMAT ERROR	Keyboard type-in contains a formatting error.
ILLEGAL ENTRY	Keyboard input is unrecognizable.
WAIT MTR FCN xx	System is in step mode (or MTR is busy) and DSD is waiting for MTR when processing keyboard input. xx is the number of the monitor function.

9.2.1 SYSTEM DISPLAY (DSD)

System Display (DSD)

The system display program (DSD) is permanently located in PP 9. DSD maintains a display on the two screens of the main console for all currently running jobs. The keyboard is used to initiate and control equipment assignment and job progress. During normal job processing, operators need not be concerned with the data or program storage displays. However, the displays give an accurate picture of system progress, and can be used for on-line debugging.

The console screens may be assigned any combination of two displays listed below.

Codes	Display	
A	Dayfile	
B	Job status	
C	Data storage	
D	Data storage	
E	Data storage	
F	Program storage	
G	Program storage	
H	Job backlog	
[I	Equipment status	SCOPE 3 only]

DAYFILE (A) DISPLAY

A file named DAYFILE is maintained on disk; it contains the job history for all jobs. The records called dayfile messages are added to DAYFILE when whenever:

A control card statement is executed.

System action with respect to a job is not in direct response to a control card statement.

An error is recognized.

A comment is entered either by control card or by the MESSAGE function.

End of job is encountered; two dayfile messages record the amounts of CP and PP time charged to the job. CP time includes compilation time as well as execution time.

The job history consists of all dayfile messages for a given job. When a job terminates, its dayfile entries are printed at the end of its listing. The total dayfile, which normally includes entries for different jobs intermixed since the jobs share the computer, is available to the system as a record of all action taken since deadstart. Usually the most recent 30 (approximately) entries in this total dayfile are displayed on a console screen for the operator.

Dayfile messages on the A display have the format: hh. mm. ss. jobname. message.

The time given in hours, minutes, and seconds is the actual time since deadstart or the real time if this was entered into the system by a TIME command to DSD. The time is followed by the name of the job and the message itself.

As a job is processed, dayfile messages are sent to the dayfile by PP programs or by central programs of the job. Each control card, including the job card, is listed at the time of execution. The dayfile messages may be inspected as follows:

On a console screen (display A). The file is moved up the scope screen as messages are generated.

At the end of job's printed output. All dayfile messages associated with that job name are printed.

The dayfile is preserved on disk storage. The entire contents of the dayfile can be accessed for logging purposes.

DAYFILE DUMPS

The operator may request that the dayfile be dumped and the disk space released. The operator initiates this action by freeing a control point and typing n. DAYFILE, xx. n is the control point number and xx is an equipment type: LP, CP, or MT.

JOB STATUS (B) DISPLAY

For the Job Status (B) display, DSD provides the status of all control points, each in the format:

```
n.jobname priority, time limit, running time s-------
      RA, FL, equipment numbers.
last dayfile message or message to operator
```

n is the control point number, s the central processor status of the job. If PP's are running at the control points, their numbers are entered in the first line. All other numbers of the display are in octal.

Example:

```
3.AJOB   5037,   100,   37 X-2----7-
      30000,40000,51,72.
RUNS
```

In this display, the job has used 37_8 seconds of central processor time so far, and is not using the central processor at present while PP's 2 and 7 are working.

The OUTPUT control point display is as follows:

```
2.OUTPUT  ,  ,  ,  ,  -2---67-
15100,6100,20.21.22.23.24.
1.PRINT 2.IDLE 3.PUNCH 4.PRINT 5.PRINT 6.PRINT
JOBNAME        JOB001   JOB002  JOB003  JOB004
```

On the third line each number refers to a buffer point. Buffer points are assigned as needed, and the number of output devices in the equipment status table determines the number of buffer points set up. Because the assignment is dynamic, each is labeled with the operation in process.

STORAGE (C-G) DISPLAYS

A storage display of type C through G has 4 groups of central memory words, numbered 0 to 3. Each group consists of 8 central memory words, each displayed in octal on a separate line, preceded by its address. The address is absolute for DSD, relative for DIS. For data display (C, D, E) words are divided into 5 groups of 4 digits; for program display (F, G), words are divided into 4 groups of 5 digits.

The address of an 8-word group follows the display type and group number in a command:

D2,1020. Set words 001020-001027 in group 2 of D.

Group number 4 is used to set all four groups at once, to display 32 consecutive words:

D4,1020. Set words

	Group
001020-001027	0
001030-001037	1
001040-001047	2
001050-001057	3

JOB BACKLOG (H) DISPLAY

The H display under DSD shows a list of input, output, locked, and common files with the name and priority of each file. Input files are jobs waiting on disk; output files are the output of completed jobs awaiting printing or punching; locked files are those associated with a checkpointed job which terminated abnormally; common files are those which may be passed from one job to another. The progress of any job not at a control point can be seen by using the H display.

H,C.	Display Common Files
H,I.	Display Input Files
H,L.	Display Locked Files
H,O.	Display Output Files
H,P.	Display Punch Files

SCOPE 3 only

EQUIPMENT STATUS (I) DISPLAY

The I display shows a list of defined system equipments and indicates the status of each. There are four entries per line; each entry has the form: ee-hhuu sss.

ee equipment status table ordinal used by the operator in keyboard entries which affect equipment status.

hh hardware mnemonic code. Only the following equipments are displayed:

SCOPE 3 only

AA	6603 Disk File
AB	6638 Disk File
AC	6603 II Disk File (option 10124 installed)
AD	3637B/865 Drum
AE	3637B/863 Drum
AP	3234/854 Disk Pack
CP	Card Punch
CR	Card Reader
DS	Console Display
LP	Line Printer
MT	1/2 inch magnetic tape
WT	1 inch magnetic tape

uu physical unit address

sss status mnemonic. The following mutually exclusive conditions are displayed:

App	Equipment is assigned to control point pp.
SYS	Equipment was assigned to the system at deadstart.
PVT	Equipment was assigned for private allocation but is inactive.
PBS	Equipment was assigned for private allocation, has an active file allocated, but is not currently assigned to a control point.
UNL	Equipment is logically unloaded to the system, but is not "off" to the system.
BSY	Equipment has active files allocated on it; it may be "off" to the system but may not be made physically not ready.
AVL	Equipment is not off, is not assigned to a control point, and no active files are allocated on it.
OFF	Equipment is "off" to the system; it is not assigned to a control point; no active files are allocated on it; the device may be physically not ready.

SYSTEM DISPLAY KEYBOARD ENTRIES

The keyboard, during the display of the overall system status, is used to initiate and control equipment assignment and job progress. The following table describes the keyboard codes and formats.

Type-In	Action Initiated
AUTO.	Used after dead-start to initiate automatic job processing with card input and printer output.
STEP.	Selects a step mode for the SCOPE monitor in PP0. Requests from other PP's are processed when the keyboard space bar is pressed. High-speed operation may be resumed by entering a period.
OFFxx.	Indicates to the system that equipment number xx must not be used, for example, during maintenance.
ONxx.	Returns equipment xx to the pool of available equipment.
n.STEP	Selects a STEP mode for a specific control point.
TIME.hh.mm.ss.	Resets the current time in hours, minutes, seconds for 24-hour clock. Only eight characters may be entered.
ONCPUx.	Turns on CPUx provided it is turned off or delegated but not if it is not locked off.
OFFCPUx.	Turns off CPUx if it is on.
MCH.nn.	Master clears channel nn.
DCNnn.	Disconnects channel nn.
FCNnn.	Enters a zero function on channel nn.
ENPR,xxxx,f,y.	y is I or O indicating input or output file. If y is absent, I is assumed. Changes the priority of file f to xxxx.
x,y.	Changes contents of word at x (absolute) to y. Leading zeros may be dropped in both address (x) and data (y).
n.DROP.	Drops the job at control point n.
n.DIS.	Assigns DIS package to control point n. If there is no other console, DSD may relinquish the main console.
n.ASSIGNuu.	Assigns equipment uu to control point n.
n.READ.	Brings READ package to control point n for loading jobs from cards.
n.OUTPUT.	Brings OUTPUT package to control point n; normally initiated by AUTO, but may be brought in separately.

Type-In	Action Initiated
n.LOAD.	Brings LOAD package to control point n to load jobs from tape (prepared by SCOPE) to end-of-information.
n.DUMP.	Brings DUMP package to control point n to dump output files on tape.
n.BLANK.	Brings BLANK package to control point n to write a blank label on a tape.
n.LOADx.	Brings LOADx package to control point n to load jobs from an externally prepared tape, n, to end-of-information.
n.NEXT.	Brings job NEXT to control point n to look for a job for the control point.

(READ, OUTPUT, LOAD, LOADx, BLANK, DUMP, and NEXT have no effect if the control point already has a job name.)

(LOADx, BLANK, LOAD, and DUMP request assignment of a tape unit.)

n.USECPUx.	Delegates CPUx to control point n. Only jobs operating at control point n will use CPUx, and they will use only that CPU. Delegation of a CPU is indicated on the B display.
n.RELCPU.	Releases the CPU delegated to the system, making it available to all control points.
n.GO.	Continues job at control point n if it has come to a pause (usually as a result of a FORTRAN pause statement, or if repeated tape transfers after parity failure are not effective).
n.ONSWx.	Sets sense switch x for FORTRAN program at control point n. Settings are preserved at RA and at a word in the control point area.
n.COMMENT.c.	Enters a comment c into dayfile.
n.OFFSWx.	Turns off sense switch x for control point n.
n.ENPR,xxxx.	Changes priority of job at control point n to xxxx.
n.ENTL,xxxx.	Changes time limit of job at control point n to xxxx, the new time limit in octal seconds.
n.ROLLOUT.	Rolls out the job at control point n.
n.ROLLIN.	Rolls in the job at control point n.

Output operations commands to DSD follow. The print file may cause the printer to stop and a message to appear at the fourth line of the control point.

x.LPnn message	x is buffer point number, nn is printer number. After specified action, print operation may be resumed by typing x.GO.

For repeating listings or changing printer forms, pseudo format characters PM cause printer to stop, and up to 30 characters following the format characters will be transmitted to console message area for operator action.

Type-In	Action Initiated
n.ENDm.	Terminates operation at buffer point m.
n.REPEATm.	Repeats current operation at buffer point m.
n.SUPPRESSm.	Suppresses page control of printer at buffer point m.

If m = 7, for END and REPEAT, all buffer points concerned with printers will be affected. After equipment malfunction, operation is restarted by typing: n.REPEATm. n.ENDm.

PRIVATE DISK PACK OPERATION COMMANDS

SCOPE 3 only

A REQUEST card or function is used to obtain access to a private disk pack. The operator must assign available equipment and initiate removal of inactive private packs and mounting packs. The keyboard entries, described below allow the system to protect private pack integrity and to maintain system tables. Both functions must be assigned to an empty control point.

Type-In	Action Initiated
n.UNLOADxx.	Requests assignment of an equipment through MTR to allow the operator to unload a disk pack on equipment xx. The tables are examined. If active files are assigned to this equipment FILES ACTIVE ON xx is typed and no other action occurs. (The equipment status (I) display may also be examined to see if there are active files on the disk pack.) The UNLOAD function must be dropped from the control point to be cleared. If the equipment is not active, it is set in a logically unloaded status to prevent further assignment and the UNLOAD function drops automatically.

SCOPE 3 only

Type-In	Action Initiated
n. DEVADDxx, P, B	Requests system to recognize that the operator has readied a pack on equipment xx. P indicates a private pack; if P is not present, the pack becomes a system allocatable equipment. If the pack is private and B is present, the pack is blank labeled and ready for assignment to a new private file. The pack is considered empty. Before tables relative to the private and blank labeling options are processed, the DEVADD function requests assignment of a disk pack through an MTR function and checks the resulting assignment. If the assigned equipment is already active, the message UNIT xx ALREADY ACTIVE is displayed and the control point is dropped to clear the activity. Otherwise assignment is made, unload status is cleared, and P, B options are processed. A system allocatable device may be redesignated as private by the DEVADD function with P, B options. New packs must be blank labeled.

9.2.2 JOB DISPLAY (DIS)

A job display (DIS), similar to DSD, is used for information more relevant to a single job. Using DIS, the B display can show the exchange jump area of the job; central memory addresses relative to the job's reference addresses are used for data and program displays.

The job display package (DIS) can be called from a control card (DIS) or by a command to DSD at any time during job execution. DIS stops automatic advance of job control cards. This display covers data pertaining to a particular job only. The keyboard is used to advance job control cards and provide any two of the following displays in the same manner as for DSD display. The B display shows only the condition of the control point to which DIS is attached; it includes the next control statement, and a picture of the job's exchange package. The exchange package is displayed only while the job is in W, X, or blank status. The operator may change priorities and suspend job execution with DIS.

Codes

A	Dayfile	
B	Job status	
C	Data storage	5 groups of 4 octal digits per group
D	Data storage	
E	Data storage	
F	Program storage	4 groups of 5 octal digits per group
G	Program storage	

JOB DISPLAY KEYBOARD ENTRIES

ENP,xxxxxx.	Set P = xxxxxx (next instruction address).
ENAi,xxxxxx.	Set Ai = xxxxxx in exchange jump area.
ENBi,xxxxxx.	Set Bi = xxxxxx in exchange jump area.
ENXi,xxxx xxxx xxxx xxxx xxxx. (Spacing is unimportant.)	Set Xi = xxxxxxxxxxxxxxxxxxxx in exchange jump area.
ENFL,x.	Set FL = x_8 in exchange jump area. (Storage will be moved if necessary.)
ENTL,x.	Set central processor time limit = x_8 seconds.
ENPR,x.	Set job priority = x_8.
DCP.	Drop central processor and display exchange jump area (in display B).
RCP.	Request central processor. If the priority of the job is sufficient, execution will begin at the next program address for a job suspended by a DCP request.
BKP,x.	Breakpoint to address x in the program. Central processor execution begins at the current value of P and stops when P = x. DIS clears x to stop the program at that point, and restores the original word when the stop occurs.
RNS.	Read and execute next control statement.
RSS.	Read next control statement and stop prior to execution.
ENEM,x.	Enter exit mode x.
ENS,xxxxxxxx.	Allows the entry of any control statement xxxxxxxx as if it had been entered on a control card. The statement can then be processed using RNS. or RSS.
GO.	Restart a program which has paused.
ONSWx.	Set sense switch x for the job.
OFFSWx.	Turn off sense switch x for the job.

HOLD.	DIS relinquishes the display console, but the job is held at the present status. A console must be reassigned to continue use of DIS.
DROP.	DIS is dropped and normal execution of the job is continued; it does not drop the job.
DMP(x, y)	Dump storage from x to y in the output file (relative to RA).
DMP(4xxxxx, 4yyyyy)	Dump absolute storage from x to y in the output file.
DMP(x)	Dump storage from the job reference address to x.
DMP.	Dump exchange jump area to output file (DMP formats are the same as if used on control cards).
x.y	Change the contents of the word at x (relative to RA) to y. Leading zeros may be dropped in both the address (x) and the data (y).
ASSIGNxx.	Assign equipment xx to the job.

DEADSTART PROCEDURES AND DISPLAYS

Deadstart procedures do not require deadstart cards. The system may be deadstarted from tape to any one of the allocatable devices. A single system tape will deadstart on all allocatable devices.† Upon request the operator types characters indicating the type of deadstart. Defined type-ins are:

D	Deadstart dump
N	Normal preload
P	Pre-address 6603-II device
R	Recovery start
S	Special preload

When the requested deadstart routine has been loaded from the tape, the tape is rewound.

---

†The tapes should be on channels 0, 12_8 or 13_8. If the tapes are not located on those channels, the deadstart panel alone will not be able to rewind the tape. The installation may choose to use deadstart cards to bootstrap the tape rather than deadstarting directly from the panel.

Both console displays are used during deadstart to inform the operator of activity status and to request information from the operator.

DEADSTART DUMPS

When a printer is available, the channel, equipment, and converter number are defined by the installation and assembled into the procedure, unless the E option is used. After the procedure is loaded, the console displays WHAT. The operator may input any one of the three following options:

E,cc,s,e	E overrides a previous assignment by defining: cc = channel s = converter number e = equipment number
Pppp...p	P calls for dump of PP memory, PP number p. p = 0 1...9
Cm,n	C calls for an absolute central memory dump: m = starting address n = terminating address Wrap around is possible so that m may be greater than n.

PRELOAD AND LOAD

Central memory resident tables and system must be preassembled to define installation options and configuration. The allocatable device onto which the system is loaded is defined by this data. When loading is completed, the following message is displayed:

MM/DD/YY

The current date is input and the system is ready to process.

PREADDRESSING

Option 10124 is installed on a 6603 disk; each sector on the disks must be pre-addressed to use this option. Pre-addressing, which takes approximately three minutes, is required only if a nonstandard system was writing on the 6603-II disks. The channel and synchronizer numbers are defined by the following displays:

THE 6603 CHANNEL IS

THE 6603 SYNCHRONIZER IS

The requested number is input after each display and pre-addressing has begun. Progress is monitored by maintaining the following display:

PREADDRESSING 6603

CHAN xx SYNCx

TRACK xxx

The x values are defined and updated. When the process is completed, the third line is changed to display:

COMPLETED

The operator may push the deadstart switch and continue to the next step.

RECOVERY RESTART

This routine is the same as the recovery restart routine under SCOPE 3.1 except that MTR, DSD, and PP resident are reloaded from the system's tape rather than from the allocatable device.

SPECIAL PRELOAD

Deadstart is the same as normal preload and load procedure, but the allocatable device used as systems storage must be specified by display output and operator input as follows:

Display: THE DEVICE TYPE IS

Input one of the following device mnemonics:

AA(6603), AB(6638), AC(6603-II), AD(865 drum), AE(863 drum), or AP(854 disk pack), followed by a carriage return.

For 6603 or 6638:

Display: THE $\begin{matrix}6603\\6638\end{matrix}$ CHANNEL IS

Input: cc (channel, followed by carriage return).

Deadstart can now proceed.

For 865, 863, or 854:

Display: THE $\begin{matrix}865\\863\\854\end{matrix}$ CHANNEL IS

Input: cc (channel, followed by carriage return).

865
Display: THE 863 CONVERTER IS
854

Input: s (6681 converter number, followed by a carriage return).

865
Display: THE 863 EQUIPMENT IS
854

Input: e (3000 equipment controller number, followed by a carriage return).

865
Display: THE 863 UNIT IS
854

Input: uu (unit number, followed by a carriage return).

Deadstart will be ready to proceed.

UTILITY PROGRAMS 10

The SCOPE library contains a set of PP and CP utility programs which can be called by control cards or by keyboard entries.

Card-to-tape, tape-to-tape, tape-to-print, card-to-central storage, and central storage-to-punch operations as well as general file manipulation are possible.. Utility operations can be performed with named files, each of which designates a specific peripheral device, such as a card reader, tape unit, printer, card punch or disk.

Before the first reference to any named file, an equipment should be assigned to it by the operator with the ASSIGN statement or by the programmer with the REQUEST statement; otherwise, the system assigns the file to a disk unit. All files, except disk, specify a unique peripheral equipment and all references to a specific equipment are made through the file name.

Utility jobs conform to the normal deck structure. The job deck contains the following cards:

Job card	first control card
Request cards	equipment assignment
Program cards	data operations
6, 7, 8, 9	end of job

The job card includes name, priority, time limit and field lengths. If only utility programs are to be executed, a short field may be specified. In all copy operations, the central memory buffer is automatically set up to use the entire field length of the job. Some operations between high speed devices may be accelerated with a larger field length.

The operator should be requested to assign equipment to all necessary files which do not reside on the disk. Tapes can be rewound and positioned upon request. Each utility program is called by specifying its name starting in column 1. Parameters for execution of the program appear in parentheses after the name.

Example:

To print the third and fourth coded files from a tape:

TAPEPRT, T520, CM 1000, P6. (Job Card)

Assign unique file names, PRINTER and MAGTAPE, with REQUEST control cards to a printer and tape unit:

REQUEST PRINTER.	(Operator would assign available printer e.g., 3.ASSIGN20.)
REQUEST MAGTAPE.	(Operator would assign specific tape unit e.g., 3.ASSIGN13.)
REWIND(MAGTAPE)	Rewind tape unit to be sure of position.
COPYCF (MAGTAPE,XX,2)	Skip tape to beginning of third file by copying first two files to an unused dummy file XX.
COPYCF (MAGTAPE, PRINTER,2)	Copy the two coded files to the printer.
	An end-of-file card completes the job.

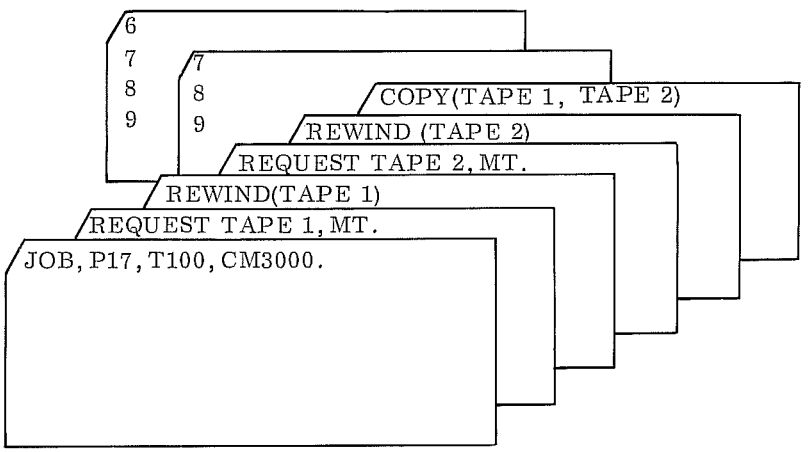
10.1 COPY ROUTINES

COPY TO END-OF-INFORMATION

```
COPY(file 1,file 2)
```

The named file 1 is copied onto file 2 until a double end-of-file or end-of-information is detected on file 1. Both files are then backspaced over the last file mark. If parameters are omitted, INPUT, OUTPUT are assumed.

This routine may be used to copy a tape even if the number of files on the tape is not known. A sample deck structure would be as follows:



COPY BINARY FILE

```
COPYBF(file 1,file 2,n)
```

The number of binary files specified by n (decimal) are copied from file 1 to file2. If the first and second parameters are omitted, INPUT and OUTPUT are assumed. If the third is omitted, one file is copied.

COPY BINARY RECORD

```
COPYBR(file 1,file 2,n)
```

The number of binary records specified by n (decimal) are copied from file 1 to file 2. If the first and second parameters are omitted, INPUT and OUTPUT are assumed. If the third is omitted, one record is copied.

This operation terminates on reading a file mark from file 1 or when the required number of records has been read. A file mark is not written on file 2.

COPY CODED (BCD) FILE

```
COPYCF(file 1,file 2,n)
```

The number of coded (BCD) files specified by n (decimal) are copied from file 1 to file 2. If the first and second parameters are omitted, INPUT and OUTPUT are assumed. If the third is omitted, one file is copied.

COPY CODED RECORD

```
COPYCR(file 1, file 2,n)
```

The number of coded logical records specified by n (decimal) are copied from file 1 to file 2. If the first and second parameters are omitted, INPUT and OUTPUT are assumed. If the third is omitted, one record is copied.

This operation terminates in reading a file mark from file 1 or when the required number of records has been read. A file mark is not written on file 2.

COPY SHIFTED BINARY FILE

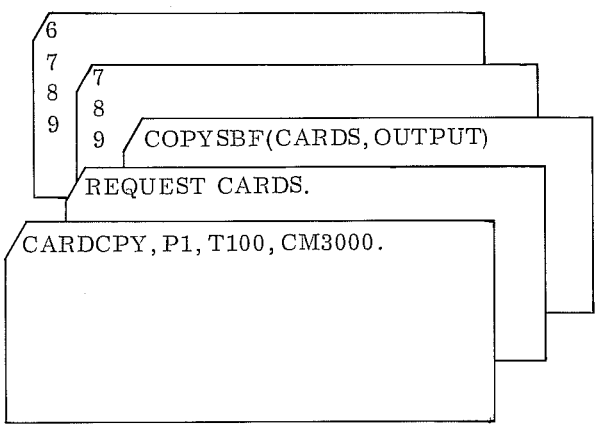
```
COPYSBF(file 1, file 2)
```

A single binary file of coded information is copied from file 1 to file 2, shifting each line one character and adding a leading space. If parameters are omitted, INPUT, OUTPUT are assumed.

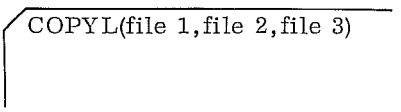
This routine is used in formatting a print file where the first character of each line is not a control character and is to be printed. The space character added will result in single line spacing when the file is printed.

Example:

Control cards to print a Hollerith card file. The Hollerith card file read by the operator-assigned card reader will be printed on OUTPUT file of job CARDCPY.

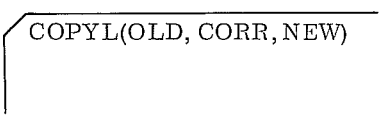


COPYL



This program allows for selective replacement of one or more routines. File 1 contains the old set of decks; file 2 contains the routines to be replaced, and file 3 contains the updated set of routines. Files 1 and 2 are not rewound, and processing continues until the end-of-file on file 1. Routines on file 2 need not be in any order.

Example:



The following job will update 1AJ, 1BJ, and 2TS from a tape (OLD) which presumably contains the binary decks of these and other system routines. CORR will contain the new 1AJ, 1BJ, and 2TS and the new file will be written on a tape file called NEW.

```
JOB, CM60000, T1000.
REQUEST OLDPL.              (UPDATE LIBRARY TAPE)
REWIND(OLDPL)
UPDATE(Q)
COMPASS (I=COMPILE, B=CORR, S=SCPTEXT)
REQUEST OLD.
REWIND(OLD)
REQUEST NEW.
REWIND(NEW)
COPYL(OLD, CORR, NEW)
UNLOAD(NEW)
REWIND(OLD)
7
8
9
```

Example continued on next page.

(Continuation)

*IDENT,TEST

Mods TO 1AJ, 1BJ and 2TS *COMPILE 1AJ, 1BJ, 2TS

7_{8_9}

$6_{7_{8_9}}$

COPYL Messages

Listing Messages: COPYL DONE
COPYL DID NOT FIND xxxxxxx.
ILLEGAL COPYL PARAMETER

Display Message: UPDATING xxxxxxx.

10.2 LOADING ROUTINES

UNLOAD FILE

```
UNLOAD(file 1)
```

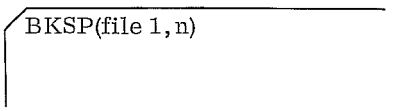
File 1 is rewound and unloaded. This does not release the file assignment to the control point.

REWIND FILE

```
REWIND(file 1)
```

This central program rewinds file 1.

BACKSPACE LOGICAL RECORD



This program allows backspacing of multiple logical records as specified by the decimal n. Backspacing terminates if file becomes rewound.

LOAD BINARY CORRECTIONS

This peripheral program may be called with a control card or from a display console. Binary corrections are read from the INPUT file and entered in central storage. If a parameter is specified in the program call, binary cards are loaded beginning with that address; otherwise, loading begins at the reference address + 100. Only one record is read from the INPUT file. A call must be made for each block of data to be loaded. This program may be called with either of the following formats:

LBC.

LBC, 2300.

This program is intended for loading cards punched through PBC.

LOAD OCTAL CORRECTIONS

This peripheral program may be called with a control card or at a display console. Octal corrections are read from the INPUT file and entered in central storage. The octal correction cards must be in the following format:

```
|1      |7
|23001  |45020    04000    00042    00044
```

Address begins in column 1; leading zeros may be dropped in the address. The data word begins in column 7; spacing in the data word is not important but the word must contain 20 digits.

LOC.	Reads all correction cards in the next INPUT file record and modifies central storage accordingly.
LOC, 1000.	Clears central storage from the reference address to the specified address; correction cards are then read from the INPUT file.

LOC (2022, 3465)	Clears central storage from the first specified address to the second; correction cards are then read from the INPUT file. This program may be called to clear storage by providing an empty record in the INPUT file.

10.3 INPUT/OUTPUT ROUTINES

PUNCH BINARY CARDS

Only a control card may be used to call this peripheral program which punches a deck of binary cards directly from central storage. Storage is not modified by this operation.

PBC, 2000.	A binary deck is punched from the reference address to the specified address.
PBC (2000, 3000)	A binary deck is punched from the first specified address up to but not including the second.
PBC.	Punches a binary deck using the contents of RA+117_8 as a control word for deck length. The deck always begins at the reference address + 100_8 and contains the number of words specified in the lower 18 bits of RA+117_8. This call may be used for punching any central or peripheral program in standard format.

READ BINARY RECORD

One binary record may be loaded from a file specified by the user.

RBR, n.

n	Specifies fifth character of file name, 1-7. (first four characters are TAPE)

Loading begins at RA+100. If the record cannot fit into central memory, a dayfile message RECORD TOO LONG appears. RBR uses central memory locations fl-5 through fl-1 for buffer parameters; the original contents of these locations are destroyed.

WRITE BINARY RECORD

A binary record may be written from central memory to a file specified by the user.

WBR, n, rl.

n Specifies fifth character of file name, 1-7. (first four characters are TAPE)

rl Record length in words. If omitted, length is taken from the lower 18 bits of RA+117_8.

WBR begins writing from RA+100. The contents of central memory locations fl-5 through fl-1 are used by WBR for buffer parameters; original contents of these locations are destroyed.

Example:

To write a program on tape after patching it:

```
REQUEST TAPE5.
REQUEST TAPE2.
REWIND(TAPE5).
REWIND(TAPE2).
RBR,5.
LOC.
WBR,2.
```

10.4 REQUEST FIELD LENGTH

The field length for the execution of a program may be changed.

RFL,nfl.

nfl New field length (octal)

This routine is also used internally by the compile (RUN). For a short 5000_8 word program, storage would be used most efficiently by specifying RFL.

Example:

```
JOB,T300,CM45000,P7.
RUN(S)
RFL, 5000.
SAM. (execute program with FL=5000)
```

10.5 DUMP STORAGE

This peripheral program may be called with a control card or from a display console in any of the forms shown below:

DMP.	Will dump the entire exchange package and 100 locations before and after a stop location.
DMP,x.	Will dump from the reference address to the parameter address.
DMP (x,y)	Will dump from the first specified address to the second. The entire control point area is also dumped.
DMP (4xxxxx, 4yyyyy)	Produces absolute core dumps. xxxxx defines the lower bound, yyyyy defines the upper bound of absolute core locations.

10.6 COMPARE

Using a control card beginning with COMPARE, one or more consecutive records on one file may be compared with the same number of consecutive records on another file to determine if they are identical. The pattern for the control card is:

```
COMPARE(f1,f2,n,l,e,r)
```

f1,f2	Files to be compared
n	Number of records in f1 to be compared to f2
l	End-of-record level number
e	Number of non-comparison records to be written to the OUTPUT file
r	Number (octal) of counted records to be processed during the comparison. Included in non-comparison record OUTPUT file if e parameter is given.

The comparison begins at the point where each file is currently positioned and continues until the number of ends-of-records of the level specified or a higher level has been passed over.

If the comparison shows that all pairs of records are identical, a dayfile message GOOD COMPARE is given; otherwise, the message is BAD COMPARE. Discrepancies may be listed on file OUTPUT depending on the parameters on the COMPARE card.

Examples:

```
COMPARE(RED,BLUE)
```

This will compare the next record on file RED with the next record on file BLUE.

```
COMPARE(RED,BLUE,6)
```

This will compare the next six records, whatever may be the level of their end-of-record marks. But each end-of-record on file RED must have the same level as the corresponding end-of-record on file BLUE.

```
COMPARE(RED,BLUE,3,2)
```

This will compare the two files from their current positions up to and including the third following end-of-record whose level number is at least 2. Both the records and the levels of their end-of-record must match to give GOOD COMPARE.

The only indication of bad comparison between corresponding records will be the message BAD REC.n on OUTPUT, where n is the record number, counting the first one read on each file as number 1. If more information is wanted, errors and records must be specified as parameters.

Example:

```
COMPARE(GREEN,BLACK,3,2,5,1000)
```

This will do the same comparison as the previous example, but for the first five discrepancies of a word in one file with the corresponding word in the other file, the words from each file will be listed in OUTPUT, together with their position in the record. The position will be indicated by an octal number, counting the first word as number 0. This will be done only for the first 1000 records read on each file in which discrepancies are found. 1000 is chosen as an indefinitely large number, because the number of records to be compared is rather small, and details are wanted about all discrepant records. If two long files were to be compared, something like 20 might be given as the records parameter, so that a reasonably large number of

discrepancies would be described in detail; but if through an error the two files were completely different, an enormous and useless listing would not be produced. Furthermore, the comparison will be abandoned if this limit is exceeded, and the files will be left positioned where they stand.

A discrepancy between the levels of corresponding ends-of-records will be listed on OUTPUT, and the comparison will be abandoned, leaving the files positioned immediately after the disagreeing ends-of-records.

Mode need not be specified in the COMPARE card. It is handled in the following manner.

The first record of the first-named file (GREEN) is first read in the binary mode. If this produces a redundancy check, it is backspaced and re-read in coded mode. If this still produces a redundancy check, the fact is noted in file OUTPUT, the corresponding record of the second-named file (BLACK) is skipped over, and the process begins again. If the coded read is successful, the corresponding record of file BLACK is read in coded mode. If this record of BLACK gives a redundancy check, the fact is noted in file OUTPUT, and nothing further is done with that record. Each record of file BLACK will be read in the same mode as that in which the corresponding record of GREEN was successfully read; but if the record of GREEN was unsuccessfully read in both modes, the record of BLACK will be read in the same mode as the preceding record of BLACK. Once a record of GREEN has been read without redundance in one mode or the other, following records of GREEN are read in the same mode until a change is forced by a redundancy check.

Disk records can be read indifferently in either mode, so that the above strategy imposes no difficulty if a tape file is being compared with a disk file.

Debugging aids include SNAP, TRACE, and DUMP and are submitted as normal jobs.

11.1 TRACE

The tracing capability provides an analysis of program execution. Instructions based on storage references, operand references, register usage and branch instructions are analyzed. Output is written on a local file named SNACE. If TRACE output is to be listed, SNACE must be rewound and copied to the standard output file (OUTPUT). TRACE output always includes a dump of the contents of the P register, all operand registers involved and the result register. An initial message indicates where tracing begins and the range involved. A terminal message is written when tracing stops.

Each instruction within a designated range is scanned for triggers, established by TRACE control card parameters. Traps are placed at instructions which contain triggers. As each trap is encountered during execution, the designated instruction is executed and the specified output is written on SNACE. TRACE continues until the specified parameters are satisfied and as long as the program remains inside the designated range.

Return jump instructions outside the range must be calls to simple subroutines, and they must return through the subroutine's entry point. Tracing stops when the subroutine is entered and resumes when the subroutine returns to the range.

Tracing ranges can overlap and multiple outputs can be triggered.

TRACE may be used with all system loading schemes except that programs loaded entirely from the library cannot be traced. OVERLAY/SEGMENT mode has special requirements (section 11.1.3). When TRACE cards are encountered, the system prepares TRACE tables to be referenced during subsequent loading. Calls for SNAP features cannot be traced.

11.1.1 SCOPE CONTROL CARD

The following SCOPE control card initiates TRACE.

```
TRACE,p1,p2,...,pn.
```

The order of parameters is not significant except as noted below. TRACE cards must be loaded contiguously.

Parameter	Description
Range Identification: ID=i	i is an optional, alphanumeric identifier (1-7 characters) printed on TRACE output. If continuation cards are used, it must appear somewhere on the first card and on all continuation cards as the first parameter.
Initial address: IA=e or e+n IA1=e-n IAC=c or c+n IAC1=c-n	e is an entry point name. c is a labeled common block name. n is an octal integer ≤ 777777.
Last address: LA=e or e+n LA1=e-n LAC=c or c+n LAC1=c-n	The tracing range includes all instructions from IA to LA (LA must be greater than IA).

Branches outside range IA-LA terminate trace output. It resumes when control passes back into the range. Tracing for the specified range terminates until control passes back through the address at which tracing begins.

The IA† flag, set when IA is encountered, is turned off only when LA† is encountered. When a trigger is encountered, only the output specifications with a set IA† flag are processed. The first time IA is passed through, the trace counter is changed to 1. The counter is incremented only if control passes through LA† of the range prior to passing through IA again. Returning to IA before LA is encountered does not affect the frequency parameter count. Output is dependent upon the frequency parameters.

---

†References to IA apply to IA1, IAC, IAC1 also; references to LA apply to LA1, LAC, LAC1 also.

Parameter	Description
Frequency:	n is an octal integer; it must not be 0. If parameter is not specified, 1 is assumed.
F1=n	F1 Tracing begins F1st time IA is encountered.
F2=n	F3 Thereafter, tracing takes place every F3rd time IA is encountered.
F3=n	F2 Tracing stops F2nd time IA is encountered.

Three trigger specification parameters are described below; at least one must appear on a TRACE card.

Register trigger:	n is the register number 0-7.
TR=P,An,Bn, or Xn	Each instruction is examined to determine whether or not the specified register is used as a result register. The P register measures satisfactory completion of a conditional jump. It must be placed before other triggers; otherwise, traps are set for previously set traps.
Masking trigger:	m is an octal mask. (5 or 10 digits)
TM=m,k_1,k_2,...,k_n	k is a match key associated with mask m. A mask (Boolean AND) of each instruction in the range is compared with all k's for that mask. If equality to any k is found, the instruction is used as a trigger.
Location trigger:	e is entry point name
TL=e or e+n	c is labeled common block name
TL1=e-n	n is octal integer ≤777777
TLC=c or c+n	b is nth location in blank common
TLC1=c-n	
TLB=b	Each instruction making an A-register reference to the location is used as a trigger.
Register dump:	If RD is specified immediately following a trigger, (TR, TM, or TL) an octal dump of all A, B and X registers is included in the output.
RD	

The two output specification parameters are activated when one of the trigger parameters is encountered.

Parameter	Description
Storage location reference:	i is an octal integer less than 100
OL=e,i or e+n,i OL1=e-n,i OLC=c,i or c+n,i OLC1=c-n,i OLB=b,i	When a trigger is encountered, i words beginning with the specified location are written in octal format on SNACE. i must be specified.
Register designator: OR=r,i	When a trigger is encountered, i words beginning at the location specified in the designated register are written in octal format on SNACE. i must be specified. r is a register designator: An, Bn, Xn n=0-7

11.1.2 TRACE EXAMPLES AND OUTPUT

```
TRACE,ID=AA,IA=ST,LA=NT,TL=NT,RD,OL=ST,77,F2=10.
```

ID=AA	AA is the range identifier in messages produced each time the start or end of range is encountered and each time output results from trap execution.
IA=ST	ST is an entry point in user's program; it designates beginning of range.
LA=NT	NT is an entry point in user's program; it designates end of range.
TL=NT,RD	Trigger which causes trap to be set each time NT is referenced within the stated range. At execution time, the trap in the instruction referencing NT causes TRACE output. RD causes a dump of the registers each time an instruction referencing NT is executed.
OL=ST,77	Output trigger. Each time the instruction referencing NT is executed, 77_8 words are dumped beginning at the entry point ST.
F2=10	Output is produced the first eight times the instruction is executed. (F parameters are assumed to equal 1 if not present on a trace card; therefore, F1 and F3=1.)

OUTPUT

Assume: ST=4567 (IA)

NT=4577 (LA)

The instruction SA5 NT is present at location 4571.

```
AA STARTS IN LOCATION 004567
TRAP FOR AA AT 4571
OPERAND REGISTERS, B0=000000
    K=004577
RESULT REGISTER IS A5=004577
B0=000000
B1=054520
B2
.
.
.
X7=01040422000000000000
004567      data      data      data      data
004573       .         .         .         .
.            .         .         .         .
.            .         .         .        data
004663      data      data      data
AA ENDS IN LOCATION 004577
```

```
TRACE,ID=REGS,IA=START,LA=NEXT,TR=P,RD,TR=X6,OR=B4,6.
```

ID=REGS — range identifier

IA=START, LA=NEXT — range limits

TR=P,RD — trap triggers: Each time a jump occurs, a trap is set and the registers are dumped (RD).

TR=X6 — trap triggers: Each time X6 is used† as a result register, a trap is set.

OR=B4,6 — Output consists of six words starting at the address in B4.

†In this case, only once: no frequency parameters are specified; each is assumed to be 1.

```
TRACE,ID=Q,IA=S,LA=E,F1=3,F2=7,F3=2,TM=00700,00600,OR=B4,7.
```

ID=Q	range identifier	
IA=S LA=E	range limits	
TM=00700, 00600	trap trigger	00700=octal mask 00600=match key Whenever the third digit of an instruction is six, the designated output (OR=B4,7) occurs if the frequency requirements are met.
F1=3,F2=7, F3=2	frequency requirements	Output is not to begin until the third time the range is passed through. It is to be repeated each second time thereafter through the seventh time the trap is encountered.
OR=B4,7	output trigger	Output consists of seven words of data starting at the address in B4 register.

11.1.3 TRACE IN OVERLAY OR SEGMENT MODE

In overlay or segment mode, the DEBUG card (11.4.1) with the T parameter must be present when the overlay file is generated. As the TRACE routine is loaded with SEGZERO or with the (0,0) level overlay, TRACE cards must appear just prior to the program call card which causes loading of the (0,0) level overlay or SEGZERO (section 11.5, Sample Deck Structures).

When TRACE cards are encountered, the system prepares TRACE tables to be referenced during subsequent loading. The loader tables for overlays are read from the overlay file. As each overlay is loaded, TRACE's which apply to it are established. Similarly, segment loading causes TRACE traps to be inserted.

11.2 SNAP

The SNAP dump capability provides selective area printouts upon execution of specified instructions. Printing frequency is established by parameters. The dump format is variable.

When SNAP cards are encountered the system prepares SNAP tables (which extend toward the reference address). During subsequent loading SNAP's are inserted which apply to the newly loaded programs. The SNAP control card may specify an entry point to a user supplied routine which is entered before the SNAP output is written.

Prior to execution, the instruction at a SNAP triggered address (IA) is replaced by a return jump to the SNAP routine or a user routine if specified. The replaced address is saved. When the trapped address is encountered during execution, the SNAP routine stores all registers. Routine parameters are contained in arrays; the addresses of the arrays are passed to the specified routine. The arrays contain: the saved registers, the parameters from the SNAP card, and the address at which the SNAP occurs. Following return from the routine, the SNAP dump is written on the local file SNACE or on an alternate file if an FET address is specified by the routine. To obtain listings, the dump written on local file SNACE must be copied onto the file OUTPUT. A user routine may set a flag to suppress output.

Following the dump, saved instructions are executed before passing control to the trapped location + 1. If an alternate address is placed in the communications area, SNAP will return to it after executing the saved instructions.

SNAP cannot be used for programs loaded entirely from the library.

11.2.1 SCOPE CONTROL CARD

The following SCOPE control card initiates SNAP:

```
SNAP, p1,p2,...,pn.
```

Parameters may appear in any order except as noted below. All SNAP cards must appear contiguously.

Parameter	Description
SNAP identifier: ID=i	i is an optional 1-7 character alphanumeric identifier printed with the dump. If continuation cards are used, ID must appear somewhere on the first card and as the first parameter on continuation cards.

Parameter	Description
Address where trap is planted: IA=e,e+n, or a IA1=e-n IAC=c or c+n IAC1=c-n	e is an entry point name c is a labeled common block name n is an octal integer a is an absolute address (relative to RA)
First word address of area dumped: FWA=e,e+n,n, or a FWA1=e-n or n FWAC=c, c+n, or n FWAC1=c-n or n FWAB=b	b is bth location in blank common; other symbols are as in IA. n is assumed if e (or c) has appeared as a previous parameter on the card. Thus, address will be e+n, e-n, c+n, or c-n, as appropriate. a is assumed if e has not appeared yet on this card.
Last word address of area dumped: LWA LWA1 LWAC LWAC1 LWAB	Format is the same as for FWA. LWA must be ≥ FWA.
Interval between words dumped: INT=n	n is a positive octal integer; if not specified, 1 is assumed. For a D dump, n is doubled.
Dump format: F=code	Designated by one or two of the following codes: One only of the characters may be: O Octal dump M Octal dump with mnemonic operation codes I Integer dump S Single precision floating point F If exponent = 0, I format; otherwise, S format. D Double precision floating point dump (two words) C Display code dump

Parameter	Description
	The second character is optional; it may suffix or prefix any other designator. R Register dump. If FWA and LWA are present and F is not specified, octal (O) and register (R) dumps are given. If FWA and LWA are not present and F is not specified, a register dump is given.
Frequency: F1=n F2=n F3=n	n is an octal integer. It cannot be zero; 1 is assumed if n is not specified. F1 Tracing begins the F1st time IA is encountered. F3 Therafter, tracing takes place every F3rd time IA is encountered. F2 Tracing stops the F2nd time IA is encountered.
User's entry point: UR=p,r1,...rn	Optional parameter; must be last on the card. p specifies the user's entry point to be called before SNAP dump is taken. r parameters, passed to the routine, may be of two forms: Alphanumeric string, 1-10 characters, terminated by a zero byte. If the string contains 9 or 10 characters, an extra word is required. Decimal integer, converted to binary, stored right justified. The parameter list is terminated by a -0 (word filled with sevens) which is used optionally by the user's routine. It has no meaning for SNAP.

SNAP enters the user entry point in the following manner:

L	RJ	P (user entry address)
L+1		TADR (FWA of loader SNAP tables) User parameters begin at FWA-10B and extend toward the reference address.
L+2		RB0 (FWA of register storage area)
L+3		Return to user routine The user program must increment the return address by two so that return to SNAP will be at L+3.

Registers are stored one per word in the first 24 words of the register storage area as follows:

B0-B7, A0-A7, X0-X7

RB0+24 has the following format:

Bits	
59	No-dump flag; if bit 59 is set, SNAP output is suppressed. This bit is cleared on entry to the user routine.
18-58	Not used
0-17	FET address can be inserted to designate an alternate file for SNAP output.

RB0+25 has the following format:

Bits	
18-59	Not used
0-17	Address to which SNAP returns (address+1 of trapped instruction). Address return can be changed by replacing the address in these bits.

11.2.2 SNAP CONTROL CARD EXAMPLES

```
SNAP (ID=AX,IA=L, FWA=B, LWA=B+150,F1=10,F2=35,F3=2, F=O)
```

Produces a dump in octal format labeled AX consisting of all locations from B to $B+150_8$. The dump starts the 8th time the instruction at location L is encountered, and is taken every 2nd time thereafter through the $34th_8$ time.

```
SNAP(ID=AX,IA=L, FWA=B, LWA=B+150,INT=5,F1=10,F2=35,F3=2, F=O)
```

Same as above, except every 5th word is dumped starting at location B and ending at location $B+144_8$.

```
SNAP(ID=AX,IA=L,FWA=B,LWA=B+150,INT=5)
```

Continuation card:

```
SNAP(ID=AX,F1=10,F2=35,F3=2,F=RO)
```

Each dump begins with the contents of all registers at the time of entry to SNAP.

```
SNAP(IA=TAG)
```

The first time location TAG is executed, a dump is produced of the contents of all registers as they appeared upon entry to SNAP.

```
SNAP(IA=HOOK,F=M,FWA=C,LWA=C+30,ID=SYM,UR=IN,1,A,2)
```

When location HOOK is executed, control passes to a user subroutine (entry point=IN). If the user routine returns control to SNAP, and if the no-dump flag is not set (in RB0+24, bit 59), a mnemonic dump is taken (labeled SYM of locations C through $C+30_8$. Parameters 1, A, and 2 are appended to the loader SNAP tables.

11.2.3 SNAP IN OVERLAY OR SEGMENT MODE

SNAP declarations are inserted as each segment is loaded. The SNAP routine is loaded with SEGZERO and with the (0,0) level overlay. The DEBUG card with the S parameter must be included when the overlay file is prepared. The DEBUG(S) card must appear immediately before the initiation of a segment load (section 11.5). With normal loads, the DEBUG card is not necessary.

11.3 DMP

Upon normal or abnormal job termination, three dump formats are available: octal, labeled, change.

Octal

Standard DMP option. If a DEBUG control card (section 11.4) is not present in a job, an octal core dump is produced when the DMP control card is encountered.

Labeled

If a DEBUG control card with no parameters is present, a labeled dump is produced when the DMP control card is encountered. Format of the dump is the same as for the octal dump; except as the origin of a common block or subprogram is encountered, the associated name is printed. Also, a relative address counter indicates the position of the first word on the line relative to the last encountered subprogram or common block. The DEBUG file is used to locate the origin and names of the subprogram and common blocks.

The DMP card uses symbolic names as well as octal numbers; the two may be combined. (A common block name is preceded by an empty parameter.) The dump begins at the origin of the first parameter name and continues through the space occupied by the subprogram (or common block) mentioned as the second parameter. The second parameter origin must be greater than the first parameter origin.

Example:

```
DMP(ALPHA,,CAT)
```

Produces a labeled dump of the program ALPHA and all locations through program CAT. Intermediate programs encountered are identified.

If a job is in overlay or segment mode, the DEBUG file is updated with the loading of each overlay or segment.

Change

If parameter C is present on a DEBUG control card, a list of core locations which have changed from their initial values is produced when the DMP control card is encountered. When a job begins an execution phase, a core image of the entire field length is written on the DEBUG file. The image is compared with the contents of memory at the time of termination. The contents of changed locations are listed. A labeled dump always precedes a change dump.

Change dumps permit a swift analysis of subprograms entered, changed data, and modified instructions. Large areas of instructions or data which have remained constant need not be considered.

A change dump will not be produced during overlay or segment mode.

11.3.1 DMP CONTROL CARDS

DMP.	Dumps the exchange package and P-77 through P+77 onto OUTPUT. The exchange package dump consists of P, RA, FL, RAECS, FLECS, EM, A0...A7, B0...B7, X0...X7 and the contents of locations A0-A7. Each line of the storage dump contains an address and the contents of from one to four central memory words starting at that address.
DMP,xxxxxx.	Dumps from the reference address through the parameter address, xxxxxx.
DMP(xxxxxx,yyyyyy)	Dumps from address xxxxxx through yyyyyy. If the high-order bit of each 18-bit address is set, an absolute dump is given. (For example, DMP (400300, 400450) causes absolute locations 300 through 450 to be dumped, not RA+300 through RA+450). If a DEBUG file is created, xxxxxx and yyyyyy may be symbolic. Printing of a central memory word is suppressed when that word is identical to the last word printed. Its location is printed and marked by a right arrow.

11.3.2 DMP EXAMPLES

DMP(1000) or DMP(100,200)	Interpreted as a standard DMP request except it can be labeled (with or without a change dump) if appropriate DEBUG cards are present.

The following dumps must be labeled:

Call	Dump Beginning	Dump End
DMP(CPC,IO)	Start of program CPC	End of program IO†
DMP(COPYL)	Reference address	Beginning of COPYL
DMP(100,COPYL)	RA+100	End of COPYL†
DMP(COPYL,2000)	Beginning of COPYL	RA+2000_8
DMP(COPYL,COPYL)	Start of program COPYL	End of program COPYL†
DMP(,RED)	Reference address	Start of common block RED†
DMP(,RED,,WHITE)	Start of common block RED	End of common block WHITE†
DMP(,RED,,RED)	Start of common block RED	End of common block RED†
DMP(100,,RED)	RA+100_8	End of common block RED†
DMP(IDA,,RED)	Start of program IDA	End of common block RED†
DMP(,WHITE,ELLA)	Start of common block WHITE	End of program ELLA†
DMP(,WHITE,70000)	Start of common block WHITE	RA+70000_8

11.4 DEBUG

The DEBUG control card is required when debug aids are used with overlay or segment jobs or when a labeled or change dump is requested. The DEBUG control card applies to all subsequent loading and executions within a job. Any program loaded completely from the system library, however, cannot use the debugging aids. Such routines can be debugged only when they are loaded from a user file.

---

† One word beyond dump end is dumped also.

11.4.1 DEBUG, CONTROL CARD

DEBUG, C, T, S. (Parameters are optional.)

C	Labeled dump is followed by a change dump when the DMP card is encountered; if C is absent, only a labeled dump is produced.
T	In overlay mode, loads TRACE and SNAP routines with the (0,0) overlay; in segment mode, loads TRACE with SEGZERO.
S	In overlay mode, loads TRACE and SNAP routines with the (0,0) overlay; in segment mode, loads SNAP with SEGZERO.

DEBUG cards with both C and T parameters (or with both C and S parameters) cannot appear in the same job: a change jump is not allowed in overlay or segment jobs. If such cards do appear, the job is not terminated, but the change dump is not produced. DEBUG(T) or DEBUG(S) does not signal a labeled dump. A DEBUG card with no parameters must be present to obtain a labeled dump without a change dump.

11.4.2 DEBUG USE

Overlay loading †

If a DEBUG card is included when an overlay is prepared, the loader inserts a record on the overlay file following the overlay. This record consists of the loader table information necessary for traces, and for snapshot and formatted dumps. When the overlay is loaded, the table information is extracted from the overlay file and placed on the DEBUG file.

Normal loading/segment loading †

Upon completion of loading, a local file is created. This file, named DEBUG, contains the loader table information necessary for formatted dumps. It is updated as each segment is loaded. During user loading, execution does not affect or is not affected by the DEBUG card except for user segment loading.

†The change dump does not apply to overlay or segment loading.

11.5 SAMPLE DECK STRUCTURES

TRACE run:

job card

COMPASS.

TRACE (params)

LGO.

TRACE and SNAP cards appear immediately before program call card that initiates the run.

REWIND (SNACE)

COPYCF (SNACE,OUTPUT)

$^{7}_{8}_{9}$

(COMPASS SOURCE DECK)

$^{7}_{8}_{9}$

$^{6}_{7}_{8}_{9}$

Combined TRACE/SNAP run:

```
job card
COMPASS.
COPYBR(INPUT, LGO)
TRACE(params)
:
TRACE(params)
SNAP(params)
:
SNAP(params)
LGO.
REWIND(SNACE)
COPYCR(SNACE, OUTPUT)
7
8
9
(COMPASS source deck)
7
8
9
(Binary of previously assembled program)
7
8
9
6
7
8
9
```

TRACE and SNAP cards appear immediately before program call card that initiates the run.

Normal execution with labeled dump if job aborts:

job card

DEBUG. — DEBUG card remains in force throughout the job.

COPYBR(INPUT, LGO)

COMPASS.

LGO.

EXIT.

DMP(6000) — Dumps locations occupied by program PA. PA must have been loaded for this card to be accepted.

DMP(PA, PA)

7_{8_9}

(Binary of previously assembled program)

7_{8_9}

(COMPASS source decks including program called PA)

7_{8_9}

$6_{7_{8_9}}$

If COMPASS terminates abnormally, the labeled dump produces labels reflecting programs loaded for COPYBR. COMPASS, a library overlay, has no loader tables to update the DEBUG file.

SNAP run with labeled and change dump:

job card

DEBUG(C)	DEBUG card must appear before the SNAP card.
SNAP(params)	SNAP cards apply only to next load; they must appear immediately before the card initiating that load. (SNAP and TRACE cards do not signal the end of the current load.)
INPUT.	
DMP(1000,2000)	

REWIND(SNACE)

COPYCF(SNACE,OUTPUT)

EXIT.

DMP(5000)

REWIND(SNACE)

COPYCF(SNACE,OUTPUT)

7
8
9

(Binary decks to execute)

7
8
9

6
7
8
9

Overlay run using both SNAP and labeled dumps:

```
job card
DEBUG.
DEBUG(S)
LOAD(FILE)
NOGO.
SNAP(params)
OVFILE.
DMP(1000)
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
EXIT.
DMP(10000)
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
7
8
9
6
7
8
9
```

FILE contains overlay directives and binary decks which comprise the overlays to be written on file OVFILE when LOAD(FILE) is processed. Once execution of the overlay file begins, dumps will be labeled because of the DEBUG card. The following rules apply:

The DEBUG(S) card must precede the LOAD(FILE) cards so that the loader tables will be placed on the overlay file as it is generated and so that in (0,0) overlay the debugging routines will be loaded.

The NOGO card must appear; otherwise, the SNAP routine is loaded in the last overlay.

The SNAP cards must appear just before the card which initiates loading of the (0,0) overlay.

Segment run using both TRACE and labeled dumps:

job card

DEBUG.

DEBUG(T)

TRACE(params)

The DEBUG cards must appear before any TRACE or SNAP cards; TRACE and SNAP cards must appear immediately before the load to which they apply.

INPUT.

DMP(40000)

REWIND(SNACE)

COPYCF(SNACE,OUTPUT)

7
8
9

(Segmentation loader directive cards)

(Binary decks)

7
8
9

6
7
8
9

SNAP run with SNACE output going to tape;
tape to be listed at a later time:

job card

REQUEST SNACE.

REWIND(SNACE)

SNAP(params)

INPUT.

COPYBF(X, SNACE)

EXIT.

COPYBF(X, SNACE)

The COPYBF(X, SNACE) cards write an end-of-file on the tape. Normally, this is a faster method of running SNAP and TRACE when output to SNACE is extensive.

$^{7}_{8}_{9}$

object deck

$^{6}_{7}_{8}_{9}$

job card

REQUEST SNACE.

REWIND(SNACE)

COPYCF(SNACE, OUTPUT)

$^{7}_{8}_{9}$

$^{6}_{7}_{8}_{9}$

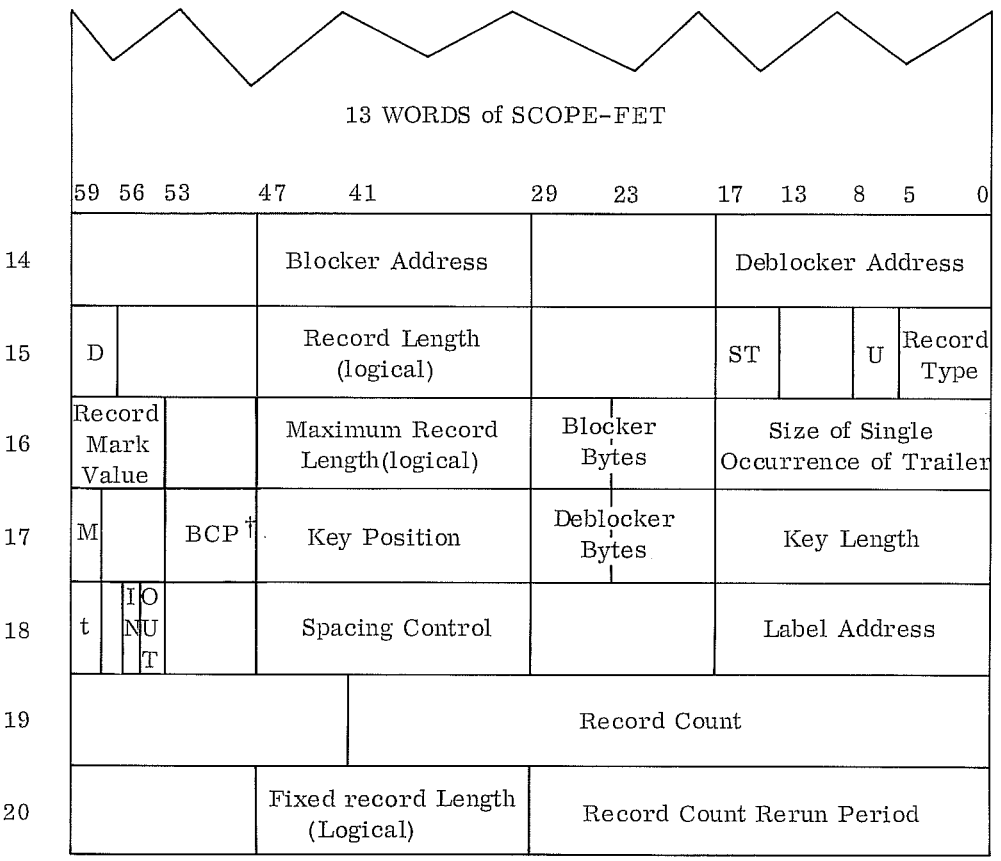
APPENDIX SECTION

CHAR (printed)	DIS-PLAY	HOLLERITH (punched)	EXT BCD	CHAR (printed)	DIS-PLAY	HOLLERITH (punched)	EXT BCD
A	01	12-1	61	0	33	0	12
B	02	12-2	62	1	34	1	01
C	03	12-3	63	2	35	2	02
D	04	12-4	64	3	36	3	03
E	05	12-5	65	4	37	4	04
F	06	12-6	66	5	40	5	05
G	07	12-7	67	6	41	6	06
H	10	12-8	70	7	42	7	07
I	11	12-9	71	8	43	8	10
J	12	11-1	41	9	44	9	11
K	13	11-2	42	+	45	12	60
L	14	11-3	43	-	46	11	40
M	15	11-4	44	*	47	11-8-4	54
N	16	11-5	45	/	50	0-1	21
O	17	11-6	46	(	51	0-8-4	34
P	20	11-7	47	)	52	12-8-4	74
Q	21	11-8	50	$	53	11-8-3	53
R	22	11-9	51	=	54	8-3	13
S	23	0-2	22	blank	55	space	20
T	24	0-3	23	,	56	0-8-3	33
U	25	0-4	24	.	57	12-8-3	73
V	26	0-5	25	≡	60	0-8-6	36
W	27	0-6	26	[	61	8-7	17
X	30	0-7	27	]	62	0-8-2	32
Y	31	0-8	30	:	63	8-2	00
Z	32	0-9	31	≠	64	8-4	14

CHAR (printed)	DIS-PLAY	HOLLERITH (punched)	EXT BCD
→	65	0-8-5	35
V	66	11-0	52†
∧	67	0-8-7	37
↑	70	11-8-5	55
↓	71	11-8-6	56
<	72	12-0	72††
>	73	11-8-7	57
≤	74	8-5	15
≥	75	12-8-5	75
¬	76	12-8-6	76
;	77	12-8-7	77
end-of-line	0000		1632

†11-0 and 11-8-2 are equivalent

††12-0 and 12-8-2 are equivalent



†for Depending on

BLOCKER ADDRESS (18 bits)

This is the address of a routine which packs logical records into an output buffer. The routine determines if there is room in the buffer to receive the logical record. If so, the BLOCKER routine transfers the logical record to the buffer and updates OUT. If not, the BLOCKER routine calls SCOPE to write the buffer area onto an external device before transferring the logical records to the output buffer. BLOCKER routines go into RECALL status while SCOPE is processing the data transfer.

DEBLOCKER ADDRESS (18 bits)

This is the address of a routine which unpacks logical records from an input buffer into a user's record area, or which supplies the user with the record's address in the input buffer. This routine determines if the input buffer already contains another logical record. If so, the DEBLOCKER processes it as required; if not, the DEBLOCKER calls SCOPE to fill the buffer. DEBLOCKER goes into RECALL status during the SCOPE operation.

D = Disposal Code

This field for magnetic tape files assumes the following values:

100	REWIND (release)
010	LOCK (save)
001	NO REWIND

RECORD LOGICAL LENGTH (18 bits)

This field is the number of 6-bit bytes in the records for fixed length records; for OCCURS (trailer) records, it is the size of the fixed portion (length if no trailer items exist); for all other types of variable length records, this field contains the minimum record size.

ST = Logical Status (4 bits)

This field indicates the logical status of a file:

1XXX	file is currently open
X1XX	file is an optional file
XX10	records must be counted to control restart dump
XX01	end of reel condition controls restart dump

U = Use Code (3 bits)

This field used by SORT/MERGE, contains an indicator as to the use of the file described by this FET. Its values are:

001	sort input
010	merge input
100	output

RECORD TYPE (6 bits)

This field describes the kind of records in the file:

000001	fixed length record
000010	variable length records (length controlled by key field)
000100	variable length records (length determined by presence of a record mark character)
001000	variable length records (fixed portion plus a variable number of fixed length trailer items). This is the COBOL OCCURS DEPENDING ON type of variability.
010000	universal record. The BLOCKER prefixes each logical record with one word containing the number of 6-bit bytes in the record and the DEBLOCKER removes this word. Each of the other record types can be mapped into the universal record format.

RECORD MARK VALUE (6 bits)

This field contains the octal value of the record delimiter for record mark types.

MAXIMUM LOGICAL RECORD LENGTH (18 bits)

This field contains maximum size for variable records of all types as the number of 6-bit bytes in the record. It is not used for fixed length records.

BLOCKER/DEBLOCKER BYTES (12 bits)

Used in combination with IN (word 3) and OUT (word 4) to specify the next logical record position in the CIO buffer to be blocked or deblocked.

SIZE OF SINGLE OCCURRENCE (18 bits)
size of one trailer item

This field contains the number of 6-bit bytes in a single occurrence of the trailer item. It is meaningful only for trailer type variable length records.

M = Mode (2 bits)

This field indicates the recording format of the length key field:

0	binary number
1	decimal number
2	floating point integer

BCP

Beginning character position for variable length records.

KEY POSITION (18 bits)

This field is interpreted by SORT/MERGE as the position of the first 6-bit byte in the length key field. If the length key field begins in the first 6-bit byte of the record, this field contains the value of 1.

This field is interpreted by COBOL as the CM address of the length key field.

KEY LENGTH (18 bits)

This field contains the number of 6-bit bytes in the length key field. The length key field contains the number of 6-bit bytes in the record for key field variable records and it contains the number of trailer items with the record for variable trailer records (OCCURS DEPENDING ON records). This field must begin within the first n 6-bit bytes when n is MINIMUM LOGICAL RECORD LENGTH less the KEY LENGTH.

t = label type (2 bits)

0	standard labels
1	non-standard labels
2	omitted labels

IN (1 bit)

Set if file is opened for input or input-output processing.

OUT (1 bit)

Set if file is opened for output or input-output processing.

SPACING CONTROL (18 bits)

This field contains the count of the number of lines to advance the pointer when the WRITE BEFORE ADVANCING or the WRITE AFTER ADVANCING option is used in COBOL.

LABEL ADDRESS (18 bits)

The address of a 120-character area where the user places the value to be checked against the value in the first physical record of a file declared as having a non-standard label.

RECORD COUNT (42 bits)

This field is used to count the records that have been processed in the file described by this FET.

FIXED RECORD LENGTH (Logical) (18 bits)

Contains zero for variable length records or the number of characters in fixed length records.

FIXED COUNT RERUN PERIOD (30 bits)

When the ST (logical status field) indicates that a restart dump is to be taken every n records (specified by user), this field contains n (the number of records to be processed between restart dumps).

The following macro generates the FET appendix.

RECORD T, O, C, E, L, D, U

The RECORD macro generates FET +13 and FET +14.

T Record type = F, V, T, R, S

- F fixed length records
- V variable length records (key field type)
- T trailer item variable length records (OCCURS DEPENDING ON)
- R record mark variable length records
- S systems record format

O Optional file indicator

- blank mandatory file
- non-blank optional file

C Restart dumps controlled by record count

- blank not controlled by
- non-blank controlled by record count

E Restart dumps controlled by end of reel condition

- blank not controlled by end of reel condition
- non-blank controlled by end of reel condition

L Record length, number of 6-bit bytes in the record

D Disposal Code

- R REWIND (release)
- L LOCK (save)
- N NO REWIND

U Usage

- S sort input
- M merge input
- O output

VARIABLE L, M, P, K, S, R

The VARIABLE macro creates FET +15 and FET +16.

- L — Maximum record length as the number of 6-bit bytes in the largest record
- M — Key mode
  - B — binary
  - D — decimal
  - F — floating point integer
- P — Key position, position in record of leftmost 6-bit byte in the key field
- K — Key length as the number of 6-bit bytes in the key field
- S — Size of a single trailer item as number of 6-bit bytes in item
- R — Record mark value as octal representation of delimiting character

CONTROL P, t, LA

The CONTROL macro creates FET +17, +18, and +19.

- P — Number of records to be processed between restart dumps
- t — Label type
- LA — Label address for files with non-standard labels

STANDARD LABELS C

Standard Labels are all recorded in the BCD mode at 556 bpi, they are 80 characters in length.

For SCOPE 3 only

The standard disk pack device label is 80 characters in length and it is recorded with the RBR table copy in bytes 0 through 321 of sectors 0, 1 and 2 of cylinder zero, head group zero. Data is formatted like a standard allocatable device PRU.] The labels, as described, are designed to conform to the proposed USA Standard for Magnetic Tape Labels and File Structure for Information Interchange submitted by the x.3.2/457 Committee on November 28, 1966.

In this appendix, "n" means any numeric digit, 0 through 9; and "a" means any of the 6-bit characters of the character set in Appendix A.

An optional field may, but does not necessarily, contain the information described. If an optional field does not contain the designated information, it should contain blanks. Fields not described as optional are considered to be mandatory and must be written as specified.

For SCOPE 3 only

DISK PACK DEVICE LABEL PRU

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be DEV
2	Label number	1	4	Must be 1
3	Reserved	6	5-10	Must be 000000
4	Visual identifier	10	11-20	Optional, unused
5	Reserved	60	21-80	Optional, unused
6	RBR table	380	81-460	Formatted RBR entry
7	Reserved	178	461-638	Optional, unused
8	Checksum	2	639-640	Logical sum of fields 1-7

VOLUME HEADER LABEL

Field	Name	Length	Position	Description
1	Label Identifier	3	1-3	Must be VOL
2	Label Number	1	4	Must be 1
3	Visual Reel Number	6	5-10	Six n characters
4	Security	1	11	Blank = not security protected Non-blank = security protected
5	Volume Density	1	12	Density of file information on tape blank or 00 = 556 bpi 1 = 200 bpi 2 = 800 bpi
6	Reserved for operating system	19	13-31	Must be blank
7	Reserved for future standardization	49	32-80	Must be blank

FILE HEADER LABEL

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be HDR
2	Label number	1	4	Must be 1
3	File label name	17	5-21	Any a characters to identify this file
4	Multi-file identification	6	22-27	Any a characters to identify the set of files that includes this one. This ID must be the same for all files of a multi-file set (mfn)
5	Reel number	4	28-31	4 n characters. Incremented by one immediately after trailer label is written on the volume
6	Multi-file (position-number)	4	32-35	4 n characters denoting position number of file within the set of files.
7	Reserved for future standardization	4	36-39	Must be blank
8	Edition number	2	40-41	Two n characters distinguishing successive iterations of same file
9	Reserved for future standard-ization	1	42	Must be blank
10	Creation Date	5	43-47	Date file was created; YYDDD, which is 2 n characters for year and 3 n characters for Julian date (001 to 366)
11	Reserved for future standardization	1	42	Must be space
12	Expiration date	5	49-53	Same format as Field 10. This file is regarded as expired when today's date is equal to or later than the date given in this field. When this condition is satisfied, the remainder of this volume may be overwritten. To be effective on multi-file volumes,therefore, the expiration date of a file must be less than or equal to the expiration date of all previous files on the volume
13	Security	1	54	Same as field 4 of the Volume Header Label
14	Block Count	6	55-60	Must be zeros
15	Reserved for future standardization	20	60-80	Must be spaces

FILE TRAILER LABEL

Field	Name	Length	Position	Description
1	Label Identifier	3	1-3	Must be EOF
2	Label Number	1	4	Must be 1
3-13	Same as corresponding fields in File Header (optional)	50	5-54	Same as corresponding fields in File Header
14	Block Count	6	55-60	Six n characters, number of data blocks (including labels and tape marks) written since last File Header Label
15	Reserved for future standardization	20	61-80	Must be blank

VOLUME TRAILER LABEL

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be EOV
2	Label number	1	4	Must be 1
3-12	Same as corresponding fields in File Header (optional)	50	5-54	Same as corresponding field in file header
13	Block count	6	55-60	6 n characters, number of data blocks (excluding labels and tape marks) written since preceding volume label
14	Reserved for future standardization	20	61-80	Must be blanks

The volume trailer label format is identical to file trailer label format except for the third character.

RELOCATABLE SUBROUTINE FORMAT

D

The deck of one subprogram (subroutine) as it is output from COMPASS or one of the other language translators comprises one logical record. Each logical record is made up of an indefinite number of tables. Each table is preceded by an identification word which specifies to the loader the procedure to be followed in loading the table. The identification word has the format:

CN Code number identifying type of data in table (text, entry points, external references, etc.)

WC Word count in table excluding identification word

LR Method of relocation for the load address

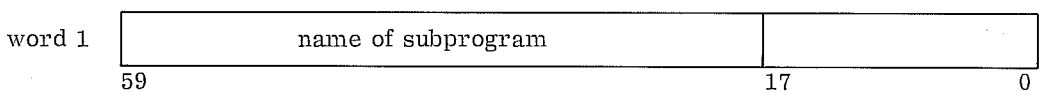
L Load address, 18-bits as defined in the following table formats:

LR and other relocation fields in the tables are nine bits long. Six of the nine are used currently; the other three are reserved for future expansion.

Prefix Table

The prefix table, if present, is the first table in a subroutine. It is bypassed by the loader. The prefix table is used by EDITLIB in constructing or modifying the SCOPE library. The format of the table is:

CN 77 LR and L are ignored.



PIDLE

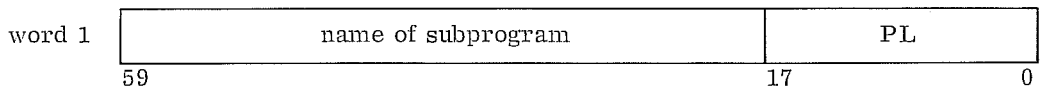
Program Identification and Length table contains the subprogram identification and declarations concerning common block allocation.

Identification Word

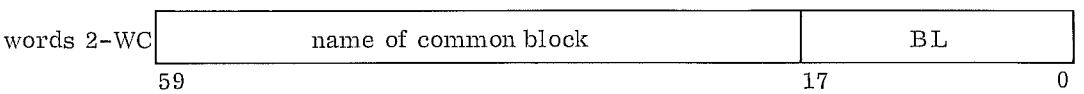
CN 34_8

LR Unused

L 0



PL Program length



Name may be 7 display code blank characters.
BL Block length

If WC = 1, no common references appear in the program. Subprogram length is relevant only in the first PIDLE table. All PIDLE tables must appear before any other tables for a given subprogram. The names of common blocks may not be duplicated in PIDLE. The list of common block names is called the Local Common Table (LCT). Since relocation of addresses relative to common blocks is designated by positions in LCT, the order of the common block names is significant.

The first word in the LCT is referred to as position 1.

ENTR

The entry point table contains a list of all the named entry points to the subprogram and its associated labeled common blocks. The ENTR table must immediately follow the PIDL table.

Identification Word

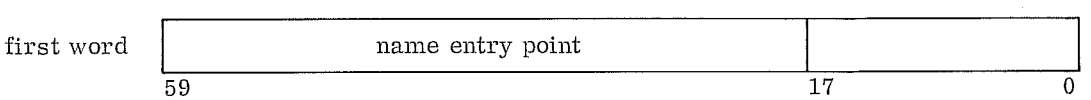
CN 36

LR ignored

L ignored

Words 1 through WC

Each entry in the table is 2 words long. The first word contains the name of the entry point. The second word contains the location of the entry point.



RL	relocation of the address specified by LOC; value is absolute zero.
1	program relocation
$3\text{-}77_8$	relative to common block M, where M is in position LR-2 of LCT. M must not refer refer to blank common.
LOC	address of data word to be modified

TEXT

Text and data tables contain data comprising the subprogram and information necessary for properly relocating the data. The table consists of: an origin for the data, the data itself, and indicators describing relocation (if any) of the three possible locations in a data word which may refer to addresses in memory. TEXT tables may appear in any order and any numbers.

WC must be in the range 2 through 20_8.

Identification word

CN	40_8
LR	relocation of load address (L)
	0 absolute, relative to RA
	1 relative to program origin
	$3\text{-}77_8$ relative to labeled common block M; M is in position LR-2 of LCT. Values of 2 and n, where n refers to blank common, are not permitted.
L	load address. Initial location of data appearing in the second word of the table. L will be relocated using LR.

First Word

Relocation word consists of a series of 4-bit bytes describing the relocation of each of the three possible address references in a 60-bit data word. The first byte (bits 56-59) describes the relocation for the data word in the second word of the TEXT table, etc. The number of relevant bytes and data words is determined by WC. Relocation is relative to program origin or the complement of the program origin (negative relocation). The value and relocation for each byte follows:

000X	no relocation
10XX	upper address, program relocation
11XX	upper address, negative relocation
010X	middle address, program relocation
011X	middle address, negative relocation
1X10	lower address, program relocation
1X11	lower address, negative relocation
0010	same as 1X10
0011	same as 1X11

The above designations permit independent and simultaneous relocation of both upper and lower addresses.

Words 2 through WC

Data words are loaded consecutively beginning at L, their addresses are relocated as specified by the corresponding byte in the relocation word.

Special attention must be paid to code relocated into labeled common, since all addresses are relocated absolute or relative to program origin, never relative to a labeled common block. Addressing relative to labeled common for text must be accomplished through FILL tables.

FILL

The FILL table contains information necessary to relocate previously loaded address fields. References to common are relocated through this table. Program relocation may also be effected using the FILL table, although the usual method (with fewer words) is to use the TEXT table.

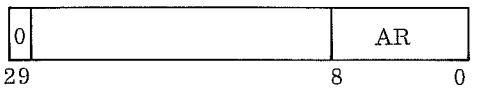
Identification Word

CN	42_8
LR	0
L	0

Words 1 through WC

All remaining words are partitioned into sets of 30-bit contiguous bytes, each set is headed by one control byte and followed by an indefinite number of data bytes. The last byte may be zero. The control byte contains information concerning each of the subsequent data bytes until another control byte is encountered.

A zero byte is treated as a control byte in the format:



AR is the relocation of the value in the address position of a word specified in the succeeding data bytes. AR has the value:

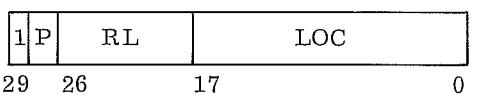
0 absolute, relative to RA (no relocation)

1 program relocation

2 negative relocation

$3\text{-}77_8$ relative to common block M where M is in position AR-2 of LCT.

One control byte suffices for several data bytes. The format of the data byte is:



P Position within word of address specified by RL and LOC.

10 upper

01 middle

00 lower

RL Relocation of address specified by LOC.

RL has the same range of values as AR in the control byte except that 2 and any reference to blank common are illegal.

LOC Address of data word to be modified. The contents of address field position (P) at location LOC relative to RL is added to the origin as specified by AR in the control byte.

LINK

The LINK table indicates external references within the subprogram. Each reference to an external symbol must appear as an entry in LINK.

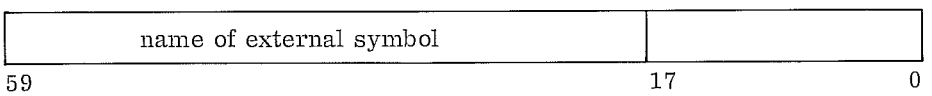
Identification Word

CN 44_8

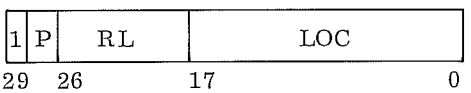
LR ignored

L 0

All remaining words are partitioned into sets consisting of one 60-bit name word and a series of 30-bit contiguous data bytes indicating address positions which refer to the external symbol described in the name word. It is possible for the name word to be split between two computer words.



Names of external symbols (7 characters) must begin with a character for which the display code representation has a high order bit equal to zero. The data bytes have the form:



P Position within the word of the reference to the external symbol:

- 10 upper
- 01 middle
- 00 lower

R Relocation of address specified by LOC

- 0 absolute, relative to RA
- 1 program relocation
- $3\text{-}77_8$ relative to common block M where M is in position RL 2 of LCT

LOC Address of the word containing the reference to the external symbol

REPL - Replication Table

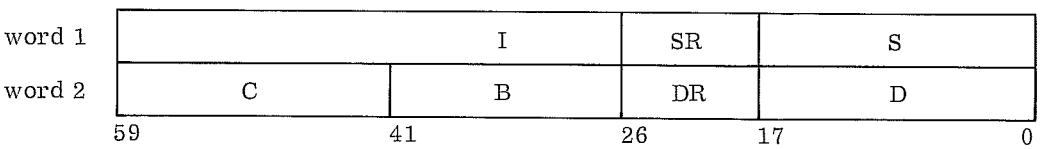
The REPL table permits the repetition of a block of data without requiring one word per location in a TEXT table.

Identification Word

CN 43_8 LR and L are ignored

Words 1 through WC

Each entry in the table consists of two words in the format:



S Initial address of the source data, must be non-zero

SR Relocation of the address specified by S.

0 Absolute, relative to RA

1 Program relocation

$3\text{-}77_8$ Relative to common block M where M is in position SR-2 of

LCT Must not refer to blank common

D Initial address of destination of data

DR Relocation of address specified by D; range of values come as SR-

B Size of data block

C Number of times data block is to be repeated

I Increment to be added to D before each data block is repeated, first repetition of block is at D, second at D+I, etc. The data block (B-long) with origin at S is repeated C times beginning at D the first time, and beginning at the previous origin plus I thereafter.

If C = 0 C is interpreted as 1

If B = 0 B is interpreted as 1

If I = 0 I is interpreted as B

If D = 0 D is interpreted as S+B

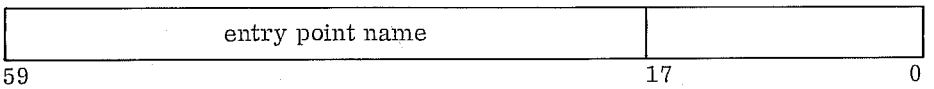
XFER - Transfer Table

The XFER table indicates the end of a subroutine and a pointer address.

Identification Word

CN 46 LR and L are ignored

Bits 0 to 35; an optional date, mmddyy in display code

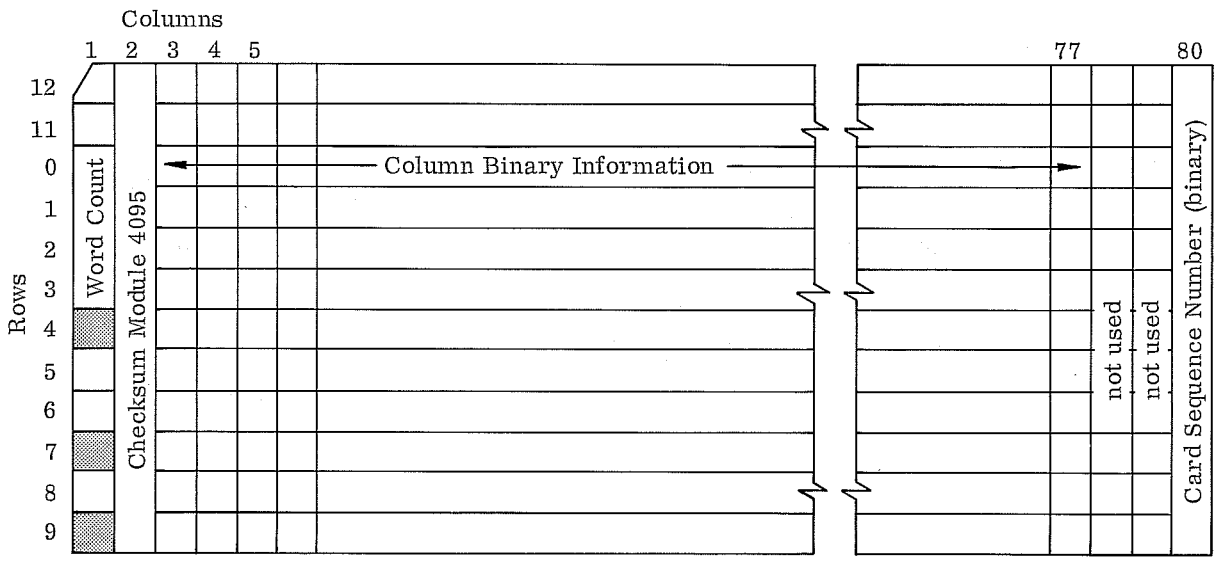


The entry point name need not be in the subprogram. If name is blank, there is no named XFER.

The location of the entry point is returned following a loader request. If a named XFER is encountered prior to an EXECUTE, control is transferred to that entry point. Otherwise, the job is aborted with the comment NO TRANSFER ADDRESS. If more than one subprogram has a named XFER, control is given to the last encountered XFER name.

Column 1

7,8,9	End of logical record
6,7,8,9	End of file
7,9	Binary card
7 and 9 not both in column 1	Coded card



A binary card can contain up to 15 central memory words starting at column 3. Column 1 also contains a central memory word count in rows 0, 1, 2 and 3 plus a check indicator in row 4. If row 4 of column 1 is zero, column 2 is used as a checksum for the card on input; if row 4 is one, no check is performed on input.

Columns 78 and 79 of a binary card are not used, and column 80 contains a binary serial number. If a logical record is output on the card punch, each card has a checksum in column 2 and a serial number in column 80, which orders it within the logical record.

Coded cards are translated on input from Hollerith to display code, and packed 10 columns per central memory word. A central memory word with a lowest byte of zero marks the end of a coded card (it is a coded record), and the full length of the card is not stored if it has trailing blanks. A compact form is thereby produced if coded cards are transferred to another device.

Card Files

All card reading is done by program 2RC, both for normal job input, and for local files that have been assigned to a card reader.

Any punched cards can be read: standard types or free-form cards.

Four types of cards are considered standard:

A card with 0017 octal in column 1 is recognized as an end-of-file marker.

A card with 0007 octal in column 1 is recognized as an end-of-record marker. The level is assumed to be zero unless columns 2 and 3 contain a level number punched in Hollerith form. The level number is read as octal. The following are valid punches (b represents a blank):

00 or 0b	04 or 4b	10	14
01 or 1b	05 or 5b	11	15
02 or 2b	06 or 6b	12	16
03 or 3b	07 or 7b	13	17

Any card other than the above with 7,9 punches in column 1 is assumed to be binary. It must contain 0105, 0205, 0305..... 1605, or 1705 in column 1 and a correct checksum in column 2; or 0145, 0245..... 1645, or 1745 in column 1, in which case column 2 is ignored. The first two digits, 01 or 17, give the word count of the card. Each word occupies 5 columns, and the first word of information begins in column 3. Columns after the last word of information, up to and including column 78, are ignored. The lower 5 bits of column 79, and all 12 bits of column 80 constitute a 17-bit serial number for the card within its record. If the cards of a binary record do not have these numbers in correct sequence (beginning at 1 for the first card), a message is given but the cards are accepted. The checksum is the one's complement of the sum of all information columns, this sum is formed as if in a 12-bit accumulator with circular carry.

Any card that does not have 7 and 9 punched in column 1 is assumed to contain Hollerith-punched information, one 6-bit character per column, or eight 60-bit words per card. Any column that does not contain a valid Hollerith combination is read as a blank, and a message containing the record number and the card number within the record is given. To be a valid Hollerith combination, a column must contain one of the following:

12 and 0, or 11 and 0, and no other punches

or

Not more than one of the punches 12, 11, and 0, with

No additional punch, or any one punch from 1 to 9

or

An 8 punch with one more punch from 2, 3, 4, 5, 6, 7

Binary and Hollerith-punched (coded) cards may be mixed within one record, but a message is given containing the number of any record containing one or more mode changes.

Free-form cards

It is also possible to set up a record of one or more cards that will be read as sixteen 60-bit words per card (80 12-bit columns), with no format checking; except that a card with octal 0017 in column 1 with no other punches, will be interpreted as an end-of-file no matter where it occurs. This rule prevents the separator between jobs in the input stack from being accidentally missed. A card with 0017 in column 1, if it has at least one punch in at least one other column, can be read as 16 words without format control, though normally it would be an end-of-file separator.

Such a special record can be set up as follows:

1. A card with octal 7777 in column 1, and 7777 in column 2, and no other punches. No information is read into memory from this card; it signals that free-form cards will follow.
2. Any number of cards punched in any manner, except that none must be identical to the above card, and none must be an absolute end-of-file card. These cards are read as binary cards, each containing 16 words of information.
3. A card identical with card 1. No information is read into memory from this card; it signals the end of free-form cards. Normal binary or coded cards, or an end-of-record, should follow.
4. An end-of-record card. This is interpreted normally, because free-form reading has terminated.

If it should be necessary for the record of free-form cards to include a card identical to the card described in 1 above, a slightly different card could be chosen to begin and end free-form reading. Any card with octal 7777 in column 1 and in one other column with no other punches is recognized as signalling that free-form cards follow until an identical card or an absolute end-of-file is read. Therefore, there are 79 possibilities to choose from.

A series of free-form cards will normally be organized into one record; however, it can be preceded and/or followed by binary and/or coded cards within the same record. The information will be read but a mode-change message will be issued for the record. Thus a record might validly consist of the following:

1. A series of Hollerith punched cards. Read as 10 words each.
2. A start-free-form card, e.g., 7777 in columns 1 and 80, and no other punches.
3. A series of cards not including an absolute end-of-file, nor any card identical with 2. Read as 16 words each.
4. A card identical with 2, acting as a close-free-form card.
5. A start-free-form card, which might be either the same as or different from 2 and 4.
6. A series of cards not including an absolute end-of-file, nor any card identical with 5. Read as 16 words each.
7. A card identical with 5, acting as a close-free-form card.

8. A series of standard binary cards. Each card contains 1 to 15 words, established by the word count in column 1. A serial number check message would be given unless the serial number in columns 79-80 of each binary card correspond with the position of that card in the record as a whole, not merely with its position in the current group of binary cards.

9. An end-of-record card, closing the record. The record would produce a mode-change message.

Cards can be punched by programs under SCOPE 3.1 in three different formats, corresponding with the modes of reading described above. If the disposition code of an output file is octal 0010, each record will be punched as one or more cards, with 80 valid Hollerith coded characters per card. Unused columns at the end of the last card will be blank, and an end-of-record card will close the record. Such a record could be read in by 2RC without producing any messages. If the disposition code of a file is octal 0012, each record will be punched as a standard-format binary record, which could be read in by 2RC without producing any messages; the serial numbers in columns 79 and 80 of the cards would all be correct. (Since a deliberate mixture of standard binary cards with other types of cards inside a single record is rare, as is the deliberate withdrawal or rearrangement of standard binary cards in one record, the serial number check message will normally indicate that the cards of a standard binary record have been accidentally misarranged.)

If the disposition code of an output file is octal 0014, each record is punched as one or more cards, with 16 words of information, five columns to a word, on each card. If the record does not contain an exact multiple of 16 words, the unused columns on the right of the last card remain blank. An end-of-record card follows the last information card; but such a record probably could not be read in by 2RC. It would be necessary:

- To make certain that the record contained no absolute end-of-file card (0017 in column 1 and 0000 in all other columns). Such a card cannot be read by 2RC as anything but an end-of-file.
- To put a start-free-form card ahead of the first card of the record, and an identical end-free-form card between the last information card and the end-of-record card. The punches for these two cards must not be identical with any of the information cards.

```
                *       SCOPE 3.1 INSTALLATION PARAMETER COMMON FILE                    IPARAMS 00003
0000001          DEBUG   EQU     1                                                      SCOP31D 00001



0000005     IP.CPD    EQU 5                                                           IPARAMS 00005
0000012     IP.DSCC   EQU 12B                                                         IPARAMS 00006
0000001     IP.DSDC   EQU 1                                                           IPARAMS 00007
0000001     IP.DSDE   EQU 1                                                           IPARAMS 00008
0000012     IP.DSTC   EQU 12B                                                         IPARAMS 00009
0000005     IP.DSTE   EQU 5                                                           IPARAMS 00010
0000000     IP.DSTU   EQU 0                                                           IPARAMS 00011
0000001     IP.DTYPE  EQU 1                                                           SCOP31E 00001
0000006     IP.IQD    EQU 6                                                           IPARAMS 00013
0000003     IP.LDEN   EQU 3                                                           IPARAMS 00014
0000011     IP.LJBUF  EQU 11B                                                         IPARAMS 00015
0000003     IP.LVF    EQU 3                                                           IPARAMS 00016
0000001     IP.MCPU EQU 1                                                             SCOP31D 00002
0000000     IP.MECS   EQU 0                                                           IPARAMS 00018
0140000     IP.MFL    EQU 140000B                                                     IPARAMS 00019
0000002     IP.MPR    EQU 2                                                           IPARAMS 00020
0000101     IP.MSCT   EQU 101B                                                        IPARAMS 00021
0077777     IP.MTL    EQU 32767                                                       IPARAMS 00022
0000010     IP.OQD    EQU 8                                                           IPARAMS 00023
0000002     IP.OSW    EQU 2                                                           IPARAMS 00024
0333333     IP.RCYC   EQU 3R000                                                       IPARAMS 00025
0000000     IP.SECS   EQU 0                                                           IPARAMS 00026
0040000     IP.SFL    EQU 40000B                                                      IPARAMS 00027
0000001     IP.SPR    EQU 1                                                           IPARAMS 00028
0000100     IP.STL    EQU 100B                                                        IPARAMS 00029
0000000     IP.TDEN   EQU 0                                                           IPARAMS 00030
            IP.YMD    MICRO 1,3,%MDY%                                                 IPARAMS 00031
0000040     LE.DFB1   EQU 40B                                                         IPARAMS 00032
0000100     LE.DFB2   EQU 100B                                                        IPARAMS 00033
0000100     LE.DFB3   EQU 100B                                                        IPARAMS 00034
0000100     LE.DFB4   EQU 100B                                                        IPARAMS 00035
0000100     LE.DFB5   EQU 100B                                                        IPARAMS 00036
0000100     LE.DFB6   EQU 100B                                                        IPARAMS 00037
0000040     LE.DFB7   EQU 40B                                                         IPARAMS 00038
0000100     LE.DFB0   EQU 100B                                                        IPARAMS 00039
```

```
                0000300    L.RQS     EQU 300B                                                    IPARAMS 00040
                0000000    L.SPI     EQU 0                                                       IPARAMS 00041
                0000100    L.EST     EQU 100B                                                    IPARAMS 00042
                0000600    L.FNT     EQU 600B                                                    IPARAMS 00043
                0000000    L.INS     EQU 0                                                       IPARAMS 00044
                0000000    L.CAT     EQU 0                                                       IPARAMS 00045
                0000003    N.DEVICE  EQU 3                                                       IPARAMS 00046
                0000004    N.RBR     EQU 4                                                       IPARAMS 00047
                           $TABLES$ MACRO                                                        IPARAMS 00048
                             MACRO $R$,Y,Z                                                       IPARAMS 00049
                           R$ SET L.Z+7                                                          IPARAMS 00050
                           T.Y EQU T.Z+R$/8*8                                                    IPARAMS 00051
                           $R$ ENDM                                                              IPARAMS 00052
                           L.CPRES EQU 30B*IP.MECS/IP.MECS+37B                                   IPARAMS 00053
                           L.DFB EQU LE.DFB0+LE.DFB1+LE.DFB2+LE.DFB3+LE.DFB4+LE.DFB5+LE.DFB6+LE.DFB IPARAMS 00054
                           .7+64                                                                 IPARAMS 00055
                           L.RBR EQU N.RBR*38                                                    IPARAMS 00056
                           T.CPRES EQU CP.MOVE                                                   IPARAMS 00057
                           EST $R$ CPRES                                                         IPARAMS 00058
                           FNT $R$ EST                                                           IPARAMS 00059
                           RBR $R$ FNT                                                           IPARAMS 00060
                           RQS $R$ RBR                                                           IPARAMS 00061
                           CAT $R$ RQS                                                           IPARAMS 00062
                           SPI $R$ CAT                                                           IPARAMS 00063
                           INS $R$ SPI                                                           IPARAMS 00064
                           DFB $R$ INS                                                           IPARAMS 00065
                           LIB $R$ DFB                                                           IPARAMS 00066
                           $TABLES$ ENDM                                                         IPARAMS 00067
                0000         $TABLES$                                                            IPARAMS 00068
```

Assigned values are shown in parentheses.

DEBUG (0)
If set to one, designates that MTR, 1SP and 1SQ run in diagnostic mode. Recommended mode during initial testing of PP routines.

IP.CPD (control point delay) (5)
Specifies the interval between recomputation of each control point's priority sublevel. The interval is defined as 2**(IP.CPD+6) milliseconds. IP.CPD must fall in the range, $0 \leq$ IP.CPD ≤ 12; therefore the interval between sublevel recomputation ranges from 64 to 262,144 milliseconds (about 4 minutes).

The following IPARAM symbols are used on the dead-start cards to specify equipment configurations:

IP.DSCC	Card reader channel
IP.DSDC	Channel of device on which the system is to reside
IP.DSDE	Controller number of this device
IP.DSTC	Channel of the tape on which the system tape is mounted
IP.DSTE	Equipment (controller) number of the tapes
IP.DSTU	Unit number of the tape unit on which the system tape is mounted

IP.DTYPE (1)
Allocatable device type on which the system library resides. The values correspond to device type numbers used elsewhere in the system; possible values are:

1 = 6603 disk

2 = 6638 disk

IP.IQD (6)
Determines delay before incrementing sublevels of a job in the input queue.

IP.LDEN (3)
Specifies density for the standard SCOPE 1/2" magnetic tape labels.

3 = 556 bpi

4 = 200 bpi

6 = 800 bpi

IP.LJBUF (11)
Length in octal hundreds of central memory buffer used by READ and the LOAD packages; ranges from 2 upwards.

IP.LVF (3)
Lowest fixed priority level (0 to 7777B). All priority levels greater than or equal to IP.LVF are fixed. Their priority level sublevels will not be recomputed.

IP.MCPU (1)
Installation option to define maximum number of CPU's to be used by the system. Will cause variations in the assembly of MTR.

The value 1 will produce the most efficient code for use on a single CPU. The system will run on a 6500 but will use only one CPU.

The value 2 will produce an MTR which will run on a 6500 using both CPU's or on a 6400 or 6600 using one CPU.

IP.MECS† (0)
Maximum number (0 to 7777B) of 1000_8-word ECS blocks which may be assigned to a single job. The value of IP.MECS is used to determine whether sections of code are to be assembled within the system to handle ECS allocation.

IP.MFL (140000)
Maximum number of central memory words which may be assigned to a single job; from 100_8 to the size of the machine less CMR size.

IP.MPR (2)
Maximum priority (0 to 7777_8) which may be assigned by declaration on the job card.

IP.MSCT (101)
Maximum number of messages (1 to 7777_8) which may be entered into the dayfile by a single job. Only messages sent through MSG are counted.

IP.MTL (32767)
Maximum time limit in seconds (1 to 77777B) which may be assigned to a job.

IP.OQD (10)
Determines delay before incrementing sublevel of a job in the output queue.

IP.OSW (2)
When priority sublevel of a job is re-computed, the old priority sublevel is assigned a weight of (2**IP.OSW)-1 which controls the rate at which a priority sublevel changes when character of job changes. A higher value will cause a slower reaction. Possible values of IP.OSW range from 0 to 6.

IP.RCYC (3R000)
Retention cycle to be used in calculating expiration date for a tape label when no retention cycle is given by the program. Values range from 0 to 999 indicating permanent retention. The address field of the symbol definition contains character data, i.e., 3Rxxx where xxx defines the retention cycle; leading zeros need not be written.

IP.SECS† (0)
Default number of ECS blocks to be assigned to a job if not declared. Ranges from 0 to IP.MECS.

---

†These parameters are presently being developed and checked out.

IP.SFL (40000)
Default central memory field length to be assigned to a job if not declared. Values range from 100_8 to IP.MFL.

IP.SPR (1)
Default priority to be assigned to a job if not declared. Values range from 1 to IP.MTR.

IP.STL (100)
Default time limit in octal seconds (1 to IP.MTL) to be assigned to a job if not declared on the job card.

IP.TDEN (0)
Density of 1/2" magnetic tape when not declared by the user:

0 = 556 bpi

1 = 200 bpi

2 = 800 bpi

IP.YMD MICRO 1,3,MDY (-)
Format of data typed in at deadstart. The six possible permutations of the letters MDY between the dollar signs constitute the possible variation of this parameter. A slash is always displayed as the separator of these fields at deadstart, but any of the display coded characters 50 through 57 will be accepted.

LE.DFBX symbols (-)
Lengths of the individual dayfile buffers. The length of the total dayfile buffer is the sum of LE.DFB0 through LE.DFB7 plus 100_8. The length of the dayfile buffer on the release version of SCOPE 3 is 1000_8. This length may be varied from 360_8 to 1100_8 by varying LE.DFB1 through LE.DFB7. Legal values for the control point dayfile buffers (LE.DFB1 through LE.DFB7) range from 20_8 to 100_8. The system dayfile buffer size (LE.DFB0) may not be varied.

LE.DFB0 (100) Length of system dayfile buffer; must be equal to one sector's length.

LE.DFB1 (40) Length of dayfile buffer for control point 1. Since this control point is typically used for the READ package, no loss is involved in shortening the buffer; READ does not write messages to its own dayfile.

LE.DFB2 (100) Length of the dayfile buffer for control point 2. Since this control point is typically used for the OUTPUT package, no loss is involved in shortening the buffer; OUTPUT does not write messages to its own dayfile.

LE.DFBx ($3 \leq x \leq 6$) (100) Lengths of the dayfile buffers for control points 3-6.

LE.DFB7 (40) Length of dayfile buffer for control point 7. Since control point 7 is typically vacant, no loss is involved in shortening the buffer.

L. CAT† (0)
Length of a table.

L. CPRES (37)
Length of central processor resident program area. L. CPRES is varied automatically depending on the value of IP. MECS, i.e., whether or not there is ECS in the system. If the length of the resident program code assembled in CMR should increase, the IPARAM cards defining the value of L. CPRES must also be changed.

L. DFB (1000)
Sum of LE. DFB0 through LE. DFB7 plus 100B. (See LE. DFBx for instructions on varying L. DFB.)

L. EST (100)
Length of equipment status table (n to 100_8).

L. FNT (600)
Length of the file name table; each entry consists of three words, so the 200_8 files are allowed. The first two entries (six words) are used by SCOPE for the DAYFILE and SYSTEM entries, the next seven for each of the control point dayfiles. These entries are preset when CMR is assembled. Possible values must be determined according to the installation job mix, with the restriction that the FNT lwa may not go beyond 7777_8.

L. INS (0)
Length of a table for installation use. A pointer word is also provided (P. INS).

L. RBR (230)
Automatically assumes the value N. RBR*38 (N. RBR is number of record block reservation tables).

L. RQS (300)
Length of request stack and device status table entries. Possible values range from twice the number of allocatable devices plus two, upwards. For a one-disk system, L. RQS may be made considerably smaller than 300, since only two words are required for the disk DST entry and the number of requests outstanding at any one time seldom rises above six or seven (12_{10} to 14_{10} words).

L. SPI† (0)
Length of a table.

N. DEVICE (3)
Number of allocatable devices present; used when CMR is assembled to compute length of some tables. Must be adjusted when an allocatable device is added or deleted.

N. RBR (4)
Number of record block reservation tables present, used in computation of RBR table lengths.

T. CAT† (3530)
First word address of a zero-length central memory table.

---

†These parameters are presently being developed and checked out.

T.DFB (3530)
First word address of central memory dayfile buffer area.

T.EST (2040)
First word address of central memory equipment status table.

T.FNT (2150)
First word address of central memory file name table.

T.INS (3530)
First word address of a zero-length table reserved for installation use.

T.LIB (4540)
First word address of the library directory.

T.RBR (2760)
First word address of the record block reservation table area.

T.RQS (3220)
First word address of request stack area, the first section of which is the device status table. Actual request stack begins at T.RQS + 2*N.DEVICE.

T.SPI † (3530)
First word address of a zero-length central memory table.

SCPTEXT is a deck which contains a *CALL, CPSYS, the symbol definitions of most of the symbols which used to be in COMFILE (with the exception of the Installation Parameters) and also the various macros which used to be in COMFILE. The symbol definitions have one of the following forms:

```
Name    EQU    absolute expression
Name    =      absolute expression
```

where the = is the shorthand equivalent of the EQU instruction and the absolute expression is either an absolute value or an expression in which all the symbols have previously been assigned an absolute value. Numbers which appear have a decimal default value.

---

† These parameters are presently being developed and checked out.

SCPTEXT SYMBOLS †

CH.FNT (15)
File name table pseudo-channel; possession of this channel provides an interlock of word one of the FNT.

CH.FST (14)
File status table pseudo-channel; possession of this channel provides an interlock of words two and three of the FNT (alternatively called FST).

CH.LIB (16)
Library directory pseudo-channel. This channel is not used by SCOPE 3.1; it is present only because of a historical quirk.

CH.RBT (17)
Record block table pseudo-channel.

CP.ECSM (2034)
Entry point for ECS storage move program. Not used if system has no ECS (IP.MECS=0).

CP.IDLE (2000)
First word address of exchange package for idle program.

CP.JOBN (2020)
Location containing current job count in display code (presently reference address of idle program). Job count is a three-character quantity, left justified in bytes 0 and 1 of CP.JOBN. Assembled as 000 (333333) and updated by 2TJ each time a job enters system. Progression of job numbers is 01-09, 1A-1Z, 10-19,...9A-9Z, 90-99, AA-AZ, A1-A9,...

CP.MOVE (2000)
First word address of exchange package for storage move program.

CP.SM (2023)
Entry point for central processor storage move program.

C.CPAR (1)
Byte within a word in control point area containing the low order portion of the auto recall pointer. Auto recall pointer is a relative address of a word within the program field length; the program remains in recall status until the low order bit of that word becomes 000 (=1), indicating operation completed.

---

† Symbols representing sequential locations in PP low core and the PP routine name in OV.xxx format are omitted in this description.

C.CPECFL (4)
Byte within a word in control point area containing the number/1000B of ECS words assigned to the control point.

C.CPEF (1)
Byte within a word in control point area which contains error flag. If C.CPEF contains zero, there is no error at this control point; otherwise, C.CPEF may contain one of the values defined by the F.x symbols.

C.CPERT (4)
Byte within a word in control point area which contains an internal error flag used by 2TS.

C.CPFL (4)
Byte within a word in control point area which contains central memory field length/100B assigned to this control point.

C.CPFP (3)
Third 12-bit byte position in a CM word—the third byte of the 40th word in control point area. Content of this byte indicates to 1BJ whether the PRU of control card data just loaded is the first.

C.CPFST (2)
Second 12-bit byte within central memory words—second byte of the 40th word in control point area. This byte contains the pointer to the FST entry of the INPUT file assigned to the control point.

C.CPNCSP (4)
Pointer to the fourth 12-bit byte in word 21 of assigned control point area; fourth byte of this word contains pointer to next control statement to be processed.

C.CPNUM (1)
Byte in PP input register word which contains control point number originating request.

C.CPOUT (3)
Pointer to 12-bit byte in CM word—C.CPOUT points to the third byte in word W.CPOUT (153rd word in control point area).

C.CPPRI (0)
References zero byte of the 22nd word in assigned control point area. This byte denotes priority of job at this control point.

C.CPRA (3)
Byte in word in control point area which contains reference address (RA)/100B of control point.

C.CPSM (2)
Byte in a word in control point area which contains storage move flag. MTR sets this field nonzero when storage attached to a control point is to be moved; all PP's operating at this control point should pause if C.CPSM is nonzero.

C.CPSTAT (0)
Byte in a word in control point area which MTR uses to note status of control point (recall, wait, or active) and which PP's are currently assigned.

C.CPTIML (2)
Byte in word in control point area which contains job time limit.

C.DFBMS (1)
Byte in P.DFB which is not now used.

C.DIRCMA (1)
Byte in each word of library directory which contains central memory address of that library routine.

C.DIRFWA (0)
Leftmost of two bytes in P.LIB which contains first word address of library directory.

C.DIRPRU (4)
Byte in second word of a directory entry which contains number of first PRU assigned to record.

C.DIRPTR (0)
Byte within each word of library directory which contains program type and residence.

C.DIRRBA (2)
Byte in second word of a directory entry containing linkage to RBT word pair defining record in which the record starts.

C.DIRRBN (3)
Byte in second word of a directory entry containing original RBT word pair and byte defining block in which record starts.

C.DIRUNT (1)
Byte in second word of directory entry containing physical unit number (DST ordinal).

C.FCIB (0)
Leftmost of two bytes in word 3 of the file name table planned for future system use.

C.FCPNUM (3)
Byte in word 1 of a file name table entry containing file's control point assignment.

C.FCS (3)
Leftmost of two bytes in word 3 of file name table entry containing code and status field.

C.FDC (2)
Byte in word 3 of file name table entry containing file disposition code.

C.FEQP (0)
Byte in word 2 of file name table entry containing file equipment code .

C.FFRBA (1)
Byte in word 2 of file name table entry containing address of first RBT word pair for a file on allocatable device.

C.FLBL (3)
Byte in word 2 of FNT entry. For magnetic tape, byte C.FLBL contains upper 12 bits of current physical record unit (PRU) count; lower 12 bits of PRU count are in byte C.FLBL+1.

For punched cards, byte C.FLBL is 12-bit byte containing upper 5 bits of card count within a record being punched; lower 12 bits of card count are in byte C.FLBL+1.

C.FLOCK (3)
Byte in word 1 of file name table entry containing the lock bit.

C.FLPRU (4)
Byte in word 2 of file name table entry containing current physical record unit (PRU) position of a file on an allocatable device.

C.FLRBEB (3)
Byte in word 2 of file name table entry containing RBT entry and byte at which a file on allocatable device is currently positioned.

C.FLRBWP (2)
Byte in word 2 of file name table entry containing address of current RBT word pair for file on allocatable device.

C.FNAME (0)
Leftmost of four bytes in word 1 of FNT entry containing seven-character file name.

C.FPDEV (2)
Byte in FNT word 2 of tape file entry containing primary device number (EST ordinal).

C.FPRI (4)
Byte in FNT word 1 containing priority of input or output file.

C.FSC (3)
Byte in FNT word 3 containing file security code, indicates whether or not file is open.

D.FSDEV (1)
Byte in FNT word 2 of a tape file entry containing secondary device number (EST ordinal).

C.FTYPE (3)
Byte in FNT word 1 containing file type field.

C.PPFWA (1000)
Location in PP memory at which PP resident is to begin execution of a primary overlay. Used both as the second parameter on IDENT card and in address field of ORG on all primary PP overlays.

C.PPSWA (2000)
Location in PP memory at which to begin execution of a secondary overlay. Used same as is C.PPFWA.

C.PPTWA (3000)
Location in PP memory at which to begin execution of a tertiary overlay. Used same as C.PPFWA.

C.PP4WA (4000)
Location in PP memory at which to begin execution of a fourth level overlay.

C.PP5WA (5000)
Location in PP memory at which to begin execution of a fifth level overlay.

C.PP7WA (7000)
Location in PP memory at which to begin execution of a seventh level overlay.

C.RBRA (3)
Byte in RBR header containing permissible allocation.

C.RBRAD (0)
Byte in the RBR/RBT pointer word containing address of first word.

C.RBRLAV (3)
Byte in second word of each RBR header containing count of unassigned record blocks defined in the RBR.

C.RBRTPA (0)
Byte in first word of each RBR header defining device type referenced.

C.RBRUNT (1)
Byte in word one of RBR header containing DST ordinal.

C.RBTAL (2)
Byte in first word of RBT word pair containing allocation type of file.

C.RBTFB (1)
Byte in first word of RBT word pair containing byte number of first record block address.

C.RBTLPR (0)
Byte in second word of an RBT word pair containing the terminal PRU number + 1 used in last edition of a random file.

C.RBTLRB (4)
References fourth byte of central memory word in record block table. Denotes byte that contains last record block in previous generation.

C.RBTPRU (3)
Byte in first RBT word assigned to each file defining last + 1 PRU assigned to that file.

C.RBTRBR (1)
Byte in first word of RBT word pair containing RBR ordinal for the file.

C.RBTWPL (0)
Byte in each RBT word pair containing linkage to next RBT word pair for that file.

C.RQSCS (0)
Byte in stack pointer word (P.RQS) containing a count of the stack entry word pairs.

C.RQSFS (2)
Byte in stack pointer word (P.RQS) containing address of first stack entry.

C.RSTA (2)
Byte in M.EREQS monitor function containing PP available flag.

C.RSTRA (1)
Byte in M.EREQS monitor function containing request accepted field which is used for monitor-PP resident communication.

C.RSTU (4)
Byte in M.EREQS monitor function contains unit number (DST ordinal).

C.RSTWP (3)
Byte in M.EREQS monitor function containing request stack word pair address.

C.RWPPCC (3)
Byte in READP/WRITEP control word into which stack processor for control phase 1 places channel number to be used in data transmission.

C.RWPPCF (0)
Byte containing phase control flag in control word for READP/WRITEP. Phases: 0 = request in stack, 1 = set channel, 2 = channel set, await transmission, 3 = transmission in progress, 4 = order completed.

C.RWPPLW (2)
Byte in READP/WRITEP control word containing lwa + 1 of data transmitted. It is updated by PP resident during order each time it completes phase 3.

C.RWPPST (3)
In READP/WRITEP control word, operation status available in phase 4 is contained in this byte.

C.RWPPWC (4)
In READP/WRITEP control word, word count for transmission during phase 3 is contained in this byte.

C.RWPPWT (1)
In READP/WRITEP control word, total number of words transmitted during all phase 3's is cumulated by PP resident in this byte.

C.STATCP (2)
Byte used by 3.1 MTR to record current status of CPU ON/OFF/DELEGATION. Contents are nncx where nn = control point address of the delegated control point. The control point number can be obtained by SHN - 7.

c
- 1 CPU x is turned off
- 2 CPU x is delegated to control point nn
- 3 CPU x is not in existence. Locked off

1 = CPUA 2 = CPUB

C.STCPU (4)
Byte in word 1 of stack request containing control point and unit number.

C.STEI (4)
Empty entry indicator in word 1 of a stack entry. If 0, entry is not in use.

C.STFB (3)
Flag byte in word 2 of a stack request.

C.STO (3)
Specific order in word 1 of a stack request; high order 6 bits are a record level number when relevant (order = O.SKF).

C.STPFW (2)
Next address in PP memory for data in word 2 of READP/WRITEP stack request. Used by PP resident to compute byte count and as fwa for data transmission in a call to R.READP or R.WRITEP.

C.STPLW (4)
Last address in PP memory for data in word 2 of a READP/WRITEP stack request. Used by PP resident to compute byte count in call to R.READP or R.WRITEP.

C.STPMS (1)
Location of message buffer of PP in word 2 of a READP/WRITEP stack request. First 3 words are used for communication with stack processor.

C.STPPRU (2)
PRU number at which to begin data transmission in word 1 of a stack request with flag set for no FNT.

C.STPRBA (0)
RBT address of word pair containing record block at which to begin data transmission in word 1 of stack request with flag set for no FNT.

C.STPRBN (1)
RBT ordinal of record block at which to begin data transmission in word 1 of stack request with flag set for no FNT.

C.STPWC (0)
Count of bytes to be transmitted in word 1 of READP/WRITEP stack request.

D.BA (40B)
D.BA through D.BA+4 contain first word of file environment table (FET) located by relative address in low order 18 bits of input register.

D.CPAD (74B)
Typically contains address of control point area currently in use by PP. A primary overlay usually stores the address as part of its initialization.

D.DTS (37B)
High order 6 bits of D.DTS contain device type found in high order portion of byte 0 of second word of FNT. Low order 6 bits of D.DTS contain allocation type found in low order portion of byte 0 of second word of FET.

D.EST (32B)
D.EST through D.EST+4 contain EST entry in process.

D.FA (57B)
Contains address of second word of FNT entry in process.

D.FIRST (60B)
D.FIRST and D.FIRST+1 contain 18-bit CM address specifying beginning of a circular buffer (contents of FIRST pointer (word 2) from a FET).

D.FL (56B)
Central memory field length/100B of control point to which PP is currently attached. Primary overlay usually stores field length as part of initialization.

D.FNT (20B)
D.FNT through D.FNT+9 contain words 2 and 3 of FNT entry for file in process. Words 2 and 3 are referred alternately to as the FST entry.

D.HN (71B)
Constant 100_8: generally, D.HN is preset by a primary overlay for use by a secondary overlay.

D.IN (62B)
D.IN and D.IN+1 contain FET.

D.JECS† (45B)
Used by 2TJ to return ECS field length requirement to calling program.

D.JFL (37B)
Used by 2TJ to return CM field length requirement to calling program.

---

† These parameters are presently being developed and checked out.

D.JPR (46B)
Used by 2TJ to return computed priority to calling program.

D.JTL (47B)
Used by 2TJ to return job time limit (in 8 second units) to calling program.

D.LIMIT (66B)
D.LIMIT and D.LIMIT+1 contain 18-bit CM address specifying lwa + 1 of circular buffer (contents of LIMIT pointer (word 5) from FET).

D.OUT (64B)
D.OUT and D.OUT+1 contain OUT pointer (word 4) from FET.

D.PPIR (75B)
Contains CM address of PP input register. Initialized at deadstart time and must never be altered.

D.PPIRB (50B)
D.PPIRB through D.PPIRB+4 hold the contents of PP input register. Primary overlay usually stores input register contents as part of initialization.

D.PPMES1 (77B)
Contains CM address of first word of PP message buffer. Initialized at deadstart time and must never be altered.

D.PPONE (70B)
Contains constant 1. Generally, D.PPONE is preset by a primary overlay for use by a secondary overlay.

D.PPOR (76B)
Contains CM address of PP output register. Initialized at deadstart time and must never be altered.

D.RA (55B)
Contains central memory reference address/100B of control point to which PP is attached. Primary overlay usually stores address as part of initialization.

D.TH (72B)
Contains constant 1000_8. Generally, D.TR is preset by a primary overlay for use by a secondary overlay.

D.TR (73B)
Contains constant 3. Generally, D.TR is preset by a primary overlay for use by a secondary overlay.

F.ERAR (2)
Value of error flag set for CP arithmetic error abort. Sensed by MTR; error message is written by 2EF.

F.ERCP (4)
Value of error flag set for CP abort. F.ERCP used if CP program aborts execution; program must write a message to dayfile.

F.EROD (6)
Value of error flag set for operator drop type-in. 2EF writes a message to dayfile.

F.ERPCE (5)
Value of error flag set for PP call error abort. Sensed by MTR; error message is written by 2EF. Used when central program requests PP program with name that does not begin with a letter.

F.ERPP (3)
Value of error flag set for PP abort. PP requesting abort is responsible for writing message.

F.ERTL (1)
Value of error flag set for CP time limit abort. Sensed by MTR; error message is written by 2EF.

LE.FNT (3)
Number of central memory words in one FNT entry.

L.CPNUM (7)
Mask of ones equal to the length in bits of highest numbered control point. Definition of this symbol is only an initial action toward varying the number of control points; altering the value of L.CPNUM would have no effect on changing the number of control points.

L.PPHDR (5)
Number of PP words comprising header information appended by assembler. Loading of all PP overlays begins at C.PPxWA minus L.PPHDR.

M.ABORT
(13B-ABORT CONTROL POINT)
(0013,0000,0000,0000,0000)
Job associated with requesting PP is terminated. Requesting processor is responsible for message in dayfile. This function is identical with function M.DPP except that error flag 3 (F.ERPP) is set for abort function in control point area.

M.AEQP (33P)
No longer used.

M.CCPA
(35B-CHANGE CONTROL POINT ASSIGNMENT)
(0035,0000,0000,000C,000N)
Requesting PP is released from current control point assignment in same manner as if it had issued an M.DPP function, except that input register is not cleared. PP is then assigned to control point N with new control point number placed in input register. Stack processor may specify control point number C to indicate that a stack request has been completed and the control point entry count should be reduced by one.

M. CDF
(11B-COMPLETE DAYFILE) (0011,0000,0000,0000,0000)
MTR ignores this function; DSD processes it and clears the output register.

M. CPUST
(36B-CHANGE CPU STATUS) (0036,000X,0000,0000,0000)
Turn CPU X off. X is 1 or 2 to indicate CPU A or B.

(0036,000X,0000,0000,000N)
Delegate CPU X to control point number N. No other control point may use a CPU delegated to control point N. Control point N will be charged for 100 percent of the time that the CPU is delegated whether it is being used or not.

(0036,0000,0000,0000,0000)
If either CPU is off or delegated, it is returned to on status. This does not effect a CPU that was locked off at deadstart load time.

M. DCH
(03-DROP CHANNEL) (0003,00NN,0000,0000,0000)
If channel NN is reserved for requesting PP, reservation will be cancelled. Otherwise MTR will not reply and requesting PP will hang up waiting for its reply.

M. DCP
(16B-DROP CENTRAL PROCESSOR) (0016,0000,0000,0000,0000)
Central processor job at PP control point is dropped; X and W flags are turned off. Auto-recall pointer is not modified. Central program status prior to M. DCP is returned in D. T1. Possible values are:

0 Central program inactive

1 Central program in RECALL status

2 Central program in execution or waiting for central processor

M. DEQP
(23B-DROP EQUIPMENT) (0023,00NN,0000,0000,0000)
MTR drops equipment number NN from control point and updates EST to indicate equipment is free for reassignment. Monitor does not check to insure that equipment number dropped from EST was assigned to this control point.

M. DFM
(01-PROCESS DAYFILE MESSAGE) (0001, 0000, 0000, 0000)
MTR ignores this function; DSD processes dayfile message and clears output register.

M. DPP
(12B-DROP PP) (0012, 0000, 0000, 0000, 0000)
MTR clears PP control point assignment, computes PP running time, adds it to accumulated PP time, provides new PP starting time for subsequent requests, and clears PP input register. If central program is in recall, attempt is made to restart it.

M. DTAPE
(32B-DROP TAPE) (0032, 00NN, 0000, 0000, 0000)
Set lockout (0) bit for equipment NN in EST. Turns equipment logically off; equipment may not be assigned to a control point until it is turned on again.

M. EREQS
(34B-ENTER REQUEST STACK) (0034, 000R, 00AA, SSSS, 0000)
If bits 0-11 of CM word ((C. RQSFS+SSSS)*2) are not zero, the specified word pair is already in use and the request is refused by setting R=1 and clearing byte zero of output register. If word pair is available, stack request is taken from PP message buffer and stored in available word pair. R set to zero indicates that request was accepted. Physical unit number from bits 0-5 of first word of stack entry is used to locate DST entry. Entry count in the DST is increased by one. Bits 6-11 of first word of stack entry are used to identify control point of entry and count (W. CPENC) for that control point is increased by one. If AA is zero, control is returned to the requesting PP normally.

If AA is not 77 or 00, requesting PP is available for reassignment. It is released in the same manner as if a M. DPP were requested. Then, if a PP is not already assigned to that device, the stack processor name from DST is assigned to the PP.

AA = 77 identifies a special request from stack processor to increase entry count in DST and control point area without inserting a new entry in stack. DST and control point to be updated are identified by an entry that is already in stack.

M. NTIME
(14B-TIME LIMIT) (0014, TTTT, 0000, 0000, 0000)
Central processor job time limit of TTTT (8-second) increments is entered at control point. Any previous time limit is superseded.

M. OPDROP
(30B-OPERATOR DROP) (0030, 0000, 0000, 0000)
See M. SEF.

M. PAUSE
(17B-PAUSE FOR RELOCATION) (0017, 0000, 0000, 0000, 0000)
Allows monitor to move central storage for job. As long as move flag at control point is set, this function inhibits any further action of requesting PP. When move flag is cleared, PAUSE function is ended. (See M. RSTOR.)

Requesting processor should check reference address for control point after completion of this function to determine if central storage for job has been moved. (See PP resident function R. PAUSE.)

M. PPTIME
(04-ASSIGN PP TIME) (0004, 0000, 0000, 0000, 0000)
MTR adds current time minus PP starting time, in locations 041-051 of central resident, to accumulated time in control point area. MTR also sets new PP starting time (seconds and milliseconds).

M.RCH
(02-REQUEST CHANNEL) (0002, BBAA, DDCC, 0000, XXXX)

AA	1st choice channel number
BB	2nd choice channel number
CC	3rd choice channel number
DD	4th choice channel number
XXXX	0000 request immediate reply
	0002 = no reply until requested channel has been granted

When channel zero is requested, it must be field AA. When BB, CC or DD is zero, it is assumed that this is not a channel request and that there are no alternate choices beyond it.

When a channel is granted, the number is returned in the PP output register byte one; byte four is set to 7777B.

M.RCLCP
(21B-RECALL CENTRAL PROCESSOR) (0021, 0000, 0000, 0000, 0000)
Central program associated with requesting PP is restarted if central recall flag (X) is set and auto-recall pointer is not pointing to incomplete status.

M.RCP
(15B-REQUEST CENTRAL PROCESSOR) (0015, 0000, 0000, 0000, 0000)
MTR sets central waiting flag (W) at control point and searches for job priorities to initiate central processor action. Ignored under following conditions:

Error flag is set

(RA + 1) = END request.

If auto-recall pointer still points to incomplete status, X flag is set instead of W flag, returning control point to recall status.

M.REM
(25B-ASSIGN ERROR EXIT MODE) (0025, 00NN, 0000, 0000, 0000)
MTR drops central processor execution for job at control point and assigns value N to exit mode field in control point exchange jump information area.

M.REQP
(22B-REQUEST EQUIPMENT) (0022, NNNN, 0000, 0000, 0000)
NNNN is two display-coded characters. If numeric, they directly define equipment request. If alphabetic, MTR searches the EST for an equipment of type NNNN and assignes it to control point. EST is updated and MTR places equipment number in first byte of PP message buffer. If equipment is not available, a zero byte is returned. If availability (Z) byte of EST contains 4000_8, equipment number is returned but availability byte is not modified.

M.RPJ
(37B-REQUEST PERIPHERAL JOB) (0037, DDDD, DDDD, BBBB, 0000)
Peripheral job request is placed in first word of message buffer of requesting PP in exact form in which it is to be placed in input register of PP to which it is assigned.

DDDD, DDDD, is a time delay feature. This 24-bit number represents the number of 250-microsecond delay units requested. This figure is rounded upward to the next larger millisecond and added to the current time. The job is then placed in the PP delay stack to be assigned to a PP at the computed time. The following sample delay values help to compute the proper 24-bit parameter. Accuracy is guaranteed to be within a range of two milliseconds.

0000 0004 = 1 millisecond

0000 0400 = 64 milliseconds

0000 1750 = 250 milliseconds

0000 7640 = 1 second

0072 4700 = 1 minute

6673 0500 = 1 hour

Output register of requesting PP is cleared immediately. When delay parameter is nonzero, BBBB is assumed to be zero.

If BBBB contains output register address of requesting PP, first byte of output register is not set to zero until a PP has been assigned. The reply from MTR contains input register address of assigned PP in place of BBBB. If no PP is immediately available, the reply is delayed while the job request advances through PP job queue. During this delay time MTR forces requesting PP into a pause for storage relocation. Therefore, a request containing BBBB parameter should be followed by R.PAUSE to obtain new RA.

If no space is available in delay stack/PP job queue buffer, entire request is ignored until a space is emptied. However, central processor is denied access to job queue buffer when there are fewer than seven empty spaces.

M.RPP
(20B-REQUEST PP) (0020, 0000, 0000, 0000, 0000)
This function requests initiation of another PP. First word of requesting PP message buffer contains input register data for new PP including control point to which it should be assigned by MTR. Input register address of new PP is placed in first byte of requesting PP message buffer. If no PP is available, a zero byte is returned.

This function should not be used in new programs. It will be replaced by the function M.RPJ.

M. RPRI
(24B-REQUEST PRIORITY) (0024, NNNN, 0000, 0000, 0000)
Assigns priority NNNN to control point, and searches priorities to initiate central processor action.

M. RSTOR
(10B-REQUEST STORAGE) (0010, CCCC, EEEE, 00XX, 0000)
OPTION 1 XX = 00 CM only

XX = 01 ECS only

XX = 02 CM only

Assigns CCCC hundred octal words of central memory and/or EEEE thousand octal words of extended core storage to control point of requesting PP if none of the following conflicts exist.

PP is waiting for previous storage request.

Storage requested exceeds that available in either CM or ECS.

CM is fully allocated and no change is requested; no requested change of ECS will be processed.

Monitor replies to this request by setting CCCC and/or EEEE to the values actually assigned to the control point and by setting byte 0 to zero. These values should be compared with the original requests to determine whether these requests have been honored; either part might have been honored without honoring the other.

Option 2 XX = 10B

Request storage for RBT area of storage. 100 (octal) *CCCC (octal) is address of lowest word memory requested by stack processor. Monitor replies with address actually allocated, which is equal to or greater than address requested. EEEE is ignored.

M. RTAPE
(31B-READY TAPE) (0031, 00NN, 0000, 0000, 0000)
Clear lockout bit for equipment NN in EST. Turns equipment NN logically on.

M. SEF
(30B-SET ERROR FLAG) (0030, 000N, 000E, 0000, 0000)
Drop job at control point N and set error flag E at control point.

M. STEP
(05-MONITOR STEP CONTROL) (0005, 0000, 0000, 0000, 0000)
Initiated by a keyboard request. MTR sets an internal step control flag and at each subsequent request, MTR pauses for console keyboard input. A space from the keyboard causes MTR to process the request. A period from the keyboard causes MTR to process the request and clear the step control flag to resume high speed operation. When N is nonzero, the corresponding control point is the only one to be placed in step mode.

N. CP (7)
Specifies number of control points within the system. Altering the value of N. CP alone has no effect on the system.

O.BPRU (16)
Backspace n PRU's. Number of PRU's to be backspaced is given in third byte of second word of the order. O.BPRU requests repositioning defined by physical rather than logical units. No data is transmitted.

O.RCHN (17)
Release chain. All record blocks assigned to a file and the RBT word pairs containing them are released. FNT is reset to an empty condition if its address is supplied in the order. Requests 16 and 17 require no communication with the device and, therefore, are given highest priority in the search for the next order to be executed. All other requests are assigned priority based on repositioning requirements.

O.RCMPR (2)
Read into central memory after dropping first three CM words of first PRU. Used by STITCH for loading program for system library eliminating the three word header added to system programs by EDITLIB.

O.RDP (10)
Read into requesting PP's memory until a short PRU is encountered or until input area is full.

O.RDPNP (11)
Read into requesting PP after dropping first three CM words of first PRU. Used for all PP system program calls.

O.RDSK (1)
Read into central memory until a short PRU is encountered or until buffer is full. Set FNT to reference first PRU following first end-of-record of level x or greater. Level is given in high-order 6 bits of the order byte.

O.READ (0)
Read into central memory until a short PRU is encountered or buffer is full (IN=OUT).

O.SKB (13)
Skip backward n records of level x or greater. Level is specified in high 6 bits of the order byte; number of records to be skipped is given in third byte of second word of the order. No data is transmitted.

O.SKF (12)
Skip forward n records of level x or greater. Level is specified in high 6 bits of the order byte; number of records to be skipped is given in third byte of second word of the order. No data is transmitted.

O.WRP (14)
Write from requesting PP, full PRU's only.

O.WRPR (15)
Write from requesting PP, ending with a short PRU of level specified in high order 6 bits of order byte. If EOF flag bit is set in this order, a zero length PRU of level 17 is written following short PRU terminating record.

O.WRT (4)
Write full PRU's from central memory.

O.WRTR (5)
Write from central memory, ending with a short PRU of level specified in high order 6 bits of the order byte. If EOF flag bit is found in this order, a zero length PRU of level 17 is written following short PRU terminating record.

P.CAT (11)
Pointer word to be used for future development.

P.CST2 (15)
Address of second word of CST. Contains bytes for channels 1, 5, 11, 15 (CH.FNT), and one unused byte.

P.CST3 (16)
Address of third word of CST. Contains bytes for channels 2, 6, 12, and two unused bytes.

P.CST4 (17)
Address of fourth word of CST. Contains bytes for channels 3, 7, 13, 17 (CH.RBT), and one unused byte.

P.DFB (3)
Address of dayfile buffer pointer word. Only the first byte (byte 0) is used; it contains CM address/10B of dayfile buffer.

P.EST (5)
Address of EST pointer word. Byte 0 contains 12-bit first word address; byte 1 contains 12-bit last word address plus one.

P.FNT (4)
Address of FNT pointer word. Byte 0 contains 12-bit first word address; byte 1 contains 12-bit last word address plus one.

P.INS (7)
Address of a pointer word to an installation area; content is unspecified.

P.LIB (1)
Address of library directory pointer word. Bytes 0 and 1 contain right justified 18-bit first word address of library directory. Bytes 2 and 3 contain right justified 18-bit last word address plus one. Byte 4 contains a deadstart load flag; it must always be zero when a disk or recovery deadstart is attempted.

P.RBR (2)
Address of RBR pointer word. (Also serves as RBT pointer word—see P.RBT.) Bytes 0 and 1 contain right justified 18-bit first word address of RBR table area.

P. RBT (2)
Address of RBT pointer word. (Also serves as RBR pointer word—see P. RBR.) Byte 2 contains first word address/2 of RBT empty chain. Byte 3 contains current length/100B entire RBT area. Byte 4 contains (lwa + 1)/100B of central memory.

P. RQS (13)
Address of request stack area pointer word. Byte 0 contains stack entry word pair count. Byte 2 contains first word address/2 of actual request stack. Byte 3 contains number of allocatable devices (N. DEVICE). Byte 4 contains fwa/10B of DST entries. All DST entries appear at beginning of request stack area, followed immediately by actual request stack.

P. SPI (12)
Pointer word for future development.

P. ZERO (0)
Address of central memory word containing all zeros. Used by most PP routines as a quick means of zeroing five successive PP locations. The system is destroyed by setting contents of P. ZERO to nonzero.

R. CPFL (627)
Location within PP resident containing field length/100B of control point to which PP is attached. R. CPFL is reset each time R. PAUSE routine is entered.

R. CPRA (631)
Location in PP resident containing reference address/100B of control point to which PP is attached. R. CPRA is reset each time R. PAUSE routine is entered.

R. DCH (714)

Calling Sequence:	LOAD	channel number
	RJM	R. DCH
	R. DCH	will cause the specified channel to be dropped.

R. DFM (650)

Calling Sequence:	LOAD	L (message)+flag bits
	RJM	R. DFM

Message from PP memory is written to dayfile and/or console. Flag bits are contained in high order 6-bits of A register upon entry to R. DFM; they determine message destinations. Flag bit values are given below; one or more bits may be on; all are optional.

1 Dayfile only (A display)

2 control point 0 (system) message

4 no A display

R.EREQS (300)
Calling Sequence: Store L (requests) in D.T0

RJM R.EREQS

Adds the control point number to the already formatted request and searches the central memory request stack for an empty entry. The monitor function, M.EREQS, is called and PP resident iterates until the monitor accepts the request.

R.ERQ (334)
A word in R.EREQS modified by LDR.

R.IDLE (100)
Calling Sequence: LJM R.IDLE

Idle loop; PP resident continually scans its input register for something to do.

R.MTR (450)
Calling Sequence: Store function parameters in D.T1 to D.T4

Load function code

RJM R.MTR

Places the function code in D.T0, writes D.T0 through D.T4 to the output register and waits for the output register.

R.OVL (124)
Calling Sequence: Load A register Load Address

RJM R.OVL

Causes an overlay whose name appears in D.T6 and D.T7 (left justified) to be loaded into the PP beginning at the address specified in the A register. R.OVL is used both by PP overlays to load higher level overlays and by PP resident to load the overlay named in the input register. PP resident does not reference the disk directly to load disk resident overlays but makes a call to the stack processor.

R.OVLJ (111)
Calling Sequence: Store name of overlay in D.T6, D.T7

LJM R.OVLJ

Go to R.OVLJ to load a new primary overlay and transfer control to it.

R.PAUSE (430)
Calling Sequence: RJM R.PAUSE

STD D.RA

Exits if PP is attached to control point zero or if storage move flag is not set. Otherwise, the monitor function M.PAUSE is issued and PP pauses until monitor has completed storage move for that control point. In any event, before an exit is made from R.PAUSE, the following information is set:

(D.T0 + C.CPST)	control point status
(D.T0 + C.CPEF)	control point error flag
(D.T0 + C.CPRA)	control point RA (hundreds)
(D.T0 + C.CPFL)	control point FL (hundreds)
A register	control point RA

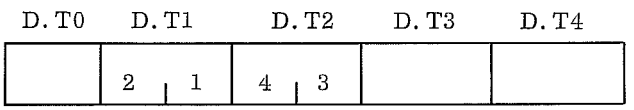
D.RA and D.FL (if significant) should always be reset after a jump to R.PAUSE.

R.PROCES (450)
Identical with R.MTR.

R.RCH (704)
Calling Sequence: Load channel number

RJM R.RCH

Channel numbers in A register are stored in byte D.T1, monitor function M.RCH inserted in D.T0, and D.T0-D.T4 written to output register for that PP. Channels are assigned by MTR on the following priority basis:



If alternate channels are specified, MTR stops looking for alternate channels upon sensing 6 bits of zero. Thus, if one alternate channel is desired, the programmer must clear D.T2 before entering R.RCH so the search terminates at that point. Procedure for requesting channel 12 with alternate channel 13:

```
LDN     0
STD     D.T2
LDC     1312B
RJM     R.RCH
```

Monitor will stop looking for alternate channels after four channels have been investigated.

When R.RCH is used, D.T4 is automatically set nonzero; the function is not considered complete (output register is not cleared) until a channel can be assigned. When complete, byte 0 of output

register is cleared and byte 4 is set to 7777B. A channel request may be made directly to monitor (M.RCH). One other option is allowed in this case: monitor cannot assign the channel, it moves byte 4 of output register to byte 0. R.RCH stores M.RCH in byte 4 of output register to wait until the channel is assigned.

R.READP (R.WRITEP) (460(470))
Calling Sequence: Load L(request)

RJM R.READP(R.WRITEP)

Computes PP word count from first and last word addresses given in already formatted request and adds computed word count, address of PP message buffer, and control point number to request. Request is entered in stack and data is transmitted via channel directly to(from) PP memory. Upon exit from R.READP (R.WRITEP), the following information is set:

(D.T3 + C.RWPPLW)	lwa + 1 of data transmitted
(D.T3 + C.RWPPST)	status
(D.T3 + C.RWPPWT)	number of PP words transmitted

R.RWP (505)
Special entry point to R.READP used by LDR.

R.RWPP (530)
Word in R.READP modified by LDR.

R.STB (620)
Calling Sequence: Load L (list)

RJM R.STB

where list has the form:

L (byte)
L (word 1)
L (word 2)
:
L (word n)

zero

An entry point to R.STB called R.STBMSK is the address of the mask "anded" with each word in the list before the word is "exclusive ored" with the byte. This mask is initially 7700B and this value should be restored by any routine which substitutes an alternate mask. R.STB is used primarily to substitute channel numbers in driver overlays.

R.STBMSK (611)
Address within PP resident of a location used by R.STB routine.

R.TFL (634)
Calling Sequence: Load relative address

RJM R.TFL

Insures that a relative address is within field length; 18-bit address is added to control point reference address (RA) and compared with field length. If address is out of range, R.TFL exits with a negative A register; if address is legal, A register contains absolute CM address (RA + relative address) upon exit. Control point RA and FL are kept locally within PP resident at R.CPRA and R.CPFL, respectively; these locations are reset when an entry to R.PAUSE is made.

R.WAIT (410)
Calling Sequence: RJM R.WAIT

PP idles until byte 0 of output register is clear.

R.WRITEP (470)
See R.READP.

S.DIRPR (8)
Number of bit positions to right shift in PP word to right justify program residence to bit zero.

S.DIRPT (4)
Number of bit positions to right shift in a PP word to right justify the program type to bit zero.

S.FNTEQP (6)
Right offset of equipment code field in FNT. Equipment code field is positioned in bits 6-11 of byte zero.

S.FNTLK (5)
Right offset of lock bit in FNT. Lock bit is positioned in bit 5 of byte 3.

S.FNTTYP (3)
Right offset of file type field in FNT. File type field is positioned in bits 3-4 of byte 3.

S.RBRUNT (6)
Right offset of DST ordinal field in RBR header.

S.RBTRBR (3)
Right offset of RBR ordinal field in RBT word pair.

S.RBTREL (7)
Right offset of release bit in flag field of RBT word pair.

S.RBTRND (6)
Right offset of random bit in flag field of RBT word pair.

S.STF (6)
Right offset of flag field in a stack request.

S.STFA (0)
Right offset (in the flag field) of PP-available bit in a stack request.

S.STFEOF (4)
Right offset (in flag field) of end-of-file bit in stack request.

S.STFETP (2)
Right offset (in flag field) of FET-present bit in stack request.

S.STFNTP (3)
Right offset (in flag field) of FNT-present bit in stack request.

S.STFPRI (5)
Right offset (in flag field) which specifies high priority for this stack request.

S.STFRCL (5)
Right offset (in flag field) of recall bit in stack request.

S.STFREL (4)
Right offset (in flag field) of release bit in stack request.

T.CIDLE (23)
Location of accumulated CP idle time (time spent at control point zero). Also includes time used to move storage.

T.CLK (30)
Location in which clock is kept, in form *HH.MM.SS. T.CLK is updated by MTR and displayed on top line of left scope. Time will be time of day, if a TIME entry to DSD was made; otherwise, it will be the time since deadstart.

T.CPAx (1≤x≤7) (200, 400, 600, 1000, 1200, 1400, 1600)
T.CPAx symbols represent first word addresses of control point areas 1-7. These symbols should be used only by MTR since other PP programs are usually given a control point number from which the address can be computed. Only T.CPA1 is used by MTR as a control point area length.

T.CPS (40)
Used by MTR as a base address to reference CM words 41-51. (PP starting times for PP1-9)

T.CPT1 (56)
Location of current CP status bytes. CPUA status is in byte 4. CPUB status is in byte 3. Status is the control point address of program using CPU, 0000 for idle or 2000 for storage move.

T.CPZ (20)
Address of control point zero status word. Not referenced. MTR uses the word as W.CPSTAT for control point zero.

T.CP8 (2000)
Used by MTR as a limiting address in control point scans and also as alternate name for CP.MOVE and CP.IDLE.

T.CST (14)
Address of first word of CST. Contains bytes for channels 0, 4, 10, 14 (CH.FST), and one unused byte.

T.DATE (31)
Location of today's date in the form entered by operator at deadstart time. Date is displayed on top line of left scope.

T.JDATE (27)
Location of today's date in Julian format. Value is computed from date entered by operator at deadstart time.

T.MON (21)
Address of control point zero job name. Contains display code constant *MONITOR*. It is not used.

T.MSC (40)
Contains the time to the millisecond since deadstart.

byte 0	(not used)
byte 1	time in seconds (reset each 2**12 seconds)
byte 2	milliseconds since updating the seconds
bytes 3-4	time in MSEC (reset each 2**24 MSCC)

T.MSP (37)
Location of monitor step flag used for communication between DSD and MTR while system is in step mode.

T.PLIDLE (24)
Location of accumulated PP idle time (time spent at control point zero). Includes time spent at control point zero by programs such as 1SP while scanning for activity to be pursued. If DSD changes control point assignment, its time will be added here also.

T.PPCx ($1 \leq x \leq 9$) (60, 70, 100, 110, 120, 130, 140, 150, 160)
Symbols represent first word addresses of communication areas for PP's 1-9.

R.PPR (25)
Not used.

T.PPSx ($0 \leq x \leq 9$) (41-51)
Locations of accumulated PP times for PP0-9. These symbols are not used and are deceptive. No time is kept for PP0. Time is kept for PP 1-9 in 41B through 51B. 52B is not used. 51B is only used when DSD changes its control point assignment.

T.SLABx (3x) (where x=1+06)
Locations containing system label displayed on top line of left scope.

T.STATCP (56)
Word containing current status of CPU ON/OFF/DELEGATION. (See C.STATCP.)

T.STO (22)
Not used.

T.TMP (55)
Location of temporary location used by MTR as a scratch pad memory.

T.UAS (56)
Location of unassigned storage length. MTR keeps tally of size of the "holes" between control points in T.UAS; 1BJ uses this size to determine whether or not there is adequate central memory to bring a given job to a control point. This test does not insure that space will be there when a storage request is made, since T.UAS is tested by all copies of 1BJ without an interlock; however, it provides a reasonable figure.

Byte 0 Central memory UAS

Byte 1 ECS UAS

W.CPAR (157)
Relative word in a control point area which contains the auto-recall pointer.

W.CPCAF (51)
Relative first word address in a control point area of the 100_8 word buffer containing current PRU of control card statements.

W.CPCAL (150)
Relative lwa in a control point area of a 100_8 word buffer containing current PRU of control card statements.

W.CPDFM (30)
First of seven words in a control point area containing dayfile message currently on the B display.

W.CPECS (22)
Relative word in a control point area containing ECS field length/1000B assigned to job.

W.CPEF (20)
Relative word in a control point area containing error flag for job.

W.CPENC (155)
Relative word in a control point area containing a count of number of stack requests pending at this control point.

W.CPERT (40)
Relative word in a control point area containing internal error flag used by 1AJ and 2TS.

W.CPFL (20)
Relative word in a control point area containing central memory field length/100B assigned to the job.

W. CPJNAM (21)
Relative word in control point area containing seven-character job name.

W. CPOAE (153)
Relative word in control point area containing byte used to communicate operator-assigned equipment. MTR sets EST ordinal of equipment requested by DSD in this byte for subsequent testing by REQ.

W. CPOUT (153)
Pointer to word 153 in assigned control point area. This word in byte C. CPOUT (3) flags to DSD that the job at this control point is an OUTPUT file.

W. CPPRI (22)
Relative word in control point area containing current job priority.

W. CPRCL (25)
Relative word in control point area containing PP recall register. This word is being replaced with the M. RPJ function with a delay.

W. CPRES1 (156)
Relative word in control point area used by RESPOND.

W. CPRES2 (160)
Relative word in control point area used by RESPOND.

W. CPSM (20)
Relative word in control point area containing storage move flag.

W. CPSTAT (20)
Relative word in control point area containing status byte.

W. CPTBUF (41)
First word of eight-word buffer in control point area containing a partial control card image. The W. CPTBUF buffer is used in event that reading of another PRU of control cards is required and the current PRU does not end on a control card boundary.

W. CPTBUL (50)
Last word of eight-word buffer in control point area containing a partial control card image. (See W. CPTBUF)

W. CPTIME (23)
Relative word in control point area containing CP time accumulated by job.

W. CPTIML (22)
Relative word in control point area containing CP time limit imposed on the job.

W. CPVRNO (154)
Relative word in control point area which transmits visual reel number typed by operator to tape labeling routine.

W.EQP (27)
Relative word in control point area containing bits indicating equipments currently assigned to this control point.

W.FSTCC (151)
Relative word in control point area containing FST entry (FNT word 2) for job input file. Contents of this word designate position on device at which next PRU of control cards may be found.

W.FSTNR (152)
Not used.

W.FTYPE (0)
Relative word in FNT entry containing file type field.

W.PPIR (0)
Relative word in a PP communication area containing PP input register.

W.PPMESx ($1 \leq x \leq 6$) (2-7)
Relative word in PP communication area containing six words PP message buffer.

W.PPOR (1)
Relative word in PP communication area containing PP output register.

W.PPTIME (24)
Relative word in control point area containing PP time accumulated by job.

W.RBRLAV (1)
Relative word in RBRT containing count of record blocks logically available.

W.RBRTPA (0)
Relative word in RBR header containing count of unassigned record blocks defined in RBR.

W.RBRUNT (0)
Relative word in RBR header containing unit number (DST ordinal).

W.RWPPCW (2)
Relative position of READP/WRITEP communication word in message buffer of PP communication area.

W.SSW (26)
Relative word in control point area containing sense switch settings for job.

W.STCPU (0)
Relative word in stack request containing control point and unit number (DST ordinal) of request.

W.STEI (0)
Relative word in stack request containing empty indicator. If this field is 0, the entry is not in use.

W.STFB (1)
Word in stack request which contains flag byte.

W.STO (0)
Word in stack request which contains stack processor order.

W.STPFW (1)
Word in stack request which contains next address in PP memory for data transmission. Field is used in this manner only on calls to R.READP or R.WRITEP.

W.STPLW (1)
Word in stack request which contains lwa + 1 in PP memory for data transmission. Field is used in this manner only on calls to R.READP or R.WRITEP.

W.STPMS (1)
Word in stack request which contains PP message buffer address.

W.STPPRU (0)
Word in stack request which contains PRU number at which to begin data transmission if no-FNT is specified.

W.STPRBA (0)
Word in stack request which contains address of RBT word pair containing record block at which to begin data transmission if no-FNT is specified.

W.STPRBN (0)
Word in stack request which contains RBT ordinal of record block at which to begin data transmission if no-FNT is specified.

W.STPWC (1)
Word in stack request which contains number of PP words (bytes) to be transmitted during a READP or WRITEP request.

SCPTEXT MACROS

LDK Macro
Generates LDN or LDC instruction, depending on size of address field. Any symbols in address field must have been previously defined. This macro is recommended for referencing SCPTEXT symbols for CM pointer words.

ADK Macro
Generates ADN or ADC instruction, depending on size of address field. Any symbols in address field must have been previously defined. This macro is recommended for referencing SCPTEXT symbols for control point additives (W.x symbols).

UJK Macro
Generates UJN or LJM instruction, depending on length of jump. In general, the jump must be backward, since symbols used in address field must have been previously defined. Macro is useful for exiting from small subroutines subject to expansion.

BIT Macro
Generates no code; merely defines a symbol in the location field. Value assigned to symbol is a 1-bit mask where the bit is positioned according to value of address field. Bits are counted from right to left, beginning with zero. Thus, the statement MASK BIT 2 would set MASK equal to 4. Macro is useful for generating 1-bit flag values with the S.x SCPTEXT symbols.

ENM Macro
Generates standard subroutine entry and exit lines. The name of the subroutine is that declared in location field of ENM; the subroutine may be entered by an RJM to that name. If address field of ENM is blank, no exit symbol is defined; otherwise, contents of address field are appended to location symbol to generate subroutine exit symbol. (Typically, address field contains only an X.) An exit from subroutine may then be made by jumping directly to the generated symbol.

PPENTRY Macro
Used as first instruction following ORG in a primary level overlay. PPENTRY generates code to set up low core parameters as follows:

D.PPIRB through D.PPIRB+4	Input register contents
D.CPAD	Control point address
D.RA	Reference address/100B
D.FL	Field length/100B

Address field of the PPENTRY macro should contain: D.PPIRB,T.TO.

LDCA Macro
Load PP A register with absolute 18-bit central address. Relative CM address is obtained from two consecutive PP low core locations, the first of which is specified in address field of LDCA macro; CM address is assumed to be right justified within these two words. Contents of D.RA are added to CM address. Macro is useful for loading many different CM addresses. Space may be conserved by using a subroutine rather that the macro if the same address is to be loaded three or more times.

CRI Macro
Reads contents of a CM word the address of which is contained in a central memory pointer. Address field of CRI macro contains X, Y, and Z subfields, in that order.

X	6-bit CM pointer word address
Y	First of five PP low core cells which will contain the desired CM word
Z	Byte within CM pointer word containing 12-bit CM address of desired word.

JOBCARD Macro

SCPTEXT contains a definition of a macro called JOBCARD. The release version is empty, consisting of a macro definition header and a terminator. System characteristics may be altered by insertion (between header and terminator) of one or more cards described below.

If the symbol SCOPE 2.0 is defined within JOBCARD, SCOPE 3.1 is altered to accept only SCOPE 2.0 job cards. The value to which the symbol SCOPE 2.0 is equated is irrelevant.

SCOPE 3.1 may be altered to accept a decimal value on one or more of the job card parameters by inserting a card or cards in the following form:

DECIMAL field

Field is one of the terms EC, CM, T, or P. Currently, all values are assumed to be octal; however, it may be declared specifically that a parameter is to be interpreted as octal by inserting a card in the following form:

OCTAL field

Field is as defined above.

Priority sublevel computation may be tailored by installation as described below.

This procedure combined with the (installation partitioned) 12-bit priority is used to order jobs within priority levels upon entry to the input queue.

The installation partitions the priority byte by specifying a maximum priority level, IP.MPR. The user-supplied priority value from the job card specifies the high order bits (level) of a job's priority. The other job card characteristics may be used as data for priority sublevel computation algorithm. This algorithm is specified by the installation by inserting a set of statements of the form:

WEIGHT field, relation value additive

Field is one of the terms EC, CM, T signifying ECS storage allocation (in 1000_8 word blocks), CM storage allocation (in 100_8 word blocks) and T is Time limit (in 8 second units).

relation is GE (greater than or equal) or LE (less than or equal)

value is a comparison quantity

additive is to be added to the sublevel if the job card field bears the stated relation to value.

The installation may tailor this algorithm to give high priority sublevel to particular classes of jobs (express type jobs). User specified priority may be relegated to its proper role of distinguishing urgent jobs from the great majority of batch jobs which may enter the system at an installation specified standard level (IP.SPR) and be fanned out along a priority spectrum by the input scheduling procedure.

The CVRT and CVDEL programs are designed to convert an EDITSYM program library plus its correction deck into a source for UPDATE and a correction deck for UPDATE.

CVRT is called to convert a program library:

```
CVRT(COSY,SOURCE)
```

COSY is the file name which contains the EDITSYM program library (COSY is assumed if no name is specified), and SOURCE is the file onto which the source lines will be written (SOURCE is assumed if no name is specified). CVRT processes the COSY file from the beginning until a double end-of-file.

CVDEL is called to convert a correction deck:

```
CVDEL(TAPE1,TAPE2,TAPE3,OUTPUT)
```

the file assignments are:

TAPE1	File which contains the EDITSYM correction deck
TAPE2	File which will contain the UPDATE correction deck
TAPE3	A scratch file for CVDEL
OUTPUT	File which will contain a copy of FORTRAN diagnostics (none expected)

Conversion starts at the current position of TAPE1 and continues until an end-of-record. *COPY, *WEOR and *COMPILE cards are ignored. So that card numbering is consistent, the correction deck which results from the CVDEL run should be run against the UPDATE program library created from the EDITSYM library via CVRT.

The CVRT program, in changing from EDITSYM to UPDATE format, regenerates a source file which is given to UPDATE via a *READ control card. In the EDITSYM numbering scheme, the *DECK cards are not included in the numbering whereas in UPDATE they are; the CVRT program will compensate for this inconsistency by discarding the first card in the EDITSYM deck with an asterisk in column 1 and a blank in column 2. By placing a comments card (asterisk) at an early place in each deck, the numbering of the cards will be unchanged and the CVDEL program will convert corrections properly and consistently.

ERROR MESSAGES H

The system messages produced by SCOPE 3.1 are listed below, in alphabetic order, together with the names of the routines which produce the errors.

Message	Routine
A DOUBLE EOF WAS FOUND BEFORE A1	COPYN
A NUMERIC EXTENDS BEYOND AN END OF FILE	COPYN
A PARAMETER BEGINS BEYOND AN EOF-EOF	COPYN
A PARAMETER IS GREATER THAN 7 CHARACTERS	COPYN
ABOVE IS ILL-FORMED AND IGNORED	EDITLIB

Preceded by reproduction of the control card in question. EDITLIB does not abort, but proceeds to the next control card. Possible reasons for rejection of a control card are given below:

Contains more than 30 elements (words and/or numbers).

First element is not an EDITLIB function.

Any other element exceeds seven characters.

An element begins with two or more digits and contains a letter.

An element which should be a name is a number.

An element which should be a residence code is not CM or DS.

On a SKIPB card, the file name is not followed by a number.

On a SKIPF card, the file name is followed by an asterisk or dash; it should be a name or number.

AN ID(P1) IS REQUIRED ON ALL TEXT CARDS	COPYN
ARG ERROR	LOC

Occurs if the lwa of the area to be cleared (via LOC control card) exceeds the field length or if lwa is less than fwa. The job is aborted.

Message	Routine
ARITH ERROR	2EF

CP program terminated because of arithmetic error.

BAD COMPARE	COMPARE

Displayed on console, system dayfile, and job dayfile if a discrepancy occurs when COMPARE control card is executed. The operator may drop the job if discrepancies are fatal to the job.

BAD NAME CHECK xxx	EDITLIB

Program name xxx read from input file is not same as first program name on ADD, ADDBCD, ADDCOS, or ADDTEXT control card. If running system library is the source, xxx is name of requested program which cannot be found in directory.

BINARY CARD READ AS FUNCTION CARD	EDITLIB

The first record from input file must contain all and only the control cards for EDITLIB. If a character other than a letter, digit, or special characters + - = . , ( ) / * $ and blank appears in this record, the card is assumed to be binary.

BINARY RECORD MISSING FROM INPUT	COPYN
BKSP(TAPE)	1TD, 1DF
BKSP HIT EOF	EDITLIB
BLANK	1BT

1BT was called to write a blank label on a tape.

BLANK COMMON EXCEEDS AVAILABLE CORE, TRUNCATED	LOADER

During blank common allocation, no length is established which would exceed FL or overlap the 20-word LOADER residence. No reference is truncated; only the allocation for core map purposes.

BUFFER ARG ERROR	2BP

FET address is not in field length, or buffer parameters are not within limits given below; the job is aborted.

FET(1) + 4+L < FL	OUT < LIMIT	OUT ≥ FIRST
LIMIT ≤ FL	IN < LIMIT	IN ≥ FIRST

Message	Routine
BUFFER IN/OUT POINTER ERROR	1SX

Called by error code 11. Buffer pointers IN or OUT are less than FIRST or not less than LIMIT.

BUFFER PARAMETER ERROR	1SX

Called by error code 10. FIRST is not less than LIMIT, or LIMIT is greater than field length.

CALL IGNORED, COMMON DECK NOT IN DICTIONARY	EDITSYM

*CALL, dn is encountered but dn is not a common deck.

CANNOT COMPLETE LOAD, JOB ABORTED	LOADER

Intended to correct fatal error conditions which occur in control card mode when LOADER requests LDR to load library routines. Since a request of this type appears to LDR as a user request, 2LE, a fatal error message does not abort, but returns to LOADER. An error message from 2LE appears before this message.

CANNOT COMPLETE THIS OVERLAY, BAD INPUT	LOADER

During overlay generation, if LDR encounters difficulty in loading text or tables, it produces a message such as USER ERROR, BAD TEXT TABLE. When LOADER attempts to complete the overlay, the fatal error bit (set by LDR) is detected, the above message is output, and the job aborted. Core map will contain the last good overlay generated.

CANT (ADDTEXT) FROM SYSTEM	EDITLIB

ADDTEXT reads records from a file and formats them into an overlay for a system file. This is not possible if input file specified is SYSTEM, as it would require running the system as defined by the current CMR directory. A simple ADD can probably get the relevant record from SYSTEM.

CANT ADD WITHOUT READYING	EDITLIB

ADD, ADDBCD, ADDCOS, or ADDTEXT control cards cannot be executed by EDITLIB unless a file has been named by a preceding READY card with no intervening COMPLETE card.

CANT FIND DIRECTORY RECORDS ON INPUT FILE	EDITLIB

EDITLIB cannot locate records that comprise the directory on a system file (should be in 6th and 7th records); presumably this is not a system file.

Message	Routine
CANT LIST BETWEEN READY AND COMPLETE	EDITLIB

EDITLIB cannot list programs in a file while it or another file is being constructed (READY card has been executed more recently than COMPLETE card).

CANT MOVE WHILE READY PENDING	EDITLIB

After a READY card is read by EDITLIB, the residence of a program in the running system cannot be changed by a MOVE card before the file named in READY is written out by a COMPLETE card.

CANT TRANSFER WITHOUT READYING	EDITLIB

EDITLIB cannot execute a TRANSFER card unless a file has been named by a preceding READY card with no intervening COMPLETE card.

CHECKPOINT COMPLETED	CKP
CHECKPOINT REQUESTED	CKP

Every call to CKP enters this message in the dayfile.

CHECKSUM ERROR DEAD-START FROM TAPE	STL

Checksum error occurred during loading from disk. Indicates information on disk is not intact. Load the system from tape.

CIO ERROR. n	1RI

Followed immediately on console display, system and job dayfiles by ROLLIN ABORTED. 1RI, called by a type-in or roll in, is about to abort job. If n=1, file named QROLOUT not at the control point; presumably it had not been properly rolled out. If n=2, 3, or 4, file QROLOUT was there, but did not read correctly (see program 1RI).

CKP FILE INCOMPLETE OR INVALID	RST
CKP FILE UNKNOWN	RST
CKSUM ERROR RC. xxxx, CD. yyyy	2RC

Appears at the beginning of OUTPUT file if input card has invalid checksum and checksum suppress punch in row 4 of column 1. Job will have been aborted as soon as it was brought to a control point. Record number is decimal xxxx, counting first record (control card) in file as 0000; card number is decimal yyyy, counting first card of record as 0001.

Message	Routine
CODED INF	EDITLIB

COMMON DECK EDITING MUST PRECEDE TEXT EDITING — EDITSYM

*COMDECK control card occurred after text deck correction or a *DECK control card in the correction input. EDITSYM run is terminated.

COMMON SECTION TOO LARGE — EDITSYM

Space available for common decks has been exceeded.

COMPARISON ABANDONED BECAUSE OF E-O-R LEVEL DIFFERENCE AFTER RECORD n FILE x LEVEL p FILE y LEVEL q — COMPARE

COMPARE puts this message on its OUTPUT file and ends the run, if the nth pair of records compared in the two files do not both terminate with the same level end-of-record. The level numbers in octal appear after the file names in the message.

((COMPLETE)) FINDS REC.MSG. IN FILE sssssss — EDITLIB

A fault occurred in disk-file use by EDITLIB; sssssss is a local file used by EDITLIB.

CONFLICTING RECORD COUNT EXHAUSTED — COMPARE

COMPARE puts this message on its OUTPUT file and ends the run if the control card specified a number such that if a higher number of record pairs compared were found to conflict, comparison would be abandoned; if that number has been exceeded, (for instance, if the CONFLICT IN RECORD n is written for five pairs of records), the above message is given and comparison is abandoned. If there is no sixth parameter on the control card, it would be done after CONFLICT IN RECORD n has been written 30000 times.

CONFLICT IN RECORD n — COMPARE

COMPARE puts this message on its OUTPUT file when the nth pair of records are not identical, word-for-word. (If one of records is longer than the other, a separate message appears. Word-by-word comparison is made to the end of the shorter record.) Depending on parameters, this message may be followed by listing of some or all words in which the two records differ. For instance,

00020,00000000000000000000/00000000000000000001

The 17th word of record n of the file named first on the COMPARE control card was 0, and the corresponding word in the file named second on the control card was 1. 00020 is octal representation for 17, and the first word of the record would be numbered 00000. The words are printed as 20 octal digits each. Comparison continues normally after the message.

Message	Routine
CONTROL CARD ERROR	COMPARE

Put on the console display, system dayfile, and job dayfile. The COMPARE control card contains fewer than 2 parameters or a parameter that is required to be a number, implied by fewer than 2, contains a non-numerical character. The job is aborted.

CONTROL CARD ERROR	COPYBCD

Displayed on console, system and job dayfiles if a parameter that begins with a digit contains a non-digit character.

CONTROL CARD ERROR	COPYBF
CONTROL CARD ERROR	2TS

Displayed, preceded by card in error.

CONTROL CARD ERROR	ATS

Improper format card.

CONTROL CARD REWIND (INPUT) IS ILLEGAL	COPYN
CONTROL POINT OCCUPIED	1DF

Operator tried to dump dayfile from occupied control point. A control point must be vacant (have no job name).

COPYBCD (DAYFILE, TAPE)	1DF
COPYBCD (xxxxxxx, TAPE)	1TD
COPYL DID NOT FIND xxxxxxx.	COPYL
COPYL DONE	COPYL
COPY REQUESTED BUT NO OLD PROGRAM LIBRARY	EDITSYM

*COPY read from correction input but no old program library requested on EDITSYM call card. EDITSYM run is terminated.

CORE OVERFLOW	UPDATE

Message	Routine
COS	EDITLIB
CPnn COMPARE ERROR	2PC

Displayed on console and system dayfile. Card punch with nn EST ordinal mispunched a card. Operator action not required, as the card punch offsets the bad card and the one after it, and repunches the two cards.

CPnn NOT READY	2PC

Displayed on console only. Card punch with nn EST ordinal is not ready; when ready, punching begins.

CPnn REJECT	2PC

Displayed on console when card punch with nn EST ordinal rejects a function code. No operator action.

CP 00000, 000 SEC	1AJ
CPnn XMSN PARITY ERROR	2PC

Displayed on console and system dayfile when card punch with nn EST ordinal finds a function transmission parity error. No operator action.

CP PROG	EDITLIB
CRnn CKSUM ERROR RC. xxxx, CD. yyyy	2RC

Appears in system and job dayfiles. User local file assigned to card reader with nn EST ordinal contains binary card with bad checksum and no checksum suppress punch in row 4 of column 1. Job is aborted. Record number is decimal xxxx, counting first record as 0000, card number is decimal yyyy, counting first card of record as 0001.

CRnn COMPARE ERROR	2RC

Appears on system dayfile and console display; on the display, it is followed by CR nn RE-READ LAST CARD. The card reader with nn EST ordinal misread last card in output stack and stopped.

Message	Routine

CRnn FORMAT ERROR RC.xxxx, CD.yyyy — 2RC

Appears in system and job dayfiles. User local file assigned to card reader with nn EST ordinal contains 7-9 card, presumably binary, of no recognizable format, and not included in a group of cards for 80-column binary reading. Job is aborted. Record number is decimal xxxx, counting the first record in the file as 0000; card number is decimal yyyy, counting the first card of the record as 0001.

CRnn FUNC XMSN PARITY ERROR — 2RC, 1LJ

Displayed on console and system dayfile. Card reader with nn EST ordinal found function code parity error. No operator action.

CRnn HOLL. CHECK RC.xxxx, CD.yyyy — 2RC

Appears in system and job dayfiles. User local file assigned to card reader with nn EST ordinal contains a card without 7-9 in column 1, presumably Hollerith, with invalid punch combination in at least one column. It has been read as a blank column. Record number is decimal xxxx, counting first record in file as 0000; card number is decimal yyyy, counting first card of record as 0001.

CRnn MODE CHANGE RC.xxxx, CD.yyyy — 2RC

Appears in system and job dayfiles. User local file assigned to card reader with nn EST ordinal found mixed cards (binary, Hollerith, 80-column) in a single record. Record number is decimal xxxx, counting first record as 0000; card number at which first change of type occurred is decimal yyyy, counting first card as 0001.

CRnn NOT READY — 2RC

Displayed on console only. Card reader with nn EST ordinal is not ready. When ready, cards are read.

CRnn REJECT — 2RC, 1LJ

Displayed on console and system dayfiles. Card reader with nn EST ordinal rejected a function code. No operator action.

CRnn RE-READ LAST CARD — 2RC

Displayed on console when CR nn COMPARE ERROR is sent to system dayfile. Card reader with nn EST ordinal misread last card in output stack and stopped. Operator should back up the card, if any, now waiting in the jaws of the reader; back up the last card in output stack to head of input queue and press CLEAR MEMORY and START buttons on the card reader. Card reading will continue.

Message	Routine
CRnn RE-READ 2 CARDS, TYPE GO.	2RC

Follows CR nn 6681 XMSN PARITY ERROR. Both messages will be written on the system dayfile, but only this one will persist on the console display. The operator should: back up the card now waiting in the jaws of the reader, move the last two cards in the output stack back to the head of the input queue of cards, and type x. GO. (x is relevant control point numbers). Card reading will resume. nn is card reader EST ordinal, in octal.

CRnn SERIAL CHK, RC. xxxx, CD. yyyy	2RC

Appears in system and job dayfiles. User local file assigned to card reader with nn EST ordinal has at least one binary card with a serial number that does not agree with its position in the record. Record number is decimal xxxx, counting first record of file as 0000; position of first card in record is yyyy, counting first card in record as 0001.

CRnn 6681 XMSN PARITY ERROR	2RC

Appears on system dayfile and console display, it is succeeded on console by CR nn RE-READ 2 CARDS, TYPE GO. Indicates a transmission parity error between the 6681 and card reader with nn EST ordinal.

DAYFILE	1DF
DAYFILE DUMPED	1DF
DEAD START RECOVERY	STL

Appears in system dayfile when recovery is successfully completed.

DECK DOES NOT END WITH *END	EDITSYM

Text deck ended, but no *END card appeared. Run is terminated.

DECK NOT ON OLD PROGRAM LIBRARY	EDITSYM

Deck specified on *EDIT or *COPY card is not on old program library. Run is terminated.

DECK STRUCTURE CHANGED	UPDATE
DELETION	EDITSYM
DEVICE CAPACITY EXCEEDED IN CKP	CKP

Since there is no provision for multi-reel checkpoint file, this message is given at end of disk or tape.

Message	Routine
DIRECTORY UNDER CONSTRUCTION GETS TOO BIG — TOO MUCH CM RESIDENCE	EDITLIB

EDITLIB is constructing a directory to replace SCOPE system directory in central memory. It is too large for field length of EDITLIB's control point. This field length allows a directory at least 300000B words long. Too many programs are assigned to CM residence. To allow a larger directory, try EDITLIB again with LENGTH(n) as first control card. n is 4-9, calling for 400000B, 1100000B words.

DISK PARITY ERROR	STL

Parity error encountered when loading from disk; loading terminates.

(DISK) 6638 CONNECT FAILED	1SX

Called by error code 02. 1SQ attempt to connect to a unit of the 6638 failed; disconnected alternate unit repeated attempt and failed again.

DISPOSITION CODE ERROR	2TS

Disposition code on REQUEST card is not recognized. Job will terminate.

DOUBLE EOF WAS FOUND BEFORE A /	COPYN

Double EOF encountered before zero length record was copied.

DMP ARG ERROR	DMP

Displayed on console, system and job dayfiles. Starting address for CM dumps is higher than final address or they are relative to RA, rather than absolute, and final address is higher than field length.

DUP COMMON FILES OF xxxxxxx	2TS

Both local and common file in FNT have same name as on COMMON card.

DUPLICATE CKP FTN FOUND	REQ
EDIT CONTROL CARD SET MUST BE FOLLOWED BY A *EDIT CARD	EDITSYM

*INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE control card sets follow a *DECK, *COMDECK, *COPY, *WEOR instead of a *EDIT card. Run is terminated.

EDITION	EDITSYM

Message	Routine
EDITLIB-CTS FAULT IN INITIALIZING COMMON FILES	EDITLIB

Program fault in EDITLIB, or CTS that is not expected.

EDITLIB PROG. FAULT IN SUBRT. MAKE	EDITLIB

EDITLIB is trying to ADD a record which begins like a binary CP program, but is badly formatted. Such a binary program must be organized in tables, and EDITLIB checks organization to get entry point names. If record appears to end in middle of a table, this message is issued.

EDITLIB PROGRAM FAULT I SUBRT. SQB	EDITLIB
EDITSYM CONTROL CARDS	EDITSYM
EDITSYM ERRORS	EDITSYM

Dayfile message.

EDITSYM LIST	EDITSYM
*********END OF FILE*******	EDITSYM

EOF was read on lfn specified by a *CATALOG, lfn control card.

END OF FILE IMPROPERLY READ ON INPUT FILE	EDITLIB

TRANSFER card directed copying one or more records from input file to system file. End-of-file was read before all records were found.

END-OF-INFO ON FILE x AS RECORD y	COMPARE

Program compared y-1 pairs of records from two files, but read end-of-information on x file before specified number of record pairs was reached. Written on job output file, and run is terminated.

END OF TAPE	2RT
ENTER DATE mm/dd/yy	DSD

Appears at deadstart time. Operator must type in current date in specified format before beginning operation.

EOF TAPE MARK READ AT OPEN	1MF

End-of-file tape mark read while positioning logical byte on multifile tape.

Message	Routine
EOR APPEARS BEFORE *END IN *DECK ADDITION	EDITSYM

EOR read before *END in *DECK addition. Run is terminated.

EOR REQUESTED BUT COMPILE FILE NOT REQUESTED	EDITSYM

*WEOR read, but compile file not requested on EDITSYM call card. Run is terminated.

EQ xx NOT READY	2LP
EQ xx PARITY ERROR	1LJ
EQ xx REJECT	2LP

Card reader connect function is rejected.

EQ xx RESERVED	2LP
EQ xx XMSN PARITY ERROR	1LP, 2LP

Transmission parity error during card reader connect function.

EQUIPMENT IS PHYSICALLY UNAVAILABLE	REQ

More equipment is requested than is connected to the system. Job terminates.

ERROR IN LOADING LOADER	LOD

LOADER routine has been destroyed on the system.

ERROR IN JOB FILE READING IN 1BJ	1BJ

1. After reading PRU of control cards, bytes 1 and 2 of first FST word are not identical.
2. Prior to reading PRU of control cards, byte 1 of first FST word currently in CM table does not correspond to byte 1 of FST entry presently stored in word W. FST NR of the control point area.
3. After reading a PRU of control cards, IN-OUT = 0.

Job is aborted.

FET MULTI-FILE POSITION INVALID	1MF

Specified position number is less than first position number on first reel of file.

Message	Routine
FIELD LENGTH NOT SUFFICIENT FOR OVERLAY GENERATION	LOADER

Field length minimum of 26000_8 for the loader plus sufficient additional length to accommodate loader tables and core map.

FIELD IS NON NUMERIC ILLEGAL TEXT CARD	COPYN
FILE DEVICE NOT ALLOCATABLE	IORANDM

Displayed on console, system and job dayfiles. IORANDM was called to read or write beginning of random access record; but FET shows file is assigned to non-allocatable equipment. Job is aborted.

FILE NAME ERROR	2BP

Name in FET does not start with a letter or all characters are not alphanumeric.

FILE NAME ERROR ON EDITSYM CALL CARD	EDITSYM

Parameter error on EDITSYM call card. The first character must be alphabetic. The job is aborted.

FILE x RECORD y HAS PHYS. REC. LONGER THAN 1024 WORDS	COMPARE

Appears on job output file and run ends when record y on file x contains too long PRU. Condition is detected by a PP program (not by COMPARE). Limit accepted by PP program is probably 512 words; but COMPARE buffer limit is 1024 words.

FILE xxxxxx RECORD xxxxxx HAS PHYS. REC. LONGER THAN 1024 WORDS	COMPARE
FILE RECORDS NOT NAMED	IORANDM

Displayed on console, system and job dayfiles. IORANDM has been called to read or write beginning of a random access record, identified by record name. If the record was first written by number it must be addressed by number. The job is aborted.

FL ON NEW JOB CARD INSUFFICIENT	RST
FL TOO SMALL FOR LOADER	LOD

A minimum of about 4000 (octal) CM words is required for CM LOADER and a small relocatable program.

FNT FILLED, DAYFILE DROPPED	2CJ
FNT IS FULL	REQ

Message	Routine
****FORMAT CONTROL SUPPRESSED****	30T

Format control is suppressed by operator type-in. First character of each line is not printed; all succeeding lines are single-spaced.

FORMAT ERROR RC.xxxx, CD.yyyy	2RC

Appears at beginning of job output if a 7-9 card, presumably binary, has no recognizable format and is not included in a group of cards for 80-column binary reading. Job will have been aborted as soon as it was brought to a control point. Record number in input file is decimal xxxx, counting first (control card) record as 0000; the card number is decimal yyyy, counting first card of record as 0001.

FWA-LWA ERROR	IO

Displayed on console display and system job dayfile. A READIN or WRITOUT macro is being executed. The workspace, according to sixth word of the FET, has a negative length. Job terminates.

GOOD COMPARE	COMPARE

Displayed on console display, system and job dayfiles just before run is ended. COMPARE control card is completely executed and no discrepancies found.

HDR REC MISSING ON FILE INCORRECTLY POSITIONED	1MR

Labeled tape file has a missing header. File may be improperly positioned to call to OPEN.

HOLL. CHECK RC.xxxx, CD.yyyy	2RC

Appears at the beginning of job output file. If an input card did not contain 7-9 in column 1, presumably Hollerith, with invalid punch combination in at least one column, which was read as blank column. Record number is decimal xxxx, counting first (control card) record as 0000; card number is decimal yyyy, counting first card of record as 0001.

ID NAME NOT IN INPUT FILES SEARCHED	COPYL
IDLE	1BJ
ILLEGAL ARG TO TIM	TIM

Occurs if request is not for CP elapsed time, today's date, current clock reading, or Julian date; job is terminated.

ILLEGAL COPYL PARAMETER	COPYL
ILLEGAL DEVICE TYPE SPECIFIED	OPE

Specified device not found in table of valid types.

Message	Routine
ILLEGAL EOF	EDITSYM

EOF appears before EOR in common section. Run is terminated.

ILLEGAL EOR OR EOF	EDITSYM

EOR or EOF appears before *END in attempting to copy a deck — Input error. Run is terminated.

ILLEGAL EQUIPMENT REQUEST	1DF

Operator did not request MT, CP, or LP on a request to dump the dayfile.

ILLEGAL FUNCTION LFN	CLO

Function code in the FET is not processed by CLO.

ILLEGAL FUNCTION CODE	CIO

Function code in FET is not allowed in assigned device, the file is closed, or a read was requested immediately following a write on a sequential device.

ILLEGAL FUNCTION REQUEST	CKP

Request is not in recall status at CP or irrelevant bits are nonzero in a specific sequence.

ILLEGAL PARAMETER LIST	CKP

Parameter address is not within field length, or low order bit of parameter first word is ON.

ILLEGAL PRU COUNT AT OPEN REEL	OPE

PRU count not zero or minus at call to OPEN REEL rewind.

ILLEGAL REQ FUNCTION	REQ

Either completion bit was set before REQUEST was processed, or the automatic recall bit was not set.

ILLEGAL REQUEST	2ES

1. Function code is illegal on allocatable device.
2. Erroneous disk address specified for a random read.
3. Read with release issued on a random file.
4. Device type is unrecognizable.

Message	Routine
ILLEGAL SEQUENCE NUMBER ON EDITSYM CONTROL CARD	EDITSYM

Sequence number contains alphabetic or special character.

ILLEGAL TERMINATOR	1DF

Operator did not use period to end dayfile dump request.

INCORRECT OPERATOR ACTION	REQ
INDEX ADDRESS NOT SPECIFIED ON RANDOM FILE	OPE
INDEX ADDRESS NOT INFIELD LENGTH	OPE
INDEX FULL AT OPEN	OPE
INPUT FILE ENDED BEFORE ((ADD)) CARD SATISFIED	EDITLIB

EDITLIB is trying to execute ADD, ADDBCD, ADDCOS, or ADDTEXT programs to a system file; input file ended before last or only program named in control card was found.

INPUT FILE ENDED PREMATURELY	EDITSYM

EOR appears before *END in a *COMDECK addition. Run is terminated.

INPUT REC. FOR ((ADDBCD)) HAS IMPROPER NAME CARD	EDITLIB

EDITLIB is trying to add record to system file. Input record must begin with Hollerith card containing record name in columns 1 ff. Either it does not, or the name is not acceptable.

INPUT REC. MISPREFIXED FOR ((TRANSFER))	EDITLIB

EDITLIB is trying to copy records from input file to new system file. TRANSFER card specified program name and input record had no prefix or a prefix contained different name, or control card did not specify a program name but input record did have prefix.

INVALID CONTROL CARD	LOD

Program call card for a PP program did not meet all requirements: Program name must not exceed three characters; first character must be alphabetic; no more than two parameters.

INVALID DEVICE TYPE FOR OPEN ALTER	OPE

Non-allocatable device type specified.

INVALID DEVICE TYPE FOR OPEN READ	OPE

Used for line printer, card punch, etc.

Message	Routine
INVALID DEVICE TYPE FOR OPEN WRITE	OPE
Used for card reader.	
INVALID FET ADDRESS Nxxxx	2CA
INVALID FILE NAME LFN	CLO
CLOSE or CLOSER request was given for file not in FNT. Illegal function code (22_8) is returned.	
INVALID OPEN PARAMETER	OPE
Z parameter not recognizable.	
INVALID PARAMETER	SCOPE2B
Number of fields in parameter card is in error.	
INVALID RECORD READ	1MF
Invalid record was read where a label was expected while positioning on a multifile tape.	
INVALID REQUEST TO CPC	CPC
CPC has been called to pass a request to monitor. Bits 54-59 of the request word are 0, so that it cannot contain the name of a PP program in bits 42-59. The only other allowable entry in bits 42-59 is a number below 000010B, put there as part of the file action macro. If the number in this field is not below 000010B, displayed on console, system and job dayfiles. Job terminates.	
INVALID STACK ENTRY	1SX
Called by error code 22. Stack entry contains an undefined order code.	
I/O ERROR IN CHECKPOINT	CKP
All other I/O errors are covered by this message.	
I/O ERROR, RST CANNOT BE COMPLETED	RST
JOB CARD ERROR (20 characters)	2TJ
Error on job card when 1BJ called 2TJ. The first 20 characters of the job card are included. Job terminates.	

Message	Routine
JOB FILE READING ERROR IN 1BJ	1BJ
JOB PRE-ABORTED	2RC

Appears at beginning of job output file if input card caused either CHSUM ERROR RC.xxxx, CD.yyyy or FORMAT ERROR RC.xxxx, CD.yyyy. Job aborted as soon as it was brought to a control point.

LABEL MISSING ON DECLARED LABELED TAPE UNIT xx	1MR
LABELED TAPE NOT DECLARED ON MULTIFILE	OPE

Multifile request entry did not contain label parameter.

LEVELS NOT PERMITTED IN STANDARD CALL	LOADER

If S and V bits are not ON in user call, load is interpreted as normal. If L1 and L2 are nonzero, the call may have been intended as overlay or segment call but appropriate bit is missing. Processing continues as for normal load.

LIST OF PROGRAMS IN FILE	EDITLIB
LOAD	1LT

1LT has been called to load jobs from a standard SCOPE 3 tape.

LOADER CONTROL CARD OUT OF SEQUENCE	LOD

NOGO or EXECUTE card before the LOAD card.

LOADX	1LT

1LT has been called to load jobs from external tape.

LOADER NOT FOUND IN LIBRARY	LOD

CM LOADER cannot be found in library directory.

LOADER TABLES GARBAGE, OR OVERLAY SEQ ERROR	LOADER

LOADER tables are threaded in a list below the LOADER in CM. If tables are destroyed by a user program (in normal mode only) and job terminates or if attempt is made to generate secondary overlay without first generating corresponding primary overlay (or zero level) the THREAD routine cannot execute properly.

Message	Routine

LOADER, xxxx ERROR FLAG SET. LDR-2LA

Precedes all error messages issued by LDR or 2LA. (The 2LE overlay is called to issue the message.) xxxx is FATAL or NON-FATAL.

ER1, ER41: xxxx ERROR, CANNOT FIND FILE NAME- LDR-2LA

xxxx is USER or CARD. If LDR was called by LOADER a user call is being processed. ER1 is produced when no file name appears in required parameter. ER41 is produced when a user or system referenced file cannot be found.

ER42: xxxx ERROR, FIELD LENGTH TOO SMALL- LDR-2LA

xxxx is same as ER1. ER42 is detected by 2LA and LDR. Field length for user's program is too small (storage available between starting point in user area and highest available location below GPSL produced tables will not accommodate user's program).

ER43: BAD TEXT- LDR-2LA

Illegal TEXT table entry; relocation code is illegal. No image is produced following the error message.

ER44: FILE INITIALLY POSITIONED WRONG- LDR-2LA

Input file is initially positioned at an end-of-file mark. Image being processed (EOF) is not sent to dayfile; however, the remaining error messages will cause the image being processed to be sent to the dayfile. If the image appears to be binary, it is unpacked and converted to octal display.

ER45: FIELD GREATER THAN 80 CHARACTERS- LDR-2LA

Loader directive is improperly implemented.

ER46: ONLY ONE PARAMETER- LDR-2LA

Overlay loader directive has only one level parameter.

ER47: INVALID CARD FORMAT- LDR-2LA

Produced termination character, . or), permaturely implemented on loader directive. Also produced when word count in a text table greater than 20.

ER50: INVALID LOADER DIRECTIVE- LDR-2LA

First seven characters on a loader directive card do not match one of the following words:

SEGZERO SECTION SEGMENT OVERLAY

Message	Routine

ER51: SEG OR OVERLAY CARD PREV PROCESSED- LDR-2LA

A SECTION card cannot be used in overlay mode. In segmentation mode, it must precede all segment cards. Sent to dayfile when 2LA determines that rules in the above paragraph have not been followed.

ER52: SEGZERO HAS NOT BEEN PROCESSED- LDR-2LA

Appears when a SEGMENT card occurs before the required initial segment card (SEGZERO) has been processed and when a SEGZERO card occurs when OVERLAY card has been processed. (Segments and overlay may not be mixed.)

ER53: SEGZERO SEGMENT NAMES DIFFER- LDR-2LA

Appears on dayfile when the segment names differ for contiguous SEGZERO cards.

ER54: NAME GREATER THAN 7 CHARACTERS- LDR-2LA

Name on a loader directive card is greater than seven characters.

ER55: NO TERMINATION FOUND- LDR-2LA

Sent to dayfile when loader directive card does not have a legal terminator, ., or).

ER56: INVALID CHARACTER- LDR-2LA

Illegal character on a loader directive card.

ER57: SEGMENT OR SECTION CARD PROCESSED- LDR-2LA

ER60: 1ST OVERLAY CARD HAS NO FILE NAME- LDR-2LA

First character of first parameter on initial overlay card is not alphabetic.

ER61: 1ST OVERLAY CARD LACKS 0,0- LDR-2LA

First overlay card does not designate a level (0,0) overlay.

ER62: C OPTION USED ON OVERLAY (0,0)- LDR-2LA

Option to load overlay a designated number of words above blank common is used on a level 0,0 overlay card.

ER63: 1ST PARAMETER MAY NOT EQUAL ZERO- LDR-2LA

Zero level overlay may not have secondary overlay levels (0,1 is illegal).

Message	Routine
ER64: ONLY 1 OVERLAY DESIGNATOR USED-	LDR-2LA

OVERLAY card does not specify both primary and secondary levels.

ER65: C OPTION NOT LAST PARAMETER-	LDR-2LA

Termination character does not follow C option on OVERLAY card.

ER66: TOO MANY CHARACTERS IN PARAMETER-	LDR-2LA

Level number on OVERLAY card is greater than 77_8.

ER67: C OPTION DOES NOT START WITH C-	LDR-2LA

Option which allows user to designate how many words above blank overlay should be loaded, must have C as first character.

ER70: DIGIT IS NOT OCTAL-	LDR-2LA
ER71: TEXT HAS BEEN PROCESSED-	LDR-2LA

SECTION card occurs after text has been processed.

ER73: ERROR IN ABS. OVERLAY FILE FORMAT-	LDR-2LA

Identification code of subroutine does not equal 50_8.

ER74: REQUESTED OVERLAY PROG. NOT FOUND-	LDR-2LA

Overlay program (of given level) cannot be found on the file. Produced by LDR when absolute overlays are being input from a file; by 2LA when absolute overlays are being loaded from a file. File is searched in end-around fashion; it is searched up to EOF mark, file is rewound, then searched up to where searching began.

LOC ARG ERROR	LOC

LOD The following messages can occur only when the CM LOADER is disk resident.

LOD SYSTEM WAIT To load LOADER from disk, LOD references the FNT/FST entry for the file SYSTEM (SSSSSSU if LOADER has been modified by EDITLIB). If LOD finds that SYSTEM (or SSSSSSU) is associated with a control point other than control point zero, it is assumed that EDITLIB is modifying the system. LOD displays this message and pauses until the file SYSTEM (or SSSSSSU) is returned to control point zero.

Message — Routine

LOD ERR 1, BAD EOR, TAKE DUMP† Occurs if end-record status is returned after CM read resulting from improper read of LOADER. In loading LOADER from disk, LOD reads as much of the binary text into CM as will fit in LOADER area after it has been relocated. However, binary text is always longer than the length of LOADER. LOD reads remainder of text into its upper PP memory.

LOD ERR 2, NO EOR, TAKE DUMP† Issued if end-record status is not returned after PP read; end-record must occur if end of LOADER text is read properly.

LOD ERR 3, BIG, TAKE DUMP† Occurs if 3500_8 or more PP words were read. After the PP read, if end-record status is correctly returned, a check is made of the amount of data read.

LOD ERR 4, XFR, TAKE DUMP† In central memory, LOADER is relocated so that the last word is stored at the highest word in the user's field length. Message is issued if the XFER table is encountered before RA+FL-1 has been written.

LOD CANNOT FIND SYSTEM IN FNT — LOD

If a program declares SYSTEM as common file, the control point assignment of SYSTEM changes. LOD uses the FNT entry for SYSTEM when CM LOADER is read from disk. To determine that it has a match on the correct FNT entry, the entry association with control point zero is verified.

LP nn NOT READY — 30T, 2LP

Displayed on console when the printer with nn EST ordinal is not ready. When it becomes ready, printing will continue.

LP nn REJECT — 30T, 2LP

Displayed on console when the printer with nn EST ordinal rejects a function code. No operator action.

LP nn XMSN PARITY ERROR — 30T, 2LP

Displayed on console and system dayfile when printer with nn EST ordinal shows transmission parity error. No operator action of a local file was being printed directly for a job (2LP), message also appears in job dayfile.

† Fatal to the system.

Message	Routine
MESSAGE FORMAT ERROR	MSG

Message contains a character 60_8 or greater. Job is terminated.

MT xx RESERVED	4LB, 2TF, 2TB, 2TW, 2TR

Displayed until condition is corrected or job is dropped if attempt is made to connect to tape xx which is reserved by another channel.

MT xx WPE BAD SPOT	7TP

A record was written successfully following a badspot skip, but a parity error was encountered in re-reading the record preceding the badspot skip. Job can be continued by typing n.GO, but record containing parity error will not be rewritten. Job can be dropped by typing n.DROP.

MT xx WPE RECOVERED	7TP

Write parity error was encountered and recovered; job continues.

MT xx WPE UNRECOVERED	7TP

Write parity error was unrecoverable; job can be continued by typing n.GO or terminated by typing n.DROP.

MT xx WPE WRITE FILE MARK ERROR	7TP

Unable to write file mark on an external tape.

MT xx XMSN PARITY ERROR	4LB, 2TF, 2TB 2TW, 2TR

Transmission parity error in 6681 data channel converter.

MULTI-FILE DISPOSITION ON UNLABELED FILE	REQ
MESSAGE LIMIT	MSG

Number of messages issued by a job exceeds installation declared maximum. Job terminates.

MF DISPOSITION OF UNLABELED TAPE	1RQ

Request card specifies multifile disposition of unlabeled tape.

Message	Routine
MODE CHANGE RC. xxxx, CD. yyyy	2RC

Appears at beginning of output file for job if input record contained cards of mixed format (binary, Hollerith, 80-column). Record number is decimal xxxx, counting first (control card) record of file as 0000; card number at which the first change of type occurred is decimal yyyy, counting first card as 0001.

MOUNT REEL SHOWN ON UNIT xx	1MF
MOVE ROUTINE FINDS CANT READ PROG. IN SYSTEM FILE	EDITLIB

Fault in disk file use by EDITLIB; not expected to occur.

MT xx EOT	CLO

Processing on magnetic tape xx is terminated because of end-of-reel condition or CLOSER request.

MT xx llll BLOCK COUNT SHOULD BE xxxxxx, IS xxxxxx.	4LB

Number of physical records written on tape does not agree with number read. The llll is label ID.

MT xx llll CREATION DATE SHOULD BE xxxxxx, IS xxxxxx.	4LB

Indicates the creation date which should be found in the tape label is not the one found therein. The first xxxxxx is obtained from central memory; the second xxxxxx is taken from the label. Operator may type in n. RECHECK or n. GO or n. DROP.

MT xx llll EDITION NUMBER SHOULD BE xx, IS xx.	4LB

Tape label does not contain expected expiration date. The first number is from FET, the second from label. Operator may provide another tape and type in n. RECHECK or n. GO or n. DROP.

MT xx llll EXPIRATION DATE SHOULD BE xxxxx, IS xxxxx	4LB

Tape label does not contain expected expiration date. First xxxxx is from FET; second is from tape label. Operator may accept tape by typing n. GO; he may mount a different tape and type n. RECHECK, or he may drop the job n. DROP.

MT xx llll FILENAME READ WAS xxxx...xx	4LB

xxxx...xx is label name of first file.

MT xx llll FILENAME SHOULD BE xxx...xx, IS xxx...xx.	4LB

File name error. First 20 characters are FET file name entry and second set are tape label's file name entry; n. GO, n. DROP, or n. RECHECK may be typed.

Message	Routine
MT xx llll FILENAME WRITTEN WAS xxx...xx.	4LB

xxx...xx is the name in the tape label for this file.

MT xx llll MULTIFILE NAME SHOULD BE xxx, IS xxx	4LB

Indicates multifile name of assigned tape does not agree with that requested. First xxx is in FET; second is in tape label. Tape may be accepted, n.GO; different tape may be mounted and checked, n.RECHECK; or job may be dropped, n.DROP.

MT xx llll REEL NUMBER WAS xxxxxx	4LB

xxxxxx is visual reel numbers for tape being read.

MT xx llll REEL NUMBER SHOULD BE xxxx, IS xxxx	4LB

Requested tape reel number does not agree with that specified in tape file label.

MT xx llll REEL NUMBER WRITTEN WAS xxxxxx	4LB

xxxxxx is visual reel number in tape volume header.

MT xx LABEL UNRECOGNIZABLE	CLO

Trailer label of MT xx is not EOF1 or EOV1 as expected. Operator action is requested by additional message.

MT xx ENTER VISUAL REEL NO. ON UNIT xx	4LB

Operator should respond with n.URN, xxxxxx. xxxxxx is the sticker number pasted on tape reel.

MT xx LABEL INFO ERR IN FET	4LB

A numeric field in 33 label area of the FET contains non-numeric data.

MT xx LABEL PARITY ERROR	4LB

Unrecoverable parity error while trying to read tape label.

MT xx NO WRITE ENABLE	4LB, 2TW

Message is displayed until ring is inserted or job is dropped.

Message	Routine
MT xx NOT READY	4LB, 2TF, 2TB, 2TW, 2TR

Message is displayed until unit is readied or job is dropped.

MT xx REJECT	4LB, 2TF, 2TB, 2TW, 2TR

Tape unit xx cannot be connected. Compare xx entry with site provided equipment select code. If possible, change equipment or unit select code; otherwise type, n.DROP.

MULTIPLY DEFINED OUTPUT xxxxxxx

LDR has loaded this routine previously and cannot load it again.

NEXT	1BJ
NO CKP FILE IN FNT	CKP
NO CP AVAILABLE	1LJ

Card punch cannot be assigned to PBC. The PP loops until punch is available or job is dropped.

NO CR AVAILABLE	1LJ

Card reader is unavailable to the READ control point.

NO FNT SPACE	2BP

Parameter in FET is zero and no space is available in FNT to create requested entry; job terminates. If parameter is nonzero, FNT full code (24_8) is returned to FET.

NO INDEX POINTER IN FET, OR 0 LGTH. INDEX	IORANDM

Put on the console display; system dayfile and job dayfile. IORANDM has been called by a READIN or WRITOUT macro to read or write the beginning of a randomly located record. But the FET is either too short to contain pointers to the record index or apparently points to an index of length zero. The job is aborted.

NO INPUT FILE ON THE COPYN CONTROL CARD	COPYN
NO OUTPUT FILE ON THE COPYN CONTROL CARD	COPYN
NO PROGRAM LIBRARY INPUT TO BE EDITED	EDITSYM

Correction input is found but no old program library is given on EDITSYM control card. Run is terminated.

Message	Routine
NO PROGRAMS FOUND FOR SEGMENT	LOADER

Not one of the programs requested by a user call could be found.

NO READY	EDITLIB

A READY card did not precede this control card.

NO REQUEST ENTRY	OPE

No request entry was made for a file with a device type 240. (No FNT entry found in search.)

NO REQUEST ENTRY FOR MULTIFILE	OPE

No request entry was made for specified multifile name (FNT entry not found).

NO ROOM IN INDEX FOR NEW NAME	IORANDM

Displayed on console, system and job dayfiles. IORANDM has been called by a WRITOUT macro to write the beginning of a random access record, identified by name. Name is not already in file index and no vacant slot is available. Job terminates.

NO TRANSFER ADDRESS	LOADER

At completion of a load, no program provides a transfer address. Error condition is overridden by completing load with a NOGO. card or an EXECUTE, ent. card, (ent. specifies an entry point). Job terminates.

NON-EXISTENT RBR REQUESTED	1SP

Called by error code 05. An RBR pointer read in RBT is greater than number of RBR's in system.

ONE OR MORE OVERLAPPING CORRECTIONS	UPDATE
OPEN REEL CALLED ON UNOPENED OR CLOSED FILE	OPE

File must have been opened before OPEN REEL call.

OPERATOR DROP	2EF
OVERLAY	EDITLIB

Message	Routine
OVERLAY CALL FROM RELOCATABLE	LOADER

Relocatable program made a user call for overlay load. A redundancy check is made to validate user calls. Illegal, since once overlay loading is initiated, control is taken away from user call and returned to called overlay. This is possible if file named in user call contains all absolute overlays. If OVERLAY 0,0 has not been called, control will return to the user call.

P2 IS NOT IN THE FILE OR IS UNDEFINED	COPYN
PACKED CARD LONGER THAN 13 WORDS	EDITSYM

Input error - pack 80 column card is longer than 13 words. Run terminates.

PAGE NO.	EDITSYM
PARITY ERROR	2RT
PARITY CHECK FILE x RECORD y	COMPARE

If x is first file named on COMPARE control card, y record cannot be read without a parity check in either mode. If x is second file named, record cannot be read without a parity check in mode corresponding to record of other file already read without a parity check. In either case, comparison of that record pair is abandoned. COMPARE outputs message and continues with the next record pair.

PARITY ERROR IN CHECKPOINT	CKP

Parity error detected in CKP.

PAUSING FOR OPERATOR ACTION	CLO, 1MW, 1MR, 1SX, 1MF

Occurs after error is detected in processing file label. Operator can enter n.GO, n.RECHECK or drop the job; routine pauses for operator decision.

PBC RANGE ERROR	PBC

lwa is outside field length.

POSITIONING A MULTIFILE	1MF
PREVIOUS FILE NEVER CLOSED	OPE
PP CALL ERROR	2EF
PP PROG	EDITLIB

Message	Routine
PP xxxx	1LJ

Amount of PP time used in loading a job from tape.

PROGRAM NOT ON DISK	ATS, EXU
PROGRAM NOT IN SYSTEM	EDITLIB

MOVE or DELETE control card named a program that is not in system file. EDITLIB passes to the next function card.

Ps xxxx KILLED MTR	MTR

s = PP. NO. xxxx = name of program communicating with monitor.

RANDOM CALL, NONRANDOM FILE	IORANDM

Displayed on console, system and job dayfiles. IORANDM called by a READIN or WRITOUT macro to read or write beginning of a random access record. FET shows file is not random. Job terminates.

RANGE ERROR	PBC

lwa exceeds field length; job terminates.

RANGE ERROR	LBC

Record length exceeds field length; job terminates.

RBR RB PRU	1SX

Parity error; device type, half-track, and sector of parity error are printed.

RBT FULL	2ES
READY FOR RESTART	2CA
READY TWICE WITHOUT COMPLETE	EDITLIB

EDITLIB can construct only one system file at a time; therefore, a second file cannot be readied before a COMPLETE card is encountered on the first file.

Message	Routine
RECORD n IN FILE x p WORDS LONGER THAN SAME RECORD IN FILE y	COMPARE

If the nth record read from file x is k+p words long, the corresponding record of file y is only k words long, the first k words of both records are compared normally, and message is put on COMPARE's output file. Comparison continues after skipping extra p words of file x.

RECORD NAME NOT IN FILE INDEX	IORANDM

Displayed on console, system and job dayfiles. IORANDM was called to read beginning of random access record, identified by name which is not in file index. Job terminates.

RECORD NUMBER TOO HIGH	IORANDM

Displayed on console, system and job dayfiles. IORANDM was called to read or write beginning of a random access record, identified by record number, the length of which would fall beyond the end. Job terminates.

RECORD TOO LONG	RBR

End-of-record not reached before field length filled. Job continues.

RECOVERY RISKY	PLR

Displayed if it appears that central memory resident may not be intact. Operator action is awaited. Type GO if recovery is to proceed; type LOAD if system should be reloaded from disk.

REDUNDANT CLOSE LFN	CLO

CLOSE or CLOSER is requested for file in closed status; CLOSE function redundant code (21_8) is returned to FET.

REEL ONE	1MF

A following message requests mounting of reel shown on unit xx.

REPLICATION EXCEEDS AVAILABLE CORE	LOADER

During replication, receiving address is in conflict with loader or its tables.

REQ DEVICE TYPE ERROR	REQ
REQ FUNCTION - DISP CODE ERROR - xxxx	REQ

Disposition code is incorrect or cannot be recognized.

Message	Routine
REQ FUNCTION LFN ERROR	REQ

File name is incorrect, greater than seven characters or does not start with alphabetic character. xxxxxxx is displayed.

REQUEST CARD LFN ERROR - xxxxxxx	REQ

Displayed if file name on request card is incorrect.

REQUEST DISPLAY	DIS

DIS is in HOLD, waiting for operator assignment of display console.

REQUEST lfn	REQ

Displayed when processing the request function or card before the operator has made the assignment. Operator is expected to assign equipment for file lfn. If device type is specified (dt), the proper equipment type must be assigned.

REQUEST FUNCTION - RECALL NOT SET	REQ

Displayed if recall bit of PP input register is not ON when REQ is entered.

REQUEST FUNCTION ADDRESS ERROR	REQ

Displayed if parameter address is not within program field length.

REQUEST LFN ERROR	REQ
REQUEST TAPE	1BT, 1DF
REQUEST TAPE	1LT

In response to this message, operator must type in assigned tape. n.ASSIGNxx,n is control point number and xx is EST ordinal for assigned tape.

REQUESTED SEGMENT INCOMPLETE	LOADER

Results from a user call for same reasons as SEGZERO INCOMPLETE. This message does not result from unsatisfied externals, rather some programs explicitly requested cannot be located.

RESTART CANNOT BE COMPLETED	RST
RESTART COMPLETE	RST

Message	Routine
RESTART CONTROL CARD ERROR	RESTART

RESTART control card has two or more parameters and the second is not CLEAR, or there are no parameters on the card.

RESTART IMPOSSIBLE, SEND RST/CLEAR	RST
ROLLIN ABORTED	1RI

Called by a type-in of ROLLIN, 1RI is about to set control point error flag to 3 and abort job. This message is always preceded by CIO ERROR.n, console, system and job dayfiles.

ROLLIN COMPLETED	1RI

Displayed on console, system and job dayfiles. 1RI has finished rolling in a job, and is about to request CP and drop PP.

ROLLIN INITIATED	1RI

Displayed on console display and system and job dayfiles. 1RI is about to request original field length of control point and start reading file QROLLIN.

ROLLIN STOPPED BY SYSTEM	1RI

Displayed on console, system and job dayfiles. Called by a type-in of ROLLIN. Error flag of control point is already nonzero.

ROLLOUT ABORTED	1RO

Displayed on console, system and job dayfiles. Error occurred in writing file QROLOUT.

ROLLOUT COMPLETED	1RO

Displayed on console, system and job dayfiles. Control point has been successfully rolled out, and its field length is reduced to 100B. 1RO will keep recalling itself until control point is dropped or rolled back in.

ROLLOUT INITIATED	1RO

Displayed on console, system and job dayfiles. Other PP activity appears to have died down, and 1RO is about to write out file QROLOUT.

ROLLOUT STOPPED BY SYSTEM	1RO

Called by a type in of ROLLOUT, error flag of control point is already nonzero.

Message	Routine
ROLLOUT WAITING ON ALL QUIET	1RO

Displayed on console, system and job dayfiles. 1RO is waiting for other PP activity at the control point to cease before writing file QROLOUT.

RST/CLEAR COMPLETED	RST
SEGMENT CALL, BUT NO SEGZERO PRESENT	LOADER

If segment load, called by user, is accomplished from normal or overlay program, it is impossible to establish the necessary delinking point, since certain pointers are established for all segment loading during the initial loading of segzero. Probably results from erroneous setting of S bit.

SEGMENT CALL NOT IN SEGMENT MODE	LOADER

User called for segment load, but first program loaded for this file did not contain SEGZERO card.

SEGMENT REQUEST WHILE IN OVERLAY	LOADER

Loading of segments and overlays is mutually exclusive, although normal loading may be done by both. OVERLOAD has called for loading of a segment; not possible since original pointers to segment tables will not have been preset by the SEGZERO processor.

SEGZERO INCOMPLETE, JOB ABORTED	LOADER

Not all programs specified on SEGZERO card can be found on input file. Job terminates. No error comment is made if not all external references from these programs can be satisfied since this is normal for a segmented job.

SERIAL CHK.RC.xxxx, CD.yyyy	2RC

Appears at the beginning of job output file if a record in input file contained at least one binary card with serial number of which did not correspond to its position in record. Record number is decimal xxxx, counting first (control card) record of file as 0000; position of the first card that showed discrepancy is yyyy, counting first card in record as 0001.

SKIPF FUNCTION MET END OF FILE	EDITLIB

Before a SKIPF request could be satisfied, an end-of-file was encountered.

SL LIST IS EMPTY, FATAL ERROR	LOADER

User call contained SL pointer which gives address at which there is a zero word (vacuuous selective load). Fatal error bit is returned in the user reply.

Message	Routine
SOURCE AND DESTINATION BOTH SYSTEM	EDITLIB

Most recent READY card was READY(SYSTEM); now an ADD card names SYSTEM as source program to be added to current system library.

SOURCE IS ZERO IN REPL	LOADER

No address is given for source stream. Source is then set to location RA+1 (usually contains zero) and replication is initiated.

STOPPED AT IMPROPER RECORD	EDITSYM

Record specified by *CATALOG lfn control card is not common or text section record.

SYSTEM COMMUNICATION ERROR (1SP-1SX)	1SX
SYSTEM ERROR IN LOADER...HELP...CALL CDC	LOADER

LOADER is designed to FAIL SAFE; all communications with the system are checked out and edited. If an interface degenerates or a new bug appears in LOADER, checking can produce this error comment.

SYSTEM COMMUNICATION ERROR (1SP...1SX)	1SX

1SP processed an error code that 1SX does not recognize (usually system error).

TAPE xx NOT DECLARED ON MULTIFILE	OPE

Tape (xx) not ready (2BT, 2RT, 2WT).

TAPE xx PARITY ERROR	PLR, 2BT, 2RT

Parity error in loading system tape. Loading terminates. Also occurs when 2BT encounters parity error during backward motion on 1-inch tape.

TERMINAL NUMBER BEYOND LAST IN DECK	EDITSYM

Number of *INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE card is greater than last sequence number in deck.

TERMINAL NUMBER NOT FOUND, FUNCTION WENT TO NEXT PRIMARY	EDITSYM

Second sequence number on *INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE card does not occur in deck to be edited.

Message	Routine
TEXT AFTER COMMON IN FIRST FILE	EDITSYM

Text decks appear in common section; run is terminated.

TEXT CARD AN ILLEGAL SEPARATOR	COPYN
THIS CARD COULDNT BE OBEYED	EDITSYM

First sequence number on *INSERT, *DELETE, *ADD, *CANCEL, *RESTORE card does not occur in deck to be edited.

THIS REC. NAME ALREADY IN OUTPUT FILE - xxx	EDITLIB

Represents the name of a program EDITLIB has been instructed to add to file xxx; program with same name already in file. ADDCOS does not produce this message, as STITCH programs are not checked for duplication; nor will the existence of a STITCH program already in the file cause the message.

TIME ARGUMENT OUT OF RANGE	TIM

Status word not within field length. Job is terminated.

TIME LIMIT	2EF

Job exceeded requested CP time limit. Job is terminated.

TOO MANY CORRECTION CARDS	EDITSYM

Space available for correction cards for one deck exceeded.

TOO MANY EDITSYM CONTROL CARDS	EDITSYM

Space available for EDITSYM control cards exceeded.

TOO MANY INPUT FILE NAMES ON COPYN	COPYN
TOO MANY NEW COMMON DECKS	EDITSYM

Names of new common decks exceeds space available in CMNAME.

TOO MANY TEXT CARDS IN THE INPUT RECORD	COPYN
TRIED TO ADD PROG. WITH ENTRY PT. WITH DUPL. NAME	EDITLIB

Program to be added to system file contains entry point with same name as an entry point of program already in file.

Message	Routine
TRUNCATED LABEL COMMON xxxxxxx	2LE

Previous declaration of label common is less than present declaration of same label common. Length and address of previous declaration is used.

UNABLE TO OPEN file-name	OPE, 1MF, 1MR

Job is terminated.

UNAVAILABLE	CTS

Displayed on console, system and job dayfiles. PP program CTS called to get common byte assigned to control point. Common file has been assigned, but it belongs to nonallocatable equipment, which cannot be assigned. nn is EST ordinal of equipment, as shown by primary device number in FNT entry.

UNCORRECTABLE PARITY ERROR	1SX

Called by error code 04. Disk read attempted three times (after varying timing margins for 6603 or repositioning heads on 6638); all attempts failed. Read is complete, and information is immediately available Y ≠ go ≠ option is selected. This message also includes identity of the last unreadable PRU encountered.

UNIDENTIFIED ERROR	1SX

Called by any error code other than 2, 4, 5, 6, 10, 11, or 22. Indicates failure of stack processor program or 1SX.

UNEXPIRED SCRATCH TAPE ON UNIT xx	1MW
UNRECOGNIZABLE EDITSYM CONTROL CARD	EDITSYM

Card image at beginning of correction input or following *EDIT card is not EDITSYM control card.

UNRECOVERABLE FILE, RESTART IMPOSSIBLE	RST
UNSAT BIT NOT PERMITTED, EXCEPT IN OVERLAY MODE	LOADER

A single program can be loaded by putting RSL or CLD name in Fn and setting SL=0. To load absolute overlay from library, LOADER must be informed by setting overlay bit and unsatisfied external bit. Setting this bit at any other time will override error flags returned by LDR and could cause LOADER to begin loading absolute overlays on top of proper programs. This condition is prohibited.

UPDATE ERRORS, JOB ABORTED	UPDATE

Message	Routine
UPDATING FINISHED	UPDATE
UPDATING xxxxxxx.	COPYL, UPDATE
WAITING FOR CKP STORAGE	CKP

No storage available for buffer needed by CKP.

WAITING FOR EQUIPMENT	REQ

Displayed when equipment is logically unavailable (all equipment of type requested is assigned and at least one is assigned to a different control point).

WAITING FOR FNT/FST SPACE	RST, 2CA
WAITING FOR STORAGE	1LJ

Occurs if MTR does not immediately grant a request for ECS or CM storage. PP enters recall.

WAITING FOR STORAGE	1LJ

Required storage not available. 1LJ is dropped in recall status.

WAITING FOR STORAGE	2CA, 1OT, 1BT, RST, 1RI, 1DI, 1BJ, RFL
WAITING FOR COMMON FILE	2TS

Displayed under control point assigned to job when common card is processed and file desired is assigned to another control point or is not in FNT.

WAITING FOR COMMON FILE	RST
WAITING FOR PP	RST
WAITING FOR xx	2TS

Displayed when common card for a file of equipment type other than MT, WT, CR, 2CP, or LP is processed and equipment assignment cannot be made.

WAITING ON STORAGE	LDR

Program is being loaded from tape. LDR's request for additional storage has not been fulfilled.

Message	Routine
WARNING BLANK COMMON GREATER THAN PREVIOUS DECL	LOADER

If blank common has been previously established in a set of programs (segment, overlay or a file loaded as result of control card) subsequent references cannot be greater than already allocated. This warning does not terminate job. However, as none of the references to blank common are truncated, it is possible for a program to destroy itself.

WARNING NO MATCHING ENTRY FOR XFER	LOADER

If a nonzero or nonblank entry point occurs in XFER table, all available entries in LOADER table are searched. If a matching entry point cannot be found, this warning is issued. If the warning was resulted from EXECUTIVE or PROGRAM CALL card,job is terminated; otherwise, nonfatal error bit is returned to user.

WRITE WITHOUT FNT REQUESTED	1SX

Called by error code 06. A write order has been read with NO-FNT flag bit set.

xxxx KILLED MTR	MTR
xxx NOT IN PPLIB	PP Resident

Occurs if resident is called to load an overlay for which the name is not in library. Job is terminated.

xx NOT READY	REQ
xx REJECT	REQ
xx RESERVED	REQ
xx XMSN PARITY ERROR	REQ
***ADDFILE INVALID FROM *READ FILE	UPDATE
***ADDFILE INVALID WITH Q-OPTION	UPDATE
***DUPLICATE DECK NAME	UPDATE
***DUPLICATE IDENT NAME	UPDATE
***FIRST CARD MUST BE *DECK OR *COMDECK	UPDATE
***INVALID NUMERIC FIELD	UPDATE

Message	Routine
***NO SUCH COMMON DECK	UPDATE
***NOT ALL MODS WERE PROCESSED	UPDATE
***PREMATURE END OF RECORD ON OLD PROGRAM LIBRARY	UPDATE
***RESERVED FILE NAME	UPDATE
***TERMINAL LIMIT EXCEEDS FIRST LIMIT ON DELETE OR RESTORE CARD	UPDATE
***UNKNOWN IDENTIFIER	UPDATE

SYSTEM SYMBOL DEFINITIONS I

System symbols (for installation parameters, CMR table lengths, locations, pointers, PP resident entry points and monitor functions) consists of three parts:

- Identifier of one or two characters, denoting the group to which it belongs
- A period following the identifier to indicate that it is a system symbol
- Mnemonic of 1-6 characters which suggest the meaning of the symbol

When SCPTEXT and IPARAMS are to be used, the definition of symbols of the above form is to be avoided.

System Symbols	Definition
CH.x	Psuedo-channel numbers
CP.x	Locations within the central processor resident program area
C.x	Byte positions (12-bit) within central memory words; bytes are numbered from left to right as 0-4. x represents the name of a field within the specified byte. For example, a central memory word containing a field called QQQ has been read into PP memory at location fwa. The following instruction would load QQQ into the A register: LDD FWA+C.QQQ
D.x	This symbol form is used when the specified location is either a scratch cell or a location used in a typical manner. Each of the 100_8 core (direct) locations within PP memory is assigned at least one symbolic label in the form: D.mn

m	Locations
Z	00-07
T	10-17
TW	20-27
TH	30-37
FR	40-47
FF	50-57
SX	60-67
SV	70-77

The value of n may range from 0 to 7. Thus location 27_8 is referred to as D.TW7.

System Symbols	Definition
F.x	Error flag values
IP.x	Miscellaneous values which may vary each installation within certain limitations
LE.x	Length of entries within tables
L.x	Table length
M.x	Monitor functions such as values transmitted to MTR through PP resident to cause MTR action
N.x	Quantities
O.x	Stack processor orders (commands). These orders do not correspond to values used in the code and status FET field; stack processor orders are designated for ease of use by the stack processor.
P.x	Locations of central memory pointer words. Most such words contain initial and terminal addresses of tables located elsewhere in central memory.
R.x	Entry points within PP resident
S.x	Right offset of a field within a PP word (byte). The number of bit positions which must be right shifted to right justify the field to bit 0. Six symbols may also be used in the address field of the BIT macro to generate a 1-bit mask.
T.x	First word addresses of central memory tables. When a table has a pointer word, the address of the table should be obtained from the pointer rather than directly from the T.x symbol value. A pointer word has the same name as the table except that the identifier is P rather than T.
W.x	Equated to values representing the relative position of central memory words within tables. For example, if the address of a control point area is contained in the PP A register, the following code would obtain the word containing the job name. ADN W. CPJNAM CRD D. TO

The SCOPE 3.1 program library contains several common decks which are used by one or more of the SCOPE subprograms. These common decks contain symbol definitions, macro definitions, or subroutines which are inserted by UPDATE in any subprogram which contains a card of the form:

*CALL, deck-name

deck-name is the common deck name. The *CALL card is processed by UPDATE and passed intact to the compile file; COMPASS treats such a card as a comment.

The SCOPE 3.1 Program Library common decks are described below:

C.1 1SP1SQ

To minimize duplication in system maintenance, the common code for the routines 1SP and 1SQ has been combined and resequenced into one common deck called 1SP1SQ. This common deck is called by each of the routines 1SP and 1SQ, which are reduced to skeletons containing the calls to SCPTEXT, IPARAMS and several equates which establish the particular character of the calling routine.

C.2 RELOC

The common deck RELOC causes a set of macros to be defined during assembly of a PP program. These macros will cause the assembled code to be relocatable: it may be loaded and executed at addresses other than the origin at which it is assembled. The macro definitions may be added to a PP overlay by the statement:

*CALL, RELOC

The macro definitions are activated by the following statement which replaces the normal entry/exit line of the overlay.

ENTRY INITIAL RELATIVE, BLOCK, SKIPC

Entry is a name to be attached to an LJM 0, the first generated instruction.

RELATIVE is the name of a direct cell which is free during the execution of the overlay. The absolute address of a base point in the generated code is stored in this cell to be used in relocating memory and address constant reference instructions. The two cells following RELATIVE are altered by the generated code at object time, but they may be used by the overlay for any purpose therafter.

BLOCK, if supplied, generates the instruction USE BLOCK. Otherwise the instruction USE (local symbol) is generated. The code following the call to INITIAL will be in the block specified by the use statement until the next use statement is encountered in the normal code. This allows the programmer to force his own code and a portion of the generated code into a specified block. One portion of the generated code, including a table which may be fairly lengthy, is forced into two locally defined blocks which follow all blocks previously declared. This code's space may be reused after the first entry to the overlay via the instruction RJM FWA+1 of the overlay.

SKIPC, if present, specifies that only memory reference instructions are to be modified, and that constant reference commands such as LDC, ADC, etc. are not to be altered. This feature is convenient in the process of conversion to relocatable form, but it should not be used thereafter.

Only RELATIVE is mandatory. The other parameters are optional, although, normally, ENTRY will be used.

The macro replaces the normal definitions of all memory and constant (however, see SKIPC above) reference commands by macro definitions which cause the address fields of such commands to be analyzed. If relocatable, the address field is decremented by a base address. For memory reference commands, if no index is specified, the direct cell address RELATIVE is inserted in the index field. Otherwise, code is generated to add the contents of RELATIVE to the memory cell containing the address field when the overlay is first entered. This latter action is taken for all relocatable addresses in constant reference commands. Although the value of RELATIVE is refreshed at each entry to the overlay, the generated initialization code just described is executed only at the first entry (see comments under BLOCK above). If the address field is not relocatable (a constant, a PP resident entry point) the instruction retains its normal form and no additional code is generated.

Some cautions must be observed in the use of this macro.

- No symbol should be attached to the END card. References to such a symbol are treated as nonrelocatable. The equivalent effect can be obtained by the statements:

```
            USE     ENDBLOCK
SYMBOL      EQU     *
            END
```

  Replacing the statement:

```
SYMBOL      END
```

  (However, see comments under BLOCK above to recover most of the space used by the macro generated code.)

- In an indexed memory reference or constant reference command, a reference to a memory address prior to the base address of the routine (such as SYMBOL–large constant, ENTRY) will result in improper relocation.
- Do not use negative address expressions resulting in negative relocation (-SYMBOL). Negative relocation is not attempted.
- A sum or difference of addresses in blocks other than block 0 is treated as relocatable, whether it is or not.
- Use of the pseudo operation HERE results in code overlapping if relocatable addresses, undefined at time of reference, appear prior to the HERE operation in a block other than the one in use when HERE appears, and the block is next entered with a USE command.
- * is relocatable, except that LJM* is treated specially as LJM*,0 so that the value of * after the initial entry to the overlay is its absolute location.

Two auxiliary macros called VAM and VRM have been defined. They are called by:

```
VAM   expression
VRM   expression
```

If expression is relocatable these calls have the effect of:

```
VFD   expression-base address
```

In addition, VRM is treated in the same way as are constant reference or indexed memory reference commands (a table entry is generated to cause the contents of RELATIVE to be added into the memory cell at first entry to the overlay, giving the absolute address of the symbol). If expression is nonrelocatable, the effect of either macro is:

```
VFD   expression
```

VAM is useful for transfer tables, where the transfer address is to be plugged into an unindexed LJM or RFM address field (use LJM SYMBOL or LJM 0,RELATIVE for the command).

VAM may be used to economize on generated relocation table entries for long lists.

Three other auxiliary macros OPC, OPM, and ANALYZE should not be redefined.

C.3 FNTSRCH

FNTSRCH contains two closed subroutines (FNTSRCH and CHKINP) which may be inserted into a PP program by UPDATE when the following card appears:

```
*CALL,FNTSRCH
```

FNTSRCH searches the file name table for a file with a specified name, type, and control point. FNTSRCH uses the subroutine CHKINP during the course of the search.

Procedure:

- Store the seven-character file name to be located in D.TW5 through D.TH0, left justified with zero fill.
- Store the file type in bits 3-4 and the control point number in bits 0-2 of D.TH1. The remainder of D.TH1 should contain zero.
- Execute a return jump to FNTSRCH.

If the file is not found, FNTSRCH exits with D.FNT containing zero. Otherwise, the following locations are set:

D.FNT through D.FNT + 9 - FNT words 2 and 3

D.FA - address of FNT word 2

The file is set to busy status to prevent any other use.

CHKINP is referenced by FNTSRCH, but may also be used separately to compare the current FNT entry with a file-specified name, type, and control point.

Procedure:

- Store the seven-character file name to be compared in D.TW5 through D.TH0, left justified with zero fill.
- Store the file type in bits 3-4 and the control point number in bits 0-2 of D.TH1. The remainder of D.TH1 should contain zero.
- Store the address of the current FNT entry (word 1) in D.TW0.
- Execute a return jump to CHKINP.

Upon exit, the A register indicates whether or not the FNT entry agrees with the specified file.

A = 0 FNT entry = specified file

A $\neq$ 0 FNT entry $\neq$ specified file

D.T0 through D.T4 are used to read in word 1 of the current FNT entry.

C.4 BUFINIT

BUFINIT contains a closed subroutine (BUFINIT) which may be inserted into a PP program by UPDATE when the following card appears:

```
*CALL,BUFINIT
```

BUFINIT may be used to format a 10_{10} PP word area (REQ) which becomes a stack request entry. The routine assumes that this area is defined elsewhere in the program and that the first word is named REQ. It is also assumed that fields other than those listed below are set elsewhere in the program.

Procedure:

- Store word 2 of the FNT entry for the file to be processed in D.FNT through D.FNT + 4.
- Execute a return jump to BUFINIT.

Upon exit from BUFINIT, the following fields are set into the stack request (REQ) buffer:

Unit number (DST ordinal)

Control point number

Current disk address from FNT entry (current word pair address, RBT ordinal, and PRU)

DX6 through D.T0 are used as scratch locations.

C.5 READPP

READPP contains a closed subroutine (READPP) which may be inserted into a PP program by UPDATE when the following card appears:

*CALL, READPP

READPP may be used to read data from an allocatable device through the stack processor and into PP memory. The stack request must already be formatted in a 10_{10} PP word buffer, the first location of which is called REQ. D.FNT through D.FNT + 9 is assumed to contain words 2 and 3 of the FNT entry for the file being read.

The following conditions pertain when an exit is made from READPP:

- The PP buffer in the stack request is either full (to the nearest PRU) or an end-of-record was read. In either case, the PP last word address + 1 of data read is in D.IN.
- The FNT entry (in D.FNT through D.FNT + 9) is updated to reflect the status of the completed request and the new position of the file.
- The stack request (in REQ through REQ + 9) is updated to reflect the new position of the file. The PP first word address given in the REQ buffer is reset to the value BUFFER. BUFFER is assumed to have been defined elsewhere into the program.

When proper conditions are set up, READPP may be re-entered without further initializing.

READPP may also be used to write (O.WRP) or write record (O.WRPR) from PP memory.

A FORTRAN program compiled under version 2.0 in binary form may experience incompatibility with version 2.3 FORTRAN if the minimum buffer size was specified. With 2.0 the minimum buffer size is 1001_8 words plus 10_8 words for FET, or a total of 1011_8 words; under 2.3 the minimum buffer size is again 1001_8 but the FET size is 21_8 words, for a total of 1022_8 words. Such a FORTRAN program in binary form using the minimum size buffer will have had only 1011_8 words allocated for each file; however, when run under version 2.3, the FET uses 21_8 words leaving 770_8 words for the buffer. This is not enough buffer to contain a PRU for a binary tape, although it is sufficient for BCD tape files or disk file. If a file with the minimum buffer size is assigned to a binary tape, the job will be aborted with the message:

BUFFER NOT BIG ENOUGH TO HANDLE BINARY TAPES.
RECOMPILE UNDER 2.3 COMPILER.

The solution is to recompile the program under the 2.3 compiler or, alternately, to use the program SCOPE2B.

SCOPE2B generates a labeled common block, named SCOPE2, large enough to contain the FET's for all files in the FORTRAN program. Q8NTRY then places the FET's in this common block and uses for just the buffers the combined storage previously allocated for both the FET's and the buffers. Thus, the full 1011_8 words will be used for the buffer and 21_8 words in the common block will be used for the FET. All files in a given program are treated in this manner under SCOPE2B. The procedure cannot be applied selectively.

SCOPE2B is called from a control card as follows:

SCOPE2B (n,lfn)

n = number (octal) of files specified on the PROGRAM card of the main program

lfn = name of file to receive the loader tables generated for the common block

If no parameters or more than two parameters are specified, the job will be aborted with the message:

BAD PARAMETER.

n must be specified; if lfn is not specified, LGO is assumed. n must be at least as large as the number of compiled file names or the job will be aborted by Q8NTRY with the message:

/SCOPE2/TOO SMALL

If n is one or two digits, it will be accepted as is. If it is more than two digits, only the leftmost two will be used. SCOPE2B will write a PIDLE table and a TEXT table on lfn completely specifying a labeled common block SCOPE2 of length 21_8*n. However, the user is responsible for loading it from lfn. The following examples illustrate some of the possibilities.

1. Normal Loading

   REQUEST TAPE.

   TAPE.

   might become

   REQUEST TAPE.

   SCOPE2B (5,X)

   LOAD (X)

   TAPE.

   A common block of 125_8 words is loaded at RA+100_8 and the original program is loaded starting at RA+225_8.

2. OVERLAY Loading

   The common block SCOPE2 must be made a part of the (0,0) overlay, so the overlays must be regenerated. For example, if the file TAPE contains:

OVERLAY (0,0)	Progs in (0,0)	OVERLAY (1,0)	Progs in (1,0)

   then

   REQUEST TAPE.

   TAPE.

   might become

   REQUEST TAPE.

   COPYBR (TAPE, LGO)

   SCOPE2B (7)

   COPYBF (TAPE, LGO)

   LGO.

   (Will not work with an absolute overlay file.)

3. SEGMENT Loading

The common block SCOPE2 † must be a part of the level zero segment and the SEGZERO card must be changed accordingly. This may be difficult to do, and, may not even be possible without involving more work than is required for a complete recompilation which would eliminate the original problem.

The following special set of circumstances must exist before SCOPE2B is required as an expedient:

There must be a FORTRAN program in binary form, compiled under version 2.0, which it is either impossible or undesirable to recompile.

The program must have specified the minimum buffer size for at least one of its files.

At least one of the files with the minimum buffer size must be assigned to a binary tape.

Other circumstances may make the use of SCOPE2B desirable. For example, if the first condition above is satisfied, the program when run under 2.3 will still have its buffer size reduced by 11_8 words because of the increased FET size. A normal size buffer of 2001_8 would become 1770_8 words; with SCOPE2B it would be 2011_8. However, decreasing the buffer size by 11_8 words will not reduce the I/O efficiency appreciably, since the I/O routines make dynamic use of the buffers. The desirability of using SCOPE2B is dependent on the circumstances of each possible application.

A final expedient to avoid recompilation is suggested. Input data tapes can be copied to disk read from disk, and then output data can be written to disk and copied from disk to tape. Since the PRU size of the disk is 100_8 words, the original problem does not occur.

---

† When using SCOPE2B, the user is prohibited from using his own labeled common block with a name of SCOPE2.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET
6400/6500/6600 SCOPE 3.1
Reference Manual

Pub. No. 60189400 A February 1968

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

_____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA 94304

FOLD FOLD

STAPLE STAPLE