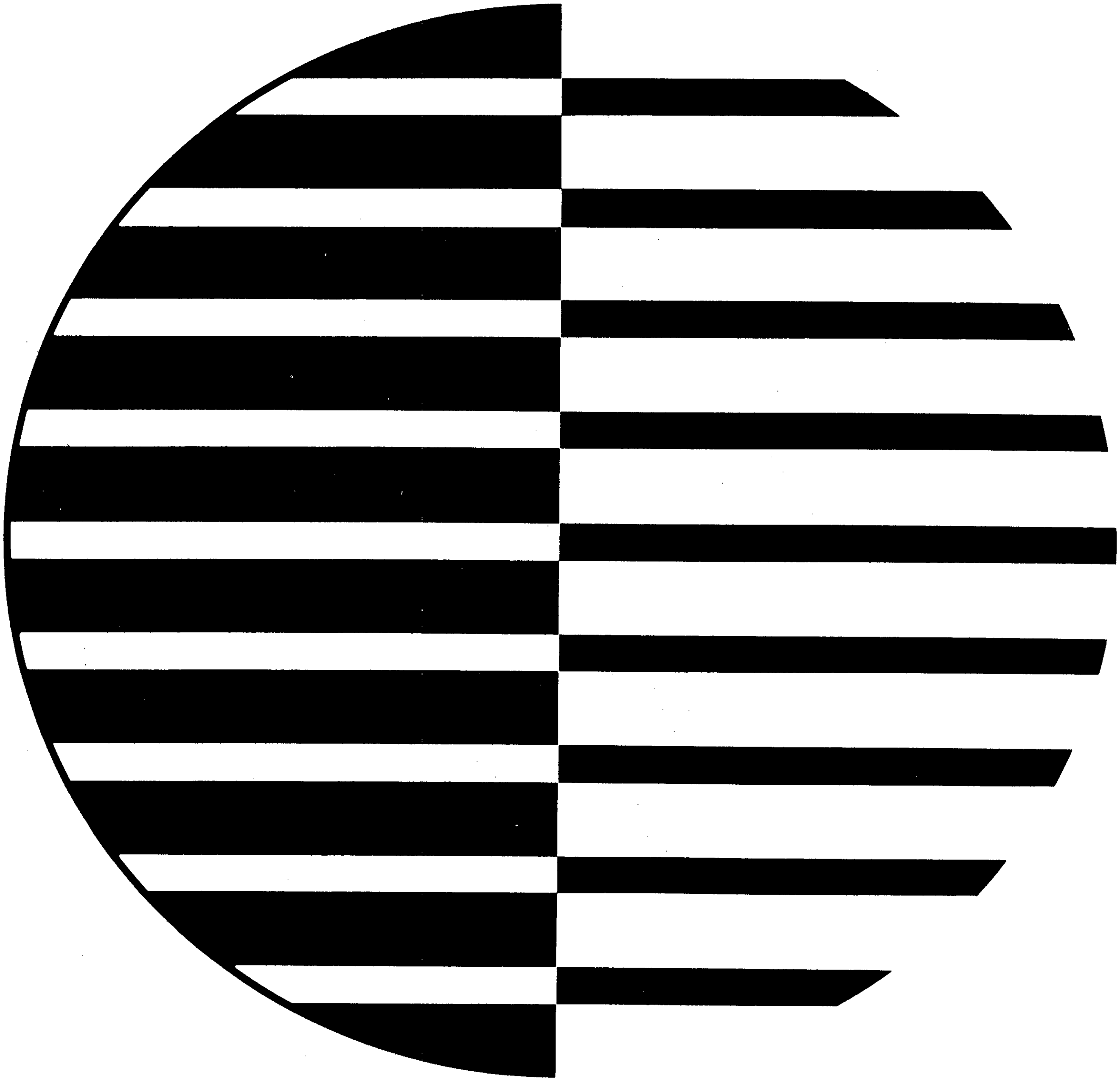
# CONTROL DATA®

# 6400/6500/6600 COMPUTER SYSTEMS

## SCOPE Reference Manual



## REVISION RECORD

| REVISION | DESCRIPTION |
|---|---|
| A | This manual incorporates Update 1 dated February 29, 1968. |
| (4-1-68) | |
| B | SCOPE 3 changes in Chapters 1, 2, 3, 9 and Appendix C. |
| (3-28-68) | |
| C | SCOPE 3.1.2 changes in Chapters 1-7, 9-11, Appendixes A-E, F, and H, |
| (9-25-68) | Contents and Index. |
| D | |
| (10-25-68) | SCOPE 3.1.2 changes in Chapters 1, 2, 3, 9, Appendix C, and Index. |
| E | SCOPE 3.1.3 changes in front matter, Chapters 1, 2, 3, 4, 5, 8, 9, 10 and |
| (12-13-68) | Appendixes E, F, and H. |
| F | Documentation changes to Chapter 6. |
| (12-27-68) | |
| G | SCOPE 3.1.4 changes include Stack Processor II, READNS function, Rewrite-in- |
| (1-31-69) | Place, DMPECS, TAPES I, REQUEST macro, Device Type Codes and C.E. |
| | Diagnostics. Affected pages include front matter, pages 1-10; 2-2, 5, 6, 9 thru |
| | 2-11; 3-2 thru 3-5, 3-8, 3-13 thru 3-15, 17, 18, 21, 23, 26 thru 3-54; 4-2, 6, |
| | 4-8 thru 4-11; 5-8; 8-1, 3; 9-4, 7, 11, 13, 9-16 thru 9-18, 21; 10-6 thru 10-18; |
| | 11-14, 15; 12-1 thru 12-5; A-1; D-5; E-1; F-3 thru F-7, H-1 thru H-4, 7, 8, 11, |
| | 12, 17, 18, 25, 26, 29, 30, 33, 34, 41, 42; L-1 thru L-15; Index-1 thru Index-9 |
| | |
| | |
| | |
| Publication No.<br>60189400 | |

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

Address comments concerning this manual to:

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

or use Comment Sheet in the back of this manual

## REVISION RECORD (Cont'd)

| REVISION | DESCRIPTION |
|---|---|
| H | SCOPE 3.1.5 changes include the use of ECS as an allocatable device, |
| (3-21-69) | CPLOADR, 512 printer support, COMBINE utility, stack processor improve- |
| | ments, and a new CHECKPOINT/RESTART. Affected pages include front |
| | matter, pages 2-4 thru 2-13; 3-4, 6, 10, 14a, 16, 17, 26, 28, 32, 36, 45, 47, |
| | 49, 50, 55; 4-1 thru 4-13; 5-2; 6-3, 9, 21, 31; 8-1 thru 8-6; 9-6 thru 9-6b, 8, |
| | 11, 19, 22, 23; 10-12, 13; 12-3, 5; A-2; F-1; H-1 thru H-48; I-2; J-5 thru J-7; |
| | L-1 thru L-3, L-5; M-1, M-2, Index-1 thru Index-9. |
| I | |
| (7-18-69) | SCOPE 3.1.6 changes include permanent file capability, improved operator |
| | capability (DSD), 512 printer selectable on REQUEST cards, magnetic tape |
| | revisions, private disk packs, user ability to process control cards and mass |
| | storage REQUEST changes. New and changed pages include front matter; |
| | pages 1-1 thru 1-15; 2-1 thru 2-16; 3-1 thru 3-72; 4-1 thru 4-13; 5-1 thru 5-12; |
| | 6-1 thru 6-32; 8-1 thru 8-6; 9-1, 9-2 (completely replaces 9-1 thru 9-23); |
| | 10-1 thru 10-18; 11-1, 11-2; 12-1, 12-2; 13-1 thru 13-20 (this is a new chapter); |
| | Glossary-1 thru Glossary-10 (this is a new section); A-1, A-2; C-1 thru C-5; |
| | D-1, D-2; E-1, E-2, F-5, F-6; G-1, G-2 (formerly Appendix I); H-1 thru H-86; |
| | I-1 thru I-20 (formerly Appendix L); J-1, J-2 (formerly Appendix M); the old |
| | Appendixes G, J and K have been dropped; Index-1 thru Index-10. |
| | |
| J | Additional documentation change for SCOPE 3.1.6 involving interpretation of the |
| (11-14-69) | MESSAGE function, and miscellaneous corrections and clarifications. Affected |
| | pages: iii; 2-3, 2-9, 2-13; 3-6, 3-21, 3-53, 3-54; 4-9, 4-11; 10-6, 10-6a, |
| | 10-11, 10-13; 13-1, 13-12, 13-15; E-1; H-11, H-17, H-31, H-35, and H-57; |
| | Index-1, Index-2, and Index-6. |
| | Comment Sheet. |
| | |
| | |
| Publication No.<br>60189400 | |

## REVISION RECORD (Cont'd)

| REVISION | DESCRIPTION |
|---|---|
| K | SCOPE 3.2 features include ECS user area accommodated by checkpoint feature, |
| (12-31-69) | level 17 always interpreted as end-of-file, multipurpose COPY routine not |
| | terminated on double end-of-file, a new record (EST) on deadstart tape, multiple |
| | CMR configurations on deadstart tape, addition of customer engineering diag- |
| | nostics. New and changed pages include front matter; pages 1-2, 1-7, 1-9; 2-1; |
| | 3-12, 3-28, 3-49, 3-54; 4-1, 4-11; 5-1 thru 5-3, 5-9 thru 5-12; 8-1, 8-2, 8-6; |
| | 10-2, 10-3, 10-18 thru 10-20; 12-1 thru 12-18; 13-1 thru 13-22; H-2, H-4, H-5, |
| | H-7, H-13, H-16, H-19, H-23, H-24, H-34, H-36, H-39, H-43, H-46 thru H-49, |
| | H-52, H-62, H-80; I-5; Index-1 thru Index-10; Comment Sheet. |
| L | Features added or changed since the previous version are indicated by a bar in |
| (4-30-71) | the outside margin or by a dot next to the page number if an entire page is |
| | affected. Additions and changes include: addition of System Engineering File |
| | Description and System Engineering File Analyzer (CEFAP) and miscellaneous |
| | corrections. New and changed pages include front matter, pages 2-10; 3-3, |
| | 3-24, 3-26, 3-27, 3-46, 3-62; 4-7, 4-8; 5-2.1; 6-13; 7-6; 10-2 thru 10-6.1, |
| | 10-13, 10-15; 11-1, 11-7, 11-13; 12-18 thru 12-30; 13-6 thru 13-10, 13-14, |
| | 13-15, 13-21; D-2, D-8; E-2; F-6, F-7; H-9, H-10, H-28, H-31, H-42, H-71; |
| | I-6; J-2; Index-1 thru Index-10; Comment Sheet. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| Publication No.<br>60189400 | |

# PREFACE

CONTROL DATA 6400/6500/6600 computers operate under the SCOPE operating system (Supervisory Control of Program Execution). This manual is a comprehensive reference document written for system and user programmers. This revision reflects the SCOPE 3.2 release.

The documents listed below are related software publications available through the nearest Control Data Corporation sales office.

| Document | Publication Number |
|---|---|
| 6400/6500/6600 Computer System Reference Manual | 60100000 |
| 6400/6500/6600 Peripheral Equipment Reference Manual | 60156100 |
| 6400/6500/6600 SCOPE 3 Product Set Diagnostic Handbook | 60252000 |
| 6400/6500/6600 SCOPE Product Set General Information Manual | 60191800 |
| 6400/6500/6600 SCOPE Installation Handbook | 60235600 |
| 6400/6500/6600 SCOPE Operator's Guide | 60179600 |

# CONTENTS

Page

# INTRODUCTION

The SCOPE operating system for CONTROL DATA® 6400/6500/6600 is a file-oriented system using mass storage, random access devices. It is designed to make use of all capabilities of the CONTROL DATA 6000 computer systems. SCOPE exploits fully the multiple-operating modes of all segments of the computer. SCOPE controls job execution, assigns storage, and performs segment and overlay loading. SCOPE features include comprehensive input/output functions and library maintenance routines. The dayfile maintains a chronological record of all jobs run and any problems encountered. Dumps and memory maps are available to aid debugging. A variety of assemblers, compilers, and utility programs may be operated under control of SCOPE, including FORTRAN, COBOL, SORT/MERGE, PERT/TIME, EXPORT/IMPORT, RESPOND, SIMSCRIPT, APT, OPTIMA, ALGOL, and utility routines.

## 6400/6500/6600 COMPUTER SYSTEM

The CONTROL DATA 6400/6500/6600 computer is composed of one or two high-speed central processors and ten peripheral processors. Each peripheral processor has its own memory and can execute programs independently of the other processors. In addition, all processors have access to the larger central memory. Up to seven jobs may operate concurrently, sharing the central processor in a multiprogramming manner. During a job, one or more peripheral processors are used for high-speed information transfer in and out of the system.

## OPERATING SYSTEM COMPONENTS

Components of the SCOPE Operating System are distributed among the central memory, the peripheral memories and system mass storage devices. The central resident portion of the system consists of system tables and pointers, communication areas used to link the peripheral and central memories, and frequently used subroutines for both central and peripheral processors.

The system monitor routine is assigned to one of the peripheral processors and the system display program to another. The other eight peripheral processors have no fixed assignments but form a common pool available for assignment as needed. They contain idling routines that repeatedly examine their communication areas in central memory for requests. The remainder of SCOPE is stored on mass storage and in central memory and called as needed.

## JOB PROCESSING

A job consists of one or more programs, preceded by control cards specifying equipment, time limits, priority, operator instructions and other information needed by SCOPE. The operator initiates job loading from the system input unit into mass storage. SCOPE selects and processes jobs from mass storage and routes output to the proper devices.

All system activities are controlled by the system monitor. The monitor accepts messages from processors and routes them to processors for action. It communicates with the resident programs in the other nine peripheral processors and with programs active in the central processor through central memory communication locations and control point areas.

Central processor job operation is initiated or interrupted by an exchange jump command from the monitor routine. The central processor enters information about a new program into registers and stores the current information of the interrupted program in the new program's control point area. Hardware features insure that the interrupted program is left in a state for re-entry.

Jobs assigned to control points and waiting for execution after interruption are stacked by priority. The job using the central processor is at the top of the stack. When it is interrupted (for example, to await completion of a peripheral processor function), the next job in the stack becomes the top, and the first job is temporarily removed from the stack. When it re-enters the stack, the job using the central processor, along with all jobs beneath it, are pushed down.

Blocks of central memory storage assigned to control points occupy positions in central memory relative to the control point number to which they are assigned. Storage assigned to control points is relocated up or down as storage is released or required. Such relocation is possible because all references to central memory are made relative to the reference address assigned by the monitor routine.

## LOADER

The loader performs the following functions: loads absolute and relocatable binary programs, links separately compiled or assembled programs, loads library subprograms and links them to user programs, detects errors and provides diagnostics, outputs a memory map, and generates overlays.

The loader is used whenever relocatable programs are transferred from an input or storage device to central memory. Loading errors are written as diagnostics on the dayfile. During the loading process, the loader links subprograms together and generates overlays as directed.

Three types of loading are performed: normal, segment, and overlay. Segments and overlays allow programs that exceed storage to be organized so that portions or groups of programs may be called, executed, and unloaded as needed.

## SYSTEM LIBRARY AND MAINTENANCE PROGRAMS

The available system library and maintenance programs are EDITSYM, EDITLIB and UPDATE. EDITSYM and UPDATE provide symbolic, language-independent editing of program files, which alleviates the burden of maintaining large card files. A two-level editing technique permits the installation to make modifications to standard software without affecting the standard sequence numbers. EDITSYM and UPDATE process symbolic card decks in compressed format. They can compress and sequence the source cards of one or more decks to form a program library or create a compile file from one or more decks in a program library. Modification of program libraries can be done independently or in conjunction with the creation of a compile file.

The system library file is created and modified through the EDITLIB routine. This file may reside on mass storage and in central memory or on a deadstart load tape. EDITLIB runs in a normal multiprogramming environment and is submitted as a normal job. It can create a deadstart load tape from any combination of the following sources: the currently operating system, an already existing deadstart tape, and a binary input file. EDITLIB can delete, add, or change the residence of routines in a currently operating mass storage/central memory system library.

## SYSTEM REQUESTS

System requests enable the user to access certain variables in his operating environment, to manipulate data files, request dumps, and perform input/output functions. These requests are grouped into file action requests and system action requests.

File action requests consist of position functions, data functions, and file initialization functions. Position functions move files on sequential devices forward or backward, and position files on random access devices at specific locations. Data functions transfer data between hardware devices and central memory. Files are readied for processing by the file initialization functions.

A system action request for a checkpoint dump causes SCOPE to record all information relevant to the execution of the program. Execution may be restarted from the checkpoint dump rather than the beginning of the job. This feature is particularly useful for long jobs, which may not be run continuously from beginning to end.

Through other system action requests the user may release or request additional central memory or extended core storage.

## DAYFILE

The dayfile is a mass storage file that holds a running account of all control cards, equipment assignments, error diagnostics, central and peripheral-processor time used, and input/output routines used by the jobs in central memory. At the end of a job printout, all dayfile messages associated with the job are printed. The dayfile can be accessed for accounting purposes. The latest dayfile messages are visible to the operator on one of the display scopes.

## SYSTEM DISPLAYS

Two console screens provide system monitor information and central memory displays. Through the console keyboard the operator can control the operation of the running system.

The operator can request various displays: the most frequently used are job status, system files, and dayfile displays. The job status display provides the status of all jobs being executed, their priority, and position in the control point stack, the last program message, and other pertinent information. The system files display shows the status of jobs not yet being executed. The dayfile display gives the latest dayfile messages. As new messages are added to the bottom of the display old messages are removed from the top.

# 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 (unless the central processor exchange jump option is available).

PP0 contains the monitor program (MTR) and is in permanent, overall control of the system; the other PP's cannot perform any function not approved in advance by MTR. 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.

PP9 is permanently assigned to the system display routine (DSD). 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 frees the channel.

## 1.2 MULTI-PROGRAMMING

### 1.2.1 CENTRAL MEMORY USAGE

CM low core, called the central memory resident or CMR, is reserved for system tables and programs 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

Up to eight areas, numbered 0 to 7, are designated as control points within central memory resident (CMR); the actual number is a configuration parameter within 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:

| 59 | 53 | 35 | 17 0 | Words |
|---|---|---|---|---|
|  | Program Address (P) | A0 (Address Registers) |  | 0 |
|  | Reference Address (RA) | A1 | B1 (Increment Register) | 1 |
|  | Field Length (FL) | A2 | B2 | 2 |
|  | Exit Mode (EM) | A3 | B3 | 3 |
| RA-ECS† | 000000 | A4 | B4 | 4 |
| FL-ECS† | 000000 | A5 | B5 | 5 |
|  | Monitor Address†† | A6 | B6 | 6 |
|  |  | A7 | B7 | 7 |
| X0 (Operand Registers) |  |  |  | 10 |
| X1 |  |  |  | 11 |
| ⋮ |  |  |  |  |
| X7 |  |  |  | 17 |

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 references to central memory instructions or data are relative to the program's CM 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.

† The ECS RA and FL are expressed in thousands (octal).

†† This applies only to machines equipped with the central processor exchange jump feature (CEJ).

## 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.

Permanent files which are attached to a job

The SCOPE operating system maintains a file name table (FNT) in central memory resident. This table contains one 3-word entry for each active file in the system. The first word identifies the file and contains other information about it. The 2nd and 3rd words which describe its status, are sometimes called a file status table or FST entry. When the user requests a file with a REQUEST control card or macro call or when he issues an I/O request referencing a file that does not exist, the SCOPE system creates an FNT entry for the file and assigns the file to a device. Thereafter, each time a user makes an I/O request, file status information is transmitted between the user's FET (section 3.2) and the FNT.

The four types of active files are: input, local, output, and common. When a permanent file is attached to a job, it becomes a special kind of local file. 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 mass 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 on mass 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, OUTPUT, PUNCH, and PUNCHB may be 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. 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 this 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. If RASP is assigned to a private disk pack, however, it will be preserved on the disk pack when the job terminates and the COMMON card will have no effect.
>
> An attempt to declare a permanent file COMMON is illegal.

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 detached 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 |
| 6603 disk | 64 CM words |
| 865 drum | 64 CM words |
| 854 disk pack | 64 CM words |

For magnetic tapes reference appendix I.

## LEVEL NUMBERS

Related logical records within a file may be grouped by the user into an organized hierarchy. The level number ($0\text{-}17_8$)† 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. A logical record of level 17 is always interpreted as a logical end-of-file, whether or not it is of zero length. 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 | 1 | I |
| 2 | 1 | 2 | | |
| 3 | 0 | 3 | 2 | |
| 4 | 0 | 4 | | |
| 5 | 2 | 5 | | |
| 6 | 0 | 6 | 3 | II |
| 7 | 0 | 7 | | |
| 8 | 1 | 8 | | |
| 9 | 0 | 9 | 4 | |
| 10 | 0 | 10 | | |
| 11 | 0 | 11 | | |
| 12 | 1 | 12 | | |
| 13 | 0 | 13 | 5 | |
| 14 | 2 | 14 | | |
| 15 | 0 | 15 | 6 | III |
| 16 | 1 | 16 | | |
| 17 | 0 | 17 | 7 | |
| 18 | 0 | 18 | | |
| 19 | 2 | 19 | | |
| End of Information | | | | |

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

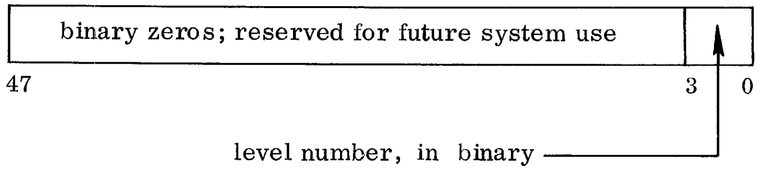
Card Files

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, or with 7,8,9 punches in column 1 and 17 punched in columns 2 and 3.

Mass Storage Files and Binary Mode 1/2-inch Magnetic Tape Files (SCOPE Standard)

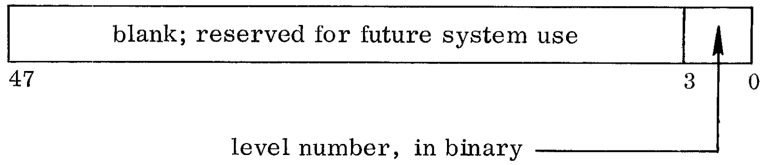
Each logical record is terminated by 8 characters (48 bits) as follows:



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 (SCOPE Standard)

Each logical record is terminated by 8 characters as follows:



The level number is the low order 4 bits of the last character. The upper two bits of this character are always zero except for level zero where they are $01_2$. For example, level five would be represented by 2020202020202005 in external BCD. Level zero would be represented by 2020202020202020 in external BCD. 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 on mass storage devices and their records can be read by direct addressing or sequential references. 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 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 assigned to private disk packs, declared as common files or cataloged as permanent files. At the end of a run which creates such a file, the user should close the file. The system then automatically appends to the end of the file the contents of the index buffer specified in the FET. When the file is to be read, the user must 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. 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. In addition to the standard SCOPE labels, an installation may optionally choose to process tapes with a label format identical to that used by Control Data 3000 series computers (Appendix C).

Tapes containing a system label (or optionally a 3000 series label) are recognized as labeled tapes. All other tapes are considered unlabeled. Label processing is not provided for unlabeled tapes. SCOPE system labels are recorded at 556 bpi (or at a value set by an installation parameter). Control Data 3000 series labels are recorded at the same density as the file.

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 user process (UP) bit is set in the FET, reel swapping for a multi-reel tape file is automatic. The system executes two function calls: CLOSER UNLOAD and OPEN REEL. Since these calls are issued by the system, the file header label is delivered in the FET 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 from 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 or a volume header label. 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 preceded and followed by a tape mark, and if it is the last file trailer in the volume, two following tapemarks 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 (* indicates tape mark):

```
                         ...File A...* EOV1* *
VOL1 HDR1** EOF1 * HDR1*...File B...
     (A)    (A)    (B)
          ...File A...* EOF1 * HDR1* * EOV1* *
VOL1 HDR1*...File B...
     (B)
```

### 1.5.2 DISK PACK FILE STRUCTURE

An 854 disk pack is organized into $2000_{10}$ record blocks of five PRU's each. Every RB fills a single track of the disk pack; its logical number is 10a+b, where a is the cylinder number and b the head group number. Of the 16 sectors physically available on a track, the last (number 15) is unused; the remainder are grouped into PRU's as follows:

| 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 |

Two revolutions are required to read PRU's in sequence in a record block.

Disk pack use can be either public or private. If public, it is available to the system for writing and reading files on the same basis as a large disk unit of the non-removable type. If private, it provides space for up to 63 files, none of which are permitted to overflow to another storage device. When a private pack is assigned to a job (in response to an RPACK control card), the names of the files it contains are read from the first three record blocks into CMR, along with the record block reservation table (RBR) and record block table (RBT) chains needed to access the files. When the job terminates, these tables are written back onto the pack before it is logically unloaded.

The first PRU of a disk pack contains its label, formatted as follows:

| Character Positions | Description |
|---|---|
| 1-4 | DEV1 |
| 5 | Binary zero |
| 6-10 | Julian date (yyddd in display code) |
| 11-20 | Visual number/identifier, right justified with display code zero padding |
| 21-30 | Binary zero |
| 31-38 | Binary zero if public device; pack name, left justified with binary zero fill, if private |

| Character Positions | Description |
|---|---|
| 39-40 | Binary zero if public; binary count of files on pack if private |
| 41-78 | Binary zero |
| 79-80 | Checksum of all other 12-bit bytes of PRU |
| 81-460 | RBR table |
| 461-640 | Binary zero |

If the pack is public, the rest of the first record block cannot be used, and usable space begins with the second record block (number 1).

If the pack is private, file names and RBT chains are stored in the rest of the first record block and may extend through the second and third; usable space begins with the fourth (number 3).

Files on a private pack can be accessed only by the job to which it is assigned. Private pack files can be only type local (not common or output) and must have zero disposition codes.

# 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 mass 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 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.
- The name of the job file is changed to INPUT and the file is positioned to the beginning of the second record.
- 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, the job is terminated. Control cards are written in the job dayfile as they are executed.

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 starting in the first non-blank column. Flag words described in this section are reserved for the system and may not be used as a name or a program call card. All cards with a flag word not recognized by SCOPE are treated as program call cards.

The second field is optional; it may contain one or more parameters, separated by any character that is not alphanumeric, not a blank nor an asterisk. The two fields are separated by any character that is not alphanumeric nor an asterisk. The parameter field is terminated by a period or right parenthesis and a terminator must be present even when no parameters are specified. All blanks are ignored in the parameter field. Comments may appear to the right of the terminator. Characters in the range of 1 to 44 in display codes (A-Z and 0-9) are considered alphanumeric.

### 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. Time limit must suffice for the whole job including all compilation and execution. Value may not exceed $32767_{10}(77777_8)$. The value of $32767_{10}$ defines an infinite time: no time limit check is performed in this case.

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.

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 $\leq 8$; 1 is the lowest priority.

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

## SWITCH CARD

SWITCH,n.

At the beginning of a job pseudo sense switches 1-6 are set to off. Settings may be changed and preserved at the control point for reference by a subsequent program within the same job. Each use of the SWITCH,n. card changes the current status of the specified switch. For example the first SWITCH,4. card will turn that switch on, the second SWITCH,4. card will turn it off. Switches also 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:<br>Attempt was made to reference central memory or extended core storage outside established limits<br>Word count in extended core storage communication instruction is negative |
| 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 or
operand out of range as stop conditions.

A mode zero error may occur if the program jumps to relative location zero.

## CKP CARD

CKP.

CKP causes a checkpoint dump to be taken of all files currently active at the control point. The effect is identical to that of the CHECKPT macro call (section 8.1), except that no parameters can be specified.

## RESTART REQUEST

A job may be restarted from its checkpoint tape by the RESTART control card. Section 8.2 describes the five possible card formats.

## 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. The speed at which control cards are processed by SCOPE may prevent the comment from being noticed by the operator. The COMMENT card does not halt job processing.

## 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 (octal) | Condition | |
|---|---|---|
| 1 | Time expired | Job has used all CP time it requested; any further attempts to use CP 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 error entered in RA+1 by a central program. |
| 6 | Operator drop | Operator requested job to be dropped. |
| 7 | Kill | Set by operator or PP program to drop a job and inhibit all output. |

| Error Flag (octal) | Condition | |
|---|---|---|
| 10 | Rerun | Set by operator to force job back into the input queue. |
| 11 | Control card error | |
| 12 | ECS parity error | |
| 13 | Job card error | |
| 14 | Job pre-abort | Job not read correctly (e.g., checksum error discovered by JANUS). |
| 15 | Auto-recall error | Job entered auto-recall with completion bit set. |
| 16 | Job loop in auto-recall | No activity exists for a job in auto-recall, and completion bit is not set. |

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

If no error condition occurs but an EXIT or EXIT(S) (section 4.7) control card is encountered, the job will terminate. If error condition 7, 10, 13, or 14 occurs the job will terminate. If error condition 11 occurs or if a user tries to load the output from a bad assembly or compilation, a search is made for an EXIT(S) control card. If the search is not successful, the job terminates; otherwise the error flag is reset and the control cards following the EXIT(S) card are processed. Any of the other error conditions (1,2,3, 4,5,6,12,15,16) result in a search for either EXIT or EXIT(S) control card. If encountered the card is written to the job dayfile and the control cards following it are processed.

Example:

| | |
|---|---|
| MYJOB,P1,T400,CM50000. | Job card |
| REQUEST,TAPE5. | 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.

### LOADER CARD

LOADER (name)

In a SCOPE system containing more than one loader, this card selects a loader for the job in execution. Name is 1-7 alphanumeric characters chosen from a prescribed list (currently PPLOADR and CPLOADR).

The primary operations of PPLOADR (text relocation and loader table building) are performed in the PP. The primary operations of CPLOADR are performed in the CP.

When this control card is omitted from a job, or if no name is given, the system selects a loader by default. The default option is defined by a system installation parameter. An unrecognized name produces a dayfile message and the job is terminated.

Four control cards that relate directly to the loader are discussed in the next section: LOAD, EXECUTE, program-call-card, and NOGO. Each of these control cards is processed by either loader in the same way.

## 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 function or program being executed. The contents of list depend upon the specific program. 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.

## 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 in relocatable routines 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: $\left.\begin{array}{l}\text{LOAD (BTGN)}\\ \text{EXECUTE.}\end{array}\right\} = \text{BTGN.}$

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.

## REDUCE CARD

REDUCE.

When the REDUCE card is encountered, a flag is set for the loader. After loading and just prior to execution, the field length is reduced to the highest word address loaded (or the top of blank common) rounded up to the next 100 (octal). The field length remains reduced until the end of the job or until an RFL card is encountered. The REDUCE card is not honored if an external reference is made to LOADER. The REDUCE card may be positioned anywhere before the EXECUTE or PROGRAM CALL card. Any succeeding runs that require a larger field length will be terminated unless an RFL card is used to increase the field length.

Example:

| | |
|---|---|
| LOAD(FAT) | Load large program |
| REDUCE. | |
| EXECUTE. | Field length will be reduced |
| RFL(60000) | Increase field length for next load |
| LOAD(FAT) | |
| REDUCE. | |
| LOAD(NOTFAT) | Load more |
| NOTFAT. | Field length reduced again |

### MAP CONTROL CARDS

Any of the three MAP cards may appear prior to an EXECUTE or PROGRAM CALL card. The MAP option selected by a MAP control card prevails until end-of-job, or until changed by another MAP control card.

Default, as defined by installation parameter, prevails when a MAP control card does not appear. The default option may be set by installation parameter to be any one of the three options described below:

MAP(ON)

When this card is used, a map is produced after loading is completed. Programs loaded from the system library as a result of explicit control card calls, such as RUN, COMPASS, etc, do not appear in the map.

MAP(OFF)

When this card appears, no map is produced after loading is completed.

MAP(PART)

This card is used to produce a partial map after loading is completed. The partial map is identical to that produced by MAP(ON) except that entry addresses are omitted.

## 2.4 EQUIPMENT ASSIGNMENT

If a file is not specifically assigned by a REQUEST card or function, the system assigns that file to mass storage. A job need not assign the card reader, printer or punch for normal input/output, as this is done automatically by the system. In addition, certain files with special names (OUTPUT, PUNCH or PUNCHB) will always be processed 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 file.

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

This card declares properties of a file and requests assignment of a physical unit. Assignment may be automatic or may require operator action; and it may be to a specific mass storage unit by EST ordinal, or to any available mass storage unit by equipment type and allocation style. Assignment of a file to other than a mass storage device requires a REQUEST card or function.

Because the control cards of a job are processed in order, equipment assignments must be made before the file is referenced. A REQUEST card must have at least one parameter, and the first parameter is assumed to be lfn; all other parameters 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 beginning with a letter), specifies name of the file to which equipment is to be assigned and the name by which the user refers to the file.

dt Designates type of device to which file is to be assigned. When dt is specified, the operator must assign the proper type of device; otherwise any equipment may be assigned by the operator. dt may take the following forms:

hh Hardware type mnemonic from the basic file environment table (section 3.2.1). In addition, A* may be used for mass storage if the user is not concerned as to which mass storage device is assigned. For mass storage, hh is equivalent to hhaa with aa = 00. The hh specification requests the operator to assign a device of the specified hardware type. No other hardware type assignment will be accepted.

hhaa Designates a mass storage device where hh is as above and aa is an optional allocation style code (two octal digits) as listed in section 3.2.1. The operator must assign a mass storage device of the specified hardware type and allocation style; no other hardware type or style will be accepted. By using A*aa, the user indicates any mass storage hardware type with the aa allocation style.

*hhaa Same as for hhaa, except that no operator intervention is required. When *A* is used, the system will assign any mass storage hardware type with the allocation style aa.

*hh Same as for hh, except that no operator intervention is required.

Dnnnn Meaningful only for mass storage, nnnn is equivalent to the combination, ddaa, corresponding to a hardware type mnemonic and allocation style under the specification hhaa. dd is the octal device type code corresponding to hh.

PK Valid only for private disk packs. The format of the request card must be REQUEST, lfn, PK, pname; where pname is the name of the private pack already assigned by the operator in response to an RPACK control card. This form of request card does not cause a halt for operator assignment; the assignment is made automatically. If no pname is assigned to the job, a dayfile message is issued and the job is terminated.

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

MT 1/2" magnetic tape at installation default density

When the equipment type is MT and the tape has SCOPE system labels, input tape is read at density specified in volume header label. Output tape is written at density specified by an installation parameter. For unlabeled tape, a density specified by an installation parameter is used.

zhh z has significance only for MT, LO, HI and HY. When z = 2, two 1/2-inch magnetic tape units are requested, and they are used in the order assigned by the operator. When the tape on the first unit reaches end-of-reel, the system begins processing the tape on the second unit while the tape on the first unit is rewound and unloaded. When the tape on the second unit reaches end-of-reel, the system returns to the first unit which should have been mounted with a new tape in the interim. The tape on the second unit is rewound and unloaded. This alternating process is repeated as long as the file is referenced. When z = 1 or absent, and an end-of-reel occurs, the system rewinds and unloads the unit and waits for the unit to become ready.

Example:

REQUEST,TAPE1,2MT.

Requests two tape units for file TAPE1

REQUEST,TAPE1,AA02.

Requests operator assign 6603-I disk, outer zone for file TAPE1

REQUEST,TAPE1,*AP.

System will assign first available public disk pack to file TAPE1

REQUEST,TAPE1,*A*01.

System will assign first available mass storage unit that contains allocation style 01 (50 PRU's per assignable unit)

dc — Disposition code. Optional properties of a file may be declared by the disposition code. The following mutually exclusive values are permitted; CK and MF may be used only for magnetic tape; all others apply only to files on allocatable devices, and disposition will be ignored if they are used for files on other than allocatable devices, such as tapes.

| | |
|---|---|
| CK | Identifies checkpoint dump file. |
| MF | Identifies a multi-file tape (section 3.2.1). The lfn must be six characters or less. |
| P8 | Punch file at job completion; 80 columns are punched with no formatting of the card. |
| PR | Print file on any printer at job completion; automatically assigned to the file named OUTPUT. |
| P1 | Print file on 501 or 505 printer at job completion. |
| P2 | Print file on 512 printer at job completion. |
| PU | Punch coded file at job completion; automatically assigned to the file named PUNCH. |
| PB | Punch binary file at job completion; automatically assigned to the file named PUNCHB. |
| FR | Print on microfilm recorder at job completion; automatically assigned to the file named FILMPR (Driver not provided.) |

FL Plot on microfilm recorder at job completion; automatically assigned to the file named FILMPL (Driver not provided.)

PT Plot at job completion; automatically assigned to the file named PLOT (Driver not provided).

HR Print on hard copy device at job completion; automatically assigned to the file named HARDPR (Driver not provided).

HL Plot on hard copy device at job completion; automatically assigned to the file named HARDPL (Driver not provided).

x, y These two fields describe the data format conventions and the labeling conventions for magnetic tape files. If the fields are omitted and the file is declared to be magnetic tape, the file is assumed to be in standard SCOPE data format with no labels. Any one of the optional data formats described below may be declared in either field; any one of the label declarations may be placed in the other, except no label format may be specified with the X data format. More than one data format declaration or more than one label declaration is considered an error; a message will be given and the job will be terminated.

<u>Data Formats</u> (see Appendix I)

| | |
|---|---|
| blank | SCOPE standard |
| X | External - SCOPE 2.0 compatible |
| S | Stranger tape |
| L | Long record stranger tape |

<u>Label Formats</u>

| | |
|---|---|
| absent | Unlabeled |
| E or N | SCOPE standard labels (E and N are equivalent) |
| Y | 3000 (or installation defined) label |

eq Specific device. The eq option is available for use in an environment where the user has control over equipment assignment (such as in an open shop). It is <u>not</u> recommended for use otherwise. eq is one or two octal digits specifying an ordinal in the equipment status table (EST). The system will assign the named file to the specific device without operator intervention. If the equipment is unavailable, the operator is notified; the request is reprocessed until the equipment becomes available or the operator terminates the job. The eq option may be used in combination with the hhaa option, in which case SCOPE verifies that the specific equipment satisfies the device type and allocation style specification.

RPACK CARD RPACK,pname,N. RPACK,pname,E. RPACK,pname,E,vrno.

This card directs the operator to assign a private disk pack named pname. This is not the name of any file associated with the pack, though the pack and one of its files may, by coincidence, have identical names.

If the second parameter is N, the operator must assign a pack that is already blank labeled. When he does so, the message TYPE IN VISUAL PACK NUMBER will appear, and the operator types n.VRN,xxxxxx. where xxxxxx is the number or other identification of the physical pack. Then the pack is assigned to the job as a private pack not yet containing any files; a label, containing the pack name and visual number, is written immediately on its first PRU.

If the second parameter is E, the operator must assign a pack that is already labeled as a private pack with the name pname. If the third parameter, vrno, appears on the card, it must match the visual identification in the pack label. If the pack assigned by the operator fails either test, the operator is informed and the job waits until another unit is assigned or the job is dropped. When a pack passes the tests, its files are made available to the job, and other files can be added to it by REQUEST cards.

Examples for RPACK and REQUEST card:

1. JOB1 creates two files TAPE1 and TAPE2 to reside on a private disk pack named MYPACK with visual ID of N1122.

```
JOB1,T1000.
RUN(S)
COMMENT.  THE OPERATOR SHOULD ASSIGN A BLANK
COMMENT.  LABELED PRIVATE PACK AND HE SHOULD
COMMENT.  TYPE IN A VRN OF N1122 TO THE FOLLOWING
COMMENT.  RPACK REQUEST
RPACK,MYPACK,N.
REQUEST,TAPE1,PK,MYPACK.  AUTOMATIC SYSTEM
                             ASSIGNMENT
REQUEST,TAPE2,PK,MYPACK.  AUTOMATIC SYSTEM
                             ASSIGNMENT
LGO.
7-8-9
   FORTRAN PROGRAM TO CREATE
   FILES TAPE1 AND TAPE2
7-8-9
6-7-8-9
```

2. JOB2 reads the two files previously created by JOB1 and creates a third file TAPE3 on the private pack.

```
JOB2,T1000.
RUN(S)
COMMENT.  THE OPERATOR MUST ASSIGN PRIVATE PACK
COMMENT.  WITH VISUAL ID N1122 TO FOLLOWING RPACK
COMMENT.  REQUEST
RPACK,MYPACK,E,N1122.
REQUEST,TAPE3,PK,MYPACK.
LGO.
7-8-9
     FORTRAN PROGRAM TO READ
     FILES TAPE1 AND TAPE2 AND
     CREATE A THIRD FILE TAPE3
7-8-9
6-7-8-9
```

## REMOVE CARD

```
REMOVE,lfn.   REMOVE,lfn,pname.
```

This card requires no operator action. It removes the file named lfn from the File Name Table provided it is assigned to a private disk pack. All its disk space is released, and its name will not appear in the label written on the pack at the conclusion of the job. If parameter pname appears on the control card, a check will determine that this is the name of the pack containing file lfn.

Example:

JOB3 removes the file TAPE2 from private disk pack named MYPACK with visual ID of N1122.

```
JOB3,T1000.
RPACK,MYPACK,E,N1122.
REMOVE,TAPE2,MYPACK.
7-8-9
6-7-8-9
```

## COMMON CARD

```
COMMON,lfn.
```

If the file name, lfn, is type common in the file name table (FNT) and is not being used by another job, it is assigned to the current job until it is returned or unloaded or until job termination. If the file's status is not common or if it is being used by another job, this job must wait until the file is available. A job which uses a common file is marked by the system as being unable to be rerun.

If the file name, lfn, already appears as a local file name for the job, the file will be assigned common status in the FNT and becomes available to any succeeding job after the current job terminates. However, if a common file of the same name already exists, the COMMON card will be rejected.

If the file named on the COMMON card does not appear in the FNT, an operator message is displayed, and the job must wait until the file appears and is available.

If the file resides on a non-allocatable device such as magnetic tape, the equipment will be logically turned OFF until the common file is released.

If the file resides on a private disk pack, the COMMON card has no effect, but a diagnostic is issued.

RELEASE CARD

RELEASE, lfn.

With the RELEASE control card, the common file named lfn currently assigned to this job is dropped from common status and assigned local status in the FNT. The file is released at the end of the job.

RETURN

RETURN, $\text{lfn}_1$, $\text{lfn}_2$, . . . $\text{lfn}_n$.

When the RETURN card appears, a CLOSE, UNLOAD is performed on each file named. Files and equipments are returned to the system for subsequent processing (section 3.6.1).

# OBJECT PROGRAM-SYSTEM COMMUNICATION 3

The control cards direct the SCOPE system to initiate and terminate user programs. SCOPE also supervises user program runs, controlling all input/output operations and providing program-system interface.

These run-time functions are performed by system subroutines called by the user in the text of his program. A comprehensive set of system macros is available for calling subroutines and generating the tables for passing parameters between the system components. System subroutines and communication tables reside in the user's field length and should, therefore, be considered when specifying field length.

During a user program run, the system performs two types of operations:

- Input/output operations, initiated by file action requests in the user program
- System operations, initiated by system action requests

Both file and system requests call the Central Program Control subroutine (CPC) which provides linkage with the monitor. Before CPC can honor a file action request, the File Environment Table (FET) must have been established for the file to be processed.

## 3.1 FILE NAME TABLE

The File Name Table (FNT) is a system table containing a three-word entry for every active file in the system. It provides a link between the user's FET and the system input/output routines. The FNT is protected from user access, as it resides in low core and is outside the field length of user jobs.

The following information is contained in each FNT entry.

FILE NAME

The name of the file. If the file is created by OPEN, REQUEST, or CIO calls, the name must be 1-7 alphanumeric characters beginning with a letter, and it cannot include embedded blanks. Otherwise, the file may be any 42-bit quantity in which the high-order 12-bits are not all zeros.

## FILE TYPE

A number which identifies the file type: input, output, local, or common.

## CONTROL POINT NUMBER

The control point to which the file is assigned. If a file is associated with a job running at a control point, it is assigned to that control point. Otherwise it is assigned to control point zero.

Files of type input are always assigned to control point zero. Each input file must have a unique name. Each file assigned to any control point other than zero must have a name which is unique among files at that control point. Local files at control point zero need not have unique names. Each common file must have a name unique among all common files regardless of control point.

## EQUIPMENT TYPE

A number which specifies the type of device or equipment on which the file resides. This could be a mass storage device such as drum, disk, or disk pack; or it could be a sequentially accessible equipment such as magnetic tape, line printer, card reader, or card punch. Most mass storage devices are called allocatable, since portions of the device can be allocated to different jobs. Sequential access devices are non-allocatable. A private disk pack is a non-allocatable mass storage device.

The system enters the equipment type in the FNT when the file is created. If the user creates the file with a REQUEST control card or macro, he can specify the type of device. If the file is created when the user issues an I/O function for a non-existent file, the allocatable device with the most available space is selected. The user can specify allocation style in his FET which can help determine the device; but he cannot specify the device itself. The device type field in the FET consists of 6 bits for the allocation style and 6 bits for the hardware device type. Each time a user requests an I/O function, the device type from the FNT is placed in the hardware device type field in the user's FET and the allocation style is set in his FET. However, if the 6-bit hardware device type field in the user's FET indicates a non-allocatable device when he is opening a non-existent file, the job is terminated. The value of the hardware device type in the FET is ignored in any other case.

## LAST CODE AND STATUS

To perform an I/O operation, the user must set a code in the code and status field of his FET. If he uses a system macro, this will be done automatically. When the system starts to process the I/O request, it stores the code and status from the FET in the FNT. When the I/O operation is complete, the system stores status information in the FNT code and status field, and also sets the completion bit (bit 0 of field). The FNT code and status is then copied to the FET. If an end-of-record or end-of-file status was returned on the last READ, the user must clear these bits before the next READ is issued; otherwise CPC will not honor the next READ action requested.

## SECURITY CODE

This code indicates whether the file is currently open or closed; and if open, the code indicates whether it is open for READ, WRITE or ALTER.

## PERMISSIONS

To perform certain operations on a permanent file, a user must first obtain permission. For example, to write on a permanent file the user must have either EXTEND or MODIFY permission. The current file permission is kept in the FNT.

## DISPOSITION CODE

When it is time to dispose of a file, this code indicates the action to be taken. Normally, files with non-zero disposition are placed in the output queue. JANUS will pick up files from the output queue having disposition codes between $10_8$ and $47_8$; other disposition codes indicate EXPORT/IMPORT and RESPOND files. In addition, the system uses a bit in the disposition code to indicate when the type of a file is to be changed between common and local.

If the FET length is five words, the disposition code in the FET is not placed into FNT nor is it checked by SCOPE.

The disposition code can be set in the FNT in several ways:

> When a REQUEST control card or REQUEST macro specifies a disposition code, the code is put in the FNT.
>
> If the user issues any I/O request, including OPEN, for a file that does not exist and specifies a disposition code in his FET, this code is placed in the FNT.
>
> If the user opens or closes an existing file with an FNT disposition code of zero, the disposition code in his FET is placed in the FNT.
>
> Prior to disposing of a local file, the SCOPE system checks for. special name files such as OUTPUT, PUNCH, PUNCHB, etc.,

and if the current FNT disposition code is zero, an appropriate disposition code will be inserted automatically in the file's FNT.

If the user opens or closes a file with a non-zero FNT disposition code, the code is placed in his FET. In addition, for an I/O request other than an open or close on an existing file, the disposition code from the FNT is placed in his FET regardless of its value. Therefore, once a disposition code is set in the FNT it can never be changed; and after every I/O operation, the FET and FNT will contain the same disposition code.

In setting a disposition code from his FET or from a REQUEST macro, the user should be certain the value is legal as the system will accept any value. Normally, the user should never set a disposition code greater than $47_8$; nor should he set the EXPORT/IMPORT, RESPOND or common file change bits. These should be set only by the system.

FET ADDRESS

The relative address of the FET used for the last operation on the file.

## 3.2 FILE ENVIRONMENT TABLE

The File Environment Table (FET) is a communication area initiated by the user; it is interrogated and updated by the system and the user during file processing. An FET must be declared for each file. The system section of the FET is used by the peripheral processor input/output routines and CPC as well as by the user program. A user section may be appended to the system FET to centralize other information pertinent to the file. All FET's reside within the field length of the program. The format of the system FET is shown below.

<table>
<tr><th>Bits 59</th><th>47</th><th>44</th><th>35</th><th>32</th><th>29</th><th>23</th><th>17 0</th><th>Words</th></tr>
<tr><td colspan="7">logical file name (lfn)</td><td>code and status</td><td>1</td></tr>
<tr><td>device type</td><td colspan="2">r n up ep eb ai</td><td colspan="2">disposition code</td><td colspan="2">ℓ</td><td>FIRST</td><td>2</td></tr>
<tr><td colspan="7">0</td><td>IN</td><td>3</td></tr>
<tr><td colspan="7">0</td><td>OUT</td><td>4</td></tr>
<tr><td>FNT pointer</td><td colspan="3">record block size</td><td colspan="3">physical record unit size</td><td>LIMIT</td><td>5</td></tr>
<tr><td></td><td colspan="4">working storage fwa</td><td colspan="2"></td><td>working storage lwa+1</td><td>6</td></tr>
<tr><td colspan="5">(Magnetic Tape)<br>(Mass Storage)</td><td colspan="3">UBC MLRS<br>record request/return information</td><td>7</td></tr>
<tr><td>record number</td><td colspan="4"></td><td colspan="2">index length</td><td>index address</td><td>8</td></tr>
<tr><td></td><td colspan="4">EOI address</td><td colspan="2"></td><td>error address</td><td>9</td></tr>
<tr><td colspan="8">Label file name (first 10 chars)</td><td>10</td></tr>
<tr><td colspan="7">Label file name (last 7 characters)</td><td>position number</td><td>11</td></tr>
<tr><td>edition number</td><td colspan="4">retention cycle</td><td colspan="3">creation date</td><td>12</td></tr>
<tr><td colspan="6">Multi-file name (6 chars)</td><td colspan="2">reel number</td><td>13</td></tr>
</table>

To facilitate rapid changes of IN and OUT sizes, bits 18-59 of words 3 and 4 are never used; all other fields not specified are reserved for future system use.

### 3.2.1 BASIC FILE ENVIRONMENT TABLE

Logical File Name (lfn) (42 bits)

The lfn field contains from one to seven alphanumeric display-coded characters starting with a letter, left justified; if less than seven are declared, unused characters are zero-filled. This field is used as a common reference point by the central processor program and the peripheral processor input/output routines.

The lfn parameter declared in an FET creation macro is also used as the location symbol associated with the first word of the FET. Thus, a reference to lfn in the file action requests is a reference to the base address of the FET.

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 should be zero.

## SCOPE CIO Codes in Octal (Circular Buffer I/O)

All codes indicated by - are illegal; all reserved codes are illegal. All codes are shown for coded mode operations; add 2 for binary mode. Example: 010 is coded READ, 012 is binary READ. Upon completion of operation, code/status in FET is changed to an odd number, usually by adding 1 to the code. In some cases, code is further modified to indicate manner in which operation concluded. Example: a READ function (010), at completion, becomes 011 (buffer full), 021 (end of logical record), or 031 (end of file).

| | |
|---|---|
| 000 | RPHR |
| 004 | WPHR |
| 010 | READ |
| 014 | WRITE |
| 020 | READSKP |
| 024 | WRITER |
| 030 | - |
| 034 | WRITEF |
| 040 | BKSP |
| 044 | BKSPRU |
| 050 | REWIND |
| 054 | - |
| 060 | UNLOAD |
| 064 | - |
| 070 | - |
| 074 | - |
| 100 | OPEN,READNR |
| 104 | OPEN,WRITENR |
| 110 | - |
| 114 | EVICT |
| 120 | OPEN,ALTERNR |
| 124 | - |
| 130 | CLOSE,NR |
| 134 | - |
| 140 | OPEN,READ |
| 144 | OPEN,WRITE |
| 150 | CLOSE |
| 154 | - |
| 160 | OPEN,ALTER |
| 164 | - |
| 170 | CLOSE,UNLOAD |
| 174 | - |

200 Series for special reads or writes (reverse, skip, non-stop, rewrite, etc.)

| | |
|---|---|
| 200 | - |
| 204 | - |
| 210 | - |
| 214 | REWRITE |
| 220 | - |
| 224 | REWRITER |
| 230 | - |
| 234 | REWRITEF |
| 240 | SKIPF |
| 244 | - |
| 250 | READNS |
| 254 | - |
| 260 | READN |
| 264 | WRITEN |
| 270 | - |
| 274 | - |

300 Series used for tape OPEN and CLOSE

| | |
|---|---|
| 300 | OPEN,REELNR |
| 304 | - |
| 310 | - |
| 314 | - |
| 320 | - |
| 324 | - |
| 330 | CLOSER,NR |
| 334 | - |
| 340 | OPEN,REEL |
| 350 | CLOSER |
| 354 | - |
| 360 | - |
| 364 | - |
| 370 | CLOSER,UNLOAD |
| 374 | - |

400 Series reserved for CDC

500 Series to be reserved for installations

600 Series

| | |
|---|---|
| 600 | - |
| 604 | - |
| 610 | - |
| 614 | - |
| 620 | - |
| 624 | - |
| 630 | - |
| 634 | - |
| 640 | SKIPB |
| 644 | - |
| 650 | - |
| 654 | - |
| 660 | - |
| 664 | - |
| 670 | - |
| 674 | - |

700 Series reserved for CDC

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 Device | | Allocation or Recording Technique | | |
|---|---|---|---|---|---|
| AA | 01 | 6603-I disk†† | 00 | system default, same as 03 | |
| | | | 01 | inner zone only | alternate sector halftrack |
| | | | 02 | outer zone only | |
| | | | 03 | both zones | |
| | | | †04 | both zones | sequential sector fulltrack |
| | | | †05 | inner zone only | |
| | | | †06 | outer zone only | |
| | | | 07 | CDC reserved | |
| | | | 10 | eight sector allocation (RESPOND) | |
| | | | 11-77 | CDC reserved | |
| AB | 02 | 6638 disk | 00 | system default, same as 01 | |
| | | | 01 | alternate sector halftrack | |
| | | | 02 | CDC reserved | |
| | | | 03 | same as 01 | |
| | | | 04-07 | CDC reserved | |
| | | | 10 | eight sector allocation (RESPOND) | |
| | | | 11-77 | CDC reserved | |
| †-- | 03 | data cell | | | |
| AC | 04 | 6603-II disk†† | xx | same as for 6603-I | |
| -- | 05, 06 | CDC reserved | | | |
| AP | 07 | 3234/854 disk pack drive | 00 | system default, same as 03 | |
| | | | 01-02 | CDC reserved | |
| | | | 03 | alternate triplets of sectors, one track | |
| | | | 04-77 | CDC reserved | |

†Codes are defined but supporting software is not provided by SCOPE.

††6603-I disk is a basic 6603 with or without field option 10098 (disk speedup) installed; 6603-II is a 6603 with both field options 10098 and 10124 (speedup augment) installed.

| Hardware Device | | | Allocation or Recording Technique | |
|---|---|---|---|---|
| †AF | 10 | 814 disk file | | |
| †AE | 11 | 3637/863 drum | | |
| AD | 12 | 3637/865 drum | 00 | system default, same as 03 |
| | | | 01-02 | CDC reserved |
| | | | 03 | alternate triplets of sectors, one halftrack |
| | | | 04-77 | CDC reserved |
| -- | 13-17 | CDC reserved | | |
| AX | 20 | ECS | 00 | normal system allocation |
| -- | 21-27 | CDC reserved | | |
| -- | 30-37 | reserved for installations, mass storage only | | |
| ††MT | 40 | 60x 1/2-inch 7-track, magnetic tape | (Right 6 bits in binary) | |
| | | | xxxx00 | HI density 556 bpi |
| | | | xxxx01 | LO density 200 bpi |
| | | | xxxx10 | HY density 800 bpi |
| | | | xxxx11 | CDC reserved |
| | | | xx00xx | Unlabeled |
| | | | xx01xx | SCOPE standard label (USASI) |
| | | | xx10xx | alternate label |
| | | | xx11xx | CDC reserved |
| | | | 00xxxx | SCOPE standard data format |
| | | | 01xxxx | X data format |
| | | | 10xxxx | S data format |
| †WT | 41 | 1-inch magnetic tape | 11xxxx | L data format |
| -- | 42-43 | CDC reserved | | |
| †TR | 44 | paper tape reader | | |
| †TP | 45 | paper tape punch | | |
| -- | 46-47 | reserved for installations | | |
| LP | 50 | 501, 512, 505 line printer | | |
| L1 | 51 | 501, 505 line printer | | |
| L2 | 52 | 512 line printer | | |
| | 53-55 | CDC reserved | | |
| -- | 56-57 | reserved for installations | | |
| CR | 60 | 405 card reader | | |
| -- | 61-65 | CDC reserved | | |
| -- | 66-67 | reserved for installations | | |
| CP | 70 | 415 card punch | | |
| DS | 71 | 6612 keyboard/display console | | |
| †GC | 72 | 252-2 graphic console | | |
| †HC | 73 | 253-2 hard copy recorder | | |
| †FM | 74 | 254-2 microfilm recorder | | |
| †PL | 75 | plotter | | |
| -- | 76-77 | reserved for installations | | |

† Codes are defined but supporting software is not provided by SCOPE.

†† Codes 4000-7777 require a device assigned by REQUEST card or function before file is opened.

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.

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.

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 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 is to be notified of error conditions. Generally if EP=0, the job is terminated. When labeled tape is checked, however the operator can terminate the job (DROP) or continue it (GO).

Error Bypass (EB) (1 bit)

Reserved for future use

Absolute Index (AI) (1 bit)

Reserved for system use. If AI=1, the request/return information field will contain the record block and PRU address of the logical record. This bit should be set only by EDITLIB.

DISPOSITION CODE (dc) 12 bits

The value in this field indicates the disposition to be made of a file when the job is terminated or the file is closed. This code has no effect if the file resides on a private disk pack or is a permanent file.

| Mnemonic | Value (Octal) | Disposition | Default File Name† |
|---|---|---|---|
| ††SC | xx00 | Scratch | - |
| CK | xx01 | Checkpoint | - |
| MF | xx02 | Multi-file tape | - |
| ††SV | xx04 | Save | - |
| PU | xx10 | Punch Hollerith | PUNCH |
| PB | xx12 | Punch Binary | PUNCHB |
| P8 | xx14 | Punch 80 Columns | - |
| †††FR | xx20 | Film Print | FILMPR |
| †††FL | xx22 | Film Plot | FILMPL |
| †††HR | xx24 | Hard Copy Print | HARDPR |
| †††HL | xx26 | Hard Copy Plot | HARDPL |
| †††PT | xx30 | Plot | PLOT |
| PR | xx40 | Print (501,505,512) | OUTPUT |

†A file with the specified name will automatically be assigned the corresponding disposition code value at job completion.

††Mnemonic is defined but supporting software is not provided by SCOPE.

†††SCOPE recognizes mnemonic and its value, but does not provide drivers.

| Mnemonic | Value (Octal) | Disposition | Default File Name† |
|---|---|---|---|
| P1 | xx41 | Print (501, 505 only) | - |
| P2 | xx42 | Print (512 only) | - |
| - | xx7x | Reserved to Installation | - |
| - | x1xx | Change common file | - |
| - | 2xxx | RESPOND file | - |
| - | 4xxx | EXPORT/IMPORT file | - |

All other codes are reserved to the system.

Length of FET ($\ell$) (6 bits)

The system FET length is determined as follows: FET first word address + 5 + $\ell$ = last word address + 1. The minimum FET length is five words ($\ell$= 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 ($\ell$= 1) is used if a working storage area is needed for blocking/deblocking. A length of eight words ($\ell$= 3) is used if the r bit is set, indicating an indexed file. Length is nine words ($\ell$= 4), if OWNCODE routines are declared. The maximum system FET length is 13 words ($\ell$ = 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. (FNT is discussed in section 3.1)

Physical Record Unit Size (PRU) (15 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.

†A file with the specified name will automatically be assigned the corresponding disposition code value at job completion.

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 or end-of-file, 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 and/or process the data before taking end-of-record or end-of-file 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

Words 7 and 8 are used for communication between the peripheral processor input/output routines and the running program depending on the device and file type.

For magnetic tapes with S or L data format, the structure of word 7 of the FET is:

|  | 59 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|
| word 7 |  | UBC |  | MLRS |  |

UBC (Unused Bit Count) Bits 24-29

The UBC field is used for a file declared to have either S or L format. For a READ or READSKP function, SCOPE will store into this field the number of low-order unused bits in the last data word of the record. The UBC field is not used during a READN request. For a WRITE, WRITER or WRITEF function, SCOPE will read the contents of UBC and adjust the length of the record accordingly.

For example, to write a single record of 164 decimal characters, the data length is 17, to the nearest CM word. The number of low-order unused bits in the last word would be 36. The user would set UBC = 36, set IN and OUT pointers to reflect 17 words of data, and then issue a WRITE or a WRITER.

SCOPE does not use the UBC field during a WRITEN request. UBC may range from 0 to 59, but will always be a multiple of 12 when set as a result of a read operation. If it is not a multiple of 12 for a write request, SCOPE will truncate the value to the nearest multiple of 12: if

UBC is 18, SCOPE will execute as though it were 12, and if UBC is 6, SCOPE will execute as though it were 0. The field in the FET remains unchanged.

MLRS (Maximum Logical Record Size) Bits 0-17

The MLRS field contains the size of the largest logical record to be encountered (considered as valid when either reading or writing) when the S or L tape format is used. The size is given in number of CM words.

The MLRS field is required for all S and L tape operations; therefore, a 7-word Fet is mandatory.

For S tape format, if MLRS = 0, the value of the maximum PRU is assumed to be 512 words. For L tape format, if MLRS = 0, the assumed maximum PRU is LIMIT - FIRST - 1 for standard reads and LIMIT - FIRST - 2 for READN.

For mass storage random files, the format of word 7 of the FET is:

| 59 | 29 0 |
|---|---|
| | record request/<br>return information |

The file indexing fields (record request/return information, record number index length and index address) are used for communication between the peripheral processor input/output routines and the CP programs. Index address and index length fields are declared when the FET is generated; the index buffer must be within the program field length. 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 other than the SCOPE indexing method, the following information is pertinent. At the start of writing a new logical record, if the random access bit and the record request/return information field are non-zero, the latter field is assumed to contain the address of a location within an index. The PP routine inserts into that location (in bits 0-23) the PRU ordinal (starting from 1) of the logical record. To read the record again, the random access bit should be set to non-zero and the PRU ordinal 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 operation or beginning-of-information after backward operation |
| $02_8$ | 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 (values are octal):

04 Irrecoverable parity error on last operation, or lost data on write.

10 When reading from magnetic tape, physical record size exceeds circular buffer or maximum allowable PRU size (MLRS for S and L tapes). When writing to S or L magnetic tapes, the FET is less than 7 words, or an attempt is made to write a noise record. When writing to mass storage, all mass storage space meeting the constraints imposed by the file (allocation style and/or equipment number) is in use or otherwise unavailable.

20 OPEN function redundant.

21 CLOSE function redundant.

22 Illegal function.

23 Index full.

24 FNT full.

25 An attempt was made to read or write record number n of a random file, but the index of the file is full.

26 An attempt was made to read a named record from a random file, but the name does not appear in the index.

27 An attempt was made to write a named record on a random file, file, but the name does not appear in the index, and there is no room to add a new name.

30 Buffer argument error.

31 A READ or SKIPF was attempted beyond EOI.

32 File name does not meet the requirements of section 3.2.1.

If 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.

## 3.3 LABELED TAPE FILES

The label macro with the following format specifies information stored in the file header label (appendix C).

```
lfn LABEL fln, ed, ret, create, reel, mfn, pos
```

File label name (fln). 17 alphanumeric display code characters starts with letter, left justified; if less than 17, it is binary zero-filled. The file label name ensures the correct file is referenced. The file is checked when opened if labeled tape REQUEST card or function is specified.

Edition Number (ed). 2 characters stored in field 8 of file header label for output tape and verified for input tape. If omitted, 01 is written in output tape label and FET, and no checking is done for an input tape.

Retention Cycle (ret). 3 digits specify number of days a tape is to be protected from accidental destruction. This field 12 is added to value of creation date to obtain expiration date written on output tape label or verified for input. When ret is 999, an expiration date of 99999 establishing permanent retention is placed in the tape label.

Creation Date. First two characters specify year, remaining three Julian day within year. Stored in field 10 of file header label for output tape and verified for input tape. If omitted, today's date as stored in SCOPE system, is written in this format in label output tape and in FET. For input tape, this field is read from label and stored in FET.

Reel Number. 4 characters stored in field 5 of the file header label for output tape and verified for input tape. If omitted, 0001 is written in output label and FET. For each reel, this field is increased by one at conclusion of processing for file trailer label and tape mark is written on the tape. When file is closed, this field is set to 0001.

### 3.3.1 MULTI-FILE TAPES

Multi-file Name (mfn); left justified alphanumeric display-coded characters starting with a letter; if less than 6 characters, binary zero-filled. Field 4 of the file header label identifies all files of a multi-file volume and must be the same for all files on a volume. If this field is omitted, only a single file may be generated or read in a volume set. Tapes with Y labels do not have the multi-file capability.

A multi-file tape must be declared by stating its disposition on a REQUEST card/function; 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 (pos), 3 digits; ignored unless a multi-file name is specified. If it is absent for a multi-file output tape, the file is assigned in sequence in which it is written and this position number is returned to FET. Overwriting a file on a multi-file volume set destroys the remaining files. If absent for a multi-file input tape, the value determined in the search for the file is stored in FET.

## 3.4 FET CREATION MACROS

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

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

Coded File - Sequential

lfn FILEC fwa, f, (WSA = $addr_w$, $l_w$), (OWN = eoi, err), LBL, DTY = dt,
DSC = dc, UPR, EPR, UBC = ubc, MLR = mlrs

Binary File - Sequential

lfn FILEB fwa, f, (WSA = $addr_w$, $l_w$), (OWN = eoi, err), LBL, DTY = dt,
DSC = dc, UPR, EPR, UBC = ubc, MLR = mlrs

Coded File - Random

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

Binary File - Random

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

lfn file name

fwa substituted in FIRST, IN, and OUT

f length of circular buffer + 1 (fwa + f is 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 12-bit code described in FET field descriptions

DSC Disposition code parameter

dc 12-bit code described in FET field descriptions

UPR User specifies 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.

EPR User specifies handling of error conditions.

UBC Unused bit count
ubc 6-bit code described in FET field descriptions (S and L tapes only). It causes the generation of a 7-word FET which is mandatory for S and L tapes.

MLR Maximum logical record size
mlrs 18-bit code described in FET field descriptions (S and L tapes only). It causes the generation of a 7-word FET which is mandatory for S and L tapes.

Examples:

To create a minimum FET for the standard INPUT file:

```
LBUFFER  EQU  65
INPUT FILEC BUFFER, LBUFFER
```

To create an FET for a binary random file:

```
LBUFFER  EQU  65
LINDEX   EQU  25
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
```

## 3.5 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 saves and restores all registers except A1, A6, X1 and X6.

### 3.5.1 CALLING SEQUENCE

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

<table>
<tr><td>59</td><td>41</td><td></td><td>39</td><td></td><td>29</td><td>17 0</td></tr>
<tr><td colspan="5">x</td><td>RJ</td><td>CPC</td></tr>
<tr><td>yyy</td><td>n</td><td>r</td><td></td><td colspan="3">z</td></tr>
</table>

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:

| 59 | 41 | | 39 | 17 0 |
|---|---|---|---|---|
| yyy | 0 | r | | base address of FET |

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

| 59 | 41 | | 39 | 35 0 |
|---|---|---|---|---|
| yyy | 1 | r | | z |

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.

## 3.6 SYSTEM COMMUNICATION MACROS

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

### 3.6.1 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. File action requests will be posted but not honored if the EOI or error bits are set and OWNCODE addresses are present where a call is issued to CPC. 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. If the optional recall parameter is non-blank, r is set to one.

REQUEST

REQUEST param

<table>
<tr><td>59</td><td>41</td><td></td><td>39</td><td>29</td><td>17 0</td></tr>
<tr><td colspan="4"></td><td>RJ</td><td>CPC</td></tr>
<tr><td>REQ</td><td>1</td><td>1</td><td colspan="2"></td><td>param</td></tr>
</table>

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.

<table>
<tr><td>59</td><td>47</td><td>35</td><td>28</td><td></td><td>23</td><td>17</td><td>11 0</td></tr>
<tr><td colspan="6">logical file name</td><td colspan="2">status</td></tr>
<tr><td></td><td>eq</td><td></td><td>a</td><td>pyqx</td><td colspan="2">dc</td><td>dt</td></tr>
</table>

The values for dc and dt are given in section 3.2.1 (Basic File Environment Table). The 4-bit parameter pyqx applies only when dt specifies 1/2-inch magnetic tape. p and q are interpreted only if dt = 4000, 4001, or 4002, otherwise the corresponding entries in dt override these two parameters. y is always interpreted. x is currently not interpreted. The meaning of pyqx is:

| | |
|---|---|
| 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 |

The 1-bit parameter, a, has the same effect for mass storage assignment as the * preceding the dt on the REQUEST card (section 2.4). If a = 0, operator action is requested. If a = 1, assignment is automatic. The eq field specifies the EST ordinal of the device to which the file is to be assigned. If dt is also specified, the type must match that of the EST entry. If eq is specified, no operator action is required regardless of the value of a.

If the lfn designated by REQUEST parameters is already associated with a file, the REQUEST function is ignored and control is returned to the calling program. Therefore, the REQUEST function should be issued prior to any reference to the logical file name, since a later reference to a nonexistent lfn will cause the name to be associated with an empty file on a mass storage device. 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 octal values may be returned in bits 9-13 of the status field.

| | |
|---|---|
| 22 | Illegal function; REQUEST function was issued without the recall bit. |
| 23 | Device type of specified EST did not match dt or automatic assignment indicated for a non-allocatable device. |
| 24 | FNT is full. |
| 26 | No equipment is available; requested equipment does not exist in the configuration or all equipment of this type is already assigned to this control point. |
| 30 | Duplicate file name, file already assigned. |

OPEN

OPEN lfn,x,recall

<table>
<tr><th>59</th><th>47</th><th>41</th><th></th><th>39</th><th>29</th><th>17 0</th></tr>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">000004</td><td>0</td><td>r</td><td colspan="2"></td><td>z</td></tr>
</table>

The OPEN function readies the file for processing. The x parameter specifies the operation to be performed: READ, WRITE, READNR, WRITENR, ALTER, ALTERNR, REEL or REELNR. The OPEN function causes information to be returned to the user via the FET.

OPEN will not reset the buffer pointers; the user is responsible for correctly setting the pointers.

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 read and written until it is closed.

> x = ALTER (Z = 160)

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

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 ALTER; 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

CLOSE lfn, x, recall

| 59 | 47 | 41 39 | | 29 | 17 0 |
|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC |
| 000007 | | 0 | r | | z |

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.

| | |
|---|---|
| x is absent (Z = 150) | File is set to beginning of information or beginning of current reel. Buffer pointers (IN and OUT) are set equal to FIRST. |
| x = NR (Z = 130) | File is not rewound. |
| x = UNLOAD (Z = 170) | Termination procedures for the file are executed. Files are flagged for no rerun (as indicated below) if the initial conditions for a job have changed. Files which have special names are given the corresponding disposition (section 3.2.1). A magnetic tape file is rewound and unloaded. The lock bit is set for a file on a private pack and the normal end-of-job processing will release the space and delete the reference to the file name. |

| File Type | Disposition | Device Type | Action |
|---|---|---|---|
| Local (section 1.3.1) | 0 | Allocatable | Release space on device, delete reference to file name |
| Local | immaterial | Non-allocatable | Release device, delete reference to file name |
| Local | ≠ 0 | Allocatable | Assign to system for disposition |
| Old Common[†] | - | - | Same as for local, except prohibit rerun |
| Common (section 1.3.1) | immaterial | Allocatable | Prohibit rerun, assign to system as a common file |
| Common | immaterial | Non-allocatable | Prohibit rerun, off equipment, assign to system as a common file |
| New Common[††] | - | - | Same as for common |

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.

[†] An old common file is one that was common at the beginning of a job and has since been released by a RELEASE card or a COMMON request. It is treated as a local file except for the RERUN feature.

[††] A new common file is one that was created during the job and has not been detached from the control point.

CLOSER

CLOSER lfn,x,recall

59 47 41 39 17 0

<table>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">000007</td><td>0</td><td>r</td><td colspan="2"></td><td>z</td></tr>
</table>

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

59 47 41 39 29 17 0

<table>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">000003</td><td>0</td><td>r</td><td colspan="2"></td><td>000114</td></tr>
</table>

EVICT releases to the system all space occupied by a file on mass storage and makes it available for use by either the releasing program or other programs. The logical file name is retained. EVICT is ignored for permanent files, and an information message is issued.

CONTRLC

CONTRLC addr

59 41 39 29 17 0

<table>
<tr><td colspan="5"></td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">ACE</td><td>1</td><td>1</td><td colspan="2"></td><td>addr</td></tr>
</table>

With the CONTRLC function, a CP program can read or backspace the control card record. addr is the address of a word in the following format:

59 17 0

| | |
|---|---|
| Reserved for system use | code |

code is both the function and status reply as follows:

READ code = $000010_8$

The next control card is placed in RA+70 to RA+77 with the space remaining between the end of the card and RA+100 zeroed.

BKSP code = $000040_8$

The control statement pointer is reset to the previous statement.

When the function is completed bit zero of the code is set to one. If on the read there are no more control cards, the area RA+70 to RA+77 is zeroed and bit 4 is set to 1 for end of record. If the record is set to the start of the control card record when a BKSP function is issued, the statement pointer is unchanged and the end of record reply is issued.

With this request the user can position the control card record for SCOPE job processing, and it is his responsibility to position the record properly, for example in front of an EXIT card; otherwise results are unpredictable.

## 3.6.2 DATA FUNCTIONS

READ READ lfn, recall

| 59 | 47 | 41 | 39 | 29 | 17 0 |
|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC |
| 000002 | | 0 | r | | 000010 |

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 does not contain enough room for the next physical record

End-of-record or end-of-file is encountered (not applicable for S or L tapes)

End-of-information is encountered

An error is encountered (see FET, section 3.2)

For S or L tapes, one physical record is read

Mode is determined by bit 1 in first word of FET. If end-of-record (bit 4) is set upon entry to CPC, no operation is performed. For S or L tapes, the unused bit count is returned to the UBC field in FET word 7.

READN

READN lfn, recall

| 59 | 47 | 41 | 39 | 29 | 17 00 |
|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC |
| 000003 | | 0 | r | | 000260 |

S or L magnetic tape only: The READN function reads data from tape to the circular buffer until one of the following conditions occurs:

Buffer does not contain enough room for next record (MLRS+1 words), end-of-file is encountered, end-of-information is encountered, error is encountered.

The mode of the file is determined by bit 1 in the first word of the FET. Word 7 of the FET must contain the size of the largest possible logical record when this function is used (appendix L).

READN enables non-stop reading for maximum tape throughput. READN can be used only for S or L tapes. As long as the user provides sufficient room in his buffer (room for two records and their header words), tape reading continues without releasing and reloading the PP between logical records, and maximum utilization of interrecord gap time is realized. The concept of circular buffering is retained; however, a header word is placed in the buffer along with data from each logical record. It precedes the data and contains the number of CM words in the logical record and the number of unused bits in the last data word. IN is moved by the I/O system after a complete logical record together with its header word have been placed in the buffer. The format of the header word is:

| 59 | 29 | 23 | 17 00 |
|---|---|---|---|
| | No. Bits unused | | #CM words |

READSKP

READSKP lfn,ℓ,recall

| 59 | 47 | 41 | | 39 | 29 | 17 | 13 0 |
|---|---|---|---|---|---|---|---|
| SA1 | lfn | | | | RJ | CPC | |
| 000003 | | 0 | r | | | ℓ | 00020 |

READSKP functions like READ, except that if the buffer is filled prior to the end-of-record, the rest of the information is discarded and the file will be positioned ready to read the next logical record. If the MLRS field in the FET is zero, the buffer size is set to 512 words for S tapes and to the LIMIT - FIRST - 1 for L tapes.

If a level parameter (ℓ) is specified, information is skipped until the occurrence of an end-of-record with a level number greater than or equal to the one specified. Only a level 17 (EOF) is recognized by S and L tapes. Any other level parameters in the request will be ignored.

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

| 59 | 47 | 41 | | 39 | 29 | 17 0 |
|---|---|---|---|---|---|---|
| SA1 | lfn | | | | RJ | CPC |
| 000003 | | 0 | r | | | 000000 |

Magnetic Tape (SCOPE or X format only)

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 is read into the buffer. The mode is determined by bit 1 of the first word of the FET. For coded files, 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.

READNS READNS lfn, recall

<table>
<tr><td>59</td><td>47</td><td>41</td><td></td><td>39</td><td>29</td><td>17 0</td></tr>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">000002</td><td>0</td><td>r</td><td colspan="2"></td><td>000250</td></tr>
</table>

Mass storage files only

READNS operates in the same way as READ, except that it is not ignored by CPC if the last code/status in the FET is 02X or 03X, and reading does not necessarily stop at the end of a logical record. Instead, a READNS operation terminates under any of the following conditions:

1. Next PRU will not fit into the circular buffer.
2. A zero length logical record (any level) has been read.
3. A level 16 or 17 logical record (any length) has been read.
4. End-of-information has been encountered.
5. An irrecoverable error is detected; the error code ee is usually 04 for parity error.

The status stored in the FET for each case is as follows:

| | |
|---|---|
| case 1 | 000011 coded, 000013 binary |
| case 2 or 3 | 740031 coded, 740033 binary |
| case 4 | 741031 coded, 741033 binary |
| case 5 | 0ee011 coded, 0ee013 binary |

In cases 2 and 3 the level of the terminating logical record is lost.

READIN READIN lfn,x

READIN may be used for indexed or sequential mass storage files or tape files. The format depends on file mode, file index, working storage area, and the x parameter as shown in the following pages. In the descriptions 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 or end-of-file 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; the next READIN request to transfer data from that file will begin with the first word of the specified record.

1. x is absent: READIN lfn

| 59 | 29 | 17 0 |
|---|---|---|
|  | RJ | IOREAD |
|  |  | lfn |

This form of the READIN request transfers data to the working storage area. It may be used for tape or mass storage files since it transfers data from buffer to working storage area; if the buffer is empty a READ is issued. If lfn is a mass storage file, it may be sequential or random.

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

| 59 | 29 | 17 0 |
|---|---|---|
|  | RJ | IORR |
|  |  | lfn |
| name |  |  |

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

3. x is of the form m where m is a logical record number: READIN

| 59 | 29 | 17 0 |
|---|---|---|
|  | RJ | IORR |
|  |  | lfn |
|  |  | m |

This form of the READIN request causes the logical record number m of the file, 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. The next record is the first logical record of the file if this is the first request, or the last logical record read + 1, for any subsequent READIN.

WRITE

WRITE lfn,recall

| 59 | 47 | 41 |  | 39 | 29 | 17 0 |
|---|---|---|---|---|---|---|
| SA1 | lfn |  |  |  | RJ | CPC |
| 000002 |  | 0 | r |  |  | 000014 |

The WRITE function causes information to be written from the circular buffer. For mass storage files and SCOPE standard and X tapes, only full PRU's are written. Writing continues until the buffer is empty or there is not enough data in the buffer to fill a PRU. For S and L tapes only one record is written.

For S or L tapes, if the requested record length, indicated by the OUT and IN pointers, is greater than MLRS, device-capacity-exceeded status is returned (10) and the record is not written.

WRITER

WRITER lfn,ℓ,recall

<table>
<tr><th>59</th><th>47</th><th>41</th><th></th><th>39</th><th>29</th><th>17</th><th>13 0</th></tr>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td colspan="2">CPC</td></tr>
<tr><td colspan="2">000003</td><td>0</td><td>r</td><td colspan="2"></td><td>ℓ</td><td>00024</td></tr>
</table>

This function is processed the same as WRITE, with the following exceptions:

> For mass storage files, SCOPE standard tapes, and binary X tapes, 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.
>
> For coded X tapes, data is written in 136 character PRU's until the buffer is empty. No short 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. The ℓ field is ignored for S, L and X tapes. For S and L tapes the WRITER request is identical to the WRITE request, unless made through CPC for records less than $512_8$ words.

WRITEF

WRITEF lfn,recall

<table>
<tr><th>59</th><th>47</th><th>41</th><th></th><th>39</th><th>29</th><th>17 0</th></tr>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td>CPC</td></tr>
<tr><td colspan="2">000003</td><td>0</td><td>r</td><td colspan="2"></td><td>000034</td></tr>
</table>

For SCOPE standard tapes the WRITEF function produces a logical end-of-file mark; it is written as a zero-length physical record of level $17_8$. When this function is issued, any data present in the buffer is written and terminated with a level zero end-of-record. If the buffer is empty and the last operation was WRITE, a zero length PRU is written.

For an S or L tape, if data in the buffer is less than or equal to MLRS, it is written to tape followed by a physical tape mark. If data in the buffer exceeds MLRS, nothing is written and device capacity exceeded is returned to the FET.

For X tapes, data is written as in WRITER, and terminated with a physical tape mark.

WRITEN

WRITEN lfn, recall

| 59 | 47 | 41 | 39 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC | |
| 000003 | | 0 | r | | 000264 | |

Magnetic tape only; S and L tapes only:

WRITEN improves throughput by writing one or more logical records on tape. As long as the user provides data ahead of the I/O system, tape writing continues without releasing and reloading the PP between logical records, making full use of the interrecord gap on tapes (as long as the circular buffer contains at least two records). The concept of circular buffering is retained. However, a header word must precede each logical record in the buffer. This header word gives the number of CM words in the logical record and the number of unused bits in the last data word. If the number of unused bits is not mod 12, the I/O system will make it so for execution by subtraction. The UBC field in the FET is not changed, however,

OUT is moved by the I/O system after a complete logical record has been written to tape. Writing continues until there is no data in the buffer or until an end-of-file or error condition is detected. No writing will take place unless the difference between OUT and IN is greater than the number of CM words in the logical record. The user should not move IN beyond the header word until the header and the complete record are in place. An error will result if SCOPE detects this condition. The format of the header word is:

| 59 | 29 | 23 | 17 | 00 |
|---|---|---|---|---|
| | No. Bits unused | | #CM words | |

WPHR

WPHR lfn, recall

| 59 | 47 | 41 | 39 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC | |
| 000003 | | 0 | r | | 000004 | |

SCOPE or X magnetic tape only:

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

If the buffer contains less than 512 (decimal) words, IN and OUT pointers in the FET are set to the same value at completion of writing to indicate an empty buffer. Only internal to external BCD conversion is performed. If the buffer contains more than 512 words when the request is issued, the first 512 words are written and IN and OUT pointers are set to show that words remain in the buffer. Device-capacity-exceeded status (10) is returned.

A WPHR function issued for any device other than 1/2-inch magnetic tape in SCOPE or X format is ignored and an illegal function status (22) is returned.

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

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.

For a binary mode file, the entire working storage area is transferred to the circular buffer. In coded mode, trailing blanks are removed and a zero byte (end-of-line) is inserted as data is transferred to the buffer. The WRITER function may be requested to terminate record writing; but if the file is indexed, and no record is being written, the 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. This request terminates any previous action on the file, locates the specified logical record, and sets 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.

1. x is absent: WRITOUT lfn

| 59 | 29 | 17 0 |
|---|---|---|
|  | RJ | IOWRITE |
|  |  | lfn |

This form of the WRITOUT request adds data from the working storage area to the logical record currently being constructed; if the circular buffer contains no data and the file is indexed, this request is rejected as the system cannot determine which record is being addressed.

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

| 59 | 29 | 17 0 |
|---|---|---|
| | RJ | IORW |
| | | lfn |
| name | | |

This form of the WRITOUT request transfers data from the working storage area and adds it to the logical record currently being constructed; if there is none in the circular buffer and if the file is indexed, this request will be rejected since the system cannot determine which record of the file is being addressed.

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

| 59 | 29 | 17 0 |
|---|---|---|
| | RJ | IORW |
| | | lfn |
| | | m |

This form of the WRITOUT request begins constructing 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.

Requests 2 and 3 above perform the following functions:

1. Perform a WRITER on the file if its previous status was write, or the buffer contains data as a result of a previous WRITOUT.
2. Set the buffer to empty, and the FET status to write completed.
3. Set up the random file index and FET to point to the correct record.
4. Transfer the working storage area to the buffer.
5. Call WRITE if the buffer contains at least one PRU of data.

The WRITOUT lfn, m or /name/ statement is used only to begin an indexed record. The record can be continued with WRITOUT lfn statements. It should be terminated by a

WRITER lfn

statement, although if this step is neglected, the next WRITOUT lfn, m or /name/ statement for the same file will cause it to be carried out as step 1 above.

If the working storage area size is zero, nothing is transferred to the buffer, but steps 1, 2, 3 and 5 above are carried out.

**REWRITE**
**REWRITER**
**REWRITEF**

REWRITE lfn, recall
REWRITER lfn,ℓ, recall
REWRITEF lfn, recall

} Mass Storage Only

<table>
<tr><th>59</th><th>41</th><th></th><th>39</th><th>29</th><th>17</th><th>13 00</th></tr>
<tr><td>SA1</td><td colspan="3">lfn</td><td>RJ</td><td colspan="2">CPC</td></tr>
<tr><td>yyy</td><td>0</td><td>r</td><td colspan="2"></td><td>ℓ</td><td>zzz</td></tr>
</table>

| | |
|---|---|
| yyy | 002 for REWRITE<br>003 for REWRITER and REWRITEF |
| zzz | 214 for REWRITE<br>224 for REWRITER<br>234 for REWRITEF |

The REWRITE functions make use of a previously allocated file on a mass storage device to update records in an existing file without changing its index or mass storage allocation. They should be used to replace a record in a sequential or random indexed file by a record of the same length. A knowledge of the structure and record lengths of the file to be rewritten is essential. If the rewritten record is shorter or longer than the original record, results are unpredictable. The system cannot determine the length of the original record, so there is no protection from over or underwriting, nor is a diagnosis of such a condition made. The system guarantees only that a rewritten record does not extend beyond the end-of-information. It issues a diagnostic, the write takes place up to end-of-information, and end-of-information indicators are not moved. The user may destroy an index previously written on the file.

Writing always begins at the current position of the file. The file is rewritten with information from the CIO buffer. (REWRITE transfers a minimum of one PRU of data.) For REWRITER, an end-of-record, level number ℓ is written at the end of the data transferred. For REWRITEF, a logical end-of-file (level 17) is written at the end of the data transferred. After REWRITEF is used, other data in the file remains allocated to the file; therefore an end-of-file may occur in the middle of the file. Rewrite operations do not change the storage allocation of the file; no additional record blocks are assigned, nor is the index of a random file modified. It is, in effect, a write-in-place operation. If rewrite is used for a common file and the job is rerun by operator or user, the data of that common file will not be the same as when it was previously used by that job.

Application of rewriting for sequential files

Rewriting begins at the current file position. After each request is executed, the current file position is updated to point to the PRU that follows the last PRU rewritten by the request. A series of REWRITE operations, followed perhaps by a REWRITER, can be used to replace a single logical record.

When a logical record is to be replaced with a record of the same length in a single rewrite operation, REWRITER should be used.

When a record is replaced with a shorter record, the new record might be followed by another record which is the last part of the original record. For example, if the original record is 120 words (one full length and one short PRU) and it is replaced by REWRITER with a 60-word record, the end result is one 60-word record followed by another 56-word record (a short PRU) which is the second PRU of the original record.

If a logical record is replaced with a longer record, the new record will overwrite more than one record, and part of the surplus might appear as a logical record that immediately follows the new record. End-of-information, however, is not destroyed.

Application of rewriting for random indexed files

Rewriting begins at current file position. A logical record may be replaced by a single REWRITER or by a series of REWRITE's followed by a REWRITER. The programmer must reposition the file for each logical record to be rewritten. Otherwise, records will be rewritten in the order they were originally written — not in index order.

To position an indexed file for rewrite of a particular record, the user may use one of two methods:

> Set up the file's FET the same as for a random READ; i.e., insert into the Record Request/Return Information field in FET+6 the record address found by searching the file's index.
>
> Use the WRITIN function, which causes the system to search the user's index and set the necessary FET fields. (WRITIN will also issue a REWRITE request in some cases. Refer to the WRITIN description for detail).

Once the file is positioned for a record, a REWRITE/REWRITER sequence, or a WRITIN/REWRITER sequence, can be executed without further repositioning. The FET+6 field will be cleared by the system after the first REWRITE (or after WRITIN) and should remain cleared until repositioning for another record is required.

The methods of rewriting, and the results of underwriting or overwriting a logical record are the same as for sequential files. Index integrity, however, can be destroyed by the user if he writes shorter or longer records than those in the original file. A longer record destroys records originally written just after the one being replaced (not necessarily records which appear next in the index). A shorter record may create an extra record which is not represented in the index. The index is never modified by the system for REWRITE. The PRU's and RB's allocated to the file remain assigned, though they might contain some useless unaccessible data as a result of rewriting with shorter records. Such unused areas can be utilized later when a longer record (but shorter than or equal to the original) is rewritten. Use of such techniques requires an understanding of the file's logical and internal structure.

### WRITIN

WRITIN lfn,x

WRITIN provides automatic index and working storage area management for REWRITE. WRITIN is a write-in-place function, unlike WRITE which writes at end-of-information. It may be used for indexed or sequential mass storage files. The results of a WRITIN request depend 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.

WRITIN transfers n words from the working storage area to the circular buffer, putting them into the area where the file is currently positioned. A REWRITE 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 REWRITER function may be requested to terminate writing of a record.

If a working storage area is not specified, a WRITIN request has no effect and gives no error indication, unless it addresses a file with a name or number index. In that case, the request will terminate any previous action on the file, locate the specified logical record, and set up the pointers, so that the next request for REWRITE or REWRITER or another WRITIN (without x parameter) to continue writing the current record on that file will begin with the first word of the specified record. If the next request is WRITOUT or WRITIN, the current logical record will be terminated by execution of a REWRITER of level 0 before the function is executed.

x is absent: WRITIN lfn

| 59 | 29 | 17 0 |
|---|---|---|
| | RJ | IOREWRT |
| | | lfn |

This form of the WRITIN request transfers data from the working storage area to the buffer.

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

| 59 | 29 | 17 0 |
|---|---|---|
| | RJ | IORRW |
| | | lfn |
| name | | |

This form of the WRITIN request begins rewriting 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 named record must have been written previously, so that the file index has the name with a storage address already assigned.

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

<table>
<tr><th>59</th><th>29</th><th>17 0</th></tr>
<tr><td></td><td>RJ</td><td>IORRW</td></tr>
<tr><td colspan="2"></td><td>lfn</td></tr>
<tr><td colspan="2"></td><td>m</td></tr>
</table>

This form of the WRITIN request begins rewriting logical record number m on the file named lfn using the words in the working storage area as the first n words of the record. The file must be indexed, either by name or by number, and the logical record referenced must have been written previously.

### 3.6.3 POSITION FUNCTIONS

SKIPF SKIPF lfn,n,ℓ,recall

<table>
<tr><th>59</th><th>47</th><th>41</th><th></th><th>39</th><th>29</th><th>17</th><th>13 0</th></tr>
<tr><td>SA1</td><td colspan="4">lfn</td><td>RJ</td><td colspan="2">CPC</td></tr>
<tr><td colspan="2">000003</td><td>0</td><td>r</td><td colspan="2">n</td><td>ℓ</td><td>00240</td></tr>
</table>

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 777776$_8$. When n = 777777$_8$, a tape file is not positioned; however, a disk file is positioned at 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, using the illustration shown on page 1-9, 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 ℓ = 17$_8$, skipping is performed until record level 17 or an end-of-file mark (tape mark) is encountered.

For S and L tapes, if $\ell \neq 17_8$, the level is assumed to be zero.

If the level parameter is absent, the $\ell$ 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.

On external tapes, level numbers are not appended to records. However, level numbers may be specified for SKIPF requests. If the level number 17 is specified, a skip to end-of-file is performed. For other level numbers, one record is skipped.

### Backspace Functions

Backspace functions will not go beyond the beginning of the current reel of magnetic tape. If beginning of reel is encountered before the requested number of backspaces, the beginning of information status bit is set. Parity errors encountered during backspace operations are ignored.

### Reverse Functions

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

If the last operation on a magnetic tape was a write function, trailer label procedures will be executed before the reverse motion takes place. For X tapes, four tape marks will be written.

BKSP

BKSP lfn, recall

| 59 | 47 | 41 | | 39 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|---|
| SA1 | lfn | | | | RJ | CPC | |
| 000003 | | 0 | r | | | 000040 | |

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

| 59 | 47 | 41 | | 39 | 35 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| SA1 | lfn | | | | | RJ | CPC | |
| 000003 | | 0 | r | | n | | 000044 | |

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

| 59 | 47 | 41 | | 39 | 35 | 29 | 17 | 13 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| SA1 | lfn | | | | | RJ | CPC | | |
| 000003 | | 0 | r | | n | | ℓ | 00640 | |

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 logical record 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 $\ell$ to the last logical record of each group, and a level number less than $\ell$ to all other logical records.

Then SKIPB lfn, ,$\ell$ will skip the file backward to the beginning of the logical record group which immediately follows a logical record of level $\ell$. 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 $\ell$ 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.

On external tapes, level numbers are not appended to records; however, they may be specified for SKIPF requests. If level 17 is specified, a skip to end-of-file is performed. For other level numbers, one record is skipped. For S and L tapes, only levels 0 and 17 are recognized. If $\ell \neq 17$, zero level is assumed.

REWIND

REWIND lfn, recall

<table>
<tr><th>59</th><th>47</th><th>41</th><th></th><th>39</th><th>29</th><th>17</th><th>0</th></tr>
<tr><td colspan="1">SA1</td><td colspan="4">lfn</td><td>RJ</td><td colspan="2">CPC</td></tr>
<tr><td colspan="2">000003</td><td>0</td><td>r</td><td colspan="2"></td><td colspan="2">000050</td></tr>
</table>

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

UNLOAD

UNLOAD lfn, recall

| 59 | 47 | 41 | 39 | 29 | 17 0 |
|---|---|---|---|---|---|
| SA1 | lfn | | | RJ | CPC |
| 000003 | | 0 | r | | 000060 |

UNLOAD operates in a manner similar to REWIND. If the file resides on magnetic tape, the tape is rewound, and then unloaded.

COMMON

COMMON addr, recall

| 59 | | | 29 | 17 0 |
|---|---|---|---|---|
| | | | RJ | CPC |
| C T S | 1 | r | | addr |
| | 41 | 39 | | 17 0 |

This function may be used to manipulate common files from a central processor program. A new common file is one which was created during the job and has not been detached from the control point. An old common file is one which was common at the beginning of job and has since been released by a RELEASE card or the COMMON request (n = 2). An old common file is treated as a local file except for the RERUN feature. Addr is the address of a word in the following format:

| 59 | 17 0 |
|---|---|
| lfn | n |

lfn is left justified with binary zero fill. n specifies both the function and a status reply word.

Attach or create common file lfn: n = 0

n = 1 Request was completed normally; either a local file lfn was changed to a new common file or an unassigned common file lfn was attached to this control point. In the latter case the job will be marked as unable to be rerun. If the file resides on a non-allocatable device, the equipment will be requested and reserved until the common file is released.

n = 3 A common file named lfn is attached to another control point, but no local file named lfn is at the calling control point.

n = 5 No common file lfn is in the system at present, and no local file lfn is at the calling control point.

n = 7 Duplicate names; the calling program has a local file named lfn and a common file in the system is named lfn.

n = 11 A common file named lfn is already at this control point.

n = 13 The common file lfn resides on equipment which cannot be assigned to this control point.

n = 15 The file is a permanent file.

Release common file lfn: n = 2

n = 1 Request was completed normally; the common or the new common file lfn assigned to this control point will be changed to type local at job termination. If the file is an old common file the job will be marked as unable to be rerun.

n = 3 Common file named lfn is not at this control point.

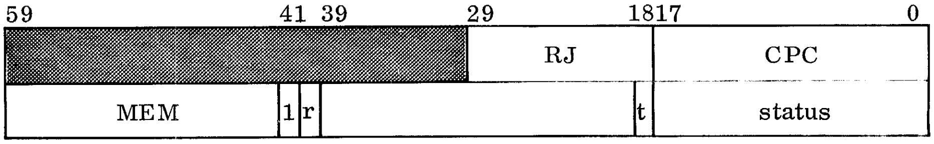
Detach common file lfn: n = 4

n = 1 Request was completed normally; the common file or the new common file lfn was detached from this control point; lfn is no longer available to this control point. The job will be marked as unable to be rerun. Operation is the same as a CLOSE, UNLOAD, except that no index is written.

n = 3 Common file named lfn is not at this control point.

### 3.6.4 SYSTEM ACTION REQUESTS

MEMORY MEMORY type, 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.

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

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

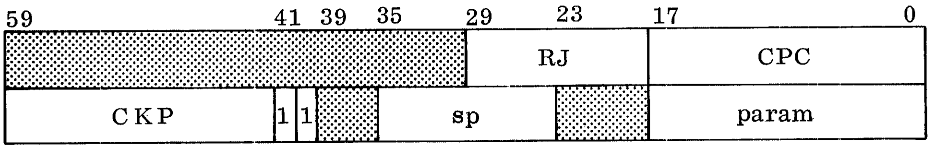
If the location addressed by status initially contains zero, no field length is altered: the current field length is returned in the upper half of the location and bit 0 is set to one.

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 param, sp



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 conductive 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   Address of a user-supplied parameter list containing logical file names for which a checkpoint is to be made. (See section 8.1 for format of list.)

sp   Special process flag that indicates all mass storage files are to be processed (sp = zero) or only a limited set of files are to be processed (sp = non-zero). If this parameter is omitted, a checkpoint is made for all local files associated with the user's control point.

## 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

| 59 | 47 | 41 |  | 39 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|---|
| SA1 | lfn |  |  |  | RJ | CPC |  |
| 000001 |  | 0 | 1 |  |  | 777777 |  |

lfn is the base address of a file environment table. Control returns to the program when bit 0 of the code and status field becomes a one, indicating completion of an input/output request 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 if some useful processing can be done between initiating and completing an input/output operation.

RECALL

| 59 | 41 |  | 39 | 29 | 17 | 0 |
|---|---|---|---|---|---|---|
|  |  |  |  | RJ | CPC |  |
| RCL | 1 | 0 |  |  | 000000 |  |

When RECALL does not specify lfn, the central processor is 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

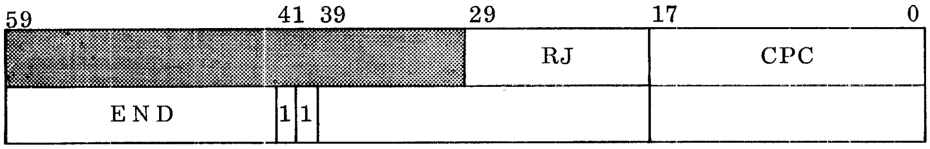
| 59 | 41 |  | 39 | 35 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| ▨ |  |  |  |  | RJ |  | CPC |  |
| MSG | 1 | r |  | x |  |  | addr |  |

The MESSAGE function enters a message into the job dayfile. The message must be stored in display code and must not contain any characters with display codes greater than $57_8$ (appendix A). The maximum message length is $80_{10}$ characters because of truncation by the dayfile routine. SCOPE considers the message to end either at the first word with all zeros in the rightmost 12 bits or at the 80th character, whichever comes first.

Any characters beyond the 40th appear on a second line. If the x parameter is non-blank, the message is displayed but not entered into the dayfile. addr is the address of the start of the message.

ENDRUN

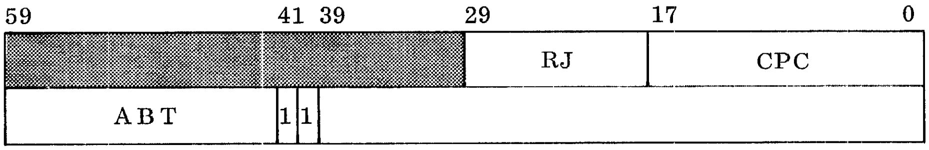
ENDRUN



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

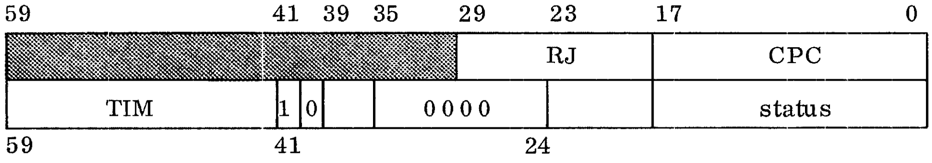
ABORT



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



Before clearing RA+1, monitor returns in status the job time limit and the central processor time already used by the job in the following binary format:

| 59 | 35 | 11 | 0 |
|---|---|---|---|
| TIME limit (seconds) | CPU time (seconds) | milliseconds | |

CLOCK

CLOCK status

| 59 | 41 | | 39 | 35 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | RJ | | CPC | |
| TIM | 1 | 0 | | 0002 | | | status | |

Before clearing RA+1, monitor returns the current reading of the system clock in status, in the following format:

| 59 | | | | | | 0 |
|---|---|---|---|---|---|---|
| (* or blank) | hh | . | mm | . | ss | * |

Character 1 = * if time from deadstart, blank time is entered by operator at deadstart.

DATE

DATE status

| 59 | 41 | | 39 | 35 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | RJ | | CPC | |
| TIM | 1 | 0 | | 0001 | | | status | |

Before clearing RA+1, monitor returns in status the current date, as typed by the operator, with one leading blank and one trailing blank.

JDATE

JDATE status

| 59 | 41 | | 39 | 35 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | RJ | | CPC | |
| TIM | 1 | 0 | | 0003 | | | status | |

Before clearing RA+1, the Julian date is returned in status in the following format: (yyddd is in display code)

| 59 | 29 | 0 |
|---|---|---|
| Zeros | yyddd | |

RTIME

RTIME status

| 59 | 41 | 40 | 39 | 35 | 29 | 23 | 17 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | RJ | | CPC | |
| TIM | 1 | 0 | | 0004 | | | status | |

Before clearing RA+1, the real time clock maintained by monitor is returned in status in the following format:

| 47 | 35 | 23 | 0 |
|---|---|---|---|
| seconds (modulo 4096), | millisec. (modulo 1000) | milliseconds | |

LOADER

LOADER param

| 59 | 29 | 17 | 0 |
|---|---|---|---|
| ▨ | RJ | 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, a fatal error flag is returned to the user; the PPLOADR will leave the file positioned at its original starting point, and the CPLOADR will leave the file positioned at end-of-file.

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.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 59 | | | | | | | | | | | | | 17 | 0 |
| lfn (logical file name) | | | | | | | | | | | | | sl | |
| $l_1$ | $l_2$ | | r | p | u | v | m | k | s | f | c | lwa | fwa | |
| 59 | 53 | 47 | | 43 | | 41 | | 39 | | 37 | 35 | | 17 | 0 |

lfn One of the following:

Name of file from which programs will be loaded (sl may or may not be 0)

Name of an entry point in a program (sl = 0)

Program name (sl = 0)

Zero (sl ≠ 0)

sl If non-zero, the 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:

| | | |
|---|---|---|
| 59 | 17 | 0 |
| subprogram name | | |

$l_1$ Segment level if s ≠ 0 and v = 0. Primary overlay level if s = 0 and v ≠ 0.

$l_2$ Secondary overlay level if v ≠ 0.

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

p Partial map bit. If p ≠ 0 a one-line partial core map is given.

u Library flag. When u ≠ 0, and v = 0, sl refers to a list of externals that are to be satisfied by loading from the system library. When u ≠ 0 and v ≠ 0, an overlay will be loaded from the system library. In this case sl = 0 and lfn = program name.
When u ≠ 0, v = 0, s ≠ 0, and lfn = 0, sl points to a list of sections, a segment, or list of programs to be loaded as a segment from the library.

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

m NOMAP flag. If m ≠ 0, all maps are suppressed.

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 relocatable 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:

| | 59 | 53 | 37 | 36 | 35 | 17 | 0 |
|---|---|---|---|---|---|---|---|
| Word 1 | 0 | | | | | | |
| Word 2 | $\ell$ | | ne | fe | aa | ea | |

$\ell$ Level at which segment was loaded. $\ell = 0$ if segment loading not requested.

ne Non-fatal error flag. ne $\neq 0$ 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.

No entry point for a named transfer.

fe Fatal error flag. fe $\neq 0$ 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 point. If v $\neq 0$, 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:

| 59 | 17      0 |
|---|---|
| $name_1$ | addr |
| $name_2$ | addr |
| ⋮ | |
| 0 | |

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

The file is not rewound before loading. If no file is found, the system library is searched as in the following example. Subprograms will be loaded from lfn until end-of-information is encountered. $l_1$ and $l_2$ are ignored. If c ≠ 0, loading will be completed.

Load Named Entry:

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

sl = 0

u = 0

v = 0

k = 0

s = 0

The File Name Table is searched first; if found, action is a Load from file as in the previous example; if not found, the system library is searched. $l_1$ and $l_2$ are ignored. If $c \neq 0$, loading will be completed.

Load Segment from File:

The segment defined by the list at sl will be loaded from lfn at level $l_1$. If $l_1 >$ current segment level, the segment will be loaded at the current level + 1. If $l_1 \leq$ current level, segments at a higher level will be removed. If a subprogram specified in the segment list is not located on lfn after the entire file has been searched, the fatal error flag will be set. lfn is not rewound prior to loading.

lfn = name of file

sl = address of list, contains a segment name or section names and subprogram names only

$l_1$ = desired level

u = 0

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

$l_1$ & $l_2$ ignored

u = 0

v = 0

k = 0

s = 0

If $c \neq 0$, loading will be completed.

Load Overlay from File:

lfn = name of file

sl = 0

$l_1$ = primary level

$l_2$ = secondary level

u = 0

v = 1

r, p, m, k, s, f and c are ignored.

The overlay file (constructed during the initial load from overlay cards and binary text) is searched for the unique identifier $l_1$, $l_2$. If FWA = 0, the overlay will be loaded at the address at which the overlay was created, otherwise the load will be at FWA. The absence of such an overlay will cause the loader to set the fatal error flag. No map is produced.

Load Overlay from System Library:

lfn = program name

sl = 0

$l_1$ = primary level

$l_2$ = secondary level

u = 1

v = 1

r, p, m, k, s, f, and c are ignored.

LOADREQ

LOADREQ, param

| 59 | | | | 29 | 17 | 0 |
|---|---|---|---|---|---|---|
| | | | | RJ | CPC | |
| LDV | 1 | 0 | | | param | |
| 59 | 41 | 39 | | | 17 | 0 |

This request is used to load absolute overlays from the system library or a user file.

The LOADREQ request results in a call to the PP overlay LDV. The functions of LDV depend upon the value of param:

1. A non-zero param has the same significance as when used in the LOADER request (p. 3-56). LDV examines the two-word entry pointed to by param.

   - If the v bit in word 2 indicates that this operation is not an overlay load, LDV calls LDR. The results are unpredictable.
   - If this is an overlay and the u bit in word 2 indicates that this is from a user file, LDV calls LDR to perform the operation.
   - If this is an overlay and the u bit in word 2 indicates that this is from the library, LDV loads into the user's area the CP overlay named in bits 18-59 of word 1 of the two-word entry pointed to by param. If the named CP overlay does not appear in the library, LDR is called to handle the error condition.

2. If param is zero or omitted, this call is to the loader for loading from a file. The name of the file must be specified in RA + 64. LDV calls the loader for the load-and-go operation selected by the most recent LOADER control card encountered in this job; or if no LOADER control card had been used for this job, the loader is selected by default.

After the CP overlay has been loaded, LOADREQ generates reply information for the user in the same format as the LOADER request.

## 3.7 FILE PROCESSING

The FNT entry for a file in the input or output queue is slightly different from other FNT entries. If a file is in either queue the FNT will contain a priority field. The priority given to input queue files depends on the priority specified on the job card, the length of time the file has been in the queue, and possibly the other parameters on the job card. Input queue priorities are used to determine the order in which jobs are brought to control points.

The priority given to a file in the output queue is the priority of the job at the time the file was put in the queue. This priority is incremented with time. JANUS and other output routines normally process files in order of priority. The priority of files in either queue can be changed by an operator type-in.

In addition to the priority field, the input queue FNT entry contains other parameters from the job card including central memory and ECS field length requirements and the time limit. Output queue FNT entries contain an additional field which specifies how much data is on the file. This number is given as a multiple of blocks of $100_8$ central memory words.

### 3.7.1 INPUT QUEUE FILES

Files are placed into the input queue by JANUS, as it reads in jobs from the card reader, by other system jobs such as LOAD, LOADX and RESQ, or by EXPORT/IMPORT or RESPOND. An input queue file contains the entire job which consists of a record of control cards followed by any number of data records. Files in the input queue must be type input, assigned to control point zero, and have non-zero priority. The file name will be the job name. Input queue files are always put on allocatable devices.

### 3.7.2 OUTPUT QUEUE FILES

Files can be put in the output queue by the user when CLOSE, UNLOAD is performed on a file or by the system at job termination. Files in the output queue must be type local or output, assigned to control point zero, and have non-zero disposition and non-zero priority. The file name is the name of the job which created the file. Files are always rewound before they are put into the output queue.

A file will not be put in the output queue unless it is on an allocatable device since JANUS and other output routines cannot handle non-allocatable files. A random file can be put in the output queue; however, JANUS handles random files sequentially. The categories of local files which may be attached to the control point of a running job are described below.

### 3.7.3 INPUT FILE

When a job is brought to a control point, the input queue FNT entry is changed so that the file is assigned to the control point, has the name INPUT, and is type local. The input file is still the same file as the input queue file. It contains the control card and data records and it resides on an allocatable device. Since the input file contains the control cards for the job, it is illegal to CLOSE the input file; any attempt to do so will be ignored. SCOPE maintains special pointers to the control card record on the input file. Normal operations such as READ or REWIND on the input file do not affect these pointers. The user can, however, call the system macro, CONTRLC, which can change the pointers to the control card record, and can affect the order in which control cards will be processed.

The user should never write on the file INPUT or detach it from his control point, although it is possible to do both. In the first case, the user could destroy the control card record; in the second, the entire file could be destroyed or lost.

### 3.7.4 OUTPUT FILE

The output file is a local file at the user's control point with the name OUTPUT. When an error condition occurs, the system will dump to the output file the exchange package and $100_8$ central memory locations before and $100_8$ after the address to which the P register pointed at the time of error. At job termination, the dayfile is copied to the end of the output file (unless the job has been terminated by KILL, in which case no output is produced).

If no output file exists, the system will create one. The output file will always be on an allocatable device since it is illegal to request OUTPUT with either the control card or macro. Any attempt to do so will terminate the job.

When it is time to dispose of the OUTPUT file, it is treated as a special name file. Unless the user specified a disposition code, the file will be given print disposition and be placed in the output queue. If an output file is disposed of by a CLOSE,UNLOAD it is put into the output queue with type local. At job termination the output file is put into the output queue with type output. Therefore only one file for each job can have type output and that file will contain the dayfile for the job.

### 3.7.5 SPECIAL NAME FILES

Several other file names are treated as special cases. The user can specify disposition for these files and that disposition will be honored; however, if a local file with one of the special names has a zero disposition code, the system will automatically set the disposition code to an appropriate value before disposing of the file.

Names treated as special cases are the following:

OUTPUT
PUNCH
PUNCHB
FILMPR
FILMPL
HARDPR
HARDPL
PLOT

### 3.7.6 NON-ZERO DISPOSITION FILES

All local files which have disposition set by the user and all special name files which have disposition set by the system will be rewound and placed in the output queue at disposal time if they reside on allocatable devices.

### 3.7.7 NON-ALLOCATABLE FILES

If a local file resides on a non-allocatable device it will be dropped (its FNT zeroed) at disposal time regardless of its name or disposition code. It will not be put into the output queue because JANUS and other output routines cannot process non-allocatable files.

### 3.7.8 COMMON FILES

A user can change the type of any local file to common if it is attached to his control point and is neither a permanent file nor a private disk pack file.

He can also attach an existing common file to his control point. If a common file is not being used, it will be attached to control point zero. A common file may be on an allocatable or non-allocatable device, it may have a special name, and it may have non-zero disposition.

When it is time to dispose of a common file, the file is assigned to control point zero and made type common regardless of its name, disposition code, or the type of device on which it resides; however, the disposition code is saved and when the file is released and made local, it is treated as a special name file or a non-zero disposition code file if appropriate.

### 3.7.9 PERMANENT FILES

A permanent file can exist across deadstarts. The file and sufficient information to access the file is maintained on a mass storage device. An FNT entry is made for a permanent file only when the file is attached to a control point; it will be type local and will be marked as a permanent file. A permanent file can never be assigned to control point zero.

At job termination, a permanent file is disposed of by deleting its FNT entry and releasing its RBT chain. The file will still exist on the mass storage device. A CLOSE,UNLOAD on a permanent file also saves the file but deletes the FNT. A special name or a non-zero disposition for a permanent file is meaningless and is ignored.

A user can completely remove a permanent file from the system with the PURGE command. The mass storage space occupied by the file will be released.

A permanent file cannot be used as a common file; any attempt to do so will be ignored, and a non-fatal error message will be issued.

## 3.7.10 PRIVATE DISK PACK FILES

A disk pack unit may be designated as public or private. A public unit is treated the same as any other mass storage device; it may contain one or more files of various types assigned to different control points.

A private disk pack unit is treated somewhat like a magnetic tape. The unit can be assigned to a control point with the RPACK control card, and it may contain one or more files associated with the control point. A private disk pack unit cannot be attached to control point zero. The REQUEST control card must be used to create a file on a unit attached to a user's control point. The name on the REQUEST card cannot be one of the special names; if it is, the job is terminated with a control card error.

When a private disk pack unit is attached to a control point an FNT entry is made for each file on the unit as well as one FNT entry for the unit itself. The latter is marked so a file on the pack can have the same name as the pack. The files will be type local and will be assigned to the control point.

A private disk pack file cannot be made type common or put into the output queue. If a user gives a disposition code to a private disk pack file, it will be ignored. At job termination information about each private disk pack file (such as name and physical disk locations used by the file) is written to the disk pack; and the file itself remains on the pack; but the FNT entry for each file is deleted from the FNT. The FNT entry for the pack is also deleted.

Other entries in system tables refering to the private pack files are deleted, and the equipment is detached from the control point.

To eliminate a file from the private disk pack and from the system, the REMOVE control card is used.

## 3.7.11 RANDOM FILES

If a random file is to be saved, the file index must be written as the last logical record on the file. A user may write the index himself or he may call the system macro CLOSE or CLOSE,UNLOAD to write the index for him. CLOSE automatically writes out an index for a random file if the file security is open, write or open, alter. A permanent file must also have EXTEND permission.

If the user neglects to write out the index on a random file or an error terminates the job before he has a chance to do so, the system will write out the index if the file satisfies all the following conditions:

File must be random and open for write or for alter.

The FET indicated by the pointer in the FNT must appear valid. (The random bit must be set, the FNT pointer must be correct, the index length must be non-zero, and the address of the last word in the index must be less than the field length.) This is to prevent the system from writing out the index if the index and the FET which points to the index have been destroyed.

The file must be permanent with EXTEND permission, a common file, or a private disk pack file.

## 3.8 FILE DISPOSITION

When jobs terminate, the method of file disposal varies according to the circumstances that cause termination.

### 3.8.1 NORMAL TERMINATION

1. All local files with non-zero disposition code (including output and other special name files) which reside on allocatable devices are rewound and put into the output queue by changing the file name to the job name, and assigning the file to control point zero.

2. All common files are assigned to control point zero with type common. They are not rewound. If a common file is on a non-allocatable device, the equipment is dropped (detached from the control point) and set to logically off to prevent its assignment to another file.

3. All FNT entries for permanent files at the control point are deleted.

4. All FNT entries for private disk pack files and the FNT entry for the disk pack unit are deleted from the FNT. Information identifying each file is written on the disk pack and the equipment is detached from the control point.

5. The index is written out prior to disposing of a file if it is random, open for write or alter, has a valid FET, and if the file is one of the following:

   permanent file with EXTEND permission

   private disk pack file

   common file

6. All other files, including the input file, local files with zero disposition, and local non-zero disposition files on non-allocatable devices are dropped. This involves deleting the entry in the FNT for the file, releasing any reserved mass storage space on an allocatable device, or dropping the equipment if it is non-allocatable. A file on a non-allocatable device remains at its current physical position.

### 3.8.2 KILL

When the operator types in n.KILL at the control point, the job terminates and all local files associated with the job, including the output file, are dropped regardless of name or disposition. Permanent files, private disk pack files, and common files are treated as for normal termination. An index is written if appropriate. No files are put into the output queue.

### 3.8.3 RERUN

When the operator types in n.RERUN at the control point, the job is terminated and returned to the input queue, so that it can be run later. Files associated with the job are handled as follows:

> The input file is returned to the input queue. Its file name will be the job name and it will be type input at control point zero.
>
> The output file is dropped and a new output file is created. The job dayfile is copied to the new output file which will not be rewound, but made local at control point zero; the file name will be the job name. It is called a pre-output file, and becomes an output file when the job is run again. The output file for the rerun job will contain the dayfile from the first partial run of the job and the output and dayfile from the second complete run of the job.
>
> Any files attached to the job's control point that were common at the start of the job are returned to control point zero as common files even if they had already been released by the job. If they are random files and meet the other requirements, the index will be written out.
>
> Common files created by the job are dropped.
>
> Permanent files and private disk pack files are treated as for normal termination. If they meet the requirements, an index will be written out.
>
> All other files, regardless of name or disposition, are dropped.

### 3.8.4 NO RERUN

In some cases, a job might perform a function which would make it impossible to restore conditions to their initial state (before the job was run). For example, if a job writes on an existing common file, that information cannot be erased. Trying to rerun such a job could have bad effects, and therefore RERUN should be prevented on such a job. The no-rerun flag can be set in the control point area. If set, the operator type-in n.RERUN will be rejected. The no-rerun flag will be set in the following cases:

1. The user attaches an existing common file to his control point (because it cannot be determined if information has been written on the file).

2. The user detaches a common file from his control point with CLOSE,UNLOAD or COMMON macros, or the RETURN control card.
3. This is a RESPOND job.
4. The job has attached a permanent file to its control point with EXTEND, MODIFY, or CONTROL permission, or has cataloged a file.

### 3.8.5 DISPOSAL PRIOR TO TERMINATION

Prior to job termination a user can dispose of any file attached to his control point by calling the system macro CLOSE,UNLOAD, or by using the RETURN control card. A common file can be disposed of by calling the system macro COMMON. After the user disposes of a file, he may create and reference a new file with the same name.

CLOSE,UNLOAD

The system macro CLOSE can include the parameter UNLOAD to specify disposal of a file at the user's control point. Files disposed of in this way are handled as for normal job termination with the following exceptions:

1. CLOSE cannot be used for the INPUT file.
2. A sequential file is rewound.
3. A file on magnetic tape is physically unloaded.
4. A file on a private disk pack is locked.
5. CLOSE,UNLOAD (or any other CLOSE) will write out the index to any non-permanent random file which is open for write or alter with EXTEND permission.

RETURN

The RETURN control card can be used to dispose of a file at the user's control point. RETURN will perform a CLOSE,UNLOAD on the file.

COMMON

The system macro COMMON can be used to dispose of a common file at the user's control point. Disposal is the same as in normal termination. If the file referenced in the call is not type common, no action is taken, but an error indication is returned to the user.

### 3.8.6 BUFFER EMPTYING

The system will empty the buffers of certain files before disposing of them. A write end-of-record will be issued to write any information still in a user's buffer onto the associated file. Buffers are emptied when a job terminates and the error flag is not KILL or RERUN; or when an error (other than KILL or RERUN) occurs which does not cause termination because the job contains an EXIT card followed by more control cards.

In the former case, the system will empty the buffers as described above. In the latter, the system will empty the buffers, process the error condition, and continue processing cards after the EXIT card. A CLOSE,UNLOAD will not empty the buffers.

Files with non-zero disposition (including output and other special names) are emptied if they meet all the following conditions:

> The last code and status in the file's FNT indicates that the last operation on the file was an OPEN,WRITE (with or without rewind) or any type of write (WRITE, WPHR, WRITER, WRITEF).
>
> The FET pointer in the FNT points to an FET with the same name as the file name.
>
> The buffer is not empty.

For common files, buffers are emptied regardless of disposition.

The FET pointer in the FNT points to the address of the FET used in the last I/O operation. If a user has two FET's for a file, it may not be possible to determine which buffer is emptied.

### 3.8.7 EXPORT/IMPORT AND RESPOND

Files associated with EXPORT/IMPORT and RESPOND jobs are handled as described below:

EXPORT/IMPORT

When files attached to an EXPORT/IMPORT job are disposed of by the user or the system, they are handled the same as for a normal job, except that $4000_8$ is added to the disposition codes of all files placed in the output queue. Files in the input queue with disposition codes of $4xxx_8$ are ignored by JANUS, as they are picked up by the EXPORT/IMPORT package. In some cases, the EXPORT/IMPORT package may reset the disposition so that JANUS will process the output.

## RESPOND

If the user disposes of a file attached to a RESPOND job, the file is treated normally. All others, except for the output file, are disposed of by the RESPOND package at end of job. The output file will be put into the output queue with $2000_8$ added to its disposition. JANUS will ignore the output file, as it will be picked up by the RESPOND package.

# LOADER OPERATION 4

The relocatable loader provides high-speed transfer from input and storage devices to central memory. Initially, the loader is called by SCOPE control cards; later it may be called from an object program.

The two loaders included in this system may be selected with a control card. If the user does not select a loader, the system selects it by default. Loaders are specified by a name of 1-7 alphanumeric characters; PPLOADR and CPLOADR are currently available. These loaders are externally compatible except for file positioning at the end of selective load operations (described under LOADER request, section 3.6.4) and the lack of recognition of code assembled at absolute origins by the CP loader (section 4.5.1). Both loaders function as follows:

> 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

Loading for PPLOADR proceeds in the following general manner:

1. The loader may be called by the user program or as a result of control card requests.
2. The initial control card results in loading of the PRU 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 as described in appendix D. Duplicate entry points produce a non-fatal diagnostic.

7. At completion of load and at the user's discretion, all unsatisfied references are filled with references to entry points in relocatable routines 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.

CPLOADR operates primarily in the central processor within the user's field length. Within CPLOADR the following functions are reserved for the central processor:

- Text relocation
- Loader table generation
- Loader table processing
    - Linking
    - Replication
    - Labeled and blank common assignments
- Library loading
- Issuing I/O requests to PP routines
- Segment handling operations (including delinking)
- Overlay generation
- Overlay loading
- Issuing maps and error diagnostics

Components of the central processor portion of the loader follow:

| | |
|---|---|
| LOADERQ | Main module, for loading overlays, relocatable programs and library subroutines in normal load operations. |
| LOADERS | Used when segment loading is in operation. |
| LOADERV | For overlay generation. |
| LOADERE | For fatal errors. |
| MAPOUT | For editing full maps. |

Functions of peripheral processor components of the loader follow:

| | |
|---|---|
| LOQ | Load main central processor module, LOADERQ. |
| LDQ | Called by LOADERQ for:<br>Reading input files from mass storage into central memory buffer<br>Reading library directory into user's field length<br>Reading relocatable subroutines from library into user's central memory buffer<br>Reading central processor time spent in loading for map output<br>Checking validity of loader directives (SEGMENT, OVERLAY, etc.) |

Other peripheral processor routines called are:

| | |
|---|---|
| CIO | Called for:<br>Reading input files from tape into central memory buffer<br>Reading input files from any device into central memory for selective load from a file |
| LDR | Called as a result of a user request to load an overlay from input file (device independent) or system library. |

The control cards described in section 2 and the user request formats described in section 3 have the same significance to CPLOADR as they do to PPLOADR.

## 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 subprograms to be included in a segment with loader 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 but not between segments. Therefore, delinking is not required. Blank common references are established between programs within a segment and also between segments. 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† | | | |
| 6 | 1 | 0 | 1 | | | |
| 7 | 2 | 0 | 1 | 2† | | |
| 8 | 5 | 0 | 1 | 2 | 5† | |
| 9 | 7 | 0 | 1 | 2 | 5 | 7† |

†newly loaded

## 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, $L_1$, $L_2$, 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 the generation of overlays to the new lfn. The source file for overlay generation may not be used as the lfn on any overlay cards. Loader writes the first overlays to the output file before all overlays on the source file have been processed. $L_1$ is the primary level number in octal; $L_2$ is the secondary level number in octal. $L_1$, $L_2$ 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. Writing to the overlay file takes place when a directive is encountered which specifies an overlay level which would overlay a level currently residing in memory. Writing also takes place when the last overlay has been created.

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. Similarly, common blocks defined in a lower level overlay can be referenced from a higher level overlay. Data can be preloaded into a labeled common block if the overlay which defined the common block has not been written to the overlay file.

## 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.

| Contents | 5000 | $L_1$ | $L_2$ | fwa | ea | |
|---|---|---|---|---|---|---|
| Bits | 59 | 47 | 41 | 33 | 17 | 0 |

$L_1$ primary overlay level

$L_2$ 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 the loader during initial load.

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

sname is the name of the section and $pn_i$ 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 names must be a maximum of 7 alphanumeric characters.

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 the SEGZERO card and all SECTION and SEGMENT cards before 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. All names must be a maximum of 7 alphanumeric characters.

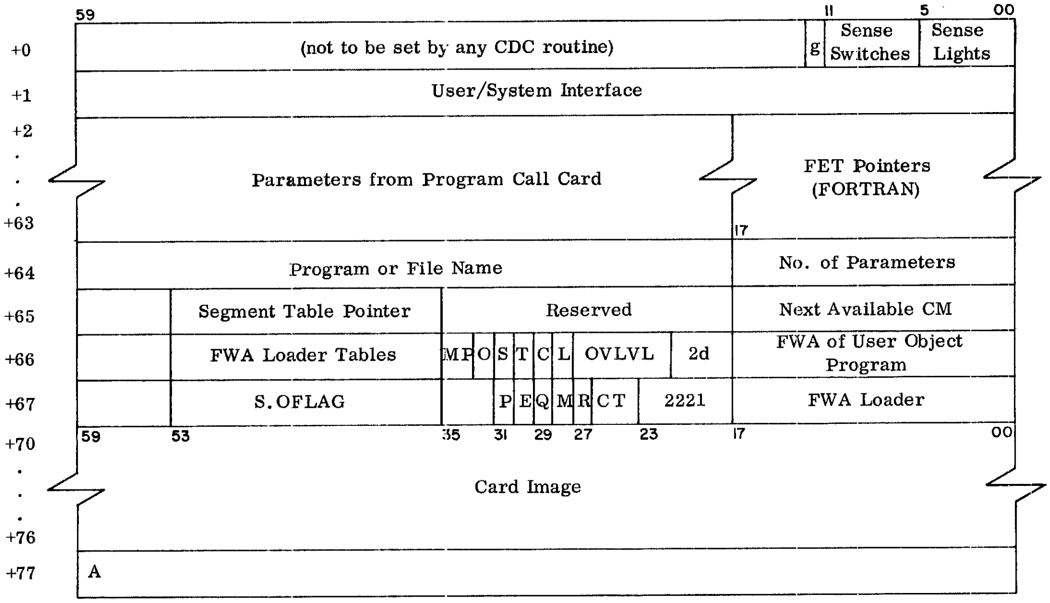
SEGMENT (sn, $pn_1$, $pn_2$, . . . , $pn_n$)

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:



In the above diagram RA+64 through RA+67 are reserved for the SCOPE system-loader interface and bytes are assigned as follows:

| | |
|---|---|
| 2d | Indicator for loader directive |
| OVLVL | Level of incoming overlay |
| CT | Control card type NOGO, LOAD, EXECUTE, PROGRAM |
| R | RSS mode indicator |
| M | No map flag for library load |
| Q | Request flag - communication between LDR and LOADER |
| E | End of load flag |
| S.OFLAG | Segment - FWA of tables for lowest segment in user's job area<br>Overlay - FWA blank common |
| g | GO/PAUSE flag; 0 =GO, 1 = Pause, wait for GO |
| A | ASA flag (used by RUN2.3); 0 - not ASA; -0 = ASA<br>FORTRAN does not use 70-76(7) in the card image manner |
| P | Partial map flag |
| S | SNAP |
| T | TRACE |

| | |
|---|---|
| C | Change dump |
| L | Labeled dump |
| O | REDUCE (OBESE) flag |
| MP | Map flag: 001 = Full map, 010 = No map, 100 = Partial map |
| Card image | Upon initial entry from a named routine call or an EXECUTE card, these locations will contain the card image (in display code) of the card which called for execution. |

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

Absolute programs on the system library which are loaded by program call cards are loaded at location RA+$100_8$ regardless of the declared origin of the program. Relocatable programs are loaded originally at RA+$100_8$. The origin of a program made absolute by COMPASS must be location 101 as COMPASS places a control word in front of the first location.

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. (An additional $5000_8$ is usually sufficient .) 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 this field length is long enough to accommodate the loader, its tables, and blank common if he is concerned with preservation of data.

The CP loader uses memory beyond the last loaded address of the user's programs for mapping purposes. Code assembled by COMPASS at absolute origins is not included in the range of that last address. Therefore such code may be destroyed by the loader. The PP loader does not have this restriction.

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.

### 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.

## 4.7 ERRORS IN ASSEMBLY/ COMPILATION

Errors encountered in assembly and compilation do not automatically cause the system to terminate a job. Rather, the job is terminated when the user attempts to load the assembled program. The loader recognizes the directive ERRORS, which is produced in the first record in the first character position on the binary output file. ERRORS IN ASSEMBLY. is produced by COMPASS, ERRORS IN RUN COMP. is produced by RUN. To avoid indiscriminate dumping of system routines, such as LOADER, COMPASS, etc., EXIT processing will not occur if the ERRORS directive caused termination; that is, control cards following an EXIT card will not be processed. When the EXIT card contains an S parameter:

```
EXIT(S)
```

EXIT processing will occur in all cases.

# SYSTEM LIBRARY PREPARATION AND MAINTENANCE 5

The SCOPE system library includes system routines, library maintenance and utility routines, members of the SCOPE product set such as COMPASS, FORTRAN, COBOL, etc. 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 mass 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 CMR.
- Files containing one or more logical records in a form suitable for insertion into SYSTEM or a system library file.

The released system library consists of 18+n records followed by an end-of-file, where n is the number of programs and overlays in the library. Any installation, however, may expand the 18 records to 25 since the SCOPE 3.2 system permits up to eight different CMR records on the library to correspond to eight different equipment configurations at an installation. The released SCOPE 3.2 system has only one CMR record on it. If the installation places additional copies of CMR on its system tape, the number of records preceding the library programs will increase by one record for each additional CMR. The programmer should modify the parameters in his EDITLIB control cards if necessary to match his installation tape.

| Name/Record No. | | Description |
|---|---|---|
| CEA | 1 | PP0 save program. If the save switch on the deadstart panel (word 7, bit 2) is down (set to 0), PP0 is copied to central memory (starting at IP.SVADR) allowing the contents of PP0 to be dumped by the deadstart dump program. |
| CED | 2 | Deadstart control program |
| D | 3 | Deadstart dump program |
| CMR | 4 | Central memory resident (up to eight copies allowed) |
| EST | 5 | Deadstart equipment reconfiguration program |

| Name/Record No. | | Description |
|---|---|---|
| IRP | 6 | Deadstart I/O control program |
| 5CP | 7 | Deadstart 6603-I driver |
| 5CQ | 8 | Deadstart 6638 driver |
| 5CR | 9 | Deadstart 865 driver |
| 5CS | 10 | Deadstart 854 driver |
| 5CT | 11 | Deadstart 6603-II driver |
| P | 12 | Preaddress 6603-II program |
| STL | 13 | Deadstart system initiation program |
| IRCP | 14 | Deadstart main program (performs device labeling, permanent file processing, preloading, loading, and recovery) |
| MTR | 15 | System monitor program (resides in PP0 during system operation) |
| DSD | 16 | Display control program (resides in PP9 during system operation) |
| | 17 | Entry point table copied into directory area of CMR as its first section. Each entry contains name of an entry point in a central processor program in system library, and a number which locates it in the program name table. |
| | 18 | Program name table copied into directory area of CMR as its second section. Each entry contains name of a PP program, CP program, CP overlay, or ERT, and its length, type, edition number, and residence. |
| | 19 and following | Programs and overlays; each has an entry in the program name table. As these records are read, the loader inserts disk addresses into corresponding program name table entries. |

## 5.1 EDITLIB CALL CARDS

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.

EDITLIB uses the following file names. These names should be unique, since it is impossible to protect these files under the current system; therefore, the user must NOT CREATE ANY FILES with the following names:

SSSSSSS
SSSSSST
SSSSSSU
SSSSSSV
SSSSSSW
SSSSSSX
SSSSSSY
SSSSSSZ
SYSTEM

## 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. In this case, the running system, SYSTEM, will be modified.

#### 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.

DELETE ($p_1$-$p_2$)

This function deletes from SYSTEM directory records $p_1$ through $p_2$, as listed in the program name table. If $p_2$ does not appear after $p_1$ in the program name table, the job is terminated with the message:

((DEL)) EXHAUSTS PNT BEFORE SATISFACTION

If any record ($p_1$-$p_2$) is 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.

Before any change is made to SYSTEM, a message appears on the B display at the control point.

EDITLIB WARNING - GO or DROP

The operator must type n.GO to allow the MOVE or DELETE to proceed. If the operator prefers not to risk the change, he can DROP the job, and SYSTEM will remain unaltered.

LIST

LIST (s)

File s is rewound and searched until DSD is found, and the next two 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 or OVERLAY)

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 The records from CEA through DSD 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.

## 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 $\neq$ 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.

DELETE ($p_1$-$p_2$)

This function deletes from the system library under construction, records $p_1$ through $p_2$ inclusive, as listed in the program name table of the model directory under construction. If $p_2$ does not appear after $p_1$ in this table, the job is terminated with the message:

((DEL)) EXHAUSTS PNT BEFORE SATISFACTION

LENGTH

LENGTH (p)

When p is a digit from 4 to 9, EDITLIB 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 at least $30000_8$ words.

The directory models to be accommodated are those of the running system as EDITLIB finds it initially, plus any modifications EDITLIB may make between obeying a READY (SYSTEM) card and the succeeding COMPLETE card. Directory length = 2+2p+e+r. p = number of programs and overlays in library; e = number of entry points in CP programs in library; r = number of words in the bodies of all CM-resident programs. Normally no more than $30000_8$ words are required; however, a larger field length can be established by using LENGTH (p) as the first function card read by EDITLIB.

COMPLETE

COMPLETE

This function causes the file id 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. Before the replacement is made, a message appears at the control point on the B display.

EDITLIB WARNING - GO or DROP

The operator must type n.GO to permit replacement, or n.DROP to prevent it by terminating the job.

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

### 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.                    Saves current directory and begin EDITLIB run.
7/8/9
REWIND(SYSTEM)
READY(NEWSYS)               Prepares NEWSYS to receive records.
TRANSFER(SYSTEM,16)         Copies first 16 records of system.
ADD(*,SYSTEM)               Copies to a scratch file all programs found in
                            directory.
COMPLETE.                   Copies 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 bc replaced in NEWSYS by another program called 2TS, which is at hand as a deck of cards; 2TS is to have mass storage residence:

```
job card
REQUEST,NEWSYS.          Operator will assign a tape.
EDITLIB.                 Saves current directory and begin EDITLIB run.
7/8/9
REWIND(SYSTEM)
READY(NEWSYS)
TRANSFER(SYSTEM,16)      Transfers 16 SYSTEM records.
ADD(*,SYSTEM)            Copies all programs in the directory to a scratch
                         file.
DELETE(2TS)              Removes program called 2TS from new directory.
ADD(2TS,INPUT,DS)        Adds 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,14)      Transfers first 14 records.
TRANSFER(INPUT,MTR)      Transfers new MTR to MYTAPE.
SKIPF(SYSTEM,1)          Skips over the old MTR.
TRANSFER(SYSTEM,1)       Transfers DSD to MYTAPE.
ADD(*,SYSTEM)
COMPLETE.
7/8/9
New MTR binary deck
6/7/8/9
```

- Simulate the deadstart loading of a system tape called MAYBE, assuming its first 16 records to be the same as on file SYSTEM:

```
job card

REQUEST,MAYBE.

EDITLIB.

7/8/9

SKIPF(MAYBE,18)          Skips over the 16 system records and the 2
                         directory records.

READY(SYSTEM,*)

ADD(*,MAYBE)

COMPLETE.

6/7/8/9
```

- Any succeeding 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
```

# UPDATE 6

UPDATE is a maintenance program that creates, corrects, and manipulates program library files. UPDATE can be used with SCOPE 3.0 and newer versions for temporary testing or generating a new program library. A field length of $40000_8$ should suffice for most UPDATE runs.

UPDATE data may be source cards for a compiler or assembler run, data cards, or any other symbolic information convenient to the user. Columns 73-80, however, are destroyed by UPDATE card identifiers. The UPDATE system call (control card) directs the program to the specific files and modifies the operation of the program.

## 6.1 PROGRAM LIBRARY INFORMATION

The program library, which UPDATE creates and/or modifies, contains the symbolic data for programs being maintained on the system tape. These programs are arbitrarily divided into decks by insertion of *DECK and *COMDECK cards into the text stream. The program library file consists of a directory, a deck list, and text stream.

The first word of the program library file is composed of two counters: the count of the identifiers and the count of the deck names.

|  | 59 | 35 | 17 00 |
|---|---|---|---|
| Word 1 | zero | Count of identifiers | Count of deck names |

### Directory

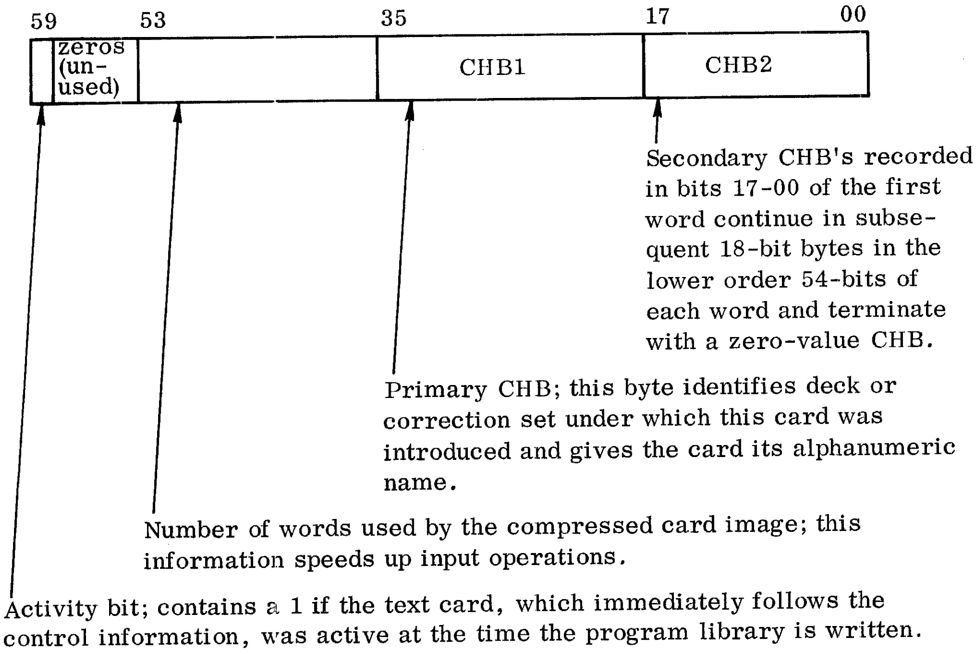
The directory lists all identifiers, left justified with zero fill, in the upper 42 bits of succeeding words. The lower 18 bits of each entry contain miscellaneous data.

### Deck List

The deck list contains the names of all decks in the file including those no longer current. The deck list entries have the same format as the directory entries.

Text Stream

The text stream contains card images and control information known as correction history bytes (CHB's). At least one word of control information precedes each card image:



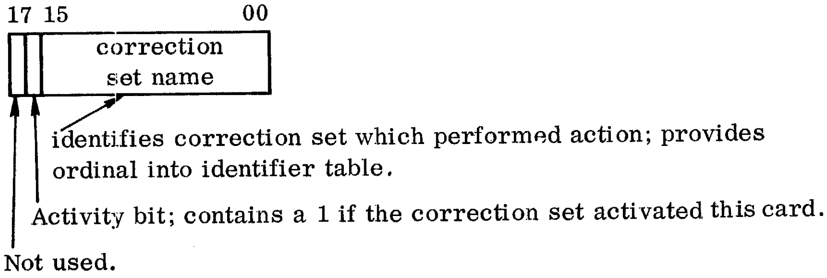
Secondary CHB's recorded in bits 17-00 of the first word continue in subsequent 18-bit bytes in the lower order 54-bits of each word and terminate with a zero-value CHB.

Primary CHB; this byte identifies deck or correction set under which this card was introduced and gives the card its alphanumeric name.

Number of words used by the compressed card image; this information speeds up input operations.

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.

CHB format:



identifies correction set which performed action; provides ordinal into identifier table.

Activity bit; contains a 1 if the correction set activated this card.

Not used.

Compressed Card

The compressed card image begins in the word subsequent to that 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

A card identification assigned by UPDATE is permanent and cannot be changed. Since the validity of sequence numbers has not been ascertained for text information following any particular card image, if an irrecoverable parity error occurs, the UPDATE run will terminate. Card identifiers are in the format:

beta.seqnum

beta is the alphanumeric identifier; maximum of seven characters. It is defined on the *DECK and *COMDECK card for program library creation or on the *IDENT cards for correction runs.

seqnum is the sequence number obtained by counting the cards with the same identification.

The two elements of an identification are separated by a period.

An identifier and sequence number are required. For example, SC12.42 is the 42nd card (sequence number) with the identifier SC12. The identifier is composed of a reference symbol (SC) combined with a reference number (12).

## 6.2 UPDATE PARAMETERS

File parameters and special options on the UPDATE card may appear in any order. Each parameter must be specified by a string of characters of the form: ident=value. ident may be any character string beginning with the letters P, N, I, L, C, or S; it cannot contain a comma, period, or parenthesis. 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.

The idents P, I, L, and C and their default file names are assumed whether or not they appear. The N and S files are generated only if the parameter is specified.

The meaning and default values of the file parameters and special options are indicated below.

| Parameter (ident) | Option Values | Default Value |
|---|---|---|
| P (OLDPL) | File containing old program | OLDPL is assumed. |
| N (NEWPL) | File containing new program | If N is not specified, 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 card images to be assembled are written | COMPILE is assumed. (COMPILE output depends upon mode of updating. If C = 0, no compile file is written.) |
| S (SOURCE) | File on which a copy of source deck is written; contains a copy of cards necessary to regenerate existing library, minus inactive cards. | If S option is not specified, no source deck file is written. |
| Special Options | | |
| F | Write all decks on compile file | If F option is not specified, only those decks modified are written. |
| Q | Speeds updating process by specifying information on routines which will be updated | This option is effective only if correction runs are being performed. |
| R | Suppress automatic rewinding of P, N, S and C files | S, C, P, and N are rewound before and after processing. |

[†] The L option does not list the card images of a program deck and their sequence identifiers in a creation run or when inserted via ADDFILE. To obtain such a listing, copy the COMPILE file onto OUTPUT.

## 6.3 CONTROL AND DATA CARDS

Control and data cards are punched on an 80-column card through column 72; columns 73-80 are used by the Update Program for sequencing. 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 succeeding column with any number of intervening blanks; but 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 MANIPULATING FILES

When text and other control cards are to be read from files other than the main input file, file manipulation cards are used to direct update operations. They may appear at any point in an input record. 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

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

```
*REWIND fname
```

| | |
|---|---|
| fname | Name of file from which information is to be read |

*SKIP

The SKIP card causes the named file 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.

```
*SKIP fname, rent
```

| | |
|---|---|
| fname | Name of file from which information is to be read |
| rent | Number of records to skip |

*READ

UPDATE reads input data from the named 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.

```
*READ fname
```

| | |
|---|---|
| fname | Name of file from which information is to be read |

*LABEL

This card is meaningful only in a creation operation. *LABEL precedes the first *DECK or *COMDECK card, and it specifies the 20-character label name to 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.

```
1         11
*LABEL label name
```

| | |
|---|---|
| label name | 20-character label name of the new program library |

### 6.3.2 CREATING A NEW PROGRAM LIBRARY

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

*DECK

This card indicates the beginning of a new deck; dname 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 control cards other than *DECK and *COMDECK may also be introduced and numbered.

```
*DECK dname
```

dname Name of deck introduced; must differ from name of any previously introduced deck or correction set.

### *COMDECK

This card introduces a common deck. *COMDECK is subject to the same rules as the *DECK card concerning naming and numbering; however, it is output in a different manner (section 6.3.4).

```
*COMDECK dname
```

dname Name of common deck introduced; must differ from name of any previously introduced deck or correction set.

### *END

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

```
*END
```

### 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, as indicated in the following example:

```
*DECK         FIRST
              IDENT      FIRST
              .....
              END
              IDENT      SECOND
              .....
              END
*END
*COMDECK      FDATA
              BLOCK DATA
              COMMON/J3/A(10)
              DATA A/3*0.,7*1.0/
              END
*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, also 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.

The following control cards may appear as input during an UPDATE creation run:

```
*DECK      *COMDECK    *END
*REWIND    *SKIP       *READ
*LABEL     */          *WEOR
*CALL                  *CWEOR
```

All other UPDATE control cards produce a diagnostic.

## 6.3.3 CORRECTING AND UPDATING

The following control cards are used for correcting and updating a program library.

### *IDENT

The *IDENT card introduces a correction set. All corrective operations except *PURGE and *ADDFILE must occur after *IDENT. All cards in the 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 original deck names or 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.

*IDENT idnam

idnam Name of correction set; must differ from the name of any previously introduced correction set or deck.

### *PURGE

The *PURGE card may appear in the place of an *IDENT card. Any reference to the correction set idnam is completely purged by changing the name to . . . . , which allows it to be reintroduced in a later correction set. When two identifiers are separated by period, these identifiers are purged as well as all identifiers that occur between them on the identifier list.

All cards introduced under the specified identification are physically removed from the new program library; all corrections to the named set are removed by squeezing out correction history bytes of that identification. Since permanent sequencing information is affected, the *PURGE card must be used with care.

*PURGE idnam$_1$,idnam$_2$,. . . ,idnam$_n$

*PURGE idnam$_1$.idnam$_2$

idnam$_i$ Name of correction set to be purged (must be a known identifier)

*PURGE simulates the effect of *YANK (described later); although 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 and their idname. 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       (978 cards are assumed)
*PURGE      CIO
```

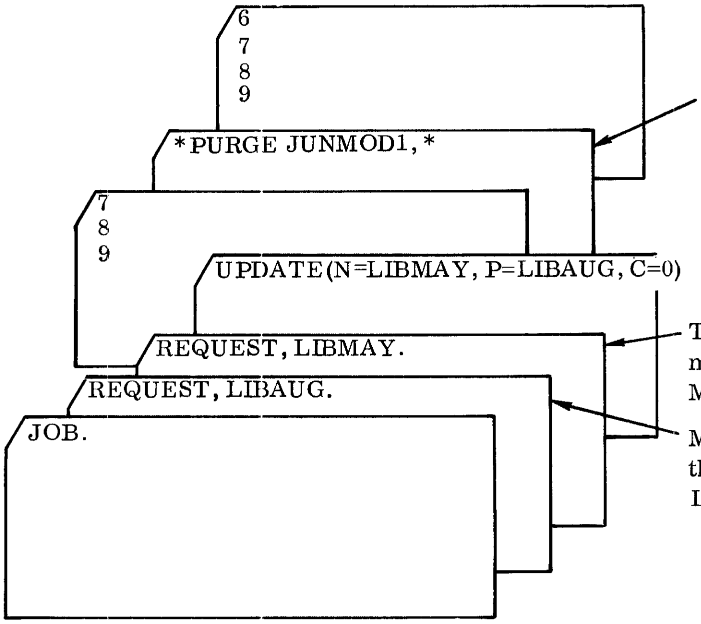
The first statement deactivates all inserts made to CIO and the second removes all CIO cards. An alternate form of the *PURGE statement is used to purge all correction sets introduced on or after the introduction of one identified by idnam.

```
*PURGE 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. Care must be taken if an idnam,*, has been introduced. If * is to be purged, it must be the first name on a PURGE card or UPDATE will assume that the * refers to an alternate form.

Example 1:

A program library LIB has been periodically modified for a number of months; at some point in time, it becomes desirable to step back permanently to a previous level of LIB. The following deck sequence illustrates this use of UPDATE:



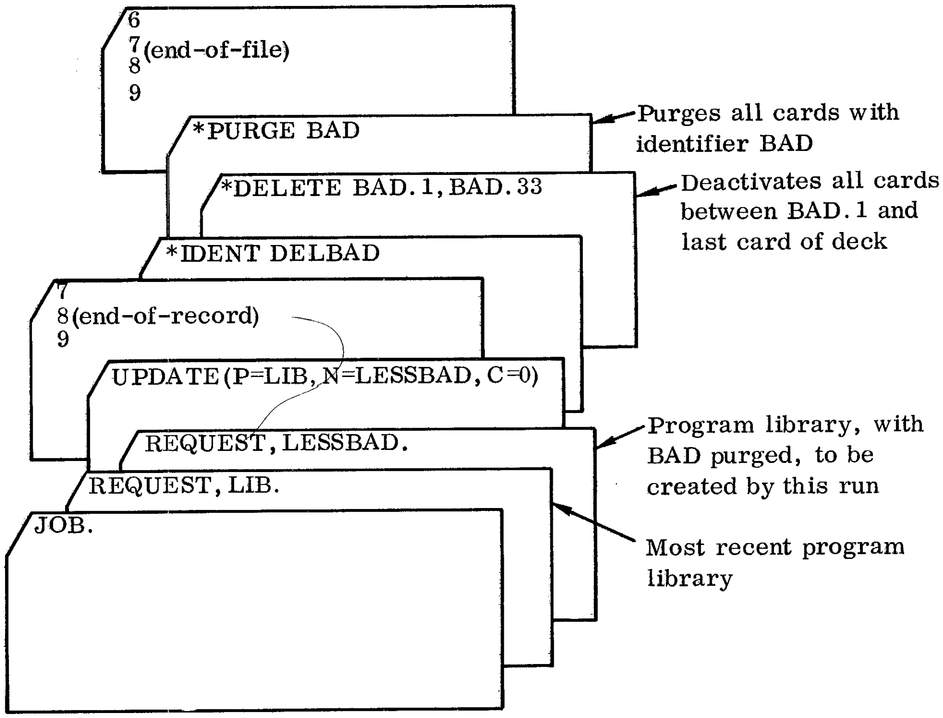
where all modifications made after May follow JUNMOD1 in the identifier list

The program library modified only through May, to be created

Most recent version of the program library LIB

Example 2:

Assume that a specific deck, BAD, on program library LIB, is no longer of any use and is to be removed permanently from the program library, The following deck sequence illustrates such permanent removal:



```
6
7
8(end-of-file)
9
*PURGE BAD
*DELETE BAD.1,BAD.33
*IDENT DELBAD
7
8(end-of-record)
9
UPDATE(P=LIB,N=LESSBAD,C=0)
REQUEST,LESSBAD.
REQUEST,LIB.
JOB.
```

Purges all cards with identifier BAD

Deactivates all cards between BAD.1 and last card of deck

Program library, with BAD purged, to be created by this run

Most recent program library

### *DELETE

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 specifically referenced cards must exist. Text cards may be inserted after the last card deleted.

```
*DELETE a.n
```

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

| | |
|---|---|
| a,b | Alphanumeric identifier (a deck, common deck or correction set name) |
| n,m | Decimal numbers corresponding to card sequence numbers |

### *RESTORE

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.

```
*RESTORE a.n
```

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

| | |
|---|---|
| a,b | Alphanumeric identifier (a deck, common deck or correction set name) |
| n,m | Decimal numbers corresponding to card sequence numbers |

### *INSERT

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

```
*INSERT a.n
```

| | |
|---|---|
| a | Alphanumeric identifier (a deck, common deck or correction set name) |
| n | Decimal number corresponding to card sequence numbers |

*YANK All effect of correction sets may be removed with the *YANK card. The correction sets must exist at the time the *YANK card is encountered. *YANK restores the library to a form it had before the correction set occurred. The effect of any *YANK card included in the correction set is nullified. Two names separated by a period indicate an inclusive list of correction sets to be removed. Insertions may not follow the *YANK card.

```
*YANK a,b,c,d
```

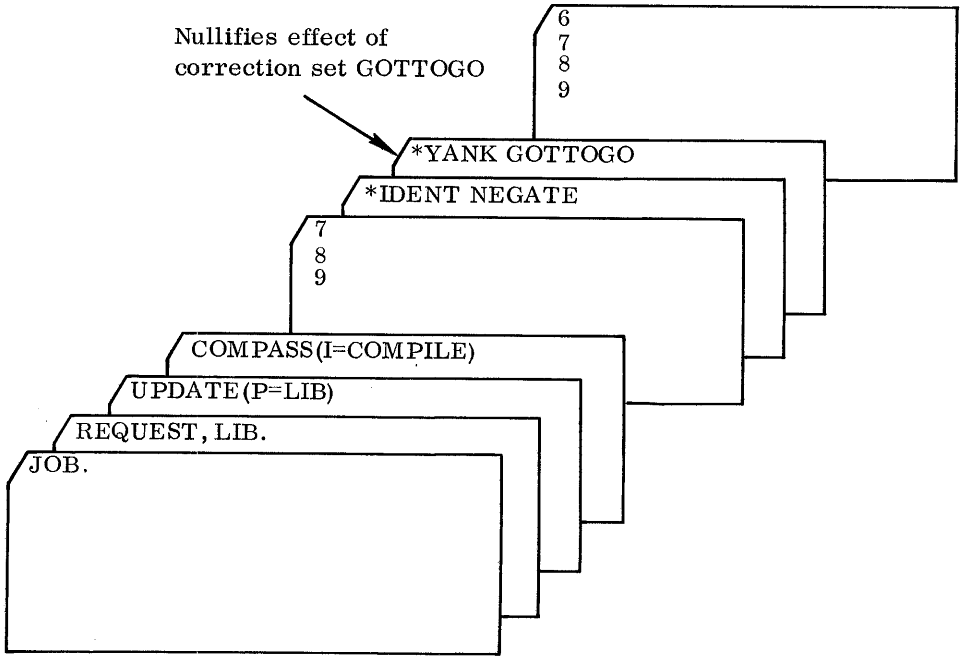
```
*YANK a.b
```

a,b,c,d Alphanumeric correction set names

Example:

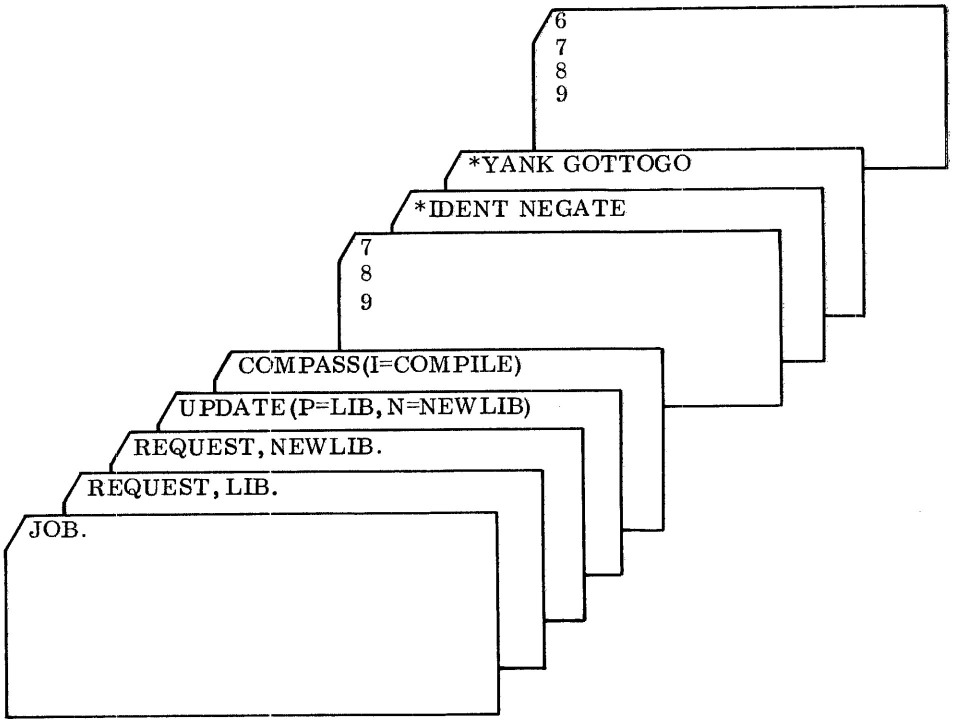
To reverse the effect of a correction set but not permanently remove the correction set from the program library LIB:

1. This change may be made temporarily for testing purposes:



```
JOB.
REQUEST, LIB.
UPDATE(P=LIB)
COMPASS(I=COMPILE)
7
8
9
*IDENT NEGATE
*YANK GOTTOGO
6
7
8
9
```

2. To put this change onto a new program library:



```
JOB.
REQUEST,LIB.
REQUEST,NEWLIB.
UPDATE(P=LIB,N=NEWLIB)
COMPASS(I=COMPILE)
7
8
9
*IDENT NEGATE
*YANK GOTTOGO
6
7
8
9
```

*/ (slash) A correction set may include comment cards; they have 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. A comment card may appear anywhere within the input deck or on a remote file; except within an ADDFILE.

```
*/Δany comments
```

*ADDFILE When *ADDFILE is encountered, UPDATE rewinds the named file fname, reads creation control cards and text card data, 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 must not be used when the UPDATE (Q) option is being used. Comment cards in this file are treated as text cards.

```
*ADDFILE fname,a.n
```

| | |
|---|---|
| fname | Name of file from which information is to be read |
| a | Alphanumeric identifier (deck, common deck or correction set name) |
| n | Decimal number corresponding to card sequence number |

### 6.3.4 DIRECTING THE OUTPUT

Output control cards are used to organize and order information output to the COMPILE file. *DECK and *COMDECK cards have additional functions during program library creation; these control cards are included in the program library.

*DECK

DECK delimits a deck for compile output. This division is meaningful during a correction run when the selective assembly feature is employed. Under the selective assembly 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.

```
*DECK dname
```

| | |
|---|---|
| dname | Deck name |

*COMDECK

This card introduces a common deck. Common decks are not written onto the COMPILE file; they are saved by UPDATE and introduced into the COMPILE file as a result of *CALL statement. <u>The common deck must precede the call</u>.

```
*COMDECK dname
```

| | |
|---|---|
| dname | Deck name |

* WEOR The WEOR card is used to organize the COMPILE file for ease of input to compilers, assemblers, etc. It writes an end-of-record of level n on the COMPILE file. If n is 15 or greater, a level $15_{10}$ end-of-record is written but not an end-file tape mark; level 0 is assumed if not specified. The *WEOR card does not appear on the COMPILE file. *WEOR may be inserted into a deck on the program library; it is effective only when that deck is written onto the COMPILE file.

```
*WEOR n
```

n Decimal number corresponding to SCOPE end-record level numbers (a number ≥ 15 writes level $15_{10}$ end-record but not an end-file tape mark). Level 0 is assumed if not specified.

*CWEOR CWEOR is a conditional WEOR. If information has been written on the COMPILE file when the CWEOR card is encountered, it acts as a WEOR card. However, if the COMPILE file has not been written on during the run, the CWEOR card is ignored and processing continues with the next card.

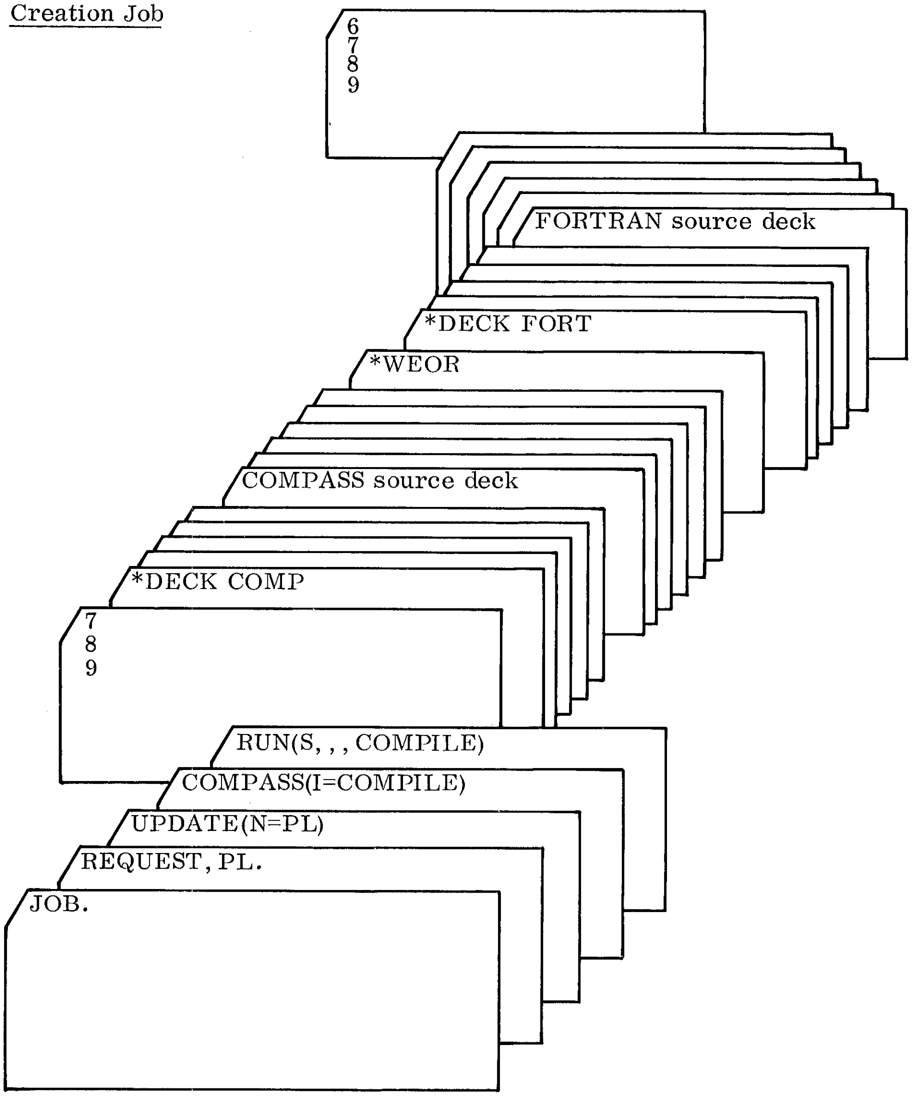
```
*CWEOR n
```

n Same as for WEOR

Example:

To create a program library (PL) consisting of a COMPASS deck and a FORTRAN deck and to generate a COMPILE file that would permit these decks to be processed by the assembler and compiler, the following deck structure is required:

Creation Job



Generated COMPILE File

| | |
|---|---|
| COMPASS deck<br>⋮ | Picked up and assembled by COMPASS |
| 7<br>8<br>9 | Terminates the COMPASS scan |
| FORTRAN deck<br>⋮ | Picked up by the FORTRAN compiler |
| 6<br>7<br>8<br>9 | Terminates the FORTRAN scan |

*CALL

This 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 repeating the data cards. The *CALL card does not appear on the COMPILE file. The contents of the common deck, excluding the *COMDECK card, follow immediately. The *CALL card cannot occur within a common deck.

```
*CALL dname
```

dname        Name of common deck name

### 6.3.5 SELECTIVE ASSEMBLY

The selective assembly feature is handled by the control cards described below. This feature is used in determining information to be written on the COMPILE file. Normally, only modified decks are written onto the COMPILE file. To control this process at the deck level, COMPILE cards are introduced.

*COMPILE

The *COMPILE card specifies decks to be assembled. With the first form shown below specific decks are mentioned. In the second form, a range of decks is specified. Decks written on the COMPILE file 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 written on the COMPILE file regardless of the existence of any modifications within them.

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

```
*COMPILE  a.d
```

a,b,c,d      Deck names

If the full assembly (F) feature is selected on the UPDATE call card, the *COMPILE cards are ignored.

### 6.3.6 SPEED UPDATING

For speed updating, the Q option must be specified on the UPDATE call card. It is effective 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. <u>The names specified in the *COMPILE statements must include all routines to be modified.</u>
2. When corrections are made to common decks, UPDATE does not automatically include on the COMPILE file the decks which call the common deck. Propagation of these modifications is left to the user; he must specify each such deck on the *COMPILE card.
3. If corrections are specified to routines not included in the *COMPILE statements, UPDATE will print unprocessed corrections.
4. Selective assembly has no effect.
5. A new program library cannot be produced; if so specified, it is ignored.
6. If a source file is requested (S), all the common decks as well as decks specified only on the *COMPILE cards will be produced.

### 6.4 LISTABLE OUTPUT FROM UPDATE

Creation of a new program library produces a list file containing a copy of all file manipulation and creation cards and a list of deck names and correction set names known at the end of the UPDATE run, as well as error diagnostic messages.

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 input by the *READ command are included and 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.

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

When a COMPILE file is written, the locations of all *WEOR and *CALL cards are listed. If L = 0 on the UPDATE card, all listable output from UPDATE is suppressed. If L = 1, the deck name list, identifier list, and continuous commentary are suppressed.

## 6.5 OVERLAPPING CORRECTIONS

A correction set can be remodified 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 sets are introduced. The order of specification is irrelevant if no correction is dependent on another. If a dependent relationship exists, however, the order is of paramount importance.

For example, if a numbered insertion in a correction set is subsequently deleted, the insertion card is present but inactive. When the deletion occurs first, however, by specifying a range of cards and then indicating the point at which insertion is to be made, the numbered insertion can be made 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; if the card following *INSERT is another UPDATE control card, no insertion will be made.

```
*IDENT MODSUB1
*/ΔTHE CHANGES MADE BY THIS CORRECTION SET
*/ΔDEPEND ON THE IMPLEMENTATION OF THE
*/ΔCORRECTION SET - MOD - IF THIS SET OF
*/ΔCORRECTIONS HAS NOT BEEN ADDED TO THE
*/ΔPROGRAM LIBRARY ABORT THIS CORRECTION
*/ΔSET
*INSERT MOD.1
*INSERT A.n
```

Correction subset (MODSUB1) depends on the initial correction set (MOD) having been previously inserted in the sequence.

## 6.6 UPDATE EDITLIB INTERFACE

Unlike EDITSYM, UPDATE does not produce a COMPILE file with routines in the order in which modifications were made to them; rather, the order of routines on the COMPILE file is determined by their position on the OLDPL.

Input cards to EDITLIB (such as ADD cards) should be ordered according to routines' positions on the OLDPL, not in the order of mention in the UPDATE modification cards.

## 6.7 FILES PROCESSED BY UPDATE

With the control card identifiers indicated below in the second column, the user may process several UPDATE files; some may be assigned to tape units.

| File | Identifier | Function | Type | Position after Update Call |
|---|---|---|---|---|
| Input | I | Provides control information | Coded | Left at end of record terminating update control cards; other cards may follow. Position after abort unpredictable. |
| Output | L | Produces listings | Coded | At current position of file. If L ≠ OUTPUT, fname not rewound. |
| Compile | C | Produces images for assembly or compilation | Coded | Rewound before and after UPDATE operation unless rewind suppressed by R option. |
| Old PL | P | Contains old PL | Binary | Rewound before and after UPDATE; no rewind if R option specified. |
| New PL | N | Contains new PL | Binary | |
| Source | S | Contains copy of cards to regenerate existing library, inactive cards omitted. | Coded | Same as P and N |

NOTE: Since the validity of sequence numbers has not been ascertained for text information following any particular card image, if an irrecoverable parity error occurs, the UPDATE run will terminate.

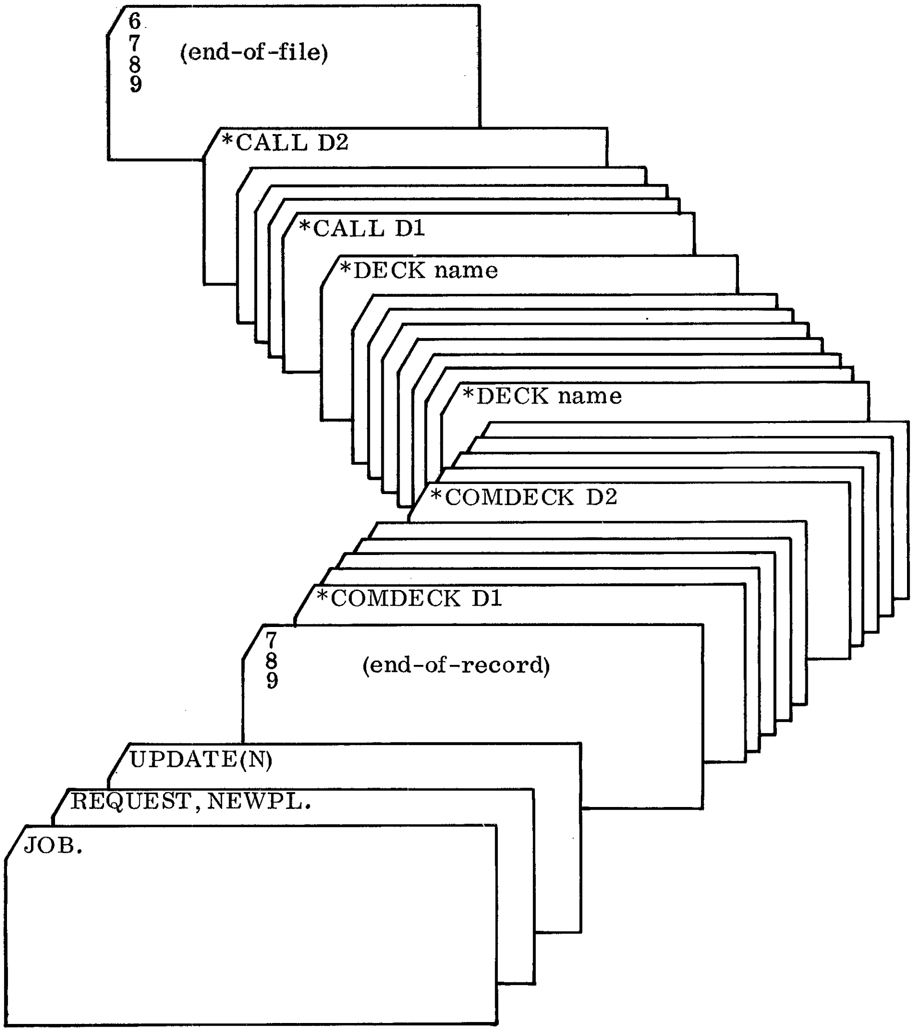
The following files are used by UPDATE in processing:

| File Name | Functions | Comments |
|---|---|---|
| CMPSCR | Scratch file. During library creation, it holds entire symbolic source. During correction runs, it holds programs as they are updated. CMPSCR is used also to copy decks to be written to COMPILE file. (To avoid disk conflicts, tape assignment is often desirable if large volume files are involved.) | Binary file. Rewound prior to and after updating. Not used if full assembly mode is selected although it is addressed by UPDATE. |
| COMDKS | Used internally by UPDATE | Must be mass storage files. Files are evicted before and after the updating operation. |
| FTEXT | Used internally by UPDATE | |

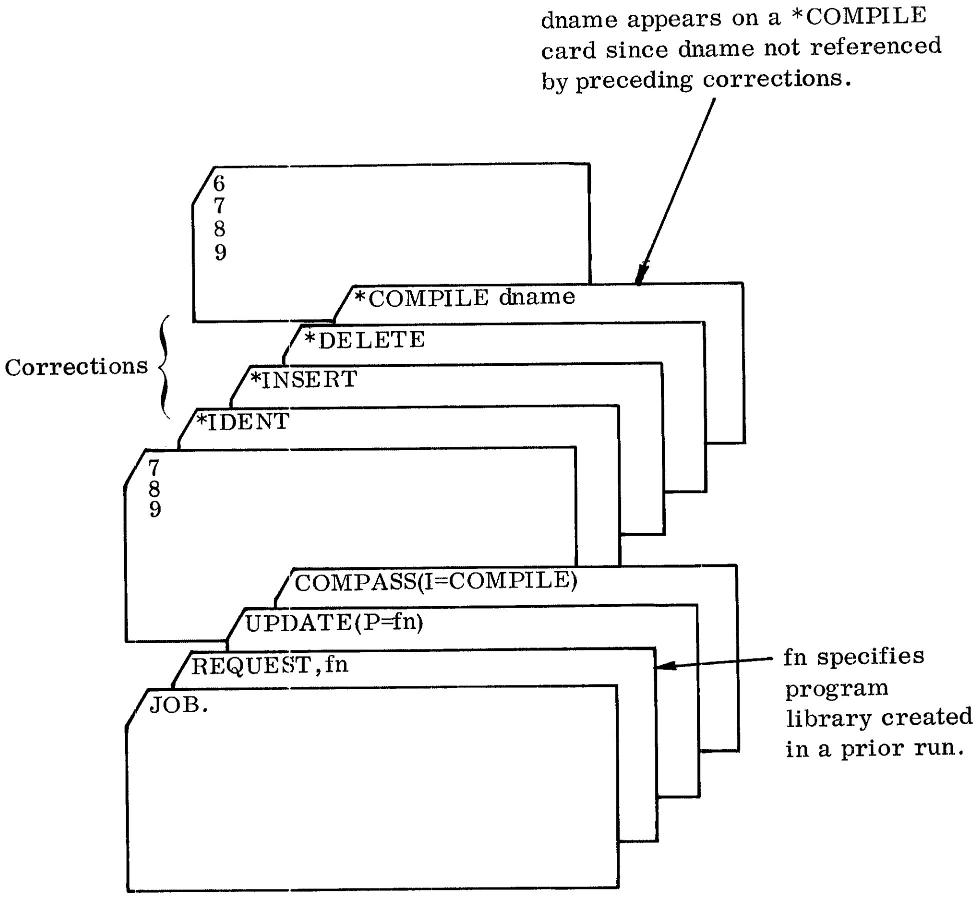
Files mentioned in ADDFILE, READ, REWIND and SKIP operations are left as defined by the latest directive encountered on the input stream.

## 6.8 UPDATE CARD DECK EXAMPLES

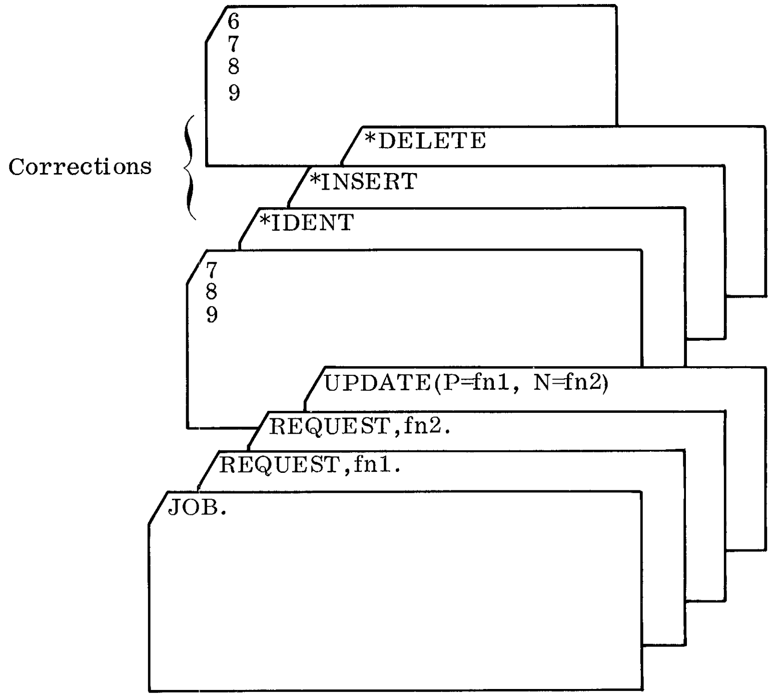
1. To set up an UPDATE program library containing two normal decks and two common decks:



2. To modify a program library and produce an assembly listing:

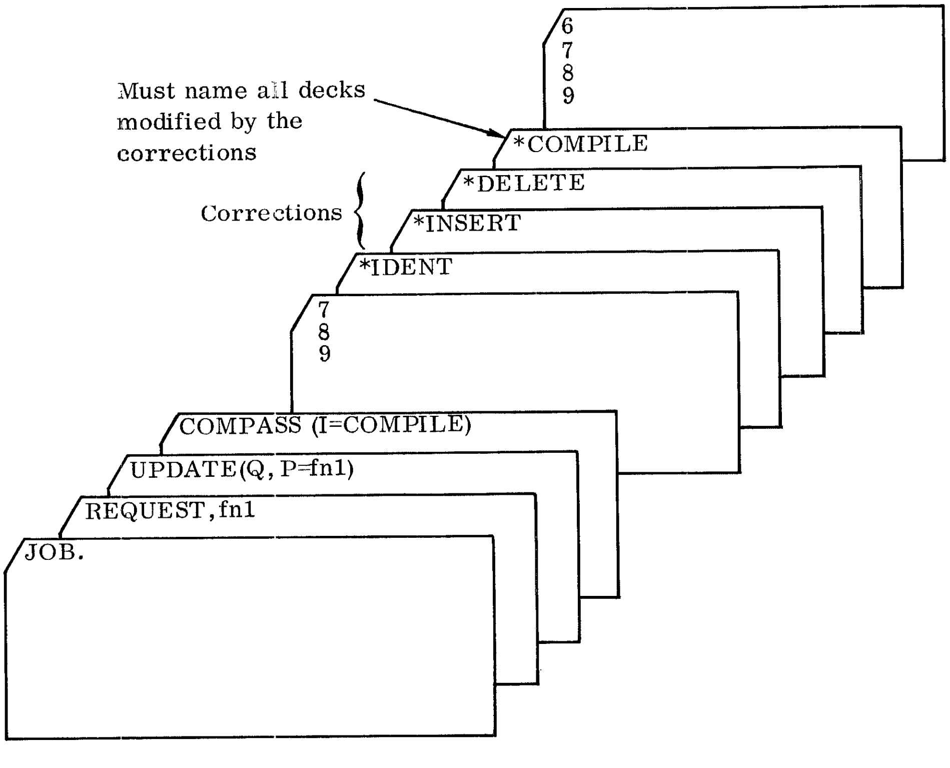

dname appears on a *COMPILE card since dname not referenced by preceding corrections.
6
7
8
9
*COMPILE dname
*DELETE
Corrections
*INSERT
*IDENT
7
8
9
COMPASS(I=COMPILE)
UPDATE(P=fn)
REQUEST,fn
fn specifies program library created in a prior run.
JOB.

3. Run to generate a new program library with corrections:

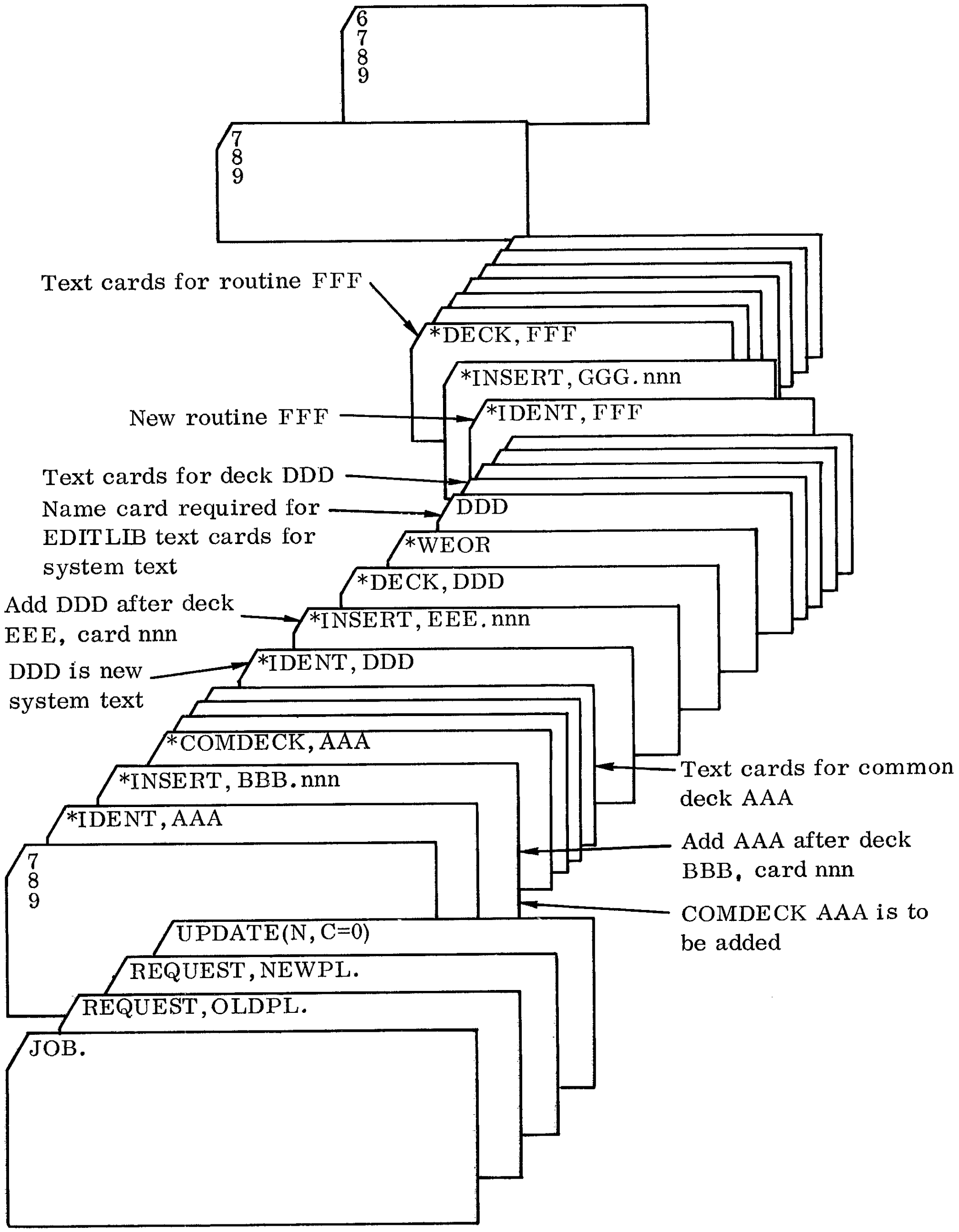


```
6
7
8
9
*DELETE
*INSERT
*IDENT
7
8
9
UPDATE(P=fn1, N=fn2)
REQUEST,fn2.
REQUEST,fn1.
JOB.
```

4. Example of Q-option use. UPDATE temporarily places the data from decks on file COMPILE (may be otherwise specified) and terminates. COMPASS picks up data from COMPILE file and assembles it.



5. To construct a new program library from the old program and add a new routine, a new common deck, and a new SYSTEXT deck by calling UPDATE.



Text cards for routine FFF

New routine FFF

Text cards for deck DDD

Name card required for EDITLIB text cards for system text

Add DDD after deck EEE, card nnn

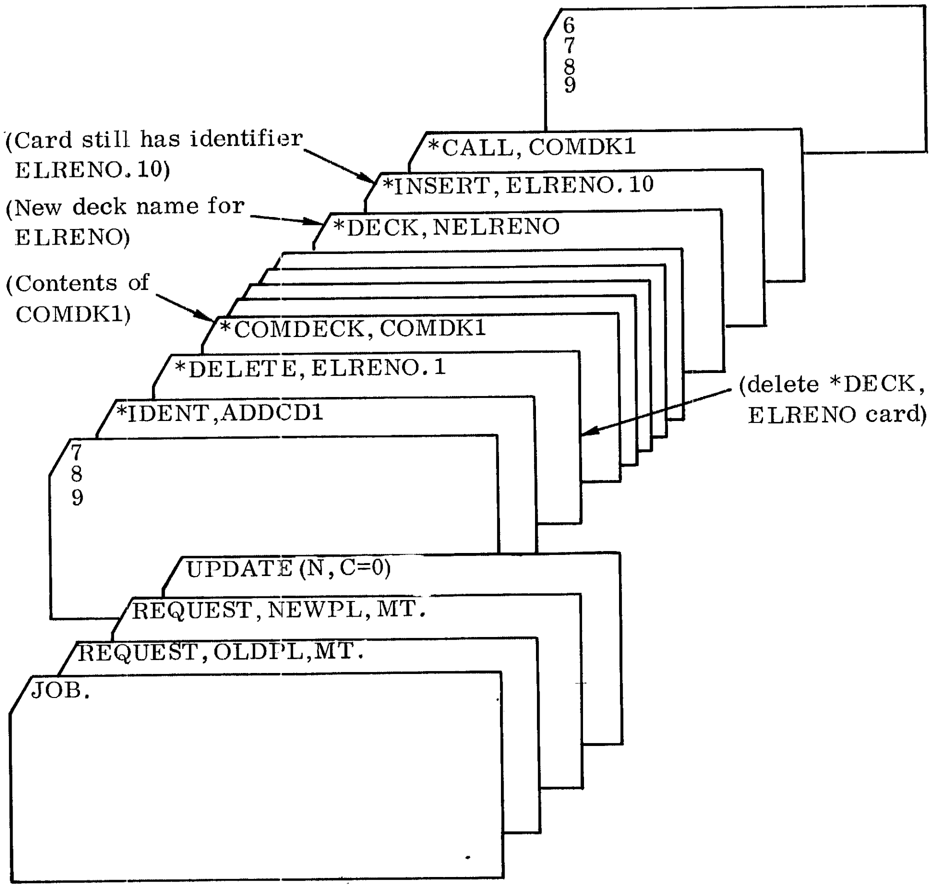
DDD is new system text

Text cards for common deck AAA

Add AAA after deck BBB, card nnn

COMDECK AAA is to be added

6. To insert a common deck (COMDK1) at the beginning of a new program library. The common deck must be at the beginning of the new program library since the first deck (ELRENO) on the old library will call for the common deck to be inserted, and common decks must precede the *CALL for the deck.



```
6
7
8
9
*CALL, COMDK1
*INSERT, ELRENO.10
*DECK, NELRENO
*COMDECK, COMDK1
*DELETE, ELRENO.1
*IDENT,ADDCD1
7
8
9
UPDATE (N,C=0)
REQUEST, NEWPL, MT.
REQUEST, OLDPL,MT.
JOB.
```

(Card still has identifier ELRENO.10)

(New deck name for ELRENO)

(Contents of COMDK1)

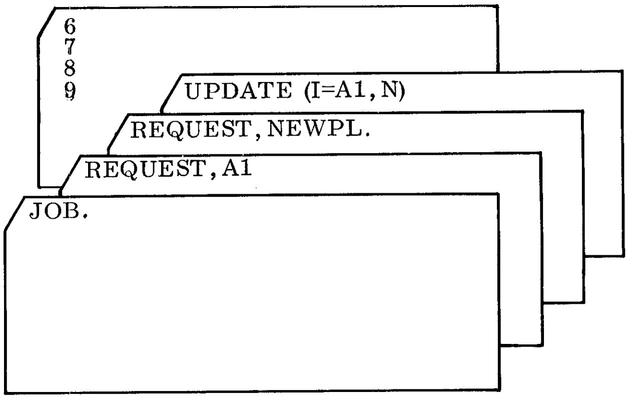
(delete *DECK, ELRENO card)

7. Simplified example for generating a program library by calling UPDATE:

Contents of file A1:

```
*DECK,SET1
   PROGRAM ZIP
C   A DO-NOTHING JOB
    END
*DECK,SET2
   SUBROUTINE JIM
   A = B - SIN(C)
   END
*COMDECK,CSET
   COMMON A,B,C
6/7/8/9
```

For the UPDATE task:



```
JOB.
REQUEST,A1
REQUEST,NEWPL.
UPDATE (I=A1,N)
```

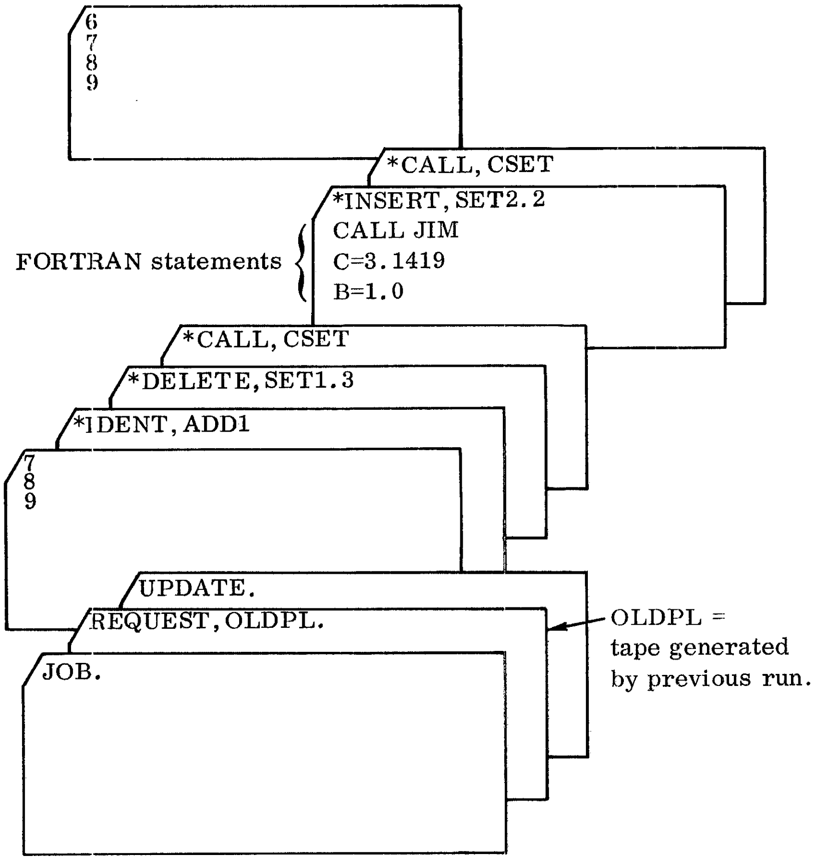
The contents of file COMPILE will contain:

```
decks     SET1     SET2
```

which contain:

```
     PROGRAM ZIP          SET1   00002
C    A DO-NOTHING JOB     SET1   00003
     END                  SET1   00004
     SUBROUTINE JIM       SET2   00002
     A = B - SIN(C)       SET2   00003
     END                  SET2   00004
```

To alter the program:



```
6
7
8
9
*CALL, CSET
*INSERT,SET2.2
CALL JIM
C=3.1419
B=1.0
*CALL, CSET
*DELETE,SET1.3
*IDENT,ADD1
7
8
9
UPDATE.
REQUEST,OLDPL.
JOB.
```

Results in decks SET1 and SET2 on the COMPILE file; contents are:

| | | |
|---|---|---|
| PROGRAM ZIP | SET1 | 00002 |
| COMMON A, B, C | CSET | 00002 |
| B=1.0 | ADD1 | 00002 |
| C=3.1419 | ADD1 | 00003 |
| CALL JIM | ADD1 | 00004 |
| END | SET1 | 00004 |
| SUBROUTINE JIM | SET2 | 00002 |
| COMMON A, B, C | CSET | 00002 |
| A = B - SIN(C) | SET2 | 00003 |
| END | SET2 | 00004 |

## 6.9 UPDATE MESSAGES

Listing Messages

***ABOVE CARD ILLEGAL DURING CREATION RUN

***ADDFILE FIRST CARD MUST BE *DECK OR *COMDECK BUT WAS:

***ADDFILE INVALID FROM *READ FILE

***ADDFILE INVALID WITH Q-OPTION

***CARD LENGTH ERROR ON OLD PROGRAM LIBRARY

***CARD NUM ZERO OR INVALID CHAR IN NUM FIELD

***CONTROL CARD INVALID OR MISSING

***DECK NAMES SEPARATED BY PERIOD IN WRONG ORDER

***DUPLICATE DECK NAME

***DUPLICATE IDENT NAME

***ERROR______DECK DOES NOT EXIST

***______ERRORS IN INPUT: NEWPL, COMPILE, SOURCE SUPPRESSED

***FILE NAME LONGER THAN 7 CHARACTERS

***IDENT______UNKNOWN

***IDENTIFIER LONGER THAN 7 CHARACTERS

***IDENTIFIERS SEPARATED BY PERIOD IN WRONG ORDER

***INVALID NUMERIC FIELD

***NO DECK NAME

***NO SUCH COMMON DECK

***NOT ALL MODS WERE PROCESSED

***NULL ADDFILE

***ON THE ABOVE CARD THE FIRST LIMIT EXCEEDS TERMINAL LIMIT

***OR A REFERENCE IS MADE TO DECK NOT MENTIONED ON COMPILE CARD (This line appears only if the Q option is in effect)

***PREMATURE END OF RECORD ON OLD PROGRAM LIBRARY

***RESERVED FILE NAME

***THESE MAY BE MODS TO DECKS NOT MENTIONED ON COMPILE CARD OR AN INCOMPLETE ADDFILE

***UNKNOWN IDENTIFIER

Display and Dayfile Messages

CORE OVERFLOW

DECK STRUCTURE CHANGED

FILE NAME ON UPDATE CARD GR 7 CHAR, UPDATE ABORTED

IMPROPER UPDATE PARAMETER, UPDATE ABORTED

ONE OR MORE OVERLAPPING CORRECTIONS

SKIPPING (appears during Q-mode skip; does not appear in Dayfile)

UPDATING deckname (does not appear in Dayfile)

UPDATE ERRORS, JOB ABORTED

UPDATING FINISHED

______ERRORS IN UPDATE INPUT

# EDITSYM 7

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

| 59 | 53 | 47 | 35 | 11 0 |
|---|---|---|---|---|
| $77_8$ | 0 | 1 | 0 | N |

N 2-digit display code number

= 02 common section

= 03 text sections

.

.

.

= 99

Word 2 of common and text sections

| 59 | 17 | 11 0 |
|---|---|---|
| deck name | 00 | edition number |

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 |

(NPL is unlabeled, if written on tape).

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

$*COPY,dn_1,dn_2$

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. The editing process does not depend upon a *COPY function. 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

CHECKPOINT/RESTART is a system facility which captures the total environment of a job on magnetic tape so that the job may be restarted from the same point in processing. Total environment includes local files associated with the control point of the job. For mass storage files (drum or disk), the complete file is captured as well as the relative position within that file. For magnetic tape files, only the relative position on the tape is captured, so the tape may be properly re-positioned during restart.

Checkpoint/Restart cannot handle:

Respond jobs

Rolled-out jobs

Random files

Common files

Multi-file reels.

When a programmer takes a checkpoint dump during job execution a file is written containing all information needed to restart the job at that point. In the event of machine malfunction, operator error, or program error, the job can be restarted from the last checkpoint rather than the beginning of the job.

## 8.1 CHECKPOINT REQUEST

A checkpoint dump may be requested by a CKP control card in the job stream, an executing program, or by a console message entered by the operator. An executing program would request CHECKPOINT at logical points within its execution such as end-of-file, x logical records processed, x seconds of elapsed time, etc. CHECKPOINT requests may be issued more than once. CHECKPOINT is requested as follows:

CHECKPT param, sp.

<table>
<tr><td>59</td><td>41</td><td></td><td>39</td><td>35</td><td>29</td><td>23</td><td>17 0</td></tr>
<tr><td colspan="5">▨</td><td colspan="2">RJ</td><td>CPC</td></tr>
<tr><td>CKP</td><td>1</td><td>1</td><td>▨</td><td colspan="2">sp</td><td>▨</td><td>param</td></tr>
</table>

sp = zero indicates all mass storage files are to be processed

sp = nonzero indicates a limited set of files

param Address of a parameter list within user's relocatable code; format follows:

| 59 | 17 | 11 0 |
|---|---|---|
| ▨ | n | 0000 |
| lfn1 | f1 | ▨ |
| lfn2 | f2 | |
| ⋮ | | |
| lfnn. | fn | ▨ |

n defines number of terms in the list (number of lfn entries); maximum value is $42_{10}$.

$\text{lfn}_i$ name of the i-th user mass storage file in list.

$f_i$ flag indicating specific manner in which $\text{lfn}_i$ is to be processed.

The low order 12 bits of the first word in the list must be zero during each call to CHECKPOINT. The user should clear these bits before each CHECKPT request because preceding checkpoint calls will have set them to a non-zero value.

For a general call to CHECKPOINT using the macro call, the sp field will be zero:

If n = 0, all mass storage files assigned to the control point including INPUT, OUTPUT, PUNCH, PUNCHB, and LGO will be copied to the CHECKPOINT dump tape in the manner determined by the last code/status (f flags).

If n ≠ 0, all mass storage files named in the lfn list will be copied to the CHECKPOINT dump tape in the manner determined by the f flags, except for system mass storage files which are copied as determined by the last operation performed on each file.

The f flags can have only the following values:

If f = 0, the mass storage file is copied from beginning of information to its present position at checkpoint time; and only that portion will be available at restart time. The file is positioned at the latter point.

If f = 1, the mass storage file is copied from the present position at checkpoint time to end of information; and only that portion will be available at restart time. The file is positioned at the former point.

If f = 2 or 3, the mass storage file is copied as determined by the last operation on that file. Generally, these values of f are used when the value of sp is non-zero.

If the value of the sp field is non-zero in the macro call, only the lfn's supplied by the user in the param list plus system files will be processed. Processing is determined by the f flag settings.

When the manner of copying a mass storage file is to be determined from the last operation on the file, CHECKPOINT derives f-values from the last code/status as follows:

f = 0 if code/status ends in 4, 5, 6, or 7.

f = 0 if code/status ends in 0, 1, 2, or 3 and end-of-information bit is set.

f = 1 if code/status ends in 0, 1, 2, or 3 and end-of-information bit is not set.

This generally causes the entire mass storage file to be copied for write operations, read operations resulting in end-of-information status, and rewind operations (excluding some OPEN functions).

Examples:

For COMPASS users:

```
           CHECKPT    PARAM
              :
PARAM      DATA          0
```

All mass storage files would be CHECKPOINT processed (sp=0, n=0).

All operator or CKP control card requests are processed in the same manner as this example.

For FORTRAN (RUN) users:

```
    DATA (variable=0)
        .                          or
        .                                variable=0
    CALL CHEKPTR (variable)              CALL CHEKPTR (variable)
```

For FORTRAN Extended users:

```
    DATA (variable=0)
        .                          or
        .                                variable=0
    CALL CHEKPTX (variable)              CALL CHEKPTX (variable)
```

For the above examples checkpoint processing is performed for all mass storage files as described above for sp=0, n=0. Selected files may be processed if the pattern shown in the following example is followed.

```
    DIMENSION KPARAM(4)
    KPARAM(1)=30000B
    KPARAM(2)=5LTAPE1.OR.10000B
    KPARAM(3)=6LTAPE23.OR.10000B
    KPARAM(4)=5LTAPE3
        .
        .
        .
    CALL CHEKPTR(KPARAM,1) or CALL CHEKPTX(KPARAM,1)
```

CHECKPOINT requests from overlay programs

The user should rewind the overlay files prior to requesting CHECKPOINT if they are on mass storage.

FORTRAN overlay programs should declare the mass storage overlay files to be TAPEn files and use REWINDn before CALL CHECKPTR (variable) or CALL CHEKPTX (variable).

Example for FORTRAN (RUN) users:

```
    OVERLAY(TAPE9,0,0)
    PROGRAM MAIN(...,TAPE9)
    DATA(FILE=5LTAPE9)
        :
    CALL OVERLAY(FILE,1,0)
        :
    END
    OVERLAY(TAPE9,1,0)
    PROGRAM OVER1
    DATA PARAM/0B/
        :
    REWIND9
    CALL CHEKPTR(PARAM)
        :
    END
```

CHECKPOINT requests from COBOL and SORT/MERGE

Refer to the COBOL and SORT/MERGE manuals for CHECKPOINT/RESTART description.

CHECKPOINT dump tape

With a REQUEST control card the user may specify an unlabeled tape with checkpoint disposition on which the checkpoint'dumps are to be written. This REQUEST should be the first control card of the job. If no such tape is supplied, CHECKPOINT will define an unlabeled tape with the name CCCCCCC as a local file the first time CHECKPOINT is requested including operator initiated checkpoints. In any event, only one checkpoint dump tape should be defined for the job.

CHECKPOINT/RESTART defines the following files for its use:

CCCCCCC<br>
CCCCCCI<br>
CCCCCCM

The user should refrain from using these file names.

## 8.2 RESTART REQUEST

The RESTART control card directs a job to be restarted from its checkpoint tape. This card has five possible formats:

```
RESTART,name,#.     RESTART,name.     RESTART.
RESTART,#,name.     RESTART,#.
```

name  Name of checkpoint file as defined at checkpoint time. If the name expression is omitted, the file name CCCCCC is assigned as a default value.

#  Number (decimal) of checkpoint to be restarted. If the expression is omitted, a default value of 1 is assigned by RESTART. If the number is greater than the number of the last checkpoint taken, the restart attempt will be terminated.

After locating the proper checkpoint dump on the checkpoint tape, the restart program requests all tape files which were defined at checkpoint time, and repositions these files. The restart program also re-establishes all mass storage files from the copies appearing on the checkpoint tape, restores the central processor program, and restarts the user's job.

The restart job should not contain any REQUEST control cards; RESTART requests all necessary files internally.

Permanent files are not copied to the checkpoint tape. However, if any permanent files are attached to the control point when CHECKPOINT is called, their local file names will be listed in the job dayfile with a message.

The user should attach all these permanent files to the control point, and reposition them before calling RESTART.

Any ECS user area attached to the control point will be copied in its entirety to the checkpoint tape. At restart time, it will be recopied to ECS from the checkpoint tape. On the job card for the restart job, the user must request at least as much ECS as was attached to the original checkpointed job. If reconfiguration results in insufficient ECS available to the user, restart is not possible.

## 8.3 UNRESTARTABLE CHECKPOINT DUMPS

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

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

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

# SYSTEM/OPERATOR COMMUNICATION 9

## 9.1 DISPLAY CODES

SCOPE communicates with the operator through two or more console display screens and a keyboard. The major display programs are the System Display (controlled by the program DSD), and the Control Point Job Display (controlled by the program DIS).

The system indicates the status of operations on the console screens. The operator may introduce jobs, change job priorities, and examine selected portions of memory via the keyboard. Data entered from the console is also displayed. A permanent record of all system/console communication is retained by the system in a dayfile which may be printed at operator request.

### 9.1.1 DSD

The display console is normally controlled by the system display package DSD, which permanently resides in peripheral processor 9. DSD maintains a current display of the system status and processes keyboard entries from the operator. At the console keyboard, the operator may assign equipment, exercise control over execution and job scheduling, initiate utility programs, select displays, etc.

The screens may be assigned to any combination of two displays:

| Name | Display |
|---|---|
| A | System or control point dayfile |
| B | Job status (for all control points) |
| C | Data storage |
| D | Data storage |
| E | Equipment status |
| F | File name table |
| G | Program storage |
| H | Input/output queues |
| I | Unprocessed control cards |
| J | JANUS control point status |

| Name | Display |
|---|---|
| K | Control point area or system table addresses |
| L | Central programmable |
| M | PP communications area |
| N | Reserved for PP debugging display; available from VIM |
| O | Operator messages |
| Y | Command format syntax dictionary |
| Z | Display dictionary |

### 9.1.2 DIS

DIS, similar to DSD, displays information relevant to a single job assigned to a control point. Under DIS, the B display shows the exchange jump area of a job. Central memory addresses relative to a job's reference addresses are used for data and program display. DIS can be brought to a control point to monitor the progress of a job, or it can be brought to an empty control point to initiate utility routines, change priorities and suspend job execution.

For a complete description of DSD and DIS, their displays, keyboard commands and error messages, see the SCOPE Operator's Guide.

# 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 mass storage unit.

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 mass storage unit. All files, except mass storage, 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 mass storage. 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,CM1000,P6.  (Job Card)
```

Assign unique file name MAGTAPE with a REQUEST control card to a tape unit:

| | |
|---|---|
| REQUEST,MAGTAPE,MT. | Operator would assign specific tape unit. |
| REWIND(MAGTAPE) | Rewinds tape unit to be sure of position. |
| COPYCF(MAGTAPE,XX,2) | Skips tape to beginning of third file by copying first two files to an unused dummy file XX. |
| COPYCF(MAGTAPE,OUTPUT,2) | Copies the two coded files to the output file. OUTPUT is automatically printed at end of job. |
| 6/7/8/9 | End-of-file card completes the job. |

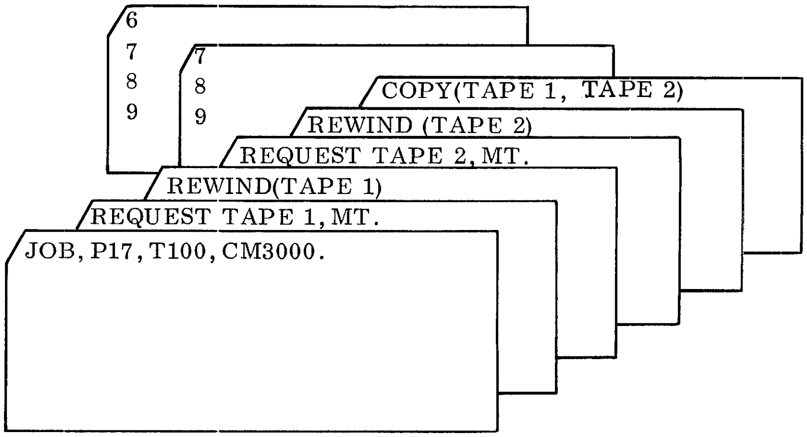
## 10.1 COPY ROUTINES

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. COPY will not operate on S or L tapes, on labeled tapes, or on BCD tapes.

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:



```
JOB,P17,T100,CM3000.
REQUEST TAPE 1,MT.
REWIND(TAPE 1)
REQUEST TAPE 2,MT.
REWIND (TAPE 2)
COPY(TAPE 1, TAPE 2)
```

## MULTI-PURPOSE COPY

The system library multi-purpose copy routine is a CP routine with entry points COPYCR, COPYCF, COPYBR, COPYBF. If the number of records or files is unknown or not readily determined, an exceedingly large parameter may be specified. The following parameter information is pertinent for the four copy control cards that follow:

> file 1 and file 2 name the input and output files. Information is copied from file 1 onto file 2. If these files are not specified by name, INPUT and OUTPUT are assumed.
>
> n is a decimal number indicating how many files or records are to be copied. If n is omitted only one file or record is assumed.

L indicates that the first record on the input file contains label information that is to be copied onto the output file header label. When the L parameter is used, file 1, file 2, and n are also required.

### COPY BINARY FILE

```
COPYBF(file1,file2,n,L)
```

### COPY CODED FILE

```
COPYCF(file1,file2,n,L)
```

### COPY BINARY RECORD

```
COPYBR(file1,file2,n,L)
```

### COPY CODED RECORD

```
COPYCR(file1,file2,n,L)
```

The first two routines terminate when the specified number of files are read, or when an end-of-information is encountered.

The latter two routines terminate when the specified number of records are read or when a file mark is encountered. For example, if the card specifies 100 records but the file contains only 50 records, the copy operation terminates after 50 records.

### General Comments pertaining to COPYBF,COPYCF,COPYBR,COPYCR

Error recovery is handled by SCOPE. If, after a number of re-trys, a parity error persists (a PARITY ERROR message appears on the console display), the copy should be abandoned by the operator due to the indeterminate state of the data.

When an end-of-reel is detected, the next reel is obtained, label checking/ writing is performed if the tape is labeled, and the function continues normally on the next reel.

If an end-of-file is encountered on the input file before the record count is exhausted, the copy operation will cease (but not abort) at that point. A message is entered in the dayfile. An EOF is written on file2 and backspaced over and file is left open.

If an end-of-information is encountered on the input file before the record/ file count is exhausted, the copy operation will cease (but not abort) at that point. A message is entered in the dayfile; an EOF is written on file 2; both files are closed.

The copy routines open the files specified on the copy control card. At the conclusion of the copy operation, only those files on which EOI has been encountered are closed. The COPY routines require a field length of 20000B.

Although not primarily implemented for that purpose, the copy routine is capable of limited format conversion. The following matrix shows format conversion copies that can be handled successfully:

| | | OUTPUT | | | |
|---|---|---|---|---|---|
| | | SCOPE | X | S | L |
| INPUT | SCOPE | Yes | Bin Yes<br>BCD Yes 2 | Yes 1 | Yes |
| | X | Bin Yes<br>BCD Yes 2 | Yes 3 | Bin Yes 1<br>BCD Yes | Yes |
| | S | Bin Yes<br>BCD Yes 1 | Bin Yes<br>BCD Yes 2 | Yes | Yes |
| | L | Bin Yes<br>BCD Yes 1 | Bin Yes<br>BCD Yes 2 | Yes 1 | Yes |

NOTES:

1. Conversion of this type cannot be guaranteed because of possible truncation of the input record. Maximum record size for S tape output files is 512 words (5120 coded characters). Maximum physical record size for coded SCOPE tapes is 1280 characters. If these sizes are exceeded, the output record is truncated and the copy allowed to proceed after entering a message into the dayfile.

2. Because of potential loss of data significance within the format conversion, the output data cannot be guaranteed; a diagnostic is entered in the dayfile and the copy allowed to proceed.
3. BCD records may be up to 136 characters on input and will always be 136 characters on output.

COPY LABEL CARD FORMAT

```
*COPYLAB(LAB=x,EN=x,RN=x,CD=x,RC=x,MFN=x,PN=x)
```

If an L parameter is specified on any copy control card that routine will read the next record on the input file and that record should contain the *COPYLAB card. The LAB parameter is required; all others are optional and will be set to system default values if omitted (section 3.3).

| | | Maximum characters |
|---|---|---|
| LAB= | Label name (written on output and used to open input if labeled) | 17 |
| EN= | Edition number | 2 |
| RN= | Reel number | 4 |
| CD= | Creation date | 5 |
| RC= | Retention cycle | 3 |
| MFN= | Multi-file name | 6 |
| PN= | Position number | 3 |

Columns 1-9 must contain *COPYLAB, or *COPYLAB(

LAB must be the first parameter, others are order independent.

All fields are separated by commas.

Blanks are not permitted between fields.

Imbedded blanks may appear in the LAB field only.

The end delimiter is a period or right parenthesis.

The L parameter must appear on the copy control card if the input or output file is declared labeled. If the copy of a labeled tape is to retain the existing input label, the COPYLAB card requires only the label name.

Example: *COPYLAB(LAB=NAME).

COPY SHIFTED BINARY FILE

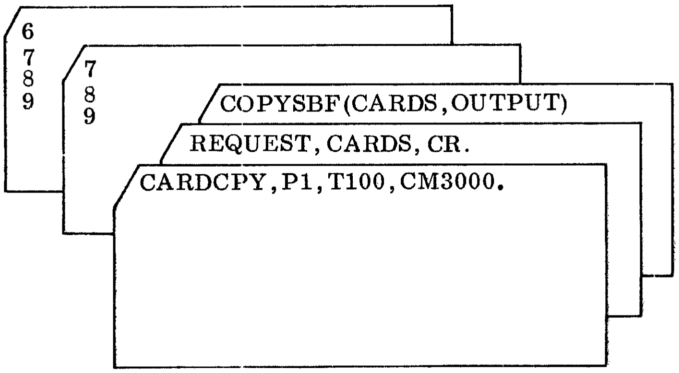
```
COPYSBF(file 1, file 2)
```

A single file 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.



```
CARDCPY,P1,T100,CM3000.
REQUEST,CARDS,CR.
COPYSBF(CARDS,OUTPUT)
```

COPYBCD

```
COPYBCD(file1,file2,n)
```

This routine copies packed output files to a magnetic tape where each line image is a discrete physical record, so the tape may be listed offline.

Default values for the parameters are INPUT, OUTPUT, and 1 respectively.

Allowable control card formats and the interpretation of each is given below:

| | |
|---|---|
| COPYBCD. | (INPUT,OUTPUT,1) |
| COPYBCD(n) | (INPUT,OUTPUT,n) |
| COPYBCD(file1) | (file1,OUTPUT,1) |
| COPYBCD(file1,n) | (file1,OUTPUT,n) |
| COPYBCD(file1,file2) | (file1,file2,1) |
| COPYBCD(file1,file2,n) | (file1,file2,n) |

Any other format or illegal file names will cause the job to terminate with the dayfile message CONTROL CARD ERROR.

COPYN

COPYN ($p_1$, out, $in_1$, $in_2$, . . . , $in_{10}$)

Logical records from up to ten binary input files ($in_1$-$in_{10}$) may be extracted and written on an output file (out). Record format is indicated by $p_1$; a non-zero value indicates the identification field (ID) of the logical records is to be omitted from the output file, zero indicates the records are to be copied verbatim. If records do not contain an ID, they are copied verbatim.

Text cards associated with the COPYN routine determine the order of the final tape. A routine may be selected from a composite tape, temporarily written on a scratch tape and transmitted as input to a translator, assembler, or programmer routine, eliminating the need for tape manipulation by the second program. Several tapes may be merged with COPYN to create a composite COSY or library tape. In its most basic form, COPYN can perform a tape copy.

The file names ($in_1$-$in_{10}$) reference binary files on tape, mass storage, or cards. A binary tape file consists of the information contained between load point and a double end-of-file; the tape file may contain any number of single end-of-file marks. A mass storage file ends with one file mark, and a card deck must be terminated by a record separator (7,8,9 punch in column one). The output file name may reference mass storage, tape, or card punch. A file mark for an output tape is written by a WEOF card or it may be copied in a range of records and counted as a record.

Records to be copied may or may not have an ID prefix control number (12 bits), number of words in the prefix (12 bits), and the name associated with the logical record. A record ID format consists of the first seven characters of the first word of each logical record. If logical records of the input file are not prefixed, all record identification cards must use the record number - the position of the logical record from the current position of the file.

REWIND, SKIPF, SKIPR, WEOF (write End-Of-File), and record identification cards may be used in conjunction with COPYN: these text cards are read from INPUT and are terminated by a record separator (7, 8, 9 punch in column 1). The text cards are free field; they may contain blanks but must include the separators indicated in each card description.

REWIND

```
REWIND (p)
```

This card generates a rewind of file p which must be one of the input or output file names given on the COPYN control card. File p may not be the system INPUT file.

SKIPF

```
SKIPF (p, ±n)
```

With this card, n file marks on file p may be skipped. File p must be a tape; requests for other types of files will be ignored. The skip may be forward (+n) or backward (-n). There is no indication when SKIPF causes a tape to go beyond the double end-of-file or when the tape is at load point.

## SKIPR

```
SKIPR (p, ±n)
```

With this card, n records may be skipped on file p. File p must be a tape; requests for other types of files will be ignored. The skip may be forward (+n) or backward (-n). Zero length records and file marks must be included in n.

## WEOF

```
WEOF (p)
```

This card writes a file mark on file p, which must be one of the input or output file names on the COPYN control card.

## RECORD IDENTIFICATION CARD

```
p1,p2,p3
```

The parameters on this card identify a record or set of records to be copied from a given file.

$p_1$ Record to be copied or the beginning record of a set of records to be copied. The name associated with the record or a number giving the position of the record from the current position of the file may be specified.

$p_2$ Last record to be copied in a set of records.

| | |
|---|---|
| name | logical records $p_1$ through $p_2$ are copied. |
| decimal integer n | n logical records are copied, beginning with $p_1$. Zero length records and file marks are counted. |
| * | $p_1$ through an end-of-file mark are copied. |
| ** | $p_1$ through a double end-of-file mark are copied. |
| / | $p_1$ through a zero length record are copied. |
| 0 or blank | only $p_1$ is copied. |

$p_3$ Input file to be searched. If $p_1$ is a name, and $p_3$ is omitted, all input files declared on the COPYN card are searched until the $p_1$ record is found. If it is not located, a diagnostic is issued. If $p_1$ is a number and $p_3$ is omitted, the last input file referenced is assumed. If this is the first text card, the first input file on the COPYN card is used.

Examples:

| | |
|---|---|
| SIN, TAN, INPUTA | Copies all logical records from SIN through TAN from file INPUTA. |
| SIN, 10, INPUTA | Copies 10 logical records from file INPUTA, from SIN through SIN+9. |
| SIN, TAN | Searches all input files beginning with current file. (If this is the first text card, the first input file named on the COPYN card is used). When SIN is encountered, all logical records from SIN through TAN are copied. |
| SIN, , INPUTA | Copies logical record SIN from file INPUTA. |
| 1, TAN, INPUTA | Copies the current logical record through TAN from file INPUTA. |
| 1, 10, INPUTA | Copies 10 logical records, beginning with the current logical record on file INPUTA. |
| 1, *, INPUTA | Copies the current logical record through the first file mark encountered on file INPUTA. |

## FILE POSITIONING

The files manipulated during a COPYN operation are left in the position indicated by the previously executed text card, they are moved only during a search. If file name ($p_3$) is omitted from the record identification card, all files on the COPYN card will be searched end-around. The end of a file is determined by a double end-of-file if tape, or a single end-of-file if mass storage. The first input file declared is searched until either $p_1$ or the original position of the file is reached, whereupon a search of the second input file begins. In this manner, all files remain effectively in the same position except the file containing $p_1$, which is positioned at $p_2$+1.

The output file is not repositioned after a search so that the COPYN routine may be re-entered, if desired. Therefore, the programmer is responsible for any REWIND, SKIP, or WEOF requests referencing the output file that may be necessary prior to exiting the job.

Example 1: Record identification card: REC,,INPUT1

Input file INPUT1:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ABLE | BAKER | ... | REC | SIN | TAN | ZEE | EE<br>OO<br>FF |

If INPUT1 were positioned at TAN, TAN and ZEE would be examined for REC. The double end-of-file would cause ABLE to be the next logical record examined, continuing until REC is read and copied to the output file. INPUT1 would then be positioned at SIN.

Example 2: Record identification card: RECA

Input file INPUT, 1 positioned at B1:

| | | | | |
|---|---|---|---|---|
| A1 | B1 | ... | Z1 | EE<br>OO<br>FF |

Input file INPUT2, positioned at loadpoint

| | | | |
|---|---|---|---|
| A2 | RECA | D2 | EE<br>OO<br>FF |

Input file INPUT3, positioned at loadpoint

| | | | | | |
|---|---|---|---|---|---|
| A3 | B3 | C3 | ... | Z3 | EE<br>OO<br>FF |

All routines from B1 through A1 are compared to RECA and INPUT1 is repositioned at B1. A2 is compared, then RECA is copied to the output file and INPUT2 is positioned at D2. INPUT3 is not searched.

Example 3:

Record Identification cards and binary logical records on INPUT file.

```
REC,,INPUT
JOB1,JOB3,INPUT
ABLE,,IN2
Record Separator (7,8,9 punch in column 1)
REC (binary)
Record Separator
JOB1 (binary)
Record Separator
JOB2 (binary)
Record Separator
```

JOB3 (binary)

Record Separator

Since there is no end-around search of the INPUT file, REC and JOB1-JOB3 must directly follow the requesting record identification cards in the order specified by those cards. An incorrect request for an INPUT record terminates the job.

## ERROR MESSAGES

The text cards are written on the system OUTPUT as they are read and processed. When an error occurs, the abort flag is set, and a message is printed (Appendix H) on OUTPUT followed by the card in error. This card is not processed and an attempt is made to process the next text card. When the last text card is processed, the abort flag is checked; if it is set, the job is terminated. Otherwise, control is given to the next control card.

## COPYL

```
COPYL(file 1,file 2,file 3)
```

This program allows for selective replacement of one or more routines. File 1 contains the old set of decks; file 2 contains the replacement routines, and file 3 contains the updated set of decks. 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:

```
COPYL(OLD, CORR,NEW)
```

The following job will update 1AJ, 1RA, and 2TS from a tape (OLD) which presumably contains the binary decks of these and other system routines. CORR will contain the new 1AJ, 1RA, 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
*IDENT,TEST
Modifications to 1AJ, 1RA and 2TS
*COMPILE 1AJ, 1RA, 2TS
7/8/9
6/7/8/9
```

COPYL can be used to find out the contents and order of routines on a deadstart tape by declaring the deadstart tape to be the correction file for a dummy file.

Example

```
Jobcard.
REQUEST,SYSTAPE,HI.      Assign deadstart tape
REWIND(SYSTAPE)
COPYL(DUMMY,SYSTAPE,DUM)
7/8/9
6/7/8/9
```

COPYL will list all the routines on SYSTAPE in order as none exist on the file DUMMY.

COPYL Messages

Listing Messages: COPYL DONE

COPYL DID NOT FIND xxxxxxx.
ILLEGAL COPYL PARAMETER

Display Message: UPDATING xxxxxxx.

## 10.2 FILE MANIPULATION

REWIND FILE

```
REWIND(file 1...,file n)
```

All files specified are rewound.

UNLOAD FILE

```
UNLOAD(file 1 ...,file n)
```

All files specified are rewound and unloaded. This function issues a CLOSE, UNLOAD and releases the file assignment to the control point. The UNLOAD macro only rewinds and unloads the files. The UNLOAD control card, however, is similar to RETURN.

SKIP OPERATIONS

SKIPF (lfn, n, lev, m)

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.

SKIPB(lfn, n, lev, m)

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.

| | |
|---|---|
| lfn | Logical file name (1-7 digits or letters); must begin with a letter. |
| n | Number of logical records or record groups to be skipped, maximum value is $777776_8$. n is a decimal number. A value equivalent to $777777_8$ will be treated as a no-operation for SKIPF and as a rewind for SKIPB. |
| lev | Logical records are skipped until n end-of-records with level numbers greater than or equal to the requested level is reached; the file is positioned immediately following (for SKIPF) or preceding (for SKIPB) the last record. lev is octal. |
| m | B for binary files, or C for coded files. |

The control card SKIPB. is interpreted as SKIPB(FILE,1,0,B). The control card SKIPF. is interpreted as SKIPF(FILE,1,0,B).

BACKSPACE LOGICAL RECORD

```
BKSP(file1,n)
```

Multiple logical records are backspaced as specified by the decimal n. Backspacing terminates if it results in a rewound file.

COMBINE

```
COMBINE(f1,f2,n)
```

For this operation, n(decimal) logical records are read from file f1 and written as one logical record (level 0) onto file f2. The file is not positioned prior to initiating this operation. If the files f1 and f2 have not been previously defined by REQUEST cards, they will be assumed to be on mass storage.

## 10.3 OCTAL CORRECTION ROUTINE

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.

## 10.4 REQUEST FIELD LENGTH

With the RFL card, the user can request a different field length during job execution; nfl is the new octal field length.

```
RFL,nfl.
```

RFL should be employed to obtain optimal usage central memory. For example, a FORTRAN program may require 45000 words of memory to compile, but only 5000 to execute. RFL should be used as follows:

Example:

```
JOB,T300,CM45000,P7.
RUN(S)
RFL,5000.
LGO. (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, RA to RA+100, 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 dumped also if x is equal to y and non-zero.

DMP (4xxxxx,4yyyyy) — Produces absolute core dumps. xxxxx defines the lower bound, yyyyy defines the upper bound of absolute core locations.

## 10.6 DUMP EXTENDED CORE STORAGE

The control card for this central program has the following calling sequence.

```
DMPECS(x,y,f,lfn)
```

The program dumps ECS from location x' to y'. x' is the closest multiple of 10B less than or equal to x, and y' is the (closest multiple of 10B greater than y)-1.

f selects the print format.

f = 0 or 1 4 words in octal and in display code per line.

f = 2 2 words in octal parcels and in display code per line.

f = 3 2 words in octal bytes and in display code per line.

f = 4 2 words in octal and in display code per line.

lfn Specifies the dump file; if absent or zero, file OUTPUT is assumed.

## 10.7 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 control card format is:

```
COMPARE(f1,f2,n,l,e,r)
```

f1,f2 Files to be compared

n Number (decimal) of records in f1 to be compared to f2

l End-of-record level number (octal)

e Number (decimal) of non-comparison words to be written to the OUTPUT file

r Number (decimal) of counted records to be processed during the comparison. Included in non-comparison record OUTPUT file if e parameter is given.

Comparison begins at the current position of each file and continues until the number of ends-of-records of the level specified or a higher level has been passed over. If all pairs of records are identical, the dayfile message is GOOD COMPARE; otherwise, it is BAD COMPARE. Discrepancies listed on file OUTPUT depend on parameters on the COMPARE card. Examples follow:

```
COMPARE(RED, BLUE)
```

Compares next record on file RED with next record on file BLUE.

```
COMPARE(RED, BLUE, 6)
```

Compares next six records regardless of level of 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)
```

Compares two files from their current positions up to and including the third following end-of-record with level number of 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.

Mass storage records can be read in either mode; the above strategy imposes no difficulty if a tape file is being compared with a mass storage file, as long as the tape file is named first on the COMPARE card. When tapes are compared, all label information will be ignored.

## 10.8 AUTOMATIC PROGRAM SEQUENCER (APR)

The Automatic Program Sequencer (or sequencer) allows jobs to be rerun at regular intervals and supplies necessary information to the jobs. Jobs are entered under the sequencer with control cards and, at the completion of the execution, are saved by the sequencer to be executed again at some future time. The control cards place the jobs under the sequencer, supply the interval or execution frequency, and provide the sequencer with certain utility functions. The operator can control the sequencer with console entries. The sequencer is a peripheral processor program (APR) and a table in central memory resident (T.SEQ).

The CMR table T.SEQ is used by the sequencer program APR as a working storage area. It contains two types of entries: the first is for the APR program; succeeding entries are for jobs running automatically under the sequencer. Entries are numbered 00-nn (octal) where nn is the maximum number of jobs allowed under the sequencer, equal to one less than the length (L.SEQ) of T.SEQ. In the released system, L.SEQ = 7 and the maximum value of nn is 6.

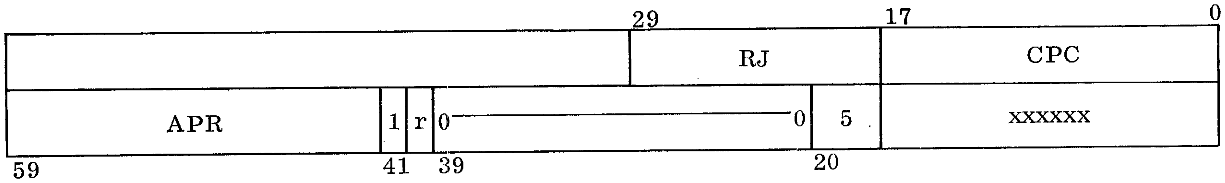
### 10.8.1 CALLS FOR APR

Calls to APR are processed by LOADER; two calls to APR are made as control cards in a job deck:

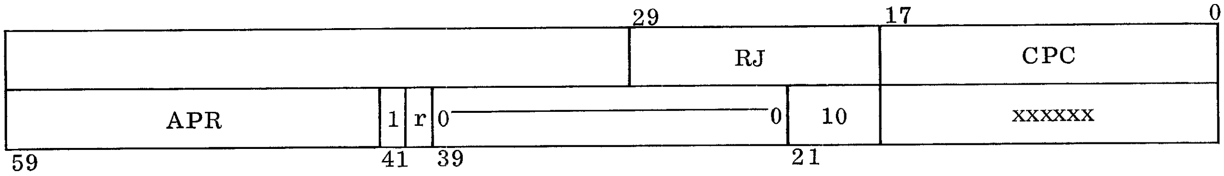
APR (1,xxxxnn) — nn is the T. SEQ ordinal to which this job is assigned; execution frequency is xxxx octal minutes. nn must be two octal digits, 00<nn< L. SEQ; xxxx may be 0000-7777$_8$. Example: a job calling APR (1,3301) is placed under the sequencer as job number 1, to be run every 33$_8$ minutes. If a job is already in the table, xxxx is ignored..

APR (11,nn) — Suppresses separator pages, output, and dayfile of calling job. This affects only the control point and has no effect on the sequencer. Usually for identification, nn is set to the T. SEQ ordinal used to place the job under sequencer control.

The following two calls to APR may be made by CPU programs by calling CPC:



Reads real-time clock and places it in byte 4 of address xxxxxx.



Places CMRA, ECRA, and ECFL in location xxxxxx. This word will contain the values: byte 0 = 0, byte 1 = 0, byte 2 = CMRA/100$_8$, byte 3 = ECRA/1000$_8$, byte 4 = ECFL/1000$_8$.

In either call, if xxxxxx is out of range, the control point is terminated and this message is issued: ILLEGAL ADDRESS REQUEST TO APR.

### 10.8.2 CONSOLE ENTRIES

The operator may control the sequencer by typing in SEQ,command. The commands are described in the SCOPE Operator's Guide (Pub. No. 60179600).

### 10.8.3 SAMPLE JOB STRUCTURE

A job to be sequenced every $35_8$ minutes would have the following structure:

JOB Card

| | |
|---|---|
| APR(1,003502) | Save job for future execution ($35_8$ min. interval). Will be job number 2 in T.SEQ. |
| ⋮ } | Any job deck without a JOB card and 6/7/8/9 card. |
| 6/7/8/9 | |

# DEBUGGING AIDS 11

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.

Instructions that are program modified or not executable may not be traced. This restriction applies to IA, LA (and all related formats) and all words between IA and LA.

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, A return jump which does not return to L+1 cannot be traced.

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. All parameters, except frequency, must appear. TRACE cards must be loaded contiguously.

| Parameter | Description |
|---|---|
| Range Identification:<br>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: | e is an entry point name. |
| IA=e or e+n | c is a labeled common block name. |
| IA1=e-n | n is an octal integer ≤ 777777. |
| IAC=c or c+n | |
| IAC1=c-n | |
| Last address: | The tracing range includes all instructions from IA to LA (LA must be greater than IA). |
| LA=e or e+n | |
| LA1=e-n | |
| LAC=c or c+n | |
| LAC1=c-n | |

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:<br>F1=n<br>F2=n<br>F3=n | n is an octal integer; it must not be 0. If parameter is not specified, 1 is assumed.<br>F1 Tracing begins F1st time IA is encountered.<br>F3 Thereafter, tracing takes place every F3rd time IA is encountered.<br>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:<br>TR=P,An,Bn, or Xn | n is the register number 0-7.<br>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:<br>TM=m,$k_1$,$k_2$,...,$k_n$ | m is an octal mask. (5 or 10 digits)<br>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:<br>TL=e or e+n<br>TL1=e-n<br>TLC=c or c+n<br>TLC1=c-n<br>TLB=b | e is entry point name<br>c is labeled common block name<br>n is octal integer ≤777777<br>b is nth location in blank common<br>Each instruction making an A-register reference to the location is used as a trigger. |
| Register dump;<br>RD | 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. |

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<br>OL1=e-n,i<br>OLC=c,i or c+n,i<br>OLC1=c-n,i<br>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:<br>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.<br>r is a register designator: An, Bn, Xn<br>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.<br>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<br>LA=NEXT | range limits | |
| TR=P,RD<br>TR=X6 | trap triggers | Each time a jump occurs, a trap is set and the registers are dumped (RD). 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<br>LA=E | range limits | |
| TM=00700, 00600 | trap trigger | 00700=octal mask<br>00600=match key<br>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 are located in front of the loader tables). 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 in the SNAP tables. 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. The instruction at IA must not be modified during program execution (a subroutine entry point called by a return jump is modified).

## 11.2.1 SCOPE CONTROL CARD

The following SCOPE control card initiates SNAP:

SNAP, $p_1,p_2,\ldots,p_n$.

Parameters may appear in any order except as noted below. All SNAP cards must appear contiguously.

| Parameter | Description |
|---|---|
| SNAP identifier:<br>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:<br>IA=e,e+n, or a<br>IA1=e-n<br>IAC=c or c+n<br>IAC1=c-n | e is an entry point name<br>c is a labeled common block name<br>n is an octal integer<br>a is an absolute address (relative to RA) |
| First word address of area dumped:<br>FWA=e,e+n,n, or a<br>FWA1=e-n or n<br>FWAC=c, c+n, or n<br>FWAC1=c-n or n<br>FWAB=b | b is bth location in blank common; other symbols are as in IA.<br>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:<br>LWA<br>LWA1<br>LWAC<br>LWAC1<br>LWAB | Format is the same as for FWA.<br>LWA must be ≥ FWA. |
| Interval between words dumped:<br>INT=n | n is a positive octal integer; if not specified, 1 is assumed. For a D dump, n is doubled. |
| Dump format:<br>F=code | Designated by one or two of the following codes: One only of the characters may be:<br>O Octal dump<br>M Octal dump with mnemonic operation codes<br>I Integer dump<br>S Single precision floating point<br>F If exponent = 0, I format; otherwise, S format.<br>D Double precision floating point dump (two words)<br>C Display code dump |

| Parameter | Description |
|---|---|
| | The second character is optional; it may suffix or prefix any other designator.<br>R  Register dump.<br>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:<br>F1=n<br>F2=n<br>F3=n | n is an octal integer. It cannot be zero; 1 is assumed if n is not specified.<br>F1  Tracing begins the F1st time IA is encountered.<br>F3  Therafter, tracing takes place every F3rd time IA is encountered.<br>F2  Tracing stops the F2nd time IA is encountered. |
| User's entry point:<br>UR=p,r1,...rn | Optional parameter; must be last on the card.<br>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:<br>Alphanumeric string, 1-10 characters, terminated by a zero byte. If the string contains 9 or 10 characters, an extra word is required.<br>Decimal integer, converted to binary, stored right justified.<br>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)<br>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<br>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, RA to RA+100, and P-77 to 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. If xxxxxx and yyyyyy are equal, DMP (1,1), the control point will be dumped.

DMP output will be suppressed when preset core is encountered. The last preset location will be marked by the symbol > in the output.

### 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 by inserting a DEBUG control cards previous to the DMP control card:

| 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 absolute 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)          TRACE and SNAP cards appear immediately
                        before program call card that initiates the
LGO.                    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 and SNAP cards appear immediately
 .                    before program call card that initiates the
 .                    run.
 .
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
```

Normal execution with labeled dump if job aborts:

```
job card
DEBUG.                   DEBUG card remains in force throughout
COPYBR(INPUT, LGO)       the job.
COMPASS.
LGO.
EXIT.
DMP(6000)             Dumps locations occupied by program PA.
DMP(PA, PA)           PA must have been loaded for this card to be
                      accepted.
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
INPUT.                   that load.  (SNAP and TRACE cards do not
DMP(1000,2000)           signal the end of the current load.)
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)
INPUT.
DMP(40000)
REWIND(SNACE)
COPYCF(SNACE, OUTPUT)
7
 8
  9
(Segmentation loader directive cards)
(Binary decks)
7
 8
  9
6
 7
  8
   9
```

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.

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)
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
```

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.

# CUSTOMER ENGINEERING (CE) DIAGNOSTICS 12

Customer Engineering (CE) diagnostic programs available under SCOPE 3 consist of:

Four CPU tests (ALS, CT3, CU1, and FST)

Two central memory tests (CM6 and MY1)

Six peripheral equipment tests (CP1, CR1, DT2, LPT, LP1, and MTT)

One ECS test (EC2)

The System Maintenance Monitor (SMM) tape contains equivalent tests with the same mnemonics. SMM is the system used by customer engineers to run diagnostic programs during preventive maintenance periods. These tests are fully described in the System Maintenance Monitor Reference Manual (Pub. No. 60160600).

## 12.1 MODES OF OPERATION

The CPU tests and memory tests (except EC2) can be run in both SMM and SCOPE modes. The peripheral equipment tests and memory test EC2 can be run only in SMM mode.

### SMM Mode

Executing a test under SCOPE in SMM mode is like executing under SMM, except that the test has been modified slightly to interface with SCOPE.

For CPU programs, the ORG card was changed from ORG 1 to ORG 101B. Therefore, a SCOPE listing of a test will be $100_8$ locations higher than a listing from the SMM tape.

For PPU programs, a call to the common deck PSSYS was substituted in place of a call to the common deck PS. PSSYS enables a PPU program to interface with SCOPE; PS allows interface with SMM.

A CE test is called to execute in SMM mode by specifying the test mnemonic under DIS or with a program call card in a control card deck, for example:

```
1
ALS.
```

SCOPE Mode

SCOPE mode was developed primarily to allow CE diagnostics to be run under the Automatic Program Sequencer (APR). Sample jobs appear in section 12.4.

A CE test is called to execute in SCOPE mode by specifying the test mnemonic followed by (SEQ). This call may be made under DIS or with a program call card in a control card deck, for example:

```
1
ALS(SEQ)
```

SCOPE Mode and SMM Mode Differences

The modes of execution differ in three respects only:

| | SCOPE Mode | SMM Mode |
|---|---|---|
| Execution Time | Tests execute for a finite length of time (usually 10-15 seconds) and then end. | Tests execute until a time limit abort or an operator DROP occurs at the control point. |
| Dayfile Messages | Tests send error message to dayfile and pauses for operator action if a hardware malfunction is detected. | Tests merely halt the central processor on a program stop instruction when a hardware error is discovered |
| Random Number Requests | ALS, FST, and CT3 tests make calls to the Automatic Program Sequencer (APR) for a random number to be used as a starter for a random instruction generator. | No random number requests are made. |

## 12.2 CPU AND MEMORY TESTS

The following test descriptions summarize the purpose of each test, and the accompanying chart gives the test calls, field lengths required, execution times, and error messages. More detailed descriptions are in the System Maintenance Monitor Reference Manual (Pub. No. 60160600).

The CPU and memory tests were added to SCOPE primarily for use with the Automatic Program Sequencer (APR). The CM error dumps shown in the chart refer to usage with APR.

Each test assumes that when it is called in SCOPE mode, the program call is part of a JOB card deck. When the test aborts after an error is detected, it assumes that the deck contains an EXIT card followed by DMP cards to dump the field length of the job.

The four CPU tests also assume that MODE,0. cards appear in the job deck. These cards are necessary to suppress the hardware exit mode, because the CPU tests use illegal operands to check for hardware end cases.

### ALS - 6X00 Random Instruction Test

ALS generates random instructions. It tests the stack registers and the scoreboard's ability to handle instructions issued at a rate faster than possible when instructions are not issued from the stack.

Errors are detected by executing an instruction sequence twice with the same initial register contents. The first pass terminates with a 04 jump instruction back to the beginning to re-execute all instructions from within the stack. The second pass terminates with a 02 jump instruction back to the beginning to prevent instructions from being executed from the stack registers. The answers are compared and an error message occurs if they do not agree.

ALS is intended for the 6600 computer. The 6400 computer can run the test but cannot execute an in-stack instruction loop.

| Program | Test Call<br>SMM Mode | <br>SCOPE Mode | Field Length Req.<br>(Octal Locations) | Execution Time<br>SMM Mode | <br>SCOPE Mode | Error Messages (SCOPE Mode)† |
|---|---|---|---|---|---|---|
| ALS | ALS. | ALS(SEQ) | 1500 | Indefinite | $4000_8$ passes thru its instruction loop ($20_8$ seconds on the 6400) | SEQUENCER DIAGNOSTIC/ALS/FAILED.<br>PASS CNT. = XXXXXX<br>RDN. NO. = YYYY<br>TYPE n.GO. SEE LINE PRINTER FOR CM ERROR DUMP. |
| FST | FST. | FST(SEQ) | 1200 | Indefinite | $5000_8$ passes thru its instruction loop ($20_8$ seconds on the 6400) | SEQUENCER DIAGNOSTIC/FST/FAILED.<br>PASS CNT. = XXXXXX<br>RDN. NO. = YYYY<br>TYPE n.GO. SEE LINE PRINTER FOR CM ERROR DUMP. |
| CT3 | CT3. | CT3(SEQ) | 4100 | Indefinite | $4000_8$ passes thru its instruction loop ($20_8$ seconds on the 6400) | SEQUENCER DIAGNOSTIC/CT3/FAILED.<br>ERROR PASS XXXXXX.<br>TYPE n.GO. SEE LINE PRINTER FOR CM ERROR DUMP. |
| CU1 | CU1. | CU1(SEQ) | 7000 | Indefinite | One pass thru its instruction loop ($40_8$ seconds on the 6400) | None. |
| CM6 | CM6. | CM6(SEQ) | Any value greater than 10,000. | Indefinite | $40,000_8$ passes; each checks $100_8$ pairs of addresses in each memory bank. Requires $15_8$ seconds on the 6400 with field length ≥ 20,000. Smaller field length increases execution time. | SEQUENCER DIAGNOSTIC/CM6/FAILED.<br>ERROR ADDRESS IN LOC. 342<br>ERROR DATA IN LOC. 343<br>TYPE n.GO. SEE LINE PRINTER FOR CM ERROR DUMP. |
| MY1 | MY1. | MY1(SEQ) | Any value greater than 1000. | Indefinite | $10_8$ passes thru its instruction loop ($10_8$ seconds on a 6400 with a field length of $30000_8$). | SEQUENCER DIAGNOSTIC/MY1/FAILED.<br>TYPE n.GO. SEE LINE PRINTER FOR ERROR DUMP. |
| EC2 | | EC2. | 55,000. | | $20_8$ seconds for a 500,000 word ECS field length. | SEQUENCER DIAGNOSTIC/EC2/FAILED.<br>TYPE n.GO. SEE LINE PRINTER FOR ERROR OUTPUT. |

† To call attention to the error, the final message remains at the control point and the job goes into recall until the operator types n.GO.

## FST - 6X00 Random Instruction Test

FST generates a set of $10_8$ random numbers, removes all jump instructions from this set and runs it as a subroutine. Passes are inserted in the last parcel in place of 30-bit instructions. All writes and reads are restricted to specific areas.

To check results, a slow loop is also generated with the same instructions; but it contains only one instruction for every two words of passes.

The B and X registers are loaded with random numbers and the A registers are set to known values before each pass. The slow loop is run first, and the results of the registers are stored, then the fast loop is run and the results compared. If the results compare, the fast loop is run and compared 31 more times. If no error occurs, the fast loop is run and compared 32 times for each set of random numbers before a new set is generated.

When an error occurs, the loops are shortened by one 60-bit word and the test rerun. If it fails again the loops are shortened again and the test is rerun until it does not fail. At this point, the last word removed is replaced; the pass count is entered in the dayfile, and the program halts.

## CT3 - 6X00 Random Instruction Test

CT3 uses a central processor simulator routine. It generates a set of random instructions and forms a random instruction loop. Branch and memory reference instructions are restricted to specific areas. The random loop is executed and the result registers saved. The simulator routine is then used to execute the random instruction loop, and the result registers are compared for differences.

The CT3 optional parameters described in the System Maintenance Monitor Reference Manual are not selectable in the SCOPE version of CT3. These parameters are automatically set for:

Central simulator

Six words in random loop

No stop on error

In-stack processing

No optimization

Try each set of random instructions twice

CU1 - 6X00 Command Test

CU1 tests central processor control hardware, central processor functional units, etc. Test of the control hardware checks the read flag settings and the unit reservations. Tests of the functional unit hardware check the arithmetic operations performed in the functional unit for a number of fixed operands.

CU1 needs a basic field length of $5200_8$; however, the last section of the test is for the branch unit and utilizes all available field length assigned to the control point.

CM6 - 6X00 Central Memory Test

CM6 checks the capability of a memory stack to switch drive lines between two addresses in the same memory bank. The first address is read many times in an in-stack instruction loop; then the read is executed immediately on the second address which contains all ones.

The test is more effective on the 6600 computer, as the 6400 computer cannot drive memory as fast because it cannot execute an in-stack loop.

Test results are not as meaningful if a storage move occurs while the test is executing.

MY1 - 6X00 Central Memory Test

MY1 checks central memory by setting each location in the field length to its relative address; it then executes five reads for each location. The data read back is held in X1 through X5 and is matched against X0, the current test address. The test accumulates and holds all error bits in X7, stores the error accumulations into memory, and rereads to check for zero. It also checks X7 for zero prior to storing.

At the end of one sweep of memory, the test will use the complement of the relative address and then repeat the check.

The test results are not as meaningful if a storage move occurs while the test is executing.

EC2 - 663X Extended Core Storage Test

EC2 tests ECS by writing data patterns to the assigned field length of ECS and then reading them back. A check is made for data differences. ECS field length can be set to any value; the program tests only that portion of ECS assigned to the control point. ECS is accessed in blocks of $10,000_8$ words.

EC2 is written in FORTRAN Extended with a COMPASS subroutine. The test makes calls to the Automatic Program Sequencer (APR). The following sense switches are used to indicate the number of ECS banks present in the system:

| Banks of ECS | Sense Switches On |
|---|---|
| 1 | None |
| 2 | 1 |
| 4 | 2 |
| 8 | 3 |
| 16 | 1, 2, and 3 |

## 12.3 PERIPHERAL EQUIPMENT TESTS

The following descriptions are for tests that run in the PPU. They have only one mode of operation and are not intended to be run with the Automatic Program Sequencer (APR) for automatic job execution. These tests, however, can be used with APR with an execution frequency of 000. (See sample job 10.)

These tests have the following optional sense switch settings:

| | Sense Switches On | | | | | |
|---|---|---|---|---|---|---|
| | MTT | DT2 | LPT | LP1 | CP1 | CR1 |
| Stop on Errors | 1 | 1 | 1 | 1 | 1 | 1 |
| Stop at End of Section | 2 | 2 | 2 | 2 | 2 | |
| Repeat Subcondition | 3 | 3 | 3 | | 5 & 6 | |
| Repeat Test | 4 | 4 | 4 | 4 | 4 | |
| Repeat Section | 5 | 5 | 5 | 5 | 5 | |
| Repeat Condition | 6 | 6 | 6 | 6 | 6 | |
| Stop at End of Test | | | | 3 | 3 | |
| Stop After Next Title Card | | | | | | 2 |
| Stop After Input Tray Empty | | | | | | 3 |
| Restart Test After Input Tray Empty | | | | | | 4 |
| Repeat Last I/O Operation | | | | | | 6 |

### MTT - 60X Magnetic Tape Test

MTT checks the 60X tape unit, a 3000 series tape controller, 6681 channel converter, and 6000 series data channel. Two calls may be used.

MTT.

All sections of the test are run on the first channel found in the equipment status table entry for the assigned tape unit.

MTT(cc,xxxxxx)

cc designates channel and xxxxxx are octal numbers; each bit represents a section of the test. If channel is entered, xxxxxx must be specified.

Examples:

| | |
|---|---|
| MTT(13,000001) | Run section 1 only; tape is on channel 13 |
| MTT(000001) | Run section 1 only; use first channel in EST |
| MTT(000277) | Run sections 1 through 6 and 8 |

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST MT. | Assign device type MT to the control point. |
| SET PARAMS. | Set sense switches for desired conditions. |
| ASSIGN EXTRA UNITS | If desired, assign additional units to be tested. To begin testing, enter n.GO on the keyboard. |

Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

### DT2 - 6638 Disk File Test

This test can be run only on a non-system disk; no other device on the same channel as the disk can be accessed while DT2 is executing. DT2 can be called by control cards or manually under DIS by its mnemonic DT2:

DT2(n) where n may have one of the following values:

| | |
|---|---|
| 1 = | Run section 1 |
| 2 = | Run section 2 |
| 4 = | Run section 3 |
| 10 = | Programmable read/write section |

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST xx. | Assign device type xx to the control point. A 3000 entry for hardware type xx must have been entered already in the Equipment Status Table. |
| SET PARAMS. | Set sense switches for desired conditions. To begin testing, enter n.GO on keyboard. |

Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

### LPT - 501 Line Printer Test

LPT checks the 501 line printer, 3659 or 3256 controller, 6681 channel converter, and 6000 series data channel. LPT has two alternate calls:

LPT.

Run all sections of the test.

LPT(xxxxxx)

xxxxxx are octal numbers; each bit represents a section of the test.

Examples:

| | |
|---|---|
| LPT(000001) | Run section 1 only. |
| LPT(000277) | Run sections 1 thru 6 and 8 |

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST LP. | Assign device type LP to the control point. |
| SET PARAMS. | Set sense switches for desired conditions. To begin testing, enter n.GO on the keyboard. |

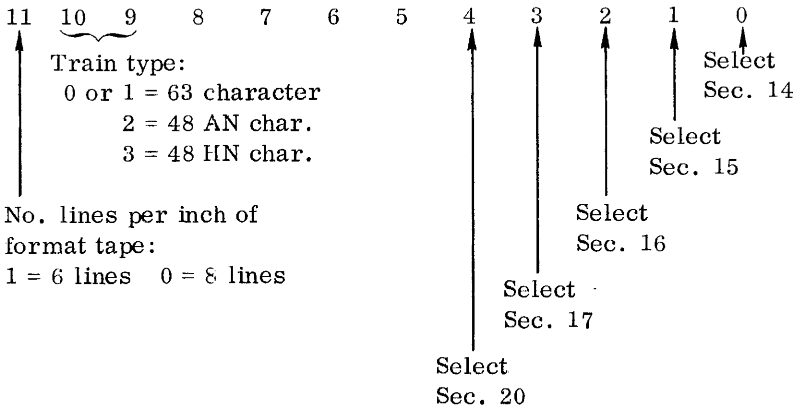
Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

## LP1 - 512 Line Printer Test

LP1 tests the 512 line printer, 3555 controller, 6681 channel converter, and 6000 series data channel. The test is called by its mnemonic LP1. If parameters are to be changed, enter:

LP1 (xxxx, yyyy)

xxxx are octal numbers with the following bit values:



yyyy are octal numbers with the following bit values:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| y bits: | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| select sections: | 13 | 12 | 11 | 10 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

Examples:

| | |
|---|---|
| LP1(4001,0002) | Selects 6 lines and sections 14 and 1. |
| LP1(0004,0030) | Selects sections 3, 4, and 16. |
| LP1(2000,7776) | Selects 48 AN character and sections 1-13. |
| LP1. | Selects all sections. |

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST LQ. | Assign device type LQ to the control point. |
| SET PARAMS. | Set sense switches for desired conditions. To begin testing, enter n.GO on the keyboard. |

Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

## CP1 - 415 Card Punch Test

CP1 tests the 415 card punch, 3446 or 3644 controller, 6681 channel converter, and 6000 series data channel. The test is called by its mnemonic CP1. If parameters are to be changed, enter:

CP1(xxxx)

xxxx are octal numbers with the following bit values:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x bits: | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| select sections: | | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

Examples:

| | |
|---|---|
| CP1(0002) | Selects section 1. |
| CP1(0306) | Selects sections 1, 2, 6, and 7. |
| CP1. | Selects all sections. |

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST CP. | Assign device type CP to the control point. |
| SET PARAMS. | Set sense switches for desired conditions. To begin testing, enter n.GO on the keyboard. |

Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

CR1 - 405 Card Reader Test

CR1 tests the 405 card reader, 3649 or 3447 controller, 6681 channel converter, and 6000 series data channel. The test is called by its mnemonic CR1. No section select parameters are used because all cards in the card reader are checked regardless of order. The card deck punched by CP1 should be put into the card reader assigned to the test.

After test execution begins, the following messages appear at the control point:

| Message | Operator Action |
|---|---|
| REQUEST CR. | Assign device type CR to the control point. |
| SET PARAMS. | Set sense switches for desired conditions. To begin testing, enter n.GO on the keyboard. |

Subsequent error messages at the control point are described in the Systems Maintenance Monitor Reference Manual. The operator enters n.GO to continue the test after an error stop.

## 12.4 SAMPLE JOBS

The call APR(7,xxxxnn) shown in the sample jobs is not described in section 10.8 (APR) because it refers only to diagnostic programs.

7 is a directive to APR to set the diagnostic flag bits.

nn is the job number.

xxxx is a set of flag bits, one bit for each CPU diagnostic program in the CE test. Bit values for xxxx are:

| Value of xxxx | Diagnostic |
|---|---|
| 0001 | CT3 |
| 0002 | MY1 |
| 0004 | CM6 |
| 0010 | CU1 |
| 0020 | ALS |
| 0040 | FST |
| 0100 | EC2 |

For example, to set CT3, MY1, and CM6: xxxx = 0007.
To set CT3 and FST: xxxx = 0041.

These bits are set in the sequencer table in CMR but do not affect the operation of the sequencer. The bits are used when the keyboard entry SEQ, LIST, nn. is made. This entry causes the names of the diagnostics assigned to job nn to be listed in the dayfile for reference.

### 12.4.1 SAMPLE JOB 1

Execute the diagnostic ALS every $10_8$ minutes.

| | |
|---|---|
| MAINT, P17, T20, CM1500. | ALS field length = 1500.<br>Time = 20 seconds. |
| APR(1,001001) | Saves job as sequencer job 1 to be executed every 10 minutes. |
| MODE,0. | ALS uses the illegal operands. |
| ALS(SEQ) | Calls sequencer version of ALS. |
| 6/7/8/9 | |

### 12.4.2 SAMPLE JOB 2

Save and execute ALS every 10 minutes; set up the diagnostic flag bit for ALS.

| | |
|---|---|
| MAINT, P17, T20, CM1500.<br>APR(1,001001)<br>MODE,0. | See sample job 1. |

| | |
|---|---|
| APR(7,002001) | Diagnostic in sequencer job 1 is ALS. |
| ALS(SEQ) | See sample job 1. |
| 6/7/8/9 | After job is completed, console entry:<br>SEQ,LIST,01. will produce dayfile message:<br>DIAG.SEQ.JOB 01 CONTAINS ALS |

## 12.4.3 SAMPLE JOB 3

Save and execute ALS every 10 minutes. If ALS fails, dump the field length.

| | |
|---|---|
| MAINT,P17,T20,CM1500. | See sample job 1. |
| APR(1,001001) | Save job, same as sample 1; executed if ALS does not fail. |
| MODE,0. | See sample job 1. |
| ALS(SEQ) | See sample job 1. |
| EXIT. | If ALS did not fail, EXIT ends job. Otherwise, cards after EXIT will be executed. |
| DMP,1500. | Gets failure data by dumping actual field length of ALS. Since ALS failed, it aborts to EXIT and system provided exchange jump package. |
| 6/7/8/9 | |

## 12.4.4 SAMPLE JOB 4

Save and execute ALS every 10 minutes. Set the flag bit for ALS. If ALS fails, dump the field length.

| | |
|---|---|
| MAINT,P17,T20,CM1500. | |
| APR(1,001001) | |
| MODE,0. | See sample job 3. |
| APR(7,002001) | Sets flag bit for ALS. After job completion, console entry SEQ,LIST,01. will produce dayfile message:<br>DIAG.SEQ.JOB 01 CONTAINS ALS |

| | |
|---|---|
| ALS(SEQ) | See sample job 3. |
| EXIT. | |
| DMP,1500. | |
| 6/7/8/9 | |

### 12.4.5 SAMPLE JOB 5

Same as sample job 4, except suppress the output if ALS executes successfully.

| | |
|---|---|
| MAINT,P17,T20,CM1500. | See sample job 4. |
| APR(1,001001) | |
| MODE,0. | |
| APR(7,002001) | |
| ALS(SEQ) | |
| APR(11,01) | Suppresses output, separators, and dayfile of job MAINT. |
| EXIT. | See sample job 4. |
| DMP,1500. | |
| 6/7/8/9 | |

### 12.4.6 SAMPLE JOB 6

Run ALS and CT3 every 10 minutes as sequencer job 3. Set flag bits for ALS and CT3. Suppress output file and dump field length of failing diagnostic.

| | |
|---|---|
| CHEK,P17,T40,CM5000. | Time limit = 40 (20 seconds for each diagnostic); field length = 5000 (for largest diagnostic, CT3). |
| APR(1,001003) | Saves CHEK as sequencer job 3; runs every 10 minutes. |
| MODE,0. | See sample job 1. |
| APR(7,002103) | Sets flag bits: ALS(0020) + CT3(0001) = 0021. |
| ALS(SEQ) | Calls sequencer version of ALS and CT3. |
| CT3(SEQ) | |

| | |
|---|---|
| APR(11,03) | Suppresses output. |
| EXIT. | See sample job 3. |
| DMP,4000. | Dumps field length (CT3 = 5000). If ALS fails, CT3 will not be executed. |
| 6/7/8/9 | |

EC2 Jobs

Because EC2 is unique, three sample EC2 jobs are shown. EC2 should be run alone. The time limit is 30 seconds, and 55,000 words are needed for central memory field length. ECS field length is specified on the JOB card in 1000$_8$-word blocks; EC50 = 50,000 words of ECS. The SWITCH cards specify the total number of ECS banks, as follows:

No SWITCH card = 1 bank

SWITCH1 card = 2 banks

SWITCH2 card = 4 banks

SWITCH3 card = 8 banks

$\left.\begin{array}{l}\text{SWITCH1}\\ \text{SWITCH2}\\ \text{SWITCH3}\end{array}\right\}$ cards = 16 banks

SWITCH cards do not specify which banks of ECS to test; they merely indicate the number of banks in the ECS unit.

The SWITCHn card turns on pseudo sense switch n at the control point.

### 12.4.7 SAMPLE JOB 7

Use EC2 to test a 1-bank ECS system. Check an ECS field length of 50,000. Run EC2 as sequencer job 7 every 30 minutes.

| | |
|---|---|
| CHEKECS,P17,T30,CM55000,EC50. | EC50 requests 50,000 words of ECS. |
| APR(1,003007) | Saves CHEKECS as sequencer job 7 to be run every 30 minutes. |
| APR(7,010007) | Sets EC2 flag bit. |
| EC2. | |
| APR(11,07) | Suppresses output file on successful execution. |
| 6/7/8/9 | EXIT and DMP cards not needed because EC2 has an error printout routine. The absence of a SWITCH card indicates only one bank of ECS is present. |

### 12.4.8 SAMPLE JOB 8

Use EC2 to test a 2-bank ECS system. Check an ECS field length of 100,000. Run EC2 as sequencer job 7 every 30 minutes.

```
CHECECS,P17,T30,CM55000,EC100.   Requests 100,000 words of ECS.
SWITCH1.                         Two banks of ECS present.
APR(1,003007)
APR(7,010007)
EC2.                             See sample job 7
APR(11,07)
6/7/8/9
```

### 12.4.9 SAMPLE JOB 9

Use EC2 to test a 16-bank ECS system. Check an ECS field length of 250,000. Run EC2 as sequencer job 7 every 30 minutes.

```
CHECECS, P17, T30, CM55000, EC250.   EC250 requests 250,000 words of ECS.
SWITCH1.
SWITCH2.                             16 banks of ECS present.
SWITCH3.
APR(1,003007)
APR(7,010007)
EC2.                                 See sample job 7
APR(11,07)
6/7/8/9
```

### 12.4.10 SAMPLE JOB 10

A peripheral test is placed under the sequencer with an interval of 0000. The test will not run unless a SEQ,RUN,nn. entry is made.

```
MAINT, P17, T1000, CM1000.    When MAINT is loaded, MTT will
APR(1,000001)                 request a tape. Enter n.DROP on
                              keyboard. MAINT will be saved as
                              sequencer job 1, interval 0000.
```

| | |
|---|---|
| MTT.<br>6/7/8/9 | Zero interval jobs are not run unless called. To run MTT, enter SEQ,RUN,01. MTT will be loaded. Proceed as under SMM. When completed, DROP control point. MAINT will be saved again under the sequencer.<br>Placing peripheral tests under the sequencer in this manner eliminates the need to use DIS. to call a peripheral test. |

## 12.5 SYSTEM ENGINEERING FILE DESCRIPTION

The System Engineering File records system hardware malfunctions detected by the SCOPE Operating System. Errors are recorded through entries from I/O drivers, and PP/CP program. The entry format, file construction and file maintenance are designed to facilitate utilization in current programs as well as future design efforts.

The file is dynamic from deadstart time; its FNT and FET entries are assembled in CMR. If a program detects a hardware malfunction, it makes an entry in the file by using one of the formats described in 12.5.1.

When calls are made to log an entry, the entry is placed in the engineering file buffer in CMR. The dayfile processing routine supplements the entry with time of day and jobname. The dayfile processing routines perform buffer maintenance and dump the contents to disk as necessary. An entry count is maintained (byte C.HEC of P.HEC in CMR).

An Analyzer (CEFAP) program examines the engineering file, collects the statistical data, and outputs the results in usable form for examination by Customer Engineering.

### 12.5.1 ERROR LOGGING PROCEDURE

Programs log anomalies in the System Engineering File similarly to logging messages in the DAYFILE.

#### Entries by PP Programs

Entries made by PP programs are placed in the PP message buffer.

A call is placed through R.MTR in the PP output register as follows:

| | | | | |
|---|---|---|---|---|
| 0001 | 0020 | lwa | ▨ | ▨ |

| | |
|---|---|
| 0001 | Process dayfile message |
| 0020 | Flag bit for DSD |
| lwa | LWA of entry |

Entries by CP Programs

Entries made by CP programs are constructed within their field length. A call to MSG is placed in RA+1 with the address of the message and a flag (bits 18-23 = 02) to MSG which indicates the message is to be entered in the engineering file. MSG assumes a six bit CM word message.

<table>
<tr><td>59</td><td>53</td><td>47</td><td></td><td></td><td>29</td><td>23</td><td></td><td>11</td><td>0</td></tr>
<tr><td>a</td><td>b</td><td colspan="3">c</td><td>d</td><td colspan="2">e</td><td colspan="2">▨</td></tr>
<tr><td>▨</td><td>f</td><td colspan="2">g</td><td colspan="2">h</td><td colspan="2">i</td><td colspan="2">j</td></tr>
</table>

Type 1 SCOPE System Tape I/O Driver

| | | |
|---|---|---|
| a | Error type | 000001 base 2 |
| b | Error description | See error descriptions |
| c | Program name | PPU program making entry |
| d | PPU number | PPU number of logging program |
| e | Connect code | Connect code of failing device |
| f | Channel number | I/O channel in use |
| g | Channel status | Last 6681 status returned |
| h | Equip status | Last Equip status returned |
| i | Channel function code | Last 6681 code issued |
| j | Equip function code | Last Equipment function code issued |

<table>
<tr><td>59</td><td>53</td><td>47</td><td></td><td></td><td>29</td><td>23</td><td></td><td>11</td><td>0</td></tr>
<tr><td>a</td><td>b</td><td colspan="3">c</td><td>d</td><td colspan="2">e</td><td colspan="2">▨</td></tr>
<tr><td>f</td><td>g</td><td colspan="2">h</td><td colspan="2">i</td><td colspan="2">j</td><td colspan="2">k</td></tr>
<tr><td>l</td><td>▨</td><td colspan="4">m</td><td colspan="4">n</td></tr>
</table>

Type 1 SCOPE System Mass Storage I/O Driver

| | | |
|---|---|---|
| a | Error type | 000001 base 2 |
| b | Error description | See error descriptions |
| c | Program name | PPU program making entry |
| d | PPU number | PPU number of logging program |
| e | Address of error | Not supplied |
| f | Logical unit number | EST number of failing device |
| g | Channel number | I/O channel in use |
| h | Channel status | Not supplied |
| i | Equip status | Not supplied |
| j | Channel function code | Not supplied |
| k | Equip function code | Not supplied |
| l | Retry count | Number of attempts to perform an operation |
| m | Last position | Actual position of disk upon failure |
| n | Current position | Not supplied |

Type 2 SCOPE System PPU Programs (non-I/O)

<table>
<tr><th>59</th><th>47</th><th></th><th>29</th><th>23</th><th>11 0</th></tr>
<tr><td>a</td><td colspan="2">b</td><td>c</td><td>d</td><td>e</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
</table>

| | |
|---|---|
| a | Error code (0200 base 8) |
| b | Program name |
| c | PPU number |
| d | Address of error |
| e | Not used |
| f | Display coded message |

Type 3 SCOPE System CPU Program

| 59 | 47 | 35 | 23 | 17 | 11 | 5   0 |
|---|---|---|---|---|---|---|
| a | b | b | b | | b | ▨ |
| ▨ | ▨ | ▨ | ▨ | c | c | |
| d | d | d | d | | d | |
| d | d | d | d | | d | |
| d | d | d | d | | d | |
| d | d | d | d | | d | |

| | |
|---|---|
| a | Error code |
| b | Program name, 7-characters |
| c | Address of error |
| d | Display coded message |

Type 4 C E Diagnostic PPU I/O Test (Not currently used)

| 59 | | 47 | | 29 | 23 | 11   0 |
|---|---|---|---|---|---|---|
| a | | b | | c | d | e |
| f | g | h | i | | j | k |
| l | ▨ | m | m | | n | n |
| o | | p | p | | p | p |

| | |
|---|---|
| a | Error code |
| b | Program name |
| c | PPU number |
| d | Address of error |
| e | Not used |
| f | Logical unit number |
| g | Channel number |
| h | Channel status |
| i | Equipment status |
| j | Channel function code |
| k | Equipment function code |
| l | Retry count |
| m | Last position |
| n | Current position |
| o | Operation code, last logical operation |
| p | Data, error data |

Type 5 C E Diagnostic Non-I/O PPU Test (Not currently used)

59 47 29 23 11 0

<table>
<tr><td>a</td><td colspan="2">b</td><td>c</td><td>d</td><td>e</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
</table>

| | |
|---|---|
| a | Error code |
| b | Program name |
| c | PPU number |
| d | Address of error |
| e | Not used |
| f | Display coded message |

Type 6 C E Diagnostic CPU Program

59 47 29 11 0

<table>
<tr><td>a</td><td colspan="2">b</td><td colspan="2">c</td><td>d</td></tr>
<tr><td>e</td><td>e</td><td colspan="2">e</td><td>e</td><td>e</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
<tr><td>f</td><td>f</td><td colspan="2">f</td><td>f</td><td>f</td></tr>
</table>

| | |
|---|---|
| a | Error code |
| b | Program name |
| c | Address of error |
| d | Pass count |
| e | Base random number |
| f | Display coded message |

Type 7 Remote Terminal (RESPOND, SENTRY, EXPORT/IMPORT, IOD) Programs

59 47 35 23 11 0

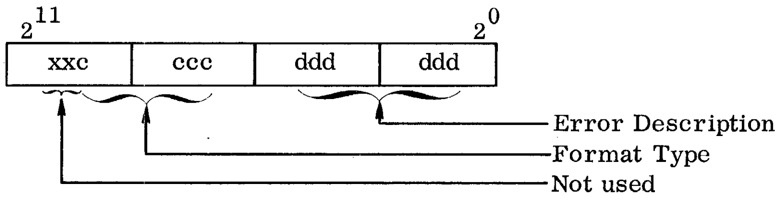
| | | | | |
|---|---|---|---|---|
| a | b | c | d | e |
| f | g | h | i | j |

| | | |
|---|---|---|
| a | Error code | 0700 base 8 |
| b | Line number | TTY line number |
| c | LRB | Line request byte |
| d | RBB | Read buffer byte |
| e | WBB | Write buffer byte |
| f | LSB | Line status byte |
| g | TSB | Terminal status byte |
| h | RSB | Reply status byte |
| i | PNB | Port Number byte |
| j | PLB | Party line byte |

Type 8-11 not defined

## 12.5.2 ERROR CODES

The following 12-bit error code is the first byte of every entry:



| | |
|---|---|
| xx | Not used |
| c | Format type used in entry |
| d | Error description of type error |

Error Code Definitions

| | |
|---|---|
| $2^{10}$, $2^{11}$ | Not used |
| $2^{6}$ - $2^{9}$ | Format types as follows: |

| | | |
|---|---|---|
| 00 | not used | |
| 01 | Type 1 | System I/O driver |
| 02 | Type 2 | System PPU program non-I/O |
| 03 | Type 3 | System CPU program |
| 04 | Type 4 | C E Diagnostic PPU I/O test |
| 05 | Type 5 | C E Diagnostic Non-I/O PPU test |

06 Type 6 C E Diagnostic CPU Program
07 Type 7 Control Message (RESPOND or EXPORT/IMPORT)
10 not used
11 not used
12 not used
13 not used
14 not used
15 not used
16 not used
17 not used

$2^0 - 2^5$ Error Descriptions for Format Type 1

00 not used
01 Channel active, should be inactive
02 Channel inactive, should be active
03 Channel full, should not be
04 Channel empty, should not be
05 Non-zero accumulator upon exit from an OAM instruction (lost data suspected)
06 Memory parity error during loading MMTC
07 Memory parity error during data flow in MMTC
10 Read parity error
11 Write parity error
12 Function reject
13 XMSN parity error
14 Compare error
15 Fail to feed
16 Lost data
17 RMS address error
20 RMS checkword error
21 Print error
22–27 not used

Error descriptions for Format types 2 - 11 are not defined.

## 12.6 SYSTEM ENGINEERING FILE ANALYZER (CEFAP)

CEFAP formats and prints raw data in the System Engineering File. The main program CEFAP is written in FORTRAN. A COMPASS PPU routine IEF performs utility functions for CEFAP. CEFAP is edited into the system library. CEFAP may then be called by job deck or console command under DIS. Once activated, CEFAP attaches the System Engineering File to its control point and reads the file and generates output. Output consists, basically, of a list of everything in the engineering file plus statistical tables plotting common error data against time. Also, a parameter may be entered on the CEFAP call to specify output sorting.

### 12.6.1 PARAMETERS

Three parameters may be entered on the CEFAP call in the order shown below:

CEFAP (p1, p2, p3)

S or SORT — CEFAP sorts data in the output list according to error type, PPU number, channel number, and equipment EST ordinal.

P or PURGE — CEFAP clears the error count in central memory and purges the engineering file after analysis. A new engineering file will be created by the system.

E or EXPAND — CEFAP prints every error entry in chronological order. Normally, repeated errors will be printed once with an indication as to how many times it occurred.

If CEFAP is dropped from a control point before normal completion after assigning the L display, a subsequent CEFAP run may give false or erroneous output.

### 12.6.2 INSTALLATION

The CEFAP FORTRAN deck and COMPASS PPU deck both reside on the PL7 tape. CEFAP may be added to SCOPE with the following job deck:

| Card No. | Content |
|---|---|
| 1 | Job Card (CM50000) |
| 2 | REQUEST OLDPL.        ASSIGN PL7 |
| 3 | UPDATE(Q) |
| 4 | COMPASS(I=COMPILE,S=SCPTEXT,B=IEF) |
| 5 | RUN(S,,,COMPILE,,,100000) |
| 6 | LOAD(LGO) |
| 7 | NOGO |
| 8 | REWIND(CEFAP) |
| 9 | REWIND(IEF) |
| 10 | EDITLIB. |
| 11 | EOR |
| 12 | *COMPILE,CEFAP,IEF. |
| 13 | EOR |
| 14 | READY(SYSTEM) |
| 15 | DELETE(CEFAP) |
| 16 | DELETE(IEF) |
| 17 | ADD(CEFAP, CEFAP) |
| 18 | ADD(IEF, IEF) |
| 19 | COMPLETE. |
| 20 | EOF |

## 12.7 IEF DESCRIPTION

IEF is called by the System Engineering File Analyzer Program (CEFAP). Upon entrance to the program, the PPU input register has the following format:

```
1105   06cc   xxxx   ffpp   pppp
```

| | |
|---|---|
| cc | Control point number |
| ff | Function code |
| xxxx | Function parameter (used only by function code 02) |
| pppppp | Relative address of replay register in CM |

IEF performs the following functions:

Function Code 00

1. Copies date from CMR to PPPPPP+1
2. Copies EST from CMR to PPPPPP+2
3. Dumps engineering file buffer (CERFILE) and creates a duplicate engineering file (FNT/FST local to control point
4. Sets reply register (pppppp) to:

```
0077   0000   aaaa   0000   1111
```

| | |
|---|---|
| 0077 | Function complete status |
| aaaa | Duplicate error file FNT address |
| 1111 | Length of CERFILE buffer |

Function Code 01 Purge CERFILE

Same as function code 00 except the FST for CERFILE is zeroed so that a new engineering file will be created by system.

Function Code 02

1. Removes the duplicate engineering file FNT/FST entry created by a function code 00 call. The FNT/FST address is given by xxxx in input register.

2. Sets reply register (pppppp) to:

```
0077   0000   0000   0000   0000
```

| | |
|---|---|
| 0077 | Function complete status |

## 12.8 OPERATION

If purge is not specified in the CEFAP call, IEF is called initially with function 00; and after the run it is called again with a function 02. This procedure allows CEFAP to read the engineering file with the duplicate FST, while the system continues to log anomalies. The function 02 call is necessary to prevent disk storage from being released at end of job; the System Engineering File remains intact. If purge is specified, IEF is called initially with function 01. The system then terminates error logging on the original engineering file (blanked FST) and creates a new one. On end of run, CEFAP does not recall IEF; thus disk storage associated with the original engineering file is released by IEJ at end of job; the error file is purged.

### IEF DAYFILE MESSAGES

The starred messages denote conditions which cause IEF to abort at control point.

*IEF -- REPLAY REG ADDRESS OUT OF RANGE

The pointer in the IEF call was beyond the control point FL. Call register bits $2^0 - 2^{17}$.

*IEF -- CERFILE NOT IN SYSTEM

CERFILE FNT/FST not present in file name table.

IEF -- WAITING FOR FNT SPACE

IEF is waiting for space in the file name table to place a duplicate CERFILE FNT.

IEF -- CERFILE PURGED

Placed in DAYFILE when function 01 is in process.
The CERFILE FST is cleared.

*IEF -- ILLEGAL FUNCTION CODE

Function code field in call was not recognized by IEF. Call register bits $2^{18} - 2^{23}$.

*IEF -- DUPLICATE CERFILE NOT PRESENT

On function 02 call, a duplicate CERFILE FNT/FST was not found in file name table.

*IEF -- NOT CALLED BY AN ABSOLUTE LIBRARY PROGRAM

On entry, IEF checks the control point area for a flag indicating the last program loaded was a system library program.

### 12.8.1 EXECUTING CEFAP

After running the job in section 12.6.2, CEFAP is ready for execution. CEFAP may be executed (called) in two ways:

```
x.DROP.cr        x=Control point number
x.XCEFAP.cr      x=Control point number
L=x.cr           x=Control point number
xxxxx.xcr        xxxxx.x=site location number
GOcr
```

Parameters pl, p2, or p3 may not be entered directly under a control point entry. 7.XCEFAP(S) will not work as SCOPE does not recognize the parentheses.

```
x.DIS.cr         x=Control point number
x.CEFAP(pl,p2,p3)cr
*                Depress asterisk KEY
L=x.cr           x=Control point number
xxxxx.xcr        xxxxx.x=site location number
```

In addition to the on-line diagnostics, SCOPE 3.2 PL7 tape has CEFAP with correction identifiers CFAP001 and CFAP002. The EXPAND parameter and the site location number features have been added via correction identifiers CFAP003 and CFAP004.

## EXAMPLE OUTPUT DEFINITIONS

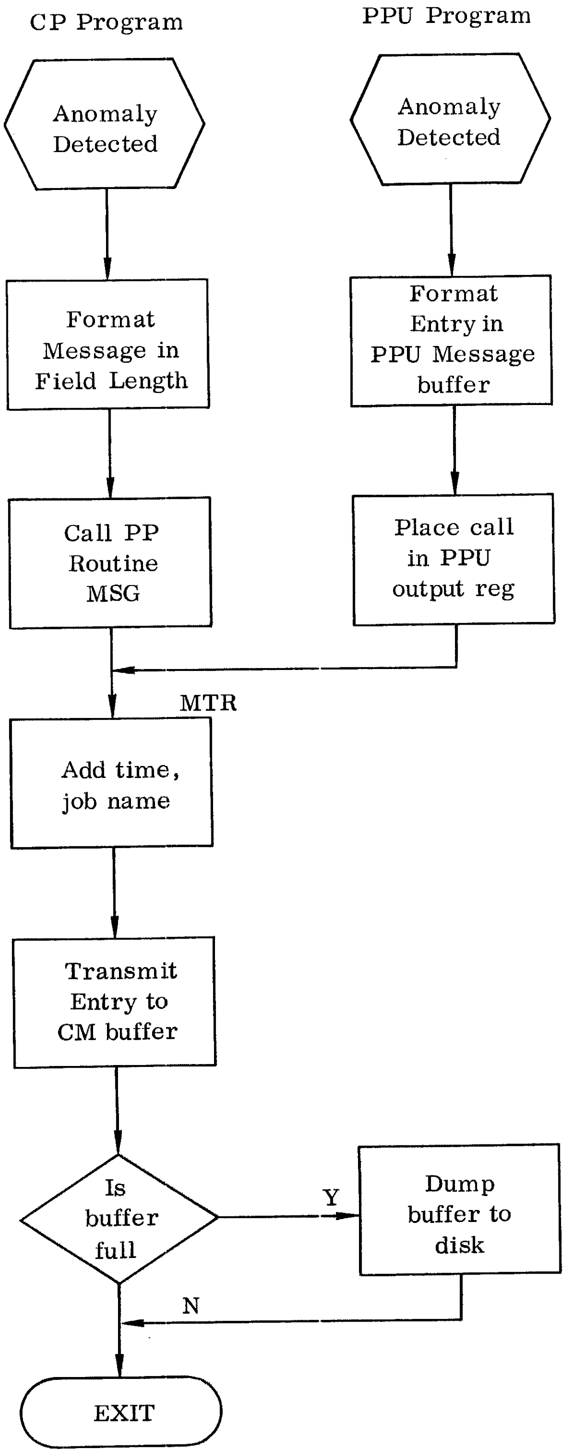
Line 1

| | |
|---|---|
| TIME=xx.xx.xx. | Error entry time. |
| JOB NAME=xxxxxxx | Name of job which had error. |
| PROG NAME=xxxxxxx | Name of program that issued error entry. |
| CC=xxxx | Connect code of offending device. |
| P=xxxxxx | Program address on error (issuing program). |
| ER CODE=xxxx | 12-bit error code. |
| Message | Error message furnished in entry or determined by error code. |

Line 2

| | |
|---|---|
| xxxx | Equipment mnemonic and EST ordinal of offending equipment. |
| PPxx | PPU number driver is running in. |
| CHxx | Channel number. |
| STAT C/E=xxxx/xxxx | Last channel (6681) and equipment status. |
| FUNC C/E=xxxx/xxxx | Last function issued to channel (6681) and equipment. |
| RETRYS=xx | Number of retrys before reporting error. |
| RMS LP/CP=xxxxxxxx/xxxxxxx | Last position and current position of RMS device. |

Flow Diagram of Error Logging



# PERMANENT FILES 13

A permanent file is a mass storage file cataloged by the system so that its location and identification are always known to the system. Permanent files cannot be destroyed accidentally during normal system operation (including deadstart) and they are protected by the system from unauthorized access according to privacy controls specified when they are created (section 13.2).

Any random or sequential file on a valid rotating mass storage device, which is not already permanent or common, can be made permanent at the option of the user regardless of mode or content. Unless the system is explicitly requested to catalog a file, it will not be made permanent.

The following functions are available under the permanent file system:

CATALOG an existing local mass-storage file, thereby making it a permanent file.

ATTACH a previously cataloged file to a control point.

EXTEND a currently attached file by making permanent an extension to it.

PURGE a currently attached file from the directory of permanent files.

These functions are available as control card commands and as run-time program calls (system macros). The CLOSE, UNLOAD macro or the RETURN control card allows a permanent file to be logically detached prior to end of job.

## 13.1 TERMS AND CONCEPTS DEFINED

Terms and concepts used throughout this chapter are defined as follows:

PERMANENT FILE

A mass storage file (of any standard mode, content, length) locatable by the system via a search of the system-maintained Permanent File Directory, and protected from unauthorized access and from accidental destruction in any normal running environment of any version of SCOPE that supports permanent files.

CATALOG

The function that enters a file's identification and location in the system-maintained Permanent File Directory. The CATALOG control card or system macro requests the system to catalog a file, thereby making it permanent.

SUBDIRECTORY

The directory of permanent files is divided into a number of subdirectories to expedite searching for files. Each permanent file has a subdirectory reference which the user may specify when requesting system action on that file.

REWRITE

Any action which modifies the content of an existing file including REWRITE, REWRITER, REWRITEF, or WRITIN macros.

WRITE

Enables normal writing at end-of-information of a file. When a permanent file is attached to a control point, any write function, except a rewrite, will cause data to be added at its end-of-information. Information added with write functions to an attached permanent file is considered temporary; it is released when the job is terminated unless EXTEND is used to make it a permanent extension.

CYCLE

Up to five files may be cataloged under one permanent file name and set of passwords; each is called a cycle. No restrictions are imposed on the content or size of any of these cycles; each is a unique file. Each cycle is identified by the combination of permanent file name and cycle number. Cycle numbers from 1 to $63_{10}$ are assigned by the creator of the file. Default value is 1 when cataloging. If cycle is not specified on the ATTACH card, it is assumed to be the largest cycle.

ATTACH

Causes a permanent file to be assigned to a control point. No user may access a permanent file until he attaches it to a control point, thereby establishing his right to read and/or write the file.

EXTEND

Makes permanent an appendage to a file. Any write results in an extension to a file; however, an extension will not be reflected in the permanent file directory unless the user specifies that it be made permanent. EXTEND permission must have been granted by ATTACH before the user can write on the file.

PURGING/RETENTION

Purging causes a permanent file to be removed from the directory and its mass storage space to be released. Purging is done only when the file owner directs it, with the PURGE function. A retention period may be specified when a permanent file is cataloged, but the file will not be purged automatically when its expiration date is reached. An installation can obtain a listing of all expired files. Only one cycle at a time may be purged.

MULTIPLE-READ ACCESS

Any number of users may attach simultaneously a permanent file for read-only access. This is possible only if all passwords are defined at CATALOG time and only the read password is specified at ATTACH time. ATTACH requests are honored in the order submitted.

## 13.2 PRIVACY

The permanent file privacy feature is designed to prevent access to and alteration of permanent files by non-authorized central processor programs. However, it offers no protection against threats by operational personnel or hardware interference. This kind of threat is dependent on the installation environment.

It is responsibility of the installation to ensure the integrity of the operating system at deadstart time and prevent any subsequent alteration.

### 13.2.1 STANDARD PRIVACY PROCEDURE

The permanent file system offers a standard set of privacy controls. If an installation requires a different kind of protection for one or more files, it can define a privacy procedure to replace the standard.

The system automatically ensures that no normal operation will cause permanent mass storage files to be overwritten or otherwise destroyed, and that the directory of permanent files will not be destroyed.

In addition to normal system protection of permanent files, the individual file owner is given the means to prevent unauthorized access to his permanentfile. He can stipulate, when he catalogs a file, the degree to which the file is to be protected from read, write, rewrite access. Once a file is cataloged, it cannot be used by any job unless the necessary passwords are given when a request is made to attach the file.

### 13.2.2 USER PRIVACY PROCEDURE

Provision has been made for an installation privacy procedure to be substituted for the system-provided privacy checking. This procedure must be a PP routine in the running system. An installation has ultimate control over the use of privacy procedures, since it may determine the content of its system, and controls the installation parameter, which when non-zero, allows use of privacy procedures.

Privacy procedures are called at CATALOG time and at ATTACH time.

### 13.2.3 PASSWORDS AND PERMISSIONS

When a file is cataloged, the file owner may specify up to five passwords to be associated with that file. Subsequent cycles assume the same passwords as the first cycle. Each password implies one type of access or permission allowed for use on subsequent ATTACH or new cycle CATALOG attempts. The passwords can by any string of 1-9 alphanumeric characters.

When a permanent file is attached, READ (RD) permission is required to read the file. EXTEND (EX) permission is required to write at end-of-information. MODIFY (MD) permission allows the user to change the text on an existing permanent file with REWRITE. CONTROL (CN) permission is required to PURGE a cycle, or to catalog an additional cycle. The turnkey password provides an extra measure of control over file usage. When the turnkey password is specified on the CATALOG card, no permission is granted unless the user includes the turnkey on the ATTACH card.

The password for the permission type defined by CATALOG must be given on the ATTACH card to access that particular function.

Example: If the CATALOG card contains RD=ALPHA and EX=BETA, the password ALPHA must be included on the ATTACH card to gain permission to read the file, and BETA is necessary to write it.

When a permission type is not defined on the CATALOG card, it is granted automatically.

Example: If the CATALOG card contains only read (RD) and modify (MD) passwords and the ATTACH card contains the modify password, control permission and extend permission are granted automatically.

1. On the CATALOG card of a file:

   TK=JIM, EX=JOHN, MD=MINE, RD=YES

   On subsequent ATTACH cards:

| Passwords | Permissions granted |
|---|---|
| PW=JIM, JOHN | Extend, control |
| PW=JIM, MINE | Control, modify |
| PW=YES, JOHN | None |
| no password | None |
| PW=JIM, YES | Read, control |

2. On the CATALOG card of a file:

   EX=JACK, CN=CONT

   On subsequent ATTACH cards:

| Passwords | Permissions granted |
|---|---|
| PW=CONT | Control, modify, read |
| PW=JACK | Extend, read, modify |
| no password | Read, modify |

3. On the CATALOG card of a file:

   TK=MYFILE, EX=PASS

   On subsequent ATTACH cards:

| Passwords | Permissions granted |
|---|---|
| PW=MYFILE | Read, control, modify |
| PW=MYFILE, PASS | Read, extend, modify, control |
| no password | None |
| PW=PASS | None |

4. On the CATALOG card of a file:

   CN=A, MD=B, EX=C, RD=D

   On subsequent ATTACH card:

| Passwords | Permissions granted |
|---|---|
| PW=D | Multi-read access |
| PW=C, D | Read, extend |

5. On the CATALOG card of a file:

   CN=A, MD=B, EX=C

   On subsequent ATTACH card:

| Passwords | Permissions granted |
|---|---|
| no password | Multi-read access |
| PW=C | Read, extend |

6. On the CATALOG card of a file:

   CN=A, MD=A, EX=A

   On subsequent ATTACH card:

| Passwords | Permissions granted |
|---|---|
| no password | Multi-read access |
| PW=A | Read, extend, modify, control |

Any attempted breach of privacy causes immediate job termination with a dayfile message. If permission is not granted when ATTACH is requested, the job is terminated. After a permanent file is attached, any attempt to perform an operation not permitted will cause job termination.

### 13.2.4 UNIVERSAL PERMISSION

The universal permission feature is an installation option. If this option has been activated, the installation must define a nine-character password as the universal permission password. SCOPE will grant CONTROL permission if the universal permission password is given by a user.

To use this option, the file must first be attached as follows:

```
JOB
ATTACH(LFN, FILES, CY=10,PW=UNIVEPERM)
PURGE(LFN)
6-7-8-9
```

UNIVEPERM is the universal permission password and IP. UP is set to one.

## 13.3 PERMANENT FILE CALLS

Permanent file functions described in this section are available as control cards or running program calls (macros). The same parameters are used for both; the difference is in the format of the call, and in the ability to test status in the running program.

### 13.3.1 PARAMETERS

The parameters described below are common to both control card and macro functions. Parameters (except lfn and pfn) may appear in any order on control cards or macros. Each is written cc (value or password) where cc is the two-letter code; only lfn and pfn are order dependent and if left blank, commas must be inserted.

lfn Logical file name, 1-7 alphanumeric name (first character alphabetic) by which file is known and referenced at a control point. Once a permanent file is attached to a control point, it is referenced by this name.

pfn Permanent file name, 1-40 alphanumeric characters assigned by file creator under which a file is catalogued.

RP Retention period (days), 0-999, specified by creator; indefinite retention indicated by 999. The installation defines default value.

PP Privacy procedure parameter; 1-9 characters, used by installations to pass information to installation-defined procedure.

CY Cycle number (1-63) assigned by creator. The default value on initial CATALOG is 1, and on ATTACH, the highest number cataloged.

PW The list of passwords indicated on ATTACH card to establish user's access permission; written as:

$$PW=psw_1,psw_2,psw_3,\ldots,psw_n$$

PW is also used on CATALOG card when a new cycle is added.

| | | |
|---|---|---|
| TK | Turnkey password | 1-9 alphanumeric characters |
| CN | Control password | |
| MD | Modify password | |
| EX | Extend password | |
| RD | Read password | |
| ID | Identifies file creator at catalog time | |
| SD | Subdirectory number, 1-999 | |

If a user assigns parameter values other than those specified above, the results will vary depending upon the parameter and value selected.

### 13.3.2 MACRO EXPANSION AND FDB

Each permanent file macro expansion contains a call to CPC (central program control). CPC, in turn, calls for execution of the function by the system routine PFx. Parameters necessary for execution of a function are contained in a central memory table called the file definition block (FDB).

The macro for generating an FDB has the format:

```
fdbaddr     FDB     lfn,pfn,parameter list.
```

fdbaddr is the symbol to be associated with word 5 of the FDB and must be present in the location field. Parameters are separated by commas, and the list is terminated by a blank. Parameters may include any of the 2-letter codes in section 13.3.1. Parameters will be entered into the FDB in the order listed.

The FDB will be generated in-line during assembly whenever the macro is called.

If the RC parameter is specified in the function call referencing an FDB, a return code will be available to the user in word 5 of the FDB which is equal to fdbaddr. These codes are listed in section 13.3.4.

### 13.3.3 CONTROL CARDS

Control card requests take the forms:

name(lfn)

name(lfn, pfn, parameters)

lfn (logical file name) and pfn (permanent file name) identified by position. Parameters may be listed in any order, separated by commas, and terminated with a right parenthesis. Each parameter is identified by its 2-character code. For example:

```
ATTACH(OLDPL, PROGRAMLIBRARYDEC13, PW=OPLREAD, SD=6)
```

Continuation control cards may be supplied as needed. If a card has no terminator, column 1 of the next card is considered the immediate continuation of column 80.

#### CATALOG

This request is used to catalog (make permanent) a newly created, local file attached to the control point. Cataloging consists of locking out the record blocks on which the file resides, and entering the file's location and identification into the permanent file directory.

A CATALOG request may create the first cycle (CY 1 assumed) of a new permanent file with a unique name (initial mode); or it may create a new cycle of an existing permanent file (newcycle mode).

The mode of operation is determined by the CATALOG call. Newcycle mode is signalled by the presence of the CY parameter, and initial mode by its absence.

| | |
|---|---|
| control card | CATALOG(lfn, pfn, parameters) |
| macro | CATALOG fdbaddr, RC |

Required parameters are lfn and pfn, or fdbaddr. The parameters may be included on the control card or in the FDB.

| | |
|---|---|
| ID | Creator identification (1-9 characters) |
| PW | Password list (has meaning only when a new cycle is cataloged) |
| RP | Retention period |
| CY | Cycle number |
| TK | Turnkey password |

| | |
|---|---|
| RD | Read password |
| EX | Extend password |
| MD | Modify password |
| CN | Control password |
| PP | Privacy procedure parameter |
| SD | Subdirectory has meaning only for newcycle mode |

If RC is specified, the user is notified by a return code on a non-fatal error condition, at fdbaddr.

The CATALOG process always searches for a duplicate name. If the SD parameter is given on a CATALOG, that subdirectory is searched first. When a duplicate name occurs, the action taken depends on the CY parameter, in the following instances, and on the IP.RNF installation parameter.

1. No CY parameter is specified for a new file.
2. CY number is a duplicate of an existing file.
3. No room in existing directory entry for a new cycle (maximum of five).

In each instance IP.RNF is examined. If IP.RNF=0, the job is terminated. If IP.RNF=1, the permanent file name is modified, a new permanent file name is created and returned to the user via the FDB, and a message is output to the dayfile.

For random files, the user must ensure that the index has been written out as the last logical record of the file before requesting CATALOG; the file must be closed.

To catalog a new cycle of an existing permanent file, the user must provide the password necessary to obtain control permission.

Once a file is cataloged, it remains attached to the control point as a local file as if it were an attached permanent file with all access permissions granted to it.

Examples: 1. Initial Mode CATALOG

```
JOB.
REQUEST (FILE1)
COMPASS.
LGO.
CATALOG(FILE1, PFILE, CN=XXX, MD=YYY)
7/8/9
  •
  •        COMPASS program to create FILE1
  •
6/7/8/9
```

In the above, a COMPASS program is assembled and executed. This job writes on FILE1 which has been assigned to a permanent file (mass storage) device by the REQUEST card.

After the COMPASS program is completed, the CATALOG control card makes FILE1 a permanent file named PFILE. Since no CY parameter is specified, cycle 1 is assumed and initial mode CATALOG is attempted. If a file named PFILE already exists, and IP.RNF=1, the system will generate a new name for the file to catalog it. The file will be assigned a control password (XXX), and a modify password (YYY); no protection is given on READ or WRITE functions.

2. Initial mode CATALOG:

```
JOB.
REQUEST(TAPE1, MT)
REQUEST(OLDPL)          Assign mass storage
COPYBF(TAPE1, OLDPL)
CATALOG(OLDPL, PROGLIB1, CY=50, CN=XX)
REWIND(TAPE1)
6/7/8/9
```

This job copies a program library from tape (TAPE1) to mass storage and makes it a permanent file. The file OLDPL is assigned to a permanent file (mass storage) device. The CATALOG card causes the file OLDPL to be made permanent under the name PROGLIB1, cycle 50. If a file of the same name already exists in the system and neither a CONTROL or TURNKEY password was defined, an attempt is made to CATALOG in newcycle mode. The latter is possible only if there is space for an additional cycle, and a cycle 50 does not already exist.

Assuming an initial CATALOG is executed, the file will be cataloged with CONTROL permission. If a newcycle CATALOG is executed, it will assume the passwords of the other cycles of the file and the CN parameter will be ignored.

3. Initial mode CATALOG:

```
            .
            .
            .
FDBA FDB  LF1,MFILE1,CN=Z,MD=X,TK=Y
            .
            .
            .
        CATALOG FDBA,RC
            .
            .
            .
```

The above assumes file LF1 exists, local to this control point, on a valid permanent file device.

The CATALOG macro references FDBA which contains the parameters necessary to make LF1 permanent. As CY is not specified, an initial mode CATALOG is attempted. The RC parameter on the CATALOG indicates that on a non-fatal error, control will be returned to the user (a return code is available in location FDBA). If RC is not specified, the user's job is terminated with a diagnostic message on detection of a non-fatal error.

4. Newcycle mode CATALOG

```
           .
           .
           .
FDB5 FDB LF16,PFILE,CY=12,PW=XXX)
           .
           .
           .
       CATALOG FDB5
           .
           .
           .
```

This job adds a second cycle to the permanent file PFILE created by example 1. LF16 is again assumed to be a valid file on a valid device. The PW password is used to submit passwords to establish control permission. XXX is the only password required, as no TURNKEY was defined on the initial CATALOG in example 1.

If for any reason, the CATALOG in example 1 failed, PFILE would not exist and a newcycle CATALOG not executed. In this event, an initial CATALOG would be attempted. However, as no passwords are defined on the FDB, the file would be cataloged unprotected. If this is undesirable, an alternative form of the FDB follows:

```
FDB5 FDB  LF16,PFILE,CY=12,PW=XXX,TK=ZZZ
```

The above FDB would perform the same as for a newcycle CATALOG as the TK parameter would be ignored. However, if an initial CATALOG occurred under the circumstances described above, the PW parameter would be ignored and the file would be protected by the TK password (or any other passwords specified).

Assuming the installation has more than one subdirectory, the user would have been informed of the subdirectory number after the successful CATALOG in example 1. He would have specified this number to expedite any subsequent attempts to perform a newcycle CATALOG of the file.

5. JOB.

```
REQUEST(LOC)              Assign mass storage
REQUEST(TAPEX,MT)
COPYBR(TAPEX, LOC)
CATALOG(LOC,PROBLIB1,SD=6,CY=21,PW=XX)
EXIT.
CATALOG(LOC,SCRATCH,TK=ABC,RD=DEF,RN=1)
6/7/8/9
```

This job copies a file from tape to a file on a permanent file device. Assuming example 2 was successful, a newcycle CATALOG is attempted. First, subdirectory 6 is scanned for the entry. CONTROL permission is established by the PW parameter, and the newcycle CATALOG is executed. If errors are encountered, the statements after the EXIT card will be executed. This will cause an attempt to CATALOG the file under a different name (SCRATCH). The latter is an initial CATALOG and the file will be given TURNKEY and READ protection.

ATTACH

ATTACH must be used to access an existing permanent file. The ATTACH procedure ensures that the user has legal access to the permanent file and makes it a local file.

| | |
|---|---|
| control card | ATTACH(lfn,pfn,parameters) |
| macro | ATTACH fdbaddr,RC |

Required parameters are lfn and pfn, or fdbaddr; pertinent optional parameters are:

| | |
|---|---|
| CY | The cycle of the file to be used |
| PP | Privacy procedure parameter |
| PW | 1 to 5 passwords to establish access permissions |
| SD | Number of the subdirectory of this file's entry |

The RC parameter is used the same as for CATALOG.

If the CY parameter is not present and the file has multiple cycles, the default cycle is the one with the highest cycle number, presumably the latest. If the CY parameter is present and the cycle number is not known to the system, the request cannot be honored.

System evaluation of passwords establishes the type of access granted to the user for each file. Subsequent to ATTACH, the user cannot access the file in any way for which he does not have permission. For example, if ATTACH results in READ permission only, the user cannot subsequently use MODIFY or EXTEND.

Multi-read access is possible by generating READ permission only. If another control point also has this file attached and multiread access is not possible, the permanent file manager will wait for access to the file.

If SD is present, but the file is not found in the specified subdirectory, the entire directory will be searched; and a dayfile message will advise the user of the correct subdirectory.

Before a file can be used after an ATTACH, the file must be opened. ATTACH does not imply opening the file. The success of an OPEN request depends upon the permissions granted when the file is attached.

An incomplete cycle may be attached if the cycle parameter has been specified with a cy keyword. Control will be the only permission granted in this case.

Examples: 1.

```
JOB.
ATTACH(LFILE,PFILE,PW=XXX,ZZZ)
  .
  .        remainder of job
  .
6/7/8/9
```

The permanent file PFILE (as created by example 1 under CATALOG) is attached by control card. It is given the logical file name LFILE and will be referenced subsequently as such. As no SD parameter was specified, all subdirectories will be searched for the file. Because only control and modify passwords were defined at CATALOG time (in example 1 under CATALOG), EXTEND and READ permissions will be granted by default. In addition, since the control password XXX was included in the password list, CONTROL permission is also granted. The extra password ZZZ will be ignored, but will not cause an error.

CY is not specified in the ATTACH so the cycle with the highest number will be attached.

2.

```
FDBZ  FDB LF,PFILE,SD=5,CY=1
      .
      .
      .
      ATTACH FDBZ,RC
```

This sequence illustrates an ATTACH by system macro using the file created in example 1 under CATALOG.

Subdirectory five (if it exists) will be scanned first for the file. If it is not found, or if the subdirectory does not exist, all subdirectories will be scanned. If example 1 under CATALOG executed correctly and the file has not been purged, it will be found. Setting CY to 1 ensures that cycle 1 is attached (compare with example 1 under ATTACH).

PW is not specified in the FDB, so READ and EXTEND permissions are granted by default. There is no multi-read access.

The RC parameters are also specified causing the immediate return of an error code to the user at FDBZ on non-fatal error detection.

3. 
```
JOB.
ATTACH(OLDPL, PROGLIB1)
UPDATE(Q)
RETURN(OLDPL)
COMPASS(I=COMPILE)
7/8/9
*IDENT ABC
.
.
.
*COMPILE DEF
6/7/8/9
```

This job uses the file created by examples 2 and 5 under CATALOG. As no cycle number was specified, the file created in example 2 (cycle 50) will be attached. As only the CONTROL password was defined, MODIFY, EXTEND, and READ will be granted.

This ATTACH causes the permanent file PROGLIB1 to be available at this control point under the logical file name OLDPL. This is a relatively simple example of an UPDATE operation. (Note that the RETURN detaches the file logically.)

4. 
```
FDB1   FDB  LF1,PERM,TK=T,MD=M,EX=E,CN=C,PW=T
            .
            .
            .
            CATALOG   FDB1
            CLOSE     LF1,UNLOAD,RECALL
            ATTACH    FDB1
            .
            .
            .
```

This sequence assumes initially that a local file LF1 has been created, and then cataloged with the name PERM (cycle 1 by default) with TURNKEY, CONTROL, MODIFY and EXTEND passwords. The PW parameter is ignored on a CATALOG operation.

After the CATALOG is completed, a CLOSE UNLOAD is performed, causing a logtcal detach as does the RETURN control card.

The file PERM can now be re-attached with the same FDB (although it is not mandatory) to conserve CM space. Cycle 1 is assigned as the default value, since it is the only (and highest number) cycle present. The PW parameter obtains TURNKEY permission, and READ is granted by default. This is an example of an implicit READ only type of ATTACH.

5.
```
JOB.
REQUEST(LF1).     Assign mass storage
COMPASS.
LGO.
CATALOG(LF1,PERM,TK=T,MD=M,EX=E,CN=C)
RETURN(LF1)
ATTACH(LF1,PERM,PW=T)
7/8/9
 .
 .        program to create LF1
 .
6/7/8/9
```

This example is the control card version of example 4 and illustrates the same points without the multiple use of an FDB.

6.
```
FDB1  FDB   LF1,PERM,TK=T,MD=M,EX=E,CN=C
FDB2  FDB   LF1,PERM,PW=T
      .
      .
      .
      CATALOG   FDB1
      CLOSE     LF1,UNLOAD,RECALL
      ATTACH    FDB2
```

This is the same as examples 4 and 5, except that it uses two FDB's.

EXTEND This function extends the length of a permanent file. The file must be attached to the control point, and EXTEND permission must be granted. A file cataloged by a given job may be extended also by that job. EXTEND makes permanent only the information that has been added to the end of the file (data written normally at end-of-information).

```
control card     EXTEND(lfn)

macro            EXTEND   fdbaddr,RC
```

Required parameters are lfn or fdbaddr. In the macro form, the FDB references may be either a new FDB containing only the lfn or the FDB used at ATTACH time. RC is used the same as for the CATALOG function.

The newly written section will acquire the privacy controls of the permanent file. If lfn is an indexed file, the current index will be rewritten at the end of the file, invalidating any prior indexes. Random files must be closed before an EXTEND is issued.

Examples: 1.

```
JOB.
ATTACH(LF1,PROGLIB1)
COMPASS.
LGO.
EXTEND(LF1)
7/8/9
  .
  .      program
  .
6/7/8/9
```

In the above, the file created in example 2 under CATALOG, is attached. A program to write at end-of-information of LF1 is compiled and executed. Prior to terminating the job, the EXTEND function will make permanent any data written at end-of-information of LF1.

2.

```
      JOB.
      ATTACH(LF1,PROGLIB1)
      COMPASS.
      LGO.
      7/8/9
      :
FDBX  FDB   LF1
      :
      EXTEND FDBX,RC
      6/7/8/9
```

This example is similar to example 1, except the EXTEND is performed by system macro prior to the end of the program. The RC parameter is also specified on the EXTEND. The EXTEND references an FDB containing the logical file name.

PURGE

PURGE removes a file from the catalog of permanent files.

| | |
|---|---|
| control card | PURGE(lfn) |
| macro | PURGE fdbaddr,RC |

Required parameters are lfn and fdbaddr. CONTROL permission must be established.

PURGE operates in two modes, complete or partial. Each is executed by logical file name.

## Complete PURGE

If, when the PURGE is issued, the user has all four permissions from the previous ATTACH, a complete PURGE is attempted. If successful, the file becomes a normal logical file without the restraints of permanency.

If the file is returned, close unloaded, or the job ends, the file will disappear. However, if prior to these three possibilities a CATALOG is successfully issued, the file can again be made permanent.

### Partial PURGE

A partial PURGE is attempted if all four permissions are not granted. This type of PURGE preserves the original privacy controls until the file disappears at job termination or by a return or close unload. If a file is partially purged, it cannot be re-cataloged. The partial PURGE possibility is necessary since there is no hierarchy of permissions. If a file were completely purged on the control password only, a user could then read or write the file.

The partial purge is also useful for preserving file privacy when the universal permission password is used. The latter can be set to grant only CONTROL permission thereby allowing the file to be purged. No other action on the file is possible. Activation of the universal permission option for control threatens permanency only and not data privacy.

Examples: 1.

```
JOB.
ATTACH(LF1,PFILE,CY=1,PW=XXX)
PURGE(LF1)
•
•
•
6/7/8/9
```

Using the file created in example 1 under CATALOG, this job initially attaches LF1, generates CONTROL permission by specification, and READ and EXTEND permission by default.

The PURGE card referencing this logical file name then executes a partial purge of the file. That is, in the program following the PURGE, LF1 can be read or written, but not modified or re-cataloged. When the job terminates, the file will disappear.

2.

```
         •
         •
         •
FDB1     FDB  LF1,PFILE,CY=1,PW=XXX,YYY
FDB2     FDB  LF2,QFILE
         •
         •
         •
         ATTACH    FDB1
         PURGE     FDB1
         •
         •
         •
         CATALOG   FDB2
```

This is similar to example 1 except that on the ATTACH the MODIFY password is given also; thus all four permissions are granted. The PURGE which follows is complete, and the file can be recataloged subsequently, as shown.

### 13.3.4 RETURN CODES

Codes for the 18-bit code/status word in FDB+4=fdbaddr are listed below. (Bits 9-17 are error codes; codes greater than 027 cause job termination.)

| User Return Code (Octal) | Meaning |
|---|---|
| 0 | Function successful |
| 1 | Not used |
| 2 | Logical file name already assigned (ATTACH) |
| 3 | Logical file name not found (CATALOG, EXTEND, PURGE, RENAME) |
| 4 | Blank permanent file name (CATALOG, ATTACH) |
| 5 | Directory full (CATALOG) |
| 6 | Catalog full (CATALOG, EXTEND) |
| 7 | Permanent file device unavailable |
| 10 | Index not written for a random file on CATALOG or EXTEND. |
| 11 | Illegal device for file residence (CATALOG) |
| 12 | ATTACH request for unknown file |
| 13 | Cycle referenced does not exist (ATTACH) |
| 14 | Invalid cycle number |
| 15 | Duplicate name/cycle or not slot on CATALOG |
| 16 | Attempt to CATALOG non-local file |
| 20 | Function attempted on non-permanent file |
| 21 | Function attempted on purged file |
| 22 | CATALOG attempt, no word pair |
| 23 | Cycle incomplete on ATTACH |
| 24 | Duplicate ATTACH request |

All errors are fatal on control card requests and on macro requests unless the RC parameter is specified.

## 13.4 PERMANENT FILE UTILITY ROUTINES

Three utility routines are associated with the permanent file system:

DUMPF dumps permanent files to tape

LOADPF loads these files from the dump tape

AUDIT provides printed reports on the status of each file

These routines are described in the SCOPE Operator's Guide.

# APPENDIX SECTION

# CHARACTER SET

# A

| 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 |

[†] Written as 12 on magnetic tape

| CHAR (printed) | DIS-PLAY | HOLLERITH (punched) | EXT BCD |
|---|---|---|---|
| → | 65 | 0-8-5 | 35 |
| ∨ | 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 |
| | 00xx | ††† | ††† |
| blank | 55 | 6-8 | 16†††† |

† 11-0 and 11-8-2 are equivalent

†† 12-0 and 12-8-2 are equivalent

††† Since the display character 00 has no BCD or Hollerith equivalent, it may be used as a flag character. The flag character together with the character immediately following it (xx) have special interpretations.

†††† A 6-8 punch is converted to a display code 55 with no diagnostic given.

# FET EXTENSION FOR COBOL AND SORT/MERGE USAGE B

13 WORDS of SCOPE-FET

| | 59 | 56 | 53 | 47 | 41 | 29 | 23 | 17 | 13 | 8 | 5 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | | | | Blocker Address | | | | Deblocker Address | | | | |
| 15 | D | | | Record Length (logical) | | | | ST | | U | Record Type | |
| 16 | Record Mark Value | | | Maximum Record Length(logical) | | Blocker Bytes | | Size of Single Occurrence of Trailer | | | | |
| 17 | M | | BCP† | Key Position | | Deblocker Bytes | | Key Length | | | | |
| 18 | t | IN<br>OUT | | Spacing Control | | | | Label Address | | | | |
| 19 | | | | | Record Count | | | | | | | |
| 20 | | | | Fixed record Length (Logical) | | Record Count Rerun Period | | | | | | |

†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

VARIABL L, M, P, K, S, R

The VARIABL 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 a density defined by an installation parameter; they are 80 characters in length.

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.

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<br>Non-blank = security protected |
| 5 | Volume Density | 1 | 12 | Density of file information on tape<br>blank or 00 = 556 bpi<br>1 = 200 bpi<br>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 standardization | 1 | 42 | Must be blank |
| 10 | Creation Date | 5 | 43-47 | Date file was created; YYDDD, which is 2n characters for year and 3n characters for Julian date (001 to 366) |
| 11 | Reserved for future standardization | 1 | 48 | 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 | 20 | 61-80 | Must be blanks |

The volume trailer label format is identical to file trailer label format except for the third character.

The 3000 series labels has the following format:

Header 80 BCD characters

Character Position

| | |
|---|---|
| 1 | Recording density, 2, 5, or 8, indicating 200, 556, or 800 bpi |
| 2-3 | Unique label identifier ( ) |
| 4-5 | Logical unit number, 2 digits |
| 6-8 | Retention cycle, 3 digits |
| 9-22 | File name, 14 alphanumeric characters |
| 23-24 | Reel number, 2 digits |
| 25-30 | Date written; Month, Day, Year (mmddyy) in File ID |
| 31-32 | Edition number, 2 digits |
| 33-80 | User supplied information |

Trailer 80 characters, EOF or EOT, each followed by block count

| | |
|---|---|
| EOT, block count | Indicates end of reel for multi-reel files; it is preceded by one and followed by two tape marks. |
| EOF, block count | Ends a file or multi-files; it is preceded by one and followed by one or two tape marks. |
| EOS | 3600 SCOPE end-of-system label |
| Two tape marks | |

# RELOCATABLE SUBROUTINE FORMAT

The deck of one subprogram (subroutine) as it is output from an assembler or compiler 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 | | WC | | LR | L |
|---|---|---|---|---|---|
| 59 | 53 | 47 | 35 | 26 | 17 0 |

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 for each type of table

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 = $77_8$ LR and L are ignored.

| | | |
|---|---|---|
| word 1 | name of subprogram | |
| word 2 | date typed by operator at deadstart time, one leading and one trailing blank. | |
| | 59 | 17 0 |

The binary output from an assembly consists of all loader control cards (LCC) written as individual records, then an identification table of 14 words is written (77-table), followed by the deck. If errors occur in assembly, no binary output, except the 77-table and any LCC records, will appear.

For absolute programs, following the 77 table is another control word followed by the absolute program. This control word contains:

CP Programs: 5000 $L_1L_2$ ffff fftt tttt

$L_1$ = primary overlay level number
$L_2$ = secondary overlay level number
$L_1, L_2$ = 00 for first overlay
ffffff = origin -1 as specified on the IDENT line
tttttt = entry point address as specified on IDENT line

PP Programs: nnnn nn00 ffff 0000 cccc

nnnnnn = program name
ffff = origin -5 as specified on the IDENT line
cccc = program length (including this control word) in central memory: (program length+9)/5

PIDL

Program identification and Length table contains the subprogram identification and declarations concerning common block allocation.

Identification Word

CN $34_8$
LR Unused
L 0

| | | |
|---|---|---|
| word 1 | name of subprogram | PL |
| | 59 | 17 0 |

PL Program length

| | | |
|---|---|---|
| words 2-WC | name of common block | BL |
| | 59 | 17 0 |

If blank common, name is 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 PIDL table. All PIDL tables must appear before any other tables for a given subprogram. The names of common blocks may not be duplicated in a PIDL table. 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

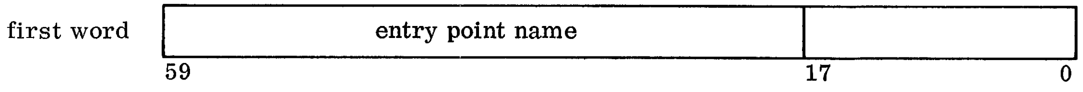
CN = $36_8$

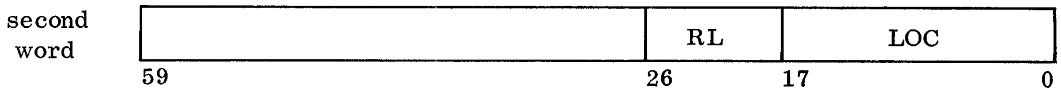
LR = Ignored

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;

| | |
|---|---|
| 0 | absolute, relative to RA (no relocation) |
| 1 | program relocatable |
| $3\text{-}77_8$ | relative to common block M, where M is in position RL-2 of LCT. M must not refer to blank common. |

LOC = address of entry point

## 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. With the text table, all addresses are relocated absolute or relative to program origin, never relative to a labeled common block. As a result, 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. Relocation specified by FILL tables is accomplished after all programs from the input file are loaded; so that text referenced by the FILL table may appear after it. Since FILL tables are processed in order, the results of a FILL table will be preset when subsequent relocation is specified. For example, multi-loads into labeled common should be avoided as the result will be unpredictable.

### Identification Word

CN = $42_8$

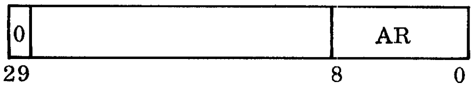
LR = 0

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.

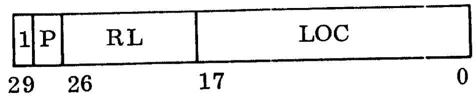
A zero byte is treated as a control byte in the format:



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-$77_8$ | 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.

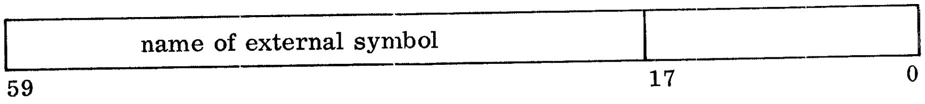
Identification Word

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:

| | | | | |
|---|---|---|---|---|
| 1 | P | RL | LOC | |
| 29 | 26 | | 17 | 0 |

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 = Ignored

L = 1 if replication is not to be deferred until all text is loaded. (Instant replication)

### Words 1 through WC

Each entry in the table consists of two words in the format:

| | | | | | |
|---|---|---|---|---|---|
| word 1 | I | | SR | S | |
| word 2 | C | B | DR | D | |
| | 59 | 41 | 26 | 17 | 0 |

S = Initial address of the source data, should be non-zero

SR = Relocation of the address specified by S.

0 Absolute, relative to RA

1 Program relocation

3-$77_8$ Relative to common block M, where M is in position SR-2 of LCT. M must not refer to blank common

D = Initial address of destination of data

DR = Relocation of address specified by D; range of values same 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 = Ignored

L = Ignored

| | |
|---|---|
| entry point name | |
| 59 | 17 0 |

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.

### SYSTEXT — Systems Text

Normally, systems text is derived from the library overlay named SYSTEXT, and is assembled prior to assembly of the source program, although this may be changed through the S option. Systems text overlays on the library look like loader overlays with the following control word:

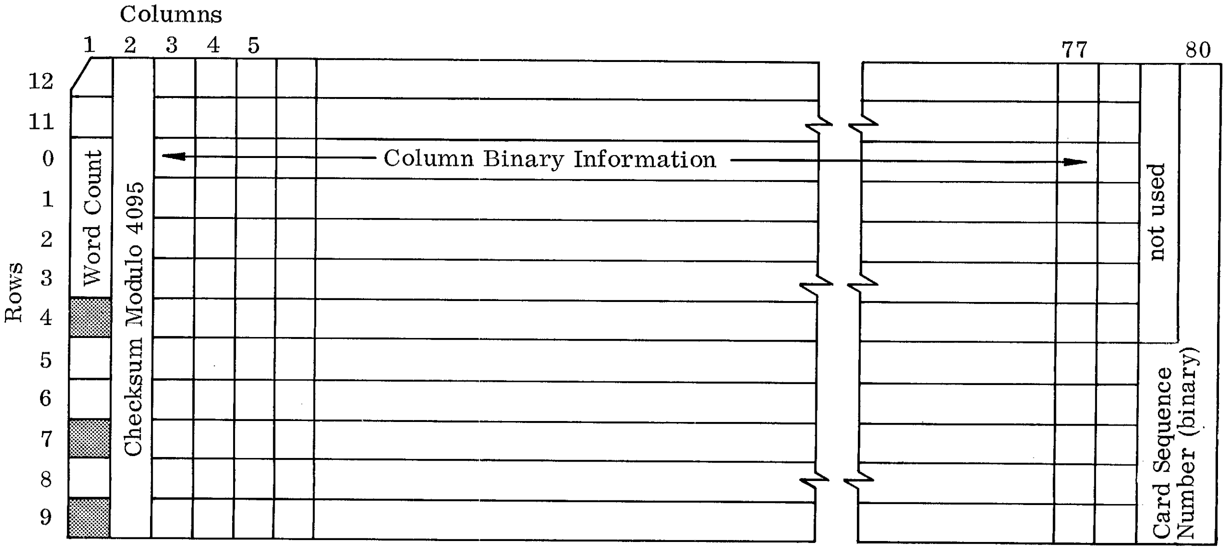
```
5000 0101 0000 0000 0000
```

Data consists of coded lines. A minus zero word follows the last coded line.

Systems text can be deleted by using the S option with a dummy (non-existent) record name. A non-fatal loader message is produced when COMPASS attempts to load the overlay.

# CARD FORMAT

| 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 79 and 80 contain 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

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
01 or 1b
02 or 2b
03 or 3b
04 or 4b
05 or 5b
06 or 6b
07 or 7b
10
11
12
13
14
15
16
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. Checksums 7777(octal) and 0000 are equivalent.

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 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 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 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. 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 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.

## FIELD LENGTH CALCULATION IN OCTAL

The Field Length (memory requirements) for a program during execution after it has been loaded is:

LWA LOAD (rounded up to the nearest 100)

However, loading a program may require more memory space for the loader program and its tables. This value, rounded up to the nearest 100, is the larger of the two quantities:

LWA LOAD

or

BLNK COMN + field length - FWA TABLES

(If BLNK CMN address = 0 or is less than LWA LOAD, replace BLNK COMN with LWA LOAD in formula.)

Field length is obtained from the JOB card or RFL card. LWA LOAD, BLNK COMN, and FWA TABLES values are obtained from the core map.

Example:

For the example program used in this appendix:

Field Length = 40000<br>
LWA LOAD = 7316<br>
BLNK COMN = 7314<br>
FWA TABLES = 34023

The field length required for execution is 7400 (7316 rounded up to the nearest 100). The field length required for loading is 13300 (13273 rounded up to the nearest 100).

LWA LOAD + field length - FWA TABLES<br>
7316 40000 34023 = 13300 (13273 rounded)

NOTE: This value does not reflect the memory required by CP LOADER for the Entry Point/External Reference Tables. If there is insufficient space to contain the Entry Point Table, loading will be aborted.

# SYSTEM SYMBOL DEFINITIONS

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:<br>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

E.x — This form is used to define symbols for the BNL ECS mods.† It occurs in SCPTEXT and ECSCOM.

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 Interim ECS System was developed by Graham Campbell, Kurt Fuchel, and Sidney Heller, of the Brookhaven National Laboratory, Upton, New York. Work performed at Brookhaven National Laboratory is supported by the U.S. Atomic Energy Commission.

# ERROR MESSAGES H

Messages produced by the SCOPE operating system are listed below. All messages appearing in the system and job dayfiles are included. Most messages that appear on the displays are also included, but some, such as indications of system status which do not require operator response are omitted. Messages that appear at the console during the initial deadstart process have been omitted, except those which indicate error conditions.

Messages are listed in alphabetical order. Items in which the first characters will change according to the parameters of the job in progress are arranged according to the second word of the message. Items beginning with numbers follow the alphabetical lists, and those beginning with asterisks appear last. The routine that produces each message is listed on the right margin.

Abbreviations that commonly appear in this section:

| | |
|---|---|
| CH | Channel |
| CM | Central memory |
| CMR | Central memory resident |
| CP | Central processor unit, or card punch |
| ECS | Extended core storage |
| EOF | End-of-file |
| EOI | End-of-information |
| EQ | Equipment |
| EST | Equipment status table |
| FET | File environment table |
| FL | Field length |
| FNT/FST | File name/status table |
| MLRS | Maximum logical record size |
| MT | Magnetic tape |
| PF | Permanent file |
| PFD | Permanent file directory |
| PP | Peripheral processor unit |
| PRU | Physical record unit |
| RBR | Record block reservation table |
| RBT | Record block table |
| RBTC | Record block table catalog |

A DOUBLE EOF WAS FOUND BEFORE A /    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.

ADDRESS xxxxxxx IS UNDEFINED    TRACE

Output specification address xxxxxxx is unsatisfied, output for that specification is ignored.

AN ID(P1) IS REQUIRED ON ALL TEXT CARDS COPYN

ARG ERROR LOC

Last word address of area to be cleared (LOC control card) equals or exceeds field length, or last word address is less than first word address; job terminated.

AUDIT ABORT - READ PAST EOR ON ORBTC AUDPP

AUDIT routine for permanent files read end-of-record on record block table catalog file before RBTC entry was complete.

AUTO-RECALL ERROR 1EJ

Job terminated because completion bit was already set when job went into auto-recall.

BACKUP GO WHEN CORRECTED (RESTART) RESTART

Checkpoint number specified exceeds that in first leader record on checkpoint tape. Dayfile message requires operator action; mount earlier checkpoint tape or rewind.

BAD COMPARE COMPARE

Displayed on console, system and job dayfile if discrepancy occurs during COMPARE. If fatal, operator may drop job.

BAD FILE NAME ON REQUEST 5DA

File name on REQUEST or REMOVE control card has bad format. Job terminated.

BAD LABEL 5DA

Unrecognizable label on disk pack read in response to RPACK control card or a DEVADD type-in; label with wrong name read in response to RPACK control card. Job terminates from DEVADD error; otherwise system waits for operator to assign another disk pack drive, assign same drive with different pack, or drop job.

BAD NAME CHECK xxx EDITLIB

Program name xxx from input file is not same as first program name on ADD, ADDBCD, ADDCOS, or ADDTEXT control card. If source is running system library, xxx program not found in directory.

BAD PACK NAME IN REQUEST 5DA

Name on RPACK or REQUEST control card has bad format; or REMOVE card file name on private pack differs from parameter. Job terminates.

BAD TEXT LDR,2LA

Illegal TEXT table entry; relocation code is illegal. No image is produced.

BCD RECORD FOR ADD IS ILLEGAL EDITLIB

EDITLIB found that the record was in BCD while trying to do an ADD. Either ADDBCD or ADDTEXT should be used.

BINARY CARD READ AS FUNCTION CARD EDITLIB

First record on input file must contain all, and only, control cards for EDITLIB. Card is assumed binary if character other than letter, digit, blank, or - . , ( ) / * $ appears.

BINARY RECORD MISSING FROM INPUT COPYN

Logical records requested from system INPUT file must begin with the next logical record on INPUT file.

BKSP(TAPE) 1TD,1DF

BKSP HIT EOF EDITLIB

BLANK 1BT

1BT was called to write blank label on tape.

BLANK COMMON EXCEEDS AVAILABLE CORE, TRUNCATED LOADER

During blank common allocation, no length can exceed FL or overlap 20-word LOADER residence. No reference is truncated, only the allocation for core map.

BLANK PERMANENT FILE NAME PFC,PFA

At least first two characters of permanent file name are blank.

BLANK TAPE READ 1RS,1RT,1TF

Reading of blank tape caused run away. EOI status is returned to FET.

BUFFER ARG ERROR 2BP

FET address is not in field length, or buffer parameters are not within limits;job terminates.

$FET(1) + 4 + L < FL$, $LIMIT < FL$,
$FIRST \leq IN < LIMIT$, $FIRST \leq OUT < LIMIT$.

BUFFER PARAMETER ERROR 1SX

Called by error code 11. Circular buffer pointers in FET fail to satisfy: $0 \leq FIRST < LIMIT \leq FL$, $FIRST \leq IN < LIMIT$, $FIRST \leq OUT < LIMIT$.

C OPTION DOES NOT START WITH C LDR,2LA

C must be first character of OVERLAY control card option which allows user to designate how many words above blank common overlay should be loaded.

C OPTION NOT LAST PARAMETER LDR,2LA

Termination character does not follow C option on OVERLAY card.

C OPTION USED ON OVERLAY (0,0) LDR,2LA

This option cannot be used on level 0,0 overlay card.

CALL IGNORED, COMMON DECK NOT IN DICTIONARY EDITSYM

*CALL,dn does not name a common deck.

CALLING ERROR (CKP)    CKP

Bit 0 of param is not zero, RECALL bit is not set in call to CKP, or param list is outside field length. Fatal dayfile message, job terminated.

CANNOT CATALOG OPEN RANDOM FILE    PFC

Message is followed by logical file name.

CANNOT COMMON P.F.    1AJ,CTS

CANNOT COMPLETE LOAD, JOB ABORTED    LOADER

Intended to correct fatal error conditions in control card mode when LOADER requests LDR to load library routines. Since such a request 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, this message is output, and job is terminated. Core map will contain last good overlay generated.

CANNOT EXTEND OPEN RANDOM FILE    PFE

Message is followed by logical file name.

CANNOT LOAD TEXT INTO xxxxxx    LDR

Overlay that first declared labeled common at xxxxxx has already been written out.

CANT (ADDTEXT) FROM SYSTEM    EDITLIB

ADDTEXT reads and formats records into an overlay for system file. This is not possible if SYSTEM is specified as the input file. This would require running system as defined by current CMR directory. A simple ADD can probably get 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 ASSIGN xx

4ES

xx = equipment type (0 for any mass storage). Requirements for device type, allocation style, or EST ordinal are not satisfied. Operator should type n.GO or n.DROP.

CANT FIND DIRECTORY RECORDS ON INPUT FILE

EDITLIB

EDITLIB cannot locate records that comprise directory on system file (should be in two records immediately following DSD); presumably this is not a system file.

CANT GET PACK ASSIGNED

1BT

Disk pack named in a BLANK type-in is eligible to be blank labeled, but monitor refuses to assign pack to control point. Report this message to a system analyst.

CANT GET PACK ASSIGNED TO C.P.

5DA

Monitor refused a request to set EST status of a disk pack to 402n, (n is control point number). Unless system error occurred, system has assigned pack to another control point. If RPACK control card was processed, system waits for operator to drop or re-assign job; otherwise job terminates.

CANT LIST BETWEEN READY AND COMPLETE

EDITLIB

EDITLIB cannot list programs in a file while it or another file is being constructed (READY card executed more recently than COMPLETE card).

CANT MOVE WHILE READY PENDING

EDITLIB

After READY card is read, location of a program in running system cannot be changed by MOVE card before file named in READY is written out by COMPLETE card.

CANT SET PACK AVAILABLE 5DA

Monitor refused request to set EST status of disk pack to 4000. Unless system error occurred, system assigned pack to another PP before DEVADD type-in was completely processed. Job terminates.

CANT TRANSFER WITHOUT READYING EDITLIB

EDITLIB can execute TRANSFER card only after file is named by preceding READY card with no intervening COMPLETE card.

CANT UNLOAD PACK 5DA

Monitor refused request to set EST status of a disk pack to 4040 because of system error. Job terminates.

CANT UNLOAD PACK AFTERWARDS 1BT

A pack was blank labeled, but monitor refuses to restore pack status to unloaded. Report this message to system analyst.

CARD ERROR, CANNOT FIND FILE NAME LDR,2LA

A control card call for loader does not contain a file name, or the file name cannot be found.

CARD ERROR, FIELD LENGTH TOO SMALL LDR,2LA

Control card call for loader specified inadequate field length.

CARDS MISSING FROM OBJECT DECK LDR,2LA

Word count for loader table not satisfied. Word count in a text table greater than 20 (octal).

CATALOG 4/5 FULL PFC

Warning to the operator that the permanent file catalog is 80 per cent full.

CATALOG ATTEMPT ON NON-LOCAL FILE PFC

Message is followed by logical file name.

CATALOG FULL, CYCLE NOT EXTENDED PFC

CATALOG FULL, FILE NOT PERMANENT PFC

Cycle of file specified on preceding line has not been made permanent because of current catalog condition.

CHECKPOINT xxxx ON lfn

Specified checkpoint completed on checkpoint dump tape; informative dayfile message.

CHECKSUM ERROR IN xxxxxxxx

During deadstart loading computed checksum of some physical record unit did not agree with that appended to PRU during preloading. Indicates trouble with the device identified on the display. Deadstart ceases.

CIO CODE NOT DEFINED ON DEVICE CIO

Either function code issued to CIO does not exist, or it is not applicable to present device.

CIO ERROR.n — 1RI

Followed immediately on console display, system and job dayfile by ROLLIN ABORTED. 1RI, called by type-in or roll in, is about to abort a job. If n = 1, QROLOUT file was not at control point, presumably not properly rolled out. If n = 2, 3, or 4, QROLOUT was at control point, but did not read correctly.

CKP FILE UNKNOWN (RST) — RST

FNT does not contain checkpoint file name given to RST at RA+4. Fatal dayfile message; job terminated.

CKP REQUESTED — CKP

CKP has received control from a user call. Informative dayfile message.

CKP TAPE INVALID (RESTART) — RESTART

Tape supplied was not a checkpoint tape or it was partly destroyed after checkpoint was taken. Fatal dayfile message; job terminated.

CKSUM ERROR RC.xxxx, CD.yyyy — 2RC

Appears at beginning of OUTPUT file if input card has invalid checksum and no checksum suppress punch in row 4 of column 1. Job was terminated 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.

CLOSE ILLEGAL ON NON-EXISTENT FILE — CLO

CLOSE request was issued for a non-existent file.

CODED INF — EDITLIB

COMMON DECK EDITING MUST PRECEDE TEXT EDITING EDITSYM

*COMDECK control card occurred after text deck correction or a *DECK control card. EDITSYM run terminated.

COMMON EQUIPMENT nn ASSIGNED 1AJ,CTS

Local file on a non-allocatable equipment with EST ordinal nn has been made common.

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

Message appears on OUTPUT file and run ends if nth pair of records do not both terminate with same level end-of-record; level numbers are octal.

((COMPLETE)) FINDS REC.MSG. IN FILE sssssss EDITLIB

Fault in disk file used by EDITLIB; sssssss is a local file.

CONFLICTING RECORD COUNT EXHAUSTED COMPARE

Run ends if number specified in control card is such that comparison would be abandoned if a higher number of record pairs were in conflict. For instance, if CONFLICT IN RECORD n is written for five pairs of records and control card has no sixth parameter, run terminates after CONFLICT IN RECORD n has been written 30000 times.

CONFLICT IN RECORD n COMPARE

nth pair of records are not identical, word-for-word. When one record is longer, a separate message appears. Depending on parameters, this message may be followed by a listing of some or all words which differ. For example: 0020,00000000000000000000/00000000000000000001. The 17th word (octal 20) of record n of first file named on COMPARE control card was 0 and the corresponding word in the second file named is 1. First word of record would be 00000. Words are printed as 20 octal digits each. Comparison continues after message.

CONTROL CARD ERROR COMPARE

Console display, system and job dayfile. COMPARE control card contains fewer than 2 parameters or parameter required to be a number, implied by fewer than 2, contains a non-numerical character. Job terminated.

CONTROL CARD ERROR COPYBCD

Console, system and job dayfiles if parameter that begins with digit contains non-digit character.

CONTROL CARD ERROR 1AJ<br>COPYBF

Job terminated; card display immediately precedes this message.

CONTROL CARD ERROR, NO CKPFILE (RESTART) RESTART

RESTART card has two numbers instead of a number and file name. Job terminated.

CONTROL CARD ERROR, NUMBER ERR (RESTART) RESTART

Checkpoint number must be unsigned decimal integer greater than zero. Job terminated.

```
CONTROL CARD ERROR PARAM CNT (RESTART)                          RESTART

    More than two parameters on RESTART card.  Job terminated.


CONTROL CARD REWIND (INPUT) IS ILLEGAL                          COPYN

    System INPUT cannot be rewound.


CONTROL POINT ABORTED                                           1LT

    Operation error was detected and FNT was not full.


jobnameCONTROL POINT DROPPED                                    1EJ

    System job named jobname was dropped and control point
    cleared in response to n.DROP. or n.KILL.


CONTROL POINT ERROR                                             1LT

    Control point error detected following storage relocations.


CONTROL POINT IN USE                                            1LT

    LOAD or LOADX was called to an occupied control point.


CONTROL POINT OCCUPIED                                          1DF

    Operator tried to dump dayfile from occupied control
    point; control point must be vacant (have no job name).


COPY REQUESTED BUT NO OLD PROGRAM LIBRARY                       EDITSYM

    *COPY read from correction input but no old program library
    requested on EDITSYM call card.  Run terminated.

COPYBCD (DAYFILE,TAPE)                                          1DF

COPYBCD (xxxxxx,TAPE)                                           1TD

COPYL DID NOT FIND xxxxxxx.                                     COPYL

COPYL DONE                                                      COPYL

COPYLAB INCORRECT                                               COPYCR,COPYCF,
                                                                COPYBR,COPYBF
```

| | |
|---|---|
| CORE OVERFLOW | UPDATE |
| COS | EDITLIB |
| CP xxxxxx.xxx SEC | 1EJ |
| Central processor running time for job. | |
| CPnn COMPARE ERROR | 2PC |
| Console and system dayfile. Card punch with nn EST ordinal mispunched a card. Operator action not required, as punch offsets bad card and one after it and repunches both cards. | |
| CPnn NOT READY | 2PC |
| Displayed on console only. Card punch unit 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 |
| 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

System and job dayfiles. User local file assigned to reader with nn EST ordinal contains binary card with bad checksum and no checksum suppress punch in row 4 of column 1. Job terminated. 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

System dayfile and console display; on display, it is followed by CR nn RE-READ LAST CARD. Card reader with nn EST ordinal misread last card in output stack and stopped.

CRnn FORMAT ERROR RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to card reader with nn EST ORDINAL CONTAINs 7-9 card, presumably binary, with unrecognizable format, and not included in cards for 80-column binary reading. Job terminated. Record number is decimal xxxx, counting first record in the file as 0000; card number is decimal yyyy, counting first card record as 0001.

CRnn FUNC XMSN PARITY ERROR 2RC

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

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 was 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

System and job dayfiles. User local file assigned to card reader with nn EST ordinal had mixed cards (binary, Hollerith, 80-column) in a single record. Record number is decimal xxxx, counting first record as 0000; first change of type occurred at card number yyyy decimal, 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

Console and system dayfile. Card reader with nn EST ordinal rejected a function code. No operator action.

CRnn RE-READ LAST CARD 2RC

Displayed on console when CRnn 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 any card waiting in the reader, back up last card in output stack to head of input queue, and press CLEAR MEMORY and START on the reader. Reading will continue.

CRnn RE-READ 2 CARDS, TYPE GO. 2RC.

Follows CRnn 6681 XMSN PARITY ERROR. nn is card reader EST ordinal, in octal. Both messages are written on system dayfile, but only this one will persist on console display. Operator should back up the card waiting in the reader, move last two cards in output stack back to head of input queue, and type n.GO. (n is relevant control point number). Card reading will resume.

CRnn SERIAL CHK,RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to card reader with nn EST ordinal has at least one binary card out of sequence. 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

System dayfile and console display; followed by CRnn RE-READ 2 CARDS, TYPE GO. Indicates a transmission parity error between the 6681 and card reader with nn EST ordinal.

CTS CALL ON PROTECTED FILE CTS

User tried to make COMMON a system file. Job terminated.

CYCLE xx, pfn    LPF,LOADPF

Permanent file pfn cycle xx is being loaded.

CYCLE xx pfn    LPF,PFC,PFE,PFP,PFA

Permanent file pfn cycle number xx. If xx is **, cycle has not yet been determined by system or is in error.

CYCLE HAS BEEN DUMPED    PFA

Cycle of permanent file on preceding line was dumped by DUMPF routine but not released.

CYCLE INCOMPLETE    PFA

Cycle indicated on preceding line is incomplete because another control point is cataloging it, or a job terminated while cataloging. For the latter, initial deadstart required to reload all files.

CYCLE NOT IN SYSTEM    PFE,PFP,PFA

Requested cycle of permanent file specified in preceding line not presently cataloged.

CYCLE SUCCESSFULLY PURGED    LPF,PFP

Requested cycle of permanent file specified in preceding line has been purged; other cycles still remain.

DAYFILE    1DF

DAYFILE DUMPED    1DF

DEBUG CARD OUT OF ORDER    1AJ

Issued if DEBUG card appears after TRACE or SNAP cards during load. Job terminated.

DECK DOES NOT END WITH *END EDITSYM

Text deck ended, but no *END card appeared. Run terminated.

DECK NOT ON OLD PROGRAM LIBRARY EDITSYM

Deck specified by *EDIT or *COPY card is not on old program library. Run terminated.

DECK STRUCTURE CHANGED UPDATE

((DEL)) EXHAUSTS PNT BEFORE SATISFACTION EDITLIB

*DELETE (P1-P2) card used, but P2 parameter does not appear after P1 in program name table.

jobname DELETED FROM QUEUE, PARITY ERROR XXXRESQ

Parity error detected when restoring a job; job named will be deleted; job in question should be rerun.

DELETION EDITSYM

DEVICE LOST DATA

Lost data status detected during deadstart preloading. Deadstart again.

DEVICE NOT AVAILABLE 4ES

Result of operator decision to drop a job in answer to CANT ASSIGN xx message.

DEVICE REJECT

Device rejected during deadstart preloading. Deadstart again.

DIAG. SEQ. ALREADY ON. APR

Result of console entry SEQ,ON.

DIAG. SEQ. HAS NO SUCH JOB. APR

Result of console entry SEQ,RUN,nn. Sequencer does not contain job nn.

DIAG. SEQ. IS ON. APR

Result of console entry SEQ,ON.

DIAG. SEQ. JOB nn CONTAINS CU1. FST. ALS. MY1. CM6. CT3 APR

Result of keyboard entry SEQ,LIST,nn. None, some, or all diagnostics may appear in the list.

DIGITS IS NOT OCTAL LDR,2LA

DIRECTORY ALMOST FULL LPF

Warning that permanent file directory is 80 percent full.

DIRECTORY IS 4/5 FULL PFC

Warning that the permanent file directory is 80 percent full.

DIRECTORY FULL, FILE NOT PERMANENT PFC

File specified on preceding line was not made permanent because directory is full.

DIRECTORY UNDER CONSTRUCTION GETS TOO BIG - TOO MUCH CM RESIDENCE EDITLIB

Directory to replace SCOPE system directory in central memory exceeds field length of EDITLIB control point, 30000 (octal) words. Too many programs are assigned to CM residence. Trv EDITLIB again with LENGTH(n) as first control card; n is 4-9, for 4C000 (octal), 110000 (octal) words.

DISK PARITY ERROR

Parity error encountered when loading from disk; loading terminates.

DOUBLE EOF WAS FOUND BEFORE A / COPYN

Double EOF encountered before zero length record was copied.

DMP ARG ERROR DMP

Console, system and job dayfiles. Starting address for CM dumps greater than final address or relative to RA, rather than absolute, and final address greater than field length.

DPF ABORT - NO ENTRY IN FNT FOR 0SD000 DPF

Dump routine called into system that does not define permanent file system.

DPF ABORT - NO RBTC ENTRY IN FNT DPF

Dump routine called into system that does not define permanent file system RBTC file.

DPF ABORT - SYSTEM ERROR NO. xx DPF

DPF found a dump or permanent file manager system error, xx interpreted:

1 Bad address for CM write.
2 APF pointer does not point within PFD chain.
3 Bad RBT chain word; no RB pointer after header information.
4 CM write beyond LIMIT of tape buffer.
5 PFD pointer does not point to PFD header word.
6 RBTC pointer in PFD does not point to correct RBTC entry.

DPF ABORT - UNIT SPECIFIED NON-ALLOCATABLE DPF

EST ordinal in dump control card points to non-allocatable device. Permanent files are only on allocatable devices.

DPF ABORT - UNIT SPECIFIED = ECS DPF

EST ordinal specified on dump control card points to ECS. Permanent files cannot be assigned to ECS.

DPF NOT CALLED BY DUMP ROUTINE DPF

PP program DPF called to a control point not attached to DUM, dump permanent file.

DPF STOPPED BY SYSTEM DPF

Error flag on in control point area.

DPF STOPPED - FULL AND UNIT DUMPS RUNNING TOGETHER DPF

Full dump conflicts with a copy of a unit dump. Unit dump is terminated.

DPF STOPPED - PARITY ERROR ON DUMP TAPE DPF

Parity error on dump tape; tape was positioned to end of previous file dumped. DPF called to terminate dump program.

DUMPING pfn DUMPF

pfn is permanent file name of job being dumped.

DUMPF ABORT - IO ERROR RETURN DUMPF

I/O error code other than parity error caused dump program to terminate.

DUMPF - BAD TAPE - MOUNT ANOTHER TAPE DUMPF

Parity error during OPEN. Bad tape will be unloaded by system. OPEN will be issued by system after operator mounts new tape.

DUMPF FINISHED DUMPF

DUMPF - PARITY ERROR ON CLOSE - DUMP RISKY

Two attempts to close dump tape failed because of parity errors. Status of last file on dump tape is questionable; reloading of this dump tape is risky.

DUMPF - PARITY ERROR ON DISC PF NAME - pfn    DUMPF

Disk parity error while copying permanent file pfn; copying continues.

DUMPF - PARITY ERR WHILE IN OWNCODE    DUMPF

Dump program found a second parity error while trying to reposition tape back to previous file because of first parity error.

DUMPF SIMULATE EOT DUE TO PARITY ERROR    DUMPF

Parity error on dump tape. System will issue request to close reel to simulate end-of-tape and issue open reel for new tape. Dump program will then continue to copy file.

DUP COMMON FILES OF xxxxxxx    1AJ

Local and common file in FNT named (xxxxxxx) same as on COMMON card. Job terminated with control card error.

DUPLICATE CYCLE    PFC

Cycle of permanent file specified on preceding line already exists.

DUPLICATE ENTRY POINT xxxxxxx    LOADER

DUPLICATE FILE NAME    5DA,REQ

Job terminated because name of file at control point was duplicated in label of a private pack being attached as a result of RPACK control card or in parameter of REQUEST control card.

DUPLICATE FILE NAME (RESTART)    RESTART

The requested tape is already assigned to this control point. Job aborts.

DUPLICATE PACK NAME 5DA

Pack name on RPACK control card same as a pack name already at that control point. Job terminated.

ECS EST BAD, CANNOT CONTINUE IRCP

ECS EST is improperly defined on deadstart tape. Deadstart with another tape.

ECS PARITY ERROR 1EJ

ECS parity error during a system storage move terminated job. Job may be active at a control point or in job initialization phase requesting storage at a control point.

ECS PARITY, RESTART AND OFF ECS IRCP

Parity error detected in first 100 (octal) words of ECS. Operator should deadstart again, OFF ECS, and continue. Notify customer engineer of this message.

ECS READ ERROR (CKP) CKP

Irrecoverable parity error or error hang encountered while trying to read ECS. Job terminated.

ECS WRITE ERROR (RESTART) RESTART

Irrecoverable parity error or error hang encountered while trying to write ECS. Job terminated.

EDIT CONTROL CARD SET MUST BE FOLLOWED BY A *EDIT CARD EDITSYM

*INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE control card sets follow *DECK, *COMDECK, *COPY, *WEOR instead of a *EDIT card. Run terminated.

EDITION EDITSYM

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 terminated.

END OF REEL - MOUNT NEXT REEL AND TYPE GO

Instruction to operator during deadstart preloading of system.

ENTER DATE mm/dd/yy

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.

EOF/EOI DETECTED — COPYCR,COPYCF, COPYBR,COPYBF

End-of-file encountered before record count on control card was exhausted, or end-of-information was encountered before file count was exhausted.

EOR APPEARS BEFORE *END IN *DECK ADDTION — EDITSYM

EOR read before *END in *DECK addition. Run terminated.

EOR REQUESTED BUT COMPILE FILE NOT REQUESTED — EDITSYM

*WEOR read, but compile file not requested on EDITSYM call card. Run terminated.

EPF ABORT - NO ENTRY IN FNT FOR OSD000 — EPF

AUDIT routine called into system that does not define directory header permanent file.

EPF ABORT - NO RBTC ENTRY IN FNT — EPF

AUDIT routine called into a system that does not define permanent file system RBTC file.

EPF ABORT - NO. OF RBRS EXCEEDS RBRWD BY xx EPF

Number of record block reservation table entries in system exceeds CM array size in AUDIT routine by xx words. CM array must be large enough to contain first RBR header word for each record block reservation table entry.

EPF ABORT - SYSTEM ERROR NO. xx EPF

AUDIT routine or permanent file manager system error. xx = 1, indicates bad address for central memory write.

EQ xx NOT READY 2LP

EQ xx REJECT 2LP,1SX

(1) Card reader connect function is rejected. (2) Equipment cannot be connected. Compare xx entry with installation equipment select code. If possible, change equipment or unit select code; otherwise drop the control point.

EQ xx RESERVED 2LP

EQ xx XMSN PARITY ERROR 1LP,2LP

Transmission parity error during card reader connect function.

EQUIPMENT IS PHYSICALLY UNAVAILABLE REQ

Requested equipment does not exist in system, is logically off, or is already assigned to this job.

EQUIPMENT OFF - REASSIGN REQ

All equipment of the type assigned in DEVTYPE type-in is off. REQ waits for new assignment.

ERROR CONDITION NOT CLEARED LAST REQUEST CIO

Previous I/O operation terminated abnormally, error processing bit was set, but error number was not cleared in FET.

ERROR IN ABS OVERLAY FILE FORMAT LDR,2LA

Identification code of subroutine does not equal 50B.

ERROR IN FL (1RC) 1RC

Field length not large enough to accommodate memory dump. Fatal dayfile message; job terminated.

ERROR IN LOADING LOADER LOD

LOADER routine has been destroyed on system.

ERROR IN PARAD (1RC) 1RC

Address of parameter list is outside field length. Fatal dayfile message; job terminated.

ERROR MODE = x. ADDRESS = xxxxxx 1EJ

Program terminated because of address or operand error. If x = 0, program attempted to jump to location 0 when address =0; or an attempt was made to execute an invalid instruction.

ERROR ON TRACE OR SNAP CARD DEBUG

Parameter errors on TRACE/SNAP card or no parameters. In latter case, job terminated.

ERROR ON TRACE xxxxxxx CARD TRACE

Error on TRACE card with xxxxxxx ID.

EST ORDINAL OUT OF RANGE 1BT

EST ordinal specified by BLANK type-in exceeds table length.

EST ORDINAL TOO HIGH 5DA

EST ordinal specified by a DEVADD, UNLOAD, or ASSIGN type-in exceeds table length. When ASSIGN was a response to RPACK control card, system waits for operator to drop job or try another assignment. Otherwise, job terminated.

EVICT NOT ALLOWED ON PERMANENT FILES 4ES

EXTEND PERMISSION NOT SET FOR WRITE ON P.F. 4ES

Correct password for receiving permission to extend permanent file was not on ATTACH card.

FET LESS THAN 7 WORDS, xxxxxx 1MT

On attempt to read or write an L tape FET at xxxxxx is less than 7 words. Request is terminated with message DEVICE CAPACITY EXCEEDED, and no data is transferred.

FD-ALL CYCLES FULL LPF

Full reload, no spare cycles available for this file. File is skipped.

FD-CYCLE ALREADY IN PFD LPF

Full reload-cycle already exists in system. File is skipped.

FD-FILE ASSIGNED TO ANOTHER DEVICE LPF

Full reload-allocation type not available on original device.

FDB ADDRESS INVALID PFC,PFA, PFP,PFE

FET MULTI-FILE POSITION INVALID 1MF

Specified position number is less than first position number on first reel of file.

FET OUTSIDE FL CIO

Address of file environment table outside user field length.

FIELD GREATER THAN 80 CHARACTERS LDR,2LA

Loader directive improperly implemented.

FIELD LENGTH NOT SUFFICIENT FOR OVERLAY GENERATION LOADER

Minimum field length is 26000 (octal) for the loader plus sufficient additional length to accommodate loader tables and core map.

FIELD IS NON NUMERIC ILLEGAL TEXT CARD COPYN

FILE ALREADY AT THIS CPT PFA

FILE DEVICE NOT ALLOCATABLE IORANDM

Console, system and job dayfiles. IORANDM was called to read or write beginning of random access record; but file is assigned to non-allocatable equipment. Job terminated.

FILE EXTENDED PFE

Permanent file indicated on preceding line has been extended.

FILE HAS BEEN ATTACHED PFA

Cycle of permanent file indicated on preceding line has been attached.

FILE HAS BEEN CATALOGED AS CYCLE xx, pfn IN SD xxx PFC

Signifies successful cataloging of a permanent file giving cycle, permanent file name, and subdirectory.

FILE IN SUBDIRECTORY xxx LPF

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. Job tzrminated.

FILE NAME NOT FOUND 1LT

1LT did not find TAPE in the FNT; can occur only when external tape is used.

FILE NAME OR SL ILLEGAL OVERLOG

Bad load sequence parameter list entry. lfn = zero or Sℓ non-zero.

FILE NAME TOO BIG (RESTART) RESTART

File name on RESTART card exceeds seven characters. Fatal dayfile message; job terminated.

FILE NAME ON UPDATE CARD GR 7 CHAR, UPDATE ABORTED UPDATE

FILE NOT FOUND

System cannot find file specified in ENPR or EVICT type-in.

FILE NOT IN SYSTEM PFA

Permanent file specified on the preceding line not known to system.

FILE NOT IN USERS FL CTS

Address of request word is outside user's field length. Job terminated.

FILE NOT ON MASS STORAGE DEVICE pfn PFC

FILE NOT ON PRIVATE PACK — 5DA

A REMOVE control card named a file at control point, but file was not assigned to a private pack. Job terminated.

FILE NOT OPEN FOR WRITE — CIO

Write attempted on a file opened for READ ONLY.

FILE x RECORD y HAS PHYS. REC. LONGER THAN 1024 WORDS — COMPARE

Appears on job output file. 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

Console, system and job dayfiles. IORANDM called to read or write beginning of random access record, identified by record name. If record was first written by number it must be addressed by number. Job terminated.

FILE SUCCESSFULLY PURGED — PFD

Last cycle of a file has been purged.

FILE VACUOUS — LDR,2LA

Input file initially positioned at EOF mark. Job terminated.

FIXED PRIORITY JOB jobname  WAITING FOR STORAGE    1RA

Storage not sufficient to bring this job to control point. Message appears on B display at a NEXT control point.

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 FULL    1EJ

Output file cannot be created as FNT is full. Loop will continue until empty entry is found.

FNT IS FULL    REQ

REQ cannot assign requested file, as FNT is full. It will loop until it finds an empty entry.

FNT NAME ERROR (1RC)    1RC

Memory dump file name not correct. Fatal dayfile message; job terminated.

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 terminated 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.

FORMATS INCOMPATIBLE***COPY UNCERTAIN***    COPYCR,COPYCF, COPYBR,COPYBF

Tape formats selected for input/output files can cause a possible loss of data significance. Job continues.

FOURTH PARAMETER SHOULD BE "B" OR "C"    SKIPF, SKIPB

Control card did not indicate B for binary or C for coded file.

FWA-LWA ERROR IO

Console, system and job dayfile. READIN or WRITOUT macro is being executed; workspace, according to sixth word of FET, has a negative length. Job terminated.

GOOD COMPARE COMPARE

Console display, system and job dayfiles just before run ends. COMPARE executed with no discrepancies found.

HDR REC MISSING ON FILE INCORRECTLY POSITIONED 1MR

Labeled tape file missing header or file may be improperly positioned to call 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 and at least one column contained an invalid punch combination. 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 COPYN

Either P1 or P2 cannot be located by the COPYN routine.

ILLEGAL ADDRESS REQUEST TO APR APR

The address xxxxxx in an APR 5 or APR 10 call is out of range for the control point. APR will abort the control point.

ILLEGAL COPYL PARAMETER COPYL

ILLEGAL DEVICE TYPE SPECIFIED OPE

Specified device not found in table of valid types.

ILLEGAL DIAG. SEQ. PARAMETER          APR

Diagnostic sequencer received unacceptable parameter for console entry or program call card.

ILLEGAL ENTRY

Execution of command typed is illegal.

ILLEGAL EOF          EDITSYM

EOF appears before EOR in common section. Run terminated.

ILLEGAL EOR OR EOF          EDITSYM

EOR or EOF appears before *END in copying a deck. Run terminated.

ILLEGAL EQUIPMENT REQUEST          1DF

Operator did not request MT, CP, or LP on a request to dump dayfile.

ILLEGAL FILE NAME          CIO

File name has embedded blanks, does not begin with alphanumeric character, or is longer than 7 characters.

ILLEGAL FUNCTION LFN          CLO

Function code in FET is not processed by CLO.

ILLEGAL FUNCTION CODE          CIO

Function code in FET not allowed for assigned device, file is closed, or a read was requested immediately following a write on a sequential device.

ILLEGAL LEVEL NUMBER          SKIPF,SKIPB

ILLEGAL PARAM OPTION          COPYCR,COPYCF,
COPYBR,COPYBF

Illegal parameter on *COPYLAB control card.

ILLEGAL PRU COUNT AT OPEN REEL    OPE

PRU count not zero or minus at call to OPEN REEL rewind.

ILLEGAL RECATALOG ATTEMPT pfn    PFC

File specified is already permanent.

ILLEGAL REQ FUNCTION    REQ

The REQ routine was called by a central program without auto-recall, with a non-zero status, or with an EST ordinal not in range of EST table.

ILLEGAL REQUEST    4ES

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.

ILLEGAL REQUEST FUNCTION (RESTART)    RESTART

An error occurred when RESTART attempted to request a tape. Job aborts.

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.

IMPROPER UPDATE PARAMETER,UPDATE ABORTED    UPDATE

INCORRECT DUMP TAPE MOUNTED    LOADPF

INCORRECT IDENTIFIER, START OVER    IRCP

Message to operator during ECS deadstart.

INCORRECT OPERATOR ACTION

INCORRECT OPERATOR ASSIGNMENT REQ

Assigned equipment does not match type requested; operator should re-assign correct type.

INDEX ADDRESS NOT SPECIFIED ON RANDOM FILE OPE

INDEX ADDRESS NOT IN FIELD LENGTH OPE

INDEX BUFFER PARAMETER ERROR CLO

FET index address not specified,or outside user field length, or FET not long enough. CLOSE request for random file could not be processed.

INDEX FULL AT OPEN OPE

INDEX LENGTH + INDEX ADDRESS .GT. FL OPE

INPUT FILE ENDED BEFORE ((ADD)) CARD SATISFIED EDITLIB

EDITLIB is trying to execute ADD, ADDBCD, ADDCOS, or ADDTEXT; 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 terminated.

INPUT REC. FOR ((ADDBCD)) HAS IMPROPER NAME CARD EDITLIB

EDITLIB is trying to add record to system file. Input record does not begin with Hollerith card containing record name starting in column 1, 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.

INSUFFICIENT FIELD LENGTH 1AJ

A program call card initiated the load of a 0,0 overlay, but there was not enough field length to load it. The job is terminated.

INVALID CYCLE PFC,PFA

Cycle number requested for permanent file specified in previous line is invalid.

INVALID CHARACTER LDR,2LA

Illegal character on a loader directive card.

INVALID CONTROL CARD LOD

Program call card for a PP program did not meet all requirements. Program name must begin with letter, cannot exceed three characters; no more than two parameters allowed.

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.

INVALID DEVICE TYPE FOR OPEN WRITE OPE

Used for card reader.

INVALID FET ADDRESS Nxxxx 2CA

INVALID L CHAR — COPYCR,COPYCF, COPYBR,COPYBF

Fourth parameter on a copy control card can be L only; signifies presence of a label card.

INVALID LOADER DIRECTIVE — PFC,PFA

First seven characters on a loader directive card do not match any of following: SEGZERO, SECTION, SEGMENT, OVERLAY.

INVALID OPEN PARAMETER — OPE

Z parameter not recognizable.

INVALID OVERLAY - LEVEL OR FWA — 1AJ

Program call card initiated overlay load not of level 0,0 or first word address was not 100(octal). Job terminated.

INVALID PRIVACY PROCEDURE — PFC,PFA

INVALID RECORD READ — 1MF

Invalid record where label was expected while positioning multifile tape.

INVALID REQUEST TO CPC — CPC

Bits 54-59 of request word are 0, so it cannot contain PP program name in bits 42-59. Bits 42-59 can contain only number below 00010 (octal) as part of file action macro. Console, system and job dayfiles. Job terminated.

INVALID STACK ENTRY — 1SX

Called by error code 22 (octal). Request stack entry contains an undefined order code or an out-of-range RBT address, or points to an FNT/FST that contains an out-of-range RBT address.

I/O ERROR (CKP) CKP

An I/O error other than redundant OPEN or CLOSE on any file. Fatal dayfile message; job is terminated.

I/O ERROR (1RC) 1RC

Any I/O error on any file. Fatal dayfile message; job is aborted.

I/O ERROR
DEVICE REJECT

During deadstart loading a device reject occurred while reading from device identified on display. Deadstart ceases.

I/O ERROR
PARITY ERROR OR LOST DATA

During deadstart loading, parity error or lost data error occurred while reading from device identified on the display. Deadstart ceases.

I/O ERROR
RMS DRIVER OVERLAY NOT FOUND

Driver overlay for device identified on display could not be found during deadstart loading. Deadstart ceases.

JOB CARD ERROR 1EJ

Job terminated because of incorrect job card; 20 characters from card are displayed after message.

JOB HUNG IN AUTO-RECALL 1EJ

This job terminated because completion bit was not set and there was no activity at control point.

JOB KILLED 1EJ

The operator typed in n.KILL. for the job. This message will appear only in the system dayfile since the output file for the job is dropped.

JOB PRE-ABORTED 2RC,1EJ

Appears at beginning of job output file if input card caused CHSUM ERROR RC.xxxx, CD.yyyy or FORMAT ERROR RC.xxxx, CD.yyyy. Job terminated when brought to control point.

JOB RERUN 1EJ

Appears in first of two dayfiles for a job rerun (operator typed in n.RERUN.)

JOB WAS RERUN 1RA

Message appears in second dayfile of job rerun by operator.

LABEL MISSING ON DECLARED LABELED TAPE UNIT xx 1MR

LABELED TAPE NOT DECLARED ON MULTIFILE OPE

Multifile request entry did not contain label parameter.

LEVEL NUMBER GREATER THAN 17B or 15D SKIPF,SKIPB

Third parameter on a skip control card exceeded 17 (octal) or 15 (decimal).

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, it may have been overlay or segment call but appropriate bit is missing. Processing continues as for normal load.

LFN ALREADY IN USE PFA

Logical file name on next message line is already in use at this control point.

LFN GREATER THAN 7 CHARACTERS SKIPF,SKIPB

Logical file name parameter of a skip control card is too long.

LIST OF PROGRAMS IN FILE EDITLIB

LOAD 1LT

1LT called to load jobs from standard SCOPE 3 tape.

LOADER BY THIS NAME NOT IN SYSTEM. 1AJ

User request or control card selected loader but format or name was illegal.

LOADER CONTROL CARD OUT OF SEQUENCE LOD

NOGO or EXECUTE card before LOAD card.

LOADER NOT FOUND IN LIBRARY LOD

CM LOADER not found in library directory.

LOADER TABLES GARBAGE, OR OVERLAY SEQ ERROR LOADER

LOADER tables are threaded in a list below LOADER in CM. If tables are destroyed by user program (in normal mode only) and job terminates or secondary overlay is attempted without first generating corresponding primary overlay (of zero level), THREAD routine cannot execute properly.

LOADER, xxxx ERROR FLAG SET. LDR,2LA

Precedes all error messages issued by LDR or 2LA. (2LE overlay is called to issue message). xxxx is FATAL or NON-FATAL.

LOADPF ABORTED - SYSTEM ERROR xx LOADPF

LOADPF FINISHED LOADPF

LOADX 1LT

1LT called to load jobs from external tape.

LOC ARG ERROR LOC

Incorrect parameter, such as last word address greater than field length.

LOD SYSTEM WAIT LOD

To load LOADER from disk, LOD references FNT/FST entry for file SYSTEM (SSSSSSU if LOADER was modified by EDITLIB). If SYSTEM or SSSSSSU is associated with control point other than zero, EDITLIB is modifying system. LOD displays message and pauses until file returns to control point zero. This message can occur only when CM LOADER is disk resident.

LOD CANNOT FIND SYSTEM IN FNT          LOD

If program declares SYSTEM as common file, control point assignment of SYSTEM changes. LOD uses the FNT for SYSTEM when CM LOADER is read from disk. To determine a match with correct FNT entry, entry association with control point zero is verified.

LOST FILE (CY1)          CY1

FNT entry not defined by RESTART. Fatal dayfile message; job terminated.

LP nn NOT READY          2LP

Displayed on console when printer nn EST ordinal is not ready. When it becomes ready, printing will continue.

LP nn REJECT          2LP

Displayed on console when printer with nn EST ordinal rejects function code. No operator action.

LP nn XMSN PARITY ERROR          2LP

Console and system dayfile when printer with nn EST ordinal shows transmission parity error; no operator action. For local file, message also appears in job dayfile.

LPF ABORT - BAD ADDRESS          LPF

Address specified outside user field length.

LPF ABORT - NO ENTRY IN FNT FOR 0xxxx          LPF

Subdirectory xxxx or RBTC xxxx had no entry in file name table.

LPF ABORT - NO RBTC SPACE          LPF

LPF FINDS NO FNT SPACE          LPF

LPF - FINISHED LOADING          LPF

LPF STOPPED BY SYSTEM LPF

Operator should drop job.

L.SEQ LESS THAN 2CM WORDS APR

Diagnostic sequencer lacks sufficient table space in CM.

MDF NOT MASS STORAGE (1RC) 1RC

Memory dump file not mass storage. Fatal dayfile message; job terminated.

MEM ARG ERROR STATUS ALREADY COMPLETE MEM

Request made to MEM when completion bit was set in status word.

MESSAGE ARG ERROR MSG

Address given to MSG is out of user's field length. Job terminated.

MESSAGE FORMAT ERROR MSG

Character in message is 60 (octal) or over. Job terminated.

MESSAGE LIMIT MSG

Number of messages issued by job exceeds installation maximum. Job terminated.

MF DISPOSITION OF UNLABELED TAPE REQ

REQUEST card specifies multifile disposition of unlabeled tape.

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 first change occurred is decimal yyyy, counting first card as 0001.

MLRS CHANGED TO MAX BUFFER SIZE, xxxxxx 1MT

File with FET at xxxxxx contains zero MLRS field. This field was replaced with maximum buffer size for that file.

MODIFY PERMISSION NOT SET FOR RE-WRITE ON P.F. 4ES

Permanent file cannot be modified because correct password was not given on the ATTACH card. Job terminated.

MOUNT REEL SHOWN ON UNIT xx 1MF

MOVE ROUTINE FINDS CANT READ PROG. IN SYSTEM FILE EDITLIB

Fault in disk file used by EDITLIB (not expected to occur).

MT xx BLANK TAPE READ 1TF

Appears if no data is received from the channel after one second; exit procedures are performed.

MT xx ENTER VISUAL REEL NO. ON UNIT xx 4LB

Operator should respond with n.VRN,xxxxxx. Sticker number pasted on tape reel.

MT xx EOT CLO

Processing on magnetic tape xx is terminated because of end-of-reel condition or CLOSER request.

MT xx FET TOO SHORT 1RS

FET is less than 7 words on attempt to read S tape. Request is terminated with status DEVICE CAPACITY EXCEEDED and no data is transferred.

MT xx FILE POSITION UNCERTAIN 1PE

During write parity error recovery, tape backed to position which may destroy last good record written. Operator may drop job, or may type n.GO. and WPE recovery will proceed as though file were positioned properly.

MT xx LABEL INFO ERR IN FET 4LB

Numeric field in 33 label area of FET contains non-numeric data.

MT xx LABEL PARITY ERROR 4LB

Irrecoverable parity error while trying to read tape label.

MT xx LABEL UNRECOGNIZABLE CLO

Trailer label of MT xx is not EOF1 or EOV1 as expected. Operator action requested by additional message.

MT xx MLRS INVALID, 512 USED 1RS

MLRS field in FET word 7 exceeds device capacity; processing continues with capacity assumed (for execution) as the MLRS.

MT xx NO WRITE ENABLE 4LB,1WX,1WI,1WS

Message displayed until ring is inserted or job is dropped.

MT xx NOT READY 4LB,1TF,2TB,1RT,1RS,1WI,1WX,1WS,1PE

Message displayed until unit is readied or job is dropped.

MT xx PARITY ERROR 1RT,1RS

Read parity error irrecoverable; to continue type n.GO. or terminate with n.DROP.

MT xx POSSIBLE RECORD FRAGMENT — 1PE

Completing WPE recovery may leave data fragment greater than noise record. Operator may drop the job or type n.GO. If job is continued, attempt to recover WPE will continue as though file were positioned properly.

MT xx REJECT — 4LB,1TF,2TB, 1RT,1RS,1WI, 1WX,1WS,1PE

Tape unit xx cannot be connected. Compare xx entry with installation equipment select code. If possible, change equipment or unit select code; otherwise type n.DROP.

MT xx RESERVED — 4LB,1RT,1RS, 1WI,1WS,1WX, 1TF

Displayed until condition is corrected or job is dropped if attempt is made to connect to tape xx reserved by another channel.

MT xx WPE BAD SPOT — 1PE

Record was written successfully following skip bad spot, but parity error occurred in re-reading record preceding the skip bad spot. Operator may drop job, or type n.GO. and job will continue, but record with parity error will not be rewritten.

MT xx WPE RECOVERED — 1PE

Write parity error encountered and recovered; job continues.

MT xx WPE UNRECOVERED — 1PE

Write parity error could not be corrected by erasing and rewriting. Operator can continue or drop job.

MT xx WPE WRITE FILE MARK ERROR — 1PE

Unable to write file mark on external tape. Operator can type n.GO. or drop job.

MT xx XMSN PARITY ERROR 4LB,1TF,
2TB,1RT,1RS,
1WI,1WS,1WX

Transmission parity error in 6681 data channel converter.

MT xx labl BLOCK COUNT SHOULD BE xxxxxx, is xxxxxx. 4LB

Number of physical records written on tape with label identifier labl does not agree with number read.

MT xx labl CREATION DATE SHOULD BE xxxxxx, is xxxxxx. 4LB

Label labl does not contain specified creation date. First xxxxxx obtained from central memory; second xxxxxx from label. Operator may type in n.RECHECK. or n.GO. or n.DROP.

MT xx labl EDITION NUMBER SHOULD BE xx, IS xx. 4LB

Tape label labl does not contain expected expiration date. 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 labl EXPIRATION DATE SHOULD BE xxxxx, IS xxxxx. 4LB

Tape label labl 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 by typing n.DROP.

MT xx labl FILENAME READ WAS xxxx...xx 4LB

xxxx...xx is label name of first file of label identified as labl.

MT xx labl FILE NAME SHOULD BE xxx...xx, IS xxx...xx. 4LB

File name error in label identified as labl. First 20 characters are FET file name and second set are tape label's file name entry; n.GO. or n.DROP, or n.RECHECK. may be typed.

MTxx CHxx 1MT
NOT READY

Operator should make unit ready or drop job.

MTxx CHxx 1MT
PARITY ERROR

Irrecoverable parity error while writing L tape. Operator can continue or drop job.

MTxx CHxx 1MT
READ PARITY ERROR

Irrecoverable parity error while reading L tape. Operator can continue or drop job.

MTxx,CHxx 1MT
RESERVED

Magnetic tape xx is reserved by another channel. Message persists until condition is corrected or operator drops job.

MT xx CHxx 1MT
WPE RECOVERED

Write parity error encountered and recovered.

MTxx CHxx 1MT
XMSN PARITY ERROR

Transmission parity error on 6681 or 6684 data channel converter on channel xx.

MTR DEAD

Monitor did not respond to a monitor function issued by DSD. Notify a system analyst of this message.

MULTI-FILE DISPOSITION ON UNLABELED FILE REQ

Multi-file disposition should occur only on labeled files.

MULTIPLY DEFINED OUTPUT xxxxxxx LDR

LDR loaded this routine previously and cannot load it again.

N EQUAL INVALID CHAR COPYCR,COPYCF, COPYBR,COPYBF

Number of records/files to be copied is zero, a letter, or special character on the copy control card.

NAME GREATER THAN 7 CHARACTERS LDR,2LA

Name on a loader directive card is too large. (Usually caused by assembly or compilation errors.)

NEW ECS FE TOO SMALL, REQUEST xxxx (RESTART) RESTART

The ECS field length at restart time must be greater than or equal to that at checkpoint time. xxxx is the minimum field length, /1000 (octal), which should be on the restart job card.

NEXT 1EJ

Job name at a control point waiting for a job.

NEXT .CONTROL POINT CLEARED 1EJ

Appears in system dayfile when NEXT control point is cleared because the clear flag was set.

NO aa AVAILABLE

Job is waiting for equipment. Operator should turn on equipment of allocation style aa. If equipment is logically off but operative, type ONxx; xx is EST ordinal. If equipment is available but assigned to another control point, job must wait until equipment is released.

NO CORRECT PASSWORDS SUBMITTED PFA

NO CHECKPOINT TAKEN CKP,1RC

No checkpoint dump because job is: rolled out, RESPOND job, uses ECS, or uses READ-RELEASE. Informative dayfile message.

NO E OR N ON CONTROL CARD 5DA

RPACK control card does not specify E or N parameter for operator assignment of private disk pack. Job terminated.

NO EOR,EOF, OR EOI LOADPF

End-of-record, end-of-file, or end-of-information is missing. Loading terminates.

NO EQUIPMENT AVAILABLE WITH ALLOCATION TYPE FOR FILE LPF

File is skipped.

NO EXTEND PERMISSION PFE

NO FNT SPACE 2BP

UP parameter in FET is zero and no space is available in FNT to create requested entry; job terminated. If UP is nonzero, FNT full code (24) 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

At least one input file must be specified in the COPYN card.

NO OUTPUT FILE ON THE COPYN CONTROL CARD    COPYN

An output file must be specified in the COPYN control card.

NO PERMISSION GRANTED    PFA

NO PERMISSION TO ADD CYCLE    PFC

NO PERMISSION TO PURGE    PFP

NO PF DEVICE    PFA,PFC,<br>PFE,PFP

Installation error. Report this message to a system analyst.

NO PFM ACTION, WHILE UTILITY RUNS    PFA,PFC,<br>PFE,PFP

NO PRIVACY PROCEDURE    PFA

No privacy procedure specified on ATTACH, when required.

NO PROGRAM LIBRARY INPUT TO BE EDITED    EDITSYM

Correction input encountered but no old program library specified on EDITSYM control card. Run terminated.

NO PROGRAMS FOUND FOR SEGMENT    LOADER

Not one program requested by user call could be found.

NO RBR FOR THIS PACK    5DA

The system cannot find RBR for disk pack EST ordinal referenced in an ASSIGN, DEVADD, or UNLOAD type-in. Report this message to a system analyst.

NO RBR TABLE FOR PACK 1BT

Disk pack named by BLANK type-in is on and unloaded, but it does not have RBR table in central memory. Report this message to a system analyst.

NO READY EDITLIB

A READY card did not precede this control card.

NO REQUEST ENTRY OPE

No request entry was made for file with device type 240 (no FNT entry).

NO REQUEST ENTRY FOR MULTIFILE OPE

No request entry was made for specified multifile name (no FNT entry).

NO ROOM FOR NEW CYCLE PFC

Not possible to add new cycle to permanent file specified in previous line.

NO ROOM IN INDEX FOR NEW NAME IORANDM

Console, system and job dayfiles. WRITOUT macro called to write beginning of random access record named. Name is not in file index and no vacant slot is available. Job terminated.

NO SUCH FILE NAME 5DA

REQUEST or REMOVE card names a private pack that is not at control point. Job terminated.

NO SUCH PRIVATE PACK 5DA

This message is sent to the dayfile by 1AJ when a REQUEST or REMOVE card names a private pack that is not at the control point. The job is terminated.

NO TERMINATION FOUND LDR,2LA

Sent to dayfile when loader directive card is not terminated by period or right parenthesis.

NO TRANSFER ADDRESS LOADER

At completion of load, no program provides a transfer address. Error condition is overridden by NOGO. card or EXECUTE,ent. card. Job terminates.

NO WORD PAIRS ASSIGNED PFC

NON-EXISTENT RBR REQUESTED 1SX

Called by error code 05. RBR pointer in RBT is greater than number of RBR's in system.

NON-PERMANENT FILE CANNOT BE EXTENDED PFE
lfn

File with name lfn is not permanent file.

NOT AVAILABLE

Equipment indicated by ASSIGN type-in is off, already assigned, or does not exist.

NOT DISK PACK 5DA

DEVADD, UNLOAD, or ASSIGN type-in specified EST ordinal which is not disk pack. For ASSIGN in response to RPACK control card, system waits for operator to drop the job or try another assignment. Otherwise, job terminated.

NOT DISK PACK, OR OFF 1BT

Equipment named by BLANK type-in cannot be blank labeled because it is not disk pack or it is off to system.

| | | |
|---|---|---|
| NUMBER OF RECORDS GREATER THAN 777777B | Second parameter of a skip control card exceeded maximum number of allowable records. | SKIPF,SKIPB |
| ONE OR MORE OVERLAPPING CORRECTIONS | | UPDATE |
| ONLY 1 OVERLAY DESIGNATOR USED | Overlay card does not specify both primary and secondary levels. | LDR,2LA |
| ONLY ONE PARAMETER | Overlay loader directive has only one level parameter. | LDR,2LA |
| OPEN REEL CALLED ON UNOPENED OR CLOSE FILE | File must have been opened before OPEN REEL call. | OPE |
| OPERATOR DROP | The operator typed in n.DROP. to terminate job. | 1EJ |
| OVERLAY | | EDITLIB |
| OVERLAY CALL FROM RELOCATABLE | Relocatable program called for overlay load. Redundancy check to validate user call is illegal; once overlay loading is initiated, control is taken from user call and returned to called overlay. If file named in user call contains all absolute overlays and OVERLAY 0,0 has not been called, control will return to the user call. | LOADER |
| P.F. DEVICE DOWN | | PFA,PFC,<br>PFE,PFP |

P.F. NAME ALREADY IN SYSTEM PFC

Permanent file indicated on previous line already exists in system.

PACK FILE COUNT ALREADY 0 5DA

In response to REMOVE control card (with or without pname parameter) file was removed from private pack; however, when system tried to reduce file count in the FNT entry, count was already zero. Job terminated.

PACK LABEL NOT BLANK 5DA

Private disk pack label read should have been blank. If in response to DEVADD type-in, job terminates. Otherwise, system waits for operator to drop job or try another assignment.

PACK NOT PUBLIC AND EMPTY 5DA

Only disk packs with an EST status of 4000 can be unloaded by an UNLOAD type-in. Job terminated.

PACK STATUS NOT UNLOADED 1BT

BLANK type-in named disk pack that is on and in use as a public or private pack; therefore, it cannot be logically unloaded.

PACKED CARD LONGER THAN 13 WORDS EDITSYM

Input error; run terminated.

PAGE NO. EDITSYM

PARAM LIST TOO LONG (CKP) CKP

The value of n in the user parameter exceeds 42 (decimal). Job terminated.

WAITING FOR xx          REQ

Requested equipment is logically off; operator should turn equipment on or drop job.

WAITING FOR STORAGE          1BT,RST,
1RI,RFL

Occurs if MTR does not immediately grant request for ECS or CM storage. PP enters recall. Required storage not available. Job waits. Operator can create space by rolling out another job.

WAITING - RBT STORG          1SX

No empty chain members exist.

WAITING ON STORAGE          LDR

During program loading from tape, LDR request for additional storage has not been fulfilled.

WAITING PFM IDLE          DPF

DPF waiting for all permanent file activity to cease before dump execution begins. On B display only.

WARNING BLANK COMMON GREATER THAN PREVIOUS DECL          LOADER

Subsequent references to blank common in a set of programs (segment, overlay, or a file loaded as a result of control card) cannot exceed first allocation. Job does not terminate; however, as no reference to blank common is truncated, it is possible for a program to destroy itself.

WARNING NO MATCHING ENTRY FOR XFER          LOADER

When nonzero or nonblank entry point occurs in XFER table, all available entries in LOADER table are searched. If this warning results from EXECUTIVE or PROGRAM CALL card job is terminated; otherwise, nonfatal error bit is returned to user.

WPE UNRECOVERED. EOT FORCED, TYPE GO          xxxDMPQ

Permanent write error. Typing n.GO. will force EOT on bad reel and continuation reel will be assigned. Both reels must be read back in when queue is restored.

WRONG TAPE, GO WHEN CORRECTED (RESTART)          RESTART

Name on checkpoint tape leader record does not match name on RESTART card. Dayfile message requires operator action.

WRONG VRNO          5DA

Type-in response to RPACK control card specified correct pack name but wrong visual identification number. System waits for operator to drop job or try another assignment.

xx ASSD.          5DA

xx BLANKED

xxxxxxx CONTROL POINT DROPPED          1EJ

The system job named xxxxxxx was dropped and the control point cleared as a result of the operator entering n.DROP or n.KILL.

xxxx KILLED MTR          MTR

xxx NOT IN PPLIB          PP RESIDENT

Resident called to load overlay not named in library. Job terminated.

xx NOT READY          REQ

xx RESERVED REQ

xx XMSN PARITY ERROR REQ

1MT ARG ERROR 1MT

Routine for reading and writing L TAPES WAS LOADED WITH illegal function code, or function code it cannot process.

1ST OVERLAY CARD HAS NO FILE NAME LDR,2LA

First character of first parameter on initial overlay card is not alphabetic.

1ST OVERLAY CARD LACKS 0,0 LDR,2LA

First overlay card does not designate level (0,0) overlay.

1ST PARAMETER MAY NOT EQUAL ZERO LDR,2LA

Overlay level (0,0) may not have secondary overlay levels (0,1 is illegal).

8DX NOT IN LIB

DSD could not find one of its overlays in PP library. Notify system analyst of this message.

9DM ABORTED

PP routine DSD calls to load mass storage resident overlays terminated because it is not in PP library; or it encountered parity error in loading overlay. Operator should dump last 1000(octal) locations of all PP's and first 12000(octal) words of CM. Notify system analyst of this message.

63 FILES ALREADY ON PACK 5DA

REQUEST card called for new file on private pack, but pack has maximum of 63 files already. Job terminated.

xxx ERRORS IN UPDATE INPUT UPDATE

*COPYLAB INCORRECT COPYCR,COPYCF
COPYBR,COPYBF

*COPYLAB control card is misspelled or next record in job stream is not *COPYLAB as expected.

*EC STATS xx aaaaaaaaaaaa bbbbbbbbbbbb* EC1

a...a last 12 octal digits of word xx. b...b last 12 octal digits of word xx+1. Gives ECS statistics accumulated since last call. Dayfile message.

*EC STATS DISCONTINUED.* EC1

Dayfile message if ECS is off or delay interval 7777.

*EC STATS INTERVAL xxxx SECONDS.* EC1

Dayfile message issued each time delay interval xxxx is changed.

*GO OR DROP EC STATS DELAY BEING SET.* EC1

Dayfile message informs operator of impending change in delay interval and requests approval.

*NO ECS, OR ECS TURNED OFF.* EC1

Dayfile message. EC1 cannot find EST entry for ECS or ECS is turned off.

*NOT AN ECS SYSTEM.* EC1

Dayfile message. EC1 initiated in system that does not define ECS.

**DUPLICATE PARAMETER cc PF MACROS

**DUPLICATE PARAMETER ON PF CONTROL CARD PFCCP

| | |
|---|---|
| **FOLLOWING PF CONTROL CARD IN ERROR | PFCCP |
| **ILLEGAL OPTION cc = cccc | PF MACROS |
| **ILLEGAL OPTION ON PF CONTROL CARD | PFCCP |
| **MISSING NAME FOR OPTION cc = | PF MACROS |
| **PARAMETER VALUE EXCEEDS NINE CHARS | PFCCP |
| **PERMANENT FILE NAME EXCEEDS xx CHARS | PFCCP,MACROS |
| **PW OPTION HAS TOO MANY PARAMETERS | PFCCP |
| ***ADDFILE FIRST CARD MUST BE *DECK OR *COMDECK BUT WAS:.... | UPDATE |
| ***ADDFILE INVALID FROM *READ FILE | UPDATE |
| ***ADDFILE INVALID WITH Q-OPTION | UPDATE |
| ***ABOVE CARD ILLEGAL DURING CREATION RUN | UPDATE |
| ***CARD LENGTH ERROR ON OLD PROGRAM LIBRARY | UPDATE |
| ***CARD NUM ZERO OR INVALID CHAR IN NUM FIELD | UPDATE |
| ***CONTROL CARD INVALID OR MISSING | UPDATE |
| ***DECK NAMES SEPARATED BY PERIOD IN WRONG ORDER | UPDATE |
| ***DUPLICATE DECK NAME | UPDATE |
| ***DUPLICATE IDENT NAME | UPDATE |
| ***ERROR xxxxxxx DECK DOES NOT EXIST | UPDATE |
| ***FILE NAME LONGER THAN 7 CHARACTERS | UPDATE |
| ***IDENT xxxxxxx UNKNOWN | UPDATE |
| ***IDENTIFIER LONGER THAN 7 CHARACTERS | UPDATE |
| ***IDENTIFIERS SEPARATED BY PERIOD IN WRONG ORDER | UPDATE |
| ***INVALID NUMERIC FIELD | UPDATE |
| ***NO DECK NAME | UPDATE |

| | |
|---|---|
| ***NO SUCH COMMON DECK | UPDATE |
| ***NOT ALL MODS WERE PROCESSED | UPDATE |
| ***NULL ADDFILE | UPDATE |
| ***ON THE ABOVE CARD THE FIRST LIMIT EXCEEDS TERMINAL LIMIT | |
| ***OR A REFERENCE IS MADE TO DECK NOT MENTIONED ON COMPILE CARD<br>Second line of this message appears only if Q-option is in effect. | |
| ***PREMATURE END OF RECORD ON OLD PROGRAM LIBRARY | UPDATE |
| ***RESERVED FILE NAME | UPDATE |
| ***THESE MAY BE MODS TO DECKS NOT MENTIONED ON COMPILE CARD OR AN INCOMPLETE ADDFILE | UPDATE |
| ***UNKNOWN IDENTIFIER | UPDATE |
| ***xxx ERRORS IN INPUT: NEWPL, COMPILE SOURCE SUPPRESSED | UPDATE |
| ****DEVICE CAPACITY EXCEEDED****<br>Tape physical record size is greater than buffer on read or write or physical record size is greater than data format declared on REQUEST control card. | COPYCR,COPYCF,<br>COPYBR,COPYBF |

# MAGNETIC TAPE I/O

This appendix describes in detail the operations which take place on 1/2 inch magnetic tape files during file action requests, data functions, and position functions.

DEFINITIONS

Physical Record: Continuous data on tape between two successive inter-record gaps.

Noise Record: A physical record of 6 characters or less (or as defined by IP.NOISE) is considered to be noise on SCOPE and X tapes and is discarded without notification to the user. On S and L tapes, noise record size is set by the installation parameter IP.NOISE and will not exceed $62_{10}$ bytes (124 characters).

Physical Record Unit (PRU): Some data formats of tapes acceptable to SCOPE are defined in terms of a maximum size physical record. A PRU is a physical record of the maximum size for that format. Operations terminate with an error indication if tapes have physical records greater than the maximum size allowed. A short PRU is a physical record of size less than the maximum. A zero length PRU is a special physical record containing no data; it is 8 characters in length and contains a level number, if appropriate. Zero length PRU's appear only on SCOPE tapes and on binary X tapes.

| | | | |
|---|---|---|---|
| PRU sizes: | SCOPE Standard Tape | Binary | 512 CM words |
| | | Coded | 128 CM words |
| | X Tape | Binary | 512 CM words |
| | | Coded | 136 characters |
| | S Tape | | 2 to 5120 characters |
| | L Tape | | 2 to n characters (n is size of user buffer) |

Logical Record: A logical record is written on a standard SCOPE tape or binary X tape as one or more PRU's, the last of which is a short or zero length PRU. A zero-length PRU is written if the logical record size is a multiple of the PRU size. The term logical record is used to mean either one physical record or the above mentioned multiple of PRU's. The declared data format of the tape defines precisely the exact format of the logical record. The logical record definition is normally considered a block by the user, e.g., COBOL BLOCK CONTAINS clause.

End of Line Terminator: A 12-bit zero byte in the low order byte of a CM word is the end-of-line for coded records on SCOPE standard or X tapes, but is not applicable to S or L tapes. During conversion from display code to BCD on SCOPE standard tapes, the zero byte terminator becomes external 1632. If the file is destined for an X tape, SCOPE writes each line as a 136-character physical record with blank fill and converts the zero byte to blanks.

## TAPE FILE STRUCTURE

### Label Formats

SCOPE standard, S and L tape files may be unlabeled (assumed if not declared otherwise) or contain SCOPE standard labels (E or N declaration on the REQUEST card). X tapes cannot be labeled.

### Data Formats

SCOPE Standard: (Assumed if no other declaration is given). A logical record is one or more PRU's. A level number is appended to the last PRU. A PRU is 5120 characters in binary mode or 1280 characters in coded mode. A short PRU is 10n + 8 characters where $0 \leq n \leq 511$ in binary mode or $0 \leq n \leq 127$ in coded mode; the last eight characters, containing the logical record level number, are not transmitted to or from the user's CM buffer. Physical records must contain a multiple of ten characters.

S (Stranger Tapes): A logical record is equivalent to a physical record. Physical records contain multiples of two characters with a maximum of 5120 for a binary or a coded file. No level numbers are appended to data, and no line terminators are recognized in data.

L (Long Record Stranger Tapes): Data format for the L tape is identical to the S tape except the size of a physical record is not restricted. The method of conversion depends on the data channel converter in use. With the Control Data 6681 Data Channel Converter conversion takes place in central memory; with the 6684, conversion is done by the hardware. L tapes should be specified only when the files are being processed by specially designed routines which can also handle "long" records. Routines which cannot handle long records can find an indication in the FET in the device field that the L tape format is being used. Care should also be taken not to write records that are longer than the length of tape from the end of reel reflective spot to the physical end of tape. In such a case 1MT is unable to detect end of tape before it writes off the end of the reel.

X (External Tapes): Retained for compatibility with tapes created under SCOPE 2.0 and earlier systems. For binary files, a logical record is one or more PRU's terminated by a short or zero-length PRU; no level number is appended. Physical records contain exact multiples of 10 characters. For coded files, a logical record is a physical record of up to 136 characters. If a line terminator occurs before 136 characters are written, remaining characters are blank. If 136 characters occur before a line terminator, the last four characters in the last CM word are lost. No level number is appended.

## END POINT PROCESSING

### End-of-File (Tape Mark) Procedures

A physical end-of-file (tape mark) can appear on a SCOPE standard tape only as part of a label, the WRITEF function writes a zero-length logical record with level 17. End-of-file marks may be written on X , L and S tapes with the WRITEF function. On labeled tapes, end-of-file marks are written as part of the label. On an input tape, the I/O system determines whether it is part of the label. For a SCOPE standard tape, the end-of-file mark indicates a label, since trailer labels are always written on SCOPE standard tapes (labeled or unlabeled). X tapes cannot be labeled. For S or L tapes, SCOPE determines whether a labeled tape has been declared. If so the I/O system determines if a label record is next. If so, end-of-reel or end-of-information procedures are performed. If the next record is not a label record or if the tape is not labeled, the end-of-file mark is treated the same as a zero-length record of level 17; the end-of-file bit is set in the FET status field and the function encountered is completed normally. Since X tapes cannot be labeled, when an end-of-file mark is read, the end-of-file bit is set in the FET status field and the function is completed.

## End-of-Reel Procedures

End-of-reel procedures for an output tape are performed when the end-of-reel reflective spot is encountered according to the following table.

| SCOPE Standard | y | y | y | y | n | n | n | n |
|---|---|---|---|---|---|---|---|---|
| Labeled | n | y | n | y | n | n | y | y |
| UP bit set | n | n | y | y | n | y | n | y |
| Backspace over last physical record | x | x |  |  |  |  |  |  |
| Write EOV trailer label | x | x |  |  |  |  | x |  |
| Write 4 tape marks |  |  |  |  | x |  |  |  |
| Rewind unload reel | x | x |  |  | x |  | x |  |
| Locate next reel | x | x |  |  | x |  | x |  |
| Write header label |  | x |  |  |  |  | x |  |
| Continue function | x | x |  |  | x |  | x |  |
| Set end-of-reel bit in FET |  |  | x | x |  | x |  | x |
| Exit to user |  |  | x | x |  | x |  | x |

y = yes

n = no

End-of-reel procedures for an input tape are performed when an EOV label is encountered on a labeled or SCOPE standard tape or when the end-of-reel reflective spot is encountered on an unlabeled tape. If the UP bit is not set the next reel is obtained, label checking is performed if the tape is labeled and the function continues normally on the next reel.

WARNING: When the UP bit is set and control is given to the user, the zero length PRU may not have been written if the longest record was an exact multiple of the PRU size. Any further writing may appear as part of the previous record. The user is responsible and should terminate his file with an end-of-file mark or some other action which will permit proper reading of the file.

## End of Information Procedures

For an output tape, before backward motion takes place, an EOF trailer label is written on labeled or SCOPE standard tapes or four end-of-file marks are written on unlabeled tapes.

For an input tape, end of information is defined only for labeled and SCOPE standard tapes when the EOF trailer label is encountered. The end-of-information bit is set in the FET as long as the file remains positioned at the end of information. End of information for unlabeled tapes is not defined. It is the user's responsibility to determine by some other means when he has processed to the end of information.

## DEFINITION OF I/O REQUESTS

The following tables specify the logical occurrence of events during various I/O operations. In every case of conversion between BCD and Display Code, code is not converted if the tape is connected to a 6684 data channel converter.

| READ | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 1. Exit if not enough room in buffer for one maximum size physical record. | x | x | x | x | | | | |
| 2. Exit if not enough room in buffer for MLRS words. | | | | | x | x | x | x |
| 3. Read one physical record into PP. | x | x | x | x | x | x | | |
| 4. Read one physical record into CM. | | | | | | | x | x |
| 5. If physical record exceeds maximum allowable return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | x | x | | | | |
| 6. If physical record exceeds maximum logical record size, return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 7. If end-of-file mark was read, perform end-of-file mark procedures. | x | x | x | x | x | x | x | x |
| 8. If noise records encountered, go to 3. | x | x | x | x | x | x | x | x |
| 9. If parity error, perform parity procedures. | x | x | x | x | x | x | x | x |
| 10. If end-of-tape reflective spot was encountered and tape is unlabeled, perform end-of-reel procedures. | | | x | x | x | x | x | x |
| 11. If short PRU was read, strip level number. | x | x | | | | | | |
| 12. If zero length PRU was read, go to 21. | x | x | x | | | | | |
| 13. When 6681 present, convert data in PP from BCD to Display Code. | | x | | x | | x | | |
| 14. When 6681 present, convert data in CM from Ext. BCD to Display Code. | | | | | | | | x |
| 15. Convert 1632 line terminator to 0000. | | x | | | | | | |
| 16. Transmit data to CM. | x | x | x | x | x | x | | |
| 17. Update IN. | x | x | x | x | x | x | x | x |
| 18. Fetch OUT from CM. | x | x | x | x | | | | |

Note: Event 6 above is a change in specification from SCOPE 3.1. Currently, if a long record is encountered, the information past the PRU size is discarded without notification to the user.

| READ (CONT'D) | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 19. Place in word 7 of FET the number of unused bits in the last data word. | | | | | x | x | x | x |
| 20. If full PRU go to 1. | x | x | x | x | | | | |
| 21. If last record was level 17 of tape mark, set end-of-file status. | x | x | x | x | x | x | x | x |
| 22. Set end-of-record in status field of FET and exit. | x | x | x | x | x | x | x | x |

| READN | | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|
| 1. | Fetch size of MLRS from word 7 of FET. | x | x | x | x |
| 2. | Exit if not enough room in the circular buffer for one logical record plus header word. The buffer size must > ℓ(record) + 1 (header) to avoid OUT=IN when buffer is full. | x | x | x | x |
| 3. | Read one physical record into PP. | x | x | | |
| 4. | Read one physical record into CM. | | | x | x |
| 5. | If physical record exceeds maximum allowable, return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | | |
| 6. | If logical record exceeds MLRS, return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | | | x | x |
| 7. | If end-of-file (tape mark) was read, perform end-of-file mark procedures. Go to 18. | x | x | x | x |
| 8. | If noise records encountered, go to 3. | x | x | x | x |
| 9. | If parity error, perform parity procedures. | x | x | x | x |
| 10. | If end-of-tape reflective spot was encountered and tape is unlabeled, perform end-of-reel procedures. | x | x | x | x |
| 11. | When 6681 present, convert data in PP from BCD to Display Code. | | x | | |
| 12. | When 6681 present, convert data in CM from BCD to Display Code. | | | | x |
| 13. | Transmit data to CM. | x | x | | |
| 14. | Update IN in PP memory. | x | x | x | x |
| 15. | Place in buffer header word, length of record and number of unused bits in last data word. | x | x | x | x |
| 16. | Update IN | x | x | x | x |
| 17. | Fetch OUT. | x | x | x | x |
| 18. | If last record was tape mark, set end-of-file status and exit. | x | x | x | x |
| 19. | Go to 2. | x | x | x | x |

| READSKP | | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|---|
| 1. | Read one physical record into PP. | x | x | x | x | x | x | | |
| 2. | If physical record exceeds maximum allowable (i.e. 512 CM words, etc.), return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | x | x | x | x | | |
| 3. | Read one physical record directly from tape to CM buffer, stopping without error when available buffer space is full. | | | | | | | x | x |
| 4. | If end-of-file (tape mark) was read, perform end-of-file mark procedures. | x | x | x | x | x | x | x | x |
| 5. | If noise records encountered, go to 1. | x | x | x | x | x | x | x | x |
| 6. | If parity error, perform parity procedures. | x | x | x | x | x | x | x | x |
| 7. | If end-of-tape reflective spot was encountered and tape is unlabeled, perform end-of-reel procedures. | | | x | x | x | x | x | x |
| 8. | If short PRU was read, strip level number. | x | x | | | | | | |
| 9. | If zero length PRU was read, go to 10. | x | x | | | | | | |
| 10. | When 6681 present, convert data in PP from BCD to Display Code. | | x | | x | | x | | |
| 11. | When 6681 present, convert data in CM from BCD to Display Code. | | | | | | | | x |
| 12. | Convert 1632 line terminator to 0000. | | x | | | | | | |
| 13. | Transmit data to CM. If entire record does not fit in circular buffer, stop without error at buffer full. | x | x | x | x | x | x | | |
| 14. | Place in word 7 of FET the number of unused bits in the last data word. | | | | | x | x | x | x |
| 15. | Update IN | x | x | x | x | x | x | x | x |
| 16. | Fetch OUT from CM. | x | x | x | x | | | | |
| 17. | If any unused space in circular buffer, go to 1. | x | x | x | | | | | |
| 18. | If last record was full PRU, set n=1 and proceed to SKIPF. | x | x | x | | | | | |
| 19. | If L is less than 17 set L=0. | | | x | x | x | x | x | x |
| 20. | If record was end of file mark (tape mark), assume level= 17. | | | x | x | x | x | x | x |

| READSKP (CONT'D) | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 21. If level number is less than L, set n = 1 and proceed to SKIPF. | x | x | x | x | x | x |  |  |
| 22. If level number is less than L, set n = 1 and skip down tape to first end-of-file mark (tape mark). |  |  |  |  |  |  | x | x |
| 23. If last record was level 17, set end-of-file status and exit. | x | x | x | x | x | x | x | x |
| 24. If last record was not level 17, return end-of-record status and exit. | x | x | x | x | x | x | x | x |

| RPHR | | Standard Binary | Standard Coded | X Binary | X Coded |
|---|---|---|---|---|---|
| 1. | Set OUT = IN. | x | x | x | x |
| 2. | Exit if not enough room in buffer for one maximum size physical record. | x | x | x | x |
| 3. | Read one physical record into PP. | x | x | x | x |
| 4. | If physical record exceeds maximum allowable, return error status DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | x | x |
| 5. | If end-of-file mark was read, perform end-of-file mark procedures. | x | x | x | x |
| 6. | If noise records encountered, go to 3. | x | x | x | x |
| 7. | If parity error, perform parity procedures. | x | x | x | x |
| 8. | If end-of-tape reflective spot was encountered and tape is unlabeled, perform end-of-reel procedures. | | | x | x |
| 9. | If zero length PRU was read, go to 13. | x | x | x | |
| 10. | Transmit data to CM. | x | x | x | x |
| 11. | Update IN. | x | x | x | x |
| 12. | If last record was level 17 or tape mark, set end-of-file status. | x | x | x | x |
| 13. | Exit. | x | x | x | x |

Note: Event 4 above is a change in specification from SCOPE 3.1. Currently, if a long record is encountered, the information past the PRU size is discarded without notification to the user.

| WRITE | | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|---|
| 1. | Exit if not full PRU. | x | x | x | | | | | |
| 2. | If data from OUT to IN exceeds maximum logical record size from FET, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 3. | Fetch number of unused bits in last data word from FET and adjust record length. If record length constitutes a noise record, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 4. | Read one PRU of data starting at OUT from CM to PP. | x | x | x | x | | | | |
| 5. | Read data contained between OUT and IN from CM to PP. Adjust by unused bit count. | | | | | x | x | | |
| 6. | When 6681 present, convert Display Code to BCD in PP memory. | | x | | x | | x | | |
| 7. | When 6681 present, convert from Display Code to BCD in CM. | | | | | | | | x |
| 8. | Convert zero byte line terminator to 1632. | | x | | | | | | |
| 9. | Convert zero byte line terminator to blanks. If less than 136 characters, fill record with blanks to 136. | | | | x | | | | |
| 10. | Write record to tape. | x | x | x | x | x | x | | |
| 11. | Write,from CM to tape, data contained between OUT and IN, adjusted by unused bit count. | | | | | | | x | x |
| 12. | When 6681 present, convert data in CM buffer back to Display Code. | | | | | | | | x |
| 13. | If parity error, perform parity procedures. | x | x | x | x | x | x | x | x |
| 14. | If end-of-tape reflective spot, perform end-of-reel procedures. | x | x | x | x | x | x | x | x |
| 15. | Update OUT. | x | x | x | x | x | x | x | x |
| 16. | Exit. | | | | | x | x | x | x |
| 17. | Fetch IN from CM. | x | x | x | x | | | | |
| 18. | Go to 1. | x | x | x | x | | | | |

In events 1 and 5, a PRU for coded X tapes is 136 characters or less than 136 terminated by zero bytes.

WRITER

| | | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|---|
| 1. | If IN = OUT exit. | | | x | x | x | x | x | x |
| 2. | If PRU not full, insert level number in PP buffer. | x | x | | | | | | |
| 3. | If data from OUT to IN exceeds maximum logical record size from FET, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 4. | Fetch number of unused bits in last data word from FET and adjust record length. If record length constitutes a noise record, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 5. | Read one PRU of data starting at OUT or data between OUT and IN, whichever is smaller, from CM to PP. | x | x | x | x | | | | |
| 6. | Read data between OUT and IN from CM to PP. Adjust by unused bit count. | | | | | x | x | | |
| 7. | When 6681 is present, convert Display Code to BCD in PP memory. | | x | | x | | x | | |
| 8. | When 6681 is present, convert Display Code to BCD in CM. | | | | | | | | x |
| 9. | Convert zero byte line terminator to 1632. | | x | | | | | | |
| 10. | Convert zero byte line terminator to blanks. If less than 136 characters, fill record to 136 with blanks. | | | | x | | | | |
| 11. | If IN = OUT, write zero length record. Go to 12. | x | x | x | | | | | |
| 12. | Write record to tape. | x | x | x | x | x | x | | |
| 13. | Write, from CM to tape, data contained between OUT and IN, adjusted by unused bit count. | | | | | | | x | x |
| 14. | When 6681 is present, convert data in CM buffer to Display Code. | | | | | | | | x |
| 15. | If parity error, perform parity procedure. | x | x | x | x | x | x | x | x |
| 16. | If end-of-tape reflective spot, perform end-of-reel procedures. | x | x | x | x | x | x | x | x |
| 17. | Update OUT | x | x | x | x | x | x | x | x |
| 18. | Exit. | | | | | x | x | x | x |
| 19. | If full PRU not written, exit. | x | x | x | x | | | | |

WRITER (CONT'D)

| | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 20. Go to 1. | x | x | x | x | | | | |

In events 1 and 5, a PRU for coded X tapes is 136 characters or less than 136 terminated by a zero byte.

| WRITEF | | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|---|
| 1. | If no data from OUT to IN, go to 23. | x | x | x | | | | | |
| 2. | If no data from OUT to IN, go to 19. | | | | x | x | x | x | x |
| 3. | If not full PRU, insert 0 level number. | x | x | | | | | | |
| 4. | If data from OUT to IN exceeds maximum logical record size, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 5. | Fetch number of unused bits in last data word from FET and adjust record length. If record length constitutes a noise record, return DEVICE CAPACITY EXCEEDED and perform error procedures. | | | | | x | x | x | x |
| 6. | Fetch one PRU of data starting at OUT or data between OUT and IN, whichever is smaller, from CM to PP. | x | x | x | x | | | | |
| 7. | If OUT = IN insert 4 zero bytes in the PP buffer. | | | x | | | | | |
| 8. | Read data contained between OUT and IN from CM to PP. Adjust by unused bit count. | | | | | x | x | | |
| 9. | When 6681 present, convert Display Code to BCD in PP memory. | | x | | x | | x | | |
| 10. | When 6681 present, convert Display Code to BCD in CM. | | | | | | | | x |
| 11. | Convert zero byte line terminator to 1632. | | x | | | | | | |
| 12. | Convert zero byte line terminator to blanks. If less than 136 characters, fill record to 136 with blanks. | | | | x | | | | |
| 13. | Write record to tape. | x | x | x | x | x | x | | |
| 14. | Write, from CM to tape, the data contained between OUT and IN, adjusted by unused bit count. | | | | | | | x | x |
| 15. | When 6681 present, convert data in CM buffer to Display Code. | | | | | | | | x |
| 16. | If parity error, perform parity procedures. | x | x | x | x | x | x | x | x |
| 17. | If end-of-tape reflective spot, perform end-of-reel procedures. | x | x | x | x | x | x | x | x |
| 18. | Update OUT. | x | x | x | x | x | x | x | x |
| 19. | Write end-of-file mark and exit. | | | | | x | x | x | x |

| WRITEF (CONT'D) | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 20. If full PRU not written, write zero length level 17 record and exit. | x | x | | | | | | |
| 21. If full PRU not written, write file mark and exit. | | | x | x | | | | |
| 22. Go to 3. | x | x | x | x | | | | |
| 23. If last operation was WRITE, write zero length PRU. | x | x | x | | | | | |
| 24. Go to 17. | x | x | x | | | | | |

| | WRITEN | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|
| 1. | If OUT = IN, exit. | x | x | x | x |
| 2. | Fetch header word from OUT. Set PPOUT = OUT + 1. Set PPIN = PPOUT + no. of CM words in logical record. If PPIN has passed IN, exit. | x | x | x | x |
| 3. | If data from PPOUT to PPIN exceeds maximum physical record size, return DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | | |
| 4. | Adjust record length by number of unused bits in last data word (from header word). If record length constitutes a noise record, return DEVICE CAPACITY EXCEEDED and perform error procedures. | x | x | x | x |
| 5. | Fetch data contained between PPOUT and PPIN. Adjust by unused bit count. | x | x | | |
| 6. | When 6681 present, convert Display Code to BCD in PP memory. | | x | | |
| 7. | When 6681 present, convert Display Code to BCD in CM. | | | | x |
| 8. | Write record to tape. | x | x | | |
| 9. | Write, from CM to tape, the data contained between OUT and IN, adjusted by unused bit. | | | x | x |
| 10. | When 6681 present, convert data in CM buffer back to display code. | | | | x |
| 11. | If parity error, perform parity procedures. | x | x | x | x |
| 12. | If end-of-tape reflective spot, perform end-of-reel procedures. | x | x | x | x |
| 13. | Update PPOUT. | x | x | | |
| 14. | Update OUT. Fetch IN. Go to 1. | x | x | x | x |

WPHR

| | Standard Binary | Standard Coded | X Binary | X Coded |
|---|---|---|---|---|
| 1. If IN = OUT, exit. | x | x | x | x |
| 2. If more than 512 words in buffer, return DEVICE CAPACITY EXCEEDED to FET. | x | x | x | x |
| 3. Fetch data from OUT to IN, or 512 words from OUT, whichever is smaller. | x | x | x | x |
| 4. Write record to tape. | x | x | x | x |
| 5. If parity error, perform parity procedures. | x | x | x | x |
| 6. If end-of-tape reflective spot, perform end-of-reel procedures. | x | x | x | x |
| 7. Update OUT and exit. | x | x | x | x |

| SKIPF | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 1. If n = 0, set n = 1. | x | x | x | x | x | x | x | x |
| 2. If L is less than 17, interpret L as being equal to 0. | | | x | x | x | x | x | x |
| 3. Read a physical record. | x | x | x | x | x | x | x | x |
| 4. If noise record encountered, go to 3. | x | x | x | x | x | x | x | x |
| 5. If end-of-tape reflective spot encountered and tape is not labeled, perform end-of-reel procedures. | | | x | x | x | x | x | x |
| 6. If record is full PRU, go to 3. | x | x | x | | | | | |
| 7. If end-of-file mark encountered and tape is not labeled, assume level number equals 17. | | | x | x | x | x | x | x |
| 8. If record is not end-of-file mark, assume level number equals 0. | | | x | x | x | x | x | x |
| 9. If end-of-file mark encountered and tape is labeled, perform end-of-file procedures. | x | x | | | x | x | x | x |
| 10. If level number is less than L, go to 3. | x | x | x | x | x | x | x | x |
| 11. Subtract 1 from n. If n ≠ 0, go to 3. | x | x | x | x | x | x | x | x |
| 12. Return end-of-record to status. If last level number was 17 return end-of-file to status. Exit. | x | x | x | x | x | x | x | x |

Events 2, 7, and 8 differ from SCOPE 3.1. Currently L is assumed equal to zero for X tapes.

Event 4 is not done in SCOPE 3.1.

| SKIPB | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 1. If n = 0, set n = 1. | x | x | x | x | x | x | x | x |
| 2. If L is less than 17, interpret L as being equal to 0. | | | x | x | x | x | x | x |
| 3. If reel is at beginning of data (either physical load point or zero physical record count), set beginning of information and exit. | x | x | x | x | x | x | x | x |
| 4. Read one physical record backwards. | x | x | x | x | x | x | x | x |
| 5. If noise record encountered, go to 4. | x | x | x | x | x | x | x | x |
| 6. If record was full PRU, go to 3. | x | x | x | x | | | | |
| 7. If this is first read backwards, go to 3. | x | x | x | | | | | |
| 8. Position forward over short PRU. | x | x | x | | | | | |
| 9. If end-of-file mark encountered, assume level number = 17. If not end-of-file mark, assume level number = 0. | | | x | x | x | x | x | x |
| 10. If level number is less than L, go to 3. | x | x | x | x | x | x | x | x |
| 11. Subtract 1 from n. If n is not equal to zero, go to 3. | x | x | x | x | x | x | x | x |
| 12. Exit. | x | x | x | x | x | x | | |

In SCOPE 3.1 the beginning of information is not set as specified in event 3.

Event 5 is not performed in SCOPE 3.1.

Event 9 differs from SCOPE 3.1; the system level numbers are ignored for files other than standard files.

BKSP

The BKSP function is identical to SKIPB with n = 1 and L = 0.

| BKSPRU | Standard Binary | Standard Coded | X Binary | X Coded | S Binary | S Coded | L Binary | L Coded |
|---|---|---|---|---|---|---|---|---|
| 1. If at load point or PRU count = 0, set beginning of information in FET and exit. | x | x | x | x | x | x | x | x |
| 2. Backspace one physical record. | x | x | x | x | x | x | x | x |
| 3. Subtract 1 from n. If n not equal to 0, go to 1. | x | x | x | x | x | x | x | x |
| 4. Exit. | x | x | x | x | x | x | x | x |

SCOPE 3.1 does not set beginning of information as specified in event 1.

# PRINT FILE CONVENTIONS J

Files with a print disposition (including OUTPUT) and files assigned to a printer, must adhere to specific format rules as follows:

1. All characters must be in display code.
2. The end of a print line must be indicated by a zero byte in the lower 12 bits of the last central memory word of the line. Any other unused characters in the last word should be filled with display code blanks ($55_8$). For example, if the line has 137 characters (including carriage control), the last word would be aabbccddeeff550000 in octal; the letters represent the last seven characters to be printed in the line. No line should be longer than 137 characters.
3. Each line must start in the upper 6 bits of a CM word.
4. The first character of a line is the carriage control, which specifies spacing as shown in the following table. It will never be printed, and the second character in the line will appear in the first print position; therefore a maximum of 137 characters can be specified for a line, 136 is the number of characters that will be printed. All characters apply to both the 501 and the 512 unless they are specifically designated otherwise.

Carriage Control Characters

| Character | Action Before Printing | Action After Printing |
|---|---|---|
| A | Space 1 | Eject to top of next page† |
| B | Space 1 | Skip to last line of page† |
| C | Space 1 | Skip to channel 6 |
| D | Space 1 | Skip to channel 5 |
| E | Space 1 | Skip to channel 4 |
| F | Space 1 | Skip to channel 3 |
| G | Space 1 | Skip to channel 2 |
| H | Space 1 | Skip to channel 1 (501)<br>Skip to channel 11 (512) |
| I | Space 1 | Skip to channel 7 (512) |
| J | Space 1 | Skip to channel 8 (512) |
| K | Space 1 | Skip to channel 9 (512) |
| L | Space 1 | Skip to channel 10 (512) |
| 1 | Eject to top of next page | No space† |
| 2 | Skip to last line on page | No space† |
| 3 | Skip to channel 6 | No space |

† The top of a page is indicated by a punch in channel 8 of the carriage control tape for the 501 printer and channel 1 for the 512 printer. The bottom of page is channel 7 in the 501 and 12 in the 512.

| Character | Action Before Printing | Action After Printing |
|---|---|---|
| 4 | Skip to channel 5 | No space |
| 5 | Skip to channel 4 | No space |
| 6 | Skip to channel 3 | No space |
| 7 | Skip to channel 2 | No space |
| 8 | Skip to channel 1 (501) | No space |
|  | Skip to channel 11 (512) | No space |
| 9 | Skip to channel 7 (512) | No space |
| X | Skip to channel 8 (512) | No space |
| Y | Skip to channel 9 (512) | No space |
| Z | Skip to channel 10 (512) | No space |
| + | No space | No space |
| 0 (zero) | Space 2 | No space |
| - (minus) | Space 3 | No space |
| blank | Space 1 | No space |

When the following characters are used for carriage control, no printing takes place. The remainder of the line will not be printed.

| | |
|---|---|
| Q | Clear auto page eject |
| R | Select auto page eject |
| S | Clear 8 vertical lines per inch (512) |
| T | Select 8 vertical lines per inch (512) |
| PM<br>(col 1-2) | Output remainder of line (up to 30 characters) on the B display and the dayfile and wait for the JANUS typein /OKuu. For files assigned to a printer, n.GO. must be typed to allow the operator to change form or carriage control tapes. |
| any other | Acts as a blank |

Any pre-print skip operation of 1, 2 or 3 lines that follows a post skip operation will be reduced to 0, 1 or 2 lines.

The functions S and T should be given at the top of a page; in other positions, S and T can cause spacing to be different from the stated spacing. Q and R need not be given at the top of the page as each will cause a page eject before performing its function.

# GLOSSARY OF SCOPE TERMS

Active file

A file immediately available to the system because of its location on a mass storage device, tape, or extended core storage. Any file with an entry in the file name table is an active file.

Allocatable device

A mass storage device such as a disk, drum, or extended core storage which can be shared by more than one job.

Central memory resident (CMR)

Variable length low core area of central memory reserved for tables, pointers, and subroutines necessary for operation of the SCOPE system.

Central program control (CPC)

A SCOPE subroutine which the system loads into the field length of every user program that contains a file action request or a system action request. Its function is to communicate central processor requests to the peripheral processors by way of the system monitor routine.

CIO

A SCOPE peripheral processor routine which directs the processing of input/output function requests according to the parameters established by the user in the file environment table.

Code and status field (CS)

A field in the file environment table, the file status table, or in the permanent file definition block. SCOPE establishes the CS field and uses it to communicate information about functions requested by the user program and to return information to the user.

Coded file

A file whose contents are assumed to be alphanumeric characters and special characters such as =, +, and $.

Common file

A file in the system that any job at a control point may attach and use as a part of that job. The file remains in the system until it is released by a control card or file action request, or until the system is deadstarted.

Control points

The concept by which the multiprogramming capability of the 6000 series computers is exploited. Assigning a control point number to a job results in allocation of some of the resources of the system to that job.

Control point area

A $200_8$ word area of central memory resident for each control point. It contains the exchange jump package, flags, pointers and other information pertinent to each job assigned to a corresponding control point number.

Cycle

One of the five files that may be cataloged under one permanent file name.

Data channel

One of the twelve 12-bit bi-directional channels by which information passes between the peripheral processors and peripheral devices.

Dayfile

A chronological file maintained on system mass storage device which forms a permanent accounting and job history file. Entries, called dayfile messages, are generated by operator action or by the system when control cards are processed or other significant action occurs. A portion of the most recent system dayfile is displayed at the console; a copy of the job dayfile is printed with the output for each job.

Deadstart

The process of initializing the system by loading the SCOPE library programs from magnetic tape or mass storage. Deadstart recovery is re-initialization after system failure.

DSD (system display)

The SCOPE system program that provides communication between the operator and the system by accepting control information typed on the console keyboard and by driving the displays to present the operator with information pertinent to all jobs known to the system. DSD is permanently assigned to peripheral processor 9.

Device status table (DST)

A central memory resident table, with entries for each mass storage device controller, that gives the current position of the heads of a device and other information. It is used by the stack processor to determine which request for the device can be most easily satisfied.

Device type code (dt)

An optional parameter on a REQUEST card or in the file environment table which specifies the type of device to be used for that file.

DIS (job display)

A system peripheral processor program similar to system display DSD that provides for communication between a job in central memory and the user at console, and permits the user to control execution of the program by means of the keyboard.

Disposition code (dc)

An optional parameter on a REQUEST card or in the file environment table that indicates how a file is to be processed after the job is terminated or the file is closed.

EDITLIB

A library maintenance program that allows insertion, replacement, and deletion of object language programs in the system library during normal system operation.

End-of-file (EOF)

A short physical record unit or zero-length physical record unit containing a level number of $17_8$ or a card or card image with 6-7-8-9 punches in column 1 that indicates the logical file end.

End-of-information (EOI)

The physical end of data in a file. Some files may have more than one end-of-file indication, but no file may have more than one end-of-information.

End-of-record (EOR)

A short physical record unit or a zero-length physical record unit containing a level number less than $16_8$ or a card or card image containing 7-8-9 punches in column 1.

Equipment number

A number from 0-7 which identifies the setting on a peripheral device controller.

Equipment status table (EST)

A table within central memory resident, with an entry for each hardware device attached to the system, that shows the data channel assignment and whether the equipment is currently in use.

EST ordinal

The number designating the position of an entry within the equipment status table established at each installation.

Field length (FL and FE)

FL is the number of central memory words a job requires, as established by the user in a JOB card or by an RFL control card parameter or by a MEMORY system action request. FE is the number of words in extended core storage that a job requires. Within central memory or extended core storage, the field length added to the reference address defines the upper address limit of a job.

File definition block (FDB)

A table used for communication between a user program and the permanent file manager portion of the SCOPE system. An FDB for each permanent file in the user program must reside in the field length.

File environment table (FET)

A table used for communication between a user program and the operating system when files are processed. An FET created by a compiler or by the user is required within the user field length for each file in the program.

File name table/file status table (FNT/FST)

A table within central memory resident with a three-word entry for each file known to the system. The first word (the file name table entry) consists of the logical file name and control information; the second and third words (the file status table entry) show the current use being made of the file.

Hang

A system stop that may be caused by hardware failure or by an error in a peripheral processor program.

JANUS

The SCOPE peripheral processor routine which controls the processing of up to 4 card readers, 3 card punches, and 12 line printers. It normally functions at control point 1, but may be assigned to another control point by the operator.

Labeled tape

A magnetic tape with header and trailer labels having the format of the 6000 series standard labels or the 3000 series labels.

Level

An indicator specifying relative position in a hierarchy. For priority considerations, level 0 is the lowest priority. For segment loading, level 0 is the initial segment loaded. For overlay loading, both primary and secondary designations are required, with level (0,0) being the main overlay.

Level number

A number from $0\text{-}17_8$ that the user appends to a short physical record unit or places in a zero-length physical record unit to form logical record groups within files. Level number $17_8$ indicates a logical end-of-file. Level number $16_8$ is used by checkpoint/restart and should not otherwise be specified by the user.

Local file

A file attached to a job assigned to a control point. Unless a local file is to be processed further (because it has a user-declared non-zero disposition code, is a system output file, or is declared to be a common file or a permanent file), it vanishes when the job is terminated.

Locked file

A local file with the lock bit set in its file name table entry. The file cannot be processed further by the system until the lock bit is cleared.

Logical file name (lfn)

The 1-7 alphanumeric display coded characters by which the operating system recognizes a file. The name is taken from the file environment table or from a REQUEST card parameter and placed in the file name table when the file is established.

Logical record

A grouping of data consisting of one or more physical record units immediately followed by a short physical record unit or a zero-length physical record unit.

L tape

A tape containing physical records whose size ranges from one central memory word to an upper limit specified by the size of the buffer for that tape.

Monitor (MTR)

The SCOPE routine which coordinates and controls all activities of the system. It occupies peripheral processor 0. It schedules the use of the central processor and the other peripheral processors.

Non-allocatable device

A device such as a magnetic tape or a private disk pack which can be used by only one job at a given time.

Overlay

A block of absolute object code called by a user program and loaded into a specified area of the field length at the time it is needed for execution.

Owncode

An optional parameter of the file environment table which specifies the address of a user-supplied routine to be used when an end-of-information or an error is encountered.

Permanent file

A file on a mass storage device which is protected from unauthorized access and accidental destruction. Unlike a common file, it is not destroyed by normal deadstart.

Permanent file directory (PFD)

A mass storage resident directory of all permanent files with their passwords and other pertinent information. It cannot be accessed by any central processor program.

Permanent file manager (PFM)

Peripheral processor programs corresponding to permanent file functions which implement user requests for permanent file processing.

Physical record unit (PRU)

The smallest amount of information transmitted by a single physical operation of a specified equipment, measured in central memory words. A PRU for mass storage devices is $64_{10}$ words long; that for SCOPE binary magnetic tape is $512_{10}$ words; etc.

Program library

The source language programs which compile or assemble into the SCOPE system library. The program library is maintained by using the UPDATE program.

Recall

The state of a program when it has released control of the central processor until a fixed time has elapsed or until a requested function is complete. Recall is a system action request, as well as an optional parameter of some file action requests.

Record block

A storage area of a fixed size, relative to specific mass storage devices, which is the smallest division of the device that can be assigned to a file.

Record block reservation table (RBR)

A central memory resident table for each allocatable device. It is used during input/output processing to indicate whether a given record block of the device is available for assignment to a file.

Record block table (RBT)

A chain of two-word entries determining the record blocks occupied by each file on an allocatable device. It resides in high central memory when the file is active.

Record block table catalog (RBTC)

A mass storage resident table of entries for each permanent file containing the owner identification, pointers, and record block chain for the file. It cannot be accessed by any central processor program.

Reference address (RA and RE)

RA is the central memory address that is the starting, or zero, address for a program. RA + 1 is used as the communication word between the user program and monitor. RE is the extended core storage starting address used by a program.

S tape (stranger tape)

A magnetic tape (labeled or unlabeled) containing physical records ranging in size from 2 characters to $5120_{10}$ characters. This tape does not contain any level numbers.

SCOPE

The operating system for the CONTROL DATA 6400/6500/6600 computers. The name is derived from Supervisory Control of Program Execution.

SCOPE tape

A tape created under SCOPE 3 with fixed length physical record units: for coded tape, $128_{10}$ central memory words; for binary tape, $512_{10}$ central memory words. A SCOPE tape may be labeled or unlabeled.

Segment

A group of relocatable subprograms which may be loaded or removed from memory as a unit.

Short PRU

A physical record unit containing fewer than normal words, and to which a level number has been appended to indicate the end of a logical record.

Stack processor

A group of peripheral processor routines which processes the requests for all mass storage functions and efficiently sequences execution of these requests.

Standard labeled tape

A tape with labels conforming to the proposed USA Standard for Magnetic Tape Tables and File Structure for Information Interchange. Also called a system labeled tape.

System library

The collection of object language programs residing in central memory or on mass storage which are necessary for running the SCOPE system and its product set.

Transient program

A program which occupies a peripheral processor only for the time it is needed to perform a task assigned by the monitor.

Unlabeled tape

A magnetic tape that does not have a header label. Unlabeled tapes generated by SCOPE contain a trailer label similar to the trailer for a standard labeled tape for the 6000 series computers.

Unsatisfied external

An external reference in a user program for which the system does not have a link to a user program or a system library program after the user program is loaded.

UPDATE

A library maintenance program that allows a program library to be modified or a new program library to be created. It maintains a record of the changes made to individual cards in the library by using correction history bytes.

X tape

An external tape in SCOPE 2 format which: in BCD mode, consists of physical records of $136_{10}$ characters including any blank fill; in binary mode, has a logical record structure with a physical record unit size of $512_{10}$ words.

Zero length PRU

A physical record unit containing only a level number that is used to terminate a logical record; it does not contain any data.

MT xx labl FILE NAME WRITTEN WAS xxx...xx. 4LB

xxx...xx is the name in the tape label for this file.
labl is label identifier.

MT xx labl MULTIFILE NAME SHOULD BE xxx, IS xxx. 4LB

Multiple file name of assigned tape does not agree with request. First xxx is in FET; second is in tape label, labl. Tape may be accepted by typing n.GO. Different tape may be mounted and checked by typing n.RECHECK. Job may be dropped by typing n.DROP.

MT xx labl REEL NUMBER WAS xxxxxx 4LB

xxxxxx is visual reel numbers for tape being read.
labl is label identifier.

MT xx labl REEL NUMBER SHOULD BE xxxx, IS xxxx 4LB

Requested tape reel number does not agree with that specified in tape file label, labl.

MT xx labl REEL NUMBER WRITTEN WAS xxxxxx 4LB

xxxxxx is visual reel number in tape volume header.
labl is label identifier.

MTxx CHxx
EQUIPMENT REJECT 1MT

Hardware rejected equipment connection. Message persists until condition is corrected or operator drops job.

MTxx CHxx
NO RESPONSE 1MT

Equipment cannot be connected because of hardware failure.

MTxx CHxx
NO WRITE ENABLE 1MT

Message persists until operator inserts write ring or drops the job.

# INDEX

# COMMENT SHEET

CONTROL DATA
CORPORATION

TITLE: 6400/6500/6600 SCOPE 3 Reference Manual

PUBLICATION NO. 60189400 REVISION L

Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

CUT ON THIS LINE

FROM NAME: ________________ POSITION: ________________

BUSINESS
ADDRESS: ________________

________________

**NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.**

FOLD ON DOTTED LINES AND STAPLE

APLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

CUT ON THIS LINE

POSTAGE WILL BE PAID BY

**CONTROL DATA CORPORATION**
*Documentation Department*
**215 Moffett Park Drive**
**Sunnyvale, California 94086**

FOLD

FOLD

STAPLE

STAPLE

CONTROL DATA

1/2
3/4
1
1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

6400/6500/6600 SCOPE REFERENCE MANUAL

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55420
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60189400

Litho in U.S.A.