

HEWLETT  PACKARD

# MOVING-HEAD DISC OPERATING SYSTEM

# MOVING-HEAD DISC OPERATING SYSTEM



11000 Wolfe Road  
Cupertino, California 95014

02116-91779

March 1971

First Edition, Aug. 1970  
Second Edition, Mar. 1971

© *Copyright, 1971, by*  
HEWLETT-PACKARD COMPANY  
Cupertino, California  
Printed in the U.S.A.

Second Edition

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system (e.g., in memory, disc or core) or be transmitted by any means, electronic, mechanical, photocopy, recording or otherwise, without prior written permission from the publisher.

Printed in the U.S.A.

# PREFACE

*MOVING-HEAD DISC OPERATING SYSTEM* is the programmer's guide for the Hewlett-Packard Moving-Head Disc Operating System (DOS-M). DOS-M is a batch processing system that executes complete jobs with little or no operator intervention. For a full understanding of DOS-M the reader should be familiar with one of the Hewlett-Packard programming languages, as presented in the *FORTRAN* (02116-9015), *FORTRAN IV* (5951-1321), *ALGOL* (02116-9072) and *ASSEMBLER* (02116-9014) programmer's reference manuals and should refer to the appropriate modules of the *SOFTWARE OPERATING PROCEDURES*.

The Introduction of this manual explains the software and hardware elements of the system. Section I presents the system organization, while Sections II and III cover the complete set of batch and keyboard directives and program calls to the system. All facets of DOS-M programming -- FORTRAN, ALGOL, Assembler, Loader, DEBUG, and Library -- are presented in Section IV. Section V assembles all the necessary information on input/output, including the planning of I/O drivers. Section VI describes use of the Extended File Management Package. The appendices provide tables, summaries, and sample job decks.

# NEW AND CHANGED INFORMATION

This new edition reflects changes to DOS-M necessary to accommodate a new disc driver (HP 2883), the FORTRAN IV Compiler, and the Extended File Management Package. System installation has been removed from this manual and placed in the *SOFTWARE OPERATING PROCEDURES*.

# CONTENTS

iii	PREFACE
xi	INTRODUCTION
1-1	SECTION I
	SYSTEM ORGANIZATION
1-1	DOS-M
1-2	Directives
1-3	EXEC Calls
1-3	Input/Output
1-4	Core Layout
1-4	DISC USAGE
1-4	HP 2870
1-4	HP 2883
1-6	DOS-M Files
1-7	EXTENDED FILE MANAGEMENT PACKAGE
1-7	DOS-M Installation
2-1	SECTION II
	DIRECTIVES
2-4	ABORT
2-5	BATCH
2-6	COMMENT
2-7	DISC-TO-DISC DUMP
2-9	DOWN
2-10	FILE DUMP
2-12	EDIT
2-15	END-OF-FILE
2-16	EJOB
2-17	EQUIPMENT
2-19	SPECIFY SOURCE FILE
2-20	JOB

## CONTENTS

### DIRECTIVES (cont.)

2-21	LIST
2-25	LOGICAL UNIT
2-26	PAUSE
2-27	PROGRAM DUMP
2-30	PROG
2-31	PURGE
2-33	RUN
2-34	SECTOR DUMP
2-36	SYSTEM SEARCH (Optional Directive)
2-38	STORE
2-43	TRACKS
2-45	TYPE
2-46	CHANGE USER DISC
2-48	UP
2-49	DATE (Keyboard Mode Only)
2-50	GO (Keyboard Mode Only)
2-51	INITIALIZE (Keyboard Mode Only)
2-52	OFF (Keyboard Mode Only)

### 3-1 SECTION III EXEC CALLS

3-2	FORMAT OF THE ASSEMBLY LANGUAGE CALLING SEQUENCE
3-2	EXEC CALLS IN ALGOL
3-3	FORMAT OF THE FORTRAN CALLING SEQUENCE
3-4	READ/WRITE
3-7	FILE READ/WRITE
3-9	I/O CONTROL
3-11	I/O STATUS
3-13	WORK AREA LIMITS
3-14	WORK AREA STATUS
3-16	PROGRAM COMPLETION
3-17	PROGRAM SUSPEND
3-19	PROGRAM SEGMENT LOAD

## CONTENTS

### EXEC CALLS (cont.)

- 3-21 SEARCH FILE NAME
- 3-22 TIME REQUEST
- 3-23 CHANGE USER DISC
- 3-27 MAIN PROGRAM LOAD

### 4-1 SECTION IV PROGRAMMING

- 4-1 LOAD-AND-GO FACILITY
- 4-2 DOS-M FORTRAN COMPILERS (Basic FORTRAN and FORTRAN IV)
- 4-2 Compiler Operation
- 4-3 PROG,FTN[4]
- 4-4 Comments on Basic FORTRAN
- 4-10 RTE/DOS ALGOL COMPILER
- 4-10 Compiler Operation
- 4-11 PROG, ALGOL
- 4-13 ALGOL Control Statement
- 4-14 ALGOL Segmentation
- 4-14 ALGOL I/O
- 4-14 ALGOL Error Messages
- 4-15 DOS-M ASSEMBLER
- 4-15 Assembler Operation
- 4-18 DOS-M Assembly Language
- 4-20 Segmented Programs
- 4-25 DOS-M RELOCATING LOADER
- 4-26 Starting the Loader
- 4-30 DEBUG Library Subroutine
- 4-32 Loader Example
- 4-34 Loader Error Messages
- 4-35 THE RELOCATABLE LIBRARIES

### 5-1 SECTION V INPUT/OUTPUT

- 5-1 SOFTWARE I/O STRUCTURE



## CONTENTS

### INPUT/OUTPUT (cont.)

5-1	The Equipment Table
5-4	Logical Unit Numbers
5-5	The Interrupt Table
5-5	Input/Output Drivers
5-6	System I/O
5-6	User Program I/O
5-7	Interrupt Processing
5-7	PLANNING I/O DRIVERS
5-8	Initiation Section
5-11	Completion Section
5-14	LINE PRINTER FORMATTING
5-15	MAGNETIC TAPE USAGE

### 6-1 SECTION VI EXTENDED FILE MANAGEMENT PACKAGE

6-1	ENVIRONMENT
6-1	FUNCTIONS AND STRUCTURE
6-1	DOS-M Files vs. EFMP Files
6-2	EFMP Buffers and Tables
6-3	Logical Read vs. Physical Read
6-3	Logical Write vs. Physical Write
6-4	Update-Writes vs. Append-Writes
6-4	SET UP
6-5	EFMP EXEC CALLS
6-6	DEFINE
6-8	CREATE
6-10	DESTROY
6-11	OPEN
6-13	CLOSE
6-14	READ
6-16	WRITE
6-17	RESET
6-18	STATUS

## CONTENTS

### EXTENDED FILE MANAGEMENT PACKAGE (cont.)

6-19	STATUS FUNCTION NUMBER 1
6-20	STATUS FUNCTION NUMBER 2
6-21	STATUS FUNCTION NUMBER 3
6-22	STATUS FUNCTION NUMBER 4
6-23	STATUS FUNCTION NUMBER 5
6-24	STATUS FUNCTION NUMBER 6
6-25	STATUS FUNCTION NUMBER 7
6-26	REPACK (PURGE)
6-27	COPY
6-29	CHANGE FILE NAME
6-30	POST
6-31	UTIL PROGRAM -- CONVERSATIONAL USE OF EFMP
6-31	Functions
6-32	:PROG,UTIL
6-34	CREATE COMMAND
6-35	DESTROY COMMAND
6-36	OPEN COMMAND
6-37	CLOSE COMMAND
6-38	RESET COMMAND
6-39	STATUS-1 COMMAND
6-40	STATUS-2 COMMAND
6-41	STATUS-3 COMMAND
6-42	STATUS-4 COMMAND
6-43	STATUS-5 COMMAND
6-44	STATUS-6 COMMAND
6-45	STATUS-7 COMMAND
6-46	REPACK COMMAND
6-47	COPY COMMAND
6-48	CHANGE COMMAND
6-49	POST COMMAND
6-50	BRIEF COMMAND
6-51	END COMMAND
6-52	GENERAL ERRORS
6-53	EFMP FILE DISC DIRECTORY

# CONTENTS

## APPENDICES

A	TABLES
B	TYPICAL JOB DECKS
C	RELATION TO OTHER SOFTWARE
D	SUMMARY OF DIRECTIVES
E	SUMMARY OF EXEC CALLS
F	ALGOL EXEC CALLS

## INDEX

## ILLUSTRATIONS

1-6	Figure 1-1. Functional Diagram of DOS-M
4-21	Figure 4-1. Segmented Programs
4-22	Figure 4-2. Main Calling Segment
4-23	Figure 4-3. Segment Calling Segment
4-24	Figure 4-4. Main-to-Segment Jumps
5-10	Figure 5-1. I/O Driver Initiation Section
5-13	Figure 5-2. I/O Driver Completion Section
6-53	Figure 6-1. EFMP File Disc Directory Format
6-54	Figure 6-2. EFMP Disc Pack Layout
A-6	Figure A-1. General Disc Layout
A-6	Figure A-2. System Directory Format
A-7	Figure A-3. User Directory Format
A-7	Figure A-4. Directory Entry Format
A-9	Figure A-5. Disc Layout

# INTRODUCTION

In the Moving-Head Disc Operating System (DOS-M), software modules are stored permanently on the disc for high-speed batch processing, eliminating slow and inefficient paper tape loading. Input can be set up and executed in serial order to automatically edit, translate, load and execute a set of source programs written in HP FORTRAN (an extension of ASA BASIC FORTRAN), HP ALGOL, HP FORTRAN IV, or HP Assembly Language. A variety of files can be stored, edited, listed, dumped and used as input to programs.

## FEATURES OF DOS-M

DOS-M contains the following highlights and features:

- ▮ Keyboard and batch processing modes,
- ▮ Software programming aids: FORTRAN Compiler, Assembler, Relocating Loader, Relocatable Library, Debug Routine, Source File Editor, and ALGOL.
- ▮ Jobs executed in a queue with minimal operator intervention,
- ▮ Symbolic disc files, with relative addressing,
- ▮ Centralized and device-independent I/O processing,
- ▮ Modular structure,
- ▮ Custom configuration to optimize available memory and I/O,
- ▮ Cyclic error checking on disc read & write operations,
- ▮ Exchangeable discs packs, and
- ▮ Optional search of the entire system for file names.

With additional core memory, DOS-M also provides:

- ▮ Extended Files (8K more)

## MINIMUM HARDWARE

The minimum hardware requirements for the Moving Head Disc Operating System are:

1. Computer, 8,192 words of memory, Central Interrupt Processor, DMA, halt on memory parity error.
2. HP 2870 Moving-Head Disc Drive with fixed disc and removable cartridge or HP 2883 Disc File (requires greater than 8,192 words of memory) with one removable pack.
3. System Input Device: Teleprinter (HP 2752).
4. Batch I/O Device: Second Teleprinter (HP 2754).

In place of the HP 2754B teleprinter, the user can select one of the following combinations instead for batch operations:

<u>Batch List Device</u>	<u>Batch Input Device</u>	<u>Batch Punch Device</u>
HP 2752A Teleprinter	Punched Tape Reader	Punch Unit
HP 2752A Teleprinter	Mark Sense Card Reader	Punch Unit
Line Printer	Punched Tape Reader	Punch Unit
Line Printer	Mark Sense Card Reader	Punch Unit

The following hardware options are available:

1. Time Base Generator (provides accounting times).
2. Extended Arithmetic Unit (EAU): provides hardware multiply, divide, etc. for user programs. (DOS-M software contains no EAU instructions.)
3. Additional memory: 12,288, 16,384, or 32,768 words.
4. Additional I/O channels: extenders are available.
5. Memory Protect. (Without memory protect, user programs can destroy DOS-M.)
6. Photoreader.
7. Paper Tape Punch.
8. Line Printer

9. Mark Sense Card Reader.
10. HP 3030 Magnetic Tape Unit (requires 2116 DMA).
11. Additional Disc Drives. (Maximum is four on HP 2870 and two on HP 2883.)
12. Plotter

## DOS-M SOFTWARE MODULES

DOS-M consists of the following programs:

- DOS-M Supervisor and sub-modules
- DOS-M Assembler
- DOS-M FORTRAN Compiler
- DOS-M Relocating Loader
- DOS-M Moving-Head Disc Driver or Pack Disc Driver (DVR31)
- DOS-M Special Teleprinter Driver (DVR 05)
- DOS-M DSGEN (the system generator)

In addition, the following programs can be included when DOS-M is generated:

- RTE/DOS FORTRAN IV Compiler (8K version and 16K version)
- RTE/DOS ALGOL Compiler (16K memory required)
- RTE/DOS Relocatable Library (EAU or Non-EAU)
- RTE/DOS FORTRAN IV Library (extended precision arithmetic)
- DOS I/O Drivers (either core- or disc-resident):
  - Teleprinter (DVR 00)
  - Photoreader (DVR 01)
  - Tape Punch (DVR 02)
  - Line Printer (DVR 12)
  - Mark Sense Card Reader (DVR 15) (uses DMA)
  - 3030 Magnetic Tape (DVR 22) (uses 2116 DMA only)
  - Plotter (DVR 10)
- Extended File Management Package (16K required).

## DOS-M Supervisor

The DOS-M supervisory software consists of a monitor (DISCM) that is partly core-resident and partly (optionally) disc-resident and a disc-resident job processor (JOBPR):

<u>DISCM</u>	<u>JOBPR</u>
Interrupt Processor	Job Processor
Executive Processor	File Manager
I/O Processor	
Executive modules:	
\$EXØ1 through \$EX2Ø	

*NOTE: Exec modules can be made either core- or disc-resident when DOS-M is generated.*

*NOTE: JOBPR is always made disc-resident when DOS-M is generated. DISCM brings it into core when needed.*

# SECTION I

## SYSTEM ORGANIZATION

An operating system is an organized collection of programs which increases the productivity of a computer by providing common functions for all user programs.

An operating system's function is to aid in the preparation, translation, loading, and execution of programs. This is accomplished by an auxiliary, quick access memory, usually a disc storage unit. The various translators, loaders, and other software are stored permanently on the disc for use only when needed. Since the programmer requests a compiler from the disc instead of loading it by hand from paper tape, the overhead time can be significantly reduced.

### DOS-M

The Moving-Head Disc Operating System is composed of user disc files and the DOS-M Supervisor. The Supervisor consists of two parts: a Disc Monitor (DISCM) and a Job Processor (JOBPR). DISCM consists of modules which are either core- or disc-resident and handle I/O transfers, requests from programs, and other supervisory tasks. The disc-resident JOBPR handles operator and programmer directions from the batch or keyboard device.

The Moving-Head Disc Operating System affords speed and convenience. Programs can be input to DOS-M for automatic translation, loading, and execution. For example, simple punched cards carry out load-and-go operations in DOS-M as follows:

- a. DOS-M reads the FORTRAN Compiler into core from the disc.
- b. The Compiler reads the source program from an external device, such as a card reader, and stores the relocatable binary instructions on the disc.
- c. DOS-M reads the Loader into core from the disc.



## SYSTEM ORGANIZATION

- d. The Loader reads the relocatable binary programs from the disc and stores the converted binary instructions on the disc.
- e. DOS-M reads the program in from the disc and runs it.

### Directives

The DOS-M Supervisor operates in response to directives input by the programmer or operator. Directives are strings of up to 72 characters that specify tasks to DOS-M. They are entered in one of the two modes of DOS-M operation: keyboard or batch. In keyboard mode, the directives are entered manually from the teleprinter keyboard. In batch mode, directives can be input as punched cards integrated with the source program into a *job deck* or from paper tape with source in card reader.

A job is a related set of user tasks and data. In keyboard mode, the directives (tasks) are entered separately from the job data. In batch mode, they are included in a job deck that can execute without manual intervention. Jobs may be stacked directly upon one another in a queue.

The DOS-M directives are used for the following functions:

- ⌈ Create, edit, list, dump, and purge user files (relocatable, loader-generated, source and ASCII or binary data).
- ⌈ Turn on systems programs such as FORTRAN, Assembler, etc.
- ⌈ Modify the logical organization of the I/O.
- ⌈ Start and stop a job; type comments; suspend operations.
- ⌈ Translate, load and execute a user program.
- ⌈ Dump core or disc memory.
- ⌈ Resume execution of suspended programs.
- ⌈ Set the date; abort programs; transfer to batch mode (from keyboard mode or batch mode); return to keyboard mode (from batch mode).
- ⌈ Check status of user disc tracks.
- ⌈ Change the subchannel of the user disc.
- ⌈ Search the various disc subchannels for specified file names.
- ⌈ Initialize (label) disc.
- ⌈ Dump a disc to another disc.

DOS-M directives are described in detail in Section II.

## EXEC Calls

After being translated and loaded, an executing user program communicates with DOS-M by means of EXEC calls. An EXEC call is a JSB instruction which transfers control to the DOS-M Supervisor.

The EXEC calls perform the following functions:

- I/O read and write operations.
- User file and work area read and write operations.
- I/O control operations (backspace, EOF, etc.),
- Request I/O status.
- Change the subchannel of the user disc.
- Request limits and status of WORK area (temporary disc storage).
- Program completion.
- Program suspension.
- Loading of program segments or main programs.
- Request the time.

Section III describes EXEC calls in detail.

## Input/Output

All I/O operations and interrupts are channeled through the DISCM section of the DOS-M Supervisor. DISCM is always core-resident and maintains ultimate control of the computer resources. (See "*Software I/O Structure*," Section V.)

I/O programming is device-independent. Programs written in FORTRAN, ALGOL, and Assembly Language specify a logical unit number (with a predefined function, such as data input) in I/O statements instead of a particular device. Logical unit numbers are assigned to appropriate devices by the operator, depending upon what is available. Thus, the programmer need not worry about the type of input or output device performing the actual operation. (See "*Logical Unit Numbers*," Section V.)

# SYSTEM ORGANIZATION

## Core Layout

When DOS-M is active, the core memory is divided into a user program area and a system area (as shown in Figure 1-1). The Disc Monitor program handles all EXEC calls and, if they are legal, transfers them to the proper module for processing. The I/O drivers handle all actual I/O transfers of information. If some I/O drivers are disc-resident, they are read into core by the supervisor when needed. The user program area provides space for execution of user programs. In addition, large DOS-M software modules, such as the FORTRAN Compiler, Assembler, Relocating Loader, and Job Processor, reside on the disc and execute in the user program area.

If the memory protect option is present, a memory protect boundary is set between the executive area and the user program area. This boundary interrupts whenever a user program attempts to execute an I/O instruction (including a HALT) or to modify the executive area. (Instructions can reference the switch register and overflow register.) Programs to be run in the user area must use EXEC calls for input/output, termination, suspension, and other external processes.

## DISC USAGE

### HP 2870

The controller for the moving-head disc supports up to four disc drives (one is required). Each drive contains two discs: a fixed disc and a removable cartridge. Each disc is referenced through a subchannel of the controller. Therefore, the controller has eight subchannels (numbered 0 to 7). The channels are assigned as follows:

Disc Drive Numbers	0	1	2	3
Permanent Subchannels	0	2	4	6
Removable Subchannels	1	3	5	7

### HP 2883

The controller supports one or two drives. Each drive contains a removable pack of disc surfaces and is divided into 4 subchannels. Therefore, this controller also has up to 8 subchannels. The subchannels are assigned as follows:

Disc drive 0 (subchannels 0, 1, 2, 3)

Disc drive 1 (subchannels 4, 5, 6, 7)

## SYSTEM ORGANIZATION

Each subchannel contains 203 tracks. At least three of these tracks must be spares. On the HP 2870, each track contains 24 sectors; on the HP 2883, 115 sectors. (A sector contains 128 16-bit words and is the smallest addressable unit on the disc.) DOS-M normally allows two subchannels to be available to the user: one subchannel contains the system disc and the other contains the user disc (may be the same subchannel as the system disc). The user subchannel can be changed during job or program execution. In addition, an optional system search mode is available to allow searching for user files on any specified subchannels.

The disc storage has four parts:

1. The System Area:  
Executable code created by the system generator and hardware protected; includes DOS-M Supervisor and other system programs.
2. The User Area (optional):  
User file directory and user files (data, object programs, source statements, etc.).
3. The Work Area:  
Temporary storage for the current job.
4. Job Binary Area:  
Temporary storage for relocatable object code generated by the assembler and compilers; this is an area of variable size and starts from the end of the disc.

All four of these areas can reside on the system subchannel. Or the user area can be on a separate subchannel. Only one user area is available to the system at a time. The standard user subchannel is assigned at system generation time; this can be the system disc or another subchannel (removable or permanent disc). The :UD directive and an analogous EXEC call allow the user to temporarily change the user area to another subchannel.

Automatic track switching is provided within each subchannel.

## SYSTEM ORGANIZATION

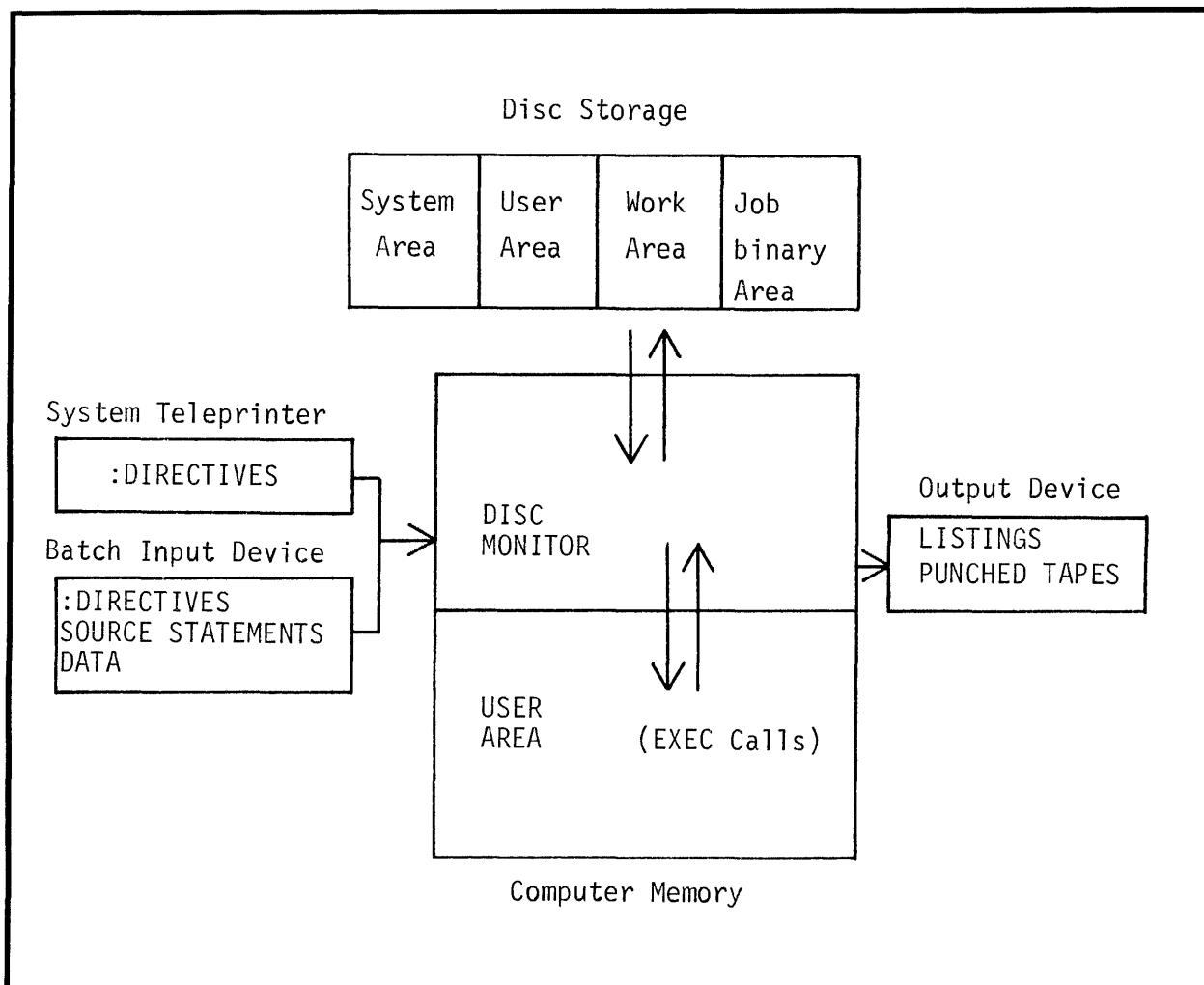


Figure 1-1. Functional Diagram of DOS-M

### DOS-M Files

The disc provides quick access and mass storage for user files consisting of source statements, relocatable and loader-generated object programs, and ASCII or binary data. Each file has a name that is used to reference it.

Programs use the work area of the disc for temporary storage. The system area contains files of systems programs, EXEC modules, a system directory, and library subroutines (see *LIST*, Section II).

## SYSTEM ORGANIZATION

### EXTENDED FILE MANAGEMENT PACKAGE

DOS-M installations with 16K of memory can use the Extended File Management Package (EFMP). This set of optional EXEC modules allows the user program to set aside certain subchannels for a more powerful file structure than that provided by DOS-M. EFMP files allow logical record size (varied under program control), security codes, flexible buffering, sequential reads and writes with a pointer, and detailed status information. In addition, a utility program is available that operates in the user area. It makes these EFMP functions, normally only usable through EXEC calls, usable from the keyboard. For more information on EFMP, see Section VI.

### DOS-M Installation

DOS-M is a series of relocatable binary software modules. Since each module is an independent, general purpose program, the hardware and software configuration of each DOS-M is quite flexible. A separate absolute program, DSGEN, accepts the software modules and generates a configured DOS-M following dialogue-type instructions from the user.

Certain DOS-M modules may be either core- or disc-resident. In a minimum 8K core system, all possible modules are disc-resident; but a 16K memory allows more modules to be core-resident for greater efficiency.

An absolute copy of the configured DOS-M is stored on the disc and is protected from alteration by a hardware override switch. A bootstrap program is used to initiate DOS-M from the disc.

## SECTION II

# DIRECTIVES

Directives are the direct line of communication between the keyboard or batch input device and the Moving-Head Disc Operating System. The operator enters these directives manually through the keyboard or the programmer enters them on punched cards within his job deck. Directives are able to:

- ⌈ Initiate, suspend, terminate, and abort jobs,
- ⌈ Switch between keyboard and batch mode,
- ⌈ Execute, suspend, and resume programs (including compilers, loaders, etc.),
- ⌈ Print the status of the disc tracks and the I/O tables,
- ⌈ Create and purge files of source statements, relocatable and loader-generated binary programs, and ASCII or binary data,
- ⌈ Edit source statement files,
- ⌈ Set up source files for compilers and assemblers,
- ⌈ List and dump files, dump disc and core,
- ⌈ Declare I/O devices up and down,
- ⌈ Set the date and print comments,
- ⌈ Change user disc subchannel,
- ⌈ Dump a copy of a disc onto another subchannel,
- ⌈ Search specified subchannels for file names,
- ⌈ Initialize a disc, and
- ⌈ Turn off an executing program.
- ⌈ Write an end-of-file on magnetic tape.

Directives may enter DOS-M in two modes: keyboard and batch. In either mode, all directives are listed on the teleprinter. Certain directives are legal in one mode only; other directives are operable in both. In keyboard mode, the operator manually inputs the directives through the teleprinter keyboard. In batch mode, the programmer prepares the directives on punched cards or paper tapes and inputs them along with programs, data, etc, in a complete job.

## DIRECTIVES

Directives have the same format, regardless of the mode in which they occur: ":" followed by a directive word (first two characters are significant) and, if necessary, a list of parameters separated by commas (maximum is 15).

For example,

:PROG,FTN,99

When the sequence of parameters is significant, missing parameters must be represented by commas if the following parameters are to be recognized. The first blank character not preceded by a comma is the end of the directive. Comments may appear after this blank; they are ignored by DOS-M. A "rubout" anywhere in a directive deletes the entire directive, while a "control-A" (striking the "A" key and the "control" key simultaneously) deletes the previous character.

DOS-M has two conventions for notifying the operator that directives may be entered. An asterisk (\*) means that DOS-M is waiting for an operator attention directive (see below). An "@" with the bell signals that DOS-M is waiting for further directions. (During some operations, such as editing, there may be perceptible waits while DOS-M processes the directive.)

The operator attains control of DOS-M at any time by striking any system teleprinter key. If the teleprinter is available, DOS-M prints an asterisk (\*) on it; if it is busy, DOS-M prints an asterisk as soon as it is free. At this time, the operator may enter any of the following directives (described in detail in this section):

:ABORT

:DN

:EQ

:LU (reports only)

:TYPE

:UP

:OFF

:PAUSE

*NOTE: Operator attention is disabled during the completion phase of :EDIT and during :PURGE.*



## DIRECTIVES

If the operator types any other directives, DOS-M prints the following message and returns to the executing program.

IGNORED

Some system conditions restrict allowable directives; e.g., after an I/O  
ERR NR EQT# *nn*, the system is waiting for an :UP,*nn*, followed by :GO.  
Under such conditions, otherwise legitimate directives will be IGNORED.

## DIRECTIVES

# ABORT

### Purpose

To terminate the current job before the next JOB or EJOB directive.

### Format

:ABORT

### Comments

ABORT carries out all the operations of an EJOB. All I/O devices are cleared. When it returns to the batch device, DOS-M ignores all directives, except TRACKS, OFF, BATCH, or TYPE, until it finds a new JOB directive. An ABORT may be entered through the keyboard, even if DOS-M is in batch mode.

## BATCH

### Purpose

To switch from keyboard mode to batch mode or to reassign the batch device.

### Format

:*BATCH*,*logical unit*

where *logical unit* is the device to be used as the batch input device.

### Comments

See "TYPE" in this section for the opposite procedure of returning from batch mode to keyboard mode.

## COMMENT

### Purpose

To print a message on the system teleprinter.

### Format

:COMMENT *Character String*

where *Character String* is a message to be printed on the teleprinter.

### Comments

The programmer may use the COMMENT directive with the PAUSE directive to relay instructions to the operator about setting up magnetic tapes, etc. A space (but not a comma) is required between the directive word and the comment string.

### Examples

```
:COMMENT PLACE MAGTAPE LABELED "INPUT" ON THE M.T. UNIT  
:COMMENT PUT "INPUT" PAPERTAPE IN PHOTOREADER
```

# DISC-TO-DISC DUMP

## Purpose

- i. To dump an entire disc onto another subchannel (:DD)
- ii. To dump the system area (including system buffer) onto another subchannel (:DD,X)
- iii. To dump all or specified files of the user area (optionally assigning some new file names) onto another subchannel (:DD,U...)

## Formats

- i. :DD
- ii. :DD,X
- iii. :DD,U[,*file 1*[(*file A*)],*file 2*[(*file B*)],...]

where X specifies the system area,

U specifies the user area,

*file 1, file 2, ...* specify the files to be dumped  
(the entire user area if no files  
are specified),

*file A, file B, ...* specify the optional new names  
for *file 1, file 2, etc.* (renamed  
files can be intermixed with un-  
changed files).

The destination disc must be specified by a :UD directive immediately following the :DD directive. Any other directive will negate the :DD. (For :DD and :DD,X, the directive must be :UD,\*,*n* where *n* is not the system disc.)

## DIRECTIVES

### Comments

When the destination for a :DD,U is a system disc, other than the current system, the user files are dumped in the user area following the system files. This allows the user to dump a system and selected user files to a single disc. (See also :IN.)

The :SS directive does not apply to :DD.

If the files of the source disc cannot completely fit on the destination disc, DOS-M transfers as many whole files as possible and prints

TRAC # TOO BIG

If DOS-M cannot find some of the files specified to be dump, the messages

*file*

UNDEFINED

is printed. This does not effect dumping of the files which are defined.

If a file specified to be dumped has the same name as an existing file on the destination disc, the message

*file*

DUPLICATE FILE-NAME

is printed and the file is not dumped. This does not effect dumping of other files.

## DOWN

### Purpose

To declare an I/O device unavailable for use either before or during the execution of a program.

### Format

:DN,*n*

where *n* is the equipment table entry number for the device to be set down.

### Comments

The system teleprinter and the disc (logical units 1,2, and 3) cannot be set down. :UP resets the down condition.

# FILE DUMP

## Purpose

To dump a user file to a specified peripheral I/O device in a format appropriate to the file content.

## Format

:DUMP,*logical unit*,*file*[,*S1*[,*S2*]]

where *logical unit* is the output device to be used for the dump,  
*file* is the user file to be dumped,  
*S1* and *S2* are the first and last relative sectors to be dumped.

If *S1* and *S2* are not given, the entire file is dumped. If only *S1* is given, then the file, starting with *S1*, is dumped.

## Comments

Files may be dumped on list devices or punch devices. The dump format varies with the type of file and the type of device. See Table 2-1.

Table 2-1  
FILE DUMP Formats

<u>File Type</u>	<u>Punch Device</u>	<u>List Device</u>
ASCII data	64 characters/record	64 characters/record
Binary data	64 words/record	8 octal words/line
Rel. binary programs	Relocatable binary records (loadable)	8 octal words/line
Source statements	1 statement/record	1 statement/line



## DIRECTIVES

Source statements are packed and do not necessarily start on sector boundaries. Thus, if the *S1* and *S2* parameters are used, dumping begins with the start of the first statement beginning in sector *S1*, and ends with the last statement beginning in sector *S2* (this will probably end in the following sector).

Files in the system area cannot be dumped. Errors occur when *S1* > *S2*, or when either *S1* or *S2* is greater than the length of the file.

### Examples

Where *L* is a source file:

```
:DUMP,1,L
A
BB
CCC
DDDD
EEEE
FFFFF
GGGGGGG
@
```

Where *SSERH* is a binary file:

(On the keyboard:)

```
:DU,6,SSERH,1,1
@
```

(On the list device:)

001	000000	062125	072121	114535	010010	010075	010156	010100
	002400	052100	026014	026036	062006	042154	072023	114535
	010025	010076	010077	010006	010153	114535	010033	010076
	010077	010101	010117	102501	002002	026056	062006	072046
	114535	010050	010123	010076	010127	010124	010006	010122
	114535	010056	010076	010077	010126	010153	036006	036006
	036006	036121	026003	114535	010071	010076	010077	010106
	010120	114535	010074	010074	000006	000022	000002	000001
	000000	020116	047524	020106	047525	047104	020120	051117
	043522	040515	020103	047515	050114	042524	042504	000005
	000011	000000	000000	000016	000002	177746	020040	020040
	020040	020040	020040	020040	020040	020040	020040	020040
	020040	020040	020040	020040	020040	020040	020040	020040
	020040	020040	020040	000003	177777	020040	020501	040440
	020040	041102	041040	020040	041503	041440	020040	042104
	042040	020040	042505	042440	020040	043106	043040	020040

## EDIT

### Purpose

To perform listed edit operations on a user source file.

### Format

:EDIT,*file*,*logical unit*[,*new file*]

where *file* is the name of a source file (follows the :SS condition) to be edited according to an edit list (edit operations plus associated source statements) input on the specified *logical unit*. If *new file* appears, the edited source file is stored in a new file (with the name *new file*) on the same subchannel and the old file is not purged. Otherwise, the edited source file is the updated old file. (Follows :SS in searching for duplicate file names.)

Position one of a source statement must not be a slash (/) or a colon (:). The legal edit operations in an edit list are described under Comments.

### Comments

An edit list consists of several edit operations and, optionally, a series of associated source statements (i.e., following REPLACE, INSERT). Edit operations are executed when they are entered. When using the keyboard, the operator must not enter the next operation until the previous one is completed (completion is signaled by "@" output on the keyboard).

All edit operations begin with a slash (/), and only the first character following the slash is required. The rest are ignored up to a comma. If a colon (:) is encountered in column one before the end of the edit list, the job is aborted. In the edit operation formats, the letters *m* and *n* are the

## DIRECTIVES

sequence numbers of the source statements to be edited, starting with one. Letter *m* signifies the starting statement, and *n* is the ending statements of the operation, inclusive. In all cases, *n* must be greater than or equal to *m*; neither can be less than one, nor greater than the last source statement of the file. The *m* must be greater than the *n* of the previous operation.

All edit operations are listed on the system teleprinter as they are executed.

### EDIT OPERATIONS

The following operation causes source statements *m* through *n*, inclusive, to be deleted from the file.

`/DELETE,m[,n]`

If only *m* is specified, only that one statement will be deleted.

By means of an edit operation, the source statements *m* through *n* can be replaced by one or more source statements following `/REPLACE` in the edit list.

`/REPLACE,m[,n]`

Again, if *n* is absent, only *m* is replaced.

The format for the INSERT operation is:

`/INSERT,m`

The source statements which follow `/INSERT` in the edit list are inserted in the file after statement *m*.

In the END operation,

`/END`

the edit directive is terminated and DOS-M returns to its previous mode for further directives.

## DIRECTIVES

### Examples

If a file named SOURC contains:

<i>Statement 1</i>	ASMB,R,B,L
<i>Statement 2</i>	NAM START
<i>Statement 3</i>	A EQU 30
<i>Statement 4</i>	B EQU 20
<i>Statement 5</i>	START NOP
<i>Statement 6</i>	LDA A
<i>Statement 7</i>	END

and the EDIT directive is:

:EDIT,SOURC,5

and the edit list, which follows :EDIT on the batch device, is:

```
/R,3
A EQU 100
B NOP
/D,4
/I,6
STA B
/E
```

then the new file equals:

<i>Statement 1</i>	ASMB,R,B,L
<i>Statement 2</i>	NAM START
<i>Statement 3</i>	A EQU 100
<i>Statement 4</i>	B NOP
<i>Statement 5</i>	START NOP
<i>Statement 6</i>	LDA A
<i>Statement 7</i>	STA B
<i>Statement 8</i>	END

## END-OF-FILE

### Purpose

To write an end-of-file mark a magnetic tape.

### Format

:EF[,*lu*]

where *lu* is the logical unit number of the desired magnetic tape (default is 8).

## EJOB

### Purpose

To terminate the current job normally and return to keyboard mode.

### Format

:EJOB

### Comments

EJOB condenses all user discs by eliminating spaces left by non-permanent programs. (:EJOB follows the :SS condition.) EJOB outputs a message recording the total job and execution time, then returns to keyboard mode. (See STORE directive and *Relocating Loader*, Section IV.) All directives except TRACKS, OFF, or BATCH are ignored until the next JOB directive.

EJOB resets logical units 1 through 9 and resets the :SS condition. EJOB resets the user disc assignment to the standard subchannel unless the standard is not ready or a new cartridge has been inserted (with a different label and without a :UD directive).

When the EJOB directive occurs, a message is printed, similar to that of JOB, giving the total run time of the job and total execution time (if a time-base generator is present). For example,

END JOB START RUN = 0007 MIN. 52.6 SEC. EXEC = 0001 MIN. 21.0 SEC.  
or  
END JOB START (No TBG)

This message is printed on the system teleprinter and on the standard list device.

## EQUIPMENT

Purpose

To list one or all entries in the equipment table.

Format

:EQ[,*n*]

where *n*, if present, indicates the one entry to be listed. If *n* is absent, the entire equipment table is listed.

Comments

Each entry is output in the following format:

EQT *nn* CH *vv* DVR*mm* *d* *r* *Uu* *Ss*

where *nn* is the decimal number of the entry,

*vv* is the octal channel number of the device,

DVR*mm* is the I/O driver number for the device,

*d* specifies DMA if equal to D, no DMA if Ø,

*r* specifies core-resident if equal to R, disc-resident if Ø,

*u* is one decimal digit used for subchannel addressing,

*s* is the availability status of the device:

Ø for not busy, and available,

1 for disabled (down),

2 for busy,

## DIRECTIVES

### Example

```
:EQ
EQT 01 CH 10 DVR31 D R U0 S0
EQT 02 CH 12 DVR22 D 0 U0 S0
EQT 03 CH 14 DVR05 0 R U0 S0
EQT 04 CH 15 DVR01 0 0 U0 S0
EQT 05 CH 16 DVR02 0 0 U0 S0
EQT 06 CH 17 DVR12 0 0 U0 S0
EQT 07 CH 21 DVR15 D 0 U0 S0
@
```



## SPECIFY SOURCE FILE

### Purpose

To specify the user source file to be used as input by the assembler and compilers. (Follows the :SS condition.)

### Format

`:JFILE, file`

where *file* is the name of a TYPE-S file on any active subchannel.

### Comments

If logical unit 2 is specified as the input device when the compiler or assembler is turned on (using :PROG) and a :JFILE has been defined, then the compiler or assembler reads the source statements from the :JFILE.

Only one program can be translated from a file; any statements beyond the end of the source program will be ignored. The JFILE assignment is only changed at the end of the current job or by another JFILE directive.

It is highly recommended that the :JFILE directive immediately precede the corresponding :PROG directive.

# JOB

## Purpose

To initiate a user job and assign it a name for accounting purposes.

## Format

:JOB[,*name*]

where *name* is a string of up to five characters (starting with an alphabetic character) which identifies the job.

## Comments

When DOS-M processes the JOB directive, it prints an accounting message on the system teleprinter and the list device recording the job's *name* (as specified in the JOB directive), the date (as specified in the DATE directive) and the current time (if a time base generator is present):

```
JOB name date TIME = xxxx MIN. xx.x SECS.  
or  
JOB name date (if no time-base generator)
```

For example,

```
:JOB,START  
JOB START MON 6.16.9    TIME = 0013 MIN 41.6 SEC.  
or  
JOB START MON 6.16.9
```

If an EJOB directive has not been encountered, JOB also acts as the EJOB for the previous job. In this case, all actions of the EJOB are carried out, except for returning to keyboard mode from batch mode, before starting the new job.

Only the first two characters of JOB are significant. DOS-M skips everything up to the comma.

# LIST

## Purpose

To list file information recorded in the user or system directories. To list and number the contents of a source file sequentially statement-by-statement.

## Format

(System) :LIST,X,logical unit[,file<sub>1</sub>,...] (Unaffected by :SS)

(User) :LIST,U,logical unit[,file<sub>1</sub>,...]

(Lists the specified directory entries from all the subchannels defined by :SS.)

where X specifies the system area directory, and

U specifies a user area directory,

logical unit specifies the list device, and

file<sub>1</sub>,... names the entries to be listed (if none is specified, the entire directory is listed).

(Source) :LIST,S,logical unit, file[,m[,n]] (follows :SS)

where file names the source file to be listed on the

logical unit specified.

m and n, if present, specify the first and last statements to be listed. If n is absent, then all statements from m on are listed. If neither appear, then the entire field is listed. The restrictions for m and n are the same as those for the EDIT directive.

## Comments

### DIRECTORY LISTING OUTPUT

The first line is a heading, identifying the information that follows:

NAME TYPE SCTRS DISC ORG PROG LIMITS B.P.LIMITS ENTRY LIBR. P-B

SUBCHAN = n (This is printed when :LIST switches to the next subchannel under :SS.)

## DIRECTIVES

The following lines are then printed:

*name type sctrs trk sec lower<sub>p</sub> upper<sub>p</sub> lower<sub>b</sub> upper<sub>b</sub> entry libr p-b*

where *name* identifies the file,

*type* tells what kind of file *name* is,

AD = ASCII data	}	User File Only
BD = binary data		
RB = relocatable binary program		
SS = source statements		
DR = disc resident I/O driver	}	System File Only
LB = library		
SR = system core-resident program		
XS = supervisor module	}	Either File
UM = user main program		
US = user program segment		

*sctrs* is the number of sectors in the file,

*trk* is the track origin of the file,

*sec* is the starting sector of the file within the track specified.

The information below does not appear for types AD, BD, LB, RB and SS.

*lower<sub>p</sub>* is the lower limit (octal) of the program,  
*upper<sub>p</sub>* is the upper limit (octal) of the program,  
*lower<sub>b</sub>* is the lower limit (octal) of the program base page links,  
*upper<sub>b</sub>* is the upper limit (octal) of the program base page links,  
*entry* is the absolute octal address where execution begins,  
*libr* is the beginning absolute octal address of the first library routine included in the program, and  
*p-b* is equal to T if the file is temporary and will be purged by :EJOB unless stored by :ST,P.

If the requested file does not exist, a message appears,

*file* UNDEFINED

## DIRECTIVES

### SOURCE LISTING FORMAT

Each source statement is preceded by a four-digit decimal sequence number.  
If the requested file is not a source file, a three-line message appears,

*file*

ILLEGAL

RE-ENTER STATEMENT ON TTY

The list is terminated by the message

\*\*\*\* LIST END \*\*\*\*

### Examples

(On the keyboard:)

:LI,U,6  
@

(On the list device:)

NAME	TYPE	SCTRS	DISC	ORG	PROG	LIMITS	B.P.	LIMITS	ENTRY	LIBR.	PB
SUBCHAN=4											
EX9	SS	00080	T001	000							
EXM	RB	00063	T004	008							
BBB	SS	00001	T006	023							
SRCH	RB	00003	T007	000							
SSERH	UM	00002	T007	003	10000	10271	00713	00713	10000	10271	T
ASCII	AD	00200	T007	005							
BINRY	BD	00300	T015	013							

NOTE: T on the "PB" column means that the entry is temporary.

# DIRECTIVES

(On the keyboard:)

```
:ST,P      (To make all temporary files permanent.)
@
:LI,U,6
@
```

(On the list device:)

NAME	TYPE	SCTRS	DISC	ORG	PROG	LIMITS	B.P.LIMITS	ENTRY	LIBR.	PB
SUBCHAN=4										
EX9	SS	00080	T001	000						
EXM	RB	00063	T004	008						
BBB	SS	00001	T006	023						
SRCH	RB	00003	T007	000						
SSERH	UM	00002	T007	003	10000	10271	00713	00713	10000	10271
ASCII	AD	00200	T007	005						
BINRY	BD	00300	T015	013						

NOTE: "PB" no longer equals "T."

(On the keyboard:)

```
:LI,S,6,EX19,926,936
```

```
@
```

(On the list device:)

```
0926 ASMB,L,R,X,C,N,B
0927     HED DUMMY $LIBR AND $LIBX FOR RTS SIMULATION ON DOS
0928     NAM DUMRX,6
0929     ENT $LIBR,$LIBX
0930     SPC 2
0931 *     CALLING SEQUENCES: ENTRY          TERMINATION
0932 *
0933 *
0934 *     PRIVILEGED .          JSB $LIBR      JSB $LIBX
0935 *                          NOP            DEF (PROGRAM ENTRY POINT)
0936 *
**** LIST END ****
```

# LOGICAL UNIT

## Purpose

To assign logical unit numbers (4 through 63) for a job or to list the device reference table (logical unit assignments).

## Format

:LU[, $n_1$ [, $n_2$ ]]

where  $n_1$  and  $n_2$ , if both present, assign the device recorded in equipment table entry  $n_2$  to logical unit number  $n_1$  (both are decimal numbers). If only  $n_1$  is present, then the equipment table entry number (see *EQUIPMENT* directive) assigned to logical unit number  $n_1$  is output. If no parameters appear, the entire device reference table is printed.

## Comments

Assignments made by :LU for logical units 4 through 9 are only valid during the current job. Assignments for 10 and above remain after EJOB. At the beginning of each new job, the device reference table for the first nine logical units is reset to the assignments given when the system was configured. This insures a standard I/O organization for all users.

## Example

```
:LU
LU01 EQT03
LU02 EQT01
LU03 EQT01
LU04 EQT05
LU05 EQT04
LU06 EQT06
LU07 EQT07
LU08 EQT02
@
```

## PAUSE

Purpose

To interrupt the current job and return to the keyboard for operator action.

Format

:PAUSE

Comments

PAUSE may be entered through the keyboard even when DOS-M is in batch mode. PAUSE suspends the current job until the operator inputs a GO directive. During this time the operator may mount magnetic tapes or prepare I/O devices. (A series of COMMENT directives or a remark in the PAUSE directive itself can be used to tell the operator what to do during the PAUSE.)

The GO directive returns DOS-M to the job in the previous mode.



## PROGRAM DUMP

### Purpose

To request that a user program be dumped when it completes execution. Two directives are provided: PDUMP for dumping on a normal completion, and ADUMP for dumping when the program aborts.

### Format

:PDUMP[,FWA[,LWA]][,B][,L]

:ADUMP[,FWA[,LWA]][,B][,L]

where *FWA* is the first word address, relative to the program origin,

B means dump the base page linkage area of the program, and,

L means dump the library subroutines used by the program.

*FWA* and *LWA* are octal numbers that specify the limits of the program being dumped.

If *LWA* is missing, the entire program, starting with *FWA*, is dumped.

B alone dumps all the main program, plus base page linkages, but not the library routines.

L alone dumps only the library routines.

If no parameters are given, everything is dumped

## DIRECTIVES

### Comments

The dump directives, PDUMP and ADUMP, must precede the RUN or PROG request in a job. They implicitly refer to the next program to be executed. DOS sets a flag when it encounters either PDUMP or ADUMP, then checks the flag the next time a program is executed. Only one of the requests will be honored, depending upon whether the program runs normally or is aborted. These flags are cleared when a program terminates. Any parameter following L is ignored. If *FWA* is greater than *LWA*, a message is printed.

### LIMIT ERROR

### RE-ENTER STATEMENT ON TTY

The main program and library subroutines are dumped eight octal words per line, along with the octal starting address for that line. For example,

<i>adr</i> <sub>8</sub>	<i>wd-1</i>	<i>wd-2</i>	<i>wd-3</i>	<i>wd-4</i>	<i>wd-5</i>	<i>wd-6</i>	<i>wd-7</i>	<i>wd-8</i>
<i>adr+10</i> <sub>8</sub>	<i>wd-1</i>	<i>wd-2</i>	<i>wd-3</i>	<i>wd-4</i>	<i>wd-5</i>	<i>wd-6</i>	<i>wd-7</i>	<i>wd-8</i>

If present, the base page dump follows the main program and library. Base page linkages exist for page boundary crossings and subroutines. For each line, the starting address appears first, followed by four pairs of octal numbers. The first number of each pair records the content of the base page word (an address elsewhere in core). The second number of each pair records the contents of the address specified by the first item. If the first item is the address of a subroutine, then the second item contains the last address from which the subroutine was called. For example,

	<u><i>pair-1</i></u>		<u><i>pair-2</i></u>		<u><i>pair-3</i></u>		<u><i>pair-4</i></u>	
<i>adr</i>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>
<i>adr+4</i> <sub>8</sub>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>	<i>item-1</i>	<i>item-2</i>

NOTE: :OFF before a program executes clears the dump flags  
:OFF during program execution causes an abort dump.  
:OFF during a dump terminates the dump.

# DIRECTIVES

## Example

```
:ADUMP,Ø15,B      (Set up dump flag)
:RUN,PRG9,6        (Run program)
LU   Ø1214Ø
ABRT Ø1214Ø        (Program aborted)
(Page Eject)
```

(Main program dump)

12ØØØ	16ØØØ1	ØØ2ØØ2	13Ø573	17Ø574	ØØ6ØØ4	16ØØØ1	ØØ2ØØ3	Ø26Ø12
12Ø1Ø	13Ø575	17Ø576	ØØ6ØØ4	16ØØØ1	17Ø577	ØØ6ØØ4	16ØØØ1	17Ø6ØØ

(Page Eject)

(Base page dump)

ØØ57Ø	Ø1Ø137	ØØ2Ø45	Ø1Ø711	ØØ3237	Ø1Ø763	ØØ2Ø45	Ø17Ø14	ØØØ3ØØ
ØØ574	Ø17641	ØØØØØØ	Ø17Ø15	ØØØ4ØØ	Ø17641	ØØØ4Ø6	Ø176Ø1	ØØØØØØ
ØØ6ØØ	Ø1765Ø	ØØØØØØ	Ø17615	ØØØØØØ	Ø17664	ØØØØØØ	Ø17662	ØØØ573
ØØ6Ø4	Ø17637	ØØØ573	Ø17571	1772Ø5	Ø17563	ØØ12Ø4	Ø17714	Ø17715
ØØ61Ø	Ø17562	Ø21121	Ø17534	Ø21122	Ø17536	Ø21122	Ø17633	16Ø656
ØØ614	Ø17544	Ø37626	Ø17546	Ø37626	Ø17673	ØØØØØØ	Ø176Ø5	ØØØØ4Ø

# PROG

## Purpose

To turn on (i.e., load from the disc and begin executing) a program from the system area or programs from the user file which were generated through the DOS-M Relocating Loader. (Follows the :SS condition in searching for the program.)

## Format

:PROG,name[,P<sub>1</sub>,P<sub>2</sub>....P<sub>5</sub>]

where *name* denotes a system program, such as FTN for the DOS-M FORTRAN Compiler, ASMB for the DOS-M Assembler, LOADR for the DOS-M Relocating Loader, or ALGOL for the RTE/DOS ALGOL Compiler. A user program is specified via the file name assigned in the DOS-M Relocating Loader.

P<sub>1</sub> through P<sub>5</sub> are optional parameters which DOS-M transfers to the program named. P<sub>1</sub> through P<sub>5</sub> must be positive integers less than 32767. The program must retrieve the parameters immediately. This procedure is described under :GO.

## Comment

Consult Section IV for the parameters required by FTN, FTN IV, ASMB, ALGOL, and LOADR. Additional programs may be added at system generation time if desired.

*NOTE: User programs can be run using :PROG. This may be useful when the program needs parameters. DOS-M first searches the user files for the program, then the system files.*

## Examples

:PROG,FTN,2,99

:PROG,ASMB,2,6,4

# PURGE

## Purpose

To remove a user file from the user file area.

## Format

:PURGE[,*file*<sub>1</sub>,*file*<sub>2</sub>,...]

where *file*<sub>1</sub>,*file*<sub>2</sub>,... (up to 15 file names or 72 characters per directive) designate files in the user area. These are purged from the user area. If a file cannot be found, a message is printed on the keyboard:

*FILE* UNDEFINED

If no file names are given, all temporary files are purged.

## Comments

Purge follows the :SS condition. After the files are purged from the disc, the remaining user area files are repacked for efficiency. If the end of the user area moves below a track boundary during the purge, the work area becomes a track larger. As each file is purged, DOS-M prints its name on the teleprinter.

*IMPORTANT NOTE: Operator attention is disabled during :PURGE.*

## DIRECTIVES

### Example

```
ORIGINAL CONTENTS OF USER FILE:  F1,F2,F3,F4, FLONG, and F5 (at least)
DIRECTIVE:                        :PURGE,FLONG,F1,F2,D3,D7,F3,F4,F5
OUTPUT:                           FLONG
                                F1
                                F2
                                D3 UNDEFINED
                                D7 UNDEFINED
                                F3
                                F4
                                F5
```

The fastest way to purge all files of a single disc is to use :IN,\*.

# RUN

## Purpose

To run a user program. (Follows the :SS condition.)

## Format

:RUN,*name*[,*time*][,*N*]

where *name* is a user file containing the desired program,

*time* is an integer specifying the maximum number of minutes the program may run (set to five minutes if not specified). DOS-M ignores *time* if a time-base generator is not present.

*N*, if present, tells DOS-M to allow the program to continue running even if it makes EXEC calls with illegal request codes.

## Comments

Programs which have been relocated during the current job but not stored (see STORE directive) permanently in a user file, may be run using this directive. If the program executes longer than the time limit, the current job is aborted and DOS-M scans to the next JOB directive.

If *N* is not present in the RUN directive, the current job will be aborted by any illegal request codes. The *N* option is provided so that programs can be written and tested on DOS-M ultimately to execute with other HP software which does not have the same request codes. (See Appendix C, *RELATION TO OTHER SOFTWARE*.)

## Example

```
:RUN,ROUT,15
```

executes program ROUT up to fifteen minutes not allowing illegal request codes.

## SECTOR DUMP

### Purpose

To dump any specified sector or sectors of the current user disc on the standard list device in either ASCII or octal format.

### Format

:SA,*track*,*sector*[,*number*] (ASCII)

:SO,*track*,*sector*[,*number*] (Octal)

where *track* and *sector* give the starting disc address for the dump, and

*number* gives the number of sectors to be dumped. If *number* is absent, only one sector is dumped. All three parameters are decimal numbers.

### Comments

The ASCII dump format (:SA) is 64 characters per record. The octal dump format (:SO) is eight octal numbers per line. Two ASCII characters equal one computer word (also represented by one octal number). Although :SA dumps 64 characters per record, these do not necessarily appear on one line since the binary numbers are converted to ASCII characters, some of which might be linefeeds or returns.



# DIRECTIVES

## Example

(On the keyboard:)

:S0,0,1  
@

(On the list device:)

001	000000	067767	017570	067744	077743	017613	017613	017613
	017613	064120	007004	077310	064117	044055	160001	044051
	010072	073773	053774	077761	053775	077762	077304	044056
	160001	001727	013733	073305	050060	027460	053763	027445
	067304	044066	037310	027415	027505	044052	160001	023773
	033774	170001	063773	073302	002004	073303	063774	073773
	067304	160001	073766	164000	017570	063305	050060	027440
	006004	160001	033773	170001	006004	063730	170001	006004
	003004	170001	067304	077311	027440	060154	001722	013765
	033774	001727	001723	070154	063761	067302	017606	063762
	067303	017606	002400	067774	017606	063311	067775	017606
	067761	006003	027540	044055	160001	023774	033302	170001
	067762	006003	027546	023775	033303	170001	063776	001200
	067777	006003	002004	064155	070155	054175	070175	006400
	050175	064115	074200	047740	074157	064175	074161	124003
	000000	057766	127570	037766	163766	002021	027571	013764

# SYSTEM SEARCH

(Optional Directive)

## Purpose

To specify a list of disc subchannels to be searched for file names; the :SS condition applies to all EXEC calls and directives that require a file search. (No check is made for existing duplicate file names during searches; the first file found is used.)

## Format

:SS	All active subchannels are searched, starting with the current user subchannel, then continuing from the highest to the lowest number.
:SS, $n_1, n_2, n_3, \dots$	Where $n_1, n_2, \dots$ are subchannel numbers. The current user subchannel is searched first, then the subchannels specified, starting with the lowest number.
:SS,99	Only the current user subchannel is searched. This is the default condition. Every job starts out in this condition.

## Comments

The :SS directive can only be used if it was specifically allowed during system generation. If the operator answers YES to the question

ALLOW :SS?

then :SS directives will be allowed. Otherwise, they are not, and any :SS directive will cause the following message:

BAD CONTROL STATE.

## DIRECTIVES

If a file search results in the file being found, the current user subchannel is changed to the subchannel containing the file. If the file was not found, the current user subchannel is restored to its previous assignment. The LIST, U directive is an exception: this directive does not stop after it finds the file; it continues to look for duplicate entries. When the LIST search is complete, the user subchannel is always restored.

However, if a search is interrupted before completion, the current user disc may be on any subchannel. (This should be checked with a :UD directive.)

More than one :SS can occur during a job. The job starts in :SS,99 condition until a different :SS directive is issued. Each :SS directive remains in effect until another is issued. :SS directives do not apply to file searches initiated by the Relocating Loader or to disc dumps initiated by the :DD directive.

Whenever the user subchannel assignment is changed (except by a running program), the system prints a message:

SUBCHAN = *n*

# STORE

## Purpose

To create a user file on the disc and assign it a name. The STORE directive can create relocatable object program files (type-R), loader-generated object program files (type-P), source statement files (type-S), ASCII data files (type-A), and binary data files (type-B). (Follows :SS in checking for duplicate file names.)

## Format

The format varies according to what type file is being created. See Comments below for details:

TYPE-R	:STORE,R, <i>file</i> [, <i>logical unit</i> ]
TYPE-P	:STORE,P[, <i>name</i> <sub>1</sub> , <i>name</i> <sub>2</sub> ,...]
TYPE-S	:STORE,S, <i>file</i> , <i>logical unit</i>
TYPE-A	:STORE,A, <i>file</i> , <i>sectors</i>
TYPE-B	:STORE,B, <i>file</i> , <i>sectors</i>

*NOTE: The "Control @" should not be used in file names.*

## Comments

### TYPE - R FILES

The directive format is:

:STORE,R,*file*[,*logical unit*]

where *file* is a name consisting of five characters or less.

## DIRECTIVES

A user file is created under this name, and relocatable binary programs are read into it from the logical unit specified or from the *job binary area* of the work tracks if none is specified. The *job binary area* remains as it was before the STORE directive. (See Section IV, *DOS-M FORTRAN* and *DOS-M ASSEMBLY LANGUAGE*.)

If DOS-M comes to an end-of-tape, it asks:

DONE?

If there are more tapes, the operator places the next tape in the reader and replies NO; otherwise, he answers YES.

The user should not assign any file names that will be used as program names as this will make loading impossible. The file may be input to the DOS-M Relocating Loader for relocation into an executable program. (See Section IV, *DOS-M RELOCATING LOADER*.)

### Examples

:STORE,R,RINE

(Stores all of the relocatable programs from the job binary area into the file RINE created for that purpose.)

:STORE,R,JUGG,5

(Stores relocatable programs from logical unit 5, the standard input device, into the file JUGG.)

## DIRECTIVES

### TYPE - P FILES

The directive format is:

```
:STORE,P[,name1,name2,....]
```

where *name<sub>1</sub>*, *name<sub>2</sub>* ... are programs that the DOS-M Relocating Loader had relocated into executable format during the current job. Up to 14 programs per directive are allowed. If none are specified, all programs loaded during the current job are stored. DOS-M finds these temporary programs in the user file and converts them to permanent user files; the program name automatically becomes the file name.

Programs loaded during the current job but not stored as files (as shown above) may be executed normally (RUN or PROG directive) and appear in the user directory (LIST directive). At the end of a job, however, they are purged from the directory unless they have been converted to user files by a STORE,P directive.

### Examples

```
:STORE,P
```

(Changes all programs loaded during the current job using the Relocating Loader into permanent user files.)

```
:STORE,P,ARITH,MATH,TRIG,ALGEB
```

(Searches for the programs listed and makes them permanent user files.)

## DIRECTIVES

### TYPE - S FILES

The directive format is:

`:STORE,S,file,logical unit`

where *file* is the name of the user file to be filled with source statements from the *logical unit* specified. *File* must not duplicate a name already present in the user or system files. The source statement input must be terminated by a double colon (::). If the :: is omitted, DOS-M stores the succeeding data on the disc as if it were source statements.

If DOS comes to an end-of-tape before finding the ::, it asks

DONE?

If there are more tapes, the operator replies NO: otherwise, he answers YES.

When DOS-M completes the STORE, it prints

`nnnn LINES`

where *nnnn* is the number of statements stored.

### Example

`:STORE,S,SOURC,5`

(Reads source statements from the standard input device and stores them in a new file SOURC.)

## DIRECTIVES

### TYPE - A and TYPE - B FILES

The directive format is:

`:STORE,type,file,sectors`

where *type* is either A (for ASCII character data) or B (for binary data), and *file* is the name assigned to a file containing the number of *sectors* requested. These requests are made prior to executing a program to reserve a file area; no data is involved. The program may store and retrieve data from the file through a call to EXEC.

It is the programmer's responsibility to store the right kind of data in the file. The EXEC call must specify the file name and the relative sector within the file. DOS-M checks that the file name exists and contains the sector specified.

#### Example

`:STORE,A,ASCII,20`

(Creates a file name ASCII,20 sectors in length. A sector equals 128 words.)



## DIRECTIVES

# TRACKS

### Purpose

To print the next available track on the current user disc.

### Format

:TRACKS

### Comments

The number of the first track beyond the end of the current user area, followed by the number of faulty tracks that have been replaced by spares.

Tracks are replaced by spares when parity errors occur on read or write.

### Examples

The following is an example in which no faulty tracks are reported.

(INPUT) :TRACKS

(OUTPUT) NEXT AVAIL TRACK = 0010

@

(End of directive processing)

## DIRECTIVES

In this example, the system reports that 2 tracks have been replaced by spares.

```
(INPUT)      :TRACKS
(OUTPUT)     NEXT AVAIL TRACK = 0012
              BAD = 2
              @                               (End of directive processing)
```

In this example, the system reports that there are no more work tracks available.

```
(INPUT)      :TRACKS
(OUTPUT)     NEXT AVAIL TRACK = NONE
              @                               (End of directive processing)
```

## TYPE

### Purpose

To return from batch mode to keyboard mode.

### Format

:TYPE

### Comments

Control is returned to the teleprinter keyboard. TYPE may be entered through the batch device or keyboard device; but when it is entered from the keyboard, DOS-M waits until the current executing program is completed or is aborted before returning to keyboard mode. If TYPE is entered while already in keyboard mode, the directive is ignored.

# CHANGE USER DISC

## Purpose

To change the subchannel assignment for the user disc.

## Format

:UD[,*label*][,*n*]

where *label* is a six-character disc label (\* for an unlabeled disc).

*n* is the subchannel.

## Comments

Discs are labeled by the :IN directive.

Each form of the :UD directive has a different purpose:

### Example

### Action

:UD  
(without label or  
subchannel)

Interrogates the current user disc subchannel  
and prints its label on the system teleprinter:

SUBCHAN = *n*

LBL = *label* (or UNLBL)

:UD,,*n*  
(no label)

If *n* is labeled, DOS-M prints:

LBL = *label* (or UNLBL)

No assignment is made.

:UD,*label*,*n*

If *n* is labeled with the specified *label*,  
DOS-M assigns *n* as the user disc.

If *n* is unlabeled or has a different *label*,  
DOS-M prints:

LBL = *label* (or UNLBL)

Operator can then reissue :UD,*label*,*n* with  
the correct label.

## DIRECTIVES

<u>Example</u>	<u>Action</u>
<code>:UD,label</code> (no subchannel)	DOS-M searches for the <i>label</i> , starting with the highest number subchannel (determined at system generation). If <i>label</i> is found, DOS-M makes it the user disc and prints: <code>SUBCHAN = n</code> If <i>label</i> is not found, DOS-M prints: <code>DISC NOT ON SYS</code>
<code>:UD,*,n</code>	If <i>n</i> is unlabeled, DOS-M assigns <i>n</i> as the user disc.  If <i>n</i> is labeled, DOS-M makes no assignment and prints: <code>LBL = label</code>
<code>:UD,*</code>	Assigns the highest number unlabeled disc as the user disc and prints: <code>SUBCHAN = n</code>  If there are no unlabeled discs, DOS-M prints: <code>DISC NOT ON SYS</code>

If the `:UD` directive specifies a subchannel with an incorrect system proprietary code (see Appendix A), DOS-M still makes the assignment, and prints:

`TSB DISC or ??? DISC`

If the `:UD` directive specifies a subchannel whose system generation code (see Section VI) does not match that of the current system disc, DOS-M still makes the assignment but prints:

`DISC GEN CODE nnnn NOT SYS GEN CODE mmmm ERR POSS`

The changes made by `:UD` are only temporary; the user disc is reset at the end of each job.

*NOTE: Before executing a :DD or :DD,X to a TSB or ??? DISC, the disc should be initialized with :IN; otherwise, bad tracks may be reported erroneously.*

## UP

Purpose

To declare an I/O device ready for use.

Format

:UP,*n*

where *n* is the equipment table entry number corresponding to the device.

Comments

The :UP directive (followed by a :GO) is usually used in response to the following messages from DOS-M:

I/O ERR ET EQT #*n*

I/O ERR NR EQT #*n*

I/O ERR PE EQT #*n*

where ET indicates end of tape,

NR indicates device not ready,

PE indicates parity error, and

*n* is the equipment entry number.

If you enter the incorrect *n*, DOS-M replies by printing out all the down devices.

*NOTE: The directives in the rest of this section pertain to operation in the keyboard mode only.*

## DATE

### Purpose

To set the date and time for accounting purposes whenever DOS-M is started up.

### Format

:DATE,*day*[,*hour*,*min*]

where *day* is any string of ten or less characters (commas not permitted) chosen by the operator (such as 7/10/69, 10.JULY.69, etc.);

*hour* and *min* are the current time in hours and minutes on a 24-hour clock. If not given or time-base generator is not present, they are set to zero.

### Comments

The DATE directive is legal only following a start-up procedure. The directive is not accepted any other time.

### Examples

:DATE,7/10/69,12,23

:DATE,WEDNESDAY,7,45

:DATE,10JULY1969

## GO

### Purpose

To resume a program that has been suspended, and optionally, to transfer up to five parameters to that program.

### Format

:GO[, $P_1$ , $P_2$ ,... $P_5$ ]

where  $P_1$  through  $P_5$  are optional parameters and must be decimal values between 0 and 32767.

### Comments

When a program suspends itself (see Section III, *PROGRAM SUSPEND EXEC CALL*), it is restarted by a GO directive. Upon return to a suspended program, the initial address of the five parameters is located in the B-register. A FORTRAN program calls the library subroutine RMPAR to transfer the parameters to a specified 5-word array. The first statement after the suspend call, in a FORTRAN program, must be the call to RMPAR. For example,

```
DIMENSION I(5)  
CALL RMPAR (I)
```

An assembly language program should use the B-register upon return from the suspend to obtain and save the parameters prior to making any EXEC request or I/O request.



## INITIALIZE

### Purpose

To label or unlabel the current user disc.

### Format

:IN,*label*

where *label* is a six-character name to be written on the disc  
or "\*" which means unlabel the disc. (The *label*  
should not contain a "Control @.")

### Comments

If the user disc is already labeled, DOS-M prints:

DOS or TSB or ??? LABEL *nnnnnn* (*nnnnnn* is existing label)  
OK TO PURGE?

The operator must respond with

YES

to actually execute the directive, or

NO

to leave the disc unchanged.

If the label equals "\*", the user files also are purged.

If the current user disc is labeled SYSTEM and is not hardware protected  
(which means it was created by a :DD,X), the system area is destroyed and  
any files are moved down to low disc.

*NOTE: If the disc labeled SYSTEM is hardware pro-  
tected, the computer performs a HALT 31 and  
the new label is not assigned.*

## DIRECTIVES

Labeling a disc eliminates any old label on the disc but does not eliminate the directory or files on the disc.

Unlabeling a disc also purges the directory.

:IN always changes the system generation code and system proprietary code to that of the current system. :IN can prepare discs for use by DOS-M that were formatted by a diagnostic or other software.

DIRECTIVES  
(KEYBOARD MODE ONLY)

## OFF

### Purpose

To abort the currently executing user program of system operation without terminating the job.

### Format

:OFF

### Comments

:OFF returns the system to keyboard mode.

OFF can be used to terminate undesired lists, edits, disc-to-disc dumps, program loops, loader operations, assemblies, and compilations.

:OFF cancels any pending :DD, :AD, or :PD directives, unless a program is running, in which case, a pending :ADUMP is executed.

## SECTION III

# EXEC CALLS

Using EXEC calls, which are the line of communication between an executing program and DOS-M, a program is able to:

- ⌈ Perform input and output operations,
- ⌈ Request status of I/O devices
- ⌈ Determine availability of work area tracks,
- ⌈ Terminate or suspend itself,
- ⌈ Load its segments,
- ⌈ Search for file names,
- ⌈ Obtain the time of day, or
- ⌈ Change the user disc subchannel.

An EXEC call is a block of words, consisting of an executable instruction and a list of parameters defining the request. The execution of the instruction transfers control to DOS-M. DOS-M then determines the type of request (from the parameter list) and, if it is legally specified, initiates processing of the request. The executable instruction is a jump subroutine (JSB) to EXEC.

In FORTRAN, EXEC calls are coded as CALL statements. In ALGOL, procedure calls are used. In Assembly Language, EXEC calls are coded as a JSB, followed by a series of parameter definitions. For any particular call, the object code generated for the FORTRAN CALL Statement and the ALGOL procedure call is equivalent to the corresponding Assembly Language object code.

This section describes the basic formats of FORTRAN, ALGOL and Assembly Language EXEC calls, then each EXEC call is presented in detail.

## EXEC CALLS

### FORMAT OF THE ASSEMBLY LANGUAGE CALLING SEQUENCE

The following is a general model of an EXEC call in Assembly Language:

EXT EXEC	(Used to link program to DOS-M)
⋮	
JSB EXEC	(Transfer control to DOS-M)
DEF *+n+1	(Defines point of return from DOS-M, n is number of parameters; may not be an indirect address; must be the location immediately following the last parameter address)
DEF P <sub>1</sub>	} (Define addresses of parameters which may occur anywhere in program; may be multi-level indirect)
⋮	
DEF P <sub>n</sub>	
return point	(Continue execution of program)
⋮	
⋮	
⋮	
P <sub>1</sub> ---	} (Actual parameter values)
⋮	
⋮	
P <sub>n</sub> ---	

### EXEC CALLS IN ALGOL

In ALGOL, certain conventions must be followed in making EXEC calls. First, since EXEC is external to the program it must be declared a CODE procedure. Second, parameters that are going to be changed must be declared "name" and those that are not to change must be VALUE parameters. Third, when arrays are passed as parameters, the first element of the array (not the name) must be passed as an INTEGER type "name". Fourth, since ALGOL requires that the format of each procedure call be defined, a program must declare a dummy external procedure for each type of EXEC call it makes. (These dummy procedures must be compiled as separate procedures to provide proper Linkage in the Loader.)

## EXEC CALLS

### ALGOL Example

See Appendix F for an example of an ALGOL program making an EXEC call through an external CODE procedure.

### FORMAT OF THE FORTRAN CALLING SEQUENCE

In FORTRAN, the EXEC call consists of a CALL Statement and a series of assignment statements defining the variable parameters of the call:

CALL EXEC ( $P_1$ ,  $P_2$  . . . . ,  $P_n$ )

where  $P_1$  through  $P_n$  are either values or variables defined

elsewhere in the program. Variables must begin with  
a letter I through N, since they are integer variables.

#### Example

CALL EXEC (7)	}	Equivalent calling sequence
or		
IRCDE = 7		
CALL EXEC (IRCDE)		

Some EXEC call functions are handled automatically by the FORTRAN compiler or special subroutines. (Refer to "FORTRAN," Section IV, DOS-M PROGRAMMING, and the specific EXEC calls in this section.)

# READ/WRITE

## Purpose

To transfer information to or from an external I/O device or the work area of the disc. (DOS-M handles track switching automatically.)

## Assembly Language

EXT	EXEC		
:			
JSB	EXEC		(Transfer control to DOS-M)
DEF	*+5 (or 7)		(Point of return from DOS-M; 7 is for disc request)
DEF	RCODE		(Request code)
DEF	CONWD		(Control information)
DEF	BUFFR		(Buffer location)
DEF	BUFFL		(Buffer length)
DEF	DTRAK		(Track number-disc transfer only)
DEF	DSECT		(Sector number-disc transfer only)
(return point)			(Continue execution)
:			
RCODE	DEC	1 (or 2)	(1=READ, 2=WRITE)
CONWD	OCT	<i>conwd</i>	( <i>conwd</i> is described in Comments)
BUFFR	BSS	<i>n</i>	(Buffer of <i>n</i> words)
BUFFL	DEC	<i>n</i> (or -2 <i>n</i> )	(Same <i>n</i> ; words (+) or characters (-))
DTRAK	DEC	<i>f</i>	(Work area track number, decimal)
DSEC	DEC	<i>g</i>	(Work area sector number, decimal)

## EXEC CALLS

### FORTRAN

I/O transfers to regular devices are programmed by standard FORTRAN READ and WRITE Statements. I/O on the work area of the disc is done with a subroutine BINRY, described in the Comments, or the FORTRAN equivalent of the EXEC call:

CALL EXEC (ICODE, ICON, IBUF, IBUFL, ITRAK, ISECT)

### Comments

READ/WRITE EXEC calls carry out I/O transfers including those on the work area of the disc. (See FILE READ/WRITE EXEC CALL.)

### CONWD

The conwd, required in the calling sequence, contains the following fields:

	Ø	Ø	W					K	V	M	LOGICAL UNIT #					
BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø

### Field

### Function

- W      If 1, tells DOS-M to return to the calling program after starting the I/O transfer. If W = Ø, DOS-M waits until the transfer is complete before returning.
- K      Used with keyboard input, specifies printing the input as received if K = 1. If K = Ø, "no printing" is specified.
- V      Used when reading variable length records from punched tape devices in binary format (M = 1, below). If V = Ø, the record length is determined by buffer length. If V = 1, the record length is determined by the word count in the first non-zero character which is read in.
- M      Determines the mode of data transfer. If M = Ø, transfer is in ASCII character format, and if M = 1, binary format. (Disc is always binary.)



## EXEC CALLS

### BINRY

User FORTRAN programs call the FORTRAN disc read/write library routine, BINRY, to accomplish I/O in the work area. The user must specify: an array to be used as a buffer, the length of the buffer in words (equal to the number of elements in an integer array, double that for a real array), the disc logical unit, track number, sector number, and offset in words within the sector. (If the offset equals  $\emptyset$ , the transfer begins on the sector boundary. If the offset equals N, then transfer skips N words of the sector before starting). BINRY has two entry points, BREAD and BWRIT, for read and write operations respectively. An example below gives the calling procedure.

```
DIMENSION  IBUF(1 $\emptyset$ ), BUF(2 $\emptyset$ )
LUN = 2
ITRK = 12
ISECT = 63
IOFF =  $\emptyset$ 
CALL BREAD (BUF, 4 $\emptyset$ , LUN, ITRK, ISECT, IOFF),
or
CALL BWRIT (IBUF, 1 $\emptyset$ , LUN, ITRK, ISECT, IOFF)
```

### Waiting and No Waiting

If the program requests the *no waiting* option in the *conwd*, it can check for the end of the I/O operation with the I/O STATUS EXEC call. In the Assembly Language calling sequence, the buffer length can be given in words (+) or characters (-). When the transfer is complete, the amount actually transferred can be learned by the same status call. A positive number of words or characters, depending upon which were originally requested, is returned. If the WAIT option is used, DOS-M returns the number of transmitted words or characters to the B register.

*NOTE: When doing "no waiting" I/O and attempting to load program segments: 1. Under :RUN DOS-M waits for all I/O to complete before loading the segment; and 2. under :PROG, DOS-M does not wait.*

## FILE READ/WRITE

Purpose

To transfer information to or from a file on the user disc; the file must be referenced by name. (The :SS condition is followed.)

Assembly Language

```

EXT EXEC
:
JSB EXEC                (Transfer control to DOS-M)
DEF *+7                 (Point of return from DOS-M)
DEF RCODE               (Request code)
DEF CONWD               (Control information)
DEF BUFFR               (Buffer location)
DEF BUFFL               (Buffer length)
DEF FNAME               (File name)
DEF RSECT               (Relative sector within file)
return point           (Continue execution)
:
RCODE DEC 14 or 15      (14 = READ, 15 = WRITE)
CONWD OCT conwd        (See Comments, READ/WRITE EXEC CALL.)
BUFFR BSS n            (Buffer of n words)
BUFFL DEC n or -2n    (Same n; words (+) or characters (-))
FNAME ASC 3,xxxxx      (User file name = xxxxx)
RSECT DEC m           (Relative sector number )

```

## EXEC CALLS

### FORTRAN

```
DIMENSION IFILE (3)
IFILE(1) = xxxxxB           (First two characters of file name)
IFILE(2) = xxxxxB           (Second two characters)
IFILE(3) = xxxxxB           (Last character and blank)
IRCD = 14 (or 15)           (Request code)
ICNWD = xxxxxB              (conwd)
DIMENSION IBUF(10)
CALL EXEC (IRCDE, ICNWD, IBUF, 10, IFILE, 0)
```

### Comments

See the Comments under READ/WRITE EXEC CALL for a description of the *conwd* fields needed in the above calling sequences.

To read or write on the first sector of a file,  $m=0$ ; for the last sector,  $m$ =number of sectors in the file -1. To determine the size of a file, use the SEARCH FILE NAMES EXEC call.

Any type of file may be read, but only ASCII or binary data files may be written.

If the DOS-M installation is likely to have more than one user disc, the program should use the CHANGE USER DISC EXEC call without a subchannel specified to check whether the correct user disc is currently assigned. Alternatively, the user can use an :SS directive to set up a system search condition for referencing files on many subchannels.

# I/O CONTROL

## Purpose

To carry out various I/O control operations, such as backspace, write end-of-file, rewind, etc.

## Assembly Language

```

      EXT EXEC
      :
      JSB EXEC      (Transfer control to DOS-M)
      DEF *+4(or 3) (Point of return from DOS-M)
      DEF RCODE      (Request code)
      DEF CONWD      (Control information)
      DEF PARAM      (Optional parameter)
      return point   (Continue execution)
      :
      RCODE DEC 3      (Request code = 3)
      CONWD OCT conwd (See Comments)
      PARAM DEC n     (Required for some control functions;
                        see Comments)

```

## FORTRAN

Use the FORTRAN auxiliary I/O statements or an EXEC calling sequence.

```

      IRCDE = 3      (Request code)
      ICNWD = conwd (See Comments)
      IPRAM = x     (Optional; see Comments)
      CALL EXEC (IRCDE, ICNWD, IPRAM)
      CALL EXEC (IRCDE, ICNWD)

```

Comments

## CONWD

The control word value (*conwd*) has two fields:

	$\emptyset$ $\emptyset$		W	FUNCTION CODE (see below)							LOGICAL UNIT NUMBER					
BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	$\emptyset$

If  $W = 1$ , DOS-M returns to the calling program after starting the control request.

If  $W = \emptyset$ , DOS-M waits until the control request is complete before returning.

<u>Function Code (Octal)</u>	<u>Action</u>
$\emptyset\emptyset\emptyset$	Unused
$\emptyset\emptyset 1$	Write end-of-file (magnetic tape)
$\emptyset\emptyset 2$	Backspace one record (magnetic tape)
$\emptyset\emptyset 3$	Forward space one record (magnetic tape)
$\emptyset\emptyset 4$	Rewind (magnetic tape)
$\emptyset\emptyset 5$	Rewind standby (magnetic tape)
$\emptyset\emptyset 6$	Dynamic status (magnetic tape)
$\emptyset\emptyset 7$	Set end-of-paper tape
$\emptyset 1\emptyset$	Generate paper tape leader
$\emptyset 11$	List output line spacing (PARAM or IPRMA required)
$\emptyset 12$	
$\vdots$	
177	Unused

Function code  $11_8$ , list output line spacing, requires the optional parameter mentioned in the calling sequences. PARAM (or IPRAM) designates the number of lines to be spaced on the specified logical unit. A negative parameter specifies a page eject on a line printer or number of lines to be spaced on the teleprinter. For details of line printer formatting, consult Appendix E.

# I/O STATUS

## Purpose

To request the status of a particular I/O device, and the amount transmitted in the last operation.

## Assembly Language

```

EXT EXEC
:
JSB EXEC          (Transfer control to DOS-M)
DEF *+5           (Point of return from DOS-M)
DEF RCODE         (Request code)
DEF CONWD         (Logical unit)
DEF STATS         (Status returned)
DEF TLOG          (Transmission log returned)
return point      (Continue execution)
:
RCODE DEC 13      (Request code - 13)
CONWD DEC n      (Logical unit number)
STATS NOP         (Status returned here)
TLOG NOP          (Transmission log returned here)

```

## FORTRAN

```

IRCDE = 13        (Request code)
ICNWD = n         (n is decimal logical unit)
CALL EXEC (IRCDE, ICNWD, ISTAT, ITLOG)

```

## EXEC CALLS

### Comments

The status returned in the A-register and in STATS is the hardware status of the device specified by the logical unit. The transmission log in the B-register and in TLOG contains the amount of information which was transferred (a positive number of words or characters depending on which was requested by the call initiating the transfer).

## WORK AREA LIMITS

### Purpose

To ascertain the first and last tracks of the work area on the system disc and the number of sectors per track.

### Assembly Language

```

EXT EXEC
:
JSB EXEC          (Transfer control to DOS-M)
DEF *+5           (Point of return from DOS-M)
DEF RCODE         (Request code)
DEF FTRAK         (First track)
DEF LTRAK         (Last track)
DEF SIZE          (Number of sectors/track)
return point      (Continue execution)
:
RCODE DEC 17      (Request code = 17
FTRAK NOP         (Returns first work track number here)
LTRAK NOP         (Returns last work track number here)
SIZE  NOP         (Returns number of sectors per track here)

```

### FORTRAN

```

IRCDE = 17        (Request code)
CALL EXEC (IRCDE, IFTRK, ILTRK, ISIZE)

```

### Comments

This call returns the limits of the work area, that area of the system disc which programs use for temporary storage with the READ/WRITE EXEC call.



## WORK AREA STATUS

### Purpose

To ascertain whether a specified number of consecutive operable tracks exist in the work area of the system disc.

### Assembly Language

```

EXT EXEC
:
JSB EXEC      (Transfer control to DOS-M)
DEF *+5      (Point of return from DOS-M)
DEF RCODE     (Request code)
DEF NTRAK     (Number of tracks desired)
DEF RTACK     (Starting track desired)
DEF STRAK     (Actual starting track)
return point  (Continue execution)
:
RCODE DEC 16   (Request code = 16)
NTRAK DEC n   (Consecutive tracks desired)
TRACK NOP     (Desired track; from LIMITS call)
STRAK NOP     (Actual starting track available,
              Ø if n tracks not available)

```

### FORTTRAN

```

IRCDE = 16      (Request code)
ICNWD = n      (Consecutive tracks desired)
ITRAK = m      (Desired starting track)
CALL EXEC (IRCDE, ICNWD, ITRAK, ISTRK)

```

## EXEC CALLS

### Comments

This call is used with the WORK AREA LIMITS EXEC call to establish the nature of the work area. The READ/WRITE EXEC call then transmits information to and from this area, using the track numbers determined by this call. DOS-M handles track switching automatically.

If a read or write is issued to a disc address that does not lie in the WORK AREA, the message

IT nnnnn

is printed and the program is terminated.

DOS-M checks whether there are  $n$  consecutive tracks starting at the track specified. Upon location of tracks, DOS-M returns the starting track number to the program. If DOS-M does not locate  $n$  consecutive tracks, it returns  $\emptyset$  in TRAK or ITRAK.

## PROGRAM COMPLETION

### Purpose

To notify DOS-M that the calling program is finished and wishes to terminate.

### Assembly Language

```
EXT EXEC
:
JSB EXEC      (Transfer control to DOS-M)
DEF *+2      (Return point from DOS-M)
DEF RCODE     (Request code)
return point
:
RCODE DEC 6   (Request code = 6)
```

### FORTRAN

The FORTRAN and ALGOL compilers generate a PROGRAM COMPLETION EXEC CALL automatically when they compile an END\$ or STOP statement.

## PROGRAM SUSPEND

### Purpose

To suspend the calling program from execution until restarted by the GO directive.

### Assembly Language

```
EXT EXEC
:
JSB EXEC      (Transfer control to DOS-M)
DEF *+2       (Point of return from DOS-M)
DEF RCODE     (Request code)
return point  (Continue execution)
:
RCODE DEC 7    (Request Code = 7)
```

### FORTRAN

The library subroutine PAUSE, which is automatically called by a PAUSE statement, generates the SUSPEND EXEC call.

## EXEC CALLS

### Comments

DOS-M prints a message on the system teleprinter when it processes the PROGRAM SUSPEND EXEC call:

*name* SUSP

When the operator restarts the program with a GO, the B-Register contains the address of a five-word parameter array set by the GO request. (The parameters equal zero if no values have been given.) In a FORTRAN program, the library subroutine RMPAR can load these parameters; however, the call to RMPAR must occur immediately following the SUSPEND EXEC call, as in the following example:

```
DIMENSION I (5)
CALL EXEC (7)      (Suspend)
CALL RMPAR (I)     (Return point; get parameters)
```

## PROGRAM SEGMENT LOAD

### Purpose

To load a segment of the calling program from the disc into the segment overlay area and transfer execution control to the segment's entry point. (See Section IV, DOS-M PROGRAMMING, for information on segmented programs.) Follows the :SS condition.

### Assembly Language

```

      EXT  EXEC
      :
      JSB  EXEC          (Transfer control to DOS-M)
      DEF  *+3 (to 8)    (Determine number of parameters)
      DEF  RCODE         (Request code)
      DEF  SNAME         (Segment name)
      DEF  PRAM1         (First optional parameter)
      :
      DEF  PRAM5         (Fifth optional parameter)
      :
RCODE  DEC  8           (Request code = 8)
SNAME  ASC  3,xxxxxx   (xxxxxx is the segment name)
PRAM1  ---             (Up to 5 words of parameter information;
PRAM5  ---             passed to segment as parameters
                        are passed to a suspended program.
                        See PROGRAM SUSPEND.)

```

### FORTTRAN

```

IRCDE = 8
DIMENSION INAME (3)
INAME (1) = xxxxxB      (First two characters)
INAME (2) = xxxxxB      (Second two)
INAME (3)  xxxxxB       (Last character)
CALL EXEC (IRCDE, INAME [,p1...])

```

Comments

In the FORTRAN or ALGOL calling sequence, the name of the segment must be converted from ASCII to octal and stored in the INAME array, two characters per word.

See OVERLAY SEGMENTS and SEGMENTED PROGRAMS, Section IV, for a description of segmented programs.

## SEARCH FILE NAME

### Purpose

To check whether a specific file name exists in the directory of user or system files. (Follows the :SS condition.)

### Assembly Language

```

EXT EXEC
:
JSB EXEC      (Transfer control to DOS-M)
DEF *+4       (Return address)
DEF RCODE     (Request code)
DEF FNAME     (File name)
DEF NSECT     (Number of sectors)
return point
:
RCODE DEC 18   (Request code = 18)
FNAME ASC 3,xxxxx (xxxxx is the file name)
NSECT NOP     (Number of sectors returned here;
               Ø if not found)

```

On return, A-register contains track/sector address of file, and B-register contains the memory address of the track/sector address.

### FORTTRAN

```

IRCDE = 18      (request code)
DIMENSION INAME (3) (File name)
INAME (1) = xxxxxB (First two characters)
INAME (2) = xxxxxB (Next two characters)
INAME (3) = xxxxxB (Last character)
CALL EXEC (IRCDE, INAME, ISECT)

```



# TIME REQUEST

## Purpose

To request the current time.

## Assembly Language

```

EXT EXEC
:
JSB EXEC      (Transfer control to DOS-M)
DEF *+3       (Point of return from DOS-M)
DEF RCODE     (Request code)
DEF ARRAY     (Time value array)
return point  (Continue execution)
:
RCODE DEC 11   (Request code = 11)
ARRAY BSS 5    (Time value array)

```

## FORTRAN

```

IRCDE = 11
DIMENSION ITIME (5)
CALL EXEC (IRCDE, ITIME)

```

## Comments

When DOS-M returns, the time value array contains the time on a 24-hour clock:

ARRAY	or ITIME (1)	= Tens of milliseconds
ARRAY + 1	or ITIME (2)	= Seconds
ARRAY + 2	or ITIME (3)	= Minutes
ARRAY + 3	or ITIME (4)	= Hours
ARRAY + 4	or ITIME (5)	= Not used, but must be present (always = 0)

If DOS-M does not contain a time base generator, all values in the time array are set to zero (0).

# CHANGE USER DISC

## Purpose

To change the subchannel assignment for the user disc.

## Assembly Language

```

EXT EXEC
:
JSB EXEC                (Transfer control to DOS-M)
DEF *+3 (or 4)          (Point of return from DOS-M)
DEF RCODE               (Request code)
DEF LABEL               (Disc Label)
DEF SUBCH               (Disc Subchannel; optional)
return point
:
RCODE DEC 23            (Request code = 23)
LABEL ASC 3, xxxxxx     (Label = xxxxxx)
SUBCH DEC (0 to 7)

```

## FORTRAN

```

IRCDE = 23
DIMENSION LABEL (3)
LABEL (1) = xx
LABEL (2) = xx
LABEL (3) = xx
ICHNL = M              (0 through 7)
CALL EXEC (IRCDE, LABEL, ICHNL)

```

## Comments

1. If both the label and subchannel are specified, DOS-M checks whether the subchannel has that label. If it does, the assignment is made and DOS-M returns. If not, DOS-M prints:

```
LBL = name      (name is label on the subchannel)
or UNLBL
UD nnnnn        (nnnnn = address of EXEC call)
xxxxx SUSP      (xxxxx = name of program)
```

The operator can load a correctly labeled disc on the subchannel and type in

```
:GO
```

This returns to the *beginning* of the EXEC call (not the normal return point) so that the program can reissue the EXEC call.

If the operator does not have a properly labeled disc (or the subchannel is a permanent disc), he should use :OFF or :ABORT.

2. If only a label is specified, DOS-M searches for the label, starting with the highest subchannel. If DOS-M finds the label, it makes the assignment.

If DOS-M cannot find the label, it suspends the program and prints:

```
DISC NOT ON SYS
UD nnnnn
xxxxx SUSP
```

The operator can then abort the program or load a properly labeled disc and type in:

```
:GO
```

This returns to the *beginning* of the EXEC call.

## EXEC CALLS

3. If the label equals "\*" and a subchannel is specified, DOS-M checks whether the subchannel is unlabeled. If it is, DOS-M makes the assignment. If the subchannel is labeled, DOS-M suspends the program and prints:

```
LBL = name
UD nnnnn
xxxxx SUSP      (xxxxx is the program)
```

The operator can then abort the program or load an unlabeled disc on the proper channel and type in:

```
:GO
```

This returns to the *beginning* of the EXEC call.

4. If the label equals "\*" and a subchannel is *not* given, DOS-M searches for an unlabeled disc, starting with the highest subchannel. DOS-M assigns the first unlabeled disc as the user disc, or if no unlabeled discs are found, it suspends the program and prints.

```
DISC NOT ON SYS
UD nnnnn
xxxxx SUSP
```

The operator can then abort the program or load an unlabeled disc and type in:

```
:GO
```

This returns to the beginning of the EXEC call.

If the EXEC call specifies a subchannel with an incorrect system proprietary code (see Appendix A), DOS-M still makes the assignment but prints:

```
TSB DISC or ??? DISC
```

## EXEC CALLS

If the EXEC call specifies a subchannel whose system generation code (see Section VII) does not match that of the system disc, DOS-M still makes the assignment, but prints:

DISC GEN CODE *nnnn* NOT SYS GEN CODE *nnn* ERR POS

The changes made by this EXEC call are only temporary, and will be reset at the end of each job.

If the specified subchannel is not active (physically present), DOS-M aborts the program and prints

UD *nnnnn* (*nnnnn* = address of EXEC call)

# MAIN PROGRAM LOAD

## Purpose

To load a main program from the disc into core and transfer control to its entry point. Follows the :SS condition.

## Assembly Language

```

      EXT  EXEC
      :
      JSB  EXEC          (Transfer control to DOS-M)
      DEF  *+3 (to 8)    (Determine number of parameters)
      DEF  RCODE         (Request code)
      DEF  PNAME         (Program Name)
      DEF  PRAM1         (First optional parameter)
      :
      DEF  PRAM5         (Fifth optional parameter)
      :
RCODE  DEC  10
PNAME  ASC  3,xxxxx     (Program name)
PRAM1  ---              (Up to 5 words of parameter information;
PRAM5  ---              passed to the program as parameters are
                          passed to suspended programs.  See PROGRAM
                          SUSPEND.)

```

## FORTRAN

```

      IRCDE = 10
      DIMENSION INAME(3)
      INAME(1) = xxxxxB   (First two characters)
      INAME(2) = xxxxxB   (Next two characters)
      INAME(3) = xxxxxB   (Last character)
      CALL EXEC (IRCDE,INAME [,p1...])

```

# SECTION IV

## PROGRAMMING

Section IV describes the operating procedures and formatting conventions of the six user programming aids of DOS-M:

- ALGOL Compiler
- FORTRAN Compiler
- FORTRAN IV Compiler
- Assembler
- Relocating Loader
- Relocatable Libraries

Using the EDIT directives, the operator creates and edits files of source programs written in FORTRAN, ALGOL, or Assembly Language. In load-and-go operations the DOS-M FORTRAN Compiler, FORTRAN IV Compiler, ALGOL Compiler and DOS-M Assembler generate relocatable binary code onto temporary disc storage. The DOS-M Relocating Loader can then relocate and merge the code with referenced subroutines of the Relocatable Library. Once loaded, a program is executed by the PROG or RUN directive.

### LOAD-AND-GO FACILITY

The Moving Head Disc Operating System provides the facility for "load-and-go" which is defined as compilation or assembly, loading, and execution of a user program without using intervening object paper tapes. To accomplish this, the compiler or assembler generates relocatable object code from source statements and stores it on the disc in the job binary area of the WORK tracks. Then separate directives initiate loading (PROG, LOADR) and execution (RUN, *program*).

DOS-M stores the object code of several programs and associated subroutines on the disc. The Relocating Loader locates them on the disc, and relocates them into executable absolute program units.

### DOS-M FORTRAN COMPILERS (Basic FORTRAN and FORTRAN IV)

The DOS-M FORTRAN Compilers operate under control of the DOS-M Supervisor. The compilers reside on the disc and are read into core only when needed.

DOS-M FORTRAN and FORTRAN IV are problem-oriented programming languages. Source programs, accepted from either an input device or a user file, are translated into relocatable object programs, punched on paper tape, and optionally, stored in the job binary area of the disc. The object program can be loaded using the DOS-M Relocating Loader and executed using the RUN or PROG directive.

### Compiler Operation

The DOS-M FORTRAN compilers are started by a PROG directive. Before entering the PROG directive, place the source program in the input device, or, if input is from a source file, specify the file with a JFILE directive.



## PROG, FTN [4]

```
:PROG,FTN [ ,p1,p2,p3,p4,99]
:PROG,FTN4[ ,p1,p2,p3,p4,99]
```

Where

$p_1$  = logical unit of input device (standard is 5;  
set to 2 for source file input).

$p_2$  = logical unit of list device (standard is 6).

$p_3$  = logical unit of punch device (standard is 4).

$p_4$  = lines/page on listing (standard is 56).

99 = the job binary parameter. If present, the object  
program is stored in the job binary area for later  
loading. Any requested punch output still occurs.  
(The 99 may occur anywhere in the parameter list,  
but terminates the list.)

$p_1$  through  $p_4$  are optional. If not present, the standard operation is assumed. If 99 is not present, then binary is not placed in the job binary area.

### MESSAGES TO OPERATOR DURING COMPIlation

This message is printed on the operator console when an end-of-tape occurs on device #n:

```
I/O ERR ET EQT #n
```

EQT #n is unavailable until the operator declares it up:

```
:UP,n
```

```
:GO
```

Compilation continues after the GO. More than one source tape can be compiled into one program by loading the next tape before giving the GO.

## PROGRAMMING

At the end of compilation, the following message is printed.

\$END, FTN[4]

If the job binary area (where binary code is stored because of a 99 parameter) overflows, the following message is printed, and compilation continues:

JBIN OVF

There is no further loading into the job binary area.

The compiler terminates if...

□ No JFILE is declared, although logical unit 2 has been given for input. Error E-0019 is printed on the list device. (\$END,FTN[4] is not printed.)

□ There are not enough work tracks for the compiler. The following message is printed:

#TRACKS UNAVAILABLE

□ Colons occur in the first column of a source program entered through the batch device. (Blank cards in the source program are ignored.) The following message is printed.

IE nnnnn

where nnnnn is the memory location of the input request.

## REFERENCE ON FORTRAN IV

The HP FORTRAN IV language is completely described in *FORTRAN IV Reference Manual*, 5951-1321.

## Comments on Basic FORTRAN

### FORTRAN CONTROL STATEMENT

Besides the standard options described in the FORTRAN manual, two new compiler options, T and n, are available. A "T" lists the symbol table for each program in the compilation. If a "u" follows the address of a variable, that variable is undefined (the program does not assign a value to it). The A option includes this T option. If n appears, n is a decimal digit (1 through 9) which specifies an error routine. The user must supply an error routine, ERRn. If this option does not appear, the standard library error routine, ERRØ, is used. The error routine is called when an error occurs in ALOG, SQRT, .RTOR, SIN, COS, .TROI, EXP, .ITOI or TAN.

## PROGRAM STATEMENT

The program statement includes an optional type parameter.

PROGRAM *name* [(*type*)]

where *name* is the name of the program and its main entry point.

When the program is executed using a RUN directive,  
this *name* is used. (It should not equal any file name.)

*type* is a decimal digit specifying the program type.

Only types 3 (main), 5 (segment), and 6 or 7 (library)  
are significant in DOS-M. The type is set to 3 if  
not given.

Seven more parameters may be included but they are used only  
with the Real-Time Executive System. Programs can be compiled  
on DOS-M to be run under Real-Time.

### I/O LOGICAL UNIT NUMBERS

DOS-M FORTRAN function assignments for logical unit numbers are different  
from regular FORTRAN. (See Section V.)

When preparing input data for the batch device, the user never puts a  
colon (:) in column one of a record because the colon in first position  
signifies a directive. DOS-M aborts the job if a directive occurs during  
data input.

## DATA STATEMENT

A new statement, the DATA statement, has been added to DOS-M FORTRAN. DATA sets initial values for variables and array elements. The format of the DATA statement is:

$$\text{DATA } k_1/d_1/, k_2/,\dots,k_n/d_n/$$

where  $k$  is a list of variables and array elements separated by commas,

$d$  is a list of constants or signed constants, separated by commas and optionally preceded by  $j^*$  ( $j$  is an integer constant).

The elements of  $d_i$  are serially assigned to the elements of  $k_i$ . The form  $j^*$  means that the constant is assigned  $j$  times. The  $k_i$  and  $d_i$  must correspond one-to-one.

Elements of  $k_i$  may not be from COMMON.

Arrays must be defined (i.e., DIMENSION) before the DATA statements in which they appear. DATA statements may occur anywhere in a program following the specification statements.

Example,

```
DIMENSION A(3), I(2)
DATA A(1),A(2),A(3)/1.0,2.0,3.0/I(1),I(2)/2*1/
```

## EXTERNAL STATEMENT

With the new statement, EXTERNAL, subroutines and functions can be passed as parameters in a subroutine or function call. For example, the routine XYZ can be passed to a subroutine if XYZ is previously declared EXTERNAL. Each program may declare up to five EXTERNAL routines.

The format of the EXTERNAL statement is

```
EXTERNAL v1,v2,...,v5
```

Where  $v_1$  is the entry point of a function, subroutine or library program.

### EXAMPLE

```
FUNCTION RMX(X,Y,A,B)
  RMX=X(A)*Y(B)
  END
EXTERNAL XYZ,FL1
Z=Q-RMX(XYZ,FL1,3.56,4.75)
```

ERROR E-0018 means too many EXTERNALS.

Note: If a library routine, such as SIN, is used as an EXTERNAL, the compiler changes the first letter of the entry point to "%". Special versions of the library routines exist with the first character changed to "%".

## PAUSE & STOP

PAUSE causes the following message to be printed.

PAUSE *xxxx*

Where *xxxx* is an octal number.

To restart the program, the operator uses a GO directive.

STOP causes the program to terminate after the following message.

STOP *program name xxxx*

Where *xxxx* is an octal number.

### OVERLAY SEGMENTS

Segmented user programs may be written in FORTRAN, but certain conventions are required. A segment must be defined as type 5 in the PROGRAM statement. The segment must be initiated using the PROGRAM SEGMENT LOAD EXEC call from main or segment. A dummy call to main must appear in each segment. In this way, the proper linkage is established between the main and its segments.

Chaining of segments is unidirectional. Once a segment is loaded, execution transfers to it. The segment, in turn, may call another segment using an EXEC call, but a segment written in FORTRAN cannot return to the main program. All communication between the main program and segments must be through COMMON. Segments must not contain DATA Statements.

## ERRØ LIBRARY ROUTINE

ERRØ, the error print routine referred to under the FORTRAN control statement prints the following message whenever an error occurs in a library routine:

*nn xx*

Where *nn* is the routine identifier, and  
*xx* is the error type.

The compiler generates calls to ERRØ automatically. If the FORTRAN control statement includes an *n* option, the call will be to ERR*n*, a routine which the user must supply.

### REFERENCE ON FORTRAN

For a complete description of the FORTRAN language, read the *FORTRAN* programmer's reference manual (02116-9015).

## RTE/DOS ALGOL COMPILER

The RTE/DOS ALGOL Compiler consists of a main program and a data segment. It requires a 16K memory computer and can operate under the control of DOS-M, DOS, or the RTE System. The compiler resides on the disc and is read into core when called for in a :PROG directive. RTE/DOS ALGOL is very similar to the HP ALGOL language described in manual HP 02116-9072. The HP ALGOL compiler implements a language much like ALGOL 60, but it is non-recursive and has I/O capabilities.

Source programs written in DOS-M ALGOL are accepted either from an input device or from a user file and are translated by the ALGOL Compiler into relocatable object programs, punched on paper tape, and optionally, stored in the job binary area of the disc. The object program can be loaded using the DOS-M RELOCATING LOADER and executed using the RUN or PROG directive.

### Compiler Operation

The ALGOL Compiler is started by a PROG directive. Before entering the PROG directive place the source program in the input device, or, if the input is from a source file, specify the file with a JFILE directive. The PROG directive for the ALGOL Compiler should take the following form:



# PROG,ALGOL

:PROG, ALGOL,  $p_1, p_2, p_3, p_4, p_5$

Where

$p_1$  = Input unit (=5 if not specified). Input unit = 2 means source input from disc. The source file has to be specified prior to this statement (by "JFILE" control statement).

$p_2$  = List unit (=6 if not specified).

$p_3$  = Punch unit (=4 if not specified).

$p_4$  = Number of lines on a page (=56 if not specified).

$p_5$  = Load-and-go parameter. To specify load-and-go, set  $p_5=99$ . The value of 99 is reserved for the load-and-go parameter. Its appearance in any position ( $p_1$  through  $p_5$ ) will be interpreted as  $p_5=99$ , and it also signals the end of the parameter list.

## MESSAGES TO OPERATOR DURING COMPILATION

When the end of a source tape is encountered, the following will be output on the system teleprinter:

I/O ERR ET EQT # $n$

The compiler will wait until the following messages are entered on the system teleprinter:

:UP, $n$

:GO

## PROGRAMMING

At the end of the compilation, the following message is output to the system teleprinter:

\$END, ALGOL

If the job binary area (where binary code is stored because of a "99" parameter in the PROG directive) overflows, the following message is output by the system teleprinter and compilation continues:

JBIN OVF

The compilation will be completed, but there will be no further loading of binary code into the job binary area.

The compiler terminates if...

- ⌈ No JFILE is declared, although logical unit 2 had been specified as  $p_1$  of the PROG directive. The following message is output:

NO SOURCE

- ⌈ The first statement of the source file specified by the PROG directive  $p_1$  parameter does not begin with the word HPAL. Or the control statement contains an error. The following message is output:

HPAL??

- ⌈ A colon occurs in the first position of a source statement line. The following message is output:

IE *nnnnn*

where *nnnnn* is the memory location of the input request.

ALGOL Control Statement

The word HPAL is mandatory. Any combination of the following symbols may appear next, separated by commas:

- L: produce source program listing
- A: produce object code listing
- B: produce object tape
- P: a procedure only is to be compiled

If no symbols are specified, the program will run but will not produce any output other than diagnostic messages and job binary (if requested). A program name in quotes (the NAM-record name which must be a legitimate identifier without blanks) must follow the symbols. (It should not equal any file name.)

Sense switch control is not used with DOS-M. Two parameters may be specified following the NAM-record name.

$p_1$  is a decimal digit between 0 and 9 specifying the name of the error routine to be called if an error occurs in ALOG, SQRT, .RTOR, SIN, COS, .RTOI, EXP, .ITOI, TAN. The name of the error routine is ERR $n$ , where  $n = p_1$  or  $n = 0$  if  $p_1$  is not specified. ERR0 is supplied in the Relocatable Library, all other error routines must be supplied by the user.

$p_2$  is a decimal digit specifying the type of the program: 3 for a main program, 5 for a segment, and 6 or 7 for a utility subroutine or procedure. If  $p_2$  is not specified, the type is set to 3 for main programs and to 7 for procedures (P option in the control statement).

## EXAMPLE

HPAL,L,B,"TEST",1,3

## ALGOL Segmentation

ALGOL programs can be segmented if certain conventions are followed. A segment must be defined as type 5 in the HPAL statement. The segment must be initiated by using the PROGRAM SEGMENT LOAD EXEC call from the main or another segment.

In order to establish the proper linkage between a main program and its segments, each segment must declare the main a code procedure. For example, if MAIN is the main program, the following must be declared in each segment.

PROCEDURE MAIN; CODE:

Chaining of segments is unidirectional. Once a segment is loaded, execution transfers to it. The segment, in turn, may call another segment using an EXEC call, but a segment written in ALGOL cannot return to the main program.

## ALGOL I/O

The HP ALGOL I/O statements should specify the proper logical unit numbers for the DOS-M configuration. (See Section V.)

## ALGOL Error Messages

See the manual HP ALGOL (HP 02116-9072) for the meanings of HP ALGOL compilation time and run time error messages.

## DOS-M ASSEMBLER

The DOS-M Assembler, a segmented program that executes in the user program area of core, operates under control of DOS-M. The Assembler consists of a main program (ASMB) and six segments (ASMBD, ASMB1, ASMB2, ASMB3, ASMB4, ASMB5), and resides on the disc.

DOS-M Assembly Language, a machine-oriented programming language, is very similar to the HP Extended Assembly Language. Source programs, accepted from either an input device or a user source file on the disc, are translated into absolute or relocatable object programs; absolute code is punched in binary records, suitable for execution only outside of DOS-M. ASMB can store relocatable code in the load-and-go area of the disc for on-line execution, as well as punch it on paper tape. The DOS-M Relocating Loader accepts assembly language relocatable object programs from paper tape, the load-and-go area, and user files.

A source program passes through the input device only once, unless there is insufficient disc storage space. In the latter case, two passes are required.

## Assembler Operation

The DOS-M Assembler is started by a PROG directive. However, before entering the PROG directive, the operator must place the source program in the input device. If the source program is on the disc, the operator must first specify the file with a JFILE directive, and set parameter  $p_1$  = logical unit 2 in the PROG directive.

# PROG,ASMB

```
:PROG,ASMB, $p_1$ , $p_2$ , $p_3$ , $p_4$ ,99
```

Where

$p_1$  = logical unit of input device (5 is standard; 2 is used for source file input indicated by a JFILE directive)

$p_2$  = logical unit of list device (6 is standard)

$p_3$  = logical unit of punch device (4 is standard)

$p_4$  = lines/page on listing (56 is standard)

99 = job binary parameter. If present, the object program is stored in the job binary area for later loading. Any requested punching still occurs. The 99, which may follow any parameter in the list, terminates the list.

## MESSAGES DURING ASSEMBLY

The messages described in this section are printed at the teleprinter console or in the program listing.

When an end-of-tape occurs on device # $n$ , this message appears on the system teleprinter:

```
I/O ERR ET EQT # $n$ 
```

EQT # $n$  is unavailable until the operator declares it up and restarts the assembler by means of a GO directive:

```
:UP, $n$ 
```

```
:GO
```

## PROGRAMMING

Thus, more than one source tape can be assembled into one program. The next tape is loaded each time the input device goes down. The program should be placed in the input device before entering the GO.

The following message on the system teleprinter signifies the end of assembly:

\$END ASMB

If another pass of the source program is required, the message is printed on the system teleprinter at the end of pass one.

\$END ASMB PASS

The operator must replace the program in the input device and type:

:GO

If an error is found in the Assembler control statement, the following message is printed on the system teleprinter:

\$END ASMB CS

The current assembly stops.

If an end-of-file condition on source input occurs before an END statement is found, the teleprinter signals:

\$END ASMB XEND

The current assembly stops.

If source input for logical unit 2 (disc) is requested, but no file has been declared (see JFILE, Section II), the system teleprinter signals:

\$END ASMB NPRG

If the job binary area, where binary code is stored by a 99 parameter, overflows, assembly continues but the following message is printed on the system teleprinter:

JBIN OVF

However, no binary code is stored in the job binary area.

## PROGRAMMING

The next message is associated with each error diagnostic printed in the program listing during pass 1.

# *nnn*

*nnn* is the "tape" number on which the error (reported on the next line of the listing) occurred. A program may consist of more than one tape. The tape counter starts with one and increments by one whenever an end-of-tape condition occurs (paper tape) or a blank card is encountered. When the counter increments, the numbering of source statements starts over at one.

Each error diagnostic printed in the program listing during pass 2 of the assembly is associated with a different message:

PG *ppp*

*ppp* is the page number (in the listing) of the *previous* error diagnostic. PG 000 is associated with the first error found in the program.

These messages (#*nnn* and PG *ppp*) occur on a separate line, just above each error diagnostic in the listing.

### DOS-M Assembly Language

The DOS-M Assembly Language is equivalent to extended assembly language, as defined in the *ASSEMBLER* programmer's reference manual (02116-9014). A few language changes are required to run under DOS-M; programs must request certain functions, such as I/O, from the executive. These requests are made using the EXEC calls described in Section III.

### ASSEMBLER CONTROL STATEMENT

The control statement has the same form as that of regular assembly language; and although only relocatable code can be run under DOS-M, the DOS-M Assembler is able to assemble absolute code if it is specified. Absolute code is never



## PROGRAMMING

stored in the job binary area. To get absolute code, the control statement must include an "A". The "R", however, is not required for relocatable code. An "X" causes the assembler to generate non-extended arithmetic unit code.

### Examples

ASMB,L,B	List and Punch Relocatable Binary
ASMB,R,L,B,X	List and Punch Relocatable, non-EAU Binary.
ASMB,T,L	List and Print Symbol Table.
ASMB,A,B,L	List and Punch Absolute Binary.

## ORB STATEMENT

DOS-M Assembly Language does not contain the ORB statement, since information cannot be loaded into the protected base page area by user programs. However, programs can read information from base page using absolute address operands up to  $1777_8$ .

## INPUT/OUTPUT

DOS-M has different function assignments for the logical unit numbers.  
(See Section V.)

When preparing input for the batch device, the programmer must remember to never put a colon (:) in column one of a source statement. DOS-M aborts the current program if a directive (signified by : in column one) occurs during data input.

If present, the memory protect option protects the resident supervisor from alteration and interrupts the execution of a user program under these conditions:

- Any operation that would modify the protected area or jump into it.
- Any I/O instruction, except those referencing the switch register or overflow.
- Any halt instruction.

Memory protect gives control to DOS-M when an interrupt occurs, and DOS-M checks whether it was an EXEC call. If not, the user program is aborted.

## PROGRAMMING

# NAM STATEMENT

The NAM psuedo-instruction allows up to eight optional parameters. (The last seven parameters are used only by programs to be executed under the Real-Time Executive System.) Only the first parameter is significant in DOS-M. If the first parameter equals 3, the program is a main program; if 5, a program segment; if 6, a library routine; if 7, a subroutine. If the parameter equals another number, the assembler and DSGEN will accept it, but the Relocating Loader will not. (See Section VI for DSGEN program type codes.)

NAM *name* [,*type*]

where *name* is the name of the program (it should not equal any file name), and

*type* is the type code.

In addition to the *name* defined by NAM, each program has one or more *entry points* defined by an ENT statement with the exception of the main program. The transfer address on the END statement is sufficient for the main program (type 3). *Name* is used in programmer-to-DOS-M communication, while the *entry points* are program-to-program communication.

### Segmented Programs

User programs may be structured into a main program and several segments, as shown in Figure 4-1. The main program begins at the start of the user program area. The area for the segments starts immediately following the last location of the main program. The segments reside on the disc, and are read into core by an EXEC call, when needed. Only one segment may be in core at a time. When a segment is read into core, it overlays the segment previously in core.

The main program must be type 3, and the segments must be type 5. When using DSGEN to configure the system or loading programs with LOADR, the main program must be entered prior to its segments. One external reference from each segment to the main routine is required for DSGEN to link the segments and main programs. Also, each segmented program should use unique external reference symbols. Otherwise, DSGEN or LOADR may link segments and main programs incorrectly.

# PROGRAMMING

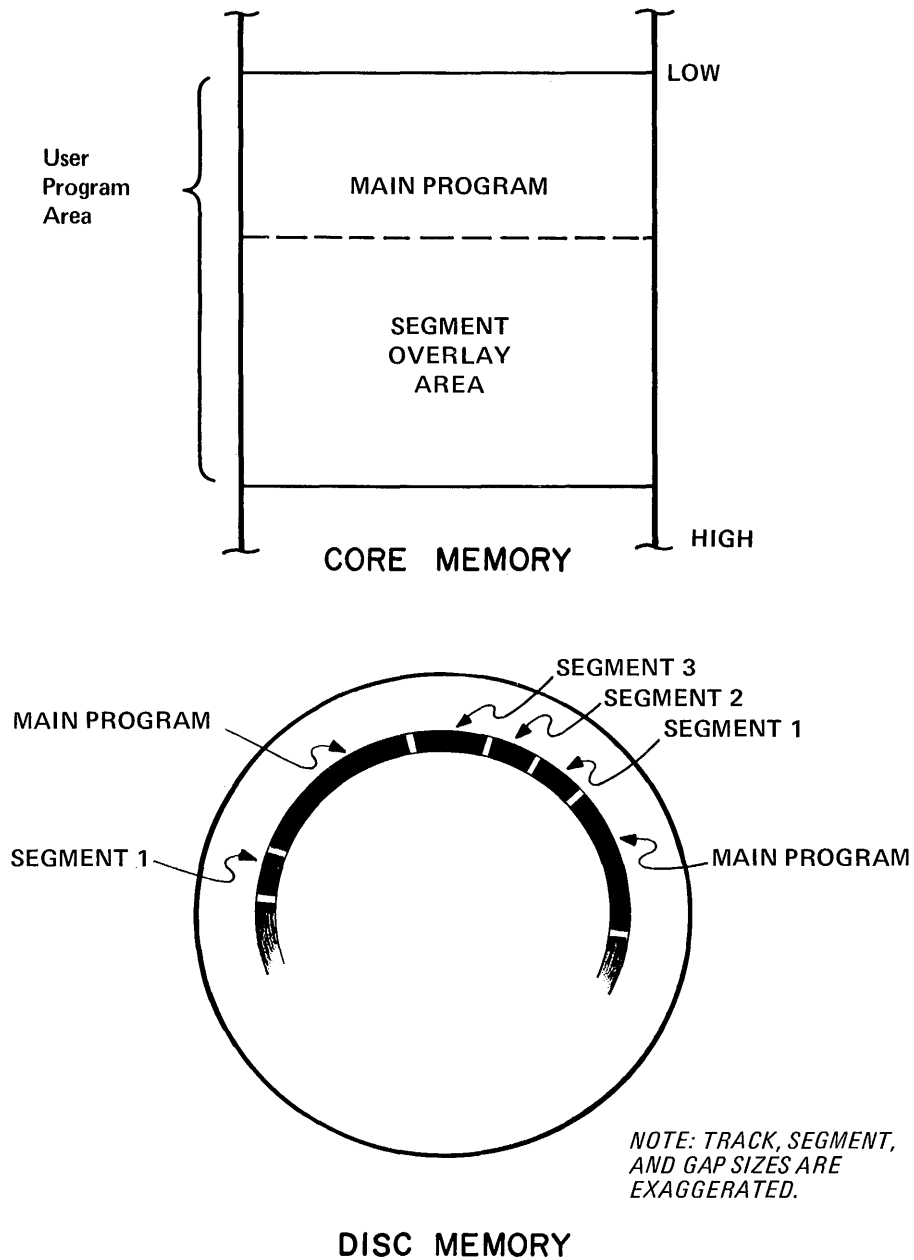


Figure 4-1. Segmented Programs

## PROGRAMMING

Figure 4-2 shows how an executing program may call in any of its segments from the disc using the PROGRAM SEGMENT LOAD EXEC request (1-2). DOS-M locates the segment on the disc (3-4), loads it into core (5) and begins executing it. The segment may call in another of the main program's segments using the same EXEC request (6).

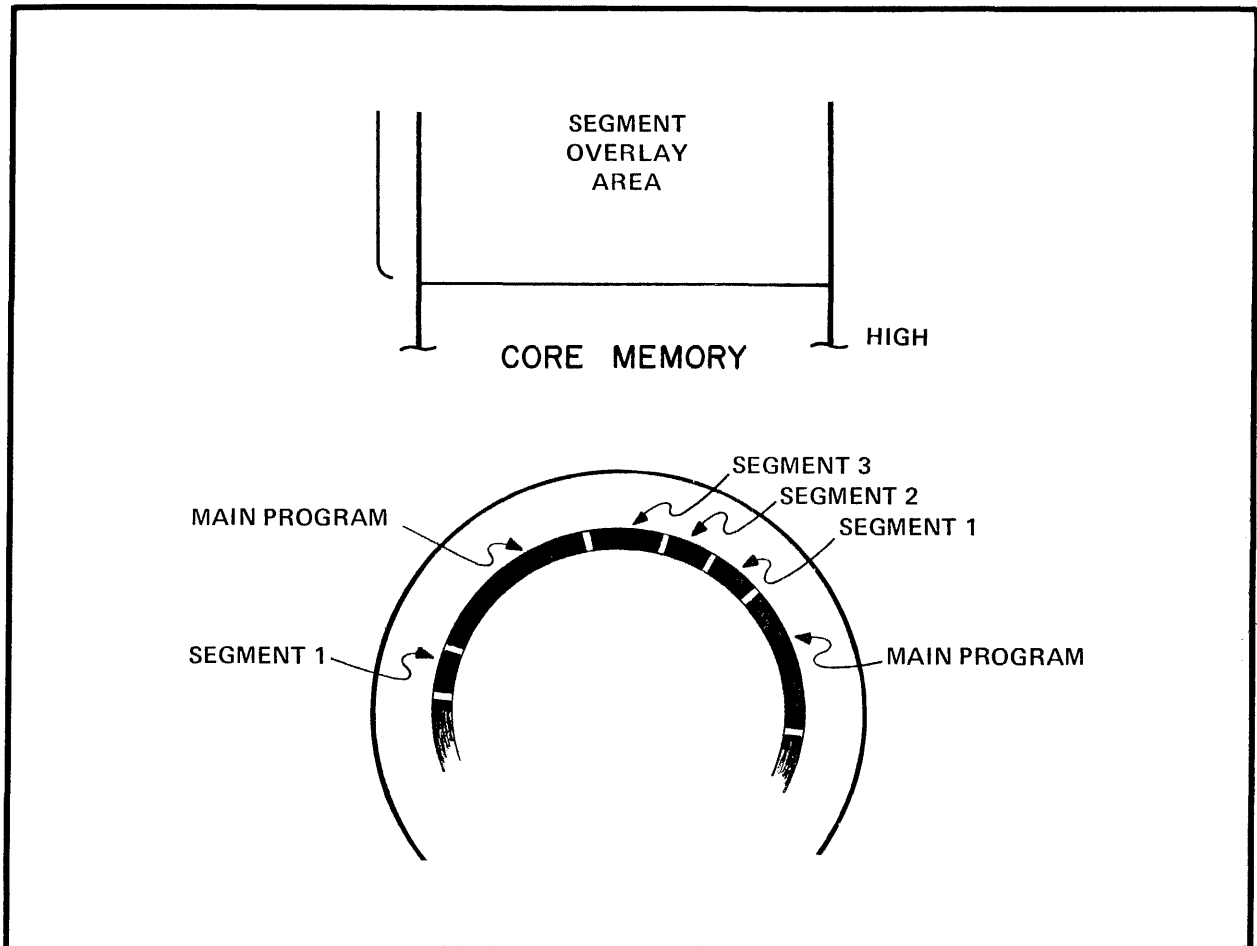


Figure 4-2. Main Calling Segment

## PROGRAMMING

Figure 4-3 shows how DOS-M processes the request from the segment (7) by locating the segment on the disc (8-9), loading it into core (10), and beginning execution of it.

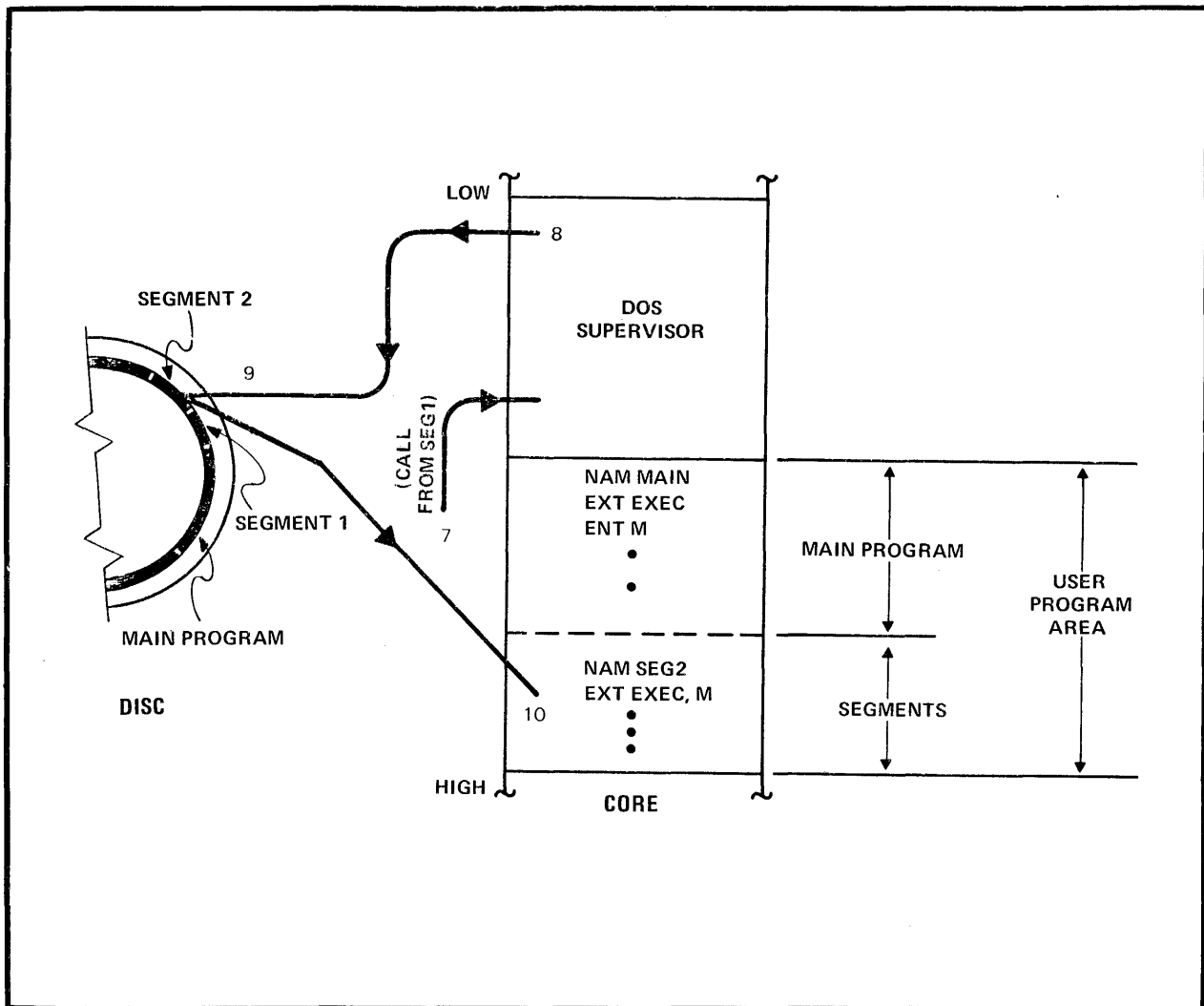


Figure 4-3. Segment Calling Segment

When a main program and segment are currently residing in core, they operate as a single program. Jumps from a segment to a main program (or vice versa) can be programmed by declaring an external symbol and referencing it via a JMP or JSB instruction. (See Figure 4-4.) A matching entry symbol must be defined as the destination in the other program. DSGEN associates

## PROGRAMMING

the main programs and segments, replacing the symbolic linkage with actual absolute addresses (i.e., a jump into a segment is executed as a jump to a specific address). The programmer should be sure that the correct segment is in core before any JMP instructions are executed.

### Reference on Assembly Language

Consult the *ASSEMBLER* programmer's reference manual (02116-9014) for a full description of assembly language.

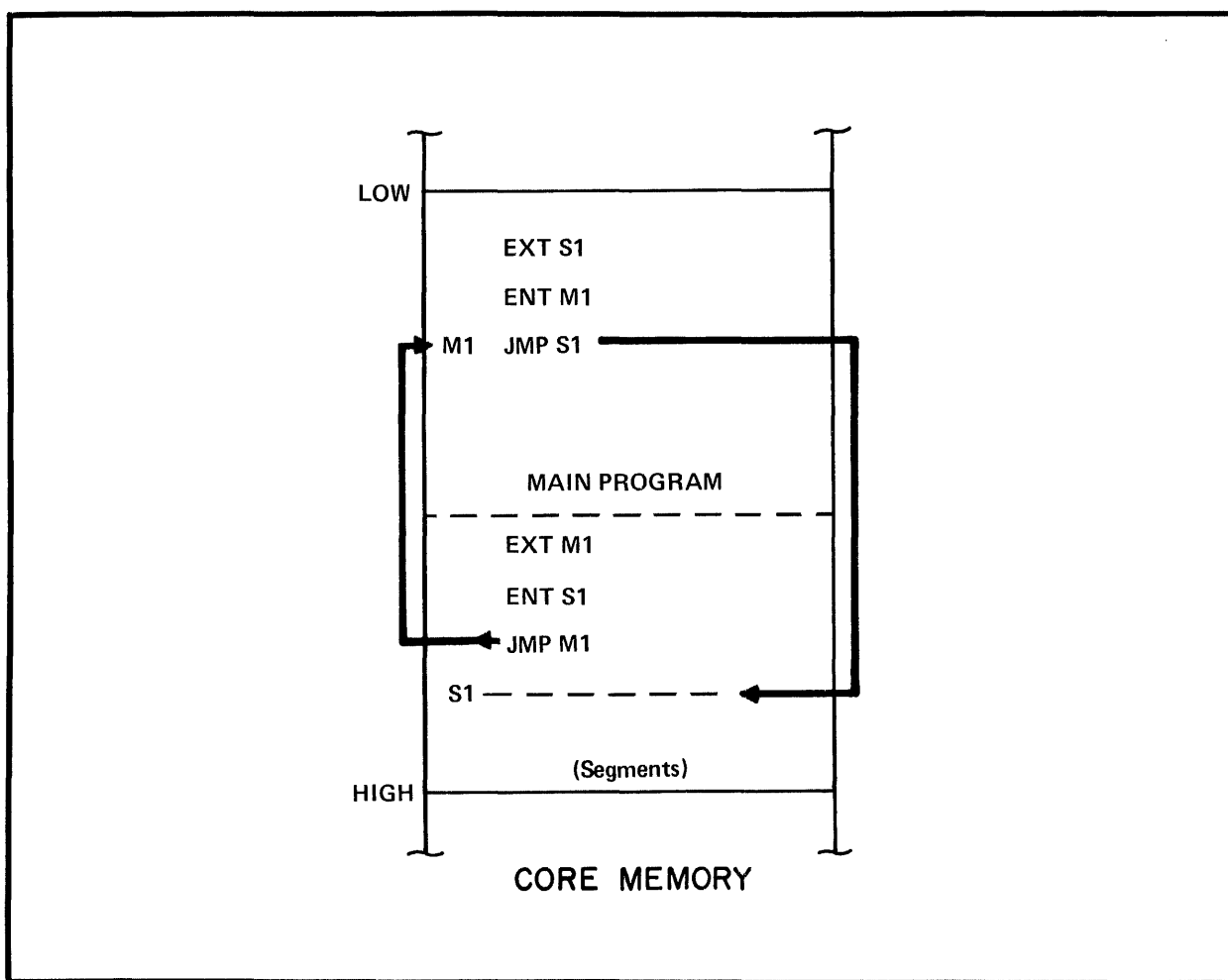


Figure 4-4. Main-to-Segment Jumps

## DOS-M RELOCATING LOADER

The DOS-M Relocating Loader accepts relocatable object programs which have been translated by the DOS-M Assembler, RTE/DOS ALGOL Compiler or DOS-M FORTRAN Compiler. It generates an executable core image of each such program on the disc. The relocatable programs may enter the loader as

- Job binary area programs translated during the current job,
- User files,
- Punched tapes, magnetic tapes, or
- Subroutines from the disc-resident Relocatable Library.

Each main program is relocated to the start of the user area and linked to its external references, such as library routines. Segments will overlay the area following the main program and its subroutines. Programs may run under control of the DEBUG library routine. The main program, plus its subroutines and its longest segment, can be as large as the user area. With a RUN or PROG directive, the program is called by name from the disc and executed, or the program is stored as a permanent user file to be run during a later job. If the JBIN is used, the loader may be executed only once during each job, so all load-and-go assemblies or compilations must be done prior to calling the loader.

Starting the Loader

The DOS-M Relocating Loader is initiated by a PROG directive from the batch or keyboard device.

**PROG,LOADR**Format

:PROG,LOADR[ $P_1, P_2, P_3, P_4, P_5$ ]

$P_1$  determines the relocatable object program input combination:

$P_1 = \emptyset$  for loading from jbin and relocatable library.

$= 2$  for loading from jbin, user files, and relocatable library.

$= n$  for loading from jbin, user files, relocatable library and paper tape, or magnetic tape (logical unit  $n$ ).

$P_2 =$  list device logical unit.

$P_3 = \emptyset$  for no DEBUG,  $\neq \emptyset$  for DEBUG.

$P_4 = \emptyset$  for list of program load map,  $\neq \emptyset$  for none.

$P_5 = \emptyset$  for list of entry point addresses,  $\neq \emptyset$  for none.

If values  $P_1, \dots, P_5$  are not set,  $P_1 = \emptyset$ ,  $P_2 = 6$ ,  $P_3 = P_4 = P_5 = \emptyset$ .

Comments

Selecting the DEBUG option causes DEBUG to be appended to each main program and segment. The loader sets the primary entry point of each to DEBUG, rather than the user routine. When the program is run, DEBUG takes control of the program's execution and seeks instructions from the keyboard.



## RELOCATABLE FILES

A list of relocatable file names follows the PROG directive (unless  $P_1$  equals  $\emptyset$ ). In batch mode, the list starts on the next record and stops at "/E". In keyboard mode, the loader prints

ENTER FILE NAME(S) OR /E

then waits for input. After each list of files is entered, the message repeats until a /E is entered. In batch mode the list of files follows the PROG directive on the batch input device.

*file-name 1, file-name 2,.../E*

The file list is a series of records containing file names separated by commas, ending with a "/E." All programs in each file are loaded unless a particular subset of the file is specified:

*file-name (prog 1, prog 2...)*

Only the programs specified within the parenthesis are loaded from the *file-name*. The file list is simply a "/E" if no files are to be loaded. (The search for these files is made only on the current user disc; the loader is unaffected by :SS.)

Operating the Loader

## SCANNING THE PROGRAMS

The loader scans the relocatable binary programs and maintains two tables--one of program names, and another of entry points and externals. Since mains are matched with segments during the scan, each main program must occur before the associated segments. Programs from tape are stored on the work tracks as they are read in.

If the job binary area contains any programs, it is scanned first. User files given in the file list (if any) are scanned for entries and externals.

## PROGRAMMING

If paper tape input is requested, the following messages are printed,

```
LOAD TAPE
LOADR SUSP
@
```

The loader suspends. The operator places a tape in the input device and types

```
:GO
```

When an end-of-tape condition occurs, three messages are printed on the system teleprinter:

```
I/O ERR ET EQT# nn (paper tape only -- not magnetic tape)
LOAD TAPE
LOADR SUSP
@
```

The operator places the next tape in the input device, enters :UP,n, and :GO to read the next tape. Enter :GO,1 to indicate that all tapes have been read in.

### Matching Entries with Externals

After matching all possible entry points and external references in the user programs, the loader scans the Relocatable Library (disc-resident) looking for entry points to match the undefined external references. If undefined external references still exist,

```
UNDEFINED EXTS
```

is printed and the external references are listed, one per line.

## PROGRAMMING

To load additional programs from paper tape, the operator types:

:GO,0[,n]

where  $n$  is the logical unit number of the input device, if different from  $P_1$  of the PROG,LOADR directive.

To continue without fulfilling external references, the operator types:

:GO,1

To specify a file name from the keyboard, the following directive is typed:

:GO,2

### RELOCATION

The main and segment names become user file names once the programs are loaded. To ensure unique file names, the loader compares all program and segment names against the names of previous system and user files (current user disc only). If duplicate names occur, an error message is printed and loading stops.

The loader converts each main program into an absolute core image, stores it on the disc, places the name in the user directory where it remains during the current job, and lists the program address map and entry points, if requested. After each main program, any associated segments are loaded in the same way. When the loader is completely finished, the following message is printed:

LOADR COMPLETE

During the current job, the absolute core images appear in the user file area (see LIST directive, Section II) and can be executed by name (see RUN and PROG directives). At the end of the job, however, they disappear from the file area, unless they are made permanent files by means of the STORE, P directive.

## PROGRAMMING

If no programs are entered, the loader prints the following messages and terminates:

NO PROGRAMS LOADED.

LOADR COMPLETE

### DEBUG Library Subroutine

RTE/DOS DEBUG, a subroutine of the Relocatable Library, allows programmers to check for logical errors during execution. If the  $P_3$  parameter of the PROG, LOADR directive equals 1, the loader combines DEBUG with the user program being loaded. The primary entry point (the location where execution begins) is set to DEBUG. Therefore, when the program is executed with a RUN directive, DEBUG takes control and prints the message:

BEGIN 'DEBUG' OPERATION

The programmer now enters any legal debug operation. DEBUG ignores illegal requests and prints a message:

ENTRY ERROR

## PROGRAMMING

### DEBUG OPERATIONS

B,A	Instruction breakpoint at address A. (NOTE: if A = JSB EXEC, a memory protect violation occurs.)
D,A,N <sub>1</sub> [,N <sub>2</sub> ]	ASCII dump of core address N <sub>1</sub> or from N <sub>1</sub> to N <sub>2</sub> .
D,B,N <sub>1</sub> [,N <sub>2</sub> ]	Binary dump of core address N <sub>1</sub> or from N <sub>1</sub> to N <sub>2</sub> .
M,A	Sets absolute base of relocatable program unit.
R,A	Execute user program starting at A. Execute starting at next location in user program (used after a breakpoint or to initiate the program at the transfer point in the user program).
S,A <sub>1</sub> ,D <sub>1</sub>	Set D <sub>1</sub> in location A <sub>1</sub> .
S,A <sub>1</sub> ,D <sub>1</sub> ,D <sub>n</sub>	Set D <sub>1</sub> to D <sub>n</sub> in successive memory locations beginning at location A <sub>1</sub> .
W,A,D <sub>1</sub>	Set A-Register to D <sub>1</sub> .
W,B,D <sub>2</sub>	Set B-Register to D <sub>2</sub> .
W,E,D <sub>3</sub>	Set E-Register (Ø=off, non-zero=on).
W,O,D <sub>4</sub>	Set Overflow (Ø=off, non-zero=on).
X,A	Clear breakpoint at address A.
A	Abort Debug operation.

Loader Example

In the following example, DOS-M is in keyboard mode.

:PROG,LOADR,5,6,0,0,0	Paper tape input is specified.
ENTER FILE NAME(S)OR/E	No files are specified.
/E	
LOAD TAPE	Place paper tape in input device.
LOADR SUSP	
@:GO	Return to loader.
I/O ERR ET EQT # 03	End of tape.
LOAD TAPE	Put in next tape.
LOADR SUSP	
@:UP,3	Declare input device ready.
@:GO	
I/O ERR ET EQT # 03	
LOAD TAPE	
LOADR SUSP	
@:UP,3	
@:GO	
I/O ERR ET EQT # 03	
LOAD TAPE	
LOADR SUSP	
@:UP,3	
@:GO	Repeat tape loading process 4 times.
I/O ERR ET EQT # 03	
LOAD TAPE	
LOADR SUSP	
@:UP,3	
@:GO	
I/O ERR ET EQT # 03	
LOAD TAPE	
LOADR SUSP	
@:UP,3	
@:GO,1	No more paper tapes.

# PROGRAMMING

## RELOCATING LOADER

NAME/ENTRY	ADDR	
QA1	12000	Main program, starting address.
*QA1	12076	Main program, entry point.
QA1A	12200	
*QA1A	12201	
QA1B	12262	
*QA1B	12263	
QA1C	12336	
*QA1C	12337	
QA1D	12364	
*QA1D	12365	
FRMTR	12431	
*.D10.	14612	
*.B10.	14665	
*.I01.	14507	
*.I0R.	14462	Subroutine starting addresses and entry points. Asterisk signifies entry point.
*.IAR.	14546	
*.RAR.	14522	
*.DTA.	14710	
.ENTR	15162	
*.ENTR	15162	
.FLUN	15230	
*.FLUN	15230	
.PACK	15243	
*.PACK	15243	
FLOAT	15350	
*FLOAT	15350	
IFIX	15355	
*IFIX	15355	
LOADR COMPLETE		End of Loading.

Loader Error Messages

During its operation the loader may print one of the following error messages on the keyboard:

<u>Message</u>	<u>Error Messages</u>
L01	Checksum error on tape
L02	Illegal record
L03	Memory overflow
L04	Base page overflow
L05	Symbol table overflow
L06	Duplicate main or segment name (may be caused by attempting to run the loader twice in one job)
L07	Duplicate entry point
L08	No main or segment transfer address
L09	Record out of sequence
L10	Insufficient directory or work area space
L11	Program name table overflow
L12	User file specified cannot be found
L13	Program name duplication
L14	Non-zero base page length
L15	Segment occurred before main
L16	Program overlay (illegal ORG)
L17	Illegal library record.

The loader aborts (programmer must start over) on each of these conditions and prints a message.

LOADR TERMINATED



## THE RELOCATABLE LIBRARIES

There are two libraries, or collections of relocatable subroutines that can be used by DOS-M: the RTE/DOS Relocatable Library (EAU or Non-EAU versions) and the RTE/DOS FORTRAN IV Library. These libraries contain mathematical routines such as SIN and COS, and utility routines such as BINRY, etc. A program signifies its need for a subroutine by means of an "external reference." External references are generated by EXT statements in assembly language, by CALL statements and the compiler in FORTRAN, and by CODE procedures and the compiler in ALGOL.

When the system is generated, several combinations of libraries are possible. Every system should contain an RTE/DOS Relocatable Library: either an EAU version or a non-EAU version, depending on the computer hardware. This library does not contain a formatter, but the FORTRAN IV Library contains a formatter that handles extended precision numbers. If extended precision arithmetic is not needed, a separate RTE/DOS Basic FORTRAN Formatter is available to take the place of the FORTRAN IV Library.

All of these libraries and the subroutines they contain are documented in the *Relocatable Subroutines* manual (02116-9032).

## SECTION V

# INPUT/OUTPUT

In the Moving-Head Disc Operating System, centralized control and logical referencing of I/O operations effect simple, device-independent programming. Each I/O device is interfaced to the computer through one or more I/O channels which are linked by hardware to corresponding core locations for interrupt processing. By means of several user-defined I/O tables, multiple-device drivers, and program EXEC calls, DOS-M relieves the programmer of most I/O problems.

### SOFTWARE I/O STRUCTURE

An Equipment Table records each device's I/O channels, driver entry points, DMA requirements, and location on disc if disc-resident. A Device Reference Table (logical unit table) assigns an equipment table number to each of its entries, thus allowing the programmer to reference changeable logical units instead of fixed physical units.

An Interrupt Table relates each channel to an entry in the Equipment Table.

A driver is responsible for initiating and continuing operations on all devices of an equivalent type.

The programmer requests I/O by means of an EXEC call in which he specifies only the logical unit, control information, buffer location, buffer length, and type of operation.

# INPUT/OUTPUT

## The Equipment Table

The Equipment Table (EQT) has an entry for each device recognized by DOS-M (these entries are established by the user when DOS-M is generated). The EQT entries reside in the permanent core-resident part of the system and have this format:

WORD	CONTENTS															
1	Driver "Initiation" Section Address															
2	Driver "Continuation" Section Address															
3	D	R					Unit #		Channel #							
4	Av		Equipment Type Code				Status									
5	(saved for driver use)															
6	(saved for driver use)															
7	Request Return Address															
8	Request Code															
9	Current I/O Request Control Word															
10	Request Buffer Address															
11	Request Buffer Length															
12	Temporary or Disc Track #															
13	Temporary or Starting Sector #															
14	Temporary Storage for Driver															
15	Upper Memory Address of Main Driver Area															
16	Upper Memory Address of Driver Linkage Area															
17	Starting Track #								Starting Sector #							
BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

<

} 0's if  
core-  
resident

D = 1 if DMA channel required.

R = 1 if driver type is core-resident.

Unit # May be used for sub-channel addressing.

Channel # I/O select code for device (lower number if multiboard interface.)

## INPUT/OUTPUT

Av        = 0 - Unit not busy and available  
          1 - Unit disabled (down)  
          2 - Unit busy

Status - Actual or simulated unit status at end of operation.

Equipment Type Code - Identifies type of device and associated software driver. Assigned equipment type codes in octal are:

00-07	Paper Tape Devices
00	Teleprinter
01	Punched Tape Reader
02	High Speed Punch
05	Teletype (System)
10-17	Unit Record Devices
10	Reserved for Plotter
12	Line Printer
15	Mark Sense Card Reader
20-37	Magnetic Tape/Mass Storage and other devices capable of both input and output
22	3030 Magnetic Tape
31	Moving-Head Disc

For equipment type codes 01 through 17, odd numbers indicate input devices and even numbers indicate output devices (except 05, which is both input and output).

When DOS-M initiates or continues an I/O operation, it places the address of the EQT entry for the device into the base page communication area (see Appendix A) before calling the driver routine.

Logical Unit Numbers

Logical unit numbers from  $1_{10}$  to  $63_{10}$  provide logical addressing of the physical devices defined in the EQT. These numbers are maintained in the Device Reference Table (DRT or logical unit table), which is created by the System Generator (DSGEN) and can be modified by the LU directive.

Each one-word entry in the DRT contains the EQT entry number of the device assigned to the logical unit. DOS-M has the following function assignments for logical unit numbers.

<u>Logical Unit Number</u>		<u>Function</u>
Restored after each :JOB.	1	System Teleprinter
	2	User Mass Storage
	3	System Mass Storage
	4	Standard Punch Device
	5	Standard Input Device
	6	Standard List Device
	7	Unassigned Recommended for magnetic tape Can be assigned to any device by user
	8	
	9	
	$10_{10}$	
	:	
	$63_{10}$	

The user determines the number of logical units when the system is generated. At the beginning of each JOB, logical units 1 through 9 are restored to the values set by DSGEN (System Generator), whereas  $10_{10}$  through  $63_{10}$  are restored only on a start-up from the disc.

Executing programs use logical unit numbers to specify the type of device for I/O transfers. In an I/O EXEC call, the program simply specifies a logical unit number and does not need to know which actual device or which I/O channel handles the transfer.

## The Interrupt Table

The interrupt table contains an entry, established at system generation time, for each I/O channel in the computer which can cause an interrupt. The entry contains the address of the EQT entry for the device on the channel.

The interrupt locations in core contain a jump subroutine to \$CIC which is the central interrupt control routine which examines the interrupt table to decide what action to take. On a power failure interrupt, DOS-M halts.

## Input/Output Drivers

The I/O driver routines, either core-or disc-resident, handle the actual transfer of information between the computer and external devices. When a transfer is initiated, DOS-M places the EQT entry addresses into the base page communication area and jumps to the driver entry point. The driver configures itself for the particular channel (in this way the same driver can handle several devices of the same type on many channels), initiates the transfer and returns to DOS-M. When an interrupt occurs on the channel, indicating continuation or completion of the transfer, DOS-M again transfers control to the driver. DOS-M contains only two drivers: the Moving-Head Disc Driver (DVR31) and the System Teleprinter Driver (DVR05). However, these drivers of the Disc Operating System (DOS, fixed-head disc/drum) are fully compatible with DOS-M:

DVR00	-	Teleprinter
DVR01	-	Photo-reader
DVR02	-	High speed punch
DVR10	-	Plotter
DVR12	-	Line Printer
DVR15	-	Mark Sense Card Reader
DVR22	-	3030 Magnetic Tape
DVR23	-	7970 Magnetic Tape

The driver name consists of the letters "DVR" added to the equipment type code. In addition, the programmer can write drivers for special devices, following the guidelines in this section. The driver is only responsible for updating the status field in the EQT entry; DOS-M handles the availability field.

### System I/O

DOS-M itself initiates many I/O transfers. It reads in directives from the batch or keyboard device and transfers modules in from the disc. These functions are accomplished by \$SYIO, a routine within the DOS-M Supervisor, which calls the appropriate driver routine.

### User Program I/O

The user program initiates an I/O transfer by means of an EXEC call--a "JSB EXEC" as described in Section III. The supervisor recognizes the EXEC call as an I/O request and sends it along to the I/O supervisor EXEC MODULE (\$EX18) which determines if the driver for the requested device is core-resident. If not, the driver is read into core from the disc.

\$EX18 places the address of the EQT entry in the base page communication area (see Appendix A, *TABLES*) and transfers control to the driver. The driver configures itself to I/O operation on the appropriate channel, initiates the transfer and returns to \$EX18. DOS-M either returns to the executing user program or waits until the I/O transfer is complete as requested by the program.

Interrupt Processing

When an interrupt occurs on the computer, control is transferred to the instruction in the interrupt location corresponding to the device. Each interrupt location (memory locations 4<sub>8</sub> through 37<sub>8</sub>) contains a "JSB \$CIC" instruction. \$CIC, the central interrupt control routine of DOS-M, then performs the following:

- a. Disables interrupt system
- b. Saves registers, point of program suspension
- c. Clears interrupt flag
- d. Determines the type of interrupt
  - 1) If power fail, halts
  - 2) If memory protect, goes to EXEC (goes directly to EXEC if no memory protect)
  - 3) If time base, goes to CLOCK routine (if installed)
  - 4) If not a legal I/O channel, returns to suspension point
  - 5) If legal I/O channel, puts EQT entry addresses in base page communication address and transfers to driver continuation address
- e. Upon return from the I/O driver, turns on interrupt system, restores registers, and returns to the point of suspension.

PLANNING I/O DRIVERS

Before attempting to program an I/O driver, the programmer should be thoroughly familiar with Hewlett-Packard computer hardware I/O organization, interface kits, computer I/O instructions and Direct Memory Access (DMA).

An I/O driver, operating under control of the Input/Output Control (\$EX18) and Central Interrupt Control (\$CIC) modules of DOS-M, is responsible for all data transfer between an I/O device and the computer. The device equipment table (EQT) entry contains the parameters of the transfer, and the base page communication area contains the number of the allocated DMA channel, if required.



## INPUT/OUTPUT

An I/O driver includes two relocatable, closed subroutines, -- the Initiation Section and the Completion Section. If *nn* is the octal equipment type code of the device, *I.nn* and *C.nn* are the entry point names of the two sections and *DVRnn* is the driver name.

### Initiation Section

The I/O control module (\$EX18) calls the initiation section directly when an I/O transfer is initiated. Locations EQT1 through EQT17 of the base page communication area contain the addresses of the appropriate EQT entry. CHAN in base page contains the number of the DMA channel assigned to the device, if needed. This section is entered by a jump subroutine to the entry point, *I.nn*. On entry, the A-register contains the select code (channel number) of the device (bits 0 through 5 of EQT entry word 3). The driver returns to \$EX18 by an indirect jump through *I.nn*.

Before transferring to *I.nn*, DOS-M places the request parameters from the user program's EXEC call into words 7 through 13 of the EQT entry. Word 9, CONWD, is modified to contain the request code in bits 0 through 5 in place of the logical unit. See the EQT entry diagram and Section III, *READ/WRITE EXEC CALL*, for details of the parameters.

Once initiated, the driver can use words 10 through 14 of the EQT entry in any way, but words, 1, 2, 3, 7, 8, 9, 15, 16 and 17 must not be altered. The driver updates the status field in word 4, if appropriate, but the rest of word 4 must not be altered.

## INPUT/OUTPUT

### FUNCTIONS OF THE INITIATION SECTION

The initiation section of the driver operates with the interrupt system disabled. The initiation section is responsible for those functions (as flow-charted in Figure 5-1):

1. Rejects the request and proceeds to step 5 if:
  - the device is inoperable, or
  - the request code, or other or the parameters is illegal.
2. Configures all I/O instructions in the driver to include the select code of the device (or DMA channel). (Does not apply to DVR05 and 2870 DVR31.)
3. Initializes DMA, if appropriate.

*NOTE: The Initiation Section must save the DMA channel number (found in CHAN) in the EQT entry, since it is not set on entry to the Continuation Section.*

4. Initializes software flags and activates the device. All variable information pertinent to the transmission must be saved in the EQT entry because the driver may be called for another device before the first operation is complete.
5. Returns to \$EX18 with the A-register set to indicate initiation or rejection and the cause of the reject:

If  $A = \emptyset$ , then the operation was initiated.

If  $A \neq \emptyset$ , then the operation was rejected with A set as:

- 1 - read or write illegal for device,
- 2 - control request illegal or undefined,
- 3 - equipment malfunction or not ready,
- 4 - immediate completion (for control requests).
- 6 - driver cannot handle a control request so the system is instructed to wait.

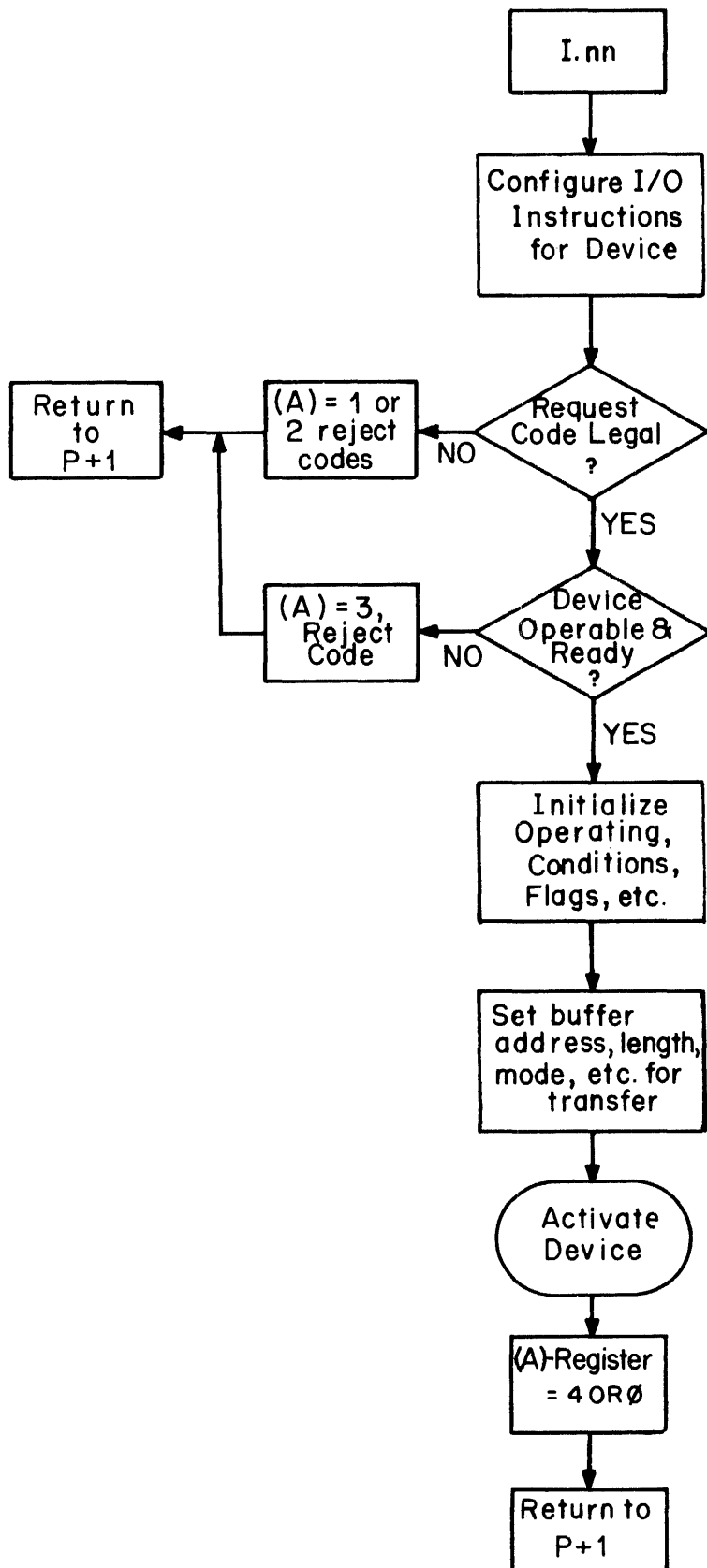


Figure 5-1. I/O Driver Initiation Section

## INPUT/OUTPUT

### Completion Section

DOS-M calls the completion section of the driver whenever an interrupt is recognized on a device associated with the driver. Before calling the driver, \$CIC sets the EQT entry addresses in base page, sets the interrupt source code (select code) in the A-register, and clears the I/O interface or DMA flag. The interrupt system is disabled. The calling sequence for the completion section is:

<u>Location</u>	<u>Action</u>
	Set A-register equal to interrupt source code
(P)	JSB C.nn
(P+1)	Completion return from C.nn
(P+2)	Continuation return from C.nn

The point of return from C.nn to \$CIC indicates whether the transfer is continuing or has been completed (in which case, end-of-operation status is returned also).

The completion section of the driver is responsible for the functions below (as flow-charted in Figure 5-2):

1. The driver configures all I/O instructions in the Completion Section to reference the interrupting device, and then proceeds to step 2.
2. If both DMA and device completion interrupts are expected and the device interrupt is significant, the DMA interrupt is ignored by returning to \$CIC in a continuation return.
3. Performs the input or output of the next data item if the device is driven under program control. If the transfer is not completed, the driver proceeds to step 6.

## INPUT/OUTPUT

4. If the driver detects a transmission error, it can re-initiate the transfer and attempt a retransmission. A counter for the number of retry attempts can be kept in the Equipment Table. The return to \$CIC must be (P+2) as in step 6.
5. At the end of a successful transfer or after completing the retry procedure, the following information must be set before returning to \$CIC at (P+1):
  - a. Set the actual or simulated device status into bits 0 through 7 of EQT word 4.
  - b. Set the number of transmitted words or characters (depending on which the user requested) to the B-register.
  - c. Set the A-register to indicate successful or unsuccessful completion.  
  
0 = successful completion,  
1 = device malfunction or not ready,  
2 = end-of-tape (information),  
3 = transmission parity error.
6. Clears the device and DMA control on end-of-operation, or sets the device and DMA for the next transfer or retry.  
Returns to \$CIC at:  
  
(P+1) - completion, with the A- and B-registers set as in step 5.  
  
(P+2) - continuation; the registers are not significant.

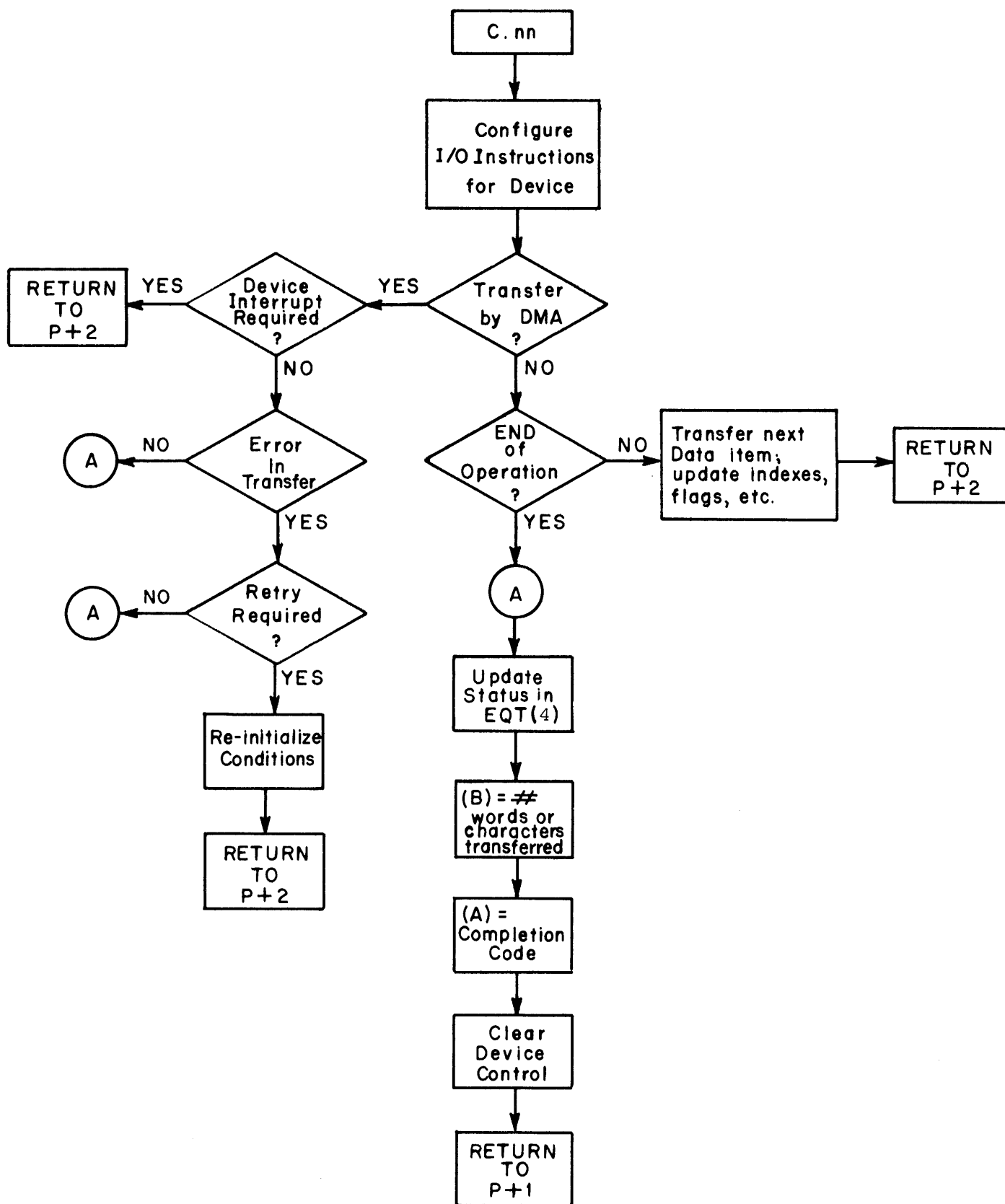


Figure 5-2. I/O Driver Completion Section

## INPUT/OUTPUT

### LINE PRINTER FORMATTING

When a user program makes a READ/WRITE EXEC call to the line printer (HP 2778A or HP 2778A-01), the line printer driver DVRL2 interprets the first character in the line as a carriage control character and prints it as a space.\* The control characters have the following meanings:

<u>Character</u>	<u>Meaning</u>
blank	Single space (print on every line),
Ø	Double space (print on every other line),
1	Eject page,
*	Suppress space (overprint next line),
others	Single space.

Each printed line is followed by an automatic single space unless suppressed by the control character asterisk (\*). Double spacing requires an additional single space prior to printing the next line. If the last line of a page is printed and the following line contains a "1", then a completely blank page occurs.

When a user program makes an EXEC call for I/O CONTROL with the function bits in the CONWD (or the ICONWD) set to  $Ø11_8$  (see Section III), the optional parameter PARAM (or IPRAM) word defines the format action to be performed by the Line Printer:

<u>Parameter Word</u> (Dec)	<u>Meaning</u>
<Ø	Page Eject;
Ø	Suppress space only the next print operation only;
1 to 55	Space 1 to 55 lines, ignoring page boundaries;
56 to 63	Use carriage control channel equal to the word - 55;
64	Set automatic page eject mode;
65	Clear automatic page eject mode.

\*DVRL2 checks for certain program names (ALGOL, FTN, ASMB, LOADR, JOBPR); for these programs it prints the first character of each line and generates a single space.

## INPUT/OUTPUT

### CARRIAGE CONTROL CHANNELS

If the parameter word is 56 to 63, the printer spaces using the standard carriage control channels, which have the following meanings:

Channel 1	Single space with automatic page eject.
Channel 2	Skip to next even line with automatic page eject.
Channel 3	Skip to next triple line with automatic page eject.
Channel 4	Skip to next 1/2 page boundary.
Channel 5	Skip to next 1/4 page boundary.
Channel 6	Skip to next 1/6 page boundary.
Channel 7	Skip to bottom of the page.
Channel 8	Skip to top of next page.

### AUTOMATIC PAGE EJECT

During non-automatic page eject mode, if the parameter word is equal to 56, then it is interpreted as equal to 1. Automatic page eject mode applies only to single space operations.

### MAGNETIC TAPE USAGE

Input/output transfers to and from a HP 3030 magnetic tape unit can be programmed using the standard READ/WRITE EXEC call. (See Section III.) When specifying the data buffer length, the programmer must know that a buffer length of zero (0) causes the driver to take no action on a write or an ASCII read. Only the amount of data that fits within the buffer is transmitted to the user on read. A zero (0) buffer length on binary read causes a forward skip one record.



## INPUT/OUTPUT

In the I/O STATUS EXEC call, bits 7-0 of the second status word contain the status of the magnetic tape unit. The bits have the following meaning when they are set (i.e., equal to one):

<u>BIT</u>	<u>MEANING</u>
------------	----------------

7	End-of-file record encountered while reading, forward spacing, or backward spacing.
6	Start-of-tape marker sensed.
5	End-of-tape marker sensed.
4	Timing error on last read/write operation.
3	I/O request rejected by magnetic tape unit.
2	No write enable ring, or the tape unit is rewinding.
1	Parity error on last read/write operation.
0	Tape unit busy, or in local mode.

The status bits are stored in the EQT entry; they are updated everytime the driver is called. A dynamic status request is processed as soon as the magnetic tape EQT entry is available (availability bits equal to 00), and returns the actual status of the device (obtained from the driver) to the calling program in the A-register and to the EQT entry.

Buffers of any length are allowed for the 7970. Buffers of less than 6 words for the 3030 are padded out to six words.

1. For binary writes they are padded with binary zeros.
2. For ASCII they are padded with ASCII Blanks.

The maximum buffer length is 16,384 words.

## INPUT/OUTPUT

### ERROR RECOVERY PROCEDURES

On a read parity error, the driver rereads the record three times before setting the parity error status bit and returning to the calling program. The final read attempt is transmitted to the program buffer.

On a write parity error, the driver continues to retry the write until one of these two conditions occurs:

- a) The record is successfully written, or
- b) The end-of-tape is encountered.

On a write without the write enable ring, the magnetic tape unit is made unavailable (magnetic tape not ready). DOS-M prints a message:

I/O ERR NR EQT#n

and waits for the operator to correct the unit and enter :GO.

At the end-of-tape there are only two legal forward motion requests:

- a) Write end-of-file, or
- b) Read record.

All other forward motion requests (write, forward space) cause the unit to be made unavailable. In addition, only one of the legal motion requests may be made after an end-of-tape. Backward motion requests clear the end-of-tape status.

# SECTION VI

## EXTENDED FILE MANAGEMENT PACKAGE

The Extended File Management Package (EFMP) extends the file handling capabilities of DOS-M by allowing the user to create and use files with different record lengths, security codes, and other conveniences. EFMP consists of a series of additional EXEC modules and a utility program; it maintains a file structure that operates within, and in addition to, the standard DOS-M file structure.

### ENVIRONMENT

EFMP functions in the DOS-M environment, but requires a computer with at least 16K memory. It is implemented through a set of EXEC modules which are incorporated into DOS-M at system generation time; the EXEC modules are invoked using the familiar EXEC call mechanism.

### FUNCTIONS AND STRUCTURE

The EFMP modules themselves allow any program executing in the user area to Create/Destroy, Open/Close, Read/Write, Reset, Repack, Copy, and Post files on the moving-head disc. Also, EFMP makes available detailed status information on all files and packs known to it. EFMP may be accessed conversationally from the keyboard by using UTIL, a utility program that executes in the user area.

### DOS-M Files vs. EFMP Files

DOS-M maintains files that are referenced by five-character names and relative sector numbers. The user can access these files in either a keyboard mode (via directives) or in a programming mode (via EXEC calls). In keyboard mode, the user creates a file with the :STORE directive and operates

## EXTENDED FILE MANAGEMENT PACKAGE

on that file with directives such as :EDIT, :DUMP, etc. In programming mode, the DOS-M files are accessed by EXEC calls such as FILE READ/WRITE and SEARCH FILE NAME.

In addition to the file structure, DOS-M maintains a subchannel (or user disc) identification scheme. User discs are first formatted either during system generation or by a special function of the system generator. These functions format the hardware tracks and set up information such as the Label Presence Code and System Proprietary Code. After a disc pack is formatted, the :INITIALIZE directive is used to set up labels (six-character codes), change labels, and purge old discs.

EFMP operates within this file structure of DOS-M to set up and maintain additional - but distinctly different - files. Selected discs within DOS-M are turned over to EFMP exclusively. The user must identify them with a pack number of the form PNxxx, where xxx is a decimal integer. The procedure for doing this is described under *SET UP*. Within a pack, EFMP creates files of its own that are not known to DOS-M. They are identified by a fixed length name, contain a grouping of specified length records, and have a security code. Since only the DOS-M files can be created and accessed by directives, all EFMP files must be used through the EFMP EXEC calls or the UTIL program. EFMP files are limited in size only by the requirement that they fit within one subchannel or pack. To avoid confusion, all references to files within this section will mean EFMP files, not DOS-M files, unless specifically stated otherwise.

### EFMP Buffers and Tables

To provide maximum flexibility in core size and speed of file accessing, EFMP allows the user to define (at execution time) the size and location of the tables and buffers required in core by EFMP. Two areas are defined by the user and provided in his program space:

1. Opened File Table
2. Temporary Record Buffers

## EXTENDED FILE MANAGEMENT PACKAGE

The Opened File Table contains all information necessary for EFMP to identify and access files belonging to the user. The minimum size of the Opened File Table is one sector (128 words) and allows approximately seven files to be opened concurrently.

EFMP uses the Temporary Record Buffers as an intermediate storage area between the disc and the user's record buffer. The user defines the number of Temporary Record Buffers and the size of each. There must be at least one buffer and it must be at least two sectors (256 words) long. Particular files and buffers can be linked to increase the access speed of files. The effect of varying the number and size of these buffers cannot be predicted exactly and must be determined empirically by trial and error.

*NOTE:* Since these tables and buffers exist in the user area and are not protected, extreme caution must be taken not to modify them in any way.

### Logical Read vs. Physical Read

A logical read occurs each time the user requests a record from a file. At that time EFMP checks the appropriate Temporary Record Buffer to determine if the requested record is already in core. If in core, the record is transferred to the user's record buffer without actually physically reading the disc. If the record is not present in core, the necessary disc transfers are performed (physical reads--and writes, if necessary) to bring the record into core. If the Temporary Record Buffer is larger than the record size, several records are brought into core at once.

### Logical Write vs. Physical Write

A logical write occurs each time a user requests that a record be written to a file. At that time, EFMP determines if that record is present in the Temporary Record Buffer; if it is, EFMP simply transfers the data in the user's record buffer to the Temporary Record Buffer and flags it as "must be

## EXTENDED FILE MANAGEMENT PACKAGE

written." Each succeeding read or write is treated in the same manner until a logical record transfer occurs for which the record is not in core, or until the last record in the Temporary Record Buffer is logically written. In these cases, the EFMP must physically write (post) the records in the Temporary Record Buffer (i.e., post them) on the disc.

If the record is not present in core on a write request, EFMP locates the record on the disc and transfers it physically into the Temporary Record Buffer. The data to be written is then transferred from the user buffer to the Temporary Record buffer and flagged as "must be written." The read before write is necessary because records do not necessarily fall on sector boundaries in the disc. If a CLOSE or POST request occurs, all buffers flagged are written to the disc.

### Update-Writes vs. Append-Writes

The purpose of an update-write is to change the contents of an existing record; the purpose of append-write is to add new records onto the end of a file. EFMP writes a record as an update-write whenever the record specified exists in a previously accessed section of a file.

EFMP writes a record as an append-write whenever the record specified is beyond the previously accessed section of a file. In this case, EFMP automatically inserts zeros into all records (if any) between the highest record previously written and the new record.

### SET UP

There are several prerequisites for EFMP. First, the EFMP EXEC modules must be included in DOS-M when the system is generated. Second, when DOS-M is running, the user must prepare EFMP disc packs from formatted DOS-M packs or cartridges.

## EXTENDED FILE MANAGEMENT PACKAGE

The mechanism for creating EFMP packs is as follows:

- a. Insert a formatted pack into the disc drive.
- b. Make the subchannel of this pack the User Disc using the :UD directive.
- c. Label the pack (if unlabeled) using the :IN directive.
- d. Set up a DOS-M file which uses the entire pack (i.e., perform a :STORE,B directive.)

The directive format for this function is:

:STORE,B,PNxxx,sectors

where xxx is a unique decimal number,

PNxxx is the unique pack number, and

sectors is the number of sectors available on the

pack = 199 \* (# sectors/track);

(4776 on a fully utilized HP 2870 or 22885

on a fully utilized HP 2883).

*NOTE: EFMP changes the file from Type B to Type A during processing.*

### EFMP EXEC CALLS

The method of communication between a user program and EFMP is through the standard DOS-M EXEC call format. One DOS-M request code--24-- is reserved for EFMP requests. This, combined with an EFMP function number, determines what action EFMP is requested to take.

Only the Assembly Language calling sequences are given for these EXEC calls. The methods for converting these calling sequences to FORTRAN or ALGOL are described in Section III.

## DEFINE

### Purpose

To define, before any other EFMP calls are made, the number of 16-bit words within the user program to be used by the EFMP for its internal buffers and tables.

### Assembly Language

JSB	EXEC	
DEF	*+9	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Number
DEF	OPNTB	Opened-File Table Address
DEF	OPNSZ	Opened-File Table Size
DEF	TRBUF	Temp. Record Buffer Address
DEF	NOTRB	No. of Temp. Record Buffers
DEF	TRBSZ	Temp. Record Buffer Size
DEF	ERRNO	Error Number
return		Continue Execution
:		
RCODE	DEF 24	
EFMPF	DEC 1	
OPNTB	BSS <i>n</i>	(Opened-File Table. <i>n</i> is the size.)
OPNSZ	DEC <i>n</i>	Size of Opened-File Table (in 16-bit words). See Comment 1.
TRBUF	BSS <i>M</i>	Beginning of Temp. Record Buffers. See Comment 2.
NOTRB	DEC <i>p</i>	No. of Temp. Record Buffers. See Comment 2.
TRBSZ	DEC <i>q</i>	Size of each Temp. Record Buffer (in sectors).
ERRNO	BSS 1	Return point for error codes. See <i>GENERAL ERRORS</i> .



## EXTENDED FILE MANAGEMENT PACKAGE

### Comments

1. The size of the Opened-File Table ( $n$ ) can be calculated by this formula:

$$n = 3 * (\text{NOTRB}) + 16 * (\text{Max. No. of Files to be OPENed})$$

The minimum size of this table is one sector (128 words). This allows approximately seven files to be OPENed concurrently.

2. There must be at least one temporary record buffer and it must be at least two sectors long (256 words). There may, however, be more buffers and they may be more than two sectors in size. All of the space for these buffers must be allocated starting at the location TRBUF. Increasing the number of buffers allows disc efficiency to be increased by assigning a buffer exclusively to one file. Increasing the size of each buffer increases the speed of disc accessing by allowing more than one sector to be transferred per disc access.

The total size of the Temp. Record Buffers ( $m$ ) can be calculated by the following formula:

$$m = \text{NOTRB} * \text{TRBSZ} * 128 \text{ (The minimum value for TRBSZ is 2.)}$$

3. All the tables and buffers are fixed by DEFINE until the end of a program, or another DEFINE. Each time a DEFINE occurs, all information contained in tables and buffers is lost, all pointers are reset, and EFMP assumes a fresh start. At the end of each program, DOS-M calls EFMP to perform a POST on any records flagged as "must be written."

## EXTENDED FILE MANAGEMENT PACKAGE

### CREATE

#### Purpose

To set up a directory on disc with all of the information necessary to create a file that can be accessed at a later time.

#### Assembly Language

JSB	EXEC	
DEF	*+9	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Number
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	FLGTH	File Length (in records)
DEF	RLGTH	Record Length (in words)
DEF	SCODE	Security Code and User Status
DEF	ERRNO	Error Number
return		Continue Execution
:		
RCODE	DEC	24
EFMPF	DEC	2
FNAME	ASC	3,xxxxx
		xxxxx is the name to be applied to the file. (First two characters cannot be zero or 177400 <sub>8</sub> .)
PAKNO	DEC	<i>p</i>
		<i>p</i> is the pack number. See Comments.
FLGTH	DEC	<i>q</i>
		<i>q</i> is the number of records in the file; (1 ≤ <i>q</i> ≤ 32,767)
RLGTH	DEC	<i>r</i>
		<i>r</i> is the number of words in a record; (1 ≤ <i>r</i> ≤ 32,767) and <i>r</i> must be less than or equal to 1/2 the size of the Temp. Record Buffer.

## EXTENDED FILE MANAGEMENT PACKAGE

### CREATE EXEC CALL (cont.)

SCODE	OCT	s	s is any 16-bit combination to be
(SCODE			checked by EFMP during OPEN and DESTROY.
+1)	OCT	t	t is any 16-bit combination of status
			information desired by the user (referred
			to as USTAT elsewhere).
ERRNO	BSS	1	Return point for error codes. See
			<i>GENERAL ERRORS.</i>

#### Comments

If PAKNO is a number between 1 and 999 it indicates the EFMP pack on which the file is to be created. When EFMP creates a file, it reserves the necessary area on the disc after the last previous file generated. No attempt is made to search for an area between files. If PAKNO is equal to -1, the file is to be created on any pack that is available.

If PAKNO equals zero, the file is placed on the work area of the disc and no area will be reserved in the EFMP packs. When such a temporary file is created, the only directory information that is maintained is in the Opened-File Table. A disc-based directory is not maintained. Also, since the directory information is established in core during the CREATE function, the OPEN function is not required. The only reason for using an OPEN call for a temporary file is to assign it to a specific Temporary Record Buffer or to change the starting record number to a value other than 1. If no OPEN call is given, the first Temporary Record Buffer is used.

When the work area is used for temporary files, EFMP reserves this whole area and identifies it as PN0000. In order to keep PN0000 from using the entire work area, the user must enter a STORE,B,PN0000 directive for the system disc with the desired number of sectors. When EFMP has terminated, the user should PURGE the file PN0000 from the work area

**DESTROY**Purpose

To eliminate the directory information for a particular file from core and the disc. The user must specify the correct security code for the file. The disc area is repacked only for temporary files. To repack the EFMP subchannels, use the REPACK EFMP call.

Assembly Language

```

      JSB  EXEC
      DEF  *+7      Return Address
      DEF  RCODE    Request Code
      DEF  EFMPF    EFMP Function Code
      DEF  FNAME    File Name
      DEF  PAKNO    Pack Number
      DEF  SCODE    Security Code
      DEF  ERRNO    Error Number
      return      Continue Execution
      .
RCODE  DEC  24
EFMPF  DEC  3
FNAME  ASC  3,xxxx
PAKNO  DEC  n      If n = 0, then FNAME refers to a temporary
                  file. If n ≥ 1 and n ≤ 999, then FNAME
                  is to be located on this pack number.
                  If n = -1, then EFMP searches all of its
                  packs until it finds a file that matches
                  FNAME.

SCODE  OCT  s      s is the security code for the file
                  established by the CREATE EFMP Call.
                  Security code ignored on temporary
                  files.

ERRNO  BSS  1      Return point for error codes. See
                  GENERAL ERRORS.

```

## EXTENDED FILE MANAGEMENT PACKAGE

### OPEN

#### Purpose

To make a previously CREATED file accessible by extracting the necessary file information from the disc directories and placing it in core. The number of files that can be OPENED at any one time is limited by the size of the Opened File Table (see DEFINE).

#### Assembly Language

JSB	EXEC	
DEF	*+9	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	RCDNO	Record Number
DEF	SCODE	Security Code
DEF	BUFNO	Buffer Number
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	4
FNAME	ASC	3,xxxxx
PAKNO	DEC	n

If  $n = \emptyset$ , the file is a temporary file on the work area. If  $n$  is between 1 and 999, EFMP looks for FNAME on the appropriate pack. If  $n = -1$ , EFMP searches all available packs for the requested file.

## EXTENDED FILE MANAGEMENT PACKAGE

### OPEN EXEC CALL (cont.)

RCDNO	DEC	<i>r</i>	If $r = \emptyset$ , EFMP sets the next record to be accessed (for sequential READS or WRITES) to the highest record previously accessed + 1. Otherwise, <i>r</i> can be any number between 1 and the maximum record number contained in the file. This allows sequential access to be initialized at any record.
SCODE	OCT	<i>s</i>	<i>s</i> is the security code established by the CREATE call. It must match.
BUFNO	DEC	<i>b</i>	<i>b</i> must be a number between 1 and the maximum number of Temp. Record Buffers available. For any other number, EFMP uses 1.
ERRNO	BSS	1	Return point for error codes. See <i>GENERAL ERRORS</i> .

## CLOSE

Purpose

To remove information about a particular file from the core-based Opened-File Table. This allows an additional file to be OPENED. Also, CLOSE updates the user status information (USTAT) and the highest record accessed on the disc.

Assembly Language

JSB	EXEC	
DEF	*+6	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Number
DEF	FNAME	File Name
DEF	USTAT	User Status
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	5
FNAME	ASC	3,xxxxx
USTAT	OCT	u
		User status information (any 16-bit combination) to be written into the disc directory for the file.
ERRNO	BSS	1
		Return point for error codes. See <i>GENERAL ERRORS</i> .

Comments

If a CLOSE is requested for a temporary file, the directory information in the Opened-File Table is deleted and the work area is automatically repacked. If a file has been COPIED to the work area, the user status (USTAT) and highest record assessed are not updated on the original copy of the file.

## READ

### Purpose

To retrieve a specified record (random access) or the next record (sequential access) from a file that has previously been OPENED and WRITTEN.

### Assembly Language

JSB EXEC		
DEF *+7		Return Address
DEF RCODE		Request Code
DEF EFMPF		EFMP Function Code
DEF FNAME		File Name
DEF RCDNO		Record Number
DEF BUFR		Buffer for Data
DEF ERRNO		Error Number
return		Continue Execution
:		
RCODE DEC	24	
EFMPF DEC	6	
FNAME ASC	3,xxxxx	
RCDNO DEC	<i>n</i>	<i>n</i> is a record number between 1 and 32,767. For sequential access and backspacing, see Comment.
BUFR BSS	<i>m</i>	<i>m</i> is the length of the buffer in words. It must be at least the record length.
ERRNO BSS	1	Return point for error codes. See <i>GENERAL ERRORS</i> .



## EXTENDED FILE MANAGEMENT PACKAGE

### Comments

If RCDNO = Ø, a sequential READ or WRITE is implied. This feature provides the program with the next record available relative to the last READ or WRITE performed (or OPEN operation). If RCDNO is a negative number, it specifies a backspace, relative to the current record (last record accessed plus 1), before the READ or WRITE. If an attempt is made to backspace the record number indicator to a value less than one, the EFMP issues an error and terminates the READ or WRITE. Unless needed, care should be taken so as not to backspace the record number indicator beyond the range of records held in the Temporary Record Buffer at that time, since this will initiate a posting operation and a physical disc access.

**WRITE**Purpose

To write into a specified record (random access) or into the next record (sequential access) of a file that has previously been OPENED.

Assembly Language

JSB	EXEC	
DEF	*+7	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Number
DEF	FNAME	File Name
DEF	RCDNO	Record Number
DEF	BUFFR	Buffer for Data
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	8
FNAME	ASC	3,xxxxx
RCDNO	DEC	<i>n</i>
		Same as for the READ EXEC CALL.
BUFFR	BSS	<i>m</i>
		Same as for READ.
ERRNO	BSS	1
		Return point for error codes.
		See <i>GENERAL ERRORS</i> .

## RESET

Purpose

To reset the highest record accessed pointer for a file to a lower value. The information beyond the pointer is lost. The file must be OPEN before it can be RESET. (PAKNO below provides an additional check.)

Assembly Language

JSB	EXEC	
DEF	*+7	
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	RCDNO	Record Number
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	9
FNAME	ASC	3,xxxxx
PAKNO	DEC	<i>n</i>
		If $n = \emptyset$ , EFMP searches the work area to find the desired file name. If $n$ is a number between 1 and 999, EFMP searches pack number PN <i>n</i> to find the desired file name. If $n = -1$ , EFMP searches all packs.
RCDNO	DEC	<i>m</i>
		<i>m</i> is a number between $\emptyset$ and 32,767 to which the highest record accessed pointer will be set. <i>m</i> must be lower than the current value.
ERRNO	BSS	1
		Return point for error codes. See <i>GENERAL ERRORS</i> .

## STATUS

### Purpose

To allow the user program access to various types of status information relative to EFMP. Several separate status functions (identified by unique Status Function Numbers) are provided; all have basically the same form of calling sequence, but they vary in the parameters used.

### Assembly Language

JSB	EXEC	
DEF	*+9	Return Address
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	FSTAT	Status Function Number
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	DUMMY	Not Used
DEF	STATB	Status Buffer
DEF	ERRNO	Error Number
return		Continue Execution
:		

*NOTE: Above is the general format for Status EFMP calls. The use and meaning of each parameter in the calling sequence varies from status call to status call. The parameters for each call are given separately below. Common to all status functions are:*

RCODE	DEC	24
EFMPF	DEC	10
DUMMY	BSS	1

## STATUS

### STATUS FUNCTION NUMBER 1

#### Purpose

To provide the user with all information, except the security code, contained in the directory for a file.

#### Parameters

FSTAT DEC 1

FNAME ASC 3,xxxxx

PAKNO DEC *m*

If  $m = \emptyset$ , EFMP searches the work area for the requested file. If  $m$  is between 1 and 999, EFMP searches the pack of that number. For  $m = -1$ , EFMP searches all available packs for the requested file.

STATB BSS 10

The pack number is returned in the first word if PAKNO = -1. The remaining nine words will receive the directory status information in the same format as the directory itself. (See *EFMP File Disc Directory*.)

ERRNO BSS 1

Return point for error code.  
See *GENERAL ERRORS*.

## STATUS

### STATUS FUNCTION NUMBER 2

#### Purpose

To determine if a file is OPEN.

#### Parameters

FSTAT DEC 2

FNAME ASC 3,xxxxx

PAKNO OCT 0

Not used.

STATB BSS 2

The first word returns the pack number if the file is OPEN. The second word returns a value of 0 if the file is OPEN or 1 if the file is not open.

ERRNO BSS 1

Return point for error codes.

See *GENERAL ERRORS*.

**STATUS****STATUS FUNCTION NUMBER 3**Purpose

To check the security code of a file.

Parameters

FSTAT	DEC	3	
FNAME	ASC	3,xxxxx	
PAKNO	DEC	<i>m</i>	Same as Function Number 1.
STATB	BSS	3	The first word returns the pack number if appropriate. The second word is used by the user program to give the security code to be checked. The third word returns 0 if the code checks or 1 if it does not check.
ERRNO	BSS	1	Return point for error codes. See <i>GENERAL ERRORS</i> .

**STATUS****STATUS FUNCTION NUMBER 4**Purpose

To determine the number of available full sectors left between the highest record accessed in a file and the end of the file.

Parameters

FSTAT DEC 4

FNAME ASC 3,xxxxx

PAKNO DEC *m*

Same as Function Number 1.

STATB BSS 2

The first word returns the pack number if appropriate. The second word returns the number of sectors available.

ERRNO BSS 1

Return point for error codes.

See *GENERAL ERRORS*.



## STATUS

### STATUS FUNCTION NUMBER 5

#### Purpose

To determine the number of available sectors left between the last file in a pack and the end of the pack.

#### Parameters

FSTAT	DEC	5	
FNAME	OCT	Ø	Not used.
PAKNO	DEC	<i>m</i>	Same as Function Number 1, but cannot equal -1.
STATB	BSS	2	The first word must be present, but is not used. The second word returns the number of sectors available.
ERRNO	BSS	1	Return point for error codes. See <i>GENERAL ERRORS</i> .

**STATUS****STATUS FUNCTION NUMBER 6**Purpose

To obtain the name of the *n*th file on a pack where *n* is an integer between 1 and the maximum number of files on a pack.

Parameters

FSTAT	DEC	6	
FNAME	BSS	3	Return point for file name or all zeroes if no file is present.
PAKNO	DEC	<i>m</i>	<i>m</i> is a number between 1 and 999.
STATB	DEC	<i>n</i>	<i>n</i> indicates the <i>n</i> th file.
ERRNO	BSS	1	Return point for error codes.

See *GENERAL ERRORS*.

## STATUS

### STATUS FUNCTION NUMBER 7

Purpose

To request all pack numbers currently available to EFMP.

Parameters

FSTAT	DEC	7	
FNAME	OCT	Ø	Not used.
PAKNO	OCT	Ø	Not used.
STATB	BSS	7	Return point for pack numbers. They are presented in ascending order of subchannel number. The list is terminated by a zero.
ERRNO	BSS	1	Return point for error codes. See <i>GENERAL ERRORS</i> .

## Example

Information returned in buffer.

STATB	<table><tr><td>200</td></tr><tr><td>10</td></tr><tr><td>762</td></tr><tr><td>0</td></tr></table>	200	10	762	0	(terminates list)
200						
10						
762						
0						

STATB	<table><tr><td>5</td></tr><tr><td>10</td></tr><tr><td>2</td></tr><tr><td>900</td></tr><tr><td>213</td></tr><tr><td>22</td></tr><tr><td>6</td></tr></table>	5	10	2	900	213	22	6	(list terminates by being complete)
5									
10									
2									
900									
213									
22									
6									

Order of pack numbers does not imply specific subchannel numbers.

**REPACK (PURGE)**Purpose

To repack the existing files on a pack(s), removing empty spaces left when files have been destroyed.

Assembly Language

JSB	EXEC	
DEF	*+5	
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	PAKNO	Pack Number
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	11
PAKNO	DEC	<i>n</i>
		For <i>n</i> between 1 and 999, only the specified pack is repacked. For <i>n</i> = -1, all the packs available to EFMP are repacked.
ERRNO	BSS	1
		Return point for error codes.
		See <i>GENERAL ERRORS</i> .

CAUTION: If the EFMP disc directory contains a large number of files and the sizes of the Temporary Record Buffers are small, repacking may require considerable time. Therefore, REPACK should be performed when sufficient time is available. Under no circumstances should an ABORT be performed during a REPACK.

## COPY

Purpose

To transfer a copy of an opened file and its directory from an EFMP pack to the work area of DOS-M, from a pack to another pack, or from the work area to a pack.

Assembly Language

JSB	EXEC	
DEF	*+6	
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEF	24
EFMPF	DEC	12
FNAME	ASC	3,xxxxx
PAKNO	DEC	n

See Comment 1.

If  $n = \emptyset$ , EFMP copies the file onto the work area. If  $n$  is between 1 and 999, EFMP copies the file onto the specified pack. If  $n$  is between -1 and -999, EFMP copies the file from the work area to a pack specified by the 10's complement of  $n$ . See Comment 2.

ERRNO BSS 1

Return point for error codes.

See *GENERAL ERRORS*.

## EXTENDED FILE MANAGEMENT PACKAGE

### Comments

1. Remember that a file must be OPENED before it can be COPIED. This is necessary to determine from which pack to copy the file. When a file has been copied to the work area, all READS and WRITES referencing that file use the work area version until the file is CLOSED. (Files copied *from* the work area to a pack continue to use the work area version for READS and WRITES.) Temporary copies of files do not have security codes. Therefore, files copied from the work area to a pack have a security code of Ø. When a file is copied from pack to pack, the original security code is retained. See CLOSE for further notes on Work Area files.
2. If there is already a file with the same name in the destination pack directory, an error code is returned and the COPY is aborted. In this case, the user can first DESTROY the name in the destination pack, and then perform the COPY again.
3. When COPYING from a pack to a pack not on the drive (and only a single removable pack is available), EFMP automatically requests that the user continually swap packs until the entire file has been COPIED. EFMP prints out a message and halts the computer with 1Ø2Ø76 in the T-Register:

INSERT DESTINATION [SOURCE] PACK AND PRESS RUN.

After the user inserts the appropriate pack and presses RUN, a check is made to determine if the proper pack has been entered. If EFMP cannot find the correct pack, the message is repeated. To allow the user an orderly exit in case the correct pack is not available, the following question is asked after each question:

ENTER C OR T

where C means to continue COPYING, and

T means to terminate the COPY and return to the program.

4. Care must be taken to insert the original pack (if it has been removed during the COPY function) into its original subchannel.

## CHANGE FILE NAME

Purpose

To change a file name. (File need not be OPEN.)

Assembly Language

JSB	EXEC	
DEF	*+7	
DEF	RCODE	Request Code
DEF	EFMPF	EFMP Function Code
DEF	FNAME	File Name
DEF	PAKNO	Pack Number
DEF	SCODE	Security Code
DEF	ERRNO	Error Number
	return	Continue Execution
	:	
RCODE	DEC	24
EFMPF	DEC	13
FNAME	ASC	3,xxxxx
	ASC	3,zzzzz
PAKNO	DEC	<i>n</i>
		<i>n</i> = Ø, indicates that the file is on the work area. If <i>n</i> is between 1 and 999, <i>n</i> indicates the pack containing the file. If <i>n</i> = -1, EFMP searches all available packs for the current file name.
SCODE	OCT	<i>m</i>
ERRNO	BSS	1
		Security code. See <i>CREATE</i> .
		Return point for error codes. See <i>GENERAL ERRORS</i> .

## POST

### Purpose

To physically write on the disc all buffers that have been flagged as "must be written" in the Temporary Record Buffer. (That is, convert all outstanding logical WRITE's into physical WRITE's.)

### Assembly Language

```
        JSB EXEC
        DEF  *+4
        DEF  RCODE          Request Code
        DEF  EFMPF          EFMP Function Code
        DEF  ERRNO          Error Number
        return              Continue Execution
        :
RCODE   DEC  24
EFMPF   DEC  14
ERRNO   BSS  1              Return point for error codes.
                               See GENERAL ERRORS.
```

### Comments

The POST operation updates the highest record accessed pointer in the disc directories, but not the user status word (USTAT).



## EXTENDED FILE MANAGEMENT PACKAGE

### UTIL PROGRAM--CONVERSATIONAL USE OF EFMP

UTIL is a program that allows access to most of the EFMP functions through the keyboard; it accepts commands or directives from the operator and converts these into EFMP calling sequences. When EFMP has processed the call, UTIL reports back a successful operation or an error given by EFMP.

#### Functions

The following EFMP functions are provided by UTIL:

1. CREATE
2. DESTROY
3. OPEN
4. CLOSE
5. STATUS (all functions)
6. REPACK
7. COPY
8. CHANGE FILE NAME
9. POST
10. RESET

When initiated, UTIL makes a DEFINE call to establish a Temporary Record Buffer of four sectors and an Opened File Table of one sector. In addition, UTIL provides one other function--BRIEF--that allows the operator to increase or decrease the amount of disc storage reserved for a file.

*NOTE: UTIL requires the FORTRAN IV version of the Formatter program to operate properly.*

## :PROG, UTIL

### Purpose

To initiate execution of the UTIL program.

### Format

:PROG, UTIL,*n*

where *n* = 0 to print a list of commands or  
*n* ≠ 0 to skip printing the list.

List of commands message:

```

/CRE,FNAME,PAKNO,FLGTH,RLGTH,SCODE,USTAT
/DES,FNAME,PAKNO,SCODE
/OPE,FNAME,PAKNO,RCDNO,SCODE
/CLO,FNAME,USTAT
/RES,FNAME,PAKNO,RCDNO
/STA,DF,FNAME,PAKNO
/STA,FO,FNAME
/STA,SC,FNAME,PAKNO,SCODE
/STA,LR,FNAME,PAKNO
/STA,LF,PAKNO
/STA,NF,PAKNO,STATB
/STA,AP
/REP,PAKNO
/COP,FNAME,PAKNO
/CHA,FNAM1,FNAM2,PAKNO,SCODE
/POS
/BRI,FNAME,SCODE
/END
    
```

(All parameters are decimal.)

## EXTENDED FILE MANAGEMENT PACKAGE

UTIL begins by printing a message to indicate that it is ready for a directive:

UTIL READY

After it processes the directive, UTIL prints out the results of the operation (where appropriate) or any error codes that may have been returned by EFMP. (See *GENERAL ERRORS*.) When it is ready for another directive, UTIL prints UTIL READY. If an incorrect directive is entered, UTIL prints

ILLEGAL OPERATION

UTIL READY

UTIL is terminated by typing in the command /END.

UTIL prints any error messages on the system terminal; normal output is printed on the list device.

## CREATE COMMAND

To create a new file (i.e., to invoke the CREATE function of EFMP).

```

/CRE, FNAME, PAKNO, FLGTH, RLGTH, SCODE, USTAT

```

See *CREATE EFMP CALL* for explanation of parameters.

```

/CRE, DATA, 42, 20, 126, 3901, 1
      ↑      ↑      ↑      ↑      ↑      ↑
      file   pack   file   record  security  user
      name   number length  length  code     status

```

```

FILE CØ      CREATED
THE FILE IS ON PACK#      12Ø
THE FILE LENGTH IS      8 RECORDS
THE RECORD LENGTH IS      8 WORDS
THE SECURITY CODE IS      Ø
THE USER STATUS WORD IS      Ø

```

## DESTROY COMMAND

### Purpose

To destroy a file by eliminating its directory entry (i.e., to invoke the DESTROY EFMP function).

### Format

/DES, *FNAME*, *PAKNO*, *SCODE*

See *DESTROY EFMP CALL* for explanation of parameters.

### Example

/DES, DATA, 42, 3901

```
graph TD
    A[DATA] --> B[file name]
    C[42] --> D[pack number]
    E[3901] --> F[security code]
```

### Example print-out:

FILE CØ DESTROYED

## OPEN COMMAND

See *OPEN EFMP CALL* for explanation of parameters.

6-36

## CLOSE COMMAND

### Purpose

To CLOSE a previously OPENED file (i.e., to invoke the CLOSE function of EFMP).

### Format

/CLO, *FNAME*, *USTAT*

See *CLOSE EFMP CALL* for explanation of parameters.

### Example

/CLO, DATA, 2  
          ↑  
          file  
          name  
                  ↑  
                  user  
                  status

### Example print-out:

FILE LOB7Ø   CLOSED  
THE USER STATUS WORD IS   Ø

## RESET COMMAND

### Purpose

To reset the highest record number accessed for a file (i.e., to invoke the RESET function of EFMP).

### Format

*/RES, FNAME, PAKNO, RCDNO*

See *RESET EFMP CALL* for explanation of the parameters.

### Example

*/RES, DATA, 42, 10*

file  
name

pack  
number

record  
number

### Example print-out:

FILE LOB70 RESET  
THE FILE IS ON PACK# 120  
THE RECORD # IS 0



## STATUS-1 COMMAND

### Purpose

To print out directory information about a file (i.e., to invoke STATUS function number 1 of EFMP).

### Format

/STA, DF, *FNAME*, *PAKNO*, *RCDNO*

See *STATUS EFMP CALL, STATUS FUNCTION NUMBER 1* for explanation of the parameters and results.

### Example

/STA, DF, DATA, 42

↑  
file  
name

↑  
pack  
number

### Example print-out:

FILE LOB7Ø STATUS  
THE FILE IS ON PACK# 12Ø  
STARTING TRACK # IS 6  
STARTING SECTOR # IS 9  
THE FILE LENGTH IS 12 RECORDS  
THE RECORD LENGTH IS 128 WORDS  
THE USER STATUS WORD IS Ø  
HIGHEST RECORD # ACCESSED IS Ø

## STATUS-2 COMMAND

### Purpose

To determine if a file is OPEN (i.e., to invoke STATUS function number 2 of EFMP).

### Format

/STA, FO, *FNAME*

See *STATUS FUNCTION NUMBER 2* for explanation of the parameters and results.

### Example

/STA, FO, DATA  
          ↑  
          file  
          name

### Example print-out:

FILE LOB7Ø STATUS  
FILE IS [NOT] OPEN

## STATUS-3 COMMAND

### Purpose

To check the security code of a file (i.e., to invoke STATUS function number 3 of EFMP).

### Format

/STA, SC, *FNAME*, *PAKNO*, *SCODE*

See *STATUS FUNCTION NUMBER 3* for explanation of parameters and results.

### Example

/STA, SC, DATA, 42, 3904

↑  
file  
name

↑  
pack  
number

↑  
security code  
to be checked

### Example print-out:

FILE LOB70 STATUS  
THE FILE IS ON PACK# 120  
THE SECURITY CODE IS 0  
CODE CHECKS [DOES NOT CHECK]

## STATUS-4 COMMAND

### Purpose

To determine the number of available full sectors left between the highest record accessed in a file and the end of the file (i.e., to invoke STATUS function number 4 of EFMP).

### Format

/STA, LR, *FNAME*, *PAKNO*

See *STATUS FUNCTION NUMBER 4* for explanation of parameters and results.

### Example

/STA, LR, DATA, 42  
          ↑      ↑  
         file  pack  
         name  number

### Example print-out:

FILE LOB7Ø STATUS  
THE FILE IS ON PACK# 12Ø  
# OF AVAILABLE SECTORS IS 12

## STATUS-5 COMMAND

### Purpose

To determine the number of available sectors left between the last file in a pack and the end of the pack (i.e., to invoke STATUS function number 5 of EFMP).

### Format

/STA, LF, *PAKNO*

See *STATUS FUNCTION NUMBER 5* for explanation of parameters and results.

### Example

/STA, LF, 42  
          ↑  
          pack  
          number

### Example print-out:

FOR PACK#     120  
# OF AVAILABLE SECTORS IS  4610

## STATUS-6 COMMAND

### Purpose

To obtain the name of the *n*th file on a pack where *n* is an integer between 1 and the maximum number of files on a pack (i.e., to invoke STATUS function number 6 of EFMP).

### Format

/STA, NF, *PAKNO*, *STATB*

See *STATUS FUNCTION NUMBER 6* for explanation of parameters and results.

### Example

/STA, NF, 42, 9  
           ↑      ↑  
           pack  
           number  
                   ↑  
                   file  
                   number

### Example print-out:

```
FILE  LOB70  STATUS
THE FILE IS ON PACK#    120
FILE #      1 IN THE DIRECTORY
```

## STATUS-7 COMMAND

### Purpose

To request all available pack numbers (i.e., to invoke STATUS function number 7 of EFMP).

### Format

/STA, AP

Example print-out:

PACK #    120 IS AVAILABLE

## REPACK COMMAND

### Purpose

To repack existing packs (i.e., to invoke the REPACK EXEC CALL function of EFMP).

### Format

/REP, *PAKNO*

See *REPACK (or purge) EFMP CALL* for explanation of parameters.

### Example

```
/REP, 42  (repacks pack 42)
/REP, -1  (repacks all packs)
```

### Example print-out:

```
ALL PACKS AVAILABLE REPACKED
      or
PACK # 120 REPACKED
```



## COPY COMMAND

### Purpose

To copy a file (i.e., to invoke the COPY function of EFMP).

### Format

/COP, *FNAME*, *PAKNO*

See *COPY EFMP CALL* for explanation of parameters and messages.

### Example

/COP, DATA, 45  
          ↑  
          file  
              ↑  
              destination  
              pack

### Example print-out:

FILE LOB70 COPIED  
THE FILE IS TEMPORARY IN WORK AREA  
  
FILE LOB70 COPIED  
THE FILE IS ON PACK# 120

## CHANGE COMMAND

### Purpose

To change the name of a file (i.e., to invoke the CHANGE FILE NAME function of EFMP).

### Format

*/CHA, FNAM1, FNAM2, PAKNO, SCODE*

*FNAM1 is the current file name; FNAM2 is the new file name.*

See *CHANGE FILE NAME EFMP CALL* for explanation of other parameters.

### Example

*/CHA, DATA, STUFF, 42, 3901*

↑  
old  
name

↑  
new  
name

### Example print-out:

```
FILE LOB7Ø OLD FILE
FILE XXXXX NEW FILE
THE FILE IS ON PACK# 12Ø
THE SECURITY CODE IS Ø
```

## POST COMMAND

### Purpose

To post files (i.e., to invoke the POST function of EFMP).

### Format

/POS

Example print-out:

ALL FILES POSTED

## BRIEF COMMAND

### Purpose

To increase or decrease the amount of disc storage reserved for a file.

### Format

/BRI, *FNAME*, *SCODE*

*FNAME* is the name of the file, and  
*SCODE* is the security code of the file.

BRIEF first prints the status of the file:

AVAILABLE RECS. = *m*      RECORDS USED = *r*  
NEW RECORD COUNT?

The user types in either:

/E to terminate the command and prepare UTIL for more commands,  
or  
*n* to change the available record count to *n*.

BRIEF stores the contents of *FNAME* on the Work Area, destroys the current file, purges the pack, and CREATES and OPENS a new file. The contents of *FNAME* are transferred from the Work Area to the new file and BRIEF prints out a message:

AVAILABLE RECS. = *n*      RECORDS USED = *r*

BRIEF terminates.

### Comment

BRIEF creates and uses a temporary file named "~~~~~" (all blanks).

## END COMMAND

### Purpose

To terminate the operation of the UTIL program.

### Format

/END

## EXTENDED FILE MANAGEMENT PACKAGE

### GENERAL ERRORS

These error numbers are returned to the user program (in ERRNO) by the EFMP.

<u>Error No.</u>	<u>Description</u>
0	No errors.
1	Invalid EFMP function number.
2	Duplicate File Name.
3	File Name not in directory.
4	File too long for this pack.
5	Invalid record length.
6	Pack number not available (or Name not in directory if a search was made on all available pack directories).
7	Invalid Security Code.
8	A Temporary File must be "OPENED" with a CREATE function. An OPEN function can only change the Temporary Record Buffer number or the starting record number for a Temporary File.
9	Buffer area specified in Exec call is not valid.
10	Invalid Record Number.
11	File not open.
12	DEFINE not previously executed.
13	Backspaced beyond "start-of-file."
14	No pack space available.
15	Invalid pack number.
16	During a pack search, a pack was found where the entire space was not allocated to PNxxx.
17	Work area space not sufficient
18	No "Open" table space available.
19	Invalid Temporary Record Buffer Number.
20	Invalid number of Executive call parameters.
21	End-of-File.
22	COPY terminated.
23	Invalid argument(s).
24	Maximum number of files exceeded.
25	File already OPEN.
26	Record size larger than one-half of a TRB.

The error numbers are also returned in the A-Register.

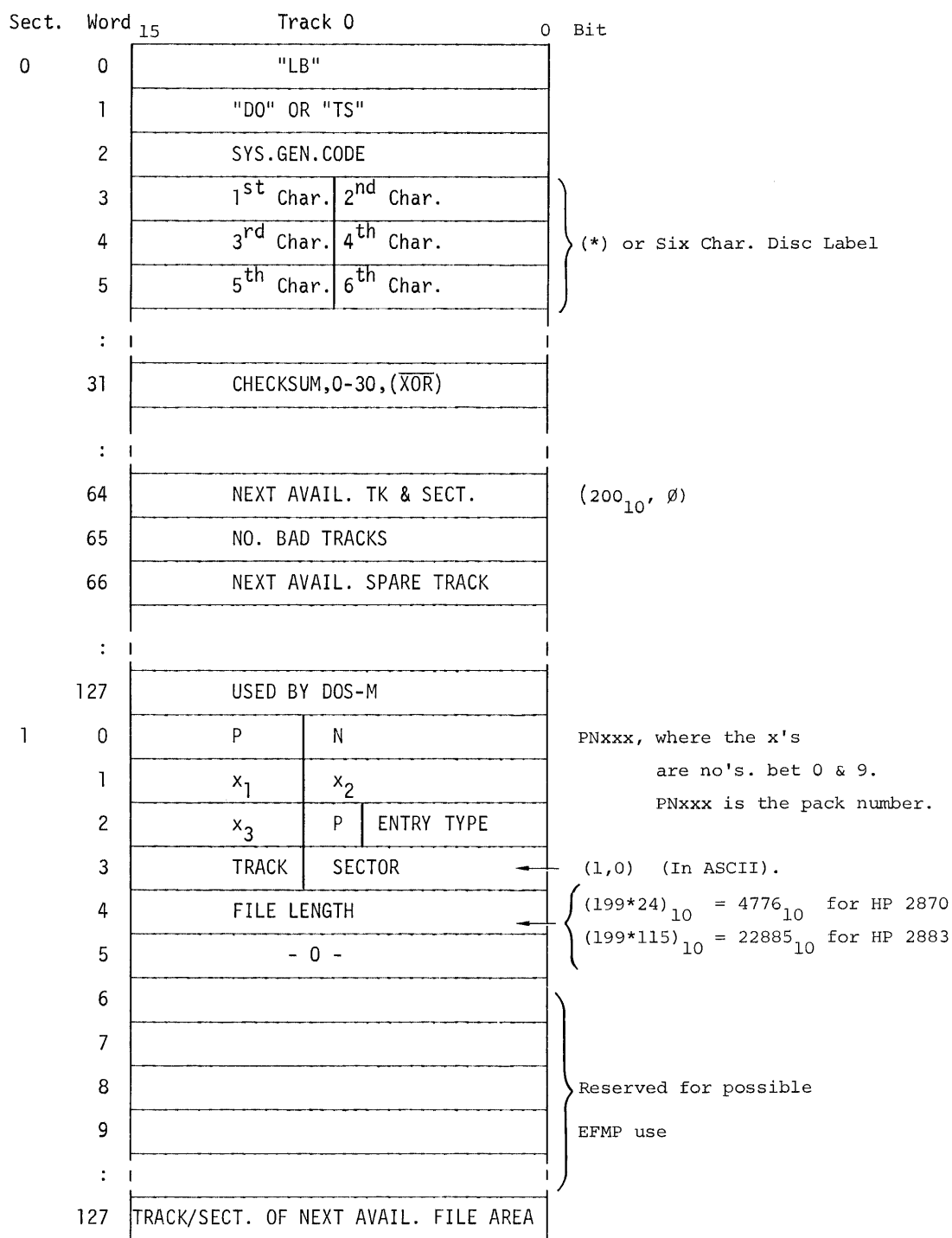
# EXTENDED FILE MANAGEMENT PACKAGE

## EFMP FILE DISC DIRECTORY

Word	15	87	Contents	0	Bit
0			1 <sup>st</sup> Char.	2 <sup>nd</sup> Char.	
1			3 <sup>rd</sup> Char.	4 <sup>th</sup> Char.	
2			5 <sup>th</sup> Char.	Not Used	
3			Starting Trk. No.	Starting Sect. No.	
4			File Length (In Records)		
5			Record Length (In Words)		
6			Security Code		
7			User Supplied Status		
8			Highest Record Number Accessed		

Figure 6-1. EFMP File Disc Directory Format

# EXTENDED FILE MANAGEMENT PACKAGE

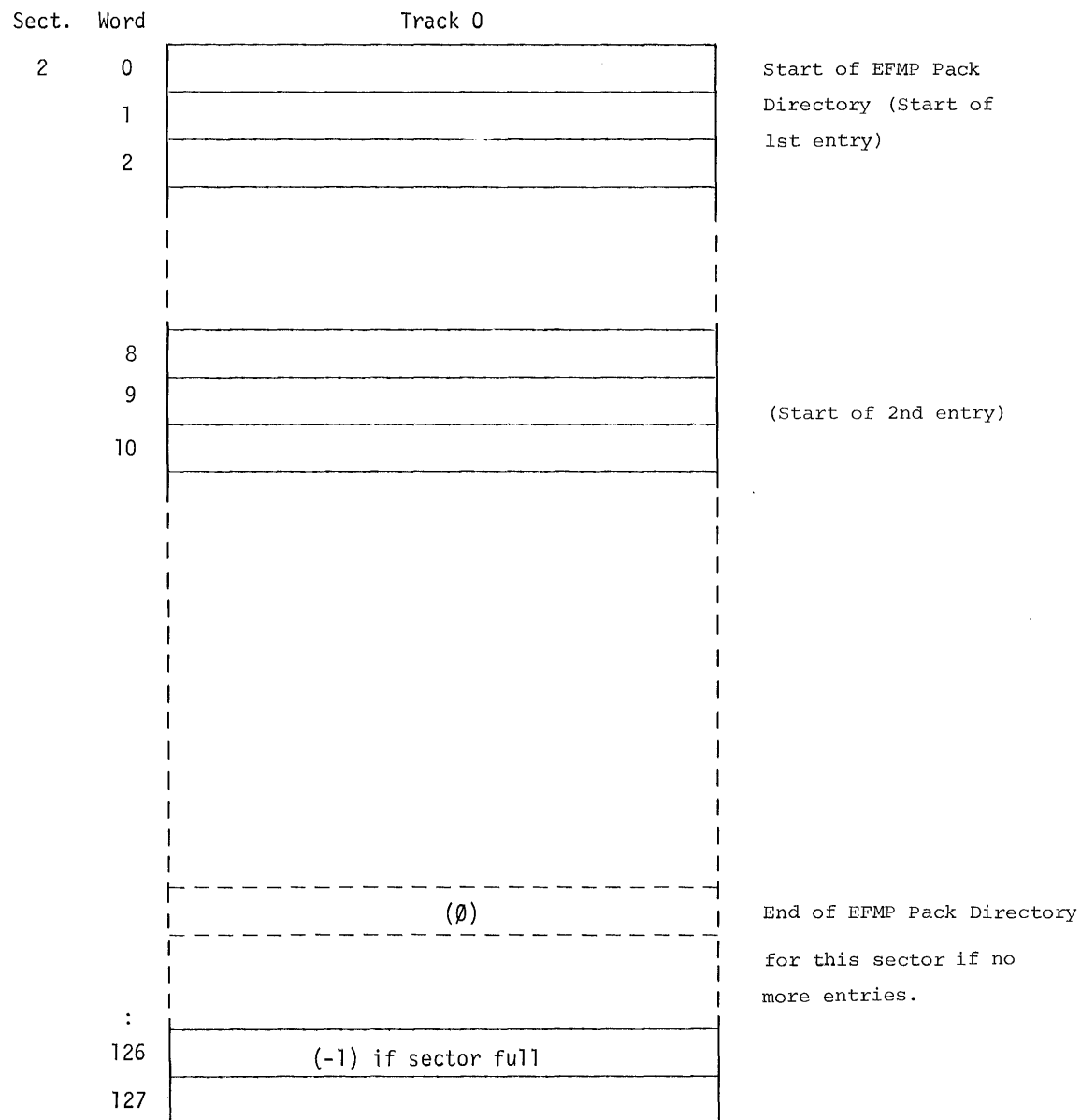


Words 0 through 127 on sector 0 and words 0 through 5 on sector 1 represent the DOS-M format. See Appendix A for further details.

Figure 6-2. EFMP Disc Pack Layout (Part 1 of 2)



# EXTENDED FILE MANAGEMENT PACKAGE



All succeeding sectors follow the same format as sector 2.

End-of-directory is indicated by a zero in the word following the last directory entry.

End-of-sector is indicated by a (-1) in the word following the last entry.

Figure 6-2. EFMP Disc Pack Layout (Part 2 of 2)

# APPENDIX A

## TABLES

Appendix A contains several useful tables and figures.

### DOS-M BASE PAGE CONSTANTS

<u>LOCATION</u>	<u>TYPE</u>	<u>VALUE</u>
40	DEC	-64
41	DEC	-10
42	DEC	-9
43	DEC	-8
44	DEC	-7
45	DEC	-6
46	DEC	-5
47	DEC	-4
50	DEC	-3
51	DEC	-2
52	DEC	-1
53	DEC	0
54	DEC	1
55	DEC	2
56	DEC	3
57	DEC	4
60	DEC	5
61	DEC	6
62	DEC	7
63	DEC	8
64	DEC	9
65	DEC	10
66	DEC	17
67	DEC	64
70	OCT	17
71	OCT	37
72	OCT	77

# TABLES

<u>LOCATION</u>	<u>TYPE</u>	<u>VALUE</u>
73	OCT	177
74	OCT	377
75	OCT	177400
76	OCT	3777
77	OCT	177700

## DOS-M BASE PAGE SYSTEM COMMUNICATION AREA

<u>LOCATION</u>	<u>NAME</u>	<u>CONTENTS</u>
100	UMLWA	Last word address of user available memory
101	JBINS	Start track/sector of job binary area
102	JBINC	Current Track/sector of job binary area
103	TBG	Time base generator I/O channel address
104-5	CLOCK	Current system clock time
106-7	CLEX	Execution clock time
110	CXMX	Maximum allowable execution time
111	BATCH	Logical unit # of batch input device
112	SYSTY	Logical unit # of system teletype
113	DUMPS	Abort/Post Mortem dump flag
114	SYSDR	System directory track/sector
115	SYSBF	System buffer track/sector
116	SECTR	Number of sectors/disc track
117	EQTAB	First word address of Equipment Table
120	EQT#	Number of Equipment entries
121	LUTAB	First word address of Logical Unit table
122	LUT#	Number of Logical Unit entries
123	JBUF	Job input buffer address
124	JFILS	Source file starting track/sector
125	JFILC	Source file current track/sector
126-40	RONBF	User area file name information (11 words)
141-53	EXPG	Directory entry for current program (11 words)

# TABLES

<u>LOCATION</u>	<u>NAME</u>	<u>CONTENTS</u>
154	DISCO	Disc I/O channel/last track on disc
155	SYSSC	System subchannel
156	SCCNT	Number of subchannels on system minus 1
157	UDNTS	Next user disc track/sector
160	SYNTS	Next system disc track/sector
161	CUDSC	Current user disc subchannel
162	CRFLG	Current disc request flag: 0 for system, not 0 for user
163	CUDLA	Current user disc last access
164	SDLA	System disc last access
165	CUMID	Computer identification
166-70	DBUFR	System disc triplet parameter buffer
171-73	UBUFR	User disc triplet parameter buffer
174	TSONE	Last track/sector referenced +1
175	GUDSC	Default user disc subchannel
176	SYSCD	System generation code
177	JFLSC	Source file subchannel
200	DISCL	User label track/sector
201	INTAB	First word address of interrupt table
202	INT#	Number of interrupt entries
203	EQT1	EQT1-EQT17 are addresses of current Equipment Table entry
204	EQT2	
205	EQT3	
206	EQT4	
:	:	
223	EQT17	
224	RQCNT	Number of request parameters.
225	RQRTN	Current request return address
226	RQP1	RQP1-RQP8 are addresses of current request parameters
:	:	
235	RQP8	

# TABLES

<u>LOCATION</u>	<u>NAME</u>	<u>CONTENTS</u>
236	NABRT	Illegal request code abort/no abort option
237	XA	A register contents at time of interrupt
240	XB	B register contents at time of interrupt
241	XEO	E and O register contents at time of interrupt
242	XSUSP	Point of suspension at time of interrupt
243	EXLOC	Address of Exec module doublet table
244	EX#	Number of Exec module doublet table entries
245	EXMOD	Exec module # currently in Exec module overlay area
246-47	EXMAN	Exec module low and high main core addresses
250-51	EXBAS	Exec module low and high base page core addresses
252	IODMN	First word address of I/O driver module main area
253	IODBS	First word address of I/O driver module base page area
254	UMFWA	First word address of user main area
255	UBFWA	First word address of user base page area
256	UBLWA	Last word address of user base page area
257	CHAN	Current DMA channel number
260	OPATN	Operator/keyboard attention flag
261	OPFLG	Operator communication flag
262	SWAP	Job processor resident flag
263-65	JOBPM	Job processor disc address/number of words in main
265	JOBPB	Job processor base page number of words
266	EJOB	End job flag
267	RTRK	Real time simulation track number
270	\$BUF	System input/output buffer (128 words)
470	\$GOPT	Point of suspension continuation address
471	\$IDCD	Input request code check
472	\$MDBF	Exec module data buffer

# TABLES

<u>LOCATION</u>	<u>NAME</u>	<u>CONTENTS</u>
474	TEMP	System temporary (7 word buffer)
503	TEMP0	System temporary
504	TEMP1	System temporary
505	UTMP0	User temporary
506	UTMP1	User temporary
507	UTMP2	User temporary
510	TEMP5	System temporary
511	MSECT	Negative number of sectors/track
512	VADR	Address of instruction causing memory protect violation
513	IODMD	Current resident I/O driver module flag
514	RCODE	Current request code value
515	SXA	Operator attention restore A register value
516	SXB	Operator attention restore B register value
517	SXEO	Operator attention restore E and O register value
520	SXSUS	Operator attention return address
521	EFMP	Extended file management package flag
522	DSCLB	Disc track/sector of relocatable library
523	DSCL#	Number of relocatable library routines
524	LSTCH	Last disc referenced
525	TRAC#	User file table validity flag
526	XFLG	Entry address for disc not ready
527	SSFLG	System search flag
530	CHARC	Batch Input Character Count
531	TYEQT	System TTY EQT4 Address

} Available to user

# TABLES

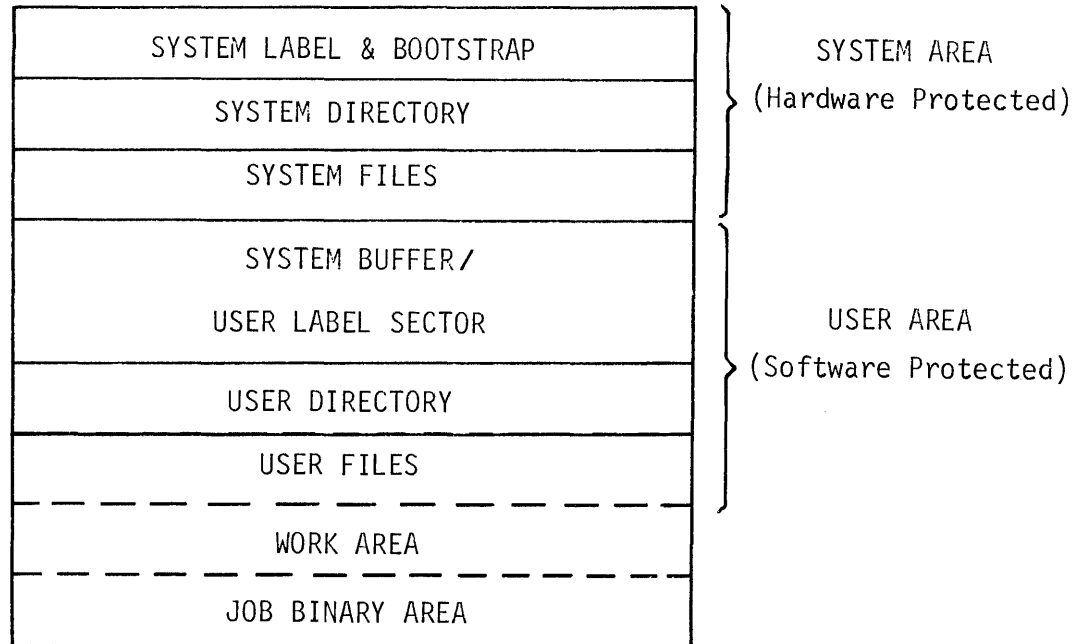


Figure A-1. General Disc Layout

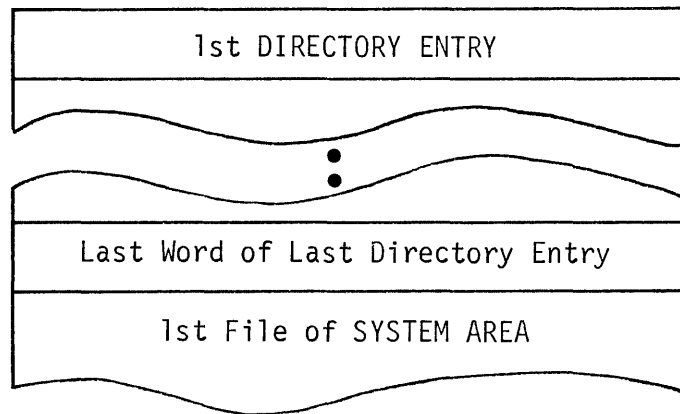


Figure A-2. System Directory Format

## TABLES

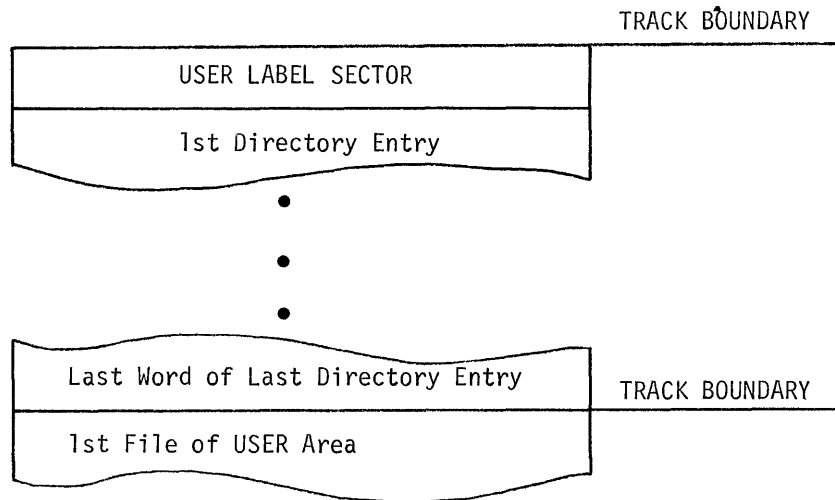


Figure A-3. User Directory Format

Word 1	F	N
Word 2	A	M
Word 3	E	P    Entry Type
Word 4	Track	Sector
Word 5	File Length (in sectors)	
Word 6	FWA Program	
Word 7	LWA Program	
Word 8	FWA Base Page Linkage Area	
Word 9	LWA Base Page Linkage Area	
Word 10	Program Entry Point	
Word 11	FWA of LIB routine section	

For System or Loader  
Generated Binary Pro-  
grams Only

The 1st five characters (in Words 1 through 3) contain the File Name

The lower character in Word 3 contains the Type Code and 'P' bit, as shown below.

Figure A-4. Directory Entry Format



## TABLES

<u>TYPE</u>	<u>FILE</u>
Ø	System Resident
1	Disc Resident Executive Supervisor Module
2	Reserved for System
3	User Program, Main
4	Disc Resident Device Driver
5	User Program, Segment
6,7	Library
10 <sub>8</sub>	Relocatable Binary
11 <sub>8</sub>	ASCII Source Statements
12 <sub>8</sub>	Binary Data
13 <sub>8</sub>	ASCII Data

### 'P' Bit

Ø = No Action

1 = Purge this entry at the end of the JOB. This bit is set by the LOADER and cleared by a :STORE,P[,*file-name*] request

The last directory entry in each sector is followed by a word containing '-1'.  
The last entry in the directory is followed by a word containing zero.

## TABLES

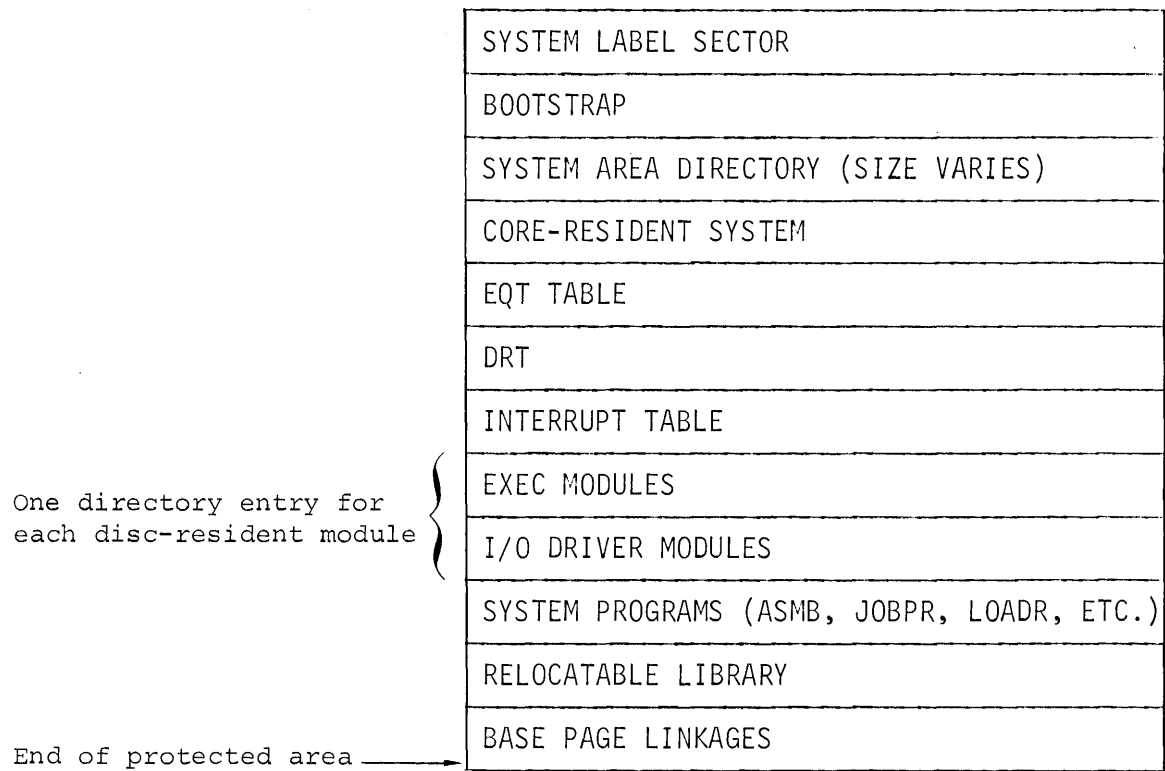


Figure A-5. Disc Layout

## DISC LABELS

Sector 0 of track 0 of each disc is used for label information. In addition, if the user area is on the system disc, a label also occurs in Sector 0 of the first track after the system area.

The contents of the label include:

- Word 0:       Label presence code (ASCII "LB").
- Word 1:       System Proprietary Code:
  - 1. "DO" for DOS-M
  - 2. "TS" for Time-Shared Basic
- Word 2:       System generation code assigned at system generation time. The code can be any 4 decimal digits.
- Words 3-5:    A six-character disc label. If the first character equals \* the disc is unlabeled. This label can only be set using :IN (for user areas) or by DSGEN (set to "SYSTEM" for system discs).
- Word 31:      Checksum of words 0-30.

The first 64 words are reserved for label information. Word 65 contains the next available track and sector. Words 66 and 67 contain the number of bad tracks and the next available spare track.

# APPENDIX B

## TYPICAL JOB DECKS

### ASSEMBLE A PROGRAM AND STORE IN FILE

```
:JOB,ASMBS
:PROG,ASMB,5,6,4,56,99
ASMB,B,L
    NAM TEST,3
    :
    END ENTER
:STORE,R,AFILE
:JOB,NEXT JOB
```

} Source Program

### LOAD AND EXECUTE A RELOCATABLE FILE

```
:JOB,LOADE
:PROG,LOADR,2
AFILE,/E
:STORE,P,TEST
:RUN,TEST
10
23
:
:
51
:JOB,NEXT JOB
```

} Data

STORE, EDIT, COMPILE, LOAD AND RUN A PROGRAM

:JOB,EVERY

:STORE,S,SOURC,5

FTN,B,L

PROGRAM ZOOM

DIM I(10)

:

ENDS\$

Source Program

::

:LIST,S,6,SOURC

:EDIT,SOURC,5

/I,2

:

/E

Edit List

:JFILE,SOURC

:PROG,FTN,2,6,4,56,99

:PROG,LOADR

:RUN,ZOOM

123.62

:

00001

Data for first run

:RUN,ZOOM

321.5

:

0.56

Data for second run

:JOB,NEXT JOB

# APPENDIX C

## RELATION TO OTHER SOFTWARE

The Hewlett-Packard general-purpose computers can handle other HP software when the Moving-Head Disc Operating System is inactive. Every computer is shipped with the software and documentation appropriate to the system configuration.

Prepare Tape System can be used to store the relocatable modules of DOS-M on a magnetic tape. DSGEN can then read from this magnetic tape to generate a system.

In an attempt to make DOS-M compatible with the Real-Time Executive, DOS-M simulates the Real-Time EXEC requests as follows (See *REAL-TIME SOFTWARE*, 02116-9139):

READ/WRITE	Identical for work area of disc and I/O devices.
I/O CONTROL	Identical
I/O STATUS	Status word 2 returns transmission log instead of Real-Time Equipment Table word 5.
DISC ALLOCATION	Simulates request in work area.
DISC RELEASE	No action; tracks cannot be released.
PROGRAM COMPLETION	Identical
PROGRAM SUSPENSION	Identical
PROGRAM SEGMENT LOAD	Identical
PROGRAM SCHEDULE	Treated as program main load.
CURRENT TIME	Word 5 set to Ø, other words identical.
EXECUTION TIME (TIMER)	Not accepted See N option of RUN request.

# APPENDIX D

## SUMMARY OF DIRECTIVES

<u>DIRECTIVE</u>	<u>DESCRIPTION</u>
:ABORT	Terminate the current job.
:ADUMP[, <i>FWA</i> [, <i>LWA</i> ][, <i>B</i> ], <i>L</i> ]	Dump a program if it aborts
:BATCH, <i>logical unit</i>	Switch from keyboard to batch mode, or reassign batch device.
:COMMENT <i>string</i>	Print a message.
:DATE, <i>day</i> [, <i>hour</i> , <i>min</i> ]	Set the date and the time (if time-base is present).
:DD	Dump the entire current disc onto a disc on another subchannel.
:DD,X	Dump the system area only to another disc.
:DD,U[, <i>file</i> [( <i>name</i> )], <i>file</i> [( <i>name</i> )]...]	Dump all or specified files of the current user disc to another disc, optionally assigning new file names.
:DN, <i>n</i>	Declare an I/O device down.
:DUMP, <i>log.unit</i> , <i>file</i> [, <i>s</i> <sub>1</sub> [, <i>s</i> <sub>2</sub> ]]	Dump all or part of a user file to a peripheral I/O device.
:EDIT, <i>file log.unit</i> ], <i>new</i> ]	Edit a source statement file stored on disc, optionally creating a new file.
:EF[, <i>logical unit</i> ]	Write end-of-file on magnetic tape.
:EJOB	Terminate the current batch and/or job normally.
:EQ[, <i>n</i> ]	List the equipment table.
:GO[, <i>P</i> <sub>1</sub> , <i>P</i> <sub>2</sub> ... <i>P</i> <sub>5</sub> ]	Continue processing a suspended program.
:IN, <i>label</i>	Label or unlabel ("*") the current user disc.

# SUMMARY OF DIRECTIVES

<u>DIRECTIVE</u>	<u>DESCRIPTION</u>
:JFILE, <i>file</i>	Specify a source file on the disc for the assembler or compiler.
:JOB[, <i>name</i> ]	Initiate a user job.
:LIST,S, <i>log.unit</i> , <i>file</i> [, <i>m</i> ][, <i>n</i> ]]	List all or part of a source statement file.
:LIST,U, <i>log.unit</i> [, <i>file</i> <sub>1</sub> ,...]	List all or part of the user directory.
:LIST,X, <i>log.unit</i> [, <i>file</i> <sub>1</sub> ,...]	List all or part of the system directory.
:LU[, <i>n</i> <sub>1</sub> ][, <i>n</i> <sub>2</sub> ]]	Assign or list logical units.
:OFF	Abort the currently executing program or operation without terminating the job.
:PAUSE	Suspend the current job or program.
:PDUMP[, <i>FWA</i> ][, <i>LWA</i> ]][,B][, <i>L</i> ]	Dump a program after normal completion.
:PROG, <i>name</i> [, <i>P</i> <sub>1</sub> , <i>P</i> <sub>2</sub> ... <i>P</i> <sub>5</sub> ]	Turn on a system or user program.
:PURGE[, <i>file</i> <sub>1</sub> , <i>file</i> <sub>2</sub> ,...]	Delete user files.
:RUN, <i>name</i> [, <i>time</i> ][, <i>N</i> ]	Run a user program
:SA, <i>track</i> , <i>sector</i> [, <i>number</i> ]	Dump disc in ASCII to standard list device.
:SU, <i>track</i> , <i>sector</i> [, <i>number</i> ]	Dump disc in octal to standard list device.
:SS	Set up system search for file names over all subchannels.
:SS, <i>n</i> <sub>1</sub> , <i>n</i> <sub>2</sub> ...	Set up system search for file names over specified subchannels.
:SS,99	Restrict search for file names to current user disc (plus system directory for RUN & PROG).



## SUMMARY OF DIRECTIVES

<u>DIRECTIVE</u>	<u>DESCRIPTION</u>
:STORE,A, <i>file</i> , <i>sectors</i>	Reserve space for an ASCII data file.
:STORE,B, <i>file</i> , <i>sectors</i>	Reserve space for a binary data file
:STORE,P[, <i>name</i> <sub>1</sub> , <i>name</i> <sub>2</sub> ,...]	Store temporary Loader generated programs as permanent files.
:STORE,R, <i>file</i> [, <i>log.unit</i> ]	Store a relocatable file from a peripheral I/O device or from the JBIN area of disc after an assembly or compilation.
:STORE,S, <i>file</i> , <i>log.unit</i>	Store a source statement file from a peripheral I/O device.
:TRACKS	Print the disc track status of the current user disc.
:TYPE	Return to keyboard mode from batch mode.
:UD[, [ <i>label</i> ][, <i>n</i> ]]	Change the subchannel assignment for the user disc, or request label & subchannel information for a user disc.
:UP, <i>n</i>	Declare an I/O device up.

# APPENDIX E

## SUMMARY OF EXEC CALLS

Consult Section III for the complete details on each EXEC call.

For each EXEC call, this appendix includes only the parameters ( $P_1$  through  $P_n$ ) of the assembly language calling sequence.

### READ/WRITE: Transfer input or output.

RCODE	DEC	1 or 2	1 = read or 2 = write
CONWD	OCT	$c$	(See Section III for control information.)
BUFFR	BSS	$n$	( $n$ -word buffer)
BUFFL	DEC	$n$ or $-2n$	(buffer length, words (+), characters (-).)
DTRAK	DEC	$p$	(disc track; optional)
DSECT	DEC	$q$	(disc sector; optional)

### I/O CONTROL: Carry out control operations.

RCODE	DEC	3	
CONWD	OCT	$c$	(See Section III for control information.)
PARAM	DEC	$n$	(Optional parameter required by some CONWDs.)

### PROGRAM COMPLETION: Signal end of program.

RCODE	DEC	6
-------	-----	---

### PROGRAM SUSPEND: Suspend calling program.

RCODE	DEC	7
-------	-----	---

## SUMMARY OF EXEC CALLS

PROGRAM SEGMENT LOAD: Load segment of calling program.

RCODE DEC 8  
SNAME ASC 3,xxxxx (xxxxx is segment name)

TIME REQUEST: Request the 24-hour time and day.

RCODE DEC 11  
ARRAY BSS 5 (Time values; tens of milliseconds, seconds, minutes, hours, returned in that order.)

I/O STATUS: Request device status.

RCODE DEC 13  
CONWD DEC n (Logical unit number)  
STATS NOP (Status returned here)  
TLOG NOP (Transmission log returned here)

FILE READ/WRITE: Read or write a user data file.

RCODE DEC 14 or 15 (14 = read or 15 = write.)  
CONWD OCT c (See Section III for control information.)  
BUFFR BSS n (Buffer of n words.)  
BUFFL DEC n or -2n (Length of buffer in words (+) or characters (-).)  
FNAME ASC 3,xxxxx (User file name = xxxxx.)  
RSECT DEC m (Relative sector within file.)

WORK AREA STATUS: Ascertain if n contiguous work tracks are available.

RCODE DEC 16  
NTRAK DEC n (Number of consecutive tracks desired.)  
TRACK NOP (Desired first track; from LIMITS call.)  
STRAK NOP (Actual starting track, or 0 if n not available.)

## SUMMARY OF EXEC CALLS

WORK AREA LIMITS: Ascertain first and last tracks of work area.

RCODE DEC 17

FTRAK NOP (Returns first work track number here.)

LTRAK NOP (Returns last work track number here.)

SIZE NOP (Returns number of sectors per track here.)

SEARCH FILE NAMES: Ascertain if a file name exists in the directory.

RCODE DEC 18

FNAME ASC 3,xxxxx (xxxxxx is the file name.)

NSECT NOP (Number of sectors in file returned here, or  
Ø if not found.)

CHANGE USER DISC: Change the current user disc subchannel.

RCODE DEC 23

LABEL ASC 3,xxxxx (Disc label = xxxxxx or ASCII 1, \* for unlabel.)

SUBCH DEC (Ø to 7) (Subchannel number; optional parameter.)

MAIN PROGRAM LOAD: Transfer a main program into core.

RCODE DEC 1Ø

PNAME ASC 3,xxxxx (Program name)

## APPENDIX F

### ALGOL EXEC CALLS

The program below (DXFER) reads one sector from the work area and writes the information into a different location in the work area. DXFER calls EXEC through the CODE procedure EXECX compiled externally. EXECX is compiled in the program DSKIO, although that program name is irrelevant to the linkage between DXFER and EXECX.

#### MAIN PROGRAM

```
HPAL,B,L,"DXFER"
BEGIN
  INTEGER ARRAY BUFFER[1:128];
  BOOLEAN READX;
  INTEGER TRACK,SECTOR;
  FORMAT F1("SOURCE TRACK,SECTOR?"),
        F2("DESTINATION TRACK,SECTOR?");
  PROCEDURE EXECX(RD,TRK,SCTR,BFR);
    VALUE RD,TRK,SCTR;
    BOOLEAN RD;
    INTEGER TRK,SCTR,BFR;
    CODE;
  WRITE(1,F1);
  READ(1,*,TRACK,SECTOR);
  READX<-TRUE;
  EXECX(READX,TRACK,SECTOR,BUFR[1]);
  WRITE(1,F2);
  READ(1,*,TRACK,SECTOR);
  READX<-FALSE;
  EXECX(READX,TRACK,SECTOR,BUFR[1]);
END$
```

## ALGOL EXEC CALLS

### PROCEDURE

```
HPAL,P,B,L,"DSKIO"  
PROCEDURE EXECX(RD,TRK,SCTR,BFR);  
  VALUE RD,TRK,SCTR;  
  BOOLEAN RD;  
  INTEGER TRK,SCTR,BFR;  
BEGIN  
  PROCEDURE EXEC(IO,LU,BFR,BFSZ,TRK,SCTR);  
    VALUE IO,LU,BFSZ,TRK,SCTR;  
    INTEGER IO,LU,BFR,BFSZ,TRK,SCTR;  
    CODE;  
    INTEGER REQCD;  
    IF RD THEN REQCD←1 ELSE REQCD←2;  
    EXEC(REQCD,2,BFR,128,TRK,SCTR);  
END;
```

# INDEX

## A

:ABORT.....2-4  
 :ADUMP.....2-27  
 ALGOL.....4-10  
 ALGOL Control Statement.....4-13  
 ALGOL EXEC Calls.....3-2,F-1  
 ALGOL Segmentation.....4-14  
 Append-Writes.....6-4  
 Assembler.....4-15  
 Assembler Control Statement.....4-18  
 Assembly Language Calling  
   Sequences.....3-2

## B

Base Page Constants.....A-1  
 Base Page System Common Area.....A-2  
 :BATCH.....2-5  
 Batch Mode.....1-2,2-1  
 Begin Job.....2-20  
 BINRY.....3-6  
 BRIEF Command.....6-50

## C

Change File Name EXEC Call..6-29,6-48  
 Change User Disc.....2-46,3-23  
 Change User Disc EXEC Call.....3-23  
 CLOSE EFMP Call.....6-13,6-37  
 :COMMENT.....2-6  
 Completion Section.....5-11  
 Copy EFMP Call.....6-27,6-47  
 Core Layout.....1-4  
 CREATE EFMP Call.....6-8,6-34

## D

DATA Statement.....4-6  
 :DATE.....2-49  
 :DD.....2-7  
 DEBUG.....4-30  
 DEFINE EFMP Call.....6-6  
 DESTROY EFMP Call.....6-10,6-35  
 Directives.....1-2,2-1,D-1  
 Directory Entry Format.....A-7  
 Disc Labels.....A-10  
 Disc Layout.....A-6,A-9  
 DISCM.....xiv,1-1,1-3  
 Disc-To-Disc Dump.....2-7  
 DOS-M.....1-1  
 :DOWN.....2-9  
 DSGEN.....1-7  
 :DUMP.....2-10

## E

:EDIT.....2-12  
 :EF.....2-15  
 EFMP.....1-7,6-1  
 :EJOB.....2-16  
 END Command.....6-51  
 EFMP File Disc Directory.....6-53  
 End-Of-File.....2-15  
 End Of Job.....2-16  
 :EQUIPMENT.....2-17  
 Equipment Table.....5-2  
 ERROR.....4-9  
 EXEC Calls.....1-3,3-1,6-5,E-1,F-1  
 EXEC Calls In ALGOL.....3-2,F-1  
 EXEC Modules.....xiv  
 Extended File Management  
   Package.....1-7,6-1  
 EXTERNAL Statement.....4-7

## F

Features.....xi  
 File Dump.....2-10  
 File Read/Write EXEC Call.....2-7  
 Files.....1-6,3-7,6-1  
 FORTRAN.....4-2  
 FORTRAN Calling Sequence.....3-3  
 FORTRAN Control Statement.....4-4  
 Function Code.....3-10

## G

:GO.....2-50

## H

Hardware.....xii  
 HP 2870 Disc.....1-4  
 HP 2883 Disc.....1-4

## I

:IN.....2-51  
 Initialize.....2-51  
 Initiation Section.....5-8  
 Input/Output.....1-3,4-19,5-1  
 Installation.....1-7  
 Interrupt Processing.....5-7  
 Interrupt Table.....5-5  
 I/O Control EXEC Call.....3-9  
 I/O Drivers.....5-5  
 I/O Status EXEC Call.....3-11

# INDEX

## J

:JOB.....2-20  
Job Binary Area.....4-25  
Job Decks.....1-2,A-1  
JOBPR.....xiv,1-1  
:JFILE.....2-19

## K

Keyboard Mode.....1-2,2-1

## L

Line Printer Formatting.....5-14  
:LIST.....2-21  
Load-And-Go.....4-1  
Logical Read.....6-3  
Logical Units.....2-25,5-4  
Logical Write.....6-3  
:LU.....2-25

## M

Magnetic Tape Usage.....5-15  
Main Program Load EXEC Call.....3-27

## N

NAM Statement.....4-20  
No Waiting.....3-6

## O

:OFF.....2-53  
OPEN EFMP Call.....6-11,6-36  
Opened File Table.....6-2,6-7  
ORB Statement.....4-19  
Other Software.....C-1  
Overlay Segments.....4-8

## P

:PAUSE.....2-26  
:PDUMP.....2-27  
Physical Read.....6-3  
Physical Write.....6-3  
POST EXEC Call.....6-30,6-49  
:PROG.....2-30  
Programming.....4-1  
Program Completion EXEC Call.....3-16  
Program Dump.....2-27  
Program Segment Load EXEC Call.....3-19  
PROGRAM Statement.....4-5  
Program Suspend EXEC Call.....3-17  
:PURGE.....2-31

## R

READ EFMP Call.....6-14  
Read/Write EXEC Call.....3-4

## R

Real-Time Executive.....C-1  
Relocatable Libraries.....4-35  
Relocating Loader.....4-25  
REPACK (PURGE) EFMP Call...6-26,6-46  
RESET EFMP Call.....6-17,6-38  
RMPAR.....3-18  
:RUN.....2-33  
Running A System Program.....2-30  
Running A User Program.....2-33

## S

:SA.....2-34  
Search File Name EXEC Call.....3-21  
Sector Dump.....2-34  
Segmented Programs.....4-20  
:SO.....2-34  
Software.....xiii  
Specify Source File:.....2-19  
:SS.....2-36  
STATUS EFMP Call.....6-18,6-39  
Status Function Number 1...6-19,6-39  
Status Function Number 2...6-20,6-40  
Status Function Number 3...6-21,6-41  
Status Function Number 4...6-22,6-42  
Status Function Number 5...6-23,6-43  
Status Function Number 6...6-24,6-44  
Status Function Number 7...6-25,6-45  
:STORE.....2-38  
:STORE,A.....2-42  
:STORE,B.....2-42  
:STORE,P.....2-40  
:STORE,R.....2-38  
:STORE,S.....2-41  
Subchannel.....1-4  
Supervisor.....xiv,1-1  
System Area.....1-5  
System I/O.....5-6  
System Directory Format.....A-6  
System Search.....2-36

## T

Tables.....A-1  
Temporary Record Buffers.....6-2,6-7  
Time Request EXEC Call.....3-22  
:TRACKS.....2-43  
:TYPE.....2-45  
Type-A Files.....2-42  
Type-B Files.....2-42  
Type-P Files.....2-40  
Type-R Files.....2-38  
Type-S Files.....2-41



## INDEX

### U

:UD.....2-46  
:UP.....2-48  
Update-Writes.....6-4  
User Area.....1-5  
User Directory Format.....A-7  
User Program I/O.....5-6  
UTIL.....6-31  
UTIL Commands.....6-32

### W

Waiting.....3-6  
Work Area.....3-13,3-14  
Work Area Limits EXEC Call.....3-13  
Work Area Status EXEC Call.....3-16  
WRITE EFMP Call.....6-16

# SOFTWARE MANUAL CHANGES

## MOVING-HEAD DISC OPERATING SYSTEM

(HP 02116-91779)

Dated March, 1971

*Some of the items below pertain not only to the MOVING-HEAD DISC OPERATING SYSTEM manual but also to the Manual Change Sheet itself. The highest-numbered entry is the most current. Therefore, these changes should be recorded first. This ensures that earlier entries which have been modified are updated on this sheet. Earlier entries which no longer apply are superseded by later entries.*

### 10-71

Change Number	Page	Instructions
1	4-6	Change the last line on the page to read: DATA A(1),A(2),A(3)/1.0,2.0,3.0/,1(1),1(2)/2*1/

### 2-72

Change Number	Page	Instructions
2	C-1	In the list of RTE exec calls, replace the description of DISC RELEASE with:  "Sets location RTRK (real-time track simulation) in base page to zero."
3	6-52	Change description of Error 12 to:  "DEFINE not previously executed or OPEN table used in previous DEFINE has been altered. Issue a new DEFINE."



READER COMMENT SHEET  
MOVING-HEAD  
DISC OPERATING SYSTEM

02116-91779

MARCH, 1971

Hewlett-Packard welcomes your evaluation of this text.  
Any errors, suggested additions, deletions, or general comments may be made below. Use extra pages if you like.

CUT ALONG LINE

FROM

PAGE\_\_OF\_\_

NAME: \_\_\_\_\_

ADDRESS: \_\_\_\_\_

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AS SHOWN ON OTHER SIDE AND TAPE

FOLD

FOLD

**BUSINESS REPLY MAIL**

No Postage Necessary if Mailed in the United States Postage will be paid by

MANAGER, SOFTWARE PUBLICATIONS  
HEWLETT - PACKARD  
CUPERTINO DIVISION  
11000 Wolfe Road  
Cupertino, California  
95014

FIRST CLASS  
PERMIT NO. 141  
CUPERTINO  
CALIFORNIA



FOLD

FOLD

