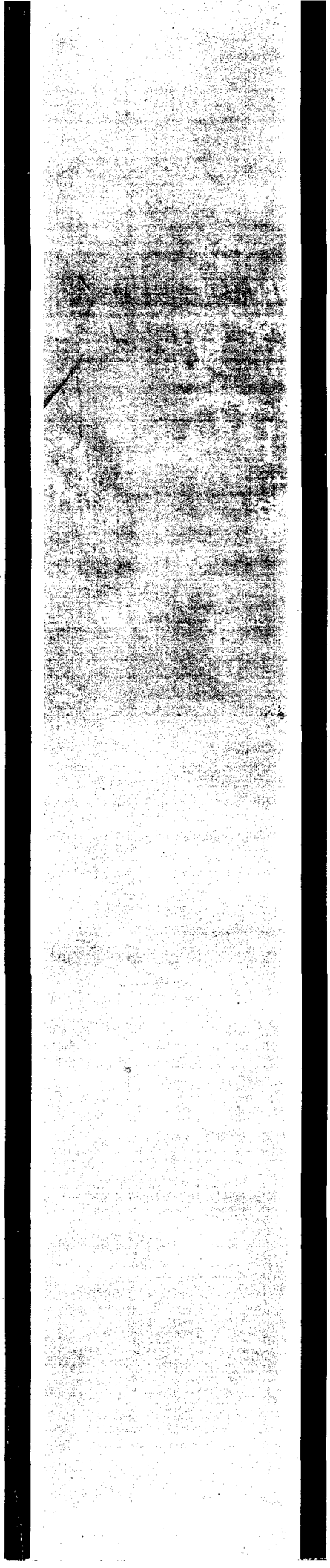

HEWLETT PACKARD

DISC OPERATING SYSTEM

DISC OPERATING SYSTEM

11000 Wolfe Road
Cupertino, California

HP 02116-91748

October 1969

Printed in the U.S.A.

PREFACE

DISC OPERATING SYSTEM is the programmer's and operator's guide to the Hewlett-Packard Disc Operating System (DOS) for the HP2116B Computer and disc or drum memory. DOS is a batch processing system that executes complete jobs without operator intervention. For a full understanding of DOS, the reader should be familiar with one of the Hewlett-Packard programming languages, as presented in the *FORTRAN* (02116-9015) and *ASSEMBLER* (02116-9014) programmer's reference manuals.

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 programming -- FORTRAN, 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. Procedures for installing and initiating the software appear in Section VI. The appendices provide tables, summaries, a complete listing of error messages, and sample job decks.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

TABLES

INTRODUCTION

In the DISC OPERATING SYSTEM (DOS) for the HP2116B Computer and disc storage unit, 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) or HP Assembly Language. A variety of files can be stored, edited, listed, dumped and used as input to programs.

FEATURES OF DOS

DOS contains the following highlights and features:

- Keyboard and batch processing modes,
- Software programming aids: FORTRAN Compiler, Assembler, Relocating Loader, Relocatable Library, Debug Routine, and Source File Editor,
- Jobs executed in a queue with no 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.

DOS HARDWARE CONFIGURATION

The minimum hardware requirements for the DOS system are:

- An HP2116B Computer with 8K memory, and:
  12578A Direct Memory Access
  12579A Extended Arithmetic Unit

12591A Memory Parity Check with Interrupt

12539A Time Base Generator

~~12591A~~ 12561A Memory Protect Check

- Disc or Drum Mass Storage Unit(maximum of 4 units, 256 tracks total)
- HP2752A (ASR-33) System Teleprinter
- HP2754B (ASR-35) teleprinter for Batch Input, Punch and List Device

In place of the HP2754B (ASR-35) teleprinter, the user may select for example the following I/O devices instead for batch operations:

Batch List Device	Batch Input Device	Batch Punch Device
HP2752A Teleprinter	Punched Tape Reader	Punch Unit
HP2752A Teleprinter	Mark Sense Card Reader	Punch Unit
Line Printer	Punched Tape Reader	Punch Unit

DOS SOFTWARE MODULES

The following software tapes are supplied to the user:

- DOS Supervisor and sub-modules
- DOS Assembler
- DOS FORTRAN Compiler
- DOS Relocating Loader
- DOS Relocatable Library
- DOS I/O Drivers
- DSGEN, the DOS Generator

DOS Supervisor

The DOS Supervisor consists of a core- and disc-resident protected section (DISCM) and a disc-resident job processor, JOBPR.

DISCM

- Interrupt Processor
- Executive Processor
- I/O Processor
- Executive modules
  - $EXØ1 thru $EX16

NOTE: $EXØ1 through $EX16 may be either core- or disc-resident when a system is configured.

JOBPR

- Job Processor
- File Manager
  - PURGE
  - DUMP
  - STORE
  - LIST
  - EDIT

NOTE: JOBPR is always disc-resident. It is called to execute control commands.

System Programs

System Programs include DOS FORTRAN, DOS Assembler, and the DOS Relocating Loader. Both the DOS FORTRAN and the DOS Assembler consist of a main program and several segments. The DOS Relocatable Library consists of math, service and I/O subroutines which may be appended to a user program by the DOS Relocating Loader.

DOS I/O Drivers

The following I/O Drivers are discussed in Section V:

Name	Device
DVR00	Teleprinter
DVR01	Punched Tape Reader
DVR02	High Speed Punch
DVR12	Line Printer
DVR15	Mark Sense Card Reader
DVR22	3030 Magnetic Tape Unit
DVR30	Disc/Drum

DSGEN, THE Disc Operating System Generator

DSGEN, the DOS Generator, is an independent program which configures complete operating systems out of the DOS software modules, user programs, and information supplied about the I/O configuration.

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 or drum. 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.

DISC OPERATING SYSTEM

The Disc Operating System (DOS) is composed of user disc files and the DOS 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 Disc Operating System affords speed and convenience. Programs can be input to DOS for automatic translation, loading, and execution. For example, simple punched cards are able to carry out load-and-go operations in DOS as follows:

a. DOS 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 reads the Loader into core from the disc.

d. The Loader reads the relocatable binary programs from the disc and stores the converted binary instructions on the disc.

e. DOS reads the program in from the disc and runs it.

Directives

The DOS 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. They are entered in one of the two modes of DOS 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*.

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 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); return to keyboard mode (from batch mode).
- Check and set status of disc tracks.

DOS directives are described in detail in Section II.

EXEC Calls

After being translated and loaded, an executing user program communicates with DOS by means of EXEC calls. An EXEC call is a JSB instruction which transfers control to the DOS 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,
- Request status of disc track,
- Request limits of WORK area (temporary disc storage),
- Program completion,
- Program suspension,
- Loading of program segments,
- 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 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 DOS FORTRAN and DOS 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.)

Core Layout

When DOS 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 software modules, such as the FORTRAN Compiler, Assembler, Relocating Loader, and Job Processor, reside on the disc and execute in the user program area.

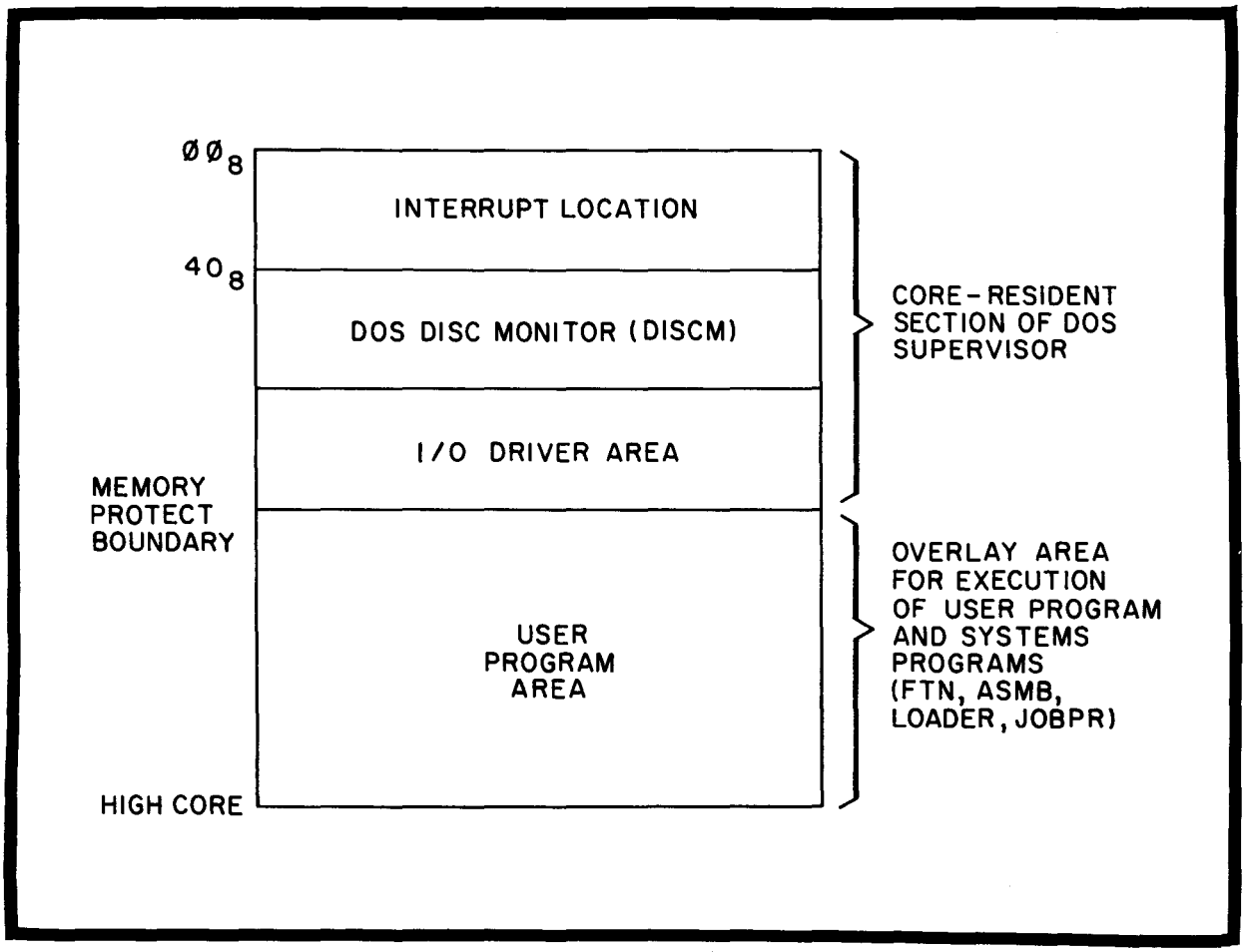

Figure 1-1. DOS Core Allocation

Disc Layout

The disc storage is divided logically into three areas: system area, user area, and work area. (See Figure 1-2.) In any installation, only the system area has a fixed size. DOS and its software reside permanently in the hardware-protected system area. Users' files of data and object programs reside in the user area. Work tracks are temporary storage for any executing program. The object code, which is generated by translators, is stored into the job binary area of the work tracks.

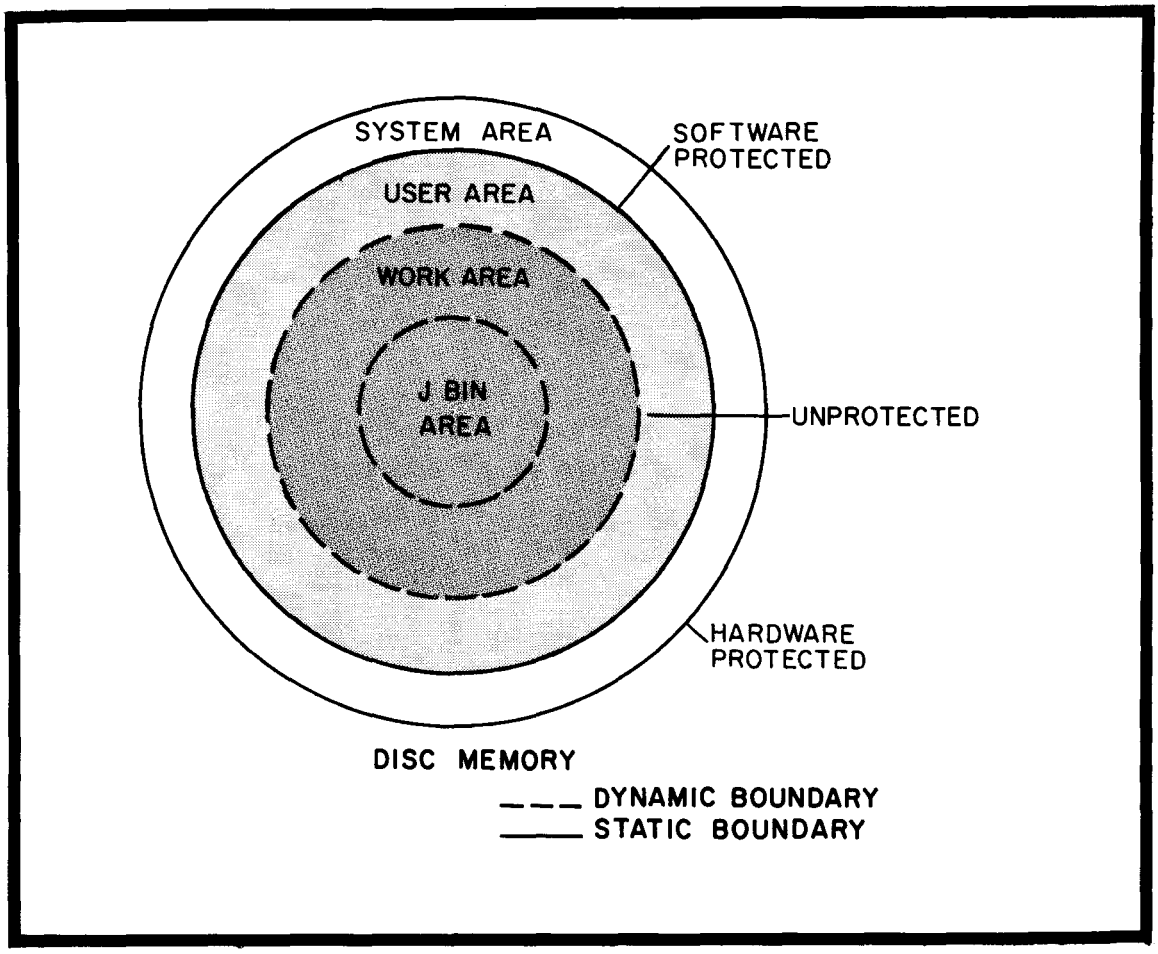

Figure 1-2. DOS Disc Storage

DOS Files

The disc or drum 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, and library subroutines (see *LIST*, Section II).

DOS Installation

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

Certain DOS 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 faster processing.

An absolute copy of the configured DOS is stored on the disc or drum and is protected from alteration by a hardware protect switch. A core-resident binary loader transfers DOS from the disc into core to start operations. (See *DOS Initiation from Disc*, Section VI.)

SECTION II
DIRECTIVES

Directives are the direct line of communication between the keyboard or batch input device and the Disc Operating System. The operator enters these directives manually through the keyboard, while 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 suspended 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, and
- Set the date and print comments.

Directives may enter DOS 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 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. For example,

```
:PROG,FTN,99
```

When optional parameters are missing, they 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. 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 has two conventions for notifying the operator that directives may be entered. An asterisk (*) means that DOS is waiting for an operator attention directive (see below). A "@" with the bell signals that DOS is waiting for further directions. (During some operations, such as editing, there may be perceptible waits while DOS processes the directive. Further directives *must* not be input until the "@" is output.)

The operator attains control of DOS at any time by striking any system teleprinter key. If the teleprinter is available, DOS prints an asterisk (*) on it; if it is busy, DOS 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

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

IGNORED

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 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 current time, and the date (as specified in the DATE directive):

JOB *name date* TIME = *xxxx* MIN. *xx.x* SECS.

For example,

```
:JOB,START
JOB START MON 6.16.9    TIME = ØØ13 MIN 41.6 SECS.
```

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 skips everything up to the comma.

EJOB

Purpose

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

Format

```
:EJOB
```

Comments

EJOB purges the user file, eliminating spaces left by non-permanent programs. 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 or BATCH are ignored until the next JOB 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. For example,

```
END JOB START RUN = ØØØ7 MIN. 52.6 SEC. EXEC = ØØØ1 MIN. 21.Ø SEC.
```

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

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 ignores all directives, except TRACKS, BATCH, or TYPE, until it finds a new JOB directive. An ABORT may be entered through the keyboard, even if DOS is in batch mode.

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 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 to the job in the previous 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
```

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 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.

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 Relocating Loader.

Format

:PROG,*name*[,P_1,P_2....P_5]

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

P_1 through P_5 are optional parameters which DOS transfers to the program named. P_1 through P_5 must be positive integers less than 32767.

Comment

Consult Section IV for the parameters required by FTN, ASMB, and LOADR. Additional programs may be added at system generation time if desired. (See *DOS Generator*, Section VI.)

Examples

```
:PROG,FTN,2,99
:PROG,ASMB,2,6,4
:PROG,LOADR,Ø,6,Ø,1,Ø
```

RUN

Purpose

To run a user program.

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).

N, if present, tells DOS 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 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 ultimately to execute with other HP software which does not have the same request codes. (See Appendix D, *RELATION TO OTHER SOFTWARE*.)

Example

:RUN,ROUT,15

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

TRACKS

Purpose

To print the status of the tracks on the disc, and optionally, to notify DOS of tracks known to be faulty.

Format

:TRACKS[,T_1,T_2,...]

where T_1,T_2.... are optional parameters which are used to notify DOS that faulty tracks exist. Faulty tracks may be reported only on a fresh start-up from the disc (following the DATE directive). Track numbers are decimal.

Comments

The number of the first track of the work area is printed, followed by the numbers of any faulty tracks. Each faulty track is listed separately.

The supervisor itself declares tracks down when a parity check on read occurs.

All tracks in the work area are available as user area tracks when STORE directives are encountered.

The operator should use TRACKS regularly to keep aware of the disc status, so that he can set faulty tracks unavailable on fresh starts.

Examples

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

```
(INPUT)    :TRACKS
(OUTPUT)   1ST WORK TRACK = ØØ1Ø
           @                          (End of directive processing)
```

In this example, the operator makes tracks 1Ø to 11 unavailable (only on a fresh start).

```
(INPUT)    :TRACKS,1Ø,11
(OUTPUT)   1ST WORK TRACK = ØØ12
           BAD=
           ØØ1Ø
           ØØ11
           @                          (End of directive processing)
```

STORE

Purpose

To create a user file on the disc. 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).

Format

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

TYPE-R	:STORE,R,*file*[,*logical unit*]
TYPE-P	:STORE,P[,$name_1$, $name_2$,...]
TYPE-S	:STORE,S,*file*,*logical unit*
TYPE-A	:STORE,A,*file*, *sectors*
TYPE-B	:STORE,B,*file*, *sectors*

Comments

TYPE - R FILES

The directive format is:

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

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

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 FORTRAN* and *DOS ASSEMBLY LANGUAGE*.)

If DOS 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 file name should not duplicate the name (in the NAM record) of any relocatable program within the file being stored if it is to be loaded via the Relocating Loader. The file may be input to the DOS Relocating Loader for relocation into an executable program. (See Section IV, *DOS 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.)

TYPE - P FILES

The directive format is:

:STORE,P[,$name_1$,$name_2$,....]

where $name_1$, $name_2$ ... are programs that the DOS 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 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 loader-generated programs--core images--in the work area into permanent user files.)

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

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

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 stores the succeeding data on the disc as if it were source statements.

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

DONE?

If there are more tapes or cards, the operator replies NO; otherwise, he answers YES.

When DOS 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.)

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. 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 checks that the file name exists and contains the sector specified.

Example

```
:STORE,A,ASCII,2Ø
```

(Creates a file named ASCII,20 sectors in length. A sector equals 64 words.)

SPECIFY SOURCE FILE

Purpose

To specify the user source file to be used as input by the assembler and compilers.

Format

:JFILE,*file*

Comments*

When the assembler or compiler is turned on, logical unit 2 (disc) must be specified as the input device. DOS looks up the *file* and transfers the source statements from the file to the translator as they are requested. (See Section IV, *DOS FORTRAN* and *DOS ASSEMBLY LANGUAGE*.)

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 another JFILE directive.

* IF LOGICAL UNIT 2 IS SPECIFIED AS THE INPUT DEVICE WHEN THE COMPILER OR ASSEMBLER IS TURNED ON (USING :PROG) AND :JFILE HAS BEEN DEFINED, THE COMPILER OR ASSEMBLER READS THE SOURCE STMNT USING A :STORE,5 DIRECTIVE.

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 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*) and the old file is not purged. Otherwise, the edited source file is the updated old file.

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

sequence numbers of the source statements to be edited, starting with one. Letter *m* signifies the starting statement, and *n* is the ending statement 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 returns to its previous mode for further directives.

Examples

If a file named SOURC contains:

```
Statement 1     ASMB,R,B,L
Statement 2             NAM START
Statement 3     A       EQU 3Ø
Statement 4     B       EQU 2Ø
Statement 5     START   NOP
Statement 6             LDA A
Statement 7             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 1ØØ
B       NOP
/D,4
/I,6
        STA B
/E
```

then the new file equals:

```
Statement 1     ASMB,R,B,L
Statement 2             NAM START
Statement 3     A       EQU 1ØØ
Statement 4     B       NOP
Statement 5     START   NOP
Statement 6             LDA A
Statement 7             STA B
Statement 8             END
```

PURGE

Purpose

To remove a user file from the user file area.

Format

:PURGE, $file_1$, $file_2$,...

where $file_1$, $file_2$,... (up to 15 file names 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

Comments

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 prints its name on the teleprinter.

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
```

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*$_1$,...]

(User) :LIST,U,*logical unit*[,*file*$_1$,...]

where X specifies the system area directory, and
U specifies the user area directory,
logical unit specifies the list device, and
file$_1$,... names the entries to be listed (if none is specified, the entire directory is listed).

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

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.

The following lines are then printed:

name type sctrs trk sec $lower_p$ $upper_p$ $lower_b$ $upper_b$ *entry libr*

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	
UM	=	user main program	Either File
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.

$lower_p$ is the lower limit (octal) of the program,

$upper_p$ is the upper limit (octal) of the program,

$lower_b$ is the lower limit (octal) of the program base page links,

$upper_b$ is the upper limit (octal) of the program base page links,

entry is the absolute octal address where execution begins, and

libr is the beginning absolute octal address of the first library routine included in the program.

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

file UNDEFINED
RE-ENTER STATEMENT ON TTY

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 a message on the system teleprinter,

**** LIST END **

Examples

```
:LIST,X,1
NAME TYPE SCTRS DISC ORG PROG LIMITS B.P. LIMITS  ENTRY  LIBR.
DVRØ1 DR  ØØØ5  TØØØ Ø64 1Ø626 11142 ØØ6Ø6 ØØ61Ø  1Ø626  11142
DVR15 DR  ØØØ5  TØØØ Ø69 1Ø626 11153 ØØ6Ø6 ØØ61Ø  1Ø626  11153
JOBPR UM  ØØ64  TØØØ Ø74 12ØØØ 21425 ØØ61Ø Ø1Ø75  12ØØØ  21425
LOADR UM  ØØ51  TØØ1 Ø53 12ØØØ 17512 ØØ61Ø Ø1211  12ØØØ  17512
ASMB  UM  ØØ41  TØØ2 Ø19 12ØØØ 16424 ØØ61Ø Ø1153  1627Ø  16424
ASMBD US  ØØØ7  TØØ2 Ø6Ø 16433 17135 Ø1153 Ø1154  16746  17135
ASMB1 US  ØØ11  TØØ2 Ø67 16672 2ØØ12 Ø1153 Ø1157  16672  2ØØ12
ASMB2 US  ØØ11  TØØ2 Ø78 16651 2ØØ4Ø Ø1153 Ø1156  16655  2ØØ4Ø
ASMB3 US  ØØØ3  TØØ3 ØØ4 171Ø4 17245 Ø1153 Ø1154  171Ø5  17245
ASMB4 US  ØØØ6  TØØ3 ØØ7 16672 17314 Ø1153 Ø1154  16672  17314
ASMB5 US  ØØ1Ø  TØØ3 Ø13 16651 17713 Ø1153 Ø1154  16655  17713
LIBRY LB  Ø173  TØØ3 Ø24
@

:LIST,U,1
NAME TYPE SCTRS DISC ORG PROG LIMITS B.P. LIMITS  ENTRY  LIBR.
F1    AD  ØØØ1  TØØ9 ØØØ
F2    BD  ØØØ2  TØØ9 ØØ1
FLONG AD  Ø9ØØ  TØØ9 ØØ3
@

:LI,S,1,DVRLP,316
Ø316  EQT12 EQU EQT1+11
Ø317  I.12  EQU I.11
Ø318  C.12  EQU C.11
Ø319        END
****  LIST  END**
@
```

FILE DUMP

Purpose

To dump a user file on a specified 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

File Type	Punch Device	List Device
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

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
EEEEE
FFFFFF
GGGGGGG
@
```

Where DVR is binary file:

```
:DUMP,1,DVR,1,1
ØØ1  Ø1Ø4ØØ  Ø2ØØØØ  164165  Ø4216  Ø51Ø6Ø  Ø3Ø44Ø  ØØØ313  ØØØØØØ
     ØØØØØØ  ØØØØØ4  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØ54ØØ  Ø4ØØØ2  Ø66352  Ø44456  Ø3ØØ61  Ø2ØØØØ  ØØØØØØ
     Ø41456  Ø3ØØ61  Ø2ØØØØ  ØØØØ52  Ø36ØØØ  Ø6Ø146  171620  ØØØØØØ
     Ø12ØØØ  ØØØØØØ  Ø16ØØØ  ØØØ266  16Ø213  Ø1ØØ56  Ø5ØØ54  12Ø12Ø
     Ø26ØØØ  ØØØØ26  ØØ64Ø4  Ø5ØØ55  Ø26ØØØ  ØØØØ24  16Ø213  ØØØØØØ
     ØØ1727  ØØ1222  Ø1ØØ74  Ø5ØØ62  ØØ2ØØ1  121ØØØ  Ø26ØØØ  ØØØØ23
     16Ø2Ø6  Ø32ØØØ  ØØØ3ØØ  17Ø2Ø6  ØØ6ØØ4  Ø12ØØØ  Ø6ØØØ1  126ØØØ
@
```

DISC DUMP

Purpose

To dump any sector of the disc storage on the system teleprinter 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.

Example

```
:SO,8,Ø
ØØ1  ØØ44ØØ  177ØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØ2  177756Ø  Ø41124  Ø4151Ø  Ø3Ø44Ø  Ø51525  Ø47Ø4Ø
     Ø2ØØ4Ø  Ø2ØØ4Ø  Ø2ØØ4Ø  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
     ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ  ØØØØØØ
```

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,

LWA is the last 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.

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. These flags are reset when a program executes. 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,

adr_8	*wd-1*	*wd-2*	*wd-3*	*wd-4*	*wd-5*	*wd-6*	*wd-7*	*wd-8*
ad_8+10_8	*wd-1*	*wd-2*	*wd-3*	*wd-4*	*wd-5*	*wd-6*	*wd-7*	*wd-8*

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,

	pair-1		*pair-2*		*pair-3*		*pair-4*	
adr	*item-1*	*item-2*	*item-1*	*item-2*	*item-1*	*item-2*	*item-1*	*item-2*
$adr+4_8$	*item-1*	*item 2*	*item-1*	*item-2*	*item-1*	*item-2*	*item-1*	*item-2*

Example

```
:ADUMP,Ø,15,B      (Set up dump flag)
:RUN,LOADR         (Run program)
LU    Ø12140
ABRT  Ø12140       (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

und[redacted]
sert the following:

"Any parameter following L is ignored. If FWA is greater than LWA, a message is printed. When the directive :PDUMP precedes a :RUN or :PROG request, the program contained in the request will be dumped, if it runs to normal completion. To dump a program that is aborted while running, the directive :ADUMP must precede the :RUN request. To make sure that a program will be dumped whether it runs normally or is aborted, both dump directives must be declared preceding the :RUN request. Only one of the requests will be honored, depending upon whether the, program runs normally or is aborted. Since DOS sets a flag when it encounters either dump directive, then clears the flag after the dump routine is executed, the flag representing the dump routine that was not executed will remain set. This flag can cause an unwanted dump of some program run later under the same :JOB directive. Either dump flag can be cleared by requesting the dump with both FWA and LWA equal to 0; all flags can be cleared by calling a new :JOB directive."

Pg 2-29

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* U*u* S*s*

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 sub-channel addressing,
s is the availability status of the device:
Ø for not busy, and available,
1 for disabled (down),
2 for busy,
3 for awaiting an available DMA channel.

Example

```
:EQUIPMENT TABLE
EQT Ø1 CH 11 DVRØØ Ø R UØ SØ
EQT Ø2 CH 17 DVR3Ø D R UØ SØ
EQT Ø3 CH 13 DVRØ1 Ø Ø UØ SØ
EQT Ø4 CH 23 DVR15 Ø Ø UØ SØ
@
```

LOGICAL UNIT

Purpose

To assign logical unit numbers 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 1 through 9 are only valid during the current job. 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. (See Section VI, *DOS Generator*). This insures a standard I/O organization for all users.

Example

```
:LUN TABLE
LUØ1 EQTØ1
LUØ2 EQTØ2
LUØ3 EQTØØ
LUØ4 EQTØ1
LUØ5 EQTØ3
LUØ6 EQTØ1
LUØ7 EQTØ4
@
```

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 is usually used in response to the following message from DOS:

$$\text{I/O ERR} \left\{ \begin{matrix} \text{ET} \\ \text{or} \\ \text{NR} \end{matrix} \right\} \text{EQT \#}n$$

where ET indicates end of tape,
NR indicates device not ready, and
n is the equipment entry number.

DOWN

Purpose

To declare an I/O device unavailable for use.

Format

:DN,*n*

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

Comments

The system teleprinter and system disc cannot be set down. Once set down, a device is unavailable until set UP by the operator.

NOTE:

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

BATCH

Purpose

To switch from keyboard mode to batch mode.

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.

DATE

Purpose

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

Format

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

where *day* is any string of ten or less characters chosen by the ~~user~~ OPERATOR (such as 7/1Ø/69,1Ø.JULY.69, etc.);

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

Comments

The DATE directive is legal only following a start-up procedure. (See Section VI, *DOS INITIATION FROM THE DISC*.) The directive is not accepted any other time.

Examples

:DATE,7/1Ø/69,12,23
:DATE,WEDNESDAY,7,45
:DATE,1ØJULY1969

GO

Purpose

To restart 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 Ø 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.

SECTION III
EXEC CALLS

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

- Perform input and output operations,
- Request status of disc tracks,
- Terminate or suspend itself,
- Load its segments,
- Search for file names, or
- Obtain the time of day.

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 causes a memory protect violation interrupt and transfers control to DOS. DOS 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 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 is equivalent to the corresponding Assembly Language object code.

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

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)
         :
        JSB     EXEC            (Transfer control to DOS)
        DEF     *+n+1           (Defines point of return from DOS, n
                                 is number of parameters; may not be
                                 an indirect address)
        DEF     P1      }       (Define addresses of parameters which
         :              }        may occur anywhere in program; may be
        DEF     Pn      }       multi-level indirect)
        return point            (Continue execution of program)
         .
         :
         .
         :
P1      ---             }
 .                      }
 .                      }       (Actual parameter values)
 .                      }
Pn      ---             }
```

FORMAT OF THE FORTRAN CALLING SEQUENCE

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

$$\text{CALL EXEC } (P_1, P_2 \ldots . , P_n)$$

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)
     or
IRCDE = 7
CALL EXEC (IRCDE)
```

} Equivalent calling sequence

Some EXEC call functions are handled automatically by the FORTRAN compiler or special subroutines. (Refer to "FORTRAN," Section IV, *DOS 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 handles track and disc switching automatically.)

Assembly Language

```
        EXT     EXEC
         :
        JSB     EXEC            (Transfer control to DOS)
        DEF     *+5 (or 7)      (Point of return from DOS; 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     conwd           (conwd is described in Comments)
BUFFR   BSS     n               (Buffer of n words)
BUFFL   DEC     n (or -2n)      (Same n; words (+) or characters (-))
DTRAK   DEC     f               (Work area track number, decimal)
DSECT   DEC     g               (Work area sector number, decimal)
```

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	1Ø	9	8	7	6	5	4	3	2	1	Ø

Field	Function
W	If 1, tells DOS to return to the calling program after starting the I/O transfer. If W = Ø, DOS 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).

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 Ø, 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(1Ø), BUF(2Ø)
      LUN = 2
      ITRK = 12
      ISECT = 63
      IOFF = Ø
      CALL BREAD (BUF, 4Ø, LUN, ITRK, ISECT, IOFF),
or
      CALL BWRIT (IBUF, 1Ø, 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 returns the number of transmitted words or characters to the B register.

FILE READ/WRITE

Purpose

To transfer information to or from a user file on the disc; the file must be referenced by name.

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC            (Transfer control to DOS)
        DEF *+7             (Point of return from DOS)
        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)
```

FORTRAN

```
DIMENSION
DIM  IFILE  (3)
IFILE(1) = xxxxxB          (First two characters of file name)
IFILE(2) = xxxxxB          (Second two characters)
IFILE(3) = xxxxxB          (Last character and blank)
IRCDE = 14 (or 15)         (Request code)
ICNWD = xxxxxB             (conwd)
DIMENSION
DIM  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*=Ø; 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.

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)
        DEF  *+4(or 3)     (Point of return from DOS)
        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:

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

If W = 1, DOS returns to the calling program after starting the control request.
If W = Ø, DOS waits until the control request is complete before returning.

Function Code (Octal)	Action
ØØØ	Unused
ØØ1	Write end-of-file (magnetic tape)
ØØ2	Backspace one record (magnetic tape)
ØØ3	Forward space one record (magnetic tape)
ØØ4	Rewind (magnetic tape)
ØØ5	Rewind standby (magnetic tape)
ØØ6	Dynamic status (magnetic tape)
ØØ7	Set end-of-paper tape
Ø1Ø	Generate paper tape leader
Ø11	List output line spacing (PARAM or IPRMA required)
Ø12 : 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)
        DEF *+5           (Point of return from DOS)
        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)
```

Comments

The status returned is the hardware status of the device specified by the logical unit. The transmission log 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). The disc is a special case because transfers are broken up by DOS when the transfer crosses track boundaries or switches physical disc files. Only the amount transmitted in the last transfer is recorded in TLOG.

WORK AREA LIMITS

Purpose

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

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC        (Transfer control to DOS)
        DEF *+5         (Point of return from DOS)
        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,ISIZE)
```

Comments

This call returns the limits of the work area, that area of the disc which programs use for temporary storage with the READ/WRITE EXEC call. Some tracks within the work area may be faulty. Therefore, if the program requires consecutive tracks, it should check the work area track status. (See WORK AREA STATUS EXEC call.)

WORK AREA STATUS

Purpose

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

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC          (Transfer control to DOS)
        DEF *+5           (Point of return from DOS)
        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).
```

FORTRAN

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

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 handles track and disc switching automatically.

DOS checks whether there are *n* consecutive operable tracks starting at the track specified. If not, DOS scans through the work area looking for *n* consecutive operable tracks. Upon location of tracks, DOS returns the starting track number to the program. If DOS does not locate *n* consecutive tracks, it returns Ø in TRAK or ITRAK.

PROGRAM COMPLETION

Purpose

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

Assembly Language

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

FORTRAN

The FORTRAN compiler generates a PROGRAM COMPLETION EXEC CALL automatically when it compiles an END statement.

The programmer may use an EXEC call instead:

```
      ICODE = 6           (Request code)
      CALL EXEC(ICODE)
```

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)
        DEF *+2           (Point of return from DOS)
        DEF RCODE         (Request code)
        return point      (Continue execution)
         :
RCODE   DEC 7             (Request Code = 7)
```

FORTRAN

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

The programmer may use an EXEC call instead:

```
      CALL EXEC (7)
```

Comments

DOS 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 PROGRAMMING*, for information on segmented programs.)

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC        (Transfer control to DOS)
        DEF *+3         (Point of return from DOS)
        DEF RCODE       (Request code)
        DEF SNAME       (Segment name)
        return point    (Continue execution)
         :
RCODE   DEC 8           (Request code = 8)
SNAME   ASC 3,xxxxx     (xxxxx is the segment name)
```

FORTRAN

```
        IRCDE = 8
        DIMENSION
        DIM INAME (3)
        INAME (1) = xxxxxB      (First two characters)
        INAME (2) = xxxxxB      (Second two)
        INAME (3) = xxxxxB      (Last character)
        CALL EXEC (IRCDE,INAME)
```

Comments

In the FORTRAN 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 NAMES

Purpose

To check whether a specific file name exists in the directory of user or system files.

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC          (Transfer control to DOS)
        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)
```

FORTRAN

```
        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 and blank)
        CALL EXEC (IRCDE,INAME,ISECT)
```

TIME REQUEST

Purpose

To request the current time.

Assembly Language

```
        EXT EXEC
         :
        JSB EXEC            (Transfer control to DOS)
        DEF *+3             (Point of return from DOS)
        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
        DIM ITIME (5)
        CALL EXEC (IRCDE,ITIME)
```

Comments

When DOS 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 = Ø)

SECTION IV
PROGRAMMING

Section IV describes the operating procedures and formatting conventions of the four user programming aids of DOS:

- FORTRAN Compiler
- Assembler
- Relocating Loader
- Relocatable Library

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

LOAD-AND-GO FACILITY

The 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 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 FORTRAN COMPILER

The DOS FORTRAN Compiler, a segmented program, operates under control of the DOS Supervisor. The compiler consists of a main program (FTN) and four overlay segments (FTNØ1, FTNØ2, FTNØ3, FTNØ4). It resides in the protected area of the disc and is read into core only when needed. The compiler requires at least a 4K user area of core.

DOS FORTRAN, a problem-oriented programming language, is very similar to regular HP FORTRAN. 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 Relocating Loader and executed using the RUN or PROG directive.

Compiler Operation

The DOS FORTRAN compiler is 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

:PROG,FTN [,p_1,p_2,p_3,p_4,99]

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.

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

$END, FTN

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-ØØ19 is printed on the list device. ($END,FTN 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.

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 o 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, ERR*n*. 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.

type is a decimal digit specifying the program type. Only types 3 (main), 5 (segment), and 6 or 7 (library) are significant in DOS. The type is set to 3 if not given.

Seven more parameters may be included but they are used only with the HP2005A Real-Time Executive System. Programs can be compiled on DOS to be run under Real-Time. (Consult the *Real-Time Software* Manual.)

I/O LOGICAL UNIT NUMBERS

DOS 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 aborts the job if a directive occurs during data input.

DATA STATEMENT

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

$$\text{DATA } k_1/d_1/,k_2/d_2/,\ldots,k_n/d_n/$$

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.Ø,2.Ø,3.Ø/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 $v_1, v_2, \ldots, v_5$

Where v_i 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-ØØ18 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 "%". See *DOS Relocatable Library*, in this section.

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 *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.

Check the *FORTRAN* manual (Chapter 9.9) for the meaning of error codes.

REFERENCE ON FORTRAN

For a complete description of the FORTRAN language, read the *FORTRAN* programmer's reference manual (02116-9015). Sections 9.5,9.6, and 9.8 are not pertinent to DOS FORTRAN.

DOS ASSEMBLER

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

DOS 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. 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 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. There are no magnetic tape assemblies.

Assembler Operation

The DOS 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.

If the values of p_1 through p_4 are not set, the standards are used.

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
```

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.

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 ØØØ 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 Assembly Language

The DOS 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; 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, the DOS Assembler is able to assemble absolute code if it is specified. Absolute code is never 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.

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 HP2005A Real-Time Executive System.) Only the first parameter is significant in DOS. 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, 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 communication, while the *entry points* are program-to-program communication.

ORB STATEMENT

DOS 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 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 aborts the entire job if a directive (signified by : in column one) occurs during data input.

The memory protect feature 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 when an interrupt occurs, and DOS checks whether it was an EXEC call. If not, the user program is aborted.

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.

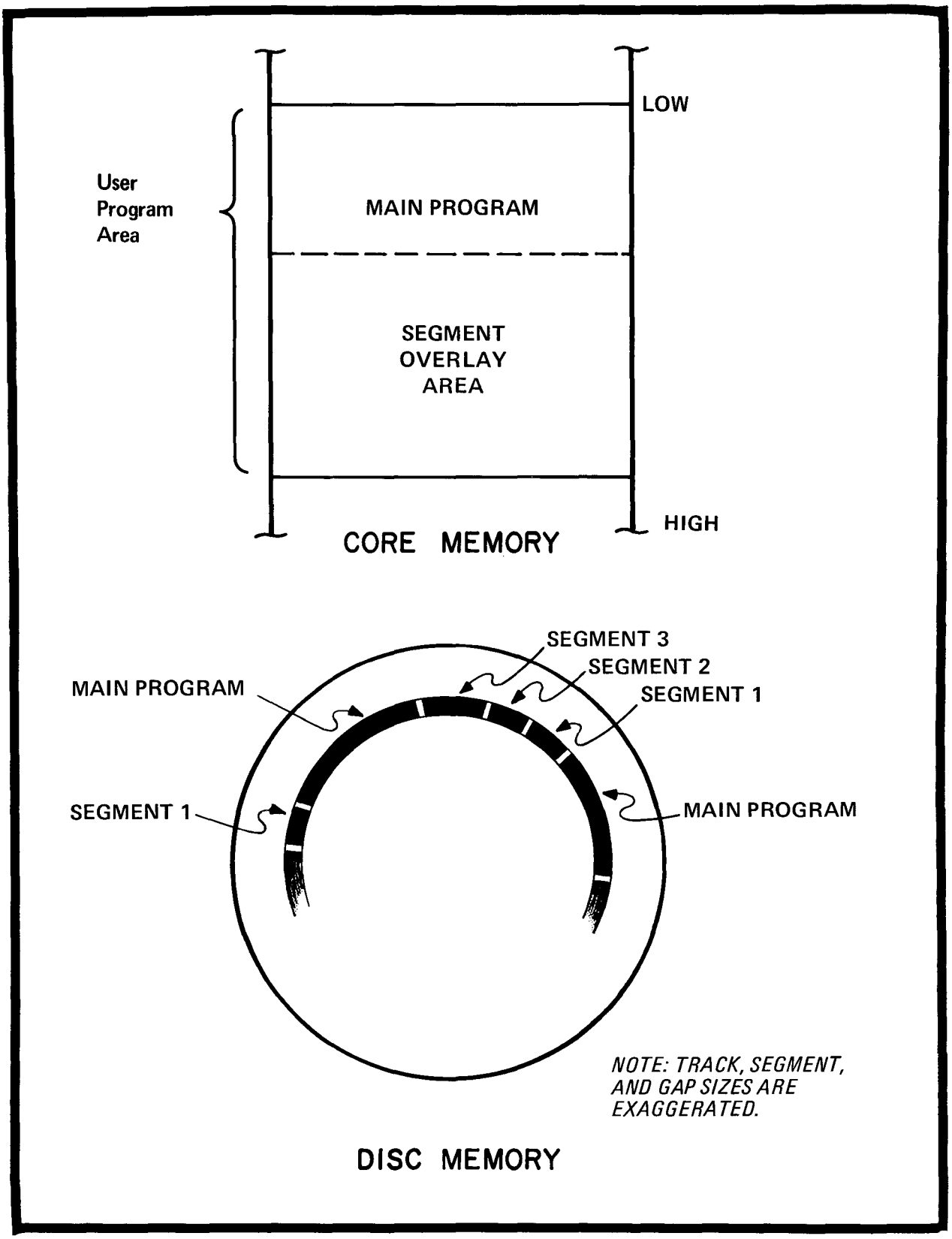

Figure 4-1. Segmented Programs

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 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).

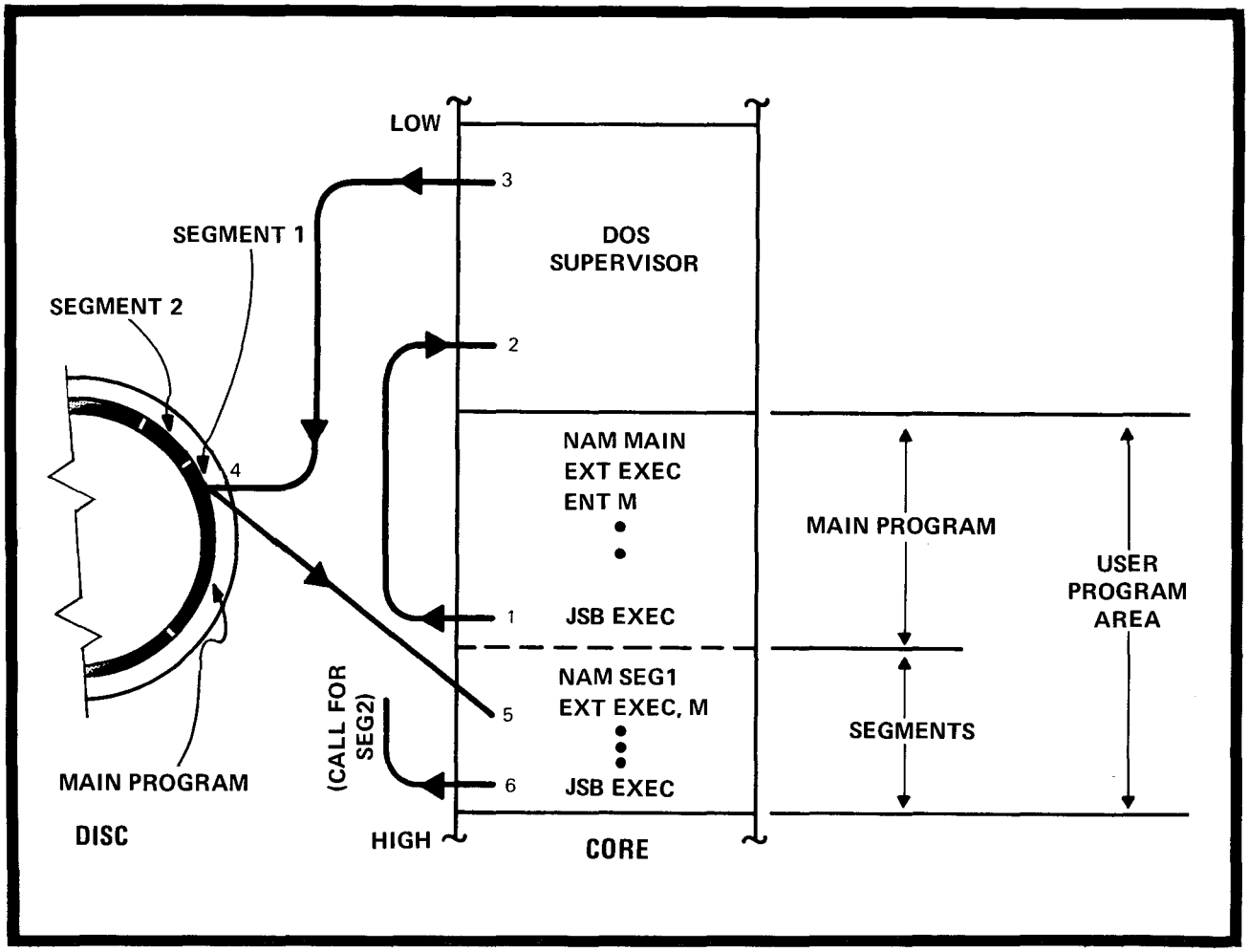

Figure 4-2. Main Calling Segment

Figure 4-3 shows how DOS 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.

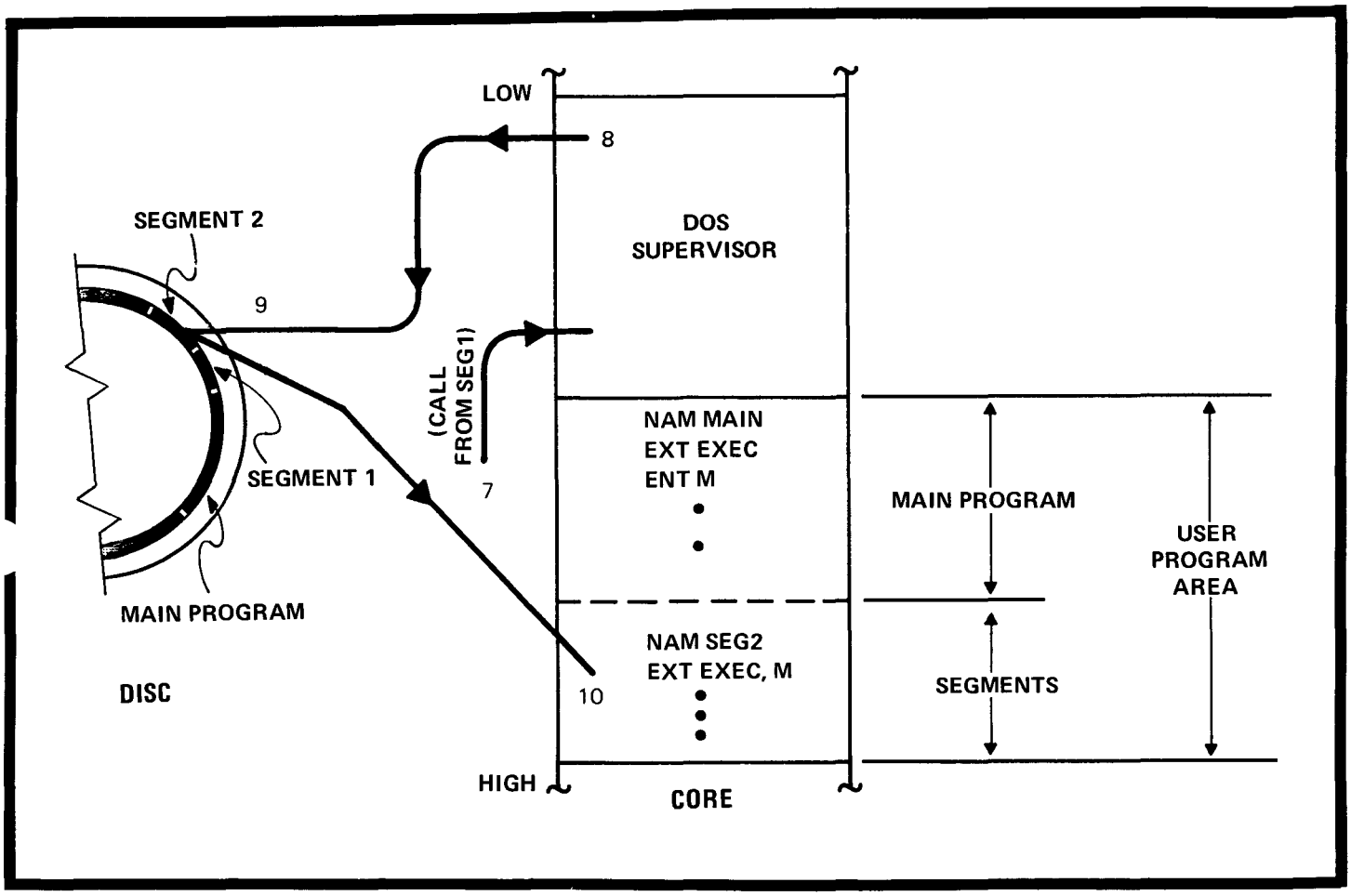

Figure 4-3. Segment Calling Segment

When a main program and segment are currently residing in core, they operate as one 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 instruction. (See Figure 4-4.) A matching entry symbol must be defined as the destination in the other program. DSGEN associates

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. Sections 5.5 and 5.6 of that text do not apply to DOS.

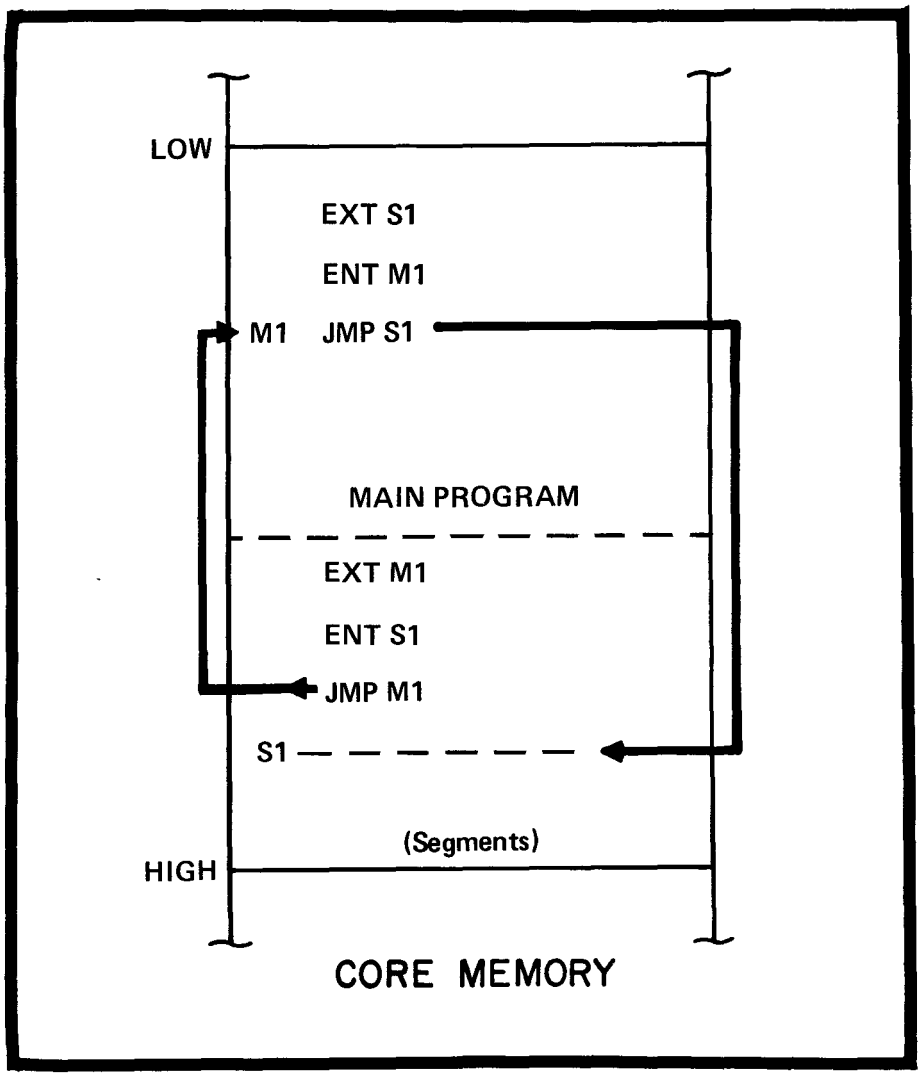

Figure 4-4. Main-to-Segment Jumps

DOS RELOCATING LOADER

The DOS Relocating Loader accepts relocatable object programs which have been translated by the DOS Assembler or DOS 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, 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. 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 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 = Ø 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 (logical unit *n*).

P_2 = list device logical unit.

P_3 = Ø for no DEBUG, ≠ Ø for DEBUG.

P_4 = Ø for list of program load map, ≠ Ø for none.

P_5 = Ø for list of entry point addresses, ≠ Ø for none.

If values $P_1,...P_5$ are not set, P_1=Ø, P_2=6, P_3=P_4=P_5=Ø.

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 Ø). 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.

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.

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
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 DOS 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.

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

:GO,Ø[,*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 user and system files. 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 COMPLETED

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 directive.

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

```
NO PROGRAM LOADED
LOADR COMPLETED
```

DEBUG Library Subroutine

DOS DEBUG, a subroutine of the DOS 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
```

DEBUG OPERATIONS

B,*A*	Instruction breakpoint at address *A*. (NOTE: if *A* = JSB EXEC, a memory protect violation occurs.)
D,A,N_1[,N_2]	ASCII dump of core address N_1 or from N_1 to N_2.
D,A,N_1[,N_2]	Binary dump of core address N_1 or from N_1 to N_2.
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_1,D_1	Set D_1 in location A_1.
S,A_1,D_1,D_n	Set D_1 to D_n in successive memory locations beginning at location A_1.
W,A,D_1	Set A-Register to D_1.
W,B,D_2	Set B-Register to D_2.
W,E,D_3	Set E-Register (Ø=off, non-zero=on).
W,O,D_4	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 is in keyboard mode.

```
:PROG,LOADR,5,6,Ø,Ø,Ø        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 # Ø3          End of tape.
LOAD TAPE                    Put in next tape.
LOADR SUSP
@:UP,3                       Declare input device ready.
@:GO
I/O ERR ET EQT # Ø3
LOAD TAPE
LOADR SUSP
@:UP,3
@:GO
I/O ERR ET EQT # Ø3
LOAD TAPE
LOADR SUSP
@:UP,3
@:GO                         Repeat tape loading process 4 times.
I/O ERR ET EQT # Ø3
LOAD TAPE
LOADR SUSP
@:UP,3
@:GO
I/O ERR ET EQT # Ø3
LOAD TAPE
LOADR SUSP
@:UP,3
@:GO,1                       No more paper tapes.
```

RELOCATING LOADER

NAME/ENTRY	ADDR	
QA1	12ØØØ	Main program, starting address.
*QA1	12Ø76	Main program, entry point.
QA1A	122ØØ	Subroutine starting addresses and entry points. Asterisk signifies entry point.
*QA1A	122Ø1	
QA1B	12262	
*QA1B	12263	
QA1C	12336	
*QA1C	12337	
QA1D	12364	
*QA1D	12365	
FRMTR	12431	
*.DIO.	14612	
*.BIO.	14665	
*.IOI.	145Ø7	
*.IOR.	14462	
*.IAR.	14546	
*.RAR.	14522	
*.DTA.	1471Ø	
.ENTR	15162	
*.ENTR	15162	
.FLUN	1523Ø	
*.FLUN	1523Ø	
.PACK	15243	
*.PACK	15243	
FLOAT	1535Ø	
*FLOAT	1535Ø	
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:

Message	Error Messages
LØ1	Checksum error on tape
LØ2	Illegal record
LØ3	Memory overflow
LØ4	Base page overflow
LØ5	Symbol table overflow
LØ6	Duplicate main or segment name (may be caused by attempting to run the loader twice in one job)
LØ7	Duplicate entry point
LØ8	No main or segment transfer address
LØ9	Record out of sequence
L1Ø	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)

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

LOADR TERMINATED

DOS RELOCATABLE LIBRARY

The DOS Relocatable Library is a collection of relocatable mathematics and service subroutines which are stored on the disc. A program signifies its need for a subroutine by means of an "external reference"--created by an EXTERNAL statement in assembly language, automatically in FORTRAN.

Many of the subroutines are equivalent to subroutines of the Hewlett-Packard *BCS RELOCATABLE PROGRAM LIBRARY*, but modified internally to run under DOS. For a list of the library subroutines and their entry points, see Table 4-1.

Table 4-1
Library Subroutines

Subroutine Name	Subroutine Entry Points
FRMTR	.DIO.
	.BIO.
	.IOI.
	.IOR.
	.IAR.
	.RAR.
	.DTA.
%ANH	%ANH
%XP	%XP
%IN	%IN
%OS	%OS
%AN	%AN
%BS	%BS
%LOG	%LOG
%QRT	%QRT
%IGN	%IGN
%LOAT	%LOAT
%FIX	%FIX
%TAN	%TAN
%ABS	%ABS
%SIGN	%SIGN

Subroutine Name	Subroutine Entry Points
%AND	%AND
%OR	%OR
%OT	%OT
GETAD	GETAD, ADRES
TANH	TANH
.RTOR	.RTOR
TAN	TAN
EXP	EXP
SICOS	SIN, COS
SQRT	SQRT
SIGN	SIGN
ALOG	LN, ALOG
.IENT	.IENT
ABS	ABS
ATAN	ARCTA, ATAN
PWR2	PWR2
FDV	.FDV
FMP	.FMP
FLOAT	FLOAT
..FCM	..FCM
IFIX	IFIX
FADSB	.FAD.
.RTOI	.RTOI
.ITOI	.ITOI
ISIGN	ISIGN
IABS	IABS
CHEBY	.CHEB
MANT	.MANT
.PACK	.PACK
..DLC	..DLC
.ENTR	.ENTR
.FLUN	.FLUN
.GOTO	.GOTO
IAND	IAND

Subroutine Name	Subroutine Entry Points
IOR	IOR
OVF	OVF
.MAP.	.MAP
RMPAR	RMPAR
PAUSE	.PAUS, .STOP
ERRØ	ERRØ
BINRY	BREAD,BWRIT
DLDST	.DLD, .DST
MPY	.MPY
DIV	.DIV
SREAD	%READ, %JFIL, %RDSC
%WRIS	%WRIS, %WRIN, %WEOF
%WRIT	%WRIT, %WRIF
ASCII	CNDEC, CNOCT
$SRCH	$SRCH
$ADDR	$ADDR
DEBUG	$DBP1, DEBUG
DBKPT	$DBP2, $MEMR
PTAPE	PTAPE

% Library Routines

In Table 4-1, some routines start with the character "%". The rest of the subroutine name is the same as some other subroutine (e.g., SIN becomes %IN). A subroutine starting with "%" is a call-by-name version of a call-by-value subroutine that does the same operation. In the call-by-value subroutine, the actual value of the parameter must be replaced in the A- and B-Registers as an integral part of the calling sequence. The subroutine searches the registers for the parameter.

Call-by-name subroutines, on the other hand, expect a list of parameter addresses following the subroutine call. (The EXEC calls given in Section III demonstrate the form of a call-by-name subroutine's calling sequence.) In FORTRAN, using the special EXTERNAL statement, subroutines may be passed as

parameters to other subroutines. Since the subroutines receiving the parameter cannot know in advance which type (call-by-value or call-by-name) will be passed, it must assume call-by-name for generality. In FORTRAN, subroutine parameters are assumed to be the call-by-name type subroutines and the appropriate calling sequence is generated.

When the FORTRAN compiler encounters SIN or another of the call-by-value subroutine within an EXTERNAL statement, the compiler knows that the subroutine is going to be used as a parameter. Since SIN will be assumed to be call-by-name, the compiler automatically changes the external reference to %IN, the call-by-name version of SIN. (*NOTE: %WRIS and %WRIT are exceptions to the % routines*).

For Example,

```
SUBROUTINE SUB (PARAM)      (PARAM is a subroutine parameter)
CALL PARAM (A)              (The call to PARAM is assumed to
                             be call-by-name)
RETURN
END
```

Subroutines Unique to DOS

RMPAR, ERRØ, BINRY, and the %-routines are unique to DOS. RMPAR is explained in Section II, *GO directive*. ERRØ is explained in this section under *DOS FORTRAN COMPILER*. BINRY is explained in Section IV, *READ/WRITE EXEC CALLS*. The % routines are explained above.

Assembly Language Calling Sequences

The calling sequences for the DOS Relocatable Library subroutines are identical to those for the regular Relocatable Program Library routines, with the following exceptions:

For SIN, COS, ALOG, SQRT, EXP, and TAN, the calling sequence is:

```
DLD          Argument
JSB          Subroutine name
JSB          Error routine (either ERRØ or a user
             routine, ERRn, where n = 1 to 9)
normal return point
```

Before returning to the error routine location, the subroutines place an ASCII error code in the A- and B-Registers. ERRØ prints this code on the system teleprinter. A user error routine may handle these errors.

For .RTOR, .RTOI, and .ITOI, the calling sequence is

```
JSB          Subroutine name
DEF          Argument one
DEF          Argument two
JSB          Error-print routine
normal return point
```

Reference

For further information on the library subroutines, see the *PROGRAM LIBRARY* programmer's reference manual (02116-9032).

SECTION V
INPUT/OUTPUT

In the 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 (10_8 through 37_8) 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 relieves the programmer of most I/O problems.

For further details on the hardware input/output organization, consult Volume One, *SPECIFICATIONS AND BASIC OPERATION MANUAL*, Model 2116B Computer (02116-9152).

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.

The Equipment Table

The Equipment Table (EQT) has an entry for each device recognized by DOS (these entries are established by the user when DOS 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 system use)																
6	(saved for system 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																Ø's if core-resident
16	Upper Memory Address of Driver Linkage Area																
17	Starting Track #								Starting Sector #								
BITS	15	14	13	12	11	1Ø	9	8	7	6	5	4	3	2	1	Ø	

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.)

Av = Ø - Unit not busy and available
1 - Unit disabled (down)
2 - Unit busy
3 - Unit waiting for an available DMA channel

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:

ØØ	Teleprinter
Ø1-Ø7	Paper Tape Devices
Ø1	Punched Tape Reader
Ø2	High Speed Punch
1Ø-17	Unit Record Devices
1Ø	Reserved for Plotter
12	Line Printer
15	Mark Sense Card Reader
2Ø-37	Magnetic Tape/Mass Storage and other devices capable of both input and output
22	3030 Magnetic Tape
3Ø	Disc/Drum

For equipment type codes Ø1 through 17, odd numbers indicate input devices and even numbers indicate output devices.

When DOS 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 $1_{1Ø}$ to $63_{1Ø}$ 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 Disc Operating 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 has the following function assignments for logical unit numbers.

Logical Unit Number	Function
1	System Teleprinter
2	System Mass Storage
3	Auxiliary Mass Storage #1
4	Standard Punch Device
5	Standard Input Device
6	Standard List Device
7	Reserved for use by DOS
8	
9	
10	Can be assigned to any device by user
:	
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), where as 10 through 63 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. 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 halts. However, the user can write his own routine to handle power failure interrupts.

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 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. When an interrupt occurs on the channel, indicating continuation or completion of the transfer, DOS again transfers control to the driver.

DOS currently includes seven standard I/O drivers:

DVRØØ - Teleprinter
DVRØ1 - Photo-reader
DVRØ2 - High speed punch
DVR12 - Line Printer
DVR15 - Mark Sense Card Reader
DVR22 - 3030 Magnetic Tape
DVR23 - " "
DVR3Ø - Disc/drum

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 handles the availability field.

System I/O

DOS 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 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 $IORQ which determines if the driver for the requested device is core-resident. If not, the driver is read into core from the disc.

$IORQ 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, the appropriate channel, initiates the transfer and returns to $IORQ. DOS 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 HP2116B computer, control is transferred to the instruction in the interrupt location corresponding to the device. Each interrupt location (memory locations 10_8 through 37_8) contains a "JSB $CIC" instruction. $CIC, the central interrupt control routine of DOS, 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
    3) If time base, goes to CLOCK routine
    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 ($IORQ) and Central Interrupt Control ($CIC) modules of DOS, 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.

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 DVR*nn* is the driver name.

Initiation Section

The $IORQ module 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 Ø through 5 of EQT entry word 3). The driver returns to $IORQ by an indirect jump through I.*nn*.

Before transferring to I.*nn*, DOS 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 Ø 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, 5, 6, 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.

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 of the parameters, is illegal.
2. Configures all I/O instructions in the driver to include the select code (and DMA channel) of the device.
3. Initializes DMA, if appropriate.
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 $IORQ with the A-register set to indicate initiation or rejection and the cause of the reject:

   If A = Ø, then the operation was initiated.

   If A ≠ Ø, 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).

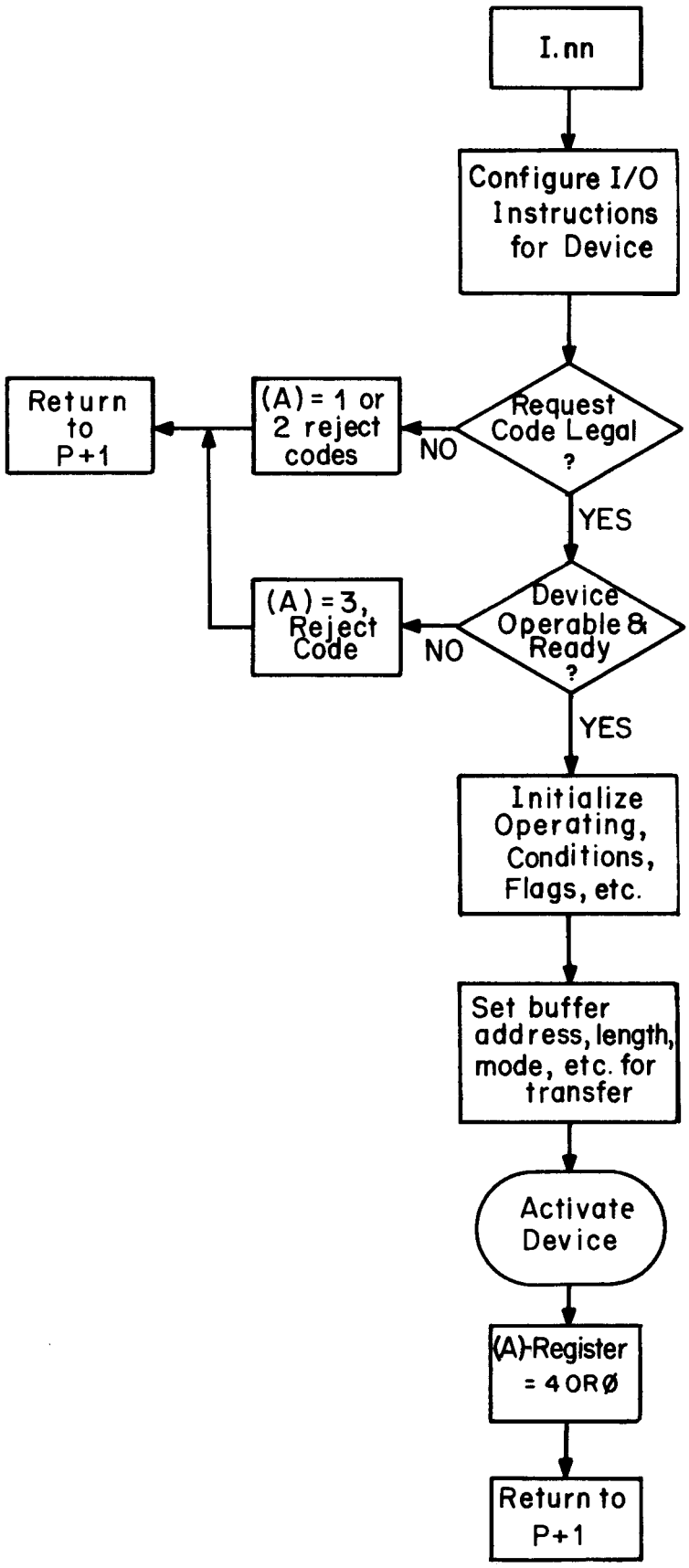

Figure 5-1. I/O Driver Initiation Section

Completion Section

DOS 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:

Location	Action
	Set A-register equal to interrupt source code
(P)	JSB C.*nn*
(P+1)	Completion return from C.*nn*
(P+2)	Continuation or error retry 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.
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 EQT 14. 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 Ø 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.

      Ø = 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.

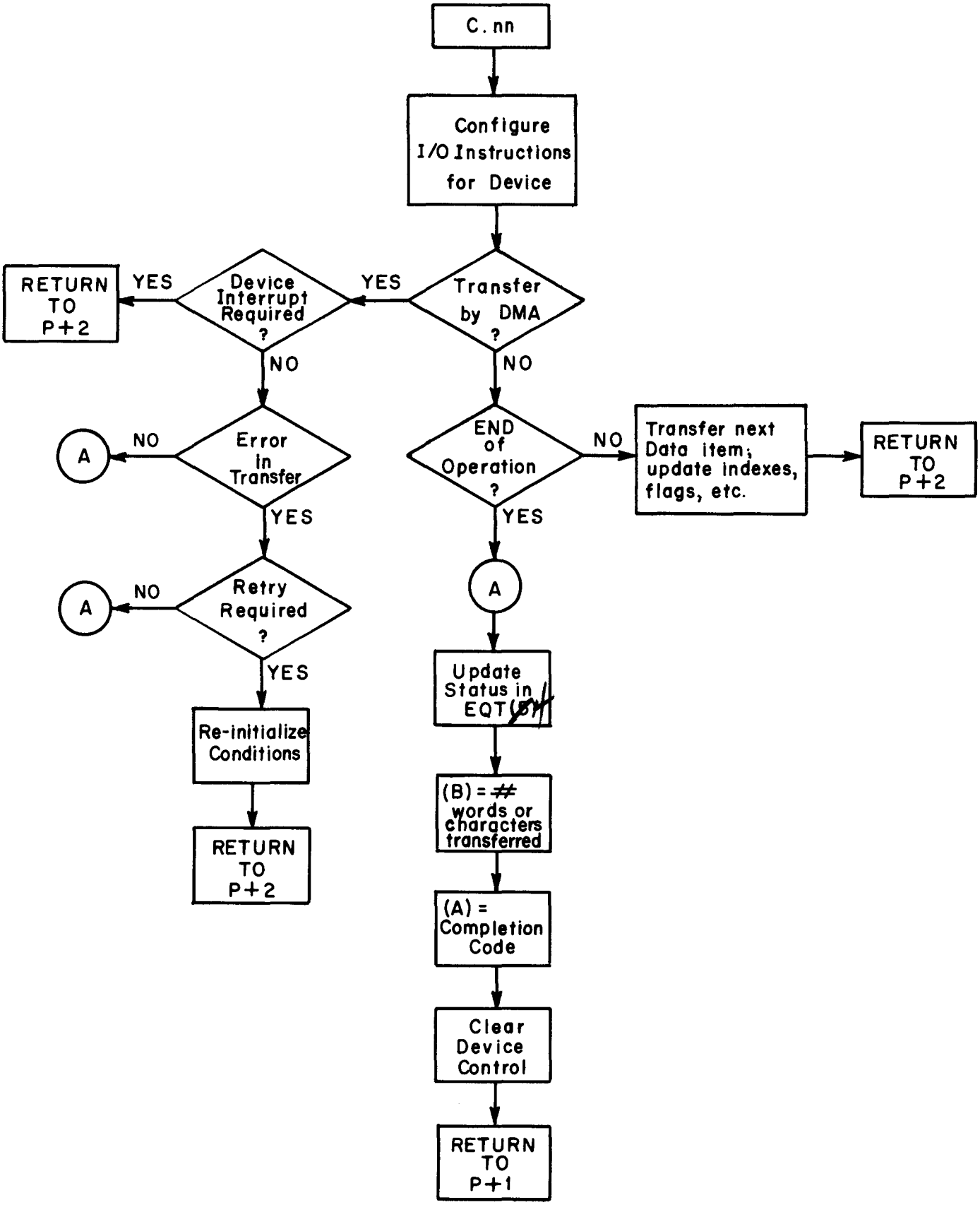

Figure 5-2. I/O Driver Completion Section

SECTION VI
INSTALLATION

The setup and operation of a Disc Operating System involves two essential steps and one optional step. DOS must be configured using the DOS Generator (DSGEN). It must be initiated from the disc by the core-resident Basic Binary Disc Loader (BBDL), and it may be dumped onto tape using the system dump (SDUMP) as protection against a disc failure.

This section describes the three routines -- DSGEN, BBDL, and SDUMP -- that are responsible for these processes.

DSGEN, THE DOS GENERATOR

DSGEN configures DOS to fit a particular user's core memory size, I/O equipment, and programming needs.

To accomplish this, DSGEN requests certain information from the user. DOS then accepts the relocatable program modules to be included in the system, determines where they belong in core or on the disc, relocates them into absolute format, and stores them on the disc. DSGEN also creates I/O tables by identifying each I/O device and its associated driver routine, and establishing procedures for interrupt processing on each channel.

DSGEN is an absolute program, loaded into core by the Basic Binary Disc Loader (BBDL) from paper tape. Since DSGEN is independent of the DOS which it generates, the I/O operations of DSGEN require special programs called SIO Drivers.

Using other standard Hewlett-Packard software, the user can create a magnetic tape or disc file of the relocatable program modules for quick and easy configuration.

DSGEN operates on the same minimum configuration as that required for a DOS.

Operating Procedures

The operation of DSGEN involves four phases:

- INITIALIZATION PHASE. DSGEN requests specifications for the DOS, including description of available disc space, memory, time base generator channel, and program input devices.
- PROGRAM INPUT PHASE. DSGEN reads in the relocatable programs provided with the system and created by the user.
- PARAMETER INPUT PHASE. Parameters to change EXEC modules or drivers from disc to core-resident may be entered. (The programs' NAM records are set for a minimum core system). DISCM, DVR3Ø (disc/drum driver) and DVRØØ (teleprinter driver) must be core-resident.
- DISC LOADING PHASE. DSGEN requests a specification of the base page linkage, and begins loading programs onto the disc. Systems programs (i.e., the modules of DOS), are loaded first, after which DSGEN requests information for the equipment table, device reference table (logical unit table), and interrupt table and proceeds to load the rest of the programs onto the disc.

To execute DSGEN and configure DOS, follow these steps:

- Turn on all equipment, set the system teleprinter to LINE, and disable the disc protect. (See Drum Memory Interface Kit Manual, 12610-9001, or Disc Memory Interface Kit Manual, 12606-9001.)
- Load the DSGEN tape into core using BBDL (the core-resident loader) and add the appropriate SIO Drivers. (If the relocatable programs are on a magnetic tape or disc file, the file must be created by the Prepare Tape System (PTS).) Refer to the *MAGNETIC TAPE SYSTEM* reference manual (5950-9292) for a description of PTS.
    a. Load the SIO Buffered Teleprinter Driver tape using the BBDL.
    b. Set the switch register to 2_8 and press LOAD ADDRESS.
    c. Set switch register bits 5-0 to the channel number of the device associated with the driver.

d. Press RUN.

e. Repeat these steps, if appropriate, for the Punched Tape Reader and a Magnetic Tape Unit or Disc Driver.

- Set the switch register to 100_8, press LOAD ADDRESS, then press RUN. DSGEN begins the initialization phase.

INITIALIZATION PHASE

During the initialization phase, DSGEN requests information necessary to begin generating the DOS. After each question is printed, the operator responds by giving the required information. The following dialogue is typical. (The operator responses are only examples; actual responses should be appropriate to the particular system being generated.)

DSGEN requests the octal channel number (higher priority channel if multi-card) of the system disc or drum unit.....SYS DISC CHNL?

Operator responds.....*20*

DSGEN requests the smallest number of sectors (decimal) per track on the discs or drums in the system. (Disc units are usually 90; drums, 128.).....#SECTORS/TRACK?

Operator responds.....*90*

DSGEN requests the number of tracks (decimal) on the system disc or drum.....SYS DISC SIZE?

Operator responds.....*32*

DSGEN requests the number of hardware protected tracks in decimal.....#PROTECTED TRACKS

Operator responds.....*8*

DSGEN requests decimal number of first track on disc available to the system. (All system tracks must be contiguous, i.e., DOS cannot be generated with intervening faulty tracks.).....FIRST SYSTEM TRACK?

Operator responds.....Ø

DSGEN requests the starting system sector number in decimal.....FIRST SYSTEM SECTOR?

Operator responds. (Operator must record this response and the previous one if he plans to save the system using SDUMP.).....3

The system cannot start on track Ø sector Ø since sectors Ø & 1 are used for the disc loaders.

DSGEN requests the I/O channel of the auxiliary mass storage device in octal (Ø if none).....AUX DISC CHNL?

Operator responds.....1

If the previous response is not zero, DSGEN requests the number of tracks on the auxiliary disc or drum.....AUX DISC SIZE?

Operator responds.....22

DSGEN requests the I/O channel of the Time Base Generator (octal).....TIME BASE GEN CHNL?

Operator responds.....1Ø

DSGEN requests the last word of available core memory, in octal.....LWA MEM?

Operator responds.....37677

DSGEN requests the type of input unit for relocatable program modules.....PRGM INPT?

Operator responds with PT (for paper tape), TY (for teleprinter), MT (for magnetic tape), or DF (for disc file).....PT

DSGEN requests the type of input unit for relocatable library programs.....LIBR INPT?

Operator responds with PT, TY, MT, or DF.....*MT*

DSGEN requests the type of input unit for parameters, describing the relocatable programs.....PRAM INPT?

Operator responds with PT or TY.....*TY*

When DSGEN finishes the initialization phase, the computer halts.

PROGRAM INPUT PHASE

During the program input phase, DSGEN accepts relocatable programs from the Program Input Unit and Library Input Unit specified during the initialization phase. The operator selects the input device by setting switch register bits Ø-1 ($ØØ_2$ for the Program Input Unit, or $1Ø_2$ for the Library Input Unit), and places the programs in the input device. Main programs must enter prior to their segments.

The suggested order of tape input is:

DOS CORE-RESIDENT SYSTEM (DISCM)
DOS DISC-RESIDENT EXEC MODULES ($EXØ1 THRU $EX16)
DOS I/O DRIVERS (DVRØØ, DVRØ1,...ETC)
DOS JOB PROCESSOR/FILE MANAGER (JOBPR)
DOS RELOCATING LOADER (LOADR)
DOS ASSEMBLER (MAIN CONTROL, SEGMENTD, SEGMENT1,....)
DOS FORTRAN (MAIN CONTROL, PASS 1,...)
DOS RELOCATABLE LIBRARY
Any user programs to be made a permanent part of DOS.

The operator presses RUN. When entering paper tape, the message "*EOT" is printed whenever an end-of-tape occurs. The computer halts.

At this point, the operator has several options: additional programs can be input from the same device by repeating the steps above; input can be switched to the other input device (by setting the switch register bits to $\emptyset\emptyset_2$ or $1\emptyset_2$).

To terminate the program input phase, the user must set switch register bits to $\emptyset 1_2$, and press RUN. If there are no undefined externals, this message is printed on the system teleprinter:

NO UNDEF EXTS

If there are undefined externals, the following message is printed:

UNDEF EXTS

The externals are listed one per line and the computer halts. External references are satisfied by loading more programs. The user must set switch register bits to $\emptyset\emptyset_2$ (for Program Input Unit) or $1\emptyset_2$ for the Library Input Unit) and press RUN.

PARAMETER INPUT PHASE

During the parameter input phase, the operator can change some modules from disc to core-resident.

Each parameter record is of this general form:

name,type

where *name* is the name of the program

type is the program type code;

- Ø - System core-resident
- 1 - System disc-resident exec modules
- 3 - User disc-resident main

4 - Disc-resident I/O driver
5 - User segment
6,7 - Library
>7 - Program deleted from the system

EXEC modules and drivers that are often used may be changed from disc to core-resident. The functions of the EXEC modules are:

Module Name		Function
$EXØ1	-	Disc Work Tracks Status
$EXØ2	-	Disc Work Track Limits
$EXØ3	-	Program Completion
$EXØ4	-	Program Suspension
$EXØ5	-	Program Segment Load
$EXØ6	-	User File Name Search
$EXØ7	-	Current Time Processor
$EXØ8	-	Real-Time Disc Allocation. (See Appendix D.)
$EXØ9	-	Execution Time :EQ Processor
$EX1Ø	-	Load and Execute Program
$EX11	-	System File Name Search
$EX12	-	System Startup
$EX13	-	Error Message Processor
$EX14	-	Execution Time, :UP, :DN, :LU Processor
$EX15	-	Abort and Post Mortem Dump
$EX16	-	:GO Parameter Processor

When EXEC modules are made core-resident, certain associated library subroutines must also be changed to be core-resident. Several EXEC modules use $ADDR:

$EXØ1
$EXØ2
$EXØ6
$EXØ7
$EXØ8

The following EXEC modules use $SRCH:

$EXØ5
$EXØ6
$EX11

These EXEC modules use ASCII:

$EXØ9
$EX13
$EX14
$EX15

To end the parameter input phase and continue on to the disc loading phase, the operator enters "/E" instead of a parameter record. DSGEN then asks two questions before entering the disc loading phase.

DISC LOADING PHASE

DSGEN requests the estimated number of system linkages required in base page.....#SYSTEM LINKS?

Operator responds with a decimal number. (The more modules that are core-resident the more links are needed, 1ØØ should be the minimum response.).....2ØØ

DSGEN requests the estimated number of user linkages required in base page.....#USER LINKS?

Operator responds with a decimal number. (Since FORTRAN requires approximately 4ØØ ~~5ØØ~~ linkages, ~~3ØØ~~ 4ØØ should be the minimum number entered.).....~~5ØØ~~ 4ØØ

Figure 6-1 shows the relative location of the various core areas. Loading of the absolute, resident supervisor begins after the establishment of the user and system linkage areas. As each program is loaded, DSGEN prints a

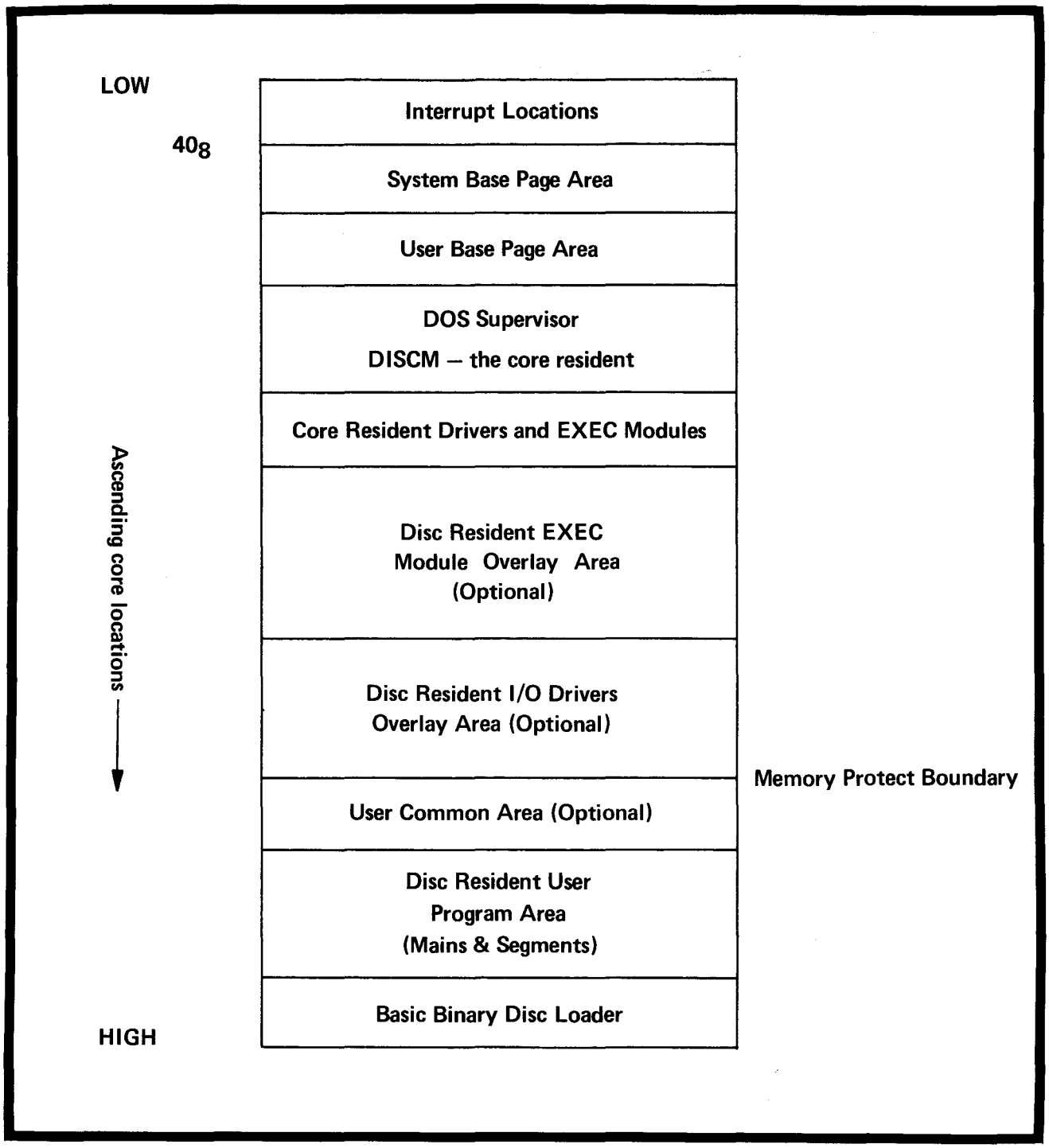

Figure 6-1. Core Allocations in DOS

memory map giving the starting locations and, if switch register bit 15 is up, the entry points for all main programs and subroutines. (Subroutines are indented two spaces, and entry point addresses are preceded by an asterisk.)

Next, DSGEN generates the three I/O tables: equipment table, device reference table (logical unit table), and the interrupt table.

DSGEN requests the equipment table entries.....EQUIPMENT TABLE ENTRY?

Operator responds with a series of one line EQT entries, which are assigned EQT numbers sequentially from one as they are entered. The EQT entry relates the EQT number to an I/O channel and driver, in this format.....*n1,DVRnn [,D][,R][,U]*

where *n1* is the I/O channel (lower number if multi-board),
DVRnn is the driver name (*nn* is the equipment type code).
D, if present, means DMA channel required,
R, if present, means driver is core-resident,
U, is the physical sub-channel number.

Operator terminates the equipment table entries by typing...../E

DSGEN requests the logical unit assignments for the device reference table.....DEVICE REFERENCE TABLE?

For each logical unit number, DSGEN prints.....*n=EQT#?*

Operator responds with an EQT entry number (*m*) appropriate to the standard definition of *n*. Numbers above 9 may be assigned any EQT entry desired.....*m*

Operator terminates entry by typing...../E

DSGEN requests the interrupt table entries.....INTERRUPT TABLE

Operator responds with an entry for each I/O location which may interrupt, in ascending order, and in this format.....*n1, n2*

FOR 3030 MAG. TAPE THE ENTRY IN THE INTERRUPT TABLE SHOULD BE THE LOCATION OF MAG. TAPE 2 BOARD

where *n1* is the octal channel number between 10_8 and 37_8 inclusive (must be entered in ascending order), and

n2 is a decimal EQT entry number.

Operator terminates entry by typing...../E

Following the completion of the I/O tables, DSGEN loads the disc-resident executive modules (if any), and the disc-resident I/O drivers (if any).

DSGEN reports the last address (plus 1) of the supervisor.....LWA SYS *xxxxx*

DSGEN requests the first word address of the user program area.....FWA USER?

Operator responds with an octal address greater than or equal to *xxxxx*. (This option is provided so that user programs can start on a page boundary, if desired).....*nnnnn*

DSGEN proceeds to load all user main programs and segments onto the disc with memory map listings as described for system programs.

When system generation is complete, DSGEN prints.....SYSTEM STORED ON DISC

DSGEN then reports the last track used in bits 15 through 8 of the A-Register, and the last sector used in bits 7 through Ø of the A-Register. (These must be recorded if one plans to use SDUMP to save the system.) At this point, the disc protect switch must be enabled to protect the system on the disc. (See Drum Memory Interface Kit Manual, 12610-9001, or Disc Memory Interface Kit Manual, 12606-9001.)

Restart

During any of the phases, DSGEN can restart that phase if any error occurs. The operator sets the switch register equal to 100_8, and presses LOAD ADDRESS and RUN.

In addition, the parameter input phase can be re-entered at $4ØØØ_8$, and the disc loading phase at $6ØØØ_8$.

Error Messages

The following messages may be printed on the teleprinter during execution of DSGEN:

Message	Meaning	Action
Messages During Initialization and Input Phase		
ERRØ1	Invalid response to initialization request.	Request is repeated. Enter valid reply.
ERRØ2	Checksum error on program input.	Computer halts; reposition tape to beginning of record and press RUN to reread.
ERRØ3	Record out of sequence.	Same as ERRØ2.
ERRØ4	Illegal record type.	Same as ERRØ2.
ERRØ5 *name*	Duplicate entry point.	Revise program by reloading the entry points (the current entry point replaces the previous entry point).
ERRØ6	Invalid base page length (must be zero).	Base page area is ignored, but memory protect error will occur if program is executed.
ERRØ7	Program name or entry point table overflow of available memory.	Irrecoverable error. Revise or delete programs.
ERRØ8 *name*	Duplicate program name.	The current program replaces the previous program.

Messages During the Parameter Phase

Message	Meaning	Action
ERRØ9	Parameter name error (no such program).	Enter valid parameter statement.
ERR1Ø	Parameter type error.	Same as ERRØ9.

General Messages

ERR13	User segment precedes user main program.	Irrecoverable.
ERR15	More than 63 subprograms called by a main program.	Revise main program (subsequent calls to subprograms are ignored).
ERR16	Base page linkage overflow.	Diagnostic printed for each word required. Revise order and composition of program loading to reduce linkage requirements.
ERR17	Current disc address exceeds number of available tracks.	Irrecoverable error.
ERR18	Memory overflow (absolute code exceeds LWA memory).	Diagnostic printed for each word required (absolute code is generated beyond LWA). Revise program.
ERR19	Program overlay (current word of absolute code has identical location to previous).	Current word is ignored (the address is printed).
ERR2Ø	Binary DBL record overflow of internal table.	Records overlay previous DBL records (diagnostic printed for each overflow record). Revise program.

Message	Meaning	Action
ERR21	Module containing entry point $CIC not loaded.	Irrecoverable error.
ERR22	Read parity/decode disc error. A-register bits 8-14 show track number; bits Ø-7 show sector number.	After ten attempts to read or write the disc sector, the computer halts. To try ten more times, press RUN.
ERR23	EQT not entered for disc-resident I/O module.	Restart at $4ØØØ_8$.

Messages During I/O Table Entry

ERR24	Invalid channel number.	Enter valid EQT statement.
ERR25	Invalid driver name or no driver entry points.	Same as ERR24.
ERR26	Invalid or duplicate D,R,U operands.	Same as ERR24.
ERR27	Invalid logical unit no.	Enter valid DRT statement.
ERR28	Invalid channel number.	Enter valid INT statement.
ERR29	Channel number decreasing.	Same as ERR28.
ERR31	Invalid EQT number.	Same as ERR28.
ERR35	Base page interrupt locations overflow into linkage area.	Restart Disc Loading Phase.
ERR36	Invalid number of characters in final operand.	Same as ERR28.

DOS INITIATION FROM THE DISC

The Basic Binary Disc Loader (BBDL), a modified version of the standard Basic Binary Loader, resides in the highest-protected 64 words of core and loads either absolute format paper tapes or disc-based systems, such as DOS.

Loading DOS

- The operator sets the switch register equal to $Ø7776Ø_8$, and presses LOAD ADDRESS. He then sets the loader switch to ENABLED, presses PRESET, and presses RUN.
- When the computer halts with $1Ø2Ø77_8$ in the T-register, the operator sets the loader switch to PROTECTED, sets switch register bit Ø to 1, and presses RUN. A halt with $1Ø2Ø11_8$ in the T-register means that a checksum error occurred. $1Ø2Ø55_8$ means an illegal address. If the BBDL itself is destroyed, it can be replaced through the switch register using the octal listing in Appendix A.
- When DOS is loaded, it types:

```
INPUT FR = FRESH; CO = CONTINUATION
```

The operator enters FR if no user files are currently saved on the disc, or CO if user files currently exist or have been loaded by SDUMP.

DOS then prints the following message until the operator types a valid DATE directive (see Section II):

```
INPUT :DATE,XXXXXXXXXX,H,M
```

Following the DATE directive, the only valid directives are TRACKS, BATCH and JOB. All others are ignored until a JOB directive is given. If this is a fresh start (FR) and there are known faulty tracks in the system (i.e. user and work areas of the disc), they should be set at this time using the TRACKS directive as shown below:

$$:\text{TRACKS},b_1,b_2,\ldots.b_n$$

where b_1 through b_n are the faulty tracks.

CREATING A BACK-UP COPY

SDUMP, the System Dump, is an independent utility program that can create back-up copies of disc-based systems on punched tape or magnetic tape. The back-up copy can later be reloaded onto the disc by SDUMP.

Because it is an independent program like DSGEN, SDUMP requires the independent SIO Drivers. For paper tape storage, the SIO Teleprinter Driver, SIO Paper Tape Reader Driver, and SIO Paper Tape Punch Driver are required. For magnetic tape storage, the SIO Teleprinter Driver and SIO Magnetic Tape Unit Driver are required. The operator loads the SDUMP tape and SIO Driver tapes as described for DSGEN. The magnetic tape driver must be loaded after SDUMP.

After loading SDUMP, execution begins at 100_8. SDUMP prints out a request guide on the teleprinter:

```
DUMP = D,T[-S][,T[-S]] ([] = OPTIONAL)
VERIFY = V
LOAD = L
TERMINATE = T
```

SDUMP requests the lower-number channel or the
disc in octal.....DISC CHNL?

Then SDUMP types.....COMMAND

The operator replies with V,L,T,
or D (followed by parameters).....*L*

D for dumping requires a set of parameters specifying the first and last tracks, inclusive, to be dumped. The values are octal. The last track and sector are reported by DSGEN after creating DOS and should be recorded then. If output is to paper tape, trailer and leader blank tape is produced, and two tracks are dumped at a time. If output is to magnetic tape, the entire information is dumped, followed by an End-Of-File which is written over by subsequent dumps.

D, 0, 57

V for verifying, involves placing the dump in the input device, reading it in, and checking each record against the contents of the disc. Comparison errors are reported. If magnetic tape is verified, only one file is checked.

L for loading causes the dumped information to be loaded back onto the disc. The information is verified after it is output to the disc, and comparison errors are reported.

An illegal command causes the message:

STATEMENT ERROR

An error in specifying the disc channel, causes the message:

PARAM ERROR: NON-NUMERIC OR NOT-OCTAL

If the magnetic tape is used, a rewind is issued during initialization, before and after a verify or load operation, and rewind/standby after T for termination.

Error Messages

The following messages may be printed on the teleprinter by SDUMP:

Statement	Action
STATEMENT ERROR	Retype input statement in correct format.
EOT	The end of the input tape being read has been reached; either load the next tape or go on to the next phase.
CHANGE INPUT TAPE, HIT RUN	Two full tracks have been dumped onto paper tape; perform the requested action.
TURN OFF DISC PROTECT, HIT RUN	Set the Disc Track Protect Switch off, then press RUN.
DISC INPUT ERROR	Disc Error Diagnostic, for a Parity, Decode or Abort status after 10 retrys. Input sequence repeated on restart.

Statement	Action
DISC WRITE ABORT	Disc Error Diagnostic, for an Abort status after a write attempt. Sequence is repeated if restarted.
TRACK *nnn* (8) SECTOR *mmm* (8)	Identification information for the Disc AND TAPE ERROR DIAGNOSTICS ARE DESCRIBED AS FOLLOWS: ~~Error Diagnostic messages described above.~~ *nnn* is the octal track number and *mmm* is the octal sector number where the error occurred.
TAPE/DISC VERIFY ERROR	Disc and tape records do not agree. Disc record is rewritten on restart.
TAPE CHECKSUM ERROR	The checksum in the tape record does not match the sum computed by SDUMP. Current record is ignored if restarted.
MT ERROR - READ PARITY MT ERROR - EOT, RESTART	Magnetic Tape Errors. Error recovery procedures are completed by driver. Restart to retry sequence.

Saving System and/or User Files

The system operator can save the system and/or user files on the disc for later use by dumping them on tape with SDUMP. This is desirable when there are more than two disc-based software systems to be used with one computer (e.g. DOS, RTE System, TSB System) or when there is a need to protect certain user files (not hardware-protected by DOS).

The operator first determines which tracks and sectors to dump. For the system file, track Ø, sectors Ø, 1 and 2 must always be saved. In the operation of DSGEN, the operator should save the responses to the following queries:

FIRST SYSTEM TRACK?
FIRST SYSTEM SECTOR?

If the system area is to be dumped, start dumping with these response values after converting them from decimal to octal.

When DSGEN is completed, the operator should note the value in the A-register; this value indicates the last track and sector of the system.

To dump only the system area of the disc, the user enters the following two SDUMP commands:

D,Ø-Ø,Ø-2
D,*FT-FS,LT-LS*

Where *FT* is the first track,
FS is the first sector,
LT is the last track, and
LS is the last sector.

The first track of the user file is that immediately following the hardware-protected area of the disc. To obtain the value of the last track, the operator uses a TRACKS directive (see Section II). The last track of the user area immediately precedes the first work area track reported by TRACKS.

Since the track and sector numbers start with zero, if there are 16 protected tracks, the user file starts with $16_{1Ø}$ ($2Ø_{8}$). To save the user file, the following SDUMP command is entered:

D,*UT*-Ø,*LU-S*

where *UT* is the first user track,
LU is the last user track, and
S is the number of sectors/track in octal minus one.

To dump and verify the entire system and user areas, the operator would use the following SDUMP commands:

COMMAND:
D,Ø-Ø,Ø-2
COMMAND:
D,FT-FS,LT-LS

```
COMMAND:
D,UT-Ø,LU-S
COMMAND:
V
COMMAND:
T
```

To reload this dump, the operator enters L in response to COMMAND:. Then DOS is initiated from the disc using BBDL and CO is entered for a continuation start (to preserve the user files).

Entering FR for a fresh start would prevent access to the user file area. Therefore, FR is used when only the system area is dumped and restored. When only the user area is saved, a CO start must be used when it is restored. However, the user area can only be reloaded on a disc containing the system area from which it was dumped.

APPENDIX A
TABLES

Appendix A contains several useful tables and figures.

DOS BASE PAGE CONSTANTS

LOCATION	TYPE	VALUE
4Ø	DEC	-64
41	DEC	-1Ø
42	DEC	-9
43	DEC	-8
44	DEC	-7
45	DEC	-6
46	DEC	-5
47	DEC	-4
5Ø	DEC	-3
51	DEC	-2
52	DEC	-1
53	DEC	Ø
54	DEC	1
55	DEC	2
56	DEC	3
57	DEC	4
6Ø	DEC	5
61	DEC	6
62	DEC	7
63	DEC	8
64	DEC	9
65	DEC	1Ø
66	DEC	17
67	DEC	64
7Ø	OCT	17
71	OCT	37
72	OCT	77

LOCATION	TYPE	VALUE
73	OCT	177
74	OCT	377
75	OCT	1774ØØ
76	OCT	3777
77	OCT	1777ØØ

DOS BASE PAGE SYSTEM COMMUNICATION AREA

LOCATION	NAME	CONTENTS
1ØØ	UMLWA	Last word address of USER useable memory
1Ø1	JBINS	Starting track/sector of current job binary area
1Ø2	JBINC	Current track/sector of current job binary area
1Ø3	TBG	Time base generator I/O channel address.
1Ø4-5	CLOCK	Current time
1Ø6-7	CLEX	Execution time
11Ø	CXMX	Maximum execution time
111	BATCH	Logical unit # of batch input device
112	SYSTY	Logical unit # of system teletype
113	DUMPS	Abort/Post Mortem dump flags
114	SYSDR	System directory track/sector address
115	SYSBF	System buffer track address; User directory sector #
116	SECTR	Number of sectors/disc track
117	EQTAB	First word address of Equipment Table
12Ø	EQT#	Number of Equipment table entries
121	LUTAB	First word address of Logical Unit table
122	LUT#	Number of Logical Unit table entries
123	JBUF	Job input buffer address
124	JFILS	Source file starting track/sector address
125	JFILC	Source file current track/sector address
126-4Ø	RONBF	System buffer
141-53	EXPG	Currently executing program directory entry.

LOCATION	NAME	CONTENTS
154-57	DISCO	Disc I/O channel/track number
16Ø	NXTTS	Next available user file track/sector address
161-2ØØ	TTABL	Track status table
2Ø1	INTAB	First word address of interrupt table
2Ø2	INT #	Number of interrupt entries
2Ø3	EQT1	EQT1-EQT17 are addresses of current Equipment table entry
2Ø4	EQT2	
2Ø5	EQT3	
2Ø6	EQT4	
⋮	⋮	
223	EQT17	
224	RQCNT	Current number of request parameters
225	RQRTN	Current request return address
226	RQP1	RQP1-RQP8 are current request parameter addresses
⋮	⋮	
235	RQP8	
236	NABRT	Illegal request code abort/no abort option
237	XA	A register contents at time of interrupt
24Ø	XB	B register contents at time of interrupt
241	XEO	E and O registers contents at time of interrupt
242	XSUSP	Point of suspension at time of interrupt
243	EXLOC	Address of Exec module table
244	EX#	Number of Exec module table entries
245	EXMOD	Current resident Exec module address
246-7	EXMAN	Exec module low and high main core addresses
25Ø-1	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

LOCATION	NAME	CONTENTS
254	UMFWA	First word address of user main area
255	UBFWA	First word address of user base area
256	UBLWA	Last word address of user base area
257	CHAN	Current DMA channel number
26Ø	OPATN	Operator/Keyboard attention flag
261	OPFLG	Operator communication flag
262	SWAP	Job Processor resident flag
263-4	JOBPM	Job Processor disc address/number of words in main
265	JOBPB	Job Processor base page number of words
266	TBSYG	Track/sector address of system track table
267	RTRK	Real time simulation track number
27Ø-367	$BUF	System buffer for disc sector
37Ø-413	DBUFR	System disc triplet parameter buffer
414	$GOPT	Point of suspension continuation address
415	$IDCD	Input request code check
416-7	$MDBF	Exec module data buffer
42Ø-6	TEMP	System temporary
427	TEMPØ	System temporary
43Ø	TEMP1	System temporary
431	TEMP2	System temporary
432	TEMP3	System temporary
433	TEMP4	System temporary
434	TEMP5	System temporary
435	MSECT	Negative of number of sectors/track
436	VADR	Address of instruction causing memory protect violation
437	IODMD	Current resident I/O driver module address
44Ø	RCODE	Current positive request code
441	SXA	Operator attention restore A register value
442	SXB	Operator attention restore B register value
443	SXEO	Operator attention restore E and O value
444	SXSUS	Operator attention return address
445	SEQT1	Operator attention restore EQT table address
446	DSCLB	Track/sector address of relocatable library

LOCATION	NAME	CONTENTS
447	DSCL#	Number of relocatable library routines
450-1	CHARC	System temporary

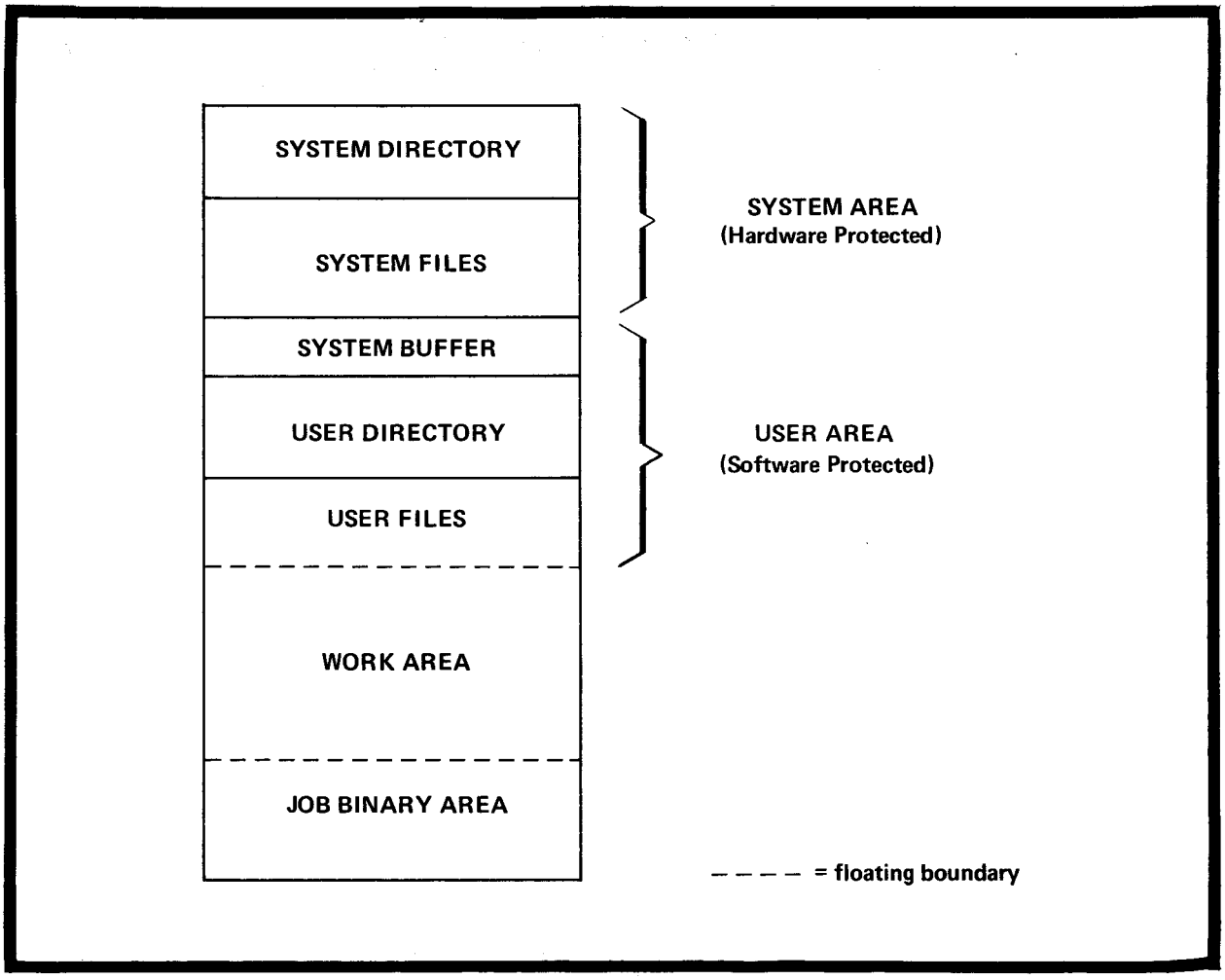

Figure A-1. General Disc Layout

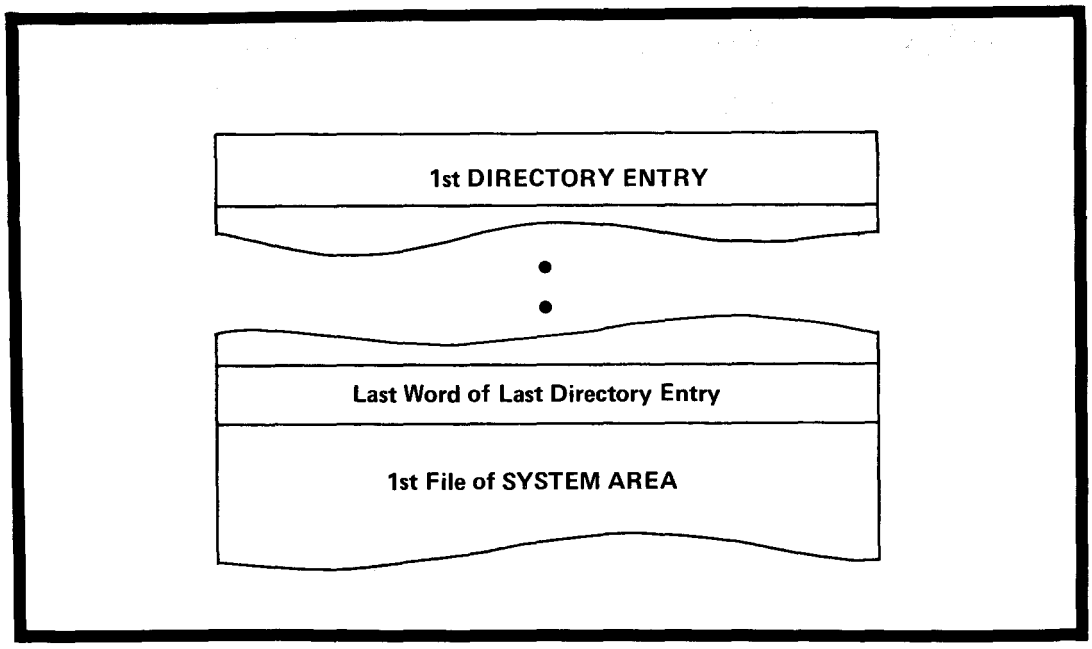

Figure A-2. System Directory Format

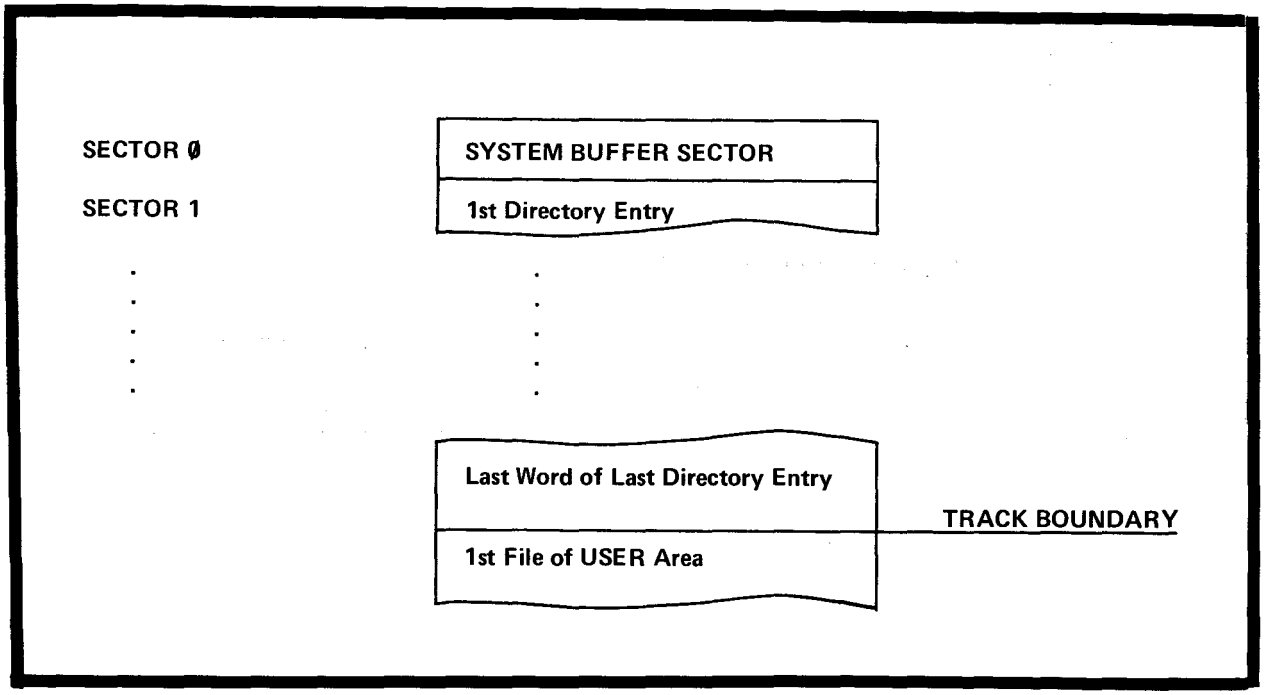

Figure A-3. User Directory Format

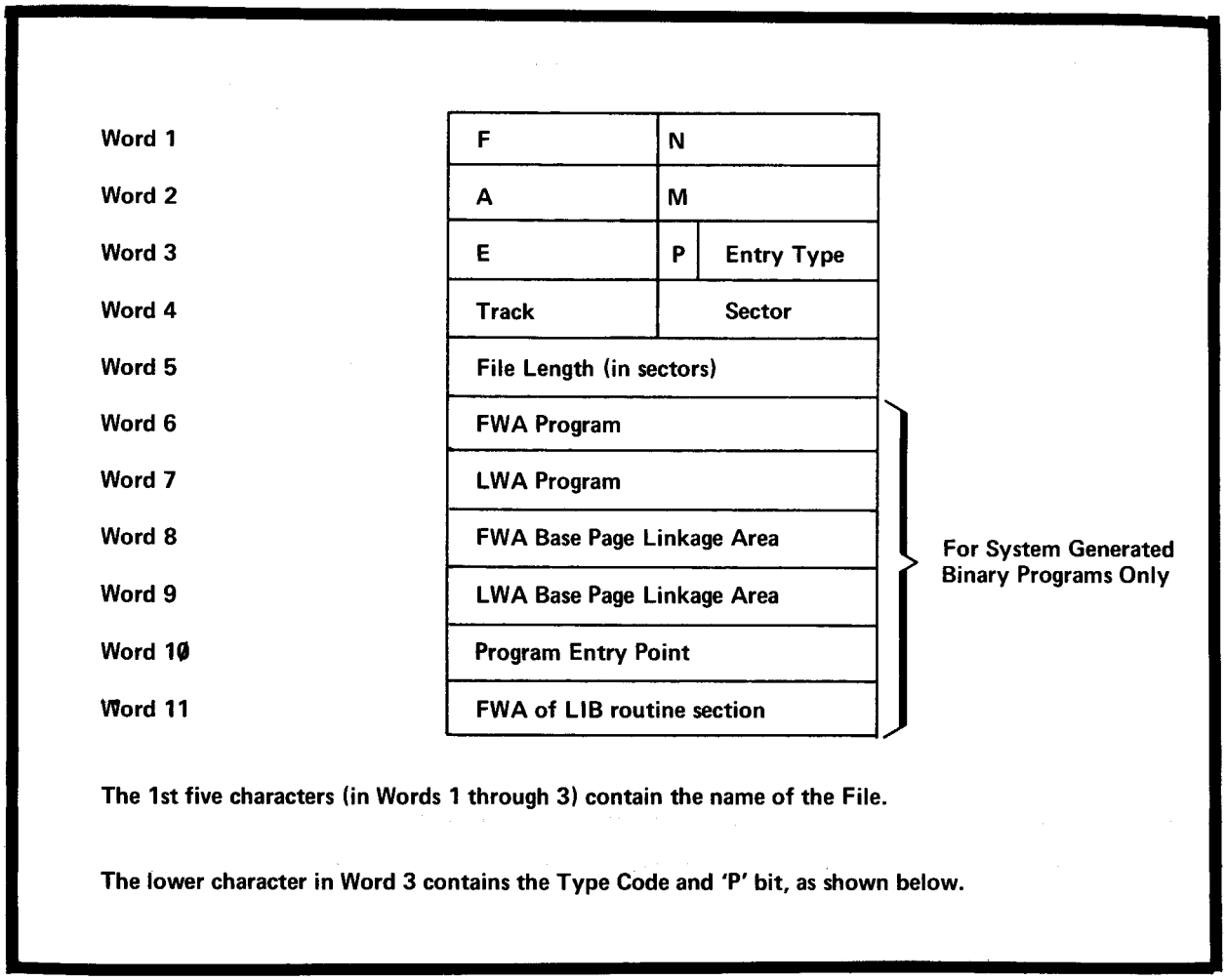

Figure A-4. Directory Entry Format

TYPE	FILE
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_8	Relocatable Binary
11_8	ASCII Source Statements
12_8	Binary Data
13_8	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.

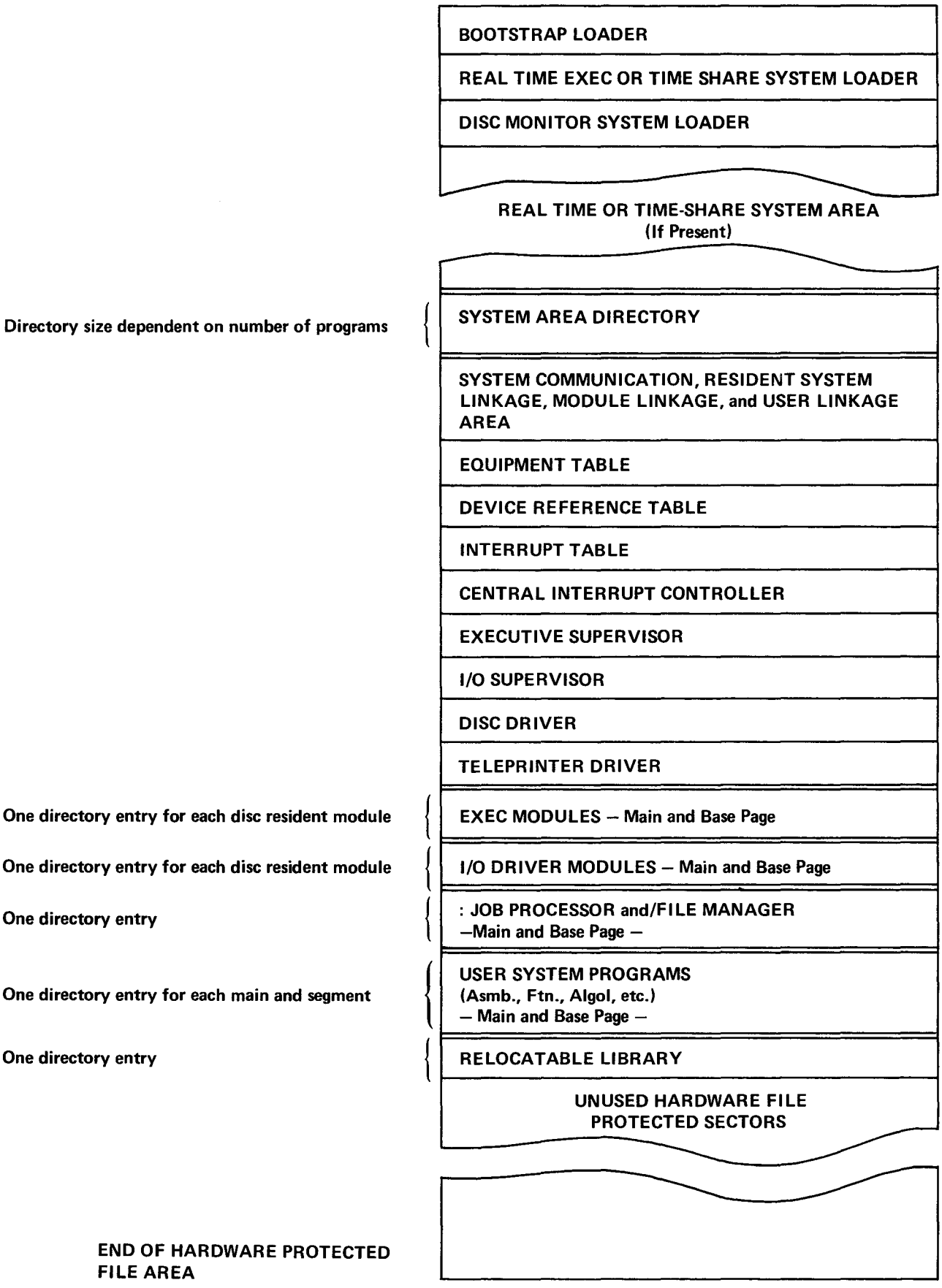

Figure A-5. Disc Allocation in System Area

BASIC BINARY DISC LOADER (BBDL)

Paper Tape Loading

The operator places the paper tape in the teleprinter reader (or photoreader, if available). He sets the switch register equal to $\emptyset 777\emptyset\emptyset_8$, and presses LOAD ADDRESS. He then sets the loader switch to ENABLED and presses RUN. After BBDL reads the tape, the operator sets the loader switch to PROTECTED.

BBDL Listing

Figure A-6 is an octal listing of the Basic Binary Disc Loader that resides in the protected, highest 64 words of core. If the loader is destroyed in core, it can be replaced through the switch register using this listing. The operator simply replaces symbolic items with the value appropriate to the configuration.

A	B 0	1	2	3	4	5	6	7
0*m*7700:	107700	002401	063726	006700	017742	007306	027713	002006
0*m*7710:	027703	102077	027700	077754	017742	017742	074000	077757
0*m*7720:	067757	047755	002040	027740	017742	040001	177757	037757
0*m*7730:	000040	037754	027720	017742	054000	027702	102011	027700
0*m*7740:	102055	027700	000000	006600	1037*cc*	1023*cc*	027745	1064*cc*
0*m*7750:	002041	127742	005767	027744	000000	1*z*0100	0200*nn*	000000
0*m*7760:	107700	063756	102606	002700	1026*qq*	001500	102602	063777
0*m*7770:	102702	102602	103706	1027*nn*	067776	074077	024077	177700

Legend: A + B = Memory Address

m = 1 for 8K, 3 for 16K, 5 for 24K, 7 for 32K memory

nn = first disc channel

qq = second disc channel

cc = photoreader or Teleprinter address

z = 6 for 8K, 4 for 16K, 2 for 24K, Ø for 32K memory

Figure A-6 BBDL LISTING

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
       :                    } Source Program
      END ENTER
:STORE,R,AFILE
:JOB,NEXTJOB
```

LOAD AND EXECUTE A RELOCATABLE FILE

```
:JOB,LOADE
:PROG,LOADR,2
AFILE
/E
:STORE,P,TEST
:RUN,TEST
10
23
 :                          } Data
 :
51
:JOB,NEXTJOB
```

STORE, EDIT, COMPILE, LOAD AND RUN A PROGRAM

```
:JOB,EVERY
:STORE,S,SOURC,5
FTN,B,L
      PROGRAM ZOOM
      DIM  I(10)
      :
      END$
::
:LIST,S,6,SOURC
:EDIT,SOURC,5
/I,2
:
/E
:JFILE,SOURC
:PROG,FTN,2,6,4,56,99
:PROG,LOADR
:RUN,ZOOM
123.62
 :
ØØØØ1
:RUN,ZOOM
321.5
 :
Ø.56
:JOB,NEXTJCB
```

Source Program

Edit List

Data for first run

Data for second run

APPENDIX C
SAMPLE DSGEN LISTINGS

A MINIMUM CORE DOS

```
SYS DISC CHNL?
21
# SECTORS/TRACK?
9Ø
SYS DISC SIZE?
64
#PROTECTED TRACKS?
24
FIRST SYSTEM TRACK?
Ø
FIRST SYSTEM SECTOR?
3
AUX DISC CHNL?
Ø
TIME BASE GEN CHNL?
13
LWA MEM?
17677
PRGM INPT?
PT
LIBR INPT?
PT
PRAM INPT?
TY
*EOT
*EOT
*EOT
```

```
NO UNDEF EXTS
ENTER PROG PARAMETERS
/E
# SYSTEM LINKS?
326
# USER LINKS?
4ØØ
SYSTEM
DISCM     Ø2ØØØ
DVR3Ø     Ø5Ø75
DVRØØ     Ø534Ø
*EQUIPMENT TABLE ENTRY
12,DVR15
14,DVR12
15,DVRØ1
16,DVRØ2
17,DVRØØ,R
2Ø,DVRØØ,R
21,DVR3Ø,D,R
/E
*DEVICE REFERENCE TABLE
 1 = EQT #?
5
 2 = EQT #?
7
 3 = EQT #?
Ø
 4 = EQT #?
4
 5 = EQT #?
1
 6 = EQT #?
```

```
2
 7 = EQT #?
3
 8 = EQT #?
6
 9 = EQT #?
/E
* INTERRUPT TABLE
12,1
14,2
15,3
16,4
17,5
2Ø,6
/E
EXEC SUPERVISOR MODULES
$EXØ1       Ø6365
  $ADDR     Ø6461
$EXØ2       Ø6365
  $ADDR     Ø6433
$EXØ3       Ø6365
$EXØ4       Ø6365
$EXØ5       Ø6365
  $SRCH     Ø6524
$EXØ6       Ø6365
  $SRCH     Ø6422
  $ADDR     Ø6554
$EXØ7       Ø6365
  $ADDR     Ø6451
$EXØ8       Ø6365
  $ADDR     Ø6536
```

```
$EXØ9      Ø6365
  ASCII    Ø6616
$EX1Ø      Ø6365
$EX11      Ø6365
  $SRCH    Ø6463
$EX12      Ø6365
$EX13      Ø6365
  ASCII    Ø6752
$EX14      Ø6365
  ASCII    Ø6627
$EX15      Ø6365
  ASCII    Ø667Ø
$EX16      Ø6365

I/O DRIVER MODULES

DVR12      Ø7Ø27
DVR11      Ø7Ø27
DVRØ2      Ø7Ø27
DVRØ1      Ø7Ø27

LWA SYA    Ø7447

FWA USER?
Ø7447
USER SYSTEM PROGRAMS

LOADR      Ø7447
JOBPR      Ø7447

*SYSTEM STORED ON DISC
```

A MAXIMUM CORE DOS

```
SYS DISC CHNL?
21
# SECTORS/TRACK?
9Ø
SYS DISC SIZE?
64
#PROTECTED TRACKS?
16
FIRST SYSTEM TRACK?
Ø
FIRST SYSTEM SECTOR?
3
AUX DISC CHNL?
Ø
TIME BASE GEN CHNL?
13
LWA MEM?
37677
PRGM INPT?
PT
LIBR INPT?
PT
PRAM INPT?
TY
*EOT
*EOT
*EOT
*EOT
*EOT
*EOT
```

```
*EOT
*EOT
*EOT
*EOT
*EOT
*EOT

NO UNDEF EXTS

ENTER PROG PARAMETERS

$EXØ1,Ø
$EXØ2,Ø
$EXØ3,Ø
$EXØ4,Ø
$EXØ5,Ø
$EXØ6,Ø
$EXØ7,Ø
$EXØ8,Ø
$EXØ9,Ø
$EX1Ø,Ø
$EX11,Ø
$EX12,Ø
$EX13,Ø
$EX14,Ø
$EX15,Ø
$EX16,Ø
$ADDR,Ø
$SRCH,Ø
ASCII,Ø
DVR15,Ø
DVR12,Ø
DVRØ1,Ø
DVRØ2,Ø
/E

# SYSTEM LINKS?
326
```

```
# USER LINKS?
4ØØ
SYSTEM
DISCM     Ø2ØØØ
$EXØ1     Ø5Ø77
$EXØ2     Ø5173
$EXØ3     Ø5241
$EXØ4     Ø5273
$EXØ5     Ø5331
$EXØ6     Ø547Ø
$EXØ7     Ø5525
$EXØ8     Ø5611
$EXØ9     Ø5762
$EX1Ø     Ø6213
$EX11     Ø6254
$EX12     Ø6352
$EX13     Ø6561
$EX14     Ø7146
$EX15     Ø741Ø
$EX16     Ø7713
DVR3Ø     1ØØ27
DVR12     1Ø272
DVR15     1Ø667
ASCII     113Ø7
$SRCH     11364
$ADDR     11516
DVRØØ     11533
DVRØ2     123Ø6
DVRØ1     125Ø7

*EQUIPMENT TABLE ENTRY
12,DVR15,R
14,DVR12,R
15,DVRØ1,R
16,DVRØ2,R
17,DVRØØ,R
```

```
2Ø,DVRØØ,R
21,DVR3Ø,D,R
/E
* DEVICE REFERENCE TABLE
 1 = EQT #?
5
 2 = EQT #?
7
 3 = EQT #?
Ø
 4 = EQT #?
4
 5 = EQT #?
1
 6 = EQT #?
2
 7 = EQT #?
3
 8 = EQT #?
6
 9 = EQT #?
/E
*INTERRUPT TABLE
12,1
14,2
15,3
16,4
17,5
2Ø,6
/E
EXEC SUPERVISOR MODULES
(NONE)
I/O DRIVER MODULES
(NONE)
```

```
LWA SYS   13235

FWA USER?

14ØØØ

USER SYSTEM PROGRAMS

LOADR     14ØØØ
JOBPR     14ØØØ
ASMB      14ØØØ
ASMBD     2Ø5ØØ
ASMB1     2Ø5ØØ
ASMB2     2Ø5ØØ
ASMB3     2Ø5ØØ
ASMB4     2Ø5ØØ
ASMB5     2Ø5ØØ
FTN       14ØØØ
FTNØ1     147Ø7
  SREAD   23ØØ5
FTNØ2     147Ø7
FTNØ3     147Ø7
FTNØ4     147Ø7
  %WRIT   21253
  FADSB   2154Ø
  .FLUN   21676
  .PACK   21711

*SYSTEM STORED ON DISC
```

APPENDIX D
RELATION TO OTHER SOFTWARE

The Hewlett-Packard 2116B is a general-purpose computer; as such, it can handle standard HP software when the Disc Operating System is inactive. Every computer is shipped with the standard software and documentation appropriate to the system configuration.

In addition, the disc/computer combination may include two disc-based software systems simultaneously (although only one can execute in core at a time): the Disc Operating System, and another software system (either the Time Shared Basic System or the Real-Time Executive System). When loading into core from the disc with Basic Binary Disc Loader (BBDL), the operator specifies which system to load by setting switch register bit 0 equal to 1 (for DOS) or Ø (for another disc-based system) after the BBDL halts. (See Section VI.)

When the two systems are generated, they must be stored on different areas of the disc. This is accomplished by protecting enough tracks to cover both systems; first generate the other system onto the initial tracks and then generate the DOS onto the remaining protected tracks. In this way, DOS does not attempt to write on the other system.

Another way to use the computer and disc for two or more software systems is to dump DOS on magnetic tape using SDUMP (see Section VI) before loading another system from magnetic tape.

In an attempt to make DOS compatible with the Real-Time Executive, DOS 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 segment load.
CURRENT TIME	Word 5 set to Ø, other words identical.
EXECUTION TIME (TIMER)	Not accepted. See N option of RUN request.

APPENDIX E
LINE PRINTER FORMATTING

When a user program makes a READ/WRITE EXEC call to the line printer (HP2778A or HP2778A-01), the line printer driver DVR12 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:

Character	Meaning
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 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 I/O CONTROL EXEC call and the function code equals 11_8 (see Section IV, I/O CONTROL EXEC CALL), then the optional parameter word defines a format action to be taken by the line printer. The parameter word has these meanings:

Parameter Word (Dec)	Meaning
< Ø	Page eject,
Ø to 55	Space Ø 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.

*DVR12 checks for certain program names (FTN, ASMB, LOADR, JOBPR); for these programs, it prints the first character of each line and generates a single space.

If the parameter word equals zero, the automatic single space is to be suppressed on the next print operation only.

CARRIAGE CONTROL CHANNELS

If the parameter word is between 55 and 64, then 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.

APPENDIX F
SUMMARY OF DIRECTIVES

DIRECTIVE	DESCRIPTION
:ABORT	Terminate the current job.
:ADUMP[,*FWA*[,*LWA*]][,B][,L]	Dump a program if it aborts.
:BATCH	Switch from keyboard to batch mode.
:COMMENT *string*	Print a message.
:DATE,*day*[,*hour*,*min*]	Set the date and time.
:DN,*n*	Declare an I/O device down.
:DUMP,*log.unit*, *file*[,S_1[,S_2]]	Dump a user file.
:EDIT,*file*,*log.unit*[,*new*]	Edit a source statement file.
:EJOB	Terminate the current batch and/or job normally.
:EQ[,*n*]	List the equipment table.
:GO[,P_1,P_2...P_5]	Restart a suspended program.
:JFILE,*file*	Specify a source file for the assembler or compiler.
:JOB[,*name*]	Initiate a user job.
:LIST,S,*log.unit*,*file*[,*m*[,*n*]]	List a source statement file.
:LIST,U,*log.unit*[,$file_1$,...]	List the user directory.
:LIST,X,*log.unit*[,$file_1$,...]	List the system directory.
:LU[,n_1[,n_2]]	Assign or list logical units.
:PAUSE	Interrupt the current job.
:PDUMP[,*FWA*[,*LWA*]][,B][,L]	Dump a program after normal completion.
:PROG,*name*[,P_1,P_2...P_5]	Turn on a system or user program.
:PURGE,$file_1$,$file_2$,...	Delete user files.
:RUN,*name*[,*time*][,N]	To run a user program.

SUMMARY OF DIRECTIVES

DIRECTIVE	DESCRIPTION
:SA,*track,sector*[,*number*]	Dump disc in ASCII.
:SO,*track,sector*[,*number*]	Dump disc in octal.
:STORE,A,*file, sectors*	Reserve space for an ASCII data file.
:STORE,B,*file, sectors*	Reserve space for a binary data file.
:STORE,P[,$name_1$, $name_2$,...]	Store loader generated programs.
:STORE,R,*file* [,*log.unit*]	Create a relocatable file.
:STORE,S,*file,log.unit*	Create a source statement file.
:TRACKS[,T_1,T_2...]	Print or set disc track status.
:TYPE	Return to batch from keyboard mode.
:UP,*n*	Declare an I/O device up.

APPENDIX G
SUMMARY OF EXEC CALLS

Consult Section III for the complete details on each EXEC call. The general format of an EXEC call in assembly language is:

```
        EXT     EXEC            (Used to link program to DOS)
         .
         .
         .
        JSB     EXEC            (Transfer control to DOS)
        DEF     *+n+1           (Defines point of return from DOS, n is
                                 number of parameters; must be a direct
                                 address)
        DEF     P1   )
         .           )          (Define addresses of parameters which may
         .           )          occur anywhere in program; may be multi-level
        DEF     Pn   )          indirect)
        return point            (Continue execution of program)
         .
         .
         .
P1      ---          )
 .                   )
 .                   )          (Actual parameter values)
 .                   )
Pn      ---          )
```

For each EXEC call, this appendix includes only the parameters (P_1 through P_n in the format above) of the assembly language calling sequence.

READ/WRITE:			Transfers 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 -2*n*	(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
```

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 Ø if n not
                             available.)
```

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       (xxxxx is the file name.)
NSECT   NOP                 (Number of sectors in file returned here, or
                             Ø if not found.)
```

APPENDIX H
MESSAGES

During the operation of DOS, certain messages may be printed on the system teleprinter. These messages may be error reports or simply informative; they are generated by various parts of DOS. Appendix H lists these messages alphabetically including where they originated, what they mean, and what response, if any, the operator must make. Messages that begin with a variable name or a non-alphabetic character are listed by the first non-variable, alphabetic character. Page references (if any) are given in parentheses.

Message	Source	Description and References
BAD CONTROL STATE.	*JOBPR*	Directive just entered is not acceptable in DOS.
BAD DIRECTORY OR SYSTEM	*JOBPR*	Parity error has occurred during read from disc of a system program, user or system directory, or the system buffer.
BEGIN 'DEBUG' OPERATION	*DEBUG*	Operator may now enter any legal DEBUG operations. (4-25)
CW *nnnnn*	*DISCM*	In a READ/WRITE EXEC call at *nnnnn*, buffer address plus the number of words (or characters) to be transferred would wrap around the top of core. The job is aborted.
DEVICE #*nn* DOWN	*JOBPR*	Logical unit *nn* is unavailable (down). An :UP,*nn* makes *nn* available (up) again.

Message	Source	Description and Reference
DISC TRACK *ttt* ERROR	*DISCM*	Disc error when attempting to read track *ttt*.
DICTIONARY OVERFLOW	*JOBPR*	No room is left for entries in the user file dictionary.
DONE?	*JOBPR*	Thirty feed frames (paper tape) or an end-of-file (magnetic tape) have occurred during input. Operator responds with YES for end of input, NO for more input. (2-15)
DUPLICATE FILENAME	*JOBPR*	Doubly defined file name found in a :STORE directive, (other than STORE,P), or an :EDIT directive with a new file.
$END ASMB	*ASMB*	Assembly has completed. (4-11)
$END ASMB CS	*ASMB*	Assembly has terminated because of an error in the Assembler Control Statement. (4-11)
$END ASMB NPRG	*ASMB*	Assembly has terminated because no JFILE was found when required. (4-12)
$END ASMB PASS	*ASMB*	Another pass of the source program through the input device is required. (4-11)
$END ASMB XEND	*ASMB*	Assembly stops because an EOF occurred in the source program before an END Statement. (4-12)

Message	Source	Description and References
END FILE	*JOBPR*	During an :EDIT, the master file ended before completion of editing or a colon occurred in column 1 of a source statement.
$END FTN	*FTN*	Compilation has completed. (4-3)
END JOB *xxxx* RUN = *xxxx* MIN. *xx.x* SEC EXEC = *xxxx* MIN. *xx.x* SEC	*JOBPR*	End of current job. Total job time and execution time are reported. (2-4)
ENTER FILE NAME(S) OR /E	*LOADR*	Enter list of relocatable program files terminated by /E. (4-22)
ENTRY ERROR	*DEBUG*	DEBUG operation entered was illegal. (4-25)
EQT *xx* CH *xx* DVR*xx* D R U*x* S*x*	*JOBPR*	Equipment table entry printed by :EQ. (2-31)
EXTRA PARAMETERS	*JOBPR*	More than 15 parameters in a directive.
FI *nnnnn*	*DISCM*	In a FILE READ/WRITE EXEC call, the file *nnnnn* cannot be found. Calling program is aborted.

Message	Source	Description and References
IB *nnnnn*	*DISCM*	Illegal buffer address in EXEC call at location *nnnnn*. Program is aborted.
IE *nnnnn*	*DISCM*	EXEC call at *nnnnn* read in a :card from batch input device. Program is aborted.
IGNORED	*DISCM*	Input from system teleprinter or batch device during program execution cannot be processed.
*IGNORED	*JOBPR*	All directives following EJOB and before next JOB except BATCH, TYPE and TRACKS are ignored.
file ILLEGAL	*JOBPR*	1) On a source file LIST directive, the requested file was not a source file. (2-24) 2) A file name begins with a non-alphabetic character.
ILLEGAL DIGIT	*JOBPR*	In a decimal number, digit is other than Ø-9. In an octal number, digit is other than Ø-7.
ILLEGAL LUN	*JOBPR*	Logical unit requested is = Ø, greater than number of logical units in the table, or is not the correct type (i.e., input for output, etc).

Message	Source	Description and References
ILLEGAL PROGRAM TYPE	*JOBPR*	Name requested in a RUN or PROG is not legal.
INP ERR	*DISCM*	Equipment table entry number of logical unit number in EQ, LU, UP or DN is illegal.
INPUT:DATE,XXXXXXXXXX,H,M	*DISCM*	When system is initiated from the disc, DOS requires a DATE directive. (6-15)
INPUT FR=FRESH; CO=CONTINUATION	*DISCM*	When system is initiated from the disc, DOS asks whether start-up is fresh (no user files) or continuation (user files on disc). (6-15)
I/O ERR NR EQT# *mm*	*DISCM*	Device #*mm* is not ready. DOS returns to program return address with status in A, B set to Ø.
I/O ERR ET EQT# *mm*	*DISCM*	An end-of-tape occurred on device #*mm*. DOS returns to program return address with status in A, B set to Ø.
I/O ERR PE EQT# *mm*	*DISCM*	Parity error on device #*mm*. DOS returns to program return address with A set to status, B set to Ø.

Message	Source	Description and References
IT *nnnnn*	*DISCM*	Illegal disc track or sector address in EXEC call from location *nnnnn*. Program is aborted.
JBIN OVF	*FTN,ASMB*	Overflow of job binary area during assembly or compilation. (4-4,4-12)
JBIN TRK BAD	*JOBPR*	Parity error when reading a program from the job binary area.
JOB ABORTED!	*JOBPR*	Current job is aborted because of: 1) parity error on disc (in user file, work file, system or user directory system file on system buffer), 2) :ABORT directive, 3) end-of-file during EDIT or source input, 4) dictionary overflow during a STORE, 5) parity error on job binary track, 6) no tracks left for writing on the disc.
JOB *xxxxx ddddddddd* TIME = *xxxx*MIN.*xx.x*SECS EXEC = *xxxx* MIN.*xx.x* SEC.	*JOBPR*	Message printed at the beginning of each job. (2-3)
LØ1 ⋮ L16	*LOADR*	Loader error messages. (4-30)

Message	Source	Description and References
LIMIT ERROR	*JOBPR*	1) Source statement numbers out of order in an EDIT; 2) dump limits incompatible in PDUMP, ADUMP; 3) sectors illegal in a DUMP; 4) or beginning source statement in LIST is greater than final statement number.
xxxx LINES	*JOBPR*	Total number of statements stored by a STORE,S directive
****LIST END****	*JOBPR*	Terminates list of source statements generated by a LIST directive. (2-25)
LN *nnnnn*	*DISCM*	Logical unit requested by an EXEC call at *nnnnn* is unassigned. Program is aborted.
LOADR COMPLETED	*LOADR*	Loading has completed. (4-24)
LOADR SUSP	*LOADR*	Loader has suspended and is waiting for a GO directive. (4-23)
LOADR TERMINATED	*LOADR*	Loader has terminated because of an error. (4-28)
LOAD TAPE	*LOADR*	In conjunction with LOADR SUSP, this message requests that next relocatable tape be loaded before GO. (4-23)

Message	Source	Description and References
LU *nnnnn*	*DISCM*	Illegal logical unit in EXEC call at *nnnnn*. Program is aborted.
LU*xx* EQT*xx*	*JOBPR*	Logical unit table entry; EQT # *xx* is assigned to LU#*xx*. (2-32)
LUN UNASSIGNED	*JOBPR*	Logical unit requested in a directive is unassigned.
xxxxx MISSING	*DISCM*	Segment *xxxxx*, requested by an EXEC call, is not in system or user directory. Job is aborted.
MISSING PARAMETER	*JOBPR*	A parameter is missing in a directive.
MP *nnnnn*	*DISCM*	Illegal memory protect violation at location *nnnnn*. Program is aborted.
NAME*IGNORED	*JOBPR*	Illegal JOB *name*; non-alphabetic first character.
NO BIN END	*JOBPR*	No END record detected when storing a relocatable binary program.
NO PROGRAM LOADED	*LOADR*	No programs were loaded into the LOADR. Loading terminates. (4-23)

Message	Source	Description and References
NO SOURCE	*JOBPR*	No source statements following a /R or /I in an EDIT directive. Job is aborted.
NO TRACKS AVAILABLE	*JOBPR*	No tracks available on disc for writing
NUMBER OVERFLO	*JOBPR*	An integer is too large.
OR *nnnnn*	*DISCM*	I/O operation requested by EXEC call at *nnnnn* is rejected. Program is aborted.
PARAMETER ILLEGAL	*JOBPR*	1) no EQT number in EQ directive; 2) slash missing in EDIT file; 3) non-source file requested in EDIT or JFILE; 4) illegal type character in STORE or LIST; 5) logical unit missing or = Ø; 6) sector count = Ø in STORE; 7) character other than B or L in PDUMP or ADUMP; 8) logical unit = system teleprinter or DISC/DRUM in UP or DN; or 9) number out of range of table in an EQ, LU, UP or DN.
PARITY ERROR/TRK=*ttt*	*JOBPR*	Parity error during disc read.

Message	Source	Description and References
PAUSE *xxxx*		
	program	Program has suspended itself. Restart with GO. (4-8)
RE-ENTER STATEMENT ON TTY		
	JOBPR	Follows most error messsges that do cause abort.
RQ *nnnnn*		
	DISCM	Illegal request code in EXEC call at *nnnnn*. Program is aborted.
STOP *xxxx*		
	program	Program has terminated. (4-28)
xxxx SUSP		
	DISCM	Program *xxxx* suspended by EXEC call. Enter GO to proceed.
TM *nnnnn*		
	DISCM	Maximum execution time exceeded. The program is currently at *nnnnn* and is aborted.
# TRACKS UNAVAILABLE		
	DISCM	Desired number of contiguous tracks (in RT disc-allocation EXEC call) is not available. Job aborted unless N present in RUN directive. (4-4)
TRAK # TOO BIG		
	JOBPR	Track requested is higher than last available disc track. (track may be in JBIN area.)
file name UNDEFINED		
	JOBPR	Undefined file-name in PURGE, LIST, RUN or STORE,P. (2-21,2-23)

Message	Source	Description and References
UNDEFINED EXTS	*LOADR*	Undefined external references exist in programs loaded. (4-24)
WAIT	*JOBPR*	DOS is purging the user files or moving them, sector by sector, because of parity error on read. Printed every 6 seconds.
WRONG INPUT	*JOBPR*	Relocatable binary input furnished for a source file request or vice-versa.
nn xx	*ERRØ*	Library routine error code. (4-9)
@	*JOBPR*	DOS is ready for further directions. (2-2)
*	*DISCM*	
1ST WORK TRACK=*ttt* BAD= *bbb*	*JOBPR*	In TRACK directive, *ttt* = first available work track; *bbb* = faulty tracks.

APPENDIX I
MAGNETIC TAPE USAGE

Input/output transfers to and from a HP3Ø3Ø 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 (Ø) 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 (Ø) buffer length on binary read causes a forward skip one record.

In the I/O STATUS EXEC call, bits 7-Ø 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):

BIT	MEANING
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.
Ø	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 ØØ), 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 less than six words are padded to six words. The maximum buffer length is 16,384.

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 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.

SOFTWARE MANUAL CHANGES

DISC OPERATING SYSTEM

(HP 02116-91748)

Dated October 1968

Some of the items below pertain not only to the 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 deleted.

11-69

Change Number	Description
1	Page 2-4, under EJOB, replace the first sentence under "Comments" with: "EJOB condenses the user file by eliminating spaces left by non-permanent programs."
2	Page 2-15, Type -S Files, in the second paragraph, delete the words "or blank card."
3	Page 6-4, before FIRST SYSTEM SECTOR? add: "(The system cannot start on track 0 sector 0 since sectors 0-2 are used for the disc loaders.)"
4	Page 6-10, at the bottom of the page, add: "(For 3030 magnetic tape, the entry in the interrupt table should be the location of the magnetic tape 2 board.)"
5	Page H-9, below OR *nnnnn*, insert the following system message: OVERFLOW JBIN There is not enough room in the user area for starting relocatable binary from JBIN area.

HEWLETT hp PACKARD Software Development Cupertino, California 95014

1-70

Change Number	Description
6	Page xii, in the third line delete "12591A" and replace it with "12581A."
7	Page 2-4, in the first line, delete the word "purges" and replace with the word "condenses."
8	Page 2-15, in the first line of the third paragraph, delete the words "or blank card."
9	Page 4-7, change the third DEBUG operation from "D,A,N[,N_2]" to "D,B,N[,N_2]."
10	Page 6-16, in the first line of the last paragraph, delete the rest of the sentence after the word "requires" and insert "a parameter specifying the first track to be dumped. The end parameter is optional."
11	Page 6-17, in line nine before the word "in" insert "in command or."
12	Page 6-18, After the error message "TRACKnnn(8) SECTORmmm(8)" delete the rest of the sentence after the word "DISC" and insert "and Tape Error Diagnostics are described as follows."

3-70

13	Page 1-4, insert after the first paragraph: "A memory protect boundary is set between the executive area and a 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. Programs to be run in the user area must use EXEC calls for input/output, termination, and suspension."
14	Page 3-8, under the word "FORTRAN", in the first and seventh lines delete "DIM" and insert "DIMENSION."
15	Page 3-13, under the word "FORTRAN", insert after "IRCDE", "IFTRK".

3-70

Change Number	Description
16	Page 3-19, under the word "FORTRAN" in the second line delete "DIM" and insert "DIMENSION."
17	Page 3-21, under the word "FORTRAN" in the second line delete "DIM and insert "DIMENSION."
18	Page 3-22, under the word "FORTRAN" in the second line delete "DIM" and insert "DIMENSION."
19	Page H-9, insert after the fourth message the following: "OVERFLO JBIN. JOBPR insufficient space in job binary area for reloactable code."

4-70

20	Page 1, of the Manual Change Sheet on the title replace "1968" with "1969."
21	Page 2, of this Manual Change Sheet, delete change number 7 and change number 8.
22	Page 2, of this Manual Change Sheet, change number 15, replace "IRCDE" with "IFTRK". Replace "IFTRK" with "IRCDE".
23	Page 3, of this Manual Change Sheet, delete change number 19.

5-70

24	Page 2-13, under the second entry should be: ":STORE,P[, name 1, name 2,...]"
25	Page 2-36, under Format, in the second line insert the following parethetical expression after "characters". "(commas not permitted);" In the third line, replace "user" with "operator." Under EXAMPLES, in the first line replace "10," with "10/".

5-70

Change Number	Description
26	Page 5-12, in the center of the flowchart, change "EQT(5)" to "EQT(4)".

6-70

27	Page 2-17, replace the first paragraph under "Comments" with: "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 statement using a :STORE,S directive."
28	Page 2-29, under "Comments" delete the first paragraph and insert the following: "Any parameter following L is ignored. If FWA is greater than LWA, a message is printed. When the directive :PDUMP precedes a :RUN or :PROG request, the program contained in the request will be dumped, if it runs to normal completion. To dump a program that is aborted while running, the directive :ADUMP must precede the :RUN request. To make sure that a program will be dumped whether it runs normally or is aborted, both dump directives must be declared preceding the :RUN request. Only one of the requests will be honored, depending upon whether the, program runs normally or is aborted. Since DOS sets a flag when it encounters either dump directive, then clears the flag after the dump routine is executed, the flag representing the dump routine that was not executed will remain set. This flag can cause an unwanted dump of some program run later under the same :JOB directive. Either dump flag can be cleared by requesting the dump with both FWA and LWA equal to 0; all flags can be cleared by calling a new :JOB directive."

7-70

Change Number	Description
29	Page 2-17, replace the first paragraph under "Comments" with: "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 using a :STORE,S directive."
30	Page 2-29, under "Comments" delete the first paragraph and insert the following: "Any parameter following L is ignored. If FWA is greater than LWA, a message is printed. When the directive :PDUMP precedes a :RUN or :PROG request, the program contained in the request will be dumped, if it runs to normal completion. To dump a program that is aborted while running, the directive :ADUMP must precede the :RUN request. To make sure that a program will be dumped whether it runs normally or is aborted, both dump directives must be declared preceding the :RUN request. Only one of the requests will be honored, depending upon whether the, program runs normally or is aborted. Since DOS sets a flag when it encounters either dump directive, then clears the flag after the dump routine is executed, the flag representing the dump routine that was not executed will remain set. This flag can cause an unwanted dump of some program run later under the same :JOB directive. Either dump flag can be cleared by requesting the dump with both FWA and LWA equal to 0; all flags can be cleared by calling a new :JOB directive."