HEWLETT hp PACKARD

2100 series computers

DOS-III

disc operating system

June 1973

02100-90136

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

List of Effective Pages

Pages	Effective Date
Title	Jun. 1973
Copyright	Jun. 1973
iii to xiii	Jun. 1973
1-1 to 1-10	Jun. 1973
2-1 to 2-57	Jun. 1973
3-1 to 3-35	Jun. 1973
4-1 to 4-7	Jun. 1973
5-1 to 5-34	Jun. 1973
6-1 to 6-2	Jun. 1973
7-1 to 7-4	Jun. 1973
8-1 to 8-25	Jun. 1973
9-1 to 9-22	Jun. 1973
10-1 to 10-21	Jun. 1973
11-1 to 11-9	Jun. 1973
12-1 to 12-10	Jun. 1973
13-1 to 13-20	Jun. 1973
14-1 to 14-3	Jun. 1973
15-1 to 15-16	Jun. 1973
A-1 to A-15	Jun. 1973
Index 1 (1) to (3)	Jun. 1973
Index 2 (1) to (2)	Jun. 1973
Index 3 (1) to (3)	Jun. 1973

Printing History

First Edition . . . . . Jun. 1973

Preface

This manual is a programming guide to DOS-III, a Hewlett-Packard Disc Operating System for 2100 Series computers. Programmers using this manual should be familiar with the functions of batch-processing operating systems and one of the programming languages supported by the DOS-III Operating System.

The Hewlett-Packard programming languages and program libraries that can operate under control of DOS-III are described in the following reference manuals:

- *HP ALGOL (02116-9072)*
- *HP ASSEMBLER (02116-9014)*
- *HP FORTRAN (02116-9015)*
- *HP FORTRAN IV (5951-1321)*
- *RELOCATABLE SUBROUTINES (02116-91780)*

Other information, which may be useful to the programmer, is included in the SMALL PROGRAMS MANUAL, the MANUAL OF DIAGNOSTICS and the SOFTWARE OPERATING PROCEDURES. These manuals contain custom-assembled modules pertaining to each customer's software and hardware configurations, and are supplied with each Hewlett-Packard computer system.

This manual is divided into six functional parts:

- Part 1. DOS-III OPERATING SYSTEM

  Part 1 defines the standard capabilities of DOS-III. It includes a summary of DOS-III organization, hardware and software; definitions of DOS-III directives, EXEC calls and I/O routines; a description of the interaction of DOS-III and its subsystems; and a set of sample job decks.

- Part 2. DOS-III EXTENDED FILE MANAGEMENT PACKAGE (EFMP)

  Part 2 describes the capabilities of the DOS-III Extended File Management Package (EFMP), which allows the programmer to extend the file-handling capabilities of the DOS-III Operating System. Part 2 contains sections on EFMP organization, EXEC calls and use of UTIL, the EFMP Utility Program.

- Part 3. GENERATING AND LOADING DOS-III

  Part 3 gives complete instructions for generating and loading a DOS-III System.

- Part 4. DOS-III SYSTEMS PROGRAMMING

  Part 4 contains information which will help the advanced programmer to write his own EXEC modules, plan I/O drivers and use the DOS-III privileged mode capabilities.

- Part 5. ERROR CODES AND MESSAGES

  Part 5 is a complete set of all DOS-III Operating System error codes and messages.

- Part 6. APPENDIX AND INDEXES

  Part 6 contains an appendix of DOS-III system tables and three indexes: the first two are convenient summaries of DOS-III directives and EXEC calls; the third refers to terms discussed in the manual.

Contents

PART 2 DOS-III Extended File Management Package (EFMP)

TABLES

FIGURES

SECTION I
DOS-III Organization

The DOS-III supervisory software consists of a Disc Monitor (DISCM) that resides in main memory; EXEC modules which may reside either in main memory or on disc; and a Job Processor (JOBPR) that is disc-resident. Together these modules manage I/O processing, interrupt processing, executive processing, job processing, and file handling.

Other DOS-III software consists of a series of relocatable binary software modules. Since each module is an independent, general-purpose program, the hardware and software configuration of the system is flexible. Modules can either reside in main memory or on the disc, at the user's option (specified during system generation). In a system with a small main memory, the modules can reside on the disc to save main memory space; in a large main memory system, modules can reside in main memory for greater efficiency.

MAIN MEMORY LAYOUT

When DOS-III is active, the main memory is divided into a User 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 main memory by the supervisor when needed. The User Area provides space for execution of user programs.

In addition, large DOS-III software modules, such as the FORTRAN Compilers, Assembler, Relocating Loader, and Job Processor, reside on the disc and execute in the User Area. (See Appendix A for figures on disc and main memory layout.)

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

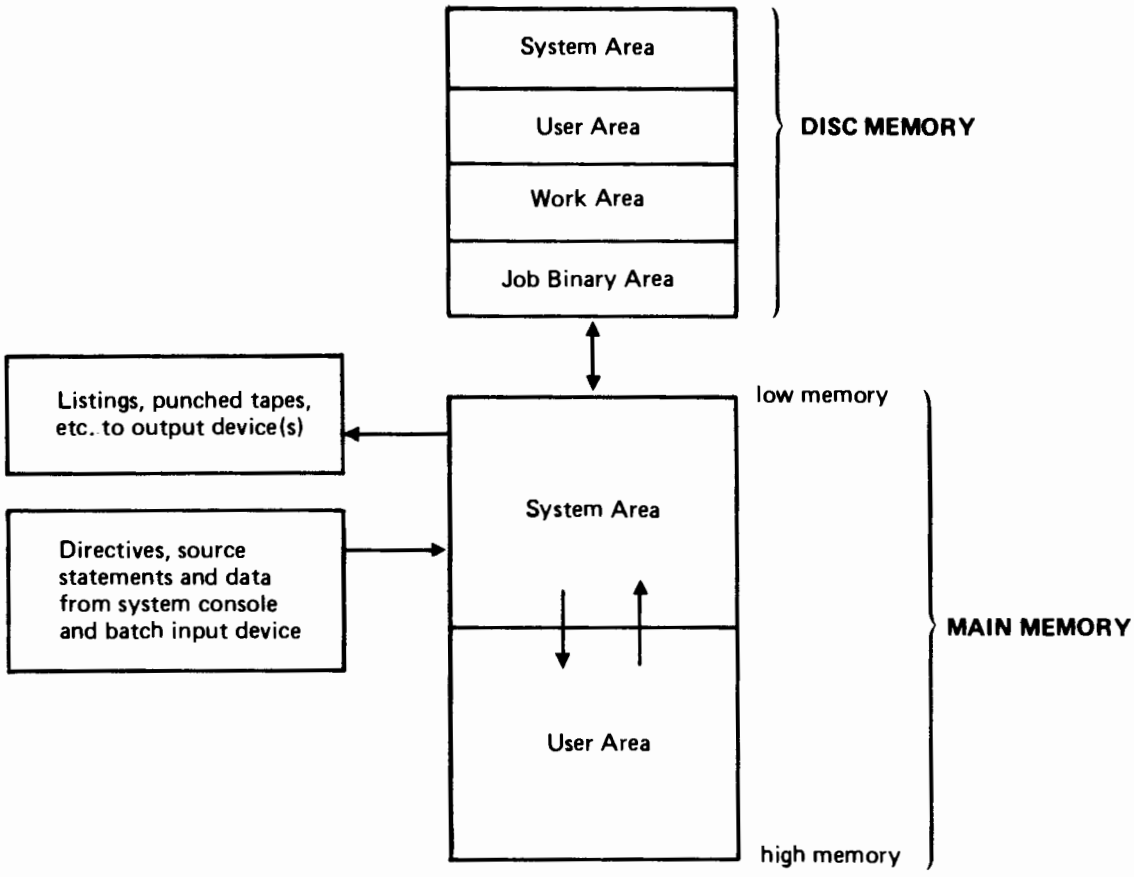

Figure 1-1. Functional Diagram of DOS-III

DOS-III OPERATION

DOS-III operates in either keyboard or batch mode. In keyboard mode, the user enters statements and commands to the system (called directives) to control his programming job through a keyboard device (system console). Each line entered must terminate with a *return* and a *linefeed.* In batch mode, the user enters directives through a batch input device, sometimes integrated with a source program on punched cards, paper tape or magnetic tape, thus forming a job deck. Jobs can be stacked one upon another in a queue.

Deleting Keyboard Errors

To delete an entire line of input, strike *rubout* then *linefeed.* To delete the character just entered, strike *Control-A* (simultaneous "A" and *control* key striking). Each *Control-A* deletes one additional preceding character.

Batch Abort

Some errors when encountered in batch mode cause a batch abort. When such an error occurs (mostly in response to a directive) DOS-III takes the following action:

1. The offending directive and an error message is printed on the list device.
2. JOB ABORTED is printed on both the system console and the list device.
3. The offending statement and subsequent statements are ignored until a JOB, EJOB, or TYPE directive is encountered. The current operation is aborted and the next input is processed.

DOS-III DIRECTIVES

The DOS-III 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-III. They are entered in one of the two modes of DOS-III operation: keyboard or batch.

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

- Create, rename, edit, list, dump, and purge user files (relocatable, absolute, loader-generated, source statements, and ASCII or binary data)
- Search the various disc subchannels for specified file names
- Check status of user disc tracks
- Turn on user programs or system programs such as FORTRAN and Assembler
- Examine and modify the logical organization of the I/O; rewind magnetic tapes and output end-of-file commands to magnetic tapes; output top-of-form commands to list devices
- Start and stop a job; type comments; suspend operations; resume execution of suspended programs

- Assemble or compile, load and execute a user program
- Dump main or disc memory
- Set the date; abort programs; transfer to batch mode (from keyboard mode or batch mode); return to keyboard mode (from batch mode)
- Change the subchannel of the user disc
- Initialize (label) a disc subchannel
- Dump all (or part of) a disc to another disc

DOS-III directives are described in Section II.

DOS-III EXEC CALLS

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

The EXEC calls perform the following functions:

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

DOS-III EXEC calls are described in Section III.

DOS-III INPUT/OUTPUT

All I/O operations and interrupts are channeled through the DISCM section of the DOS-III Supervisor. DISCM is always main-memory resident and maintains ultimate control of the computer resources.

I/O programming is device-independent. Programs written in FORTRAN, ALGOL, and Assembler 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 initially are assigned to appropriate devices by the operator during system generation, depending upon what is available and can be assigned during a job. Thus, the programmer need not worry about the type of input or output device performing the actual operation.

DOS-III FILES

Two types of files can be included in the DOS-III system: standard files (created by the STORE or EDIT directives) and files created under the Extended File Management Package (if EFMP is included in the system).

Standard Files

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

Programs use the Work Area of the disc for temporary storage. The System Area contains files of systems programs, EXEC modules, a system directory, and system library subroutines.

DOS-III Extended File Management Package

DOS-III installations with 16K of main memory can use the DOS-III Extended File Management Package (EFMP). This set of optional EXEC modules allows the user to exploit a more powerful file structure than that provided by DOS-III. EFMP files allow logical record sizes of varying lengths for different files, security codes, flexible buffering, sequential reads and writes with a pointer, and detailed status information. In addition, a utility program (UTIL) is available that operates in the User Area. UTIL makes those EFMP functions (except reads and writes), normally only usable through EXEC calls, usable from the keyboard. For more information on EFMP, see Part 2.

GENERATING A DOS-III SYSTEM

DOS-III is generated and loaded using two absolute programs:

- Configured DSGEN (the system generator)
- Main-memory resident Loader (the bootstrap loader which loads the configured DOS-III from the disc into main memory)

First, DSGEN outputs instructions to the operator asking for information about the system. At the appropriate point in the dialogue, the operator loads in the relocatable binary modules which make up DOS-III and specifies whether the modules are to be disc- or main-memory resident. Finally, DSGEN stores the configured DOS-III system on the disc in absolute form. (The disc is protected from alteration by a hardware override switch.)

DOS-III then resides as a System Area and User Area on the disc. Each area is labeled and contains a directory of all the files contained within the area. The System Area contains system main-memory resident and disc-resident modules, while the User Area contains user files.

To load DOS-III into main memory and begin system execution, the user executes the main-memory resident Loader. This Loader loads all the modules designated main-memory resident into main memory. (The disc-resident modules are brought into main memory when needed by the main-memory resident modules.)

DISC USAGE

HP 2870/7900/7901

The controller for the moving-head disc supports up to four disc drives (one is required). Each 7900/2870 drive contains two discs: a fixed disc and a removable cartridge. Each 7901 drive contains one disc: a removable cartridge. Each disc is referenced through a subchannel of the controller. Therefore, the controller has a maximum of eight subchannels (numbered 0 to 7). The channels are assigned as follows:

7900/2870					7901			
0	1	2	3	Disc Drive Numbers	0	1	2	3
1	3	5	7	Removable Subchannels	1	3	5	7
0	2	4	6	Permanent Subchannels	None			

HP 2883

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

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

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

DISC STORAGE

Each subchannel contains 203 tracks. At least three of these tracks must be reserved as spares. On the HP 2870, each track contains 24 sectors; on the HP 2883, 115 sectors; on the HP 7900/7901, 48 sectors. (A sector contains 128 sixteen-bit words and is the smallest addressable unit on the disc.) DOS-III normally allows two subchannels to be available to the user: one subchannel contains the system disc and the other contains the user disc (may be the same subchannel as the system disc). The user subchannel can be changed during job or program execution. In addition, an optional system search mode is available to allow searching for user files on any specified subchannels.

The disc storage has four parts:

1. The System Area

   Executable code created by the system generator and hardware protected; includes DOS-III Supervisor and other system programs.

2. The User Area (optional)

   User file directory and user files (data, object programs, source statements, etc.).

3. The Work Area

   Temporary storage for the current job.

4. Job Binary Area

   Temporary storage for relocatable object code generated by the Assembler and compilers; this is an area of variable size and starts from the end of the disc.

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

Automatic track switching is provided within each subchannel.

DOS-III HARDWARE

Required Hardware

The minimum hardware requirements for DOS-III are

1. HP 2100 series computer, with 12,288 words of main memory, Central Interrupt Processor, DMA, halt on memory parity error.

2. HP 7900/2870 Moving-Head Disc Drive with fixed disc and removable cartridge; or the HP 2883 Disc File with one removable pack; or the HP 7901 Moving-Head Disc Drive with removable cartridge.

3. System Console device.

4. HP 2748B Tape Reader.

Hardware Options

The following hardware options are available:

1. Time-base Generator (provides accounting times and time-of-day).

2. Extended Arithmetic Unit (EAU): provides hardware multiply, divide, etc., for user programs.

3. Floating-point hardware.

4. Additional main memory to a total of 16,384, 24,576 or 32,768 words.

5. Using extenders, additional I/O channels (up to channel 37_8).

6. Memory Protect. (Without memory protect, user programs can destroy DOS-III.)

7. Paper Tape Punch.

8. Line Printer.

9. Card Reader.

10. Magnetic Tape Unit.

11. Additional Disc Drives. (Maximum is four on HP 7900/7901/2870 and two on HP 2883.)

12. Plotter.

13. CRT display.

14. Writable Control Store.

DOS-III SOFTWARE

Required Software

The minimum software requirements for DOS-III are

1. Absolute Programs
    a. DOS-III System Generator (DSGEN)
    b. DOS-III Bootstrap Loader
2. Relocatable Programs
    a. DOS-III Disc Monitor (DISCM)
    b. DOS-III Exec Modules
    c. DOS-III Job Processor (JOBPR)
    d. DOS-M Disc Driver (DVR31)
    e. DOS-M System Console Driver (DVR05)

Software Options

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

1. DOS-III Relocating Loader
2. DOS-M Assembler
3. DOS-M FORTRAN Compiler
4. RTE/DOS FORTRAN IV Compiler
5. RTE/DOS FORTRAN IV Compiler — 10K Compiler Area (16K main memory required)
6. RTE/DOS ALGOL Compiler (16K main memory required)
7. RTE/DOS Relocatable Library (EAU, non-EAU, or floating point)
8. RTE/DOS FORTRAN IV Library (extended-precision arithmetic)

9. RTE/DOS FORTRAN Formatter

10. DOS I/O Drivers (either main-memory or disc-resident):

    Teleprinter (DVR00)

    Photoreader Driver (DVR01)

    Tape Punch (DVR02)

    Plotter (DVR10)

    Card Reader (DVR11) — uses DMA

    Line Printer (DVR12)

    Mark Sense Card Reader (DVR15) — uses DMA

    Magnetic Tape (DVR23) — uses DMA

    Writable Control Store (DVR33)

11. DOS-III Extended File Mangement Package (16K main memory required)

SECTION II
DOS-III Directives

Directives are the direct line of communication between the keyboard or batch input device and DOS-III. Directives may enter DOS-III in two modes: keyboard and batch. In either mode, all directives are listed on the system console. Certain directives can be used in one mode only; others can be used in both modes. In keyboard mode, the operator manually inputs the directives through the system console keyboard. In batch mode, the programmer prepares the directives (commonly on punched cards, paper tapes, or magnetic tape) and inputs them along with programs, data, etc., in a complete job.

FORMAT FOR DIRECTIVES

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

:PURGE,FILE1,FILE2,FILE3

When the sequence and position of parameters is significant, missing parameters must be represented by commas if the following parameters are to be recognized. The first blank character not preceded by a comma is the end of the directive. Comments may appear after this blank; they are ignored by DOS-III.

Note: The total length of an input string cannot exceed 72 characters.

ENTERING DIRECTIVES

DOS-III has two conventions for notifying the operator that directives may be entered:

1. DOS-III outputs a "commercial at" sign (@) and rings a bell (at the system console). At this time, the operator may enter any directive.

2. DOS-III outputs an asterisk (at the system console). At this time the operator may enter an "operator attention" directive only. The "operator attention" directives are

:ABORT

:DN

:EQ

:LU

:OFF

:PAUSE

:TRACKS

:TYPE

:UP

Should the operator type any other directive, DOS-III outputs the following message:

IGNORED

and returns to the executing program.

To attain control of DOS-III (to enter an "operator attention" directive) the operator can strike any system console keyboard key. If the system console is available, DOS-III immediately outputs an asterisk (*); if the system console is busy, DOS-III will output the asterisk as soon as it releases the system console.

Notes: 1. Operator attention is disabled during the completion phase of :EDIT and during :PURGE.

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

3. Some operations, such as editing, require perceptible waits while DOS-III processes the directive.

ORDER OF DIRECTIVES

The DOS-III directives described in this section are presented in alphabetic order (by function name). If a directive must be used in keyboard mode only, a note to that effect is placed at the top of each page describing the directive. A quick cross-reference index of DOS-III directives, "Summary of Directives," is included at the back of this manual.

ABORT

Purpose

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

Format

:ABORT

Comments

Abort carries out all the operations of a batch mode EJOB directive. All I/O devices are cleared.

BATCH

Purpose

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

Format

:BATCH,logical unit

where *logical unit* is the logical unit number of the desired batch input device.

Comments

See "TYPE" in this section for the opposite procedure of returning from batch mode to keyboard mode. Assigning a null device or logical unit numbers 2 or 3 as the batch device results in an ILLEGAL LUN error (see LOGICAL UNIT directive).

CLEAR

Purpose

To clear the Job Binary Area on the disc, or to issue a clear command to an I/O device.

Format

:CLEAR[,logical unit]

where *logical ι it* is the logical unit number of the device to be cleared. If *logical unit* is omitted, the disc Job Binary Area is cleared.

Comments

Using logical units 1, 2, or 3 results in an LU error.

The effect of clearing an I/O device is the transmittal of a clear function to the appropriate driver.

COMMENT

Purpose

To print a message on the system console.

Format

:COMMENT character string

where *character string* is a message to be printed on the system console.

Comments

A space (but not a comma) is required between the directive word and the comment string.

The programmer can use :COMMENT or :PAUSE to send a message to the operator at the system console; using :COMMENT causes no suspension of processing. Use :PAUSE when a processing delay is desired, for example to request that the operator mount a magnetic tape.

EXAMPLES:

:COMMENT PLACE MAGTAPE LABELED "INPUT" ON THE M.T. UNIT

:COMMENT IF THIS NEXT PROGRAM ABORTS, CALL X1234.

DATE

Purpose

To set the date and time for accounting purposes whenever DOS-III is activated.

Format

:DATE,day[,hour,min]

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

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

Comments

The DATE directive is legal only as the first directive in a start-up procedure. The directive is not accepted any other time.

EXAMPLES:

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

:DATE,WEDNESDAY,7,45

:DATE,10JULY1969

DOWN

Purpose

To declare an I/O device unavailable for use during the remainder of a job.

Format

:DN,n

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

Comments

The system console and the disc (logical units 1, 2, and 3) cannot be set down.

DUMP (DISC-TO-DISC)

Purpose

1. To dump an entire disc onto another subchannel (:DD)

2. To dump the System Area (including system buffer) onto another subchannel (:DD,X)

3. To dump all or specified files of the User Area (optionally assigning some new file names) onto another subchannel (:DD,U ...) or, onto the current subchannel (assigning new file names).

Formats

1. *:DD*

2. *:DD,X*

3. *:DD,U[,file 1[,(file A)],file 2[,(file B)] ...]*

where *X* specifies the System Area,

U specifies the User Area,

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

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

Note: No more than 14 parameters can be specified after :DD,U.

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

Comments

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

The SS directive does not apply to :DD.

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

TRAC # TOO BIG

If DOS-III cannot find some of the files specified to be dumped, the message

file

UNDEFINED

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

If a file specified to be dumped has the same name (after the optional renaming) as an existing file on the destination disc, the message

file

DUPLICATE FILE-NAME

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

DUMP (FILE)

Purpose

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

Format

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

where *logical unit* is the logical unit number of 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 (including magnetic tape). The dump format varies with the type of file and the type of device. See Table 2-1.

Table 2-1. :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
Absolute binary	Absolute binary records	8 octal words/line
Relocatable binary	Relocatable binary records (loadable)	8 octal words/line
Source statements	1 statement/record	1 statement/line

Note: Sector numbers on listings are not related to the S1 and S2 parameters.

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.

An error message occurs when *S1* > *S2*, or when either *S1* or *S2* is greater than the length of the file.

Source statements, relocatable binary and absolute binary files can be dumped to a punch device and later restored by using the appropriate STORE directive. In general, however, this cannot be done with ASCII data and binary data files.

EXAMPLES:

Where L is a source file:

```
:DUMP,1,L
A
BB
CCC
DDDD
EEEEE
FFFFFF
GGGGGGG
@
```

Where SSERH is a binary file:

(On the system console:)

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

(On the list device:)

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

DUMP (PROGRAM)

Purpose

To request that a user program be dumped to the standard list device (logical unit 6) 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 octal address, relative to the program origin, of the first word to be dumped

LWA is the octal address, relative to the program origin, of the last word to be dumped

B means dump the base page linkage area of the program

L means dump the library subroutines used by the program.

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-III sets a flag when it encounters either PDUMP or ADUMP, then checks the flag the next time a program is executed. Only one of the requests will be honored, depending upon whether the program runs normally or is aborted. The dump is labeled accordingly. These flags are cleared when a program terminates.

Any parameter following L in the directive is ignored. If *FWA* is greater than *LWA*, this message is output:

LIMIT ERROR

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*
adr_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 octal 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 main memory). 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	*item-1*	*item-2*	*item-1*	*item-2*	*item-1*	*item-2*	*item-1*	*item-2*

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

EXAMPLE:

```
:ADUMP,0,15,B  (Set up dump flag)
:RUN,PRG9,6    (Run program)
LU     012140

(Main program dump)
ADUMP
12000   160001  002002  130573  170574  006004  160001  002003  026012
12010   130575  170576  006004  160001  170577  006004  160001  170600

(Page Eject)
(Base page dump)

00570   010137  002045  010711  003237  010763  002045  017014  000300
00574   017641  000000  017015  000400  017641  000406  017601  000000
00600   017650  000000  017615  000000  017664  000000  017662  000573
00604   017637  000573  017571  177205  017563  001204  017714  017715
00610   017562  021121  017534  021122  017536  021122  017633  160656
00614   017544  037626  017546  037626  017673  000000  017605  000040
```

DUMP (SECTOR)

Purpose

To dump any specified sector or sectors of the current user disc on the standard list device (logical unit 6) 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

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:

(On the system console:)

:SO,0,1

@

(On the list device:)

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

EDIT

Purpose

To perform listed edit operations on a user source file (follows the :SS condition).

Format

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

where *file* is the name of a source file (the primary 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*) on the same subchannel and the old file is not purged. Otherwise, the edited source file destructively replaces the old file. (Follows :SS in searching for duplicate file names.)

Comments

An edit list consists of one or more edit commands and, optionally, a series of associated source statement (i.e., following REPLACE, INSERT). Edit operations are executed when they are entered. When using the system console, the operator must not enter the next operation until the "@" prompt is output on the console.

All edit operations begin with a slash(/), and only the first character following the slash is required. The rest are ignored (until a comma is reached).

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, inclusively. 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. Sequence numbers refer to the original sequence of the unedited file; inserted statements cannot be referenced until the current editing process is completed and the file automatically resequenced prior to another EDIT directive.

Source statements following /REPLACE or /INSERT on the current batch device cannot contain a colon (:) in column 1, although those entered from the system console can, with the exception of :OFF and :ABORT (which are interpreted as directives instead of data). Source statements can never contain a slash (/) in the first column. Source statements on any device other than the system console and the current batch device can contain anything else in column 1 (including :OFF or :ABORT).

Input is terminated only by an /END.

If the edit file is entered on the system console and either a

PARAMETER ILLEGAL

or

NO SOURCE

error occurs, the user merely re-enters the statement in error. If the edit list is entered on any other device, the EDIT directive is aborted (if the EDIT directive was entered in keyboard mode) or the entire job is aborted (in batch mode).

EDIT OPERATIONS

/DELETE,m[,n]

Deletes source statements *m* through *n*, inclusively, from the source file. If only *m* is specified, that one statement is deleted.

/INSERT,m

Inserts the source statements in the edit list immediately following this command into the primary file following statement *m*.

/MERGE[,k],secondary file[,m[,n]]

Merges source statements from the secondary file into the primary file named in the EDIT directive.

k is the sequence number of the primary file (named in the EDIT directive) after which source statements of the secondary file are merged. If *k=0*, the secondary file source statements are merged at the beginning of the primary file; if *k* is omitted, the secondary file source statements are merged at the end of the primary file.

Secondary file is the name of the source file to be merged with the primary file. If *m* and *n* are specified, then only lines *m* through *n* of the secondary file are merged. If only *m* is specified, then only that one line is merged.

/REPLACE,m[,n]

Replaces source statements *m* through *n* (inclusively) in the primary file with source statements following the /R in the edit list. If *n* is omitted, then only statement *m* is replaced.

/SUPPRESS

Suppresses echoing of the edit operations on the system console, providing that the logical unit specified in the EDIT directive was not the system console. Normally, echoing occurs after each EDIT directive unless /S is entered.

/UNSUPPRESS

Resumes echoing of the edit operations on the system console.

/END

Terminates the edit file and returns DOS-III to its previous mode for further directives. (The last edit command must be /END.)

EXAMPLES:

If a file named SOURC contains:

```
Statement 1     ASMB,R,B,L
Statement 2             NAM START
Statement 3     A       EQU 30
Statement 4     B       EQU 20
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 100
B       NOP
/D,4
/I,6
        STA B
/E
```

then the new file SOURC equals:

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

Assume now that there exists a source file named FILE2:

```
Statement 1     ALF,ALF
Statement 2     JMP START
```

To merge FILE2 into the new SOURC, the following EDIT directive, along with its edit list, is required:

```
:ED,SOURC,5
/M,7,FILE2
/E
```

The new file SOURC looks like this:

```
Statement 1     ASMB,R,B,L
Statement 2             NAM START
Statement 3     A       EQU 100
Statement 4     B       NOP
Statement 5     START NOP
Statement 6             LDA A
Statement 7             STA B
Statement 8             ALF,ALF
Statement 9             JMP START
Statement 10            END
```

END-OF-FILE

Purpose

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

Format

:EF[,logical unit]

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

END-OF-JOB

Purpose

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

Format

:EJOB

Comments

The EJOB directive outputs a message recording the total run time of the job and execution time, then returns to keyboard mode.

If :SS condition is active, :EJOB purges temporary files on all specified *user* subchannels. If :SS condition is not active, :EJOB purges temporary files on the current user subchannel. (See STORE directive and "DOS-III Relocating Loader," Section V.) All directives except :TRACKS, :OFF, :TYPE or :BATCH are ignored until the next JOB directive.

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

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

END JOB START RUN = 0007 MIN. 52.6 SEC. EXEC = 0001 MIN. 21.0 SEC.

or

END JOB START

This message is printed on the system console and on the standard list device (logical unit 6). A top-of-form is issued on the list device prior to the message.

EQUIPMENT TABLE

Purpose

To list one or all entries in the equipment table on the system console (see Appendix A for equipment table format).

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 DVRmm d r Uu Ss

where *nn* is the decimal number of the entry
vv is the octal channel number of the device
mm is the I/O driver number for the device
d specifies DMA if equal to D, no DMA if zero
r specifies main-memory resident if equal to R, disc-resident if zero
u is a single decimal digit used for subchannel addressing
s is the availability status of the device:
0 for not busy, and available,
1 for disabled (down),
2 for busy

EXAMPLE:

Following is a listing of a DOS-III Equipment Table.

```
:EQ
EQT Ø1 CH 11 DVRØ5 Ø R UØ SØ
EQT Ø2 CH 13 DVRØ1 Ø Ø UØ SØ
EQT Ø3 CH 14 DVR31 D R UØ SØ
EQT Ø4 CH 16 DVRØ2 Ø R UØ SØ
EQT Ø5 CH 2Ø DVR12 Ø R UØ SØ
EQT Ø6 CH 21 DVR11 D Ø UØ SØ
EQT Ø7 CH 22 DVR23 D Ø UØ SØ
@
```

GO

Purpose

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

Format

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

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

Comments

When a program suspends itself (see "Program Suspension" in Section III), 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 EXEC (7)
CALL RMPAR (I)
```

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

INITIALIZE

Purpose

To label or unlabel the current user disc, and to destroy an existing System Area (and, optionally, User Area).

Format

:IN,label

where *label* is a six-character name to be written on the disc, or "*" which means unlabel the disc.

Comments

Four basic cases are possible:

1. *:IN,** An unlabeled disc (a disc containing only a User Area). The user directory and all user files are destroyed.

2. *:IN,** A labeled disc. The message

   DOS (or TSB) LABEL xxxxxx

   OR TO PURGE?

   is output. To purge both the System and User Areas, the operator must respond with

   YES

   If the existing label is SYSTEM (the disc contains a DOS or TSB system), the Override/Protect switch must be in the override position (if the disc was created using DSGEN); otherwise, a HLT 31 will occur. If the operator responds with

   NO

   the directive is ignored.

3. *:IN,label* An unlabeled disc. Only the label is changed; no files are destroyed.

4. *:IN,label* A labeled disc. The message

??? LABEL xxxxxx

OK TO PURGE?

is output. To purge an existing DOS or TSB system, move the user files to the beginning of the disc, and assign the new label to the User Area, respond with

YES

If the existing label is SYSTEM (the disc contains a DOS or TSB system), the Override/Protect switch must be in the override position (if the disc was created using DSGEN); otherwise, a HLT 31 will occur. If the operator responds with

NO

the directive is 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 a non-numeric character) which identifies the job.

Comments

When DOS-III processes the JOB directive, it issues a top-of-form to the list device (logical unit 6), prints an accounting message on the system console and the list device recording the job's *name* (as specified in the JOB directive), the date (as specified in the DATE directive), and the current time (if a Time-base Generator is present).

For example,

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

or

```
JOB START MON 6.16.9
```

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

LIST

Purpose

To list file information recorded in the user or system directories; or to list and sequentially number the contents of all or part of a source file.

Format

(System) *:LIST,X,logical unit[,file$_1$,...]*
(Unaffected by :SS)

(User) *:LIST,U,logical unit[,file$_1$,...]*
(Lists the specified directory entries from all the subchannels defined by :SS.)

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

where *X* specifies the System Area directory

U specifies a User Area directory

S specifies a user source file

logical unit specifies the list device

file$_1$, . . . names up to 13 entries to be listed (if none is specified, the entire directory is listed)

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

Comments

A top-of-form is issued to the list device prior to listing.

DIRECTORY LISTING OUTPUT

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

NAME TYPE SCTRS DISC ORG PROG LIMITS B.P. LIMITS ENTRY LIBR. P-B SUBCHAN = *n*

The following lines are then printed:

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

where *name* identifies the file,

type tells what kind of file *name* is

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

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 AB, 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 upper 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,

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

p-b is equal to T if the file is temporary and will be purged by :EJOB unless stored by :STORE,P.

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

file UNDEFINED

SOURCE LISTING OUTPUT

Each source statement is preceded by a four-digit decimal sequence number.

If the requested file is not a source file, the following message appears,

file
ILLEGAL

The list is terminated by the message

***** LIST END *****

EXAMPLES:

(on the system console:)
:LI,U,6
@

(On the list device:)

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

Note: T in the "PB" column means that the entry is temporary.

(On the system console:)

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

(On the list device:)

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

Note: "PB" no longer equals T.

(On the system console:)

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

(On the list device:)

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

LOGICAL UNIT

Purpose

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

Format

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

where n_1 and n_2 (if present) are decimal numbers.

If neither n_1 nor n_2 is present: the entire device reference table is printed.

If only n_1 is present: the equipment table entry number assigned to logical unit number n_1 is printed. (See EQUIPMENT TABLE directive.)

If both n_1 and n_2 are present (and n_2 does not equal zero): the device recorded in equipment table entry n_2 is assigned to logical unit n_1.

If both n_1 and n_2 are present (and n_2 does equal zero): the logical unit specified by n_1 becomes a null device, and any I/O request on that device is ignored.

Comments

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

If $n_2 = 0$ (that device is to be made null), the logical unit specified by n_1 may not be equal to 1, 2, 3, or the logical unit number of the current batch device.

EXAMPLE:

```
:LU
LU01  EQT03
LU02  EQT01
LU03  EQT01
LU04  EQT05
LU05  EQT04
LU06  EQT06
LU07  EQT07
LU08  EQT02
LU09  EQT00    (null device)
@
```

OFF

Purpose

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

Format

:OFF

Comments

:OFF returns the system to keyboard mode.

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

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

PAUSE

Purpose

To interrupt the current job, optionally print a comment on the system console, and return to the system console for operator action.

Format

:PAUSE [character string]

Comments

PAUSE may be entered through the keyboard even when DOS-III 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-III to the job in the previous mode.

PROGRAM

Purpose

To turn on (i.e., load from the disc and begin executing) a program from the System Area or a program from the User Area which was generated with the DOS-III Relocating Loader. (Follows the :SS condition in searching for the program.)

Format

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

where *name* denotes a system program, such as FTN for the DOS-M FORTRAN Compiler, FTN4 for the RTE/DOS FORTRAN IV Compiler, ASMB for the DOS-M Assembler, LOADR for the DOS-III Relocating Loader, or ALGOL for the RTE/DOS ALGOL Compiler.

A user program is specified via the file name assigned by the DOS-III Relocating Loader (the name specified in the program's PROGRAM, HPAL, or NAM statement).

P_1 through P_5 are optional parameters which DOS-III transfers to the program named. P_1 through P_5 must be positive integers less than 32767. The program must retrieve the parameters immediately. This procedure is described under :GO.

Comment

Consult Section V for the parameters required by FTN, FTN4, ASMB, ALGOL, and LOADR. Additional programs may be added during system generation, if desired.

Note: User programs can be run using :PROG or :RUN. :PROG is useful when the program needs parameters. DOS-III first searches the user files for the program, then the system files. :RUN is useful when an execution time limit is desired (and a Time-base Generator is present).

EXAMPLES:

:PROG,FTN,2,99

:PROG,MYFIL,0,3,84

PURGE

Purpose

To remove a user file from the user file area. (Follows the :SS condition.)

Format

:PURGE[,file$_1$,file$_2$,...]

where *file$_1$,file$_2$,...* (up to 15 file names or 72 characters per directive) designate files in the User Area. These are purged from the User Area.

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

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-III prints its name on the system console.

The presence of undefined files in the list has no effect on the purging of named (and existing) files. However, if a file cannot be found, this message is output to the system console:

file UNDEFINED

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

CAUTION: OPERATOR ATTENTION IS DISABLED DURING :PURGE.

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

RENAME

Purpose

To rename a specified user file and, optionally, change its program type. (Follows the :SS condition.)

Format

:RNAME,oldname,newname[,type]

where *oldname* is the name of the user file to be renamed

newname specifies the new name for the file

type specifies the new type for the file.

Comments

If a file name on one of the active subchannels is the same as *newname*, the message

DUPLICATE FILE NAME

is output and the file name is not changed. If the file named *oldname* cannot be found on any of the active subchannels, the message

oldname UNDEFINED

is output.

The *type* parameter must be a decimal number from 3 to 12. File types 3-5 require 11-word directory entries and types 6-12 require 5-word directory entires. If the file *type* is incompatible in this respect, a

PARAMETER ILLEGAL

message results. (File type numbers are described in Appendix A.)

Note: It is the users responsibility to insure that the format and structure of the file contents are compatible with its new file type.

REWIND

Purpose

To rewind a magnetic tape.

Format

:RWND[,logical unit]

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

RUN

Purpose

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

Format

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

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

time is an integer specifying the maximum number of minutes the program may run (default is five minutes). DOS-III ignores *time* if a Time-base Generator is not present.

N, if present, tells DOS-III 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 a program executes longer than the time limit, the current job is aborted and DOS-III 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-III ultimately to execute with other HP software not having the same request codes.

EXAMPLE:

:RUN,ROUT,15

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

SPECIFY SOURCE FILE

Purpose

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

Format

:JFILE,file

where *file* is the name of a source file on any active subchannel.

Comments

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

Only one program can be translated from a file; any statements beyond the end of the source program will be ignored. The JFILE assignment is only valid for the current job, and can be reassigned by another JFILE directive.

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

STORE

Purpose

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

Format

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

TYPE-R	*:STORE,R,file[,logical unit]*
TYPE-P	*:STORE,P[,file$_1$,file$_2$ ...]*
TYPE-S	*:STORE,S,file,logical unit [,C]*
TYPE-A	*:STORE,A,file,sectors*
TYPE-B	*:STORE,B,file,sectors*
TYPE-X	*:STORE,X,file,logical unit*

Note: Control @ should not be used in file names.

Comments

TYPE-R FILES. The directive format is

:STORE,R,file[,logical unit]

where *file* is a name consisting of five (or fewer) characters and must not duplicate another *name* already present in the user files.

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 disc if none is specified. The Job Binary Area remains as it was before the STORE,R directive.

If DOS-III 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.*

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-III Relocating Loader had relocated into executable format during the current job. A program is stored in a file of the same name. Up to 14 programs per directive are allowed. If none are specified, all programs loaded during the current job are stored. DOS-III finds these temporary programs in the user file and converts them to permanent user files by removing their "temporary" flags (see the description of the LIST,U directive).

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

EXAMPLES:

:STORE,P

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

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

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

TYPE-S FILES. The directive format is

:STORE,S,file,logical unit [,C]

where *file* is the name of the user file to be filled with source statements from the *logical unit* specified. *File* is a name of five or fewer characters, and must not duplicate a name already present in the user files. The source statement input must be terminated by a record containing a double colon (::) if the *C* option is omitted; or a triple colon (:::) if the *C* option is included. If the termination record is omitted, DOS-III stores the succeeding data on the disc as if it were source statements.

If DOS-III comes to an end-of-tape before finding the termination record (:: or :::), it outputs

DONE?

on the system console.

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

When DOS-III completes the STORE,S it outputs

nnnn LINES

where *nnnn* is the number of statements stored.

If the *C* parameter is included in the STORE directive, statements with colons in columns 1 and/or 2 are interpreted as data and transferred to the designated source file. In this case, input is terminated with a triple colon (:::). The *logical unit* specified in the STORE, S directive (when the *C* parameter is used) must not be the current batch device. If it is, DOS-III outputs the message

ILLEGAL LUN

If the user is in keyboard mode, DOS-III outputs an @ and waits for a new directive. If the user is in batch mode, a batch abort occurs.

If the *C* parameter is used and the *logical unit* specified is the system console, then all input received prior to ::: is transferred to the designated source file, except OFF and ABORT directives. If either of the two are encountered during keyboard entry, they are interpreted as directives and executed. (:OFF returns control to keyboard mode without terminating the job. :ABORT aborts the current job if the directive was entered from the keyboard, or DOS-III performs a batch abort if the STORE, S directive was entered from the batch device.) Files containing :OFF and :ABORT can be created by storing from a device other than the system console or the current batch device.

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 to reserve a file area; no data is involved.

The program must 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-III checks only that the file name exists and that it contains the sector specified.

EXAMPLE:

:STORE,A,ASCII,20

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

TYPE-X FILES. The directive format is

:STORE,X,file,logical unit

where *file* is the name of the user file to be filled with absolute binary programs from the device specified by *logical unit.*

When an end-of-tape is encountered, DOS-III outputs

DONE?

To continue loading tapes, place the next tape in the reader and type *NO;* otherwise, type *YES.*

SYSTEM SEARCH

Purpose

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

Format

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

Comments

The SS directive can only be used if it was specifically allowed during system generation. (See "Generating and Loading DOS-III," Part 3.) Otherwise, any SS directive will cause the following message:

BAD CONTROL STATE

If a file search results in the file being found, the current user subchannel is changed to the subchannel containing the file. If the file was not found, the current user subchannel is restored to its previous assignment

The LIST,U, *file* directive is an exception: this directive does not stop after it finds the file; it continues to look for duplicate entries. When the LIST search is complete, the original user subchannel is always restored.

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

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

Whenever the user subchannel assignment is changed (except by a running program through the appropriate EXEC call), the system outputs a message:

SUBCHAN = n

TOP-OF-FORM

Purpose

To issue a top-of-form command to a list device.

Format

:TOF[,logical unit]

where *logical unit* is the logical unit number of the desired list device. If *logical unit* is omitted, then logical unit 6 receives the command.

TRACKS

Purpose

To output information about the next available track on the current user disc.

Format

:TRACKS

Comments

The decimal number corresponding to the first track beyond the end of the current user area (and the number of faulty tracks encountered, if any) is output to the system console.

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

EXAMPLES:

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

```
(INPUT)    :TRACKS
(OUTPUT)   NEXT AVAIL TRACK = 0010
           @                          (End of directive processing)
```

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

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

In this example, the system reports that there are no more tracks available in the user area.

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

TYPE

Purpose

To return from batch mode to keyboard mode.

Format

:TYPE

Comments

Control is returned to the system console. :TYPE may be entered through the batch device or the keyboard device; when it is entered from the keyboard, DOS-III waits until the currently 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.

UP

Purpose

To declare an I/O device ready for use.

Format

:UP,n

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

Comments

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

I/O ERR ET EQT #n
I/O ERR NR EQT #n
I/O ERR PE EQT #n

where *ET* indicates end of tape,
NR indicates device not ready,
PE indicates parity error, and
n is the equipment table entry number.

If the incorrect *n* is entered, DOS-III outputs a list of all the down devices.

USER DISC CHANGE

Purpose

To change the subchannel assignment for the user disc.

Format

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

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

n is the new subchannel.

Comments

Discs are labeled by the INITIALIZE directive.

Each form of the UD directive has a different purpose.

EXAMPLES:

:UD (without label or subchannel)	*Interrogates the current user disc subchannel and outputs its label on the system console:* *SUBCHAN = n* *LBL = label (or UNLBL)*
:UD,,n (no label)	*If n is labeled, DOS-III outputs* *LBL = label (or UNLBL)* *No assignment is made.*
:UD, label, n	*If n is labeled with the specified label, DOS-III assigns n as the user disc. If n is unlabeled or has a different label, DOS-III outputs* *LBL = label (or UNLBL)* *Operator can then reissue :UD,label,n with the correct label.*

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

If the UD directive specifies a subchannel with an incorrect system proprietary code (see "Disc Labels" in Appendix A), DOS-III still makes the assignment, and outputs

TSB DISC or ??? DISC

If the UD directive specifies a subchannel whose system generation code does not match that of the current system disc, DOS-III still makes the assignment but outputs

DISC GEN CODE nnnn NOT SYS GEN CODE mmmm ERR POSS

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

Notes: 1. Before executing a :DD or :DD,X to a TSB or ??? DISC, the disc should be initialized with :IN,; otherwise, bad tracks may be reported erroneously.*

2. If a disc pack is changed on a DOS-III system, the subchannel assigned to that pack must be explicitly reassigned using a :UD directive or EXEC call.

SECTION III
DOS-III EXEC Calls

DOS-III EXEC calls are the line of communication between an executing program and DOS-III. An EXEC call is a block of words, consisting of an executable instruction and a list of parameters defining the request. The execution of the instruction transfers control to DOS-III. DOS-III then determines the type of request (from the parameter list) and, if it is legally specified, initiates processing of the request.

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

This section describes the basic formats of FORTRAN, ALGOL and Assembly Language EXEC calls; presents each EXEC call in detail; and concludes with a discussion of how parameters are passed to and from a program.

The EXEC calls detailed in this section are presented alphabetically, according to their function. The Request Code (RCODE) value they have in the Assembly-language calling sequence appears at the top of each page.

Note: DOS-III may include two user-created EXEC modules, loaded along with the DOS-III system EXEC modules during system generation. The purpose of the EXEC modules (called $EX36 and $EX37) and the number of parameters needed in the EXEC call are defined by the user. User EXEC module calling sequences are defined in Section XII, "User-written EXEC Modules."

ASSEMBLY LANGUAGE EXEC CALLS

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

```
EXT EXEC          (Used to link program to DOS-III)
   .
   .
   .
JSB EXEC          (Transfer control to DOS-III)
DEF *+n+1         (Defines point of return from DOS-III, n is number of
                  parameters; may not be an indirect address; must be the
                  location immediately following the last parameter
                  address)
DEF P1            (Define addresses of parameters which may occur any-
   .              where in program; may be multi-level indirect. Seven is
   .              the maximum number of allowable parameters for any
   .              EXEC call.)
DEF Pn
return point      (Continue execution of program)
   .
   .
   .
   .
   .
   .
P1 - - -          (Actual parameter values)
   .
   .
   .
Pn - - -
```

ALGOL EXEC CALLS

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

EXAMPLE:

The program below (DXFER) reads one sector from the work area and writes the information into a different location in the work area. DXFER calls EXEC through the CODE procedure EXECX (compiled externally). EXECX is compiled in the program DSKIO, although that program name is irrelevant to the linkage between DXFER and EXECX.

MAIN PROGRAM

```
HPAL,B,L,"DXFER"
BEGIN
    INTEGER ARRAY BUFFER[1:128];
    BOOLEAN READX;
    INTEGER TRACK,SECTOR;
    FORMAT F1("SOURCE TRACK,SECTOR?"),
           F2("DESTINATION TRACK,SECTOR?");
    PROCEDURE EXECX(RD,TRK,SCTR,BFR);
      VALUE RD,TRK,SCTR;
      BOOLEAN RD;
      INTEGER TRK,SCTR,BFR;
      CODE;
  WRITE(1,F1);
  READ(1,*,TRACK,SECTOR);
  READX←TRUE;
  EXECX(READX,TRACK,SECTOR,BUFFR[1]);
  WRITE(1,F2);
  READ(1,*,TRACK,SECTOR);
  READX←FALSE:
  EXECX(READX,TRACK,SECTOR,BUFFR[1]);
END$
```

PROCEDURE

```
HPAL,P,B,L,"DSKIO"
PROCEDURE EXECX(RD,TRK,SCTR,BFR);
      VALUE RD,TRK,SCTR;
      BOOLEAN RD;
      INTEGER TRK,SCTR,BFR;
BEGIN
   PROCEDURE EXEC(IO,LU,BFR,BFSZ,TRK,SCTR);
         INTEGER IO,LU,BFR,BFSZ,TRK,SCTR;
         CODE;
   INTEGER REQCD;
   IF RD THEN REQCD←1 ELSE REQCD←2;
   EXEC(REQCD,2,BFR,128,TRK,SCTR);
END;
```

FORTRAN EXEC CALLS

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

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

where P_1 through P_n are either integer values or integer variables defined elsewhere in the program.

EXAMPLE

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

} *Equivalent calling sequences*

Some EXEC call functions are generated automatically by the FORTRAN compiler or special subroutines. (Refer to "FORTRAN," in Section V and the specific EXEC calls in this section.)

BASE PAGE STORE

Purpose

To store values into base page memory locations.

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            LDA   NUMB
            LDB   ADDR
            JSB   EXEC      (Transfer control to DOS-III)
            DEF   *+2       (Point of return from DOS-III)
            DEF   RCODE     (Request code)
            return point    (Continue execution)
             .
             .
             .
RCODE       DEC   -19       (Request code = -19)
NUMB        DEC   n         (n is value to be stored)
ADDR        DEF   LOC       (LOC is a base page location)
```

FORTRAN

This feature must not be invoked by a FORTRAN program.

Comments

Base Page Store stores values into base page locations normally protected by memory protect. Prior to using the calling sequence specified above, the user loads the value to be stored into the A register and the absolute address of the base page location in the B register. Base Page Store then performs a store indirect through the B register.

CAUTION: CARE MUST BE TAKEN NOT TO MODIFY SYSTEM-ESSENTIAL BASE PAGE LOCATIONS.

FILE NAME SEARCH

Purpose

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

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            JSB   EXEC        (Transfer control to DOS-III)
            DEF   *+4 (or 5)  (Point of return from DOS-III)
            DEF   RCODE       (Request code)
            DEF   FNAME       (File name)
            DEF   NSECT       (Number of sectors)
            DEF   IPRAM       (Optional parameter)
            return point      (Continue execution)
             .
             .
             .
RCODE       DEC   18          (Request code = 18)
FNAME       ASC   3,xxxxx     (xxxxx is the file name)
NSECT       NOP               (Number of sectors returned here; 0 if not
                               found)
IPRAM       DEC   n           n = 0   user area with wait
                              n = 1   user area without wait
                              n = 2   system area with wait
                              n = 3   system area without wait
```

FORTRAN

```
DIMENSION NAME (3)      (File name)
IPRAM     = 2           (System search, with wait)
IRCDE     = 18          (Request code)
NAME (1)  = xxxxxB      (First two characters)
NAME (2)  = xxxxxB      (Next two characters)
NAME (3)  = xxxxxB      (Last character and blank)

CALL EXEC (IRCDE, NAME, ISECT, IPRAM)
```

Comments

File searches can be performed on either the system or user area, with or without wait, according to the value of IPRAM. If IPRAM is omitted, the search is performed on the user area with wait. If the search is requested *with* wait, the A register contains the track/sector address of the file, and the B register contains the memory address of the track/sector address, upon return to the user program.

FILE READ/WRITE

Purpose

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

Assembly Language

```
            EXT  EXEC
             .
             .
            JSB  EXEC          (Transfer control to DOS-III)
            DEF  *+7 (or 8)    (Point of return from DOS-III)
            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)
            DEF  IPRAM         (Area which could have been legally transferred
                               if an overflow occurred-optional parameter)
            return point       (Continue execution)
             .
             .
RCODE       DEC  14 or 15      (Request code:  14 = read, 15 = write)
CONWD       OCT  conwd         (See Comments, I/O 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)
IPRAM       NOP                (Optional parameter; see Comments)
```

FORTRAN

```
DIMENSION NAME (3), IBUF(10)
NAME(1) = xxxxxB          (First two characters of file name)
NAME(2) = xxxxxB          (Second two characters)
NAME(3) = xxxxxB          (Last character and blank)
ICRDE   = 14 (or 15)      (Request code)
ICON    = conwd           (See comments)
IRSCT   = 0               (Relative sector number)

CALL EXEC (IRCDE, ICON, IBUF, 10, NAME, IRSCT, IPRAM)
                          or
CALL EXEC (IRCDE, ICON, IBUF, 10, NAME, IRSCT)
```

Comments

See the Comments under I/O READ/WRITE EXEC call (RCODE = 1 or 2) for a description of the *conwd* fields needed in the above calling sequences.

To read or write on the *m*th sector of a file, set RSECT = *m*−1. To determine the size of a file, use the FILE NAME SEARCH EXEC call (RCODE = 18).

Data files to be written (or read) should be created with a STORE directive before executing the EXEC call.

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

If the DOS-III installation is likely to have more than one user disc, the program should use the USER DISC CHANGE EXEC call (RCODE = 23) without a subchannel specified to check whether the correct user disc is currently assigned. Alternatively, the user can use an SS directive to set up a system search condition for referencing files on many subchannels.

This call provides an optional parameter, IPRAM, to provide the user with information concerning a file read/write overflow (where the buffer length exceeds the sector contents). If IPRAM is omitted, an overflow causes an IT error. If IPRAM is included and an overflow occurs, control is returned to the user program with IPRAM set equal to the number of words (+) or characters (−) (as defined by BUFFL) that could legally have been transferred. If an overflow occurs, no disc transfer takes place, whether IPRAM is included or not. If IPRAM is included and no overflow occurs, the value of the parameter is set to zero.

I/O CONTROL

Purpose

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

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            JSB   EXEC          (Transfer control to DOS-III)
            DEF   *+4 (or 3)    (Point of return from DOS-III)
            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 specific FORTRAN auxiliary I/O statements (see Comments) or an EXEC calling sequence.

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

Comments

CONWD

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

	0	0	W	FUNCTION CODE (see below)						LOGICAL UNIT NUMBER					
BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

WAIT FIELD (W)

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

If W = 0, DOS-III waits until the control request is complete before returning.

FUNCTION CODE FIELD

Function Code (Octal)	Action
000	Clear the device
001	Write end-of-file (magnetic tape)
002	Backspace one record (magnetic tape)
003	Space forward one record (magnetic tape)
004	Rewind (magnetic tape)
005	Rewind standby (magnetic tape)
006	Dynamic status (magnetic tape)
007	Set end-of-paper tape
010	Generate paper tape leader
011	List output line spacing (PARAM or IPRAM required)
012	Write file gap (magnetic tape)
013	Space forward one file (magnetic tape)
014	Backspace one file (magnetic tape)

LOGICAL UNIT FIELD

This field specifies the logical unit number of the device which is to receive the control request.

Note: 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 the number of lines to be spaced on the System Console. For details of line printer formatting, see "Line Printer Formatting," in Section IV.

Compiler Considerations

Within FORTRAN and ALGOL programs, various control operations for magnetic tape may be performed by the following auxiliary I/O statements:

BACKSPACE
ENDFILE
REWIND

Refer to the appropriate compiler manual for a detailed description of these statements.

I/O READ/WRITE

Purpose

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

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            JSB   EXEC        (Transfer control to DOS-III)
            DEF   *+5 (or 7)  (Point of return from DOS-III; 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)    (Request code: 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

```
DIMENSION IBUF (100)        (Define buffer)
IRCDE = 1 (or 2)            (Request code)
ICON = conwd                (see Comments)
IBUFL = 100                 (Buffer length in words)
ITRAK = 150                 (Disc track number)
ISECT = 0                   (Disc sector number)

CALL EXEC (IRCDE, ICON, IBUF, IBUFL, ITRAK, ISECT)   for disc transfers
CALL EXEC (IRCDE, ICON, IBUF, IBUFL)                 for non-disc transfers.
```

Comments

CONWD

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

	0	0	W			A		K	V	M	LOGICAL UNIT #					
BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

FIELD	FUNCTION
W	If 1, tells DOS-III to return to the calling program after starting the I/O transfer. If W = 0, DOS-III waits until the transfer is complete before returning.
A	When transferring variable length binary records (M = V = 1), A = 1 indicates absolute binary format.
K	Used with keyboard input, specifies printing the input as received if K = 1. If K = 0, "no printing" is specified.
V	1) When reading variable length records from punched tape devices in binary format (M = 1), if V = 0 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 read in. 2) When outputting ASCII records to a list device (M = 0), if V = 0 the first character in the buffer is interpreted as a carriage control character (see Section IV). If V = 1, single spacing occurs, and the entire buffer (including the first character) is output to the list device.
M	Determines the mode of data transfer. If M = 0, transfer is in ASCII character format, and if M = 1, binary format.

"Waiting and No Waiting"

If the program requests the "waiting" option in the *conwd* (W = 0), DOS-III will return the transmission log in the B register upon completion. (The transmission log is a positive number, representing the number of words or characters transmitted, depending upon which was originally requested.)

If the program requests the "no waiting" option in the *conwd* (W = 1), it can check for the completion of the I/O operation with the I/O STATUS EXEC call (RCODE = 13). When the operation is complete (STATS $\geqslant$ 0), the transmission log can be retrieved from the TLOG parameter.

Notes: When using "no waiting" I/O and loading program segments:

1. *Under :RUN, DOS-III waits for all I/O to complete before loading the segment.*
2. *Under :PROG, DOS-III does not wait.*

If a read or write is issued to a disc address that does not lie in the Work Area, the message *IT nnnnn* is output and the program is terminated.

Compiler Considerations

Within FORTRAN and ALGOL programs, I/O transfers to standard devices are programmed by the READ and WRITE statements.

I/O transfers to the Work Area and the disc may be done through the BINRY library routine. 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 number, track number, sector number, and offset in words within the sector. (If the offset equals 0, the transfer begins on the sector boundary. If the offset equals N, then N words of the sector are skipped before starting the transfer.) BINRY has two entry points, BREAD and BWRIT, for read and write operations respectively. An example below gives the calling procedure.

```
DIMENSION IBUF(10), BUF(20)
LUN = 2
ITRK = 120
ISECT = 36
IOFF = 0
CALL BREAD (BUF,40, LUN, ITRK, ISECT, IOFF)
                        or
CALL BWRIT (IBUF, 10, LUN, ITRK, ISECT, IOFF)
```

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-III)
            DEF  *+5           (Point of return from DOS-III)
            DEF  RCODE         (Request code)
            DEF  LUN           (Logical unit)
            DEF  STATS         (Status returned)
            DEF  TLOG          (Transmission log returned)
            return point       (Continue execution)
             .
             .
             .
RCODE       DEC  13            (Request code = 13)
LUN         DEC  n             (Logical unit number)
STATS       NOP                (Status returned here)
TLOG        NOP                (Transmission log returned here)
```

FORTRAN

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

Comments

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

MEMORY PROTECT CONTROL

Purpose

To enable or disable the memory protect option from a user program.

CAUTION: THE SYSTEM IS NOT PROTECTED WHEN MEMORY PROTECT IS IS DISABLED.

Assembly Language

```
            EXT   EXEC
              .
              .
              .
            JSB   EXEC      (Transfer control to DOS-III)
            DEF   *+3       (Point of return from DOS-III)
            DEF   RCODE     (Request code)
            DEF   MPTK      (Define the memory protect parameter flag)
            return point    (Continue execution)
              .
              .
              .
RCODE       DEC 30          (Request code = 30)
MPTK        DEC n           (If n = 0, memory protect is activated, and
                             is activated following any interrupt
                             completion. If n ≠ 0, then memory protect
                             is deactivated and remains off after
                             interrupt completion)
```

FORTRAN

```
IRCDE = 30
MPTK = 0 (or 1)
CALL EXEC (IRCDE,MPTK)
```

Comments

Any program termination, either normal or aborted, enables memory protect. Program segments can make memory protect EXEC calls to turn memory protect on or off, but calling and exiting from segments has no effect on memory protect settings.

PROGRAM COMPLETION

Purpose

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

Note: Every program must terminate and return to DOS-III using this EXEC call, whether the EXEC call is explicitly coded or indirectly generated by a compiler.

Assembly Language

```
        EXT  EXEC
          .
          .
        JSB  EXEC          (Transfer control to DOS-III)
        DEF  *+2           (Define end of parameter list)
        DEF  RCODE         (Request code)
          .
          .
          .
RCODE   DEC  6             (Request code = 6)
```

FORTRAN

```
IRCDE = 6
CALL EXEC (IRCDE)
```

Compiler Considerations

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

PROGRAM LOAD

Purpose

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

Assembly Language

```
        EXT  EXEC
          .
          .
        JSB  EXEC          (Transfer control to DOS-III)
        DEF  *+3 (to 8)    (Determine number of parameters)
        DEF  RCODE         (Request code)
        DEF  PNAME         (Program name)
        DEF  PRAM1         (First optional parameter)
          .
          .
          .
        DEF  PRAM5         (Fifth optional parameter)

RCODE   DEC  10
PNAME   ASC  3,xxxxx       (Program name)
PRAM1   ---                (Up to 5 words of parameter information
                            passed to the program. See "Parameter
                            Processing" at the end of this section.
```

FORTRAN

```
DIMENSION NAME(3)          (Program name)
IRCDE = 10
NAME(1) = xxxxxB           (First two characters)
NAME(2) = xxxxxB           (Next two characters)
NAME(3) = xxxxxB           (Last character and blank)
CALL EXEC (IRCDE,NAME[,p1...])
```

Comments

During main program loading, the system interrogates a system flag called AEPF (location 135_8). This flag is normally zero unless specifically set by a user program. If AEPF is not zero, the contents of AEPF are treated as an alternate entry point address. The system transfers control to the alternate entry point by performing a JMP AEPF,I (jump indirect). AEPF is then cleared. If AEPF = 0, control transfers to the program main entry point.

The Assembly language user can alter the contents of AEPF (and any other base page location) by using the BASE PAGE STORE EXEC call (RCODE = -19).

PROGRAM SUSPENSION

Purpose

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

Assembly Language

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

FORTRAN

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

Comments

DOS-III prints a message on the system console when it processes the PROGRAM SUSPENSION EXEC call:

name SUSP

When the operator restarts the program with a :GO, up to five parameters may be passed to the suspended program. (See "Parameter Processing" at the end of this section.)

Compiler Considerations

The FORTRAN and ALGOL compilers automatically generate a PROGRAM SUSPENSION EXEC call when they compile a PAUSE statement.

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 V, "DOS-III Subsystems," for information on segmented programs.) Follows the :SS condition.

Assembly Language

```
            EXT  EXEC
             .
             .
             .
            JSB  EXEC          (Transfer control to DOS-III)
            DEF  *+3 (to 8)    (Determine number of parameters)
            DEF  RCODE         (Request code)
            DEF  SNAME         (Segment name)
            DEF  PRAM1         (First optional parameter)
             .
             .
             .
            DEF  PRAM5         (Fifth optional parameter)
             .
             .
             .
RCODE       DEC  8             (Request code = 8)
SNAME       ASC  3,xxxxx       (xxxxx is the segment name)
PRAM1       ---                (Up to 5 words of parameter information
PRAM5       ---                passed to the segment. See "Parameter
                                Processing" at the end of this section.)
```

FORTRAN

```
DIMENSION NAME (3)                    (Segment name)
IRCDE = 8
NAME (1) = xxxxxB                     (First two characters)
NAME (2) = xxxxxB                     (Next two characters)
NAME (3) = xxxxxB                     (Last character and blank)
CALL EXEC (IRCDE, NAME [,p1...] )
```

Comments

In the FORTRAN or ALGOL calling sequence, the user must convert the name of the segment from ASCII to octal and store it in the NAME array, two characters per word. The RTE/DOS FORTRAN IV Compiler, however, can convert this automatically through Hollerith constants.

During program segment loading, the system interrogates a system flag called AEPF (location 135_8). This flag is normally zero unless specifically set by a user program. If AEPF = 0, control transfers to the program segment main entry point. If AEPF is not zero, the contents of AEPF are treated as an alternate entry point address. The system transfers control to the alternate entry point by performing a JMP AEPF,I (jump indirect). AEPF is then cleared. (The Assembly language user can alter the contents of AEPF (and any other base page location) by using the BASE PAGE STORE EXEC call (RCODE = -19).)

See "Segmented Programs," in Section V, for a description of segmented programs.

SEGMENT RETURN

Purpose

To return control from a segment to the main program at the instruction immediately following the program segment load call. (This provides a subroutine-like return from a segment to a main program.)

Assembly Language

```
            EXT   EXEC
              .
              .
              .
            JSB   EXEC          (Transfer control to DOS-III)
            DEF   *+2 (to 7)    (Point of return from DOS-III)
            DEF   RCODE         (Define the request code)
            DEF   PRAM1         (Define the first parameter)
              .
              .
              .
            DEF   PRAM5         (Define the fifth optional parameter)
              .
              .
              .
RCODE       DEC 29              (Request code = 29)
PRAM1       ...                 (Up to five words of parameter information
  .                              are passed from the segment to the main
  .                              program. See "Parameter Processing" at
                                 the end of this section)
PRAM5       ...
```

FORTRAN

```
IRCDE = 29
CALL EXEC (IRCDE [,P1, . . . ,P5])
```

TIME REQUEST

Purpose

To request the current time.

Assembly Language

```
        EXT  EXEC
          .
          .
          .
        JSB  EXEC          (Transfer control to DOS-III)
        DEF  *+3           (Point of return from DOS-III)
        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

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

Comments

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

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

If DOS-III does not contain Time-base Generator, all values in the time array are set to zero.

WORK AREA LIMITS

Purpose

To ascertain the first and last tracks of the Work Area on the system or current user disc and the number of sectors per track.

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            JSB   EXEC          (Transfer control to DOS-III)
            DEF   *+5 (or 6)    (Point of return from DOS-III)
            DEF   RCODE         (Request code)
            DEF   FTRAK         (First track)
            DEF   LTRAK         (Last track)
            DEF   SIZE          (Number of sectors/track)
            DEF   DISC          (Optional parameter — see Comments)
            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)
DISC        DEC   n             (n = 0 for system disc; n ≠ 0 for current user disc)
```

FORTRAN

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

Comments

This call returns the limits of the Work Area, which is that area of the system or user disc which programs use for temporary storage with the I/O READ/WRITE EXEC call (RCODE = 1 or 2). If the DISC parameter is omitted from the calling sequence, or if DISC = 0, the system disc information is returned. If DISC ≠ 0, user disc information is returned.

WORK AREA STATUS

Purpose

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

Assembly Language

```
        EXT   EXEC
         .
         .
         .
        JSB   EXEC      (Transfer control to DOS-III)
        DEF   *+5       (Point of return from DOS-III)
        DEF   RCODE     (Request code)
        DEF   NTRAK     (Number of tracks desired)
        DEF   TRACK     (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, 0 if n tracks
                         not available)
```

FORTRAN

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

Comments

This call is used with the WORK AREA LIMITS EXEC call (RCODE = 17) to establish the nature of the Work Area. The READ/WRITE EXEC call (RCODE = 1 or 2) then transmits information to and from this area, using the track numbers determined by this call. DOS-III handles track switching automatically.

DOS-III checks whether there are *n* consecutive tracks starting at the track specified. If *n* tracks are available, DOS-III returns the starting track number to the program. If DOS-III does not locate *n* consecutive tracks, it returns 0 in STRAK or ISTRK.

USER DISC CHANGE

Purpose

To change the subchannel assignment for the user disc.

Assembly Language

```
            EXT   EXEC
             .
             .
             .
            JSB   EXEC          (Transfer control to DOS-III)
            DEF   *+3 (or 4)    (Point of return from DOS-III)
            DEF   RCODE         (Request code)
            DEF   LABEL         (Disc label)
            DEF   SUBCH         (Disc subchannel; optional)
            return point        (Continue execution)
             .
             .
             .
RCODE       DEC   23            (Request code = 23)
LABEL       ASC   3,xxxxxx      (Label = xxxxxx)
SUBCH       DEC   (0 to 7)
```

FORTRAN

```
DIMENSION LABEL (3)          (New label)
IRCDE    = 23
LABEL (1) = xxxxxB           (First two characters)
LABEL (2) = xxxxxB           (Next two characters)
LABEL (3) = xxxxxB           (Last two characters)
ICHNL     = M                (0 through 7)

CALL EXEC (IRCDE, LABEL, ICHNL)
                or
CALL EXEC (IRCDE, LABEL)
```

Comments

If both the label and subchannel are specified, DOS-III checks whether the subchannel has that label. If it does, the assignment is made and DOS-III returns. If not, DOS-III outputs

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

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

:GO

to return to the *beginning* of the EXEC call (not the normal return point) so that the program can reissue the EXEC call. If the operator does not have a properly labeled disc (or the subchannel is a permanent disc), he should use :OFF or :ABORT.

If only a label is specified, DOS-III searches for the label, starting with the highest subchannel. If DOS-III finds the label, it makes the assignment. If DOS-III cannot find the label, it suspends the program and outputs

DISC NOT ON SYS
UD nnnnn
xxxxx SUSP

The operator can then abort the program or load a properly labeled disc then input

:GO

to return to the *beginning* of the EXEC call.

If the label equals "*" and a subchannel is specified, DOS-III checks whether the subchannel is unlabeled. If it is, DOS-III makes the assignment. If the subchannel is labeled, DOS-III suspends the program and outputs

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

The operator can then abort the program or load an unlabeled disc on the proper channel then input

:GO

to return to the *beginning* of the EXEC call.

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

DISC NOT ON SYS
UD nnnnn
xxxxx SUSP

The operator can then abort the program or load an unlabeled disc then input

:GO

to return to the beginning of the EXEC call.

Notes: 1. *If the EXEC call specifies a subchannel with an incorrect system proprietary code (see Appendix A), DOS-III still makes the assignment but outputs*
TSB DISC or ??? DISC

2. *If the EXEC call specifies a subchannel whose system generation code (see Section VII) does not match that of the system disc, DOS-III still makes the assignment, but outputs*
DISC GEN CODE nnnn NOT SYS GEN CODE mmmm ERR POS

3. *The changes made by this EXEC call are only temporary, and will be reset at the end of each job to the user subchannel specified during system generation.*

4. *If the specified subchannel is not active (physically present), DOS-III suspends the programaand outputs*
I/O ERR NR USER DISC
or
I/O ERR PE USER DISC
UD nnnnn *(nnnnn = address of EXEC call)*
xxxxx SUSP

PARAMETER PROCESSING

Certain user programs require parameters for their execution. DOS-III allows passing of parameters in the following environments:

(1) from a main program to a main program

(2) from a main program to a segment

(3) from a segment to a main program

(4) from a user to a suspended program

Parameter transferral from program to program (1-3) is handled programmatically by specifying parameters in an EXEC calling sequence. Parameter transferral from a user directly to a program (4) is handled by passing parameters back to the suspended program through the GO directive.

All the programs receiving parameters retrieve them in the same way. The parameters to be passed (if any) are located in the base page parameter buffer RONBF (see Appendix A). In the Assembly language environment, the B register contains the address of the parameter buffer. In the FORTRAN/ ALGOL environment, a library routine (RMPAR) is provided to transfer parameters to a user-defined buffer. (This call must be the first statement executed upon entry.)

ASSEMBLY LANGUAGE EXAMPLE

```
            EXT   EXEC
              .
              .
              .
            JSB   EXEC          (Call EXEC to suspend program)
            DEF   *+2
            DEF   RCODE
            LDA   B,I           (Get parameter from GO directive)
            SZA,RSS
            JMP   NOPAR
              .
              .
              .
RCODE       DEC 7
B           EQU 1
```

FORTRAN EXAMPLE

```
DIMENSION I(5)          (Define user parameter buffer)
CALL EXEC (7)           (Suspend program)
CALL RMPAR (I)          (Get parameters from :GO)
```

SECTION IV
Input/Output

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

Note: Refer to Section XIV, "Privileged Mode," for a discussion of privileged mode processing.

USER PROGRAM I/O

The user program requests I/O by means of an EXEC call (see Section III) which specifies the logical unit, control information, type of operation, buffer location and buffer length.

All references to I/O devices are made through logical unit numbers. This relieves the programmer of the burden of knowing which physical device or which I/O channel is actually going to perform the I/O transfer.

DOS-III has the following standard function assignments for logical unit numbers:

	Logical Unit Number	Function
Restored after each :JOB.	1	System console
	2	System mass storage
	3	User mass storage
	4	Standard punch device
	5	Standard input device
	6	Standard list device
	7	Unassigned
	8	Recommended for magnetic tape
	9	Can be assigned to any device
	10	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 established at system generation (see Section X), whereas 10 through 63 are restored only on a start-up from the disc.

SYSTEM I/O PROCESSING

System I/O processing is controlled by three I/O tables:

1) Equipment Table (EQT) — which records all devices, I/O channels, driver entry points, DMA requirements, and disc location (if disc-resident).

2) Logical Unit Table (LUT) — which assigns an equipment table number to each of its entries, thus allowing the programmer to reference changeable logical units instead of fixed physical units.

3) Interrupt Table (INT) — which relates each I/O channel to its corresponding equipment table entry.

For a detailed description of these tables see Appendix A.

When the system recognizes an EXEC call that performs I/O, the request is sent to the I/O supervisor EXEC module ($EX18). $EX18 determines if the driver for the requested device is main-memory resident; if not, the driver is loaded into main memory from the disc. Once the driver is in main-memory, the addresses of its EQT entries are placed in the base page communication area and control is transferred to the driver's initiation section. After the driver initiates the I/O operation, it returns to $EX18. If the I/O was requested "without wait", DOS-III immediately returns control to the user program; if the I/O was requested "with wait", DOS-III waits until the I/O transfer is complete before returning to the user program.

Once a driver has been initiated, interrupts from the device are channeled through a central interrupt processing routine ($CIC). (All interrupt locations in main memory contain a JSB $CIC.) $CIC determines which device interrupted, resets the addresses of the EQT entries into the base page communication area (if necessary), and transfers control to the driver's continuation section. The driver either continues or completes the I/O operation, and control is then returned to the executing user program.

INPUT/OUTPUT DRIVERS

The I/O driver routines, either main-memory or disc-resident, handle the actual transfers of information between the computer and external devices. They are responsible for initiating and continuing operations on all devices of equivalent type. When a transfer is initiated, DOS-III places the EQT entry addressed into the base page communication area and executes a subroutine jump 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-III. When an interrupt occurs on the channel, indicating continuation or completion of the transfer, DOS-III again transfers control to the driver. DOS-III requires only two drivers: the Moving-Head Disc Driver (DVR31) and the System Console Driver (DVR05). However, these additional drivers are fully compatible with DOS-III:

DVR00	—	Teleprinter
DVR01	—	Photoreader
DVR02	—	High speed punch
DVR10	—	Plotter
DVR11	—	Card Reader
DVR12	—	Line Printer
DVR15	—	Mark Sense Card Reader
DVR23	—	7970 Magnetic Tape
DVR26	—	2762A Terminal Printer
DVR33	—	Writable Control Store

The driver name consists of the letters "DVR" prefixed to the equipment type code. In addition, the programmer can write drivers for special devices, following the guidelines in Section XIII, "Planning I/O Drivers." The driver is only responsible for updating the status field in the EQT entry; DOS-III handles the availability field.

SPECIAL DRIVER CONSIDERATIONS

Some devices require special considerations while processing, particularly line printers and magnetic tape drivers.

Line Printer Formatting

When a user program makes a I/O READ/WRITE EXEC call to the line printer, the line printer driver (DVR12) checks bit 7 of the CONWD in the calling sequence. If bit 7 = 1, the line printer single-spaces each line and all characters in the output buffer are printed. If bit 7 = 0, the first character of the output buffer determines the carriage control and is printed as a space.

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

The control characters have the following meaning:

Character	Meaning
blank	Single space (print on every line)
0	Double space (print on every other line)
1	Eject page
*	Suppress space (overprint next line)
others	Single space

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

Note: DVR12 returns top-of-form status (status word bit 6 = 1) whenever top-of-form is executed.

When a user program makes an EXEC call for I/O CONTROL (RCODE = 3) with the function bits in the CONWD (or the ICNWD) set to 011_8 (see Section III), the optional parameter PARAM (or IPRAM) word defines the format action to be performed by the line printer:

Parameter Word (Dec)	Meaning
< 0	Page Eject
0	Suppress space on the next print operation only
1 to 55	Space 1 to 55 lines, ignoring page boundaries
56	Single space with automatic page eject
57	Skip to next even line with automatic page eject
58	Skip to next triple line with automatic page eject
59	Skip to next 1/2 page boundary
60	Skip to next 1/4 page boundary
61	Skip to next 1/6 page boundary
62	Skip to bottom of the page
63	Skip to top of next page
64	Set automatic page eject mode
65	Clear automatic page eject mode

Note: The automatic page eject mode is not supported on the 2767 line printer.

Automatic Page Eject

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

Magnetic Tape Usage

Input/output transfers to and from a HP 7970 magnetic tape unit can be programmed using the standard I/O READ/WRITE EXEC call (RCODE = 1 or 2). When specifying the data buffer length, the programmer must know that a buffer length of zero (0) causes the driver to take no action on a write or an ASCII read. Only the amount of data that fits within the buffer is transmitted to the user on read. A zero (0) buffer length on binary read causes a forward skip of one record.

In the I/O STATUS EXEC call (RCODE = 13), bits 7-0 of the 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
0	Tape unit busy, or in local mode

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

The maximum buffer length is 16,384 words.

Magnetic Tape Error Recovery

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-III outputs

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. A backward motion request clears the end-of-tape status.

SECTION V
DOS-III Subsystems

This section describes conventions for using the following DOS-III subsystems:

- ALGOL Compiler
- Assembler
- FORTRAN and FORTRAN IV Compilers
- Relocating Loader
- Relocatable libraries, including the DEBUG subroutine

and concludes with a discussion of program segmentation.

SOURCE PROGRAM FILES

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

LOAD-AND-GO FACILITY

DOS-III 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. Then separate directives initiate loading (PROG, LOADR) and execution (RUN,*program*).

DOS-III can store the object code of several programs and associated segments and subroutines on the disc. The Relocating Loader retrieves them from the disc, and relocates them into executable absolute program units.

ALGOL COMPILER

The ALGOL Compiler consists of a main program and a data segment which operate under the control of DOS-III. The compiler resides on the disc and is read into main memory when called for by a PROG directive.

Source programs written in ALGOL are accepted either from an input device or from a user disc file and are translated by the ALGOL Compiler into relocatable object programs optionally punched on paper tape (and optionally stored in the Job Binary Area of the disc). The object program can be loaded using the DOS-III Relocating Loader and executed using the RUN or PROG directive.

ALGOL I/O

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

Compiler Operation

The ALGOL Compiler is initiated with a PROG directive, and inputs the source program from an input device, or, if from a source file, from a file specified by a JFILE directive. The PROG directive for the ALGOL Compiler should take the following form:

PROG,ALGOL

:PROG,ALGOL[,P_1,P_2,P_3,P_4,99]

where P_1 = logical unit number of input device (default is 5; set to 2 for source file input indicated by a JFILE directive)

P_2 = logical unit number of list device (default is 6)

P_3 = logical unit number of punch device (default is 4)

P_4 = lines/page on the source listing (default 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.)

All parameters are optional. If p_1 through p_4 are not present, the default operations are assumed. If 99 is not present, the binary output is not placed in the Job Binary Area.

Messages During Compilation

When the end of a source tape is encountered, the following is output on the system console:

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 the compilation, the following message is output to the system console:

$END, ALGOL

If the Job Binary Area (where binary code is stored because of a.99 parameter in the PROG directive) overflows, the following message is output and compilation continues:

JBIN OVF

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

The compiler terminates if

- Logical unit 2 has been given for input and no :JFILE has been declared. The following message is output:

  NO SOURCE

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

  HPAL??

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

  IE nnnnn

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

Language Considerations

The HP ALGOL control statement has this format:

HPAL [,L,A,B,P], "name" [,P_1] [,P_2]

where *HPAL* is mandatory

L,A,B,P are symbols (any combination is allowed) representing:

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

"name" is the program name (the quotes and a program name are mandatory)

P_1 is a decimal digit between 0 and 9 specifying the name of the error routine to be called if an error occurs in ALOG, SQRT, .RTOR, SIN, COS, .RTOI, EXP, .ITOI, TAN. The name of the error routine is ERR*n*, where $n = P_1$, or $n = 0$ if P_1 is not specified. ERR0 is supplied in the Relocatable Library; all other error routines must be supplied by the user.

P_2 is a decimal digit specifying the type of the program: 3 for a main program, 5 for a segment, and 6 or 7 for a utility subroutine or procedure. If P_2 is not specified, the type is set to 3 for main programs and to 7 for procedures (P option in the control statement).

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

Sense switch control is not used with DOS-III.

EXAMPLE

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

ASSEMBLER

The Assembler, a segmented program that executes in the main-memory User Program Area, operates under control of DOS-III. The Assembler consists of a main program (ASMB) and six segments (ASMBD, ASMB1, ASMB2, ASMB3, ASMB4, ASMB5), and resides on the disc. The main program is read into main memory when called by a PROG directive.

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-III. ASMB can store relocatable code in the Job Binary Area of the disc for on-line execution, as well as punch it on paper tape.

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

Assembler I/O

The Assembly Language I/O EXEC calls should specify the proper logical unit numbers for the DOS-III configuration. (See Section IV.)

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

If the memory protect hardware option is present (and enabled), it protects the resident supervisor from alteration. It 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 register.
- The halt instruction.

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

Assembler Operation

The DOS-III Assembler is initiated with 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 $p_1 = 2$ in the PROG directive. The PROG directive for Assembler should take the following form:

PROG,ASMB

:PROG,ASMB[,P_1,P_2,P_3,P_4,99]

where P_1 = logical unit number of input device (default is 5; set to 2 for source file input indicated by a JFILE directive)

P_2 = logical unit number of list device (default is 6)

P_3 = logical unit number of punch device (default is 4)

P_4 = lines/page on the source listing (default 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.)

All parameters are optional. If p_1 through p_4 are not present, the default operations are assumed. If 99 is not present, the binary output is not placed in the Job Binary Area.

Messages During Assembly

When the end of a source tape is encountered, the following is output on the system console:

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.

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

$END ASMB

If another pass of the source program is required, this message is output at the end of pass one.

$END ASMB PASS

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

:GO

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

$END ASMB CS

and the current assembly stops.

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

$END ASMB XEND

and the current assembly stops.

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

$END ASMB NPRG

and the current assembly stops.

If the Job Binary Area, where binary code is stored by a 99 parameter, overflows, assembly continues but the following message is output on the system console:

JBIN OVF

However, no further binary code is stored in the Job Binary Area.

The next message is printed on a separate line just above 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 (printed on a separate line just above each diagnostic):

PG *ppp*

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

Language Considerations

ASSEMBLER CONTROL STATEMENT. Although only relocatable code can be run under DOS-III, the DOS-III 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" parameter. The "R" parameter, 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 statement allows up to eight optional parameters. Only the first parameter is significant in DOS-III. If the first parameter equals 3, the program is a main program; if 4, a disc-resident device driver; if 5, a program segment; if 6, a library routine; if 7, a subroutine. If the parameter equals any other number, the assembler and DSGEN will accept it, but the Relocating Loader will not. (See Appendix A for program type codes.)

NAM name[,type]

where *name* is the name of the program (it should not equal any file name),
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-III communication, while the *entry points* are program-to-program communication.

Note: DOS-III Assembly language does not contain the ORB statement, since information cannot be directly 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.

FORTRAN COMPILERS

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

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

FORTRAN I/O

FORTRAN I/O statements should specify the proper logical unit numbers for the DOS-III configuration. (See Section IV.)

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

Compiler Operation

The FORTRAN compilers are initiated with a PROG directive, and input the source program from an input device, or, if from a source file, from a file specified by a JFILE directive. The PROG directive for FORTRAN compilers should take the following form:

PROG,FTN[4]

:PROG,FTN[,P_1,P_2,P_3,P_4,99]

:PROG,FTN4[,P_1,P_2,P_3,P_4,99]

P_1 = logical unit number of input device (default is 5; set to 2 for source file input indicated by a JFILE directive)

P_2 = logical unit number of list device (default is 6)

P_3 = logical unit number of punch device (default is 4)

P_4 = lines/page on the source listing (default 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.)

All parameters are optional. If p_1 through p_4 are not present, the default operations are assumed. If 99 is not present, the binary output is not placed in the Job Binary Area.

Messages During Compilation

When the end of a source tape is encountered, the following is output on the system console:

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 output on the system console:

$END, FTN[4]

If the Job Binary Area (where binary code is stored because of a 99 parameter in the PROG directive) overflows, the following message is output and compilation continues:

JBIN OVF

There is no further loading into the Job Binary Area.

The compiler terminates if

- logical unit 2 has been given for input and no JFILE has been declared. ($END,FTN[4] is not output.)
- There are not enough work tracks for the compiler. The following message is output:

  # TRACKS UNAVAILABLE

- A colon occurs 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 output.

  IE nnnnn

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

Language Considerations

FORTRAN CONTROL STATEMENT. Besides the standard options described in the FORTRAN manual, two compiler options, T and *n*, are available. A "T" lists the symbol table for each program in the compilation. If a "*u*" follows the address of a variable, that variable is undefined (the program does not assign a value to it). The A option includes this T option. If *n* appears, *n* is a decimal digit (1 through 9) which specifies an error routine. The user must then supply an error routine, ERR*n*. If this option does not appear, the standard library error routine, ERR0, is used. The error routine is called when an error occurs in ALOG, SQRT, .RTOR, SIN, COS, .TROI, EXP, .ITOI, or TAN.

Extended and Auxiliary Statements

In addition to the standard FORTRAN statement, the FORTRAN compiler running under DOS-III supports the following extensions and additions:

1. extended PROGRAM statement
2. additional DATA statement
3. additional EXTERNAL statement

Execution of the following two FORTRAN statements results in special processing in the DOS-III environment:

1. PAUSE
2. STOP

PROGRAM STATEMENT

The program statement includes an optional type parameter.

PROGRAM name [,type]

name is the five-character name of the program (and its main entry point). When the program is executed using a RUN or PROG 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-III. The type is set to 3 if not given.

DATA STATEMENT

The DATA statement sets initial values for variables and array elements. The format of the DATA statement is

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

where k is a list of variables and array elements separated by commas, d is a list of (optionally signed) constants, separated by commas and optionally preceded by $j*$ (j is an integer constant).

The elements of d_1 are serially assigned to the elements of k_1. The form $j*$ means that the constant is assigned j times. The k_1 and d_1 must correspond one-to-one.

Elements of k_1 must not be from COMMON.

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

EXAMPLE

```
DIMENSION A(3), I(2)

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

EXTERNAL STATEMENT

With the EXTERNAL statement, 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 is the entry point of a function, subroutine, or library program.

EXAMPLE

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

ERROR E-0018 means too many externals.

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

PAUSE AND STOP

PAUSE causes the following message to be output to the system console:

PAUSE xxxx

where *xxxx* is an optional octal number.

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

STOP causes the program to terminate after the following message:

STOP program name xxxx

where *xxxx* is an octal number.

ERR0 LIBRARY ROUTINE

ERR0, the error print routine referred to under the FORTRAN or ALGOL control statement, outputs the following message to the system console whenever an error occurs in a library routine:

name: nn xx

where *name* is the name of the user's program,
nn is the routine identifier, and
xx is the error type.

The compiler generates calls to ERR0 automatically. If the FORTRAN (or ALGOL) control statement includes an *n* option, the call will be to ERR*n*, a routine which the user must supply.

DOS-III RELOCATING LOADER

The DOS-III Relocating Loader accepts relocatable object programs which have been translated by the Assembler, ALGOL Compiler, or FORTRAN Compilers. It generates an executable main-memory image of each such program. The relocatable programs may enter the loader as

- Job Binary Area programs translated during the current job
- User files
- Punched tapes, magnetic tapes
- 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. With the STORE,P directive, the program may be stored as a permanent user file to be run during a later job. If the Loader is to be re-executed during a single job, the Job Binary Area must be cleared (using the CLEAR directive) to prevent duplicate program names.

PROG,LOADR

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

Format

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

P_1 = 0 for loading from JBIN and relocatable library (default)
= 2 for loading from JBIN, user files, and relocatable library
= n for loading from JBIN, user files, relocatable library, and paper tape or magnetic tape (logical unit *n*)

P_2 = list device logical unit number (default is 6)

P_3 = 0 for no DEBUG, ≠ 0 for DEBUG (default is 0)

P_4 = 0 for base page linking, ≠ 0 for current page linking (default is 0)

P_5 = 0 for system default program bounds (e.g., UBFWA-UBLWA and UMFWA-UMLWA); = 1 for user-specified program bounds (default is 0)

Comments

INPUT PARAMETER [P_1]. Note the hierachy here. If *n* is specified, the JBIN area is still scanned first, then user files are requested and, finally, the peripheral relocatable input is accepted.

If P_1 ≠ zero, the Loader first expects a list of relocatable file names. In keyboard mode, the Loader requests:

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 is entered as

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

following the PROG directive (or following the bounds parameters if P_5 = 1). If there are no user files, a /E record must be entered.

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

DEBUG PARAMETER [P_3]. 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 system console.

CURRENT PAGE LINKING PARAMETER [P_4]. If requested to do so ($P_4 \neq$ zero), the Loader attempts to place necessary program links on the current page of memory as opposed to the base page, to provide more area on the base page for large programs.

Note: While using the Loader with the current page linking option, remember that:

a. Current page linking cannot be used on programs which use main memory following the program area for writing data (at execution time). For instance, the Assembler builds its symbol table immediately following the last word of the largest segment.

b. Programs should be broken into subroutines of less than 2K because links are generated only at the beginning and end of the program. Links cannot be inserted into the middle of a program since the boundary between program and links may fall in the middle of a skip or jump sequence. If the program spans more than two pages, the middle page(s) will have no area available for current links and will use base page links; thus, the potential for greater efficiency will be lost.

PROGRAM BOUNDS SPECIFICATION PARAMETER [P_5]. The user has the option of specifying the base page bounds and the main memory bounds for the relocatable modules being loaded. If parameter P_5 in the PROG,LOADR directive is zero, the program bounds are determined by the system pointers:

UBFWA	lower base page bound
UBLWA	upper base page bound
UMFWA	lower main memory bound
UMLWA	upper main memory bound

If P_5 is equal to one, the user can specify his own memory bounds. In batch mode, the Loader reads the bounds from the input device immediately following the :PROG, LOADR directive. The bounds are in the form of two records: the first record is interpreted as the lower and upper base page bounds, specified by two octal constants separated by a comma. If an error occurs in the first

record, the Loader outputs an L18 error message. The second record is interpreted as the lower and upper main memory bounds, specified by two octal constants separated by a comma. If an error occurs in the second record, the Loader outputs an L19 error message. If any of the bounds are omitted, the appropriate system default value is used. In keyboard mode, the two records are entered in response to the messages

BP BND [L,U] ?
PROG BND [L,U] ?

If an error occurs while entering the bounds in keyboard mode, the user can re-enter the bounds (after an L18 or L19 error message). If an L18 or L19 error message occurs in batch mode, the Loader aborts the job.

I/O Drivers

The Loader will accept Type 4 programs (Disc Resident Device Drivers) and store them as such in the user directory. Type 4 programs cannot be combined with any other program type during any given load operation.

Loader Operation

The DOS-III Relocating Loader is a two-pass Loader. The first pass consists of setting the bounds, inputting and scanning relocatable programs to build the necessary tables (program name table and a table of entry points and externals), and matching entry points with externals. The second pass involves the relocation of the programs into an absolute core image format on the disc.

INPUTTING AND SCANNING THE PROGRAMS. Programs are scanned (and input, if necessary) according to P_1 in the PROG,LOADR directive. (Only non-disc relocatable programs must be *input;* there are stored temporarily on the Work Area of the disc for processing during the second pass.) Since main programs are matched with segments during the scan, each main program must be loaded before any of its segments.

If paper tape input is requested, the following messages are output to the system console:

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 output to the system console:

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

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

If a checksum error occurs when loading relocatable programs from paper tape, the Loader prints an L01 error message and returns to the paper tape load point with the messages

LOAD TAPE
LOADR SUSP
@

The operator can attempt to reload the program by placing the tape in the reader at the beginning of the program and typing :GO.

Matching Entries with Externals

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

UNDEFINED EXTS

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

To load additional programs from a peripheral device, the operator types

:GO,0[,n]

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

To continue without fulfilling external references, the operator types

:GO,1

To specify a file name from the keyboard, enter

:GO,2

and the appropriate prompt is output:

ENTER FILE NAME(S) OR /E

RELOCATING PROGRAMS. The main and segment names (from the PROGRAM, HPAL, or NAM records) 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 existing user files (current user disc only). If duplicate names occur, an error message is printed and loading stops.

The Loader converts each main program into an absolute main memory image, stores it on the disc, places the name in the user directory where it remains during the current job, and lists (on the logical unit specified by the P_2 parameter) the program address map and entry points. After each main program, any associated segments are loaded in the same way. When the Loader is completely finished, the following message is output:

LOADR COMPLETE

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

If no programs are entered, the Loader outputs the following messages and terminates:

NO PROGRAMS LOADED
LOADR COMPLETE

EXAMPLE

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

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

The following is then output on the standard list device (logical unit 6):

RELOCATING LOADER

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

THE RELOCATABLE LIBRARIES

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

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

All of these libraries and the subroutines they contain are documented in the manual *Relocatable Subroutines (02116-91780).*

DEBUG LIBRARY SUBROUTINE

RTE/DOS DEBUG, a subroutine of the Relocatable Library, allows programmers to check for logical errors during execution. DEBUG is described in *A Pocket Guide to HP 2100 Computers (5951-4423).* If the P_3 parameter of the PROG,LOADR directive is not zero, 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 outputs the message

BEGIN 'DEBUG' OPERATION

The programmer now enters any legal debug operation. DEBUG ignores illegal requests and outputs the message

ENTRY ERROR

DEBUG OPERATIONS

B,A	Instruction breakpoint at octal address *A* (Note: if *A* = JSB EXEC, a memory protect violation occurs)
D,A,N_1*[,*N_2*]*	ASCII dump of octal main memory address N_1 or from N_1 to N_2
D,B,N_1*[,*N_2*]*	Binary dump of octal main memory 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 octal address *A* or 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 octal address A_1
S,A_1*,*D_1*,*D_n	Set D_1 to D_n in successive memory locations beginning at octal address A_1
W,A,D_1	Set *A* register to octal D_1
W,B,D_2	Set *B* register to octal D_2
W,E,D_3	Set *E* register (0 = off, non-zero = on)
W,O,D_4	Set Overflow (0 = off, non-zero = on)
X,A	Clear breakpoint at octal address *A*
A	Abort DEBUG operation

SEGMENTED PROGRAMS

User programs may be structured into a main program and several segments, as shown in Figure 5-1. 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 main memory by EXEC calls, when needed. Only one segment may be in main memory at a time. When a segment is read into main memory, it overlays the segment previously in main memory.

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 the Loader, the main program must be entered prior to its segments. One external reference from each segment to the main routine is required for DSGEN or the Loader to link the segments and main programs. Also, each segmented program should use unique external reference symbols. Otherwise, DSGEN or the Loader may link segments and main programs incorrectly.

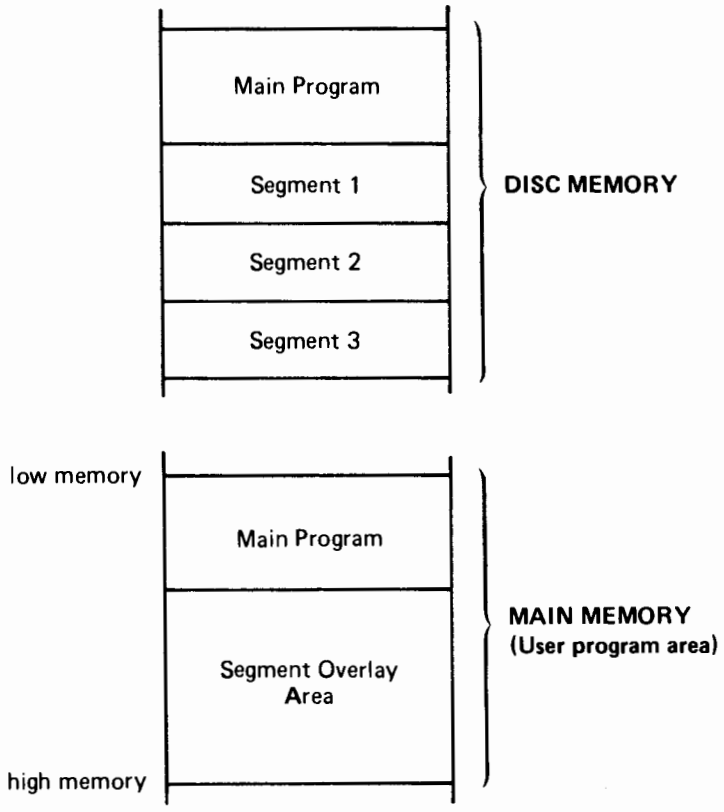

Figure 5-1. Segmented Programs

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

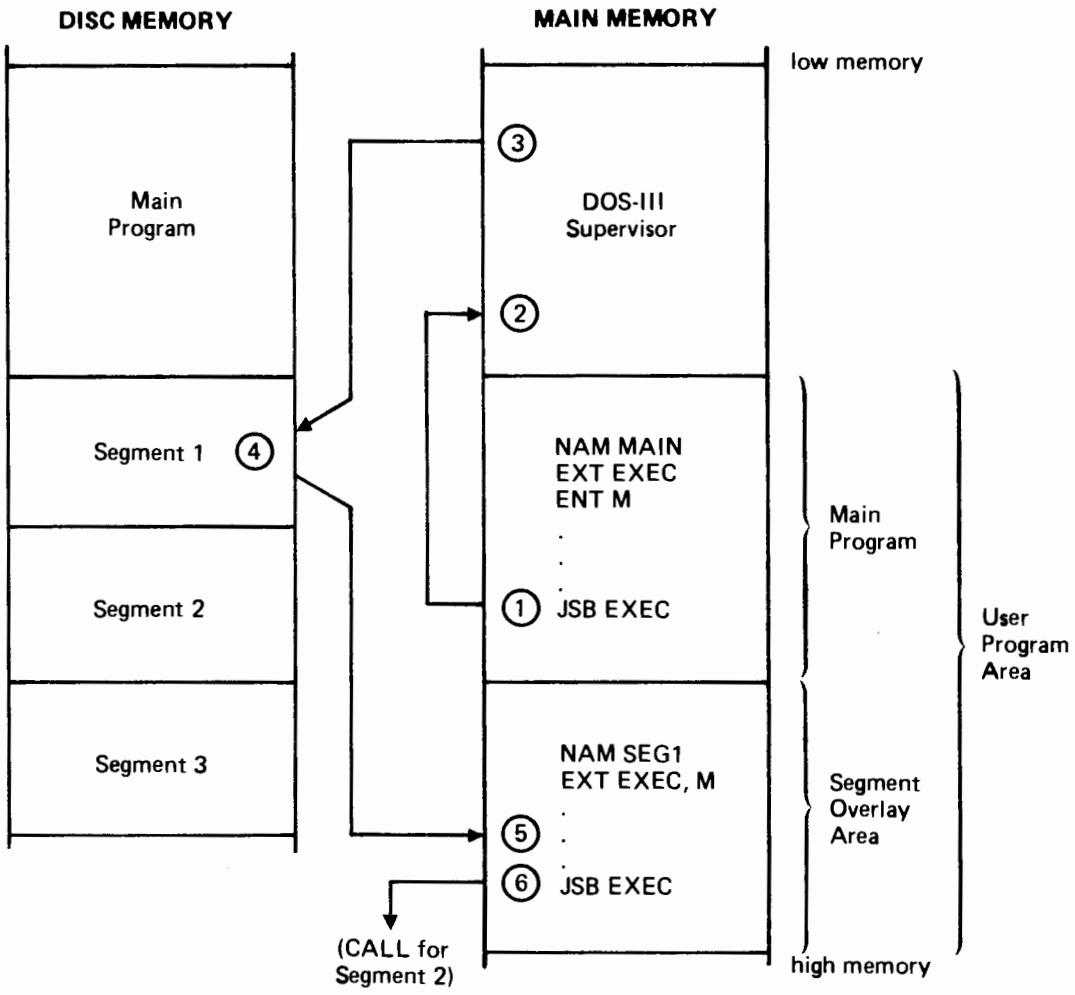

Figure 5-2. Main Calling Segment

Figure 5-3 shows how DOS-III processes the call from the segment (7) by locating the segment on the disc (8-9), loading it into main memory (10), and beginning execution of it.

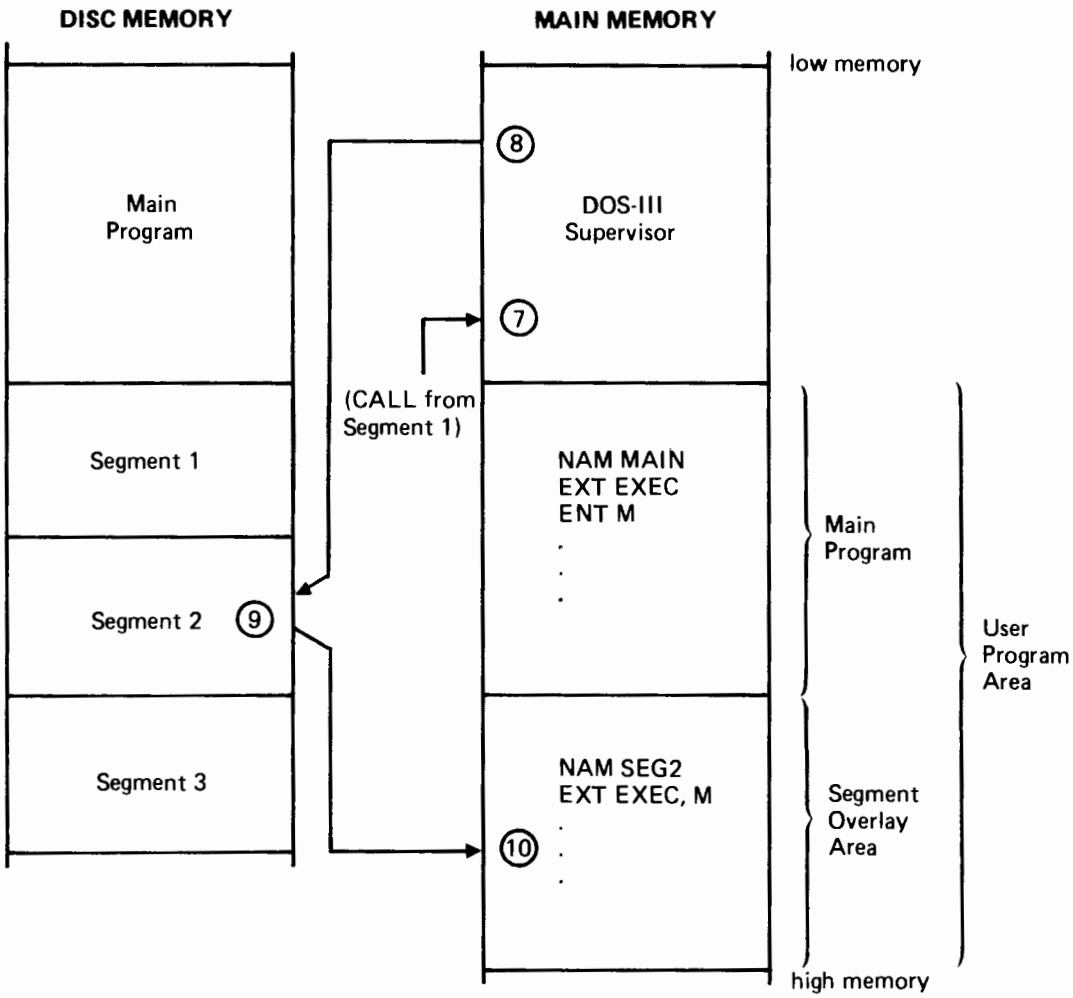

Figure 5-3. Segment Calling Segment

When a main program and segment are currently residing in main memory, they operate as a single program. Jumps from a segment to a main program (or vice versa) can be programmed by declaring an external symbol and referencing it via a JMP or JSB instruction. (See Figure 5-4.) A matching entry symbol must be defined as the destination in the other program. DSGEN or the Loader associates the main programs and segments, replacing the symbolic linkage with actual absolute a 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 main memory before any JMP instructions are executed.

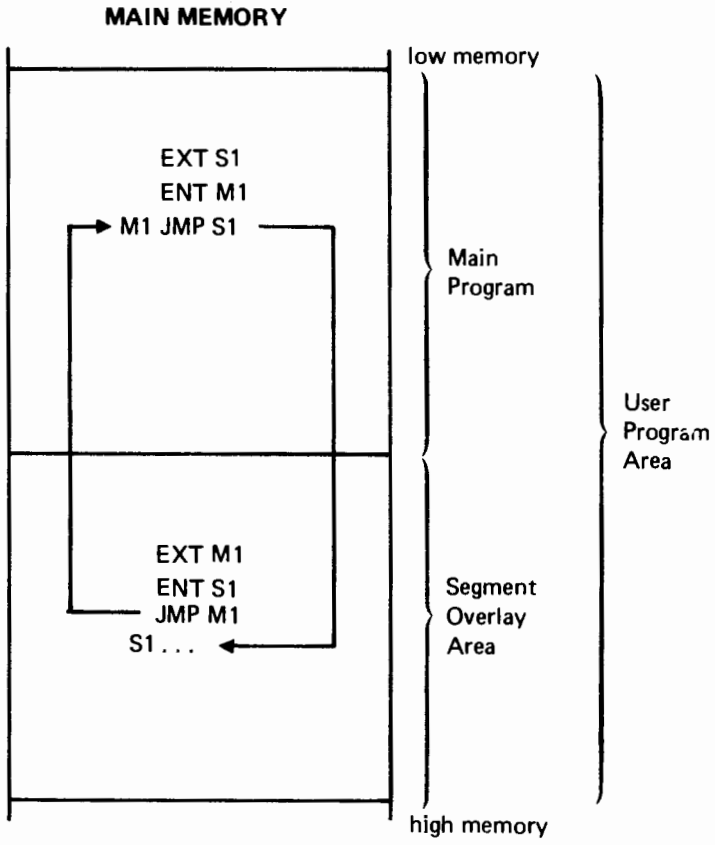

Figure 5-4. Main-to-Segment Jumps

FORTRAN 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 by using the SEGMENT LOAD EXEC call (RCODE = 8) from the main or another segment. A dummy CALL to the main must appear in each segment to ensure that proper linkage will be established between the main and its segments.

Once a segment is loaded, control is passed to it and execution begins at its primary entry point (or at the address specified in base page location 135_8). The segment, in turn, may call another segment using another SEGMENT LOAD EXEC call. Communication between the main program and segments may be through COMMON or via parameters passed in the SEGMENT LOAD or SEGMENT RETURN EXEC calls. Segments may not contain DATA statements.

Any segment may return to the main program at the statement immediately following the initial SEGMENT LOAD EXEC call (RCODE = 8) by executing a SEGMENT RETURN EXEC call (RCODE = 29). (See Section III for a description of these EXEC calls.) However, segments may not return directly to other segments.

ALGOL Segments

ALGOL programs can be segmented if certain conventions are followed. A segment must be defined as type 5 in the HPAL control statement. The segment must be initiated by using the SEGMENT LOAD EXEC call (RCODE = 8) from the main or another segment. In order to establish the proper linkage between a main program and its segments, each segment must declare the main a CODE procedure. For example, if MAIN is the main program, each segment must declare the following:

PROCEDURE MAIN;CODE;

Once a segment is loaded, control is passed to it and execution begins at its primary entry point (or at the address specified in base page location 135_8). The segment, in turn, may call another segment using another SEGMENT LOAD EXEC call. Communication between the main program and its segments may be through parameters passed in the EXEC call.

Any segment may return to the main program at the statement immediately following the initial SEGMENT LOAD EXEC call by executing a SEGMENT RETURN EXEC call (RCODE = 29). (See Section III for a description of these EXEC calls.) However, segments may not return directly to other segments.

SECTION VI
Typical DOS-III Job Decks

ASSEMBLE A PROGRAM AND STORE IN FILE

```
:JOB,ASMBS
:PROG,ASMB,5,6,4,56,99
ASMB,B,L
     NAM TEST,3
       .
       .
       .
     END ENTER
:STORE,R,AFILE
:JOB,NEXTJ
```

Source Program

LOAD AND EXECUTE A RELOCATABLE FILE

```
:JOB,LOADE
:PROG,LOADR,2
AFILE,/E
:STORE,P,TEST
:RUN,TEST
10
23
  .
  .
  .
51
:JOB,NEXTJ
```

Data

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
   .
   .
   .
00001
:RUN,ZOOM
321.5
   .
   .
   .
0.56
:JOB,NEXTJ
```

Source Program

Edit List

Data for first run

Data for second run

SECTION VII
EFMP Organization

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

ENVIRONMENT

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

FUNCTIONS AND STRUCTURE

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

DOS-III Files vs. EFMP Files

DOS-III maintains files that are referenced by five-character names and relative sector numbers. The user can access these files in either a keyboard mode (via directives) or in a programming mode (via EXEC calls). In keyboard mode, the user creates a file with the STORE directive and operates on that file with directives such as :EDIT and :DUMP. In programming mode, the DOS-III files are accessed by EXEC calls such as FILE READ/WRITE and FILE NAME SEARCH.

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

EFMP operates within this file structure of DOS-III to set up and maintain additional—but distinctly different— files. Areas of discs within DOS-III (hereafter referred to as EFMP areas) are turned over to EFMP exclusively. The user must identify them with a pack number of the form PN*xxx*, where *xxx* is a decimal integer. The procedure for doing this is described under "Set Up."

Within an EFMP area, EFMP creates files of its own that are not known to DOS-III. They are identified by a fixed-length name, contain a grouping of specified length records, and have a security code. Since only the DOS-III files can be created and accessed by directives, all EFMP files must be used through the EFMP EXEC calls or the UTIL program. EFMP files are limited in size only by the requirement that they fit within one subchannel or pack.

> *Note: All references to files within this Part will mean EFMP files, not DOS-III files, unless specifically stated otherwise.*

Duplicate Pack Numbers

EFMP pack numbers are always unique on any given platter, but not necessarily unique across platters. To minimize the possibility of accessing a duplicate pack number, the user should (if possible):

1. Create unique pack numbers.
2. Have platters containing EFMP areas mounted on the subchannel designated as the current user subchannel.

EFMP Buffers and Tables

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

1. Opened File Table
2. Temporary Record Buffers

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

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

> *CAUTION: SINCE THESE TABLES AND BUFFERS EXIST IN THE USER AREA AND ARE NOT PROTECTED, EXTREME CAUTION MUST BE TAKEN NOT TO MODIFY THEM IN ANY WAY.*

Logical Read vs. Physical Read

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

Logical Write vs. Physical Write

A logical write occurs each time a user requests that a record be written to a file. At that time, EFMP determines if that record is present in the Temporary Record Buffer; if it is, EFMP simply transfers the data in the user's record buffer to the Temporary Record Buffer and flags it as "must be written." Each succeeding read or write is treated in the same manner until a logical record transfer occurs for which the record is not in main memory, or until the last record in the Temporary Record Buffer is logically written. In these cases, the EFMP must physically write the records in the Temporary Record Buffer (i.e., post them) on the disc.

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

Update-Writes vs. Append-Writes

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

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

SET UP

There are two prerequisites for EFMP. First, the EFMP EXEC modules must be included in DOS-III when the system is generated. (Use binary tape 24309-60002 if the hardware has EAU; otherwise use binary tape 24309-60001.) Second, when DOS-III is running, the user must create EFMP areas on formatted DOS-III packs or cartridges.

An EFMP area is created by issuing a STORE, B directive in this format:

:STORE,B,PNxxx,sectors

where *xxx* is a unique decimal number,

PNxxx is the unique pack number, and

sectors is the number of sectors of the EFMP area.

Note: EFMP changes the file from Type-B to Type-A during initialization (see "Initialize").

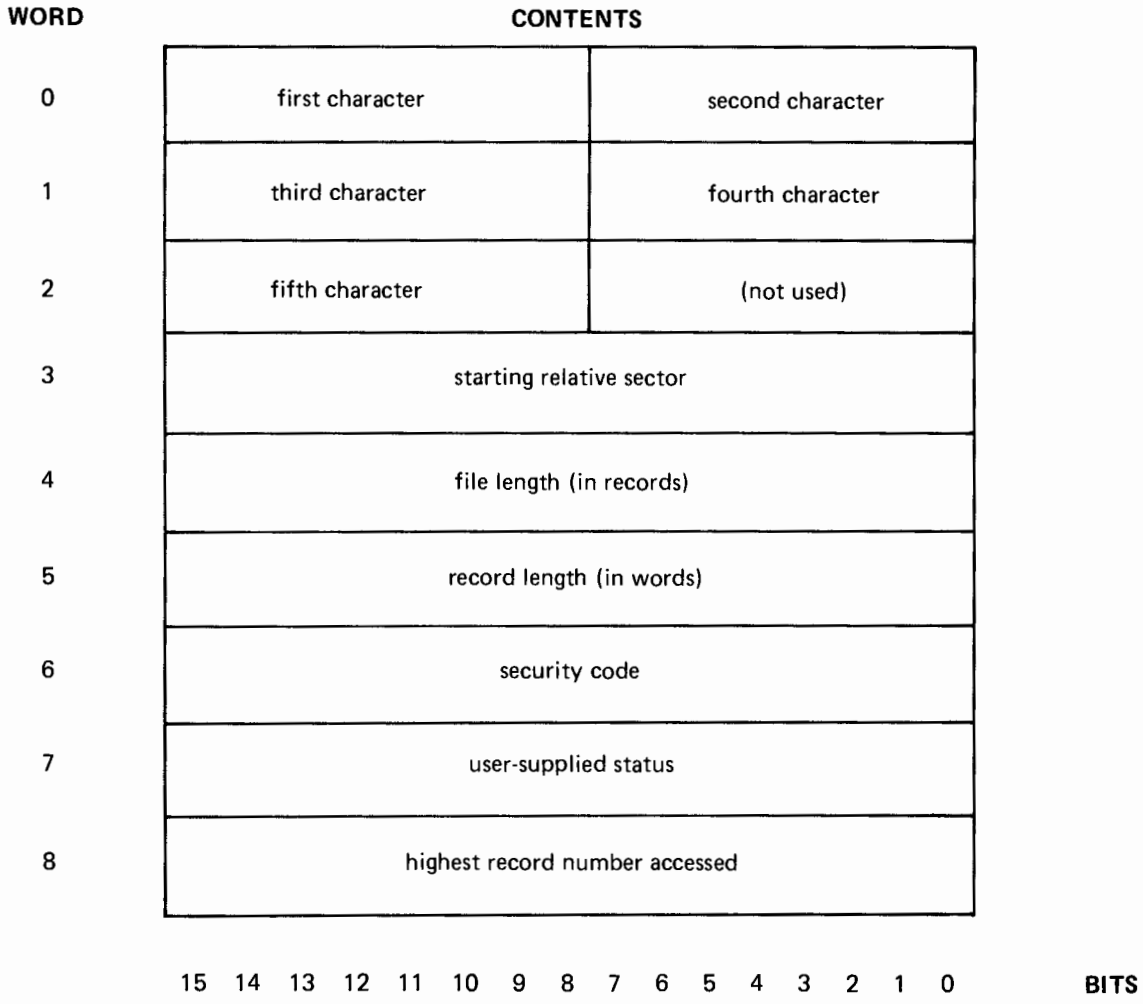

Figure 7-1. EFMP File Disc Directory Format

SECTION VIII
EFMP EXEC Calls

The method of communication between a user program and EFMP is through the standard DOS-III EXEC call format (discussed in Section III of this manual).

One standard DOS-III request code (RCODE = 24) is reserved for EFMP requests. The DOS-III operating system combines this request code with an EFMP function number to determine which action the user EXEC call is requesting. The EFMP function numbers are one element in each of the EFMP EXEC calling sequences.

FORMAT FOR EFMP EXEC CALLS

In this section, only the Assembly language calling sequences are given for the EFMP EXEC calls. The methods for converting these calling sequences to FORTRAN or ALGOL are described in Section III.

The EFMP EXEC calls described in this section are presented in ascending order, by EFMP function number. The STATUS EXEC call (EFMPF = 10) has several status function numbers: these are presented in ascending order, by status function number.

Note: A complete list of EFMP error codes can be found in PART 5 of this manual, "Error Codes and Messages."

DEFINE

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+9         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function number
                DEF     OPNTB       Opened-file table address
                DEF     OPNSZ       Opened-file table size
                DEF     TRBUF       Temp. record buffer address
                DEF     NOTRB       Number of temp. record buffers and number of
                                    active pack numbers
                DEF     TRBSZ       Temp. record buffer size
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEF     24
EFMPF           DEC     1
OPNTB           BSS     n           Opened-file table  (n is the size)
OPNSZ           DEC     n           Size of opened-file table (in 16-bit words,
                                    see Comment 1)
TRBUF           BSS     M           Beginning of temp. record buffers, see Comment 2
NOTRB           DEC     p           No. of temp. record buffers, see Comment 2
MAXPK           DEC     n           n = the maximum number of unique EFMP pack
                                    numbers active, see Comment 4
TRBSZ           DEC     q           Size of each temp. record buffer (in sectors)
ERRNO           BSS     1           Return point for error codes
```

Comments

1. The size of the Opened-file table (n) can be calculated by this formula:

   $n = 4*(MAXPK)+ 3*(NOTRB)+16*(Max.\ no.\ of\ files\ to\ be\ opened)$

   The minimum size of this table is 128 words. This allows approximately seven files to be opened concurrently.

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

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

   $m = NOTRB * TRBSZ * 128$

   (The minimum value for TRBSZ is 2.)

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

4. MAXPK indicates the maximum number of unique EFMP pack numbers a user will have active at any one time. A pack number is active when one or more of its files are opened by a user through an OPEN call (or for PN000 through a CREATE call).

CREATE

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+9         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function number
                DEF     FNAME       File name
                DEF     PAKNO       Pack number
                DEF     FLGTH       File length (in records)
                DEF     RLGTH       Record length (in words)
                DEF     SCODE       Security code and user status
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     2
FNAME           ASC     3,xxxxx     xxxxx is the name to be applied to the file
                                    (first two characters cannot be zero or 177400(8))
PAKNO           DEC     p           p is the pack number, see Comments
FLGTH           DEC     q           q is the number of records in the file;
                                    (1 ≤ q ≤ 32,767)
RLGTH           DEC     r           r is the number of words in a record;
                                    r must be less than or equal to 1/2 the size
                                    of the temp. record buffer
SCODE           OCT     s           s is any 16-bit combination to be checked by EFMP
                                    during OPEN and DESTROY
(SCODE+1)       OCT     t           t is any 16-bit combination of status information
                                    desired by the user (referred to as USTAT elsewhere)
ERRNO           BSS     1           Return point for error codes
```

Comments

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

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

3. When the Work Area is used for temporary files, EFMP reserves this whole area and identifies it as PN000. In order to keep PN000 from using the entire Work Area, the user must enter a STORE,B,PN000 directive for the system disc with the desired number of sectors. When EFMP has terminated, the user should PURGE the file PN000 from the Work Area.

DESTROY

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+7         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function code
                DEF     FNAME       File name
                DEF     PAKNO       Pack number
                DEF     SCODE       Security code
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     3
FNAME           ASC     3,xxxxx
PAKNO           DEC     n           If n = 0, then FNAME refers to a temporary file
                                    (if n ≥ 1 and n ≤ 999, FNAME is to be located in
                                    this EFMP area; if n = -1, EFMP searches all of its
                                    areas until it finds a file that matches FNAME)
SCODE           OCT     s           s is the security code for the file established by the
                                    CREATE EFMP call; security code ignored on
                                    temporary files
ERRNO           BSS     1           Return point for error codes
```

OPEN

Purpose

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

Assembly Language

```
                JSB     EXC
                DEF     *+9         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function code
                DEF     FNAME       File name
                DEF     PAKNO       Pack number
                DEF     RCDNO       Record number
                DEF     SCODE       Security code
                DEF     BUFNO       Buffer number
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     4
FNAME           ASC     3,xxxxx
PAKNO           DEC     n           If n = 0, the file is a temporary file on the work area;
                                    if n is between 1 and 999, EFMP looks for FNAME in
                                    the appropriate area; if n = -1, EFMP searches all
                                    available areas for the requested file
RCDNO           DEC     r           If r = 0, EFMP sets the next record to be accessed
                                    (for sequential READS or WRITES) to the highest
                                    record previously accessed + 1. Otherwise, r can be
                                    any number between 1 and the maximum record
                                    number contained in the file. This allows sequential
                                    access to be initialized at any record.
SCODE           OCT     s           s is the security code established by the CREATE
                                    call. It must match.
BUFNO           DEC     b           b must be a number between 1 and the maximum
                                    number of Temp. Record Buffers available. For
                                    any other number, EFMP uses 1
ERRNO           BSS     1           Return point for error codes
```

CLOSE

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+6         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function number
                DEF     FNAME       File name
                DEF     USTAT       User status
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     5
FNAME           ASC     3,xxxxx     See Comment 2
USTAT           OCT     u           User status information (any 16-bit combination) to
                                    be written into the disc directory for the file
ERRNO           BSS     1           Return point for error codes
```

Comments

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

2. To CLOSE all files in the Opened-File Table set the first word of FNAME equal to -1.

READ

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+7         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function code
                DEF     FNAME       File name
                DEF     RCDNO       Record number
                DEF     BUFFR       Buffer for data
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     6
FNAME           ASC     3,xxxx
RCDNO           DEC     n           n is a record number between 1 and 32,767. For
                                    sequential access and backspacing, see Comments.
BUFFR           BSS     m           m is the length of the buffer in words. It must be
                                    at least the record length.
ERRNO           BSS     1           Return point for error codes
```

Comments

If RCDNO = 0, a sequential read or write is implied. This feature provides the program with the next record available relative to the last read or write performed (or OPEN operation). If RCDNO is a negative number, it specifies a backspace, relative to the current record (last record accessed plus 1), before the read or write. If an attempt is made to backspace the record number indicator to a value less than one, the EFMP issues an error and terminates the read or write. Unless needed, care should be taken so as not to backspace the record number indicator beyond the range of records held in the Temporary Record Buffer at that time, since this will initiate a posting operation and a physical disc access.

INITIALIZE

Purpose

To initialize an EFMP area previously created by means of a DOS-III STORE directive.

Assembly Language

```
            JSB     EXEC
            DEF     *+6         Return address
            DEF     RCODE       Request code
            DEF     EFMPF       EFMP function number
            DEF     PAKNO       Pack number
            DEF     DIRSZ       Directory size
            DEF     ERRNO       Error number
            return              Continue execution
              .
              .
              .
RCODE       DEF     24
EFMPF       DEC     7
PAKNO       DEC     p           (1 ≤ p ≤ 999)
DIRSZ       DEC     n           (n = number of entries, one entry/file; see Comment 2)
ERRNO       BSS     1           Return point for error codes
```

Comments

1. Pack number PN000 cannot be initialized.

2. The directory occupies the first sector(s) of the EFMP area. The number of sectors allocated to a directory is determined as follows:

   $$\#\text{Sectors} = \frac{(1 + n) * 9}{128}$$

   (add 1 to #Sectors if remainder is > zero)

WRITE

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+7         Return address
                DEF     RCODE       Request code
                DEF     EFMPF       EFMP function number
                DEF     FNAME       File name
                DEF     RCDNO       Record number
                DEF     BUFFR       Buffer for data
                DEF     ERRNO       Error number
                return              Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     8
FNAME           ASC     3,xxxxx
RCDNO           DEC     n           Same as for the READ EXEC CALL
BUFFR           BSS     m           Same as for READ
ERRNO           BSS     1           Return point for error codes
```

RESET

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+7
                DEF     RCODE   Request code
                DEF     EFMPF   EFMP function code
                DEF     FNAME   File name
                DEF     PAKNO   Pack number
                DEF     RCDNO   Record number
                DEF     ERRNO   Error number
                return          Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     9
FNAME           ASC     3,xxxxx
PAKNO           DEC     n       If n = 0, EFMP searches the work area to find the
                                desired file name; if n is a number between 1 and 999,
                                EFMP searches EFMP area PNn to find the desired
                                file name; if n = -1, EFMP searches all EFMP areas
RCDNO           DEC     m       m is a number between 0 and 32,767 to which the
                                highest record accessed pointer will be set ( m must
                                be lower than the current value)
ERRNO           BSS     1       Return point for error codes
```

STATUS

Purpose

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

Assembly Language

```
            JSB     EXEC
            DEF     *+9         Return address
            DEF     RCODE       Request code
            DEF     EFMPF       EFMP function code
            DEF     FSTAT       Status function number
            DEF     FNAME       File name
            DEF     PAKNO       Pack number
            DEF     DUMMY       Not used
            DEF     STATB       Status buffer
            DEF     ERRNO       Error number
            return              Continue execution
              .
              .
              .
            Note:   Above is the general format for Status EFMP calls. The use
                    and meaning of each parameter in the calling sequence varies
                    from status call to status call. The parameters for each call
                    are given separately below. Common to all status functions
                    are
RCODE       DEC     24
EFMPF       DEC     10
DUMMY       BSS     1
```

STATUS

Purpose

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

Parameters

FSTAT	*DEC*	*1*	
FNAME	*ASC*	*3,xxxxx*	
PAKNO	*DEC*	*m*	*If m = 0, EFMP searches the Work Area for the requested file. If m is between 1 and 999, EFMP searches the EFMP area of that pack number. For m = –1, EFMP searches all available EFMP areas for the requested file.*
STATB	*BSS*	*10*	*The pack number is returned in the first word if PAKNO = –1. The remaining nine words will receive the directory status information in the same format as the directory itself (see Figure 7-1).*
ERRNO	*BSS*	*1*	*Return point for error code.*

STATUS

Purpose

To determine if a file is open.

Parameters

FSTAT	*DEC*	*2*	
FNAME	*ASC*	*3,xxxxx*	
PAKNO	*OCT*	*0*	*Not used*
STATB	*BSS*	*2*	*The first word returns the pack number if the file is open. The second word returns a value of 0 if the file is open or 1 if the file is not open.*
ERRNO	*BSS*	*1*	*Return point for error codes.*

STATUS

Purpose

To check the security code of a file.

Parameters

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

STATUS

Purpose

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

Parameters

FSTAT	*DEC*	*4*	
FNAME	*ASC*	*3,xxxxx*	
PAKNO	*DEC*	*m*	*Same as function number 1*
STATB	*BSS*	*2*	*The first word returns the pack number if appropriate. The second word returns the number of sectors available.*
ERRNO	*BSS*	*1*	*Return point for error codes.*

STATUS

Purpose

To determine the number of available sectors left between the last file in an EFMP area and the end of the EFMP area.

Parameters

FSTAT	*DEC*	*5*	
FNAME	*OCT*	*0*	*Not used*
PAKNO	*DEC*	*m*	*Same as function number 1, but cannot equal –1*
STATB	*BSS*	*2*	*The first word must be present, but is not used. The second word returns the number of sectors available.*
ERRNO	*BSS*	*1*	*Return point for error codes.*

STATUS

Purpose

To obtain the name of the *n*th file in an EFMP area where *n* is an integer between 1 and the maximum number of files in an EFMP area.

Parameters

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

STATUS

Purpose

To obtain the name of the nth pack number on a specific subchannel where n is an integer (specifying the ordinal position of the pack number) between 1 and the maximum number of pack numbers on a subchannel.

Parameters

FSTAT	*DEC*	*7*	
FNAME	*DEC*	*m*	*m = the desired subchannel* *On return, FNAME is zero if the EFMP area of the pack number is initialized and 1 if the EFMP area of the pack number is not initialized.*
.			
.			
.			
.			
PAKNO	*BSS*	*1*	*Return point for the pack number*
STATB	*DEC*	*n*	*n indicates the nth pack number.*
ERRNO	*BSS*	*1*	*Return point for error codes.*

REPACK (PURGE)

Purpose

To repack the existing files on an EFMP area(s), removing empty spaces left when files have been destroyed.

Assembly Language

```
                JSB     EXEC
                DEF     *+5
                DEF     RCODE   Request code
                DEF     EFMPF   EFMP function code
                DEF     PAKNO   Pack number
                DEF     ERRNO   Error number
                return          Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     11
PAKNO           DEC     n       For n between 1 and 999, only the specified EFMP
                                area is repacked; for n = -1, all the EFMP areas
                                available to EFMP are repacked
ERRNO           BSS     1       Return point for error codes
```

CAUTION: IF THE EFMP DISC DIRECTORY CONTAINS A LARGE NUMBER OF FILES AND THE SIZES OF THE TEMPORARY RECORD BUFFERS ARE SMALL, REPACKING MAY REQUIRE CONSIDERABLE TIME. THEREFORE, REPACK SHOULD BE PERFORMED WHEN SUFFICIENT TIME IS AVAILABLE. UNDER NO CIRCUMSTANCES SHOULD AN ABORT BE PERFORMED DURING A REPACK.

COPY

Purpose

To transfer a copy of an opened file and its directory from an EFMP area to the Work Area of DOS-III, from one EFMP area to another EFMP area or from the Work Area to an EFMP area.

Assembly Language

```
            JSB     EXEC
            DEF     *+6
            DEF     RCODE     Request code
            DEF     EFMPF     EFMP function code
            DEF     FNAME     File name
            DEF     PAKNO     Pack number
            DEF     ERRNO     Error number
            return            Continue execution
              .
              .
              .
RCODE       DEF     24
EFMPF       DEC     12
FNAME       ASC     3,xxxxx   See Comment 1
PAKNO       DEC     n         If n = 0, EFMP copies the file onto the Work Area;
                              if n is between 1 and 999, EFMP copies the file
                              into the specified EFMP area; if n is between -1
                              and -999, EFMP copies the file from the Work
                              Area to an EFMP area specified by the 10's
                              complement of n (see Comment 2)
ERRNO       BSS     1         Return point for error codes
```

Comments

1. Remember that a file must be opened before it can be copied. This is necessary to determine from which pack to copy the file. When a file has been copied to the Work Area, all reads and writes referencing that file use the Work Area version until the file is closed. (Files copied from the Work Area to an EFMP area continue to use the Work Area version for reads and writes.) Temporary copies of files do not have security codes. Therefore, files copied from the Work Area to a pack have a security code of 0. When a file is copied from pack to pack, the original security code is retained. See "CLOSE" for further notes on Work Area files.

2. If there is already a file with the same name in the destination EFMP area directory, an error code is returned and the copy is aborted. In this case, the user can first destroy the name in the destination EFMP area, and then perform the copy again.

3. When copying from one EFMP area to another EFMP area not on the drive (and only a single removable pack is available), EFMP automatically requests that the user continually swap packs until the entire file has been copied. EFMP outputs:

   INSERT DESTINATION [SOURCE] PACK AND PRESS RUN.

   and halts the computer with 102076 in the DISPLAY register.

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

   ENTER C OR T

   where C means to continue copying, and

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

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

CHANGE FILE NAME

Purpose

To change a file name (file need not be opened).

Assembly Language

```
                JSB     EXEC
                DEF     *+7
                DEF     RCODE   Request code
                DEF     EFMPF   EFMP function code
                DEF     FNAME   File name
                DEF     PAKNO   Pack number
                DEF     SCODE   Security code
                DEF     ERRNO   Error number
                return          Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     13
FNAME           ASC     3,xxxxx Current file name
                ASC     3,zzzzz New file name
PAKNO           DEC     n       n = 0, indicates that the file is on the Work Area;
                                if n is between 1 and 999, n indicates the EFMP
                                area containing the file; if n = -1, EFMP searches
                                all available EFMP areas for the current file name
SCODE           OCT     m       Security code, see CREATE
ERRNO           BSS     1       Return point for error codes
```

POST

Purpose

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

Assembly Language

```
                JSB     EXEC
                DEF     *+4
                DEF     RCODE   Request code
                DEF     EFMPF   EFMP function code
                DEF     ERRNO   Error number
                return          Continue execution
                  .
                  .
                  .
RCODE           DEC     24
EFMPF           DEC     14
ERRNO           BSS     1       Return point for error codes
```

Comments

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

SECTION IX
EFMP Utility Program

The EFMP Utility Program (UTIL) allows the user to access most of the EFMP functions through the keyboard. UTIL accepts commands or directives from the operator and converts these into EFMP calling sequences. After EFMP has processed the call, UTIL reports back (to the operator) a successful operation or an EFMP error.

This section describes how to initiate the UTIL program using the DOS-III PROG directive and then describes the following UTIL commands (presented in alphabetic order):

BRIEF
CHANGE
CLOSE
COPY
CREATE
DESTROY
END
INITIALIZE
OPEN
POST
REPACK
RESET
STATUS-1
STATUS-2
STATUS-3
STATUS-4
STATUS-5
STATUS-6
STATUS-7

All are EFMP functions, except BRIEF and END, which are UTIL program functions.

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

:PROG,UTIL

Purpose

To initiate execution of the UTIL program.

Format

:PROG, UTIL,n

where $n = 0$ to print a list of commands or
$n \neq 0$ to skip printing the list.

List of commands message (all parameters are decimal):

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

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

UTIL READY

After it processes the directive, UTIL outputs the results of the operation (where appropriate) or any error codes that may have been returned by EFMP. When it is ready for another directive, UTIL outputs

UTIL READY

If an incorrect directive is entered, UTIL outputs

ILLEGAL OPERATION
UTIL READY

UTIL is terminated when the operator inputs the command /END.

UTIL outputs any error messages on the system console; normal output is output on the list device.

BRIEF

Purpose

To increase or decrease the amount of disc storage reserved for a file. BRIEF is a UTIL program function, not an EFMP function.

Format

/BRI, fname,scode

fname is the name of the file, and

scode is the security code of the file.

BRIEF first outputs the status of the file:

AVAILABLE RECS. = m *RECORDS USED = r*

NEW RECORD COUNT?

The operator inputs either:

/E to terminate the command and prepare UTIL for more commands,
or

n to change the available record count to *n*

BRIEF stores the contents of *fname* on the Work Area, destroys the current file, repacks the EFMP area, and creates and opens a new file. The contents of *fname* are transferred from the Work Area to the new file and BRIEF prints out a message:

AVAILABLE RECS. = n *RECORDS USED = r*

BRIEF then terminates.

Comment

BRIEF creates and uses a temporary file named "ΔΔΔΔΔΔ" (all blanks).

CHANGE

Purpose

To change the name of a file (i.e., to invoke the CHANGE FILE NAME function of EFMP).

Format

/CHA,fnam1,fnam2,pakno,scode

fnam1 is the current file name
fnam2 is the new file name.

See CHANGE FILE NAME EFMP CALL for explanation of other parameters.

EXAMPLE

/CHA,LOB70,XXXXX,120,0

Example print-out:

```
FILE LOB70 OLD FILE
FILE XXXXX NEW FILE
THE FILE IS ON PACK#   120
THE SECURITY CODE IS   0
```

CLOSE

Purpose

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

Format

/CLO,fname,ustat

See CLOSE EFMP CALL for explanation of parameters. Note, however, that all the files in the Opened-File Table *cannot* be closed by setting the first word of FNAME (in the CLOSE calling sequence) to -1.

EXAMPLE

/CLO,LOB70,0

Example print-out:

```
FILE LOB70        CLOSED
THE USER STATUS WORD IS       0
```

COPY

Purpose

To copy a file (i.e., to invoke the COPY function of EFMP).

Format

/COP,fname,pakno

See COPY EFMP CALL for explanation of parameters and messages.

EXAMPLE

/COP,LOB70,120

Example print-out:

```
FILE    LOB70    COPIED
THE FILE IS TEMPORARY IN WORK AREA
FILE    LOB70    COPIED
THE FILE IS ON PACK#    120
```

CREATE

Purpose

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

Format

/CRE,fname,pakno,flgth,rlgth,scode,ustat

See CREATE EFMP CALL for explanation of parameters.

EXAMPLE

/CRE,C0,120,8,8,0,0

Example print-out:

```
FILE    C0    CREATED
THE FILE IS ON PACK#    120
THE FILE LENGTH IS    8 RECORDS
THE RECORD LENGTH IS    8 WORDS
THE SECURITY CODE IS    0
THE USER STATUS WORD IS    0
```

DESTROY

Purpose

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

Format

/DES,fname,pakno,scode

See DESTROY EFMP CALL for explanation of parameters.

EXAMPLE

/DES,C0,120,0

Example print-out:

FILE C0 DESTROYED

END

Purpose

To terminate the operation of the UTIL program. END is an UTIL program function, not an EFMP function.

Format

/END

INITIALIZE

Purpose

To initialize an EFMP area previously allocated space by means of a DOS-III STORE directive.

Format

/INI,pakno,dirsz

See INITIALIZE EFMP CALL for explanation of parameters.

EXAMPLE

/INI,100,20

Example print-out:

PACK #100 INITIALIZED

OPEN

Purpose

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

Format

/OPE,fname,pakno,rcdno,scode

See OPEN EFMP CALL for explanation of parameters.

EXAMPLE

/OPE,LOB70,120,1,0

Example print-out:

```
FILE     LOB70     OPENED
THE FILE IS ON PACK#     120
THE RECORD # IS     1
THE SECURITY CODE IS     0
```

POST

Purpose

To post files (i.e., to invoke the POST function of EFMP).

Format

/POS

Example print-out:

ALL FILES POSTED

RESET

Purpose

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

Format

/RES,fname,pakno,rcdno

See RESET EFMP CALL for explanation of the parameters.

EXAMPLE

/RES,LOB70,120,0

Example print-out:

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

REPACK

Purpose

To repack existing EFMP areas (i.e., to invoke the REPACK EXEC CALL function of EFMP).

Format

/REP,pakno

See REPACK EFMP CALL for explanation of parameters.

EXAMPLES

/REP,42 (repacks EFMP area in pack 42)
/REP, –1 (repacks all EFMP areas)

Example print-out:

PACK # 42 REPACKED

or

ALL PACKS AVAILABLE REPACKED

STATUS-1

Purpose

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

Format

/STA,DF,fname,pakno

See STATUS EFMP CALL (FSTAT = 1) for explanation of the parameters and results.

EXAMPLE

/STA,DF,LOB70,120

Example print-out:

```
FILE     LOB70     STATUS
THE FILE IS ON PACK#      120
STARTING TRACK # IS     6
STARTING SECTOR # IS     9
THE FILE LENGTH IS      12 RECORDS
THE RECORD LENGTH IS      128 WORDS
THE USER STATUS WORD IS     0
HIGHEST RECORD # ACCESSED IS      0
```

STATUS-2

Purpose

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

Format

/STA,FO,fname

See FSTAT = 2 for explanation of the parameters and results.

EXAMPLE

/STA,FO,LOB70

Example print-out:

```
FILE LOB70     STATUS
FILE IS OPEN
```

STATUS-3

Purpose

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

Format

/STA,SC,fname,pakno,scode

See FSTAT=3 for explanation of parameters and results.

EXAMPLE

/STA,SC,LOB70,120,0

Example print-out:

```
FILE    LOB70    STATUS
THE FILE IS ON PACK#    120
THE SECURITY CODE IS    0
CODE CHECKS
```

Note: The security code returned is a restatement of the security code entered; it is not necessarily the correct security code.

STATUS-4

Purpose

To determine the number of available full sectors left between the highest record accessed in a file and the end of the file (i.e., to invoke STATUS function number 4 of EFMP).

Format

/STA,LR,fname,pakno

See FSTAT=4 for explanation of parameters and results

EXAMPLE

/STA,LR,LOB70,120

Example print-out:

```
FILE     LOB70     STATUS
THE FILE IS ON PACK#     120
# OF AVAILABLE SECTORS IS     12
```

STATUS-5

Purpose

To determine the number of available sectors left between the last file in an EFMP area and the end of the EFMP area (i.e., to invoke STATUS function number 5 of EFMP).

Format

/STA,LF,pakno

See FSTAT=5 for explanation of parameters and results.

EXAMPLE

/STA,LF,120

Example print-out:

```
FOR PACK#     120
# OF AVAILABLE SECTORS IS     4610
```

STATUS-6

Purpose

To obtain the name of the *n*th file in an EFMP area where *n* is an integer between 1 and the maximum number of files in an EFMP area (i.e., to invoke STATUS function number 6 of EFMP).

Format

/STA,NF,pakno,statb

See FSTAT=6 for explanation of parameters and results.

EXAMPLE

/STA,NF,120,1

Example print-out:

```
FILE    LOB70    STATUS
THE FILE IS ON PACK#    120
FILE #    1 IN THE DIRECTORY
```

STATUS-7

Purpose

To obtain the name of the *n*th pack number on a specific subchannel where *n* is an integer (specifying the ordinal position of the pack number) between 1 and the maximum number of pack numbers on a subchannel.

Format

/STA,AP,subch,statb

EXAMPLE

/STA,AP,1,1

Example print-out:

PACK #120 IS AVAILABLE AND INITIALIZED

SECTION X
Generating DOS-III

Generating a DOS-III system consists of two operations:

1. Configuring the system to the available hardware.
2. Storing the configured system on the disc.

This section describes how to generate a DOS-III system (using the stand-alone DSGEN program) and how to use DSGEN to format disc cartridges and disc packs.

DSGEN

DSGEN (the DOS-III System Generator) is an absolute program, loaded into main memory by the paper tape portion of the main-memory resident Bootstrap Loader. Since DSGEN is independent of the DOS-III which it generates, the I/O operation of DSGEN requires SIO drivers. DSGEN operates on the same minimum configuration as that required for DOS-III.

DSGEN has two independent functions.

1. To generate a system that fits the user's main memory size, I/O equipment, and programming needs.
2. To format new disc cartridges (packs).

USING DSGEN TO GENERATE DOS-III

The operation of DSGEN involves four phases:

1. INITIALIZATION PHASE. DSGEN requests specifications for DOS-III, including description of available disc space, memory, Time-base Generator channel, system generation code, system and user disc subchannels, and program input devices.

2. PROGRAM INPUT PHASE. DSGEN reads the relocatable programs to be included in the system. The relocatable program modules can be input via paper tape, disc, or magnetic tape (the magnetic tape must be prepared off-line using the Prepare Tape System).

3. PARAMETER INPUT PHASE. Parameters to change EXEC modules or drivers from disc- to main-memory resident may be entered. The programs' NAM records are already set for a minimum main-memory system except that DVR00 should be changed to disc-resident. DISCM, $EX30 (if EFMP is used), DVR31 (moving-head disc driver) and DVR05 (system console driver) must be main-memory resident.

4. DISC LOADING PHASE. DSGEN requests a specification of the base page linkage, and begins loading programs onto the disc in absolute format. Systems programs (i.e., the modules of DOS-III) 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-III, turn on all equipment and unprotect the disc. Then follow these steps:

1. If DSGEN has been configured, start at step 4. Otherwise, configure DSGEN with the SIO Teleprinter Driver and, a) the appropriate SIO photoreader driver (paper tape input) or, b) the appropriate SIO magnetic tape driver (magnetic tape input).

   Note: Load DSGEN before the magnetic tape driver; otherwise an ERR01 occurs.

2. Configure the appropriate SIO punch driver, then load the SIO System Dump.

3. Set a starting address of 2_8 and set switch register bit 15 to one. Press RUN (the configured tape is punched). Press RUN again for an extra copy, if desired.

4. Load the configured DSGEN tape into the photoreader; press READ.

5. Start DSGEN at location 100_8. DSGEN begins the initialization phase.

Initialization Phase

During the initialization phase, DSGEN requests information necessary to begin generating the DOS-III. After each output on the system console, the operator responds by entering the required information terminated by a *return linefeed*. The following responses are typical. (The operator responses are only examples, actual responses should be appropriate to the particular system being generated.)

1. DSGEN requests a decimal system generation code. This code is written in the label field of the system disc for identification . . . . . *SYS GEN CODE?*

   Operator responds with a 1- to 4-digit decimal integer *79*

2. DSGEN requests the octal channel number (select code) of the disc controller . . . . . *SYS DISC CHNL?*

   Operator responds with the high priority (low number) channel *14*

3. DSGEN requests the number of sectors per physical track on the disc. This is half the number of sectors per logical or software track . . . . . *# SECTORS/TRACK?*

   Operator responds with 12 (for the 2870 disc) or 23 (for the 2883) or 24 (for the 7900/7901) *24*

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

   Operator responds with a decimal number less than or equal to 200. (A response of 200 leaves three tracks as spares. A response less than 200 leaves extra tracks as spares.) *200*

5. DSGEN requests the number of drives on the system . . . . . *# DRIVES?*

   Operator responds with 1 or 2 (2883 disc) or between 1 and 4 inclusive (2870, 7900/7901 discs) *3*

6. DSGEN requests the decimal number of the first track on the system disc which is available to DOS-III . . . . . *FIRST SYSTEM TRACK?*

   Operator responds *0*

7. DSGEN requests the decimal number of the first sector available to DOS-III . . . . . *FIRST SYSTEM SECTOR?*

   Operator responds. (The system area cannot begin before track 0, sector 3) *3*

8. DSGEN requests the subchannel number of the system disc . . . . *SYS DISC SUBCHNL?*

   Operator responds with a number between 0 and 7 . . . . . *0*

   Note: On a 7901 disc, only odd numbered subchannels are available.

9. DSGEN requests the subchannel number of the user disc. (This may be the same as the system disc.) . . . . . *.USER DISC SUBCHNL?*

   Operator responds with a number between 0 and 7. (System efficiency increases if the user disc is on a different drive from the system disc.) *2*

10. DSGEN requests the octal channel number (select code) of the Time-base Generator . . . . . *TIME BASE GEN CHNL?*

    Operator responds with the proper select code or 0 if the Time-base Generator is not present *0*

    DSGEN now requests the select code of the privileged-interrupt card . . . . . *PRIV. INT. CARD CHNL?*

    Operator must respond with a zero. *0*
    (This question is placed here for future enhancements of DOS-III.)

11. DSGEN requests the number of DMA channels in the system . . . . . *# DMA CHANNELS?*

    Operator responds with the number of DMA channels available *2*

12. DSGEN requests the last word of available main memory in octal . . . . . *LWA MEM?*

    Operator responds *27677*

13. DSGEN asks whether SS directives are to be allowed in the system . . . . . *ALLOW :SS?*

    Operator responds either YES or NO . . . . . *YES*

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

    Operator responds with PT (for paper tape), TY (for teleprinter), DF (for disc file), or MT (for magnetic tape; see *PREPARE TAPE SYSTEM (02116-91751)*) . . . . . *DF*

15. If the previous answer is DF, DSGEN requests the subchannel number of the disc containing the relocatable program modules . . . . . *INPUT DISC SUBCHNL?*

    Operator responds with the appropriate subchannel number. The subchannel must contain a disc (prepared by a pre-existing DOS-III) whose user area contains only relocatable modules of DOS-III. By specifying PT to the next question (LIBR INPT?) the operator can include programs from the paper tape reader in addition to those on the disc file *3*

16. DSGEN requests the type of optional input unit for relocatable program modules . . . . . *LIBR INPT?*

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

    Note: Any type of relocatable program can be entered through the Program Input Unit or the Library Input Unit.

17. DSGEN requests the type of input unit for the parameter input phase . . . . . *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 0-1 (00_2 for the Program Input Unit, or 10_2 for the Library Input Unit), and places the programs in the input device. Main programs must be entered prior to their segments. DISCM should be the first module loaded.

The suggested order of tape input is

DOS-III MAIN-MEMORY RESIDENT SYSTEM (DISCM)
DOS/DOS-M I/O DRIVERS (DVR05, DVR01, . . . ETC)
DOS-III EXEC MODULES ($EX01 . . .)
EFMP EXEC MODULES (IF DESIRED-$EX30 . . .)
DOS-III JOB PROCESSOR/FILE MANAGER (JOBPR)
DOS-III RELOCATING LOADER (LOADR)
DOS-M ASSEMBLER (MAIN CONTROL, SEGMENTD, SEGMENT1, . . .)
DOS-M FORTRAN (MAIN CONTROL, PASS 1, . . .)
DOS-III EFMP UTIL (IF $EX30 . . . AND FORTRAN IV LIBRARY ARE INCLUDED)
RTE/DOS ALGOL
RTE/DOS FORTRAN IV LIBRARY OR RTE/DOS BASIC FORMATTER
RTE/DOS RELOCATABLE PROGRAM LIBRARY (EAU OR NON-EAU)

Any relocatable user programs to be made a permanent part of DOS-III.

Note: If EFMP is to be generated with the system use the following binary tape: 24309-60001 if EAU is not present, or 24309-60002 if EAU is present.

Load the first input module and start the computer executing. When entering paper tape, the message "*EOT" is output whenever an end-of-tape occurs. The computer halts. Program input can be switched back and forth between the input units by varying the switch register bits between 00_2 and 10_2 before starting the computer.

To terminate the program input phase, the operator must set switch register bits to 01_2, and start the computer. If there are no undefined externals, this message is printed on the system console:

NO UNDEF EXTS

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

UNDEF EXTS

The externals are listed one per line and the computer halts. External references are satisfied by loading more programs. The operator must set switch register bits to 00_2 (for Program Input Unit) or 10_2 (for the Library Input Unit) and start the computer executing. If the externals are to be left unsatisfied, set the switch register bits to 01_2 and start the computer executing.

Note: $EX30 through $EX33 (the EFMP EXEC modules) and $EX36 and $EX37 (User EXEC modules) are not listed when missing.

Parameter Input Phase

During the parameter input phase, the operator can change selected I/O drivers and EXEC modules from disc- to main-memory resident. (If an I/O driver is changed from disc-resident (type 4) to main-memory resident (type 0), the associated EQT entry must include the R parameter.) Since DVR00 is a DOS driver, it is distributed as a main-memory resident driver; it should be changed to disc-resident if DVR05 is included in DOS-III. Any unnecessary I/O drivers *must* be eliminated at this time. DVR05, DVR31, DISCM, and $EX30 are distributed as main-memory resident modules; they must not be changed to disc-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;

- 0 - System main-memory 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 is deleted from system

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

Module Name	Request Codes	Function
$EX01	16	Disc work tracks status
$EX02	17	Disc work tracks limits
$EX03	6	Program completion
$EX04	7	Program suspension and associated messages
$EX05	8,10	Program main or segment search *(Note: $EX05 calls $EX10)*
$EX06	18	User file name search
$EX07	11	Current time processor
$EX08	4 (RT)	Real-time disc allocation
$EX09		:EQ processor
$EX10	8,10	Load and execute main program or segment *(Note: see also $EX05)*
$EX11	14,15	System file name search *(Note: used for file read/write)*
$EX12		System startup

Module Name	Request Codes	Function
$EX13		Error message processor
$EX14		:UP, :DN, :LU processor
$EX15		Abort and post-mortem dump
$EX16		:GO parameter processor
$EX17	23	:UD processor
$EX18	1,2,3, 14, 15	I/O initiation processor *(Note: see also $EX11)*
$EX19		:IN processor
$EX20		Disc parity processor

Functions of EFMP EXEC Modules

$EX30	—	Always main-memory resident (common routines and values).
$EX31	—	DEFINE, CREATE, DESTROY, OPEN, CLOSE
$EX32	—	READ, WRITE, RESET, STATUS, CHANGE
$EX33	—	COPY, REPACK

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

$EX01
$EX02
$EX06
$EX07
$EX08

The following EXEC modules use $LBL:

$EX17
$EX19

The following EXEC modules use $SRCH:

$EX05
$EX06
$EX11

These EXEC modules use ASCII:

$EX04
$EX09
$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.

Disc Loading Phase

1. DSGEN asks for the number of base page links . . . . . *# LINKS?*

   The operator responds with the decimal number of links. If the operator responds with a blank character, DSGEN allocates the maximum number of links (803) *540*

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

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

2. 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 . . . . . *nn,DVRnn[,D] [,R] [,U]*

where *nn* is the octal channel number (lower number if multi-board, maximum is 37_8)

DVRnn is the driver name (*nn* is the equipment type code)

D, if present, means DMA channel required

R, if present, means driver is main-memory resident (must be type 0)

U is the physical subchannel number

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

Here is a sample Equipment Table:

```
* EQUIPMENT TABLE ENTRY
10,DVR31,D,R        (EQT entry #1 = disc)
12,DVR23,D          (EQT entry #2 = magnetic tape)
14,DVR05,R          (EQT entry #3 = system console)
15,DVR01            (EQT entry #4 = photoreader)
16,DVR02            (EQT entry #5 = tape punch)
17,DVR12            (EQT entry #6 = line printer)
/E                  (End of table)
```

3. 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 6 may be assigned any EQT entry desired . . . . . *m*

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

Here is a sample Device Reference Table:

```
* DEVICE REFERENCE TABLE
1    = EQT #?      (System console on channel 14, EQT #3)
3
 2   = EQT #?      (Disc on channel 10, EQT #1)
1
 3   = EQT #?      (Disc on channel 10, EQT #1—reserved for system use)
1
 4   = EQT #?      (Standard punch unit on channel 16, EQT #5)
5
 5   = EQT #?      (Standard input unit on channel 15, EQT #4)
4
 6   = EQT #?      (Standard list unit on channel 17, EQT #6)
6
 7   = EQT #?      (Standard unit definable by user)
2
 8   = EQT #?      (End of table)
/E
```

Note: The number of responses given here determines the number of logical units allowed in the system. To allow unassigned logical units for the user, respond with a 0 to as many questions as units are desired.

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

Operator responds with an entry for each I/O channel which may interrupt, in ascending order and in the format . . . . . n_1,*option*

where n_1 is the octal channel number (high number if multi-board) between 10_8 and 37_8 inclusive (must be entered in ascending order)

option directs the system in handling the interrupt:

EQT,n_2 relates the channel to EQT entry number n_2,

ABS, *value* places an absolute octal value in the interrupt location. *value* is an octal integer.

The operator terminates entry by typing . . . . . */E*

Here is a sample Interrupt Table:

```
* INTERRUPT TABLE
11,EQT,1          (Channel 11 linked to EQT #1)
13,ABS,102077     (Channel 13 interrupt location filled with an octal halt
                   instruction)
14,EQT,4          (Channel 14 linked to EQT #4)
15,EQT,5          (Channel 15 linked to EQT #5)
16,ABS,0          (Channel 16 interrupt location filled with a NOP all zeros)
/E                (End of table)
```

Note: The EQT numbers need not appear in numerical order. This order is determined by referring back to the Equipment Table. The octal channel numbers, however, must be in ascending sequence.

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).

5. DSGEN reports the last octal address plus 1 of the system base page link area . . . . . *LWA SYS LINKS yyyyy*

6. DSGEN requests the first word base page octal address of the user link area . . . . . *FWA USER LINKS?*

   Operator responds with an octal address greater than or equal to *yyyyy* and less than 2000_8 . . . . . *mmmmm*

7. DSGEN reports the last octal address plus 1 of the supervisor . . . . . *LWA SYS xxxxx*

8. DSGEN requests the octal address of the first word 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.

9. When system generation is complete, DSGEN reports . . . . **SYSTEM STORED ON DISC*

10. Protect the disc. DOS-III may now be bootstrapped into memory.

RESTART

During any of the phases, the operator can restart that phase if any error occurs by restarting DSGEN at location 100_8.

```
SYS GEN CODE?
7777

SYS DISC CHNL?
14

# SECTORS/TRACK?
24

SYS DISC SIZE?
200

# DRIVES?
2

FIRST SYSTEM TRACK?
0
FIRST SYSTEM SECTOR?
3

SYS DISC SUBCHNL?
0

USER DISC SUBCHNL?
0

TIME BASE GEN CHNL?
24

PRIV. INT. CARD CHNL?
0

# DMA CHANNELS?
2

LWA MEM?
77677

ALLOW :SS?
YES

PRGM INPT?
DF
INPUT DISC SUBCHNL?
1

LIBR INPT?
PT
```

```
PRAM INPT?
TY

*EOT

 NO UNDEF EXTS

ENTER PROG PARAMETERS

$EX05,0
$EX10,0
$EX18,0
$SRCH,0
DVR02,0
DVR12,0
/E

# LINKS?
803

SYSTEM

DISCM       02000

$EX05       05027

$EX10       05112

$EX18       05450

$SRCH       06273

$PFAL       06730

$SETP       06733

DVR02       06754

DVR05       07201

DVR12       07450

DVR31       10035
```

```
* EQUIPMENT TABLE ENTRY

11,DVR05,R
13,DVR01
14,DVR31,D,R
16,DVR02,R
20,DVR12,R
21,DVR11,D
22,DVR23,D
/E


* DEVICE REFERENCE TABLE

 1 = EQT #?
1
 2 = EQT #?
3
 3 = EQT #?
3
 4 = EQT #?
4
 5 = EQT #?
2
 6 = EQT #?
5
 7 = EQT #?
6
 8 = EQT #?
7
 9 = EQT #?
/E


* INTERRUPT TABLE

11,EQT,1
13,EQT,2
15,EQT,3
16,EQT,4
20,EQT,5
21,EQT,6
23,EQT,7
/E
```

EXEC SUPERVISOR MODULES

```
$EX01       11021
  $ADDR     11106

$EX02       11021
  $ADDR     11111

$EX03       11021

$EX04       11021
  ASCII     11412

$EX06       11021
  $ADDR     11123

$EX07       11021
  $ADDR     11205

$EX08       11021
  $ADDR     11175

$EX09       11021
  ASCII     11421

$EX11       11021

$EX12       11021

$EX13       11021
  ASCII     11373

$EX14       11021
  ASCII     11504

$EX15       11021
  ASCII     11371

$EX16       11021

$EX17       11021
  $LBL      11411

$EX19       11021
  $LBL      11414

$EX20       11021
```

I/O DRIVER MODULES

```
DVR11       11666

DVR23       11666

DVR01       11666
```

```
LWA SYS LINKS  00646

FWA USER LINKS?
646


LWA SYS  12605

FWA USER?
14000

USER SYSTEM PROGRAMS

JOBPR       14000

LOADR       14000
  .EAU.     25113
  DUMRX     25163

ASMB        14000

ASMBD       21131

ASMB1       21131

ASMB2       21131

ASMB3       21131

ASMB4       21131

ASMB5       21131

XREF        14000
  .OPSY     17230
  DUMRX     17270

FTN4        14000

F4.0        27170

F4.1        27170

F4.2        27170

ALGOL       14000
  .EAU.     26312
  %WRIT     26362
  SREAD     27063
  DUMRX     27621
  .OPSY     27701
```

```
ALGL1       27741

XDISC       14000
  .SWCH     16620
  FMTIO     16637
  INDEX     20051
  .PRAM     20227
  EXECX     20337
  INITX     20363
  FLIB      20422
  .FLUN     20525
  .XFER     20546
  DBLE      20612
  SNGL      20647
  FRMTR     20715
  .OPSY     23455
  .EAU.     23515
  DUMRX     23565




  .ZRLB     23645
  .XPAK     23706
  .PACK     24077
  .XCOM     24213

*SYSTEM STORED ON DISC
```

USING DSGEN TO FORMAT DISCS

Before a fresh disc cartridge or pack can be used as a disc in DOS-III, it must be formatted by DSGEN. System discs (including a possible User Area) are formatted during system generation, but dedicated user discs must be formatted by running DSGEN again in a special mode. Formatting a disc involves assigning it a system generation code, reading every sector, clearing any existing user or system directory, etc. The result is a unlabeled user disc ready for use in DOS-III.

Operating Instructions

1. Turn on all equipment.

2. Unprotect the disc.

3. Load a configured DSGEN using the main-memory resident Bootstrap Loader.

4. Set up a starting address at location 100_8.

5. Set switch register bit 15 equal to 1.

6. Start the computer executing.

7. DSGEN asks for a decimal number to be written on the disc label. This number is used for identification . . . . . *SYS GEN CODE?*

   Operator responds with a decimal number *79*

8. DSGEN requests the octal channel number (select code) of the disc controller . . . . . *SYS DISC CHANNEL?*

   Operator responds with the appropriate octal number *10*

9. DSGEN asks the number of sectors per hardware track on the disc (this is half the number of sectors on a software track) . . . . . *# SECTORS/TRACK*

   Operator responds with 12 (for the 2870 disc) or 23 (for the 2883 disc) or 24 (for the 7900/7901 disc) *24*

10. DSGEN requests the subchannel number (0 to 7) of the user disc to be formatted . . . . . *USER DISC SUBCHANNEL?*

    Operator responds with a number between 0 and 7 inclusive *3*

11. DSGEN requests that the disc be unprotected (if it is still protected) . . . . . *TURN ON DISC PROTECT OVERRIDE — PRESS RUN*

    Operator unprotects the disc and starts the computer executing.

12. DSGEN carries out formatting on the specified subchannel and halts with a code of 102007_8.

13. This procedure should be repeated for each proposed user disc.

    Operator can start the computer to format a new disc of the same type (switch bit 15 must still be equal to 1). DSGEN repeats from . . . . . *USER DISC SUBCHANNEL?*

    Operator can set switch bit 15 equal to 0 and start the computer to proceed to system generation.

SECTION XI
Loading DOS-III

This section describes the loaders used to load a generated DOS-III system into main memory. In this section are

- An introduction to the main-memory resident Bootstrap Loader and the Stand-alone Bootstrap Loader
- The following operating procedures:

  "USING THE MAIN-MEMORY RESIDENT BOOTSTRAP LOADER TO LOAD ABSOLUTE BINARY PROGRAMS"

  "INITIATING DOS-III WITH THE MAIN-MEMORY RESIDENT BOOTSTRAP LOADER"

  "CONFIGURING THE DOS-III STAND-ALONE BOOTSTRAP LOADER"

  "INITIATING DOS-III WITH THE STAND-ALONE BOOTSTRAP LOADER"

- Tables presenting the addresses and contents of these main-memory resident loaders:

  HP 7900/7901 main-memory resident loader

  HP 2883 main-memory resident loader

  HP 2870 main-memory resident loader

To load a generated DOS-III system from the disc into main memory, the user executes either the main-memory resident Bootstrap Loader or the Stand-alone Bootstrap Loader. The former resides in the last 64_{10} words of main memory and is hardware protected. The main-memory resident Loader exists in three versions depending upon the type of disc included in the system (HP 7900/7901, HP 2870, HP 2883). Operation of these three loaders is essentially the same: They consist of two parts; a basic binary loader which loads absolute binary programs into main memory (from paper tape devices), and a disc loader which loads the configured DOS-III system from the disc into main memory.

The main-memory resident Loader loads the system from any active subchannel, with one major requirement: whether that particular system is loaded or not, a configured DOS-III system must exist on the disc starting at head 0, drive 0 of the disc device. Head 0, drive 0 corresponds to subchannel 0 on the HP 2883 disc, or to subchannel 1 on the 7900/7901 and 2870 discs. The main-memory resident Loader will read that system or any other configured DOS-III system on the disc as long as a configured system resides on head 0, drive 0.

To load a configured DOS-III system when no system exists on head 0, drive 0, the user must load the Stand-alone Bootstrap Loader into main memory (using the paper tape portion of the main-memory resident Loader) and execute the Stand-alone Bootstrap Loader. This program loads the configured DOS-III system from the specified disc subchannel without the existence of a configured system on head 0, drive 0 of the disc.

USING THE MAIN-MEMORY RESIDENT BOOTSTRAP LOADER TO LOAD ABSOLUTE BINARY PROGRAMS

The main-memory resident Bootstrap Loader loads absolute binary program tapes into main memory. The Loader resides in the last 64_{10} words of main memory.

Operating Instructions

1. Halt the computer.
2. Place the tape to be loaded into the paper tape input device and ready that device.
3. Set the Loader starting address according to the memory size of the computer:

Memory Size	Starting Address (octal)
12K	027700
16K	037700
24K	057700
32K	077700

4. Clear the switch register.
5. Enable the Loader
6. Press both PRESET buttons.
7. Press RUN.
8. After all or part of the tape is read, the computer halts with $1020xx_8$ displayed.

If xx = 11, a checksum error was detected. Check for torn tape or dust in the reader, check the tape for ragged edges or torn holes, then return to step 2.

If xx = 55, an address error was detected. A program being loaded attempted to enter a location reserved for the main-memory resident Loader, or a location not available in the computer. Check that an absolute binary tape was used, and that it was placed properly in the reader.

If xx = 77, the tape was loaded correctly.

INITIATING DOS-III WITH THE MAIN-MEMORY RESIDENT BOOTSTRAP LOADER

When DOS-III has been generated on the disc (by DSGEN), it can be loaded into main memory and initiated by a main-memory resident program called the main-memory resident Bootstrap Loader. This program resides permanently in the last 64_{10} words of main memory and is hardware protected. Once DOS-III has been loaded and initiated, it is ready to process user tasks.

Operating Instructions

1. Verify that a configured DOS-III system resides on head 0, drive 0 of the disc. (Head 0, drive 0 corresponds to subchannel 1 for the HP 7900/7901 and the HP 2870 discs, or to subchannel 0 for the HP 2883 disc.) If a configured system does not reside there, then use the Stand-alone Bootstrap Loader program (see *"Initiating DOS-III with the Stand-alone Bootstrap Loader,"* in this Section).

2. Set a starting address of 0x7750, where x = 2 for 12K; x = 3 for 16K; x = 5 for 24K; x = 7 for 32K.

3. If the disc being used is the HP 2870, clear the A register; otherwise, ignore this step and continue with step 4.

4. Enable (unprotect) the main-memory resident Loader.

5. Press PRESET button(s) and start the computer executing.

6. The computer halts with 102077_8 displayed in the Display register. Protect the main-memory resident Loader (if necessary).

7. Set the disc subchannel number of the system to be loaded into the switch register (bits 2 through 0).

8. Start computer execution. The system is loaded into main memory and prints the following message:

   INPUT :DATE, XXXXXXXXXX (No Time-base Generator)

   or

   INPUT :DATE, XXXXXXXXXX,H,M (Time-base Generator)

9. All other directives are ignored until a valid DATE directive is entered. Immediately following the DATE directive, the only valid directives are :TRACKS, :BATCH, :TYPE, and :JOB. All other directives are ignored until a JOB directive is entered.

CONFIGURING THE DOS-III STAND-ALONE BOOTSTRAP LOADER

Once DOS-III has been generated onto a disc, it may be initiated into operating status using the DOS-III Stand-alone Bootstrap. The Bootstrap, however, must be configured before being used.

Operating Instructions

1. Turn on all equipment.

2. Load (using the main-memory resident Bootstrap Loader) and configure the SIO Punch or Teleprinter Driver.

3. Load the Bootstrap with the main-memory resident Bootstrap Loader. (See *"Using the Main-memory Bootstrap Loader to Load Absolute Binary Programs"* in this section.)

4. Set up the Bootstrap configuration starting address at location 2_8.

5. Set switch register bits 5 through 0 equal to the octal channel number (select code) of the disc controller (low number, high priority channel).

6. Set switch register bit 15 on to punch a configured Bootstrap tape; off to configure the Bootstrap in main memory only.

7. Start the computer executing.

8. If bit 15 of the switch register is set, the Bootstrap punches out a configured copy of itself and halts. For another copy, simply start the computer executing again.

INITIATING DOS-III WITH THE STAND-ALONE BOOTSTRAP LOADER

When DOS-III has been generated onto the disc, it can be loaded into main memory and initiated by using a small stand-alone program called the Stand-alone Bootstrap Loader. Once DOS-III has been loaded and initiated, it is ready to process user tasks.

> *Note: The Stand-alone Bootstrap Loader need be used only if a configured DOS-III system does not reside on head 0, drive 0 of the disc. If a system resides on the disc in the above mentioned area, the main-memory resident Bootstrap Loader can be used.*

Operating Instructions

1. Turn on all equipment.

2. Configure a Stand-alone Bootstrap Loader (as previously described).

3. Load the configured Bootstrap into main memory using the main-memory resident Bootstrap Loader. (See "*Using the Main-memory Resident Loader to Load Absolute Binary Programs*" in this section.)

4. Set up the starting address of the Bootstrap at location 100_8.

5. Set switch register bits 2 through 0 equal to the octal subchannel of the system disc. (If this subchannel differs from that established at system generation time, the new subchannel overrides the old.)

6. Start the computer executing.

7. When DOS-III has been loaded into main memory, it prints the following message:

   INPUT :DATE, XXXXXXXXXX (no Time-base Generator)

   or

   INPUT :DATE, XXXXXXXXXX,H,M (Time-base Generator present)

8. All other directives are ignored until a valid DATE directive is entered. Immediately following the DATE directive, the only valid directives are :TRACK, :BATCH, :TYPE, and :JOB. All others are ignored until a JOB directive is entered.

MAIN-MEMORY RESIDENT LOADERS

The main-memory resident Loader resides in the last 64_{10} words of main memory (hardware protected by a button/switch on the computer front panel) and is responsible for loading main-memory resident modules from configured DOS-III systems residing on the disc into main memory. The main-memory resident Loader also loads absolute binary programs into main memory through the paper tape input device. A separate version of the main-memory resident Loader exists for each of three classes of disc, depending upon which disc type is used with the system—HP 7900/7901, HP 2883, or HP 2870. Only one version can exist in main memory at any one time. The following three tables show the last 64_{10} word addresses and their octal contents for each version of the main-memory resident Loader.

Table 11-1. HP 7900/7901 Main-memory Resident Loader

Address	Contents	Address	Contents
x7700	002701	x7740	1023kk
x7701	063722	x7741	027740
x7702	002307	x7742	1064kk
x7703	102077	x7743	002041
x7704	017735	x7744	127735
x7705	007307	x7745	005767
x7706	027702	x7746	027737
x7707	077733	x7747	030000
x7710	017735	x7750	002400
x7711	017735	x7751	1026cc
x7712	074000	x7752	1037cc
x7713	077734	x7753	067747
x7714	067734	x7754	1066dd
x7715	047777	x7755	1037dd
x7716	002040	x7756	1066cc
x7717	102055	x7757	063776
x7720	017735	x7760	102606
x7721	040001	x7761	067732
x7722	177734	x7762	106602
x7723	037734	x7763	1037cc
x7724	000040	x7764	102702
x7725	037733	x7765	106602
x7726	027714	x7766	013741
x7727	017735	x7767	1026dd
x7730	054000	x7770	1037cc
x7731	027701	x7771	103706
x7732	102011	x7772	1037dd
x7733	000000	x7773	1023dd
x7734	000000	x7774	027773
x7735	000000	x7775	127717
x7736	006600	x7776	1200cc
x7737	1037kk	x7777	1n0100

x = 2 for 12k, 3 for 16k, 4 for 20k, 5 for 24k, 6 for 28k, 7 for 32k

kk = tape input device select code

dd = low priority (higher numbered) disc select code

cc = high priority (lower numbered) disc select code

n = 5 for 12k, 4 for 16k, 3 for 20k, 2 for 24k, 1 for 28k, 0 for 32k

Table 11-2. HP 2883 Main-memory Resident Loader

Address	Contents	Address	Contents
x7700	002701	x7740	1023kk
x7701	063722	x7741	027740
x7702	002307	x7742	1064kk
x7703	102077	x7743	002041
x7704	017735	x7744	127735
x7705	007307	x7745	005767
x7706	027702	x7746	027737
x7707	077733	x7747	177600
x7710	017735	x7750	063775
x7711	017735	x7751	1026dd
x7712	074000	x7752	1037dd
x7713	077734	x7753	1023dd
x7714	067734	x7754	027753
x7715	047777	x7755	067776
x7716	002040	x7756	106606
x7717	102055	x7757	067732
x7720	017735	x7760	106602
x7721	040001	x7761	102702
x7722	177734	x7762	067747
x7723	037734	x7763	106602
x7724	000040	x7764	001000
x7725	037733	x7765	1067dd
x7726	027714	x7766	1026dd
x7727	017735	x7767	1037cc
x7730	054000	x7770	103706
x7731	027701	x7771	1037dd
x7732	102011	x7772	1023dd
x7733	000000	x7773	027772
x7734	000000	x7774	127717
x7735	000000	x7775	020000
x7736	006600	x7776	1200cc
x7737	1037kk	x7777	1n0100

x = 2 for 12k, 3 for 16k, 4 for 20k, 5 for 24k, 6 for 28k, 7 for 32k

kk = tape input device select code

dd = low priority (higher numbered) disc select code

cc = high priority (lower numbered) disc select code

n = 5 for 12k, 4 for 16k, 3 for 20k, 2 for 24k, 1 for 28k, 0 for 32k

Table 11-3. HP 2870 Main-memory Resident Loader

Address	Contents	Address	Contents
x7700	002701	x7740	1023kk
x7701	063722	x7741	027740
x7702	002307	x7742	1064kk
x7703	102077	x7743	002041
x7704	017735	x7744	127735
x7705	007307	x7745	005767
x7706	027702	x7746	027737
x7707	077733	x7747	030000
x7710	017735	x7750	1026cc
x7711	017735	x7751	1037cc
x7712	074000	x7752	067747
x7713	077734	x7753	1066dd
x7714	067734	x7754	1037dd
x7715	047777	x7755	1066cc
x7716	002040	x7756	063776
x7717	102055	x7757	102606
x7720	017735	x7760	067732
x7721	040001	x7761	106602
x7722	177734	x7762	1037cc
x7723	037734	x7763	102702
x7724	000040	x7764	106602
x7725	037733	x7765	013741
x7726	027714	x7766	1067dd
x7727	017735	x7767	1026dd
x7730	054000	x7770	1037cc
x7731	027701	x7771	103706
x7732	102011	x7772	1037dd
x7733	000000	x7773	1023dd
x7734	000000	x7774	027773
x7735	000000	x7775	127717
x7736	006600	x7776	1200cc
x7737	1037kk	x7777	1n0100

x = 2 for 12k, 3 for 16k, 4 for 20k, 5 for 24k, 6 for 28k, 7 for 32k

kk = tape input device select code

dd = low priority (higher numbered) disc select code

cc = high priority (lower numbered) disc select code

n = 5 for 12k, 4 for 16k, 3 for 20k, 2 for 24k, 1 for 28k, 0 for 32k

SECTION XII
User-written EXEC Modules

DOS-III is capable of accepting user-written EXEC modules. Up to two EXEC modules may be written; these must be loaded with all the DOS-III EXEC modules during DOS-III Generation. (See Section X, "Generating DOS-III" for details.)

This section presents the user-written EXEC call directives and calling sequences, along with a brief description of internal design and a sample EXEC module.

USER EXEC MODULES: DIRECTIVES

Purpose

To execute user EXEC modules.

Format

:EA[,p1,. . . ,p5]	*(Calls EXEC module $EX36)*
:EB[,p1,. . . ,p5]	*(Calls EXEC module $EX37)*

where all parameters are non-negative decimal integers.

Comments

Number and meaning of the parameters varies depending upon user definition of the EXEC module.

USER EXEC MODULES: EXEC CALLS

Purpose

To execute either user-created EXEC module $EX36 or $EX37. The number of parameters in the EXEC call are defined by the user. The general format of the call is

Assembly Language

```
                EXT   EXEC
                 .
                 .
                 .
                JSB   EXEC        (Transfer control to DOS-III)
                DEF   *+2 (to 7)  (Determine number of parameters—from 1 to 5)
                DEF   RCODE       (Define request code)
                DEF   PRAM1       (Define the first optional parameter)
                 .
                 .
                 .
                DEF   PRAM5       (Define the fifth optional parameter)
                 .
                 .
                 .
RCODE           DEF   27 (or 28)  (RCODE for $EX36 = 27; RCODE for $EX37 = 28)
PRAM1           - - -             (Up to five words of parameter information)
                 .
                 .
                 .
PRAM5           - - -
```

FORTRAN

```
IRCDE = 27 (or 28)
CALL EXEC (IRCDE[,P1, . . . . . .P5] )
```

USER EXEC MODULES: INTERNAL DESIGN

EXEC modules are typically type-1 Assembly-language routines which are incorporated at generation time as part of the operating system. As "system" modules, they execute with the interrupt system and memory protect off. They may directly access entry points and subroutines within the system, but must not issue any EXEC calls (EXEC processing is not re-entrant). Also, user-written EXEC modules should be defined as disc-resident supervisory modules; the NAM pseudo-instruction for these modules should indicate that the routine is a type-1 program.

Special programming considerations are required upon initiation and completion.

Initiation

Upon entry, information used in processing the EXEC function can be found in the following base page locations.

Location	Name	Definition
224_8	RQCNT	# of parameters in the calling sequence
225_8	RQRTN	return address upon completion
226_8	RQP1	address of request code
$227\text{-}233_8$	RQP2-RQP6	address(es) of specified parameters

Completion

Prior to returning to the system, the EXEC module must

1. release itself from the EXEC module overlay area if it is disc-resident. This code handles EXEC module release:

```
                LDA   EXMOD    (Get current module in overlay area)
                CPA   NUMB     (Is it this one?)
                CMA,INA        (Yes—set value positive)
                STA   EXMOD    (No—leave value alone)
                 .
                 .
                 .
EXMOD           EQU   245B
NUMB            DEC   -36 (or -37)
```

2. place the desired transfer address in XIRT (location 137_8) and jump to the label $IRT (defined as an EXTernal), for example,

```
            EXT   $IRT
             .
             .
             .
            LDA   RQRTN    (Set the return address)
            STA   XIRT
            JMP   $IRT     (Transfer to system)
RQRTN       EQU   225B
XIRT        EQU   137B
```

SAMPLE EXEC MODULE

```
 PAGE 0001

0001                ASMB,L,C,X,N,R,B   DISC WORK LIMITS MODULE (SEX02)
**  NO ERRORS*
```

```
0001                    ASMB,L,C,X,N,R,B     DISC WORK LIMITS MODULE ($EX02)
0002  00000                     NAM $EX02,1
0003                            ENT $EX02
0004                            EXT $RQER,$ADDR
0005                            EXT $IRT
0006*     $EX02 ROUTINE PROVIDES THE USER WITH DISC WORK AREA TRACK
0007*   ADDRESS LIMITS AND THE # OF SECTORS PER DISC TRACK.
0008*
0009*     CALLING SEQUENCE:
0010*
0011*          JSB EXEC
0012*          DEF *+5(OR 6)
0013*          DEF RCODE       RCODE = 17
0014*          DEF FTRAK       FTRAK = ADDR OF WORD TO STORE 1ST WORK TRK
0015*          DEF LTRAK       LTRAK = ADDR OF WORD TO STORE LAST WORK TRK
0016*          DEF SIZE        SIZE = # SECTOR/TRACK WORD ADDR.
0017*          DEF DISC(OPTIONAL)  DISC = 0 FOR SYSTEM DISC, NON-0 FOR US
0018*                              DEFAULT IS SYSTEM.
0019*
0020  00000 060224  $EX02 LDA RQCNT     CHECK PARAMETER COUNT
0021  00001 050057        CPA ..+4      4 PARAMETERS?
0022  00002 026010R       JMP CHK       YES, OK.
0023  00003 050060        CPA ..+5     5 PARAMETERS?
0024  00004 002001        RSS          YES, OK
0025  00005 026065R       JMP RQER       TOO FEW OR TOO MANY PARAMETERS.
0026  00006 060232        LDA RQP5      CHECK ADDR OF 5TH PARAM
0027  00007 016002X       JSB $ADDR
0028  00010 060227  CHK   LDA RQP2
0029  00011 016002X       JSB $ADDR          PARAMETERS
0030  00012 060230        LDA RQP3
0031  00013 016002X       JSB $ADDR
0032  00014 060231        LDA RQP4
0033  00015 016002X       JSB $ADDR
0034  00016 064224        LDB RQCNT
0035  00017 054057        CPB ..+4      DEFAULT AS SYSTEM DISC?
0036  00020 026024R       JMP SYS       YES,
0037  00021 160232        LDA RQP5,I     NO, CHECK 5TH PARAM
0038  00022 002002        SZA          0 MEANS SYSTEM DISC
0039  00023 026057R       JMP USER
0040  00024 060160  SYS   LDA SYNTS    GET START OF WORK AREA TRACK
0041  00025 040074        ADA .377
0042  00026 001727        ALF,ALF
0043  00027 010074        AND .377
0044  00030 170227        STA RQP2,I
0045  00031 060102        LDA JBINC
0046  00032 002003        SZA,RSS
0047  00033 026043R       JMP EX010
0048  00034 010074        AND .377
0049  00035 070001        STA B
0050  00036 020102        XOR JBINC
0051  00037 001727        ALF,ALF
0052  00040 010074        AND .377
```

```
   PAGE 0003 #01

0053   00041 040052          ADA N1
0054   00042 026045R         JMP EX020
0055   00043 060154  EX010 LDA DISCO     STORE END OF WORK AREA TRACK #
0056   00044 010074          AND .377
0057   00045 170230  EX020 STA RQP3,I
0058   00046 060116          LDA SECTR     STORE # OF SECTORS PER TRACK
0059   00047 170231          STA RQP4,I
0060   00050 060245          LDA EXMOD
0061   00051 050051          CPA ..-2
0062   00052 003004          CMA,INA
0063   00053 070245          STA EXMOD
0064   00054 060225          LDA RQRTN     SET UP TRANSFER ADDR FOR SIRT
0065   00055 070137          STA XIRT
0066   00056 026003X         JMP $IRT
0067   00057 060157  USER  LDA UDNTS     GET USER DISC NEXT TR/SECTR
0068   00060 040074          ADA .377
0069   00061 001727          ALF,ALF
0070   00062 010074          AND .377
0071   00063 170227          STA RQP2,I
0072   00064 026043R         JMP EX010
0073   00065 002400  RQER  CLA             FREE MODULE AREA
0074   00066 070245          STA EXMOD
0075   00067 026001X         JMP $RQER
0076*
0077   00000           A     EQU 0
0078   00001           B     EQU 1
0079   00053           ..    EQU 53B
0080   00052           N1    EQU ..-1
0081   00074           .377  EQU ..+17
0082   00100           .     EQU 100B
0083   00102           JBINC EQU .+2
0084   00116           SECTR EQU .+14
0085   00126           RONBF EQU .+26B
0086   00137           XIRT  EQU RONBF+9
0087   00154           DISCO EQU .+44
0088   00157           UDNTS EQU .+47
0089   00160           SYNTS EQU .+48
0090   00224           RQCNT EQU .+84
0091   00225           RQRTN EQU .+85
0092   00227           RQP2  EQU .+87
0093   00230           RQP3  EQU .+88
0094   00231           RQP4  EQU .+89
0095   00232           RQP5  EQU .+90
0096   00245           EXMOD EQU .+101
0097                   END
**  NO ERRORS*
```

```
    SEX02          CROSS-REFERENCE SYMBOL TABLE          PAGE 0001

$ADDR   00004    00027    00029    00031    00033

$EX02   00020    00003

$IRT    00005    00066

$RQER   00004    00075

        00082    00083    00084    00085    00087    00088    00089
        00090    00091    00092    00093    00094    00095    00096

        00079    00021    00023    00035    00061    00080    00081

 377    00081    00041    00043    00048    00052    00056    00068
        00070

A       00077

B       00078    00049

CHK     00028    00022

DISCO   00087    00055

EX010   00055    00047    00072

EX020   00057    00054

EXMOD   00096    00060    00063    00074

JBINC   00083    00045    00050

N1      00080    00053

RONBF   00085    00086

RQCNT   00090    00020    00034

RQER    00073    00025

RQP2    00092    00028    00044    00071

RQP3    00093    00030    00057

RQP4    00094    00032    00059

RQP5    00095    00026    00037

RQRTN   00091    00064

SECTR   00084    00058

SYNTS   00089    00040

SYS     00040    00036
```

```
  $EX02        CROSS-REFERENCE SYMBOL TABLE        PAGE 0002

UDNTS   00088     00067
USER    00067     00039
XIRT    00086     00065
```

SECTION XIII
Planning I/O Drivers

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

An I/O driver, operating under control of the Input/Output Control ($EX18) and Central Interrupt Control ($CIC) modules of DOS-III, is responsible for all data transfer between an I/O device and the computer. During its execution, the driver may refer to the base page communication area for information from the system: the device equipment table (EQT) entry, which contains the parameters of the transfer, and the current DMA value (CHAN), which 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 I/O control module ($EX18) calls the initiation section directly when an I/O transfer is initiated. Locations EQT1 through EQT17 of the base page communication area contain the addresses of the appropriate EQT entry. CHAN in the base page contains the number of the DMA channel assigned to the device, if needed. This section is entered by a jump subroutine (JSB) to the entry point I.*nn*. On entry, the A register contains the select code (channel number) of the device (bits 0 through 5 of EQT entry word 3). The driver returns to $EX18 by an indirect jump through I.*nn*.

Before transferring to I.*nn*, DOS-III places the request parameters from the user program's EXEC call into words 7 through 13 of the EQT entry. Word 9, CONWD, is modified to contain the request code in bits 0 through 5 in place of the logical unit. (See Figure A-4 and Section III, I/O READ/WRITE EXEC Call (RCODE = 1 or 2), for details of the parameters.)

Once initiated, the drive can use words 5, 6, and 11 through 14 of the EQT entry in any way, but words 1, 2, 3, 7, 8, 9, 10, 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 is responsible for these functions (as flow-charted in Figure 13-1):

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.

   Note: All drivers must accept a clear request. (Request code = 3, function code = 0.)

2. Configures all I/O instructions in the driver to include the select code of the device (or DMA channel). (Does not apply to DVR05 and 2870/7900/7901 DVR31.)

3. Initializes DMA, if appropriate.

   Note: The initiation section must save the DMA channel number (found in CHAN) in the EQT entry, since it is not set on entry to the continuation section.

4. Initializes software flags and activates the device. All variable information pertinent to the transmission must be saved in the EQT entry because the driver may be called for another device before the first operation is complete.

5. Returns to $EX18 with the A register set to indicate initiation or rejection and the cause of the reject:

   If A = 0, then the operation was initiated.

   If A ≠ 0, then the operation was rejected with A set as:

   1 = read or write illegal for device

   2 = control request illegal or undefined

   3 = equipment malfunction or not ready

   4 = immediate completion (for control requests)

   6 = driver cannot handle a control request; the system is instructed to wait

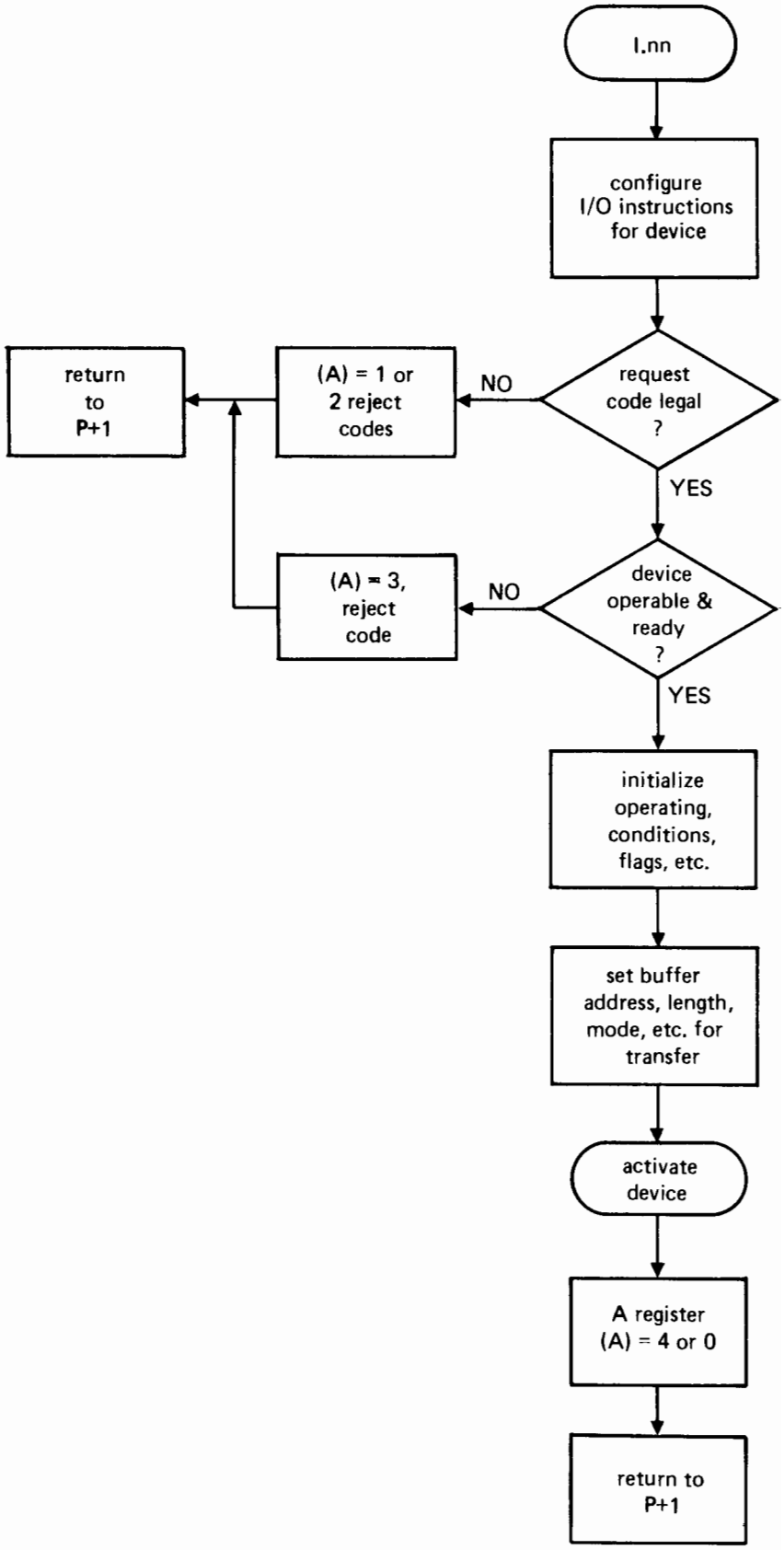

Figure 13-1. I/O Driver Initiation Section

COMPLETION SECTION

DOS-III 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 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 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).

FUNCTIONS OF THE COMPLETION SECTION

The completion section of the driver is responsible for the functions below (as flow-charted in Figure 13-2):

1. The driver configures all I/O instructions in the completion section to reference the interrupting device.

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 the Equipment Table. The return to $CIC must be (P+2) as in step 6.

5. At the end of a successful transfer or after completing the retry procedure, the following information must be set before returning to $CIC at (P+1):

   a. Set the actual or simulated device status into bits 0 through 7 of EQT word 4.

   b. Set the number of transmitted words or characters (depending on which the user requested) in the B register.

   c. Set the A register to indicate successful or unsuccessful completion.

      0 = successful completion

      1 = device malfunction or not ready

      2 = end-of-tape (information)

      3 = transmission parity error

6. Clear the device and DMA control on end-of-operation, or set the device and DMA for the next transfer or retry. Return 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.

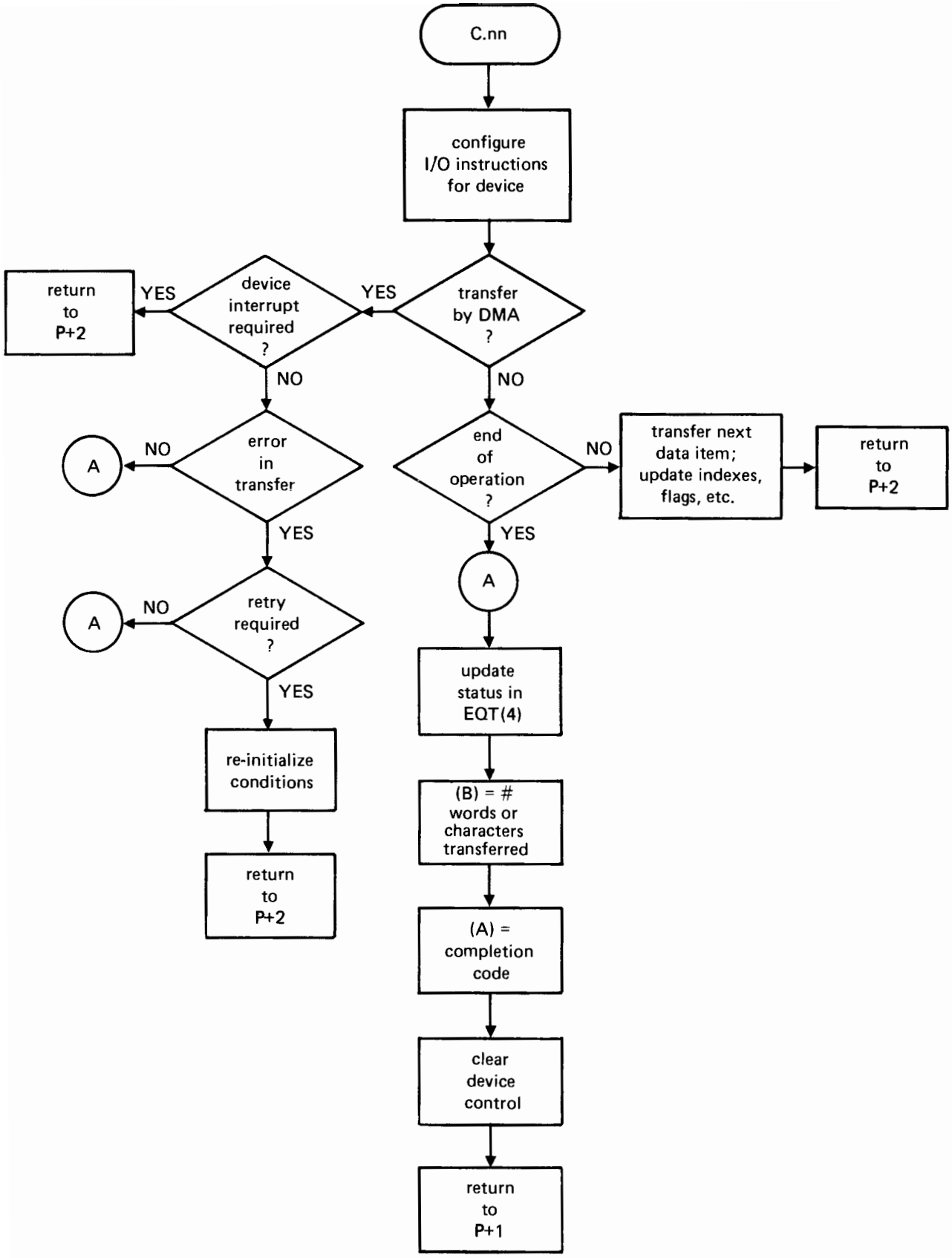

Figure 13-2. I/O Driver Completion Section

SAMPLE I/O DRIVER

The following pages provide an assembly listing and cross-reference symbol table for a sample I/O driver.

```
 PAGE 0001

0001                    ASMB,R,B,L,C
**  NO ERRORS*
```

```
0001                    ASMB,R,B,L,C
0003  00000                    NAM DVR02,4
0004*
0005**********   VERSION 8/24/72
0006*
0007*
0008                         ENT I.02,C.02
0009*
0010****** PROGRAM  DESCRIPTION *******
0011*
0012*    DRIVER 02 OPERATES UNDER THE CONTROL OF THE
0013* I/O CONTROL MODULE OF THE D.O.S. EXECUTIVE
0014* THIS DRIVER IS RESPONSIBLE FOR CONTROLLING
0015* OUTPUT DATA TRANSMISSION WITH A 2753A TAPE PUNCH.
0016* <02> IS THE EQUIPMENT TYPE CODE ASSIGNED TO THIS
0017* TYPE OF DEVICE. I.02 IS THE ENTRY POINT FOR THE
0018* *INITIATION* SECTION AND C.02 FOR THE *COMPLETION*
0019* SECTION.
0020*
0021*  -  THE INITIATION SECTION IS CALLED FROM I/O
0022*     CONTROL TO INITIALIZE A DEVICE AND INITIATE
0023*     AN OUTPUT OR CONTROL OPERATION.
0024*
0025*      CALLING SEQUENCE:
0026*
0027*           - ADDRESSES OF DEVICE EQT ENTRY
0028*             SET IN "EQT1-EQT17" -
0029*
0030*           (A) = I/O ADDRESS OF DEVICE
0031*
0032*        (P)   JSB  I.02
0033*        (P+1) - RETURN -
0034*
0035*              (A) = 0, OPERATION INITIATED, OR
0036*              (A) = REJECT CODE:
0037*
0038*                   1, ILLEGAL READ REQUEST
0039*                   2, ILLEGAL CONTROL FUNCTION
0040*
0041*
0042*  -  THE COMPLETION SECTION IS CALLED BY CENTRAL
0043*     INTERRUPT CONTROL TO CONTINUE OR COMPLETE
0044*     AN OPERATION.
0045*
0046*      CALLING SEQUENCE:
0047*
0048*           - ADDRESSES OF DEVICE EQT ENTRY
0049*             SET IN "EQT1-EQT17" -
0050*
0051*           (A) = I/O ADDRESS OF DEVICE
0052*
0053*        (P)   JSB  C.02
0054*        (P+1) -- COMPLETION RETURN   --
0055*        (P+2) -- CONTINUATION RETURN --
0056*
0057*         - COMPLETION RETURN:
```

```
0058*              (A) = 0, SUCCESSFUL COMPLETION WITH
0059*                 (B) = # WORDS OR CHARACTERS
0060*                       TRANSFERRED.
0061*              (A) = 2  IF *TAPE-SUPPLY-LOW* CONDITION
0062*                       DETECTED AFTER RECORD IS
0063*                       FINISHED.
0064*                 (B), SAME AS FOR (A) = 0
0065*
0066*           - CONTINUATION RETURN: REGISTERS
0067*               MEANINGLESS
0068*
0069*
0070*   - RECORD FORMATS:
0071*
0072*       ASCII:  A STRING OF CHARACTERS, THE NUMBER
0073*       -----   DESIGNATED BY THE BUFFER LENGTH IN
0074*               THE REQUEST, TERMINATED BY A RETURN
0075*               AND LINE-FEED (SUPPLIED BY THE DRIVER).
0076*
0077*         SPECIAL CHARACTER PROCESSING:
0078*
0079*           LEFT-ARROW: IF A LEFT-ARROW IS THE LAST
0080*                       CHARACTER IN THE USER BUFFER,
0081*                       THE RETURN/LINE-FEED AND LEFT
0082*                       ARROW CODES ARE NOT OUTPUT.
0083*
0084*            A ZERO BUFFER LENGTH CAUSES ONLY A RETURN/
0085*                LINE-FEED TO BE OUTPUT.
0086*
0087*       BINARY: A STRING OF CHARACTERS SPECIFIED
0088*       ------ BY THE "BUFFER LENGTH" IN THE REQUEST.
0089*
0090*
0091*   -   CONTROL FUNCTIONS ACCEPTED:
0092*
0093*       10 - TEN INCHES OF ZEROS (FEED-FRAMES) ARE
0094*            OUTPUT FOR LEADER/TRAILER.
0095*
0096*           11 - LINE SPACING: THE PARAMETER WORD OF THE
0097*            CONTROL REQUEST DETERMINES THE NUMBER
0098*            OF LINE-FEEDS TO BE OUTPUT.
0099*
```

```
0101*
0102********* INITIATION  SECTION **********
0103*
0104*
0105  00000 000000  I.02  NOP
0106*
0107  00001 016201R       JSB SETIO    SET I/O INSTRUCTIONS FOR UNIT.
0108*
0109  00002 160213        LDA EQT9,I   GET CONTROL WORD OF REQUEST,
0110  00003 010056        AND .3        ISOLATE.
0111*
0112  00004 050054        CPA .1       ERROR IF REQUEST IS
0113  00005 126000R       JMP I.02,I    FOR INPUT, REJECT CALL.
0114  00006 050055        CPA .2       PROCESS FOR
0115  00007 026043R       JMP D04        WRITE REQUEST.
0116*
0117* CONTROL FUNCTION REQUEST
0118*
0119  00010 160213        LDA EQT9,I   GET CONTROL WORD
0120  00011 001727        ALF,ALF       FROM REQUEST, POSITION AND
0121  00012 001222        RAL,RAL       ISOLATE FUNCTION FIELD.
0122  00013 010073        AND MASK1
0123  00014 002003        SZA,RSS       IS IT A CLEAR?
0124  00015 026024R       JMP CLEAR     YES.
0125  00016 050063        CPA .10B     FIELD = <10> TO GENERATE
0126  00017 026026R       JMP D01       LEADER (10 INCHES OF BLANK TAPE)
0127  00020 050064        CPA .11B     FIELD = <11> FOR LINE
0128  00021 026032R       JMP D02       SPACING.
0129*
0130* REQUEST ERROR - CAUSE REJECT RETURN TO I/O CONTROL
0131*
0132  00022 060055        LDA .2
0133  00023 126000R       JMP I.02,I
0134  00024 106700  CLEAR CLC 0        TURN DEVICE OFF
0135  00025 026066R       JMP I.A.6
0136*
0137* LEADER/TRAILER GENERATOR
0138*
0139  00026 062224R D01   LDA N100     SET INDEX COUNTER FOR FEED FRAMES
0140  00027 170216        STA EQT12,I   = -100.
0141  00030 002400        CLA          (A) = 0 FOR
0142  00031 026041R       JMP D03        FEED FRAME.
0143*
0144* LINE SPACING
0145*
0146  00032 160214  D02   LDA EQT10,I  GET LINE COUNTER WORD.
0147  00033 002021        SSA,RSS      INSURE VALUE
0148  00034 003004        CMA,INA       IS NEGATIVE.
0149  00035 002003        SZA,RSS      PROTECT AGAINST
0150  00036 003400        CCA           A ZERO VALUE.
0151  00037 170216        STA EQT12,I
0152  00040 060065        LDA LINF     (A) = LINE FEED CODE.
0153*
0154  00041 170217  D03   STA EQT13,I  SET ACTION CODE.
0155  00042 026056R       JMP D05
0156*
```

```
0157* WRITE REQUEST PROCESSING
0158*
0159  00043 160214  D04   LDA EQT10,I  CONVERT BUFFER ADDRESS TO EVEN
0160  00044 001200        RAL           CHARACTER ADDRESS AND SE
0161  00045 170216        STA EQT12,I   AS CURRENT BUFFER ADDRESS.
0162  00046 160215        LDA EQT11,I   GET BUFFER LENGTH.
0163  00047 002020        SSA           IF CHARACTER SPECIFIED,
0164  00050 026053R       JMP **3        USE VALUE.
0165  00051 001000        ALS           CONVERT WORDS TO NEGATIVE
0166  00052 003004        CMA,INA        CHARACTERS.
0167  00053 170217        STA EQT13,I   SET CURRENT BUFFER LENGTH.
0168*
0169  00054 002400        CLA
0170  00055 170220        STA EQT14,I    FOR BINARY WRITE.
0171*
0172* CALL *COMPLETION* SECTION TO WRITE FIRST CHAR.
0173*
0174  00056 062223R D05   LDA IEXTA     ADJUST RETURN
0175  00057 072070R       STA C.02       TO *INITIATOR* SECTION.
0176  00060 026075R       JMP D10
0177*
0178  00061 026064R       JMP I.A.4        BINARY READ WITH 0 BUFFER LEN.
0179  00062 002400  IEXIT CLA           RETURN TO I/O CONTROL WITH
0180  00063 126000R       JMP I.02,I     OPERATION INITIATED.
0181*
0182  00064 006400  I.A.4 CLB
0183  00065 174220        STB EQT14,I    CLEAR TLOG.
0184  00066 060057  I.A.6 LDA .4        SET A=4 FOR IMMED.COMPL RETURN
0185  00067 126000R       JMP I.02,I     RETURN
```

```
0187*
0188********* COMPLETION  SECTION **********
0189*
0190*
0191  00070 000000  C.02  NOP
0192*
0193  00071 016201R       JSB SETIO    SET I/O INSTRUCTIONS FOR UNIT.
0194  00072 160207        LDA EQT5,I    GET "CLEAR" FLAG.
0195  00073 002020        SSA           CLEAR?
0196  00074 026155R       JMP IO3       YES.TERMINATE.
0197*
0198  00075 160213  D10   LDA EQT9,I   GET CONTROL WORD
0199  00076 070001        STA B         SAVE FOR CODE TEST.
0200  00077 001727        ALF,ALF      ROTATE MODE BIT
0201  00100 001200        RAL           TO BIT 15
0202  00101 072222R       STA TEMP1     AND SAVE.
0203*
0204  00102 002400        CLA          IF CURRENT BUFFER ADDRESS OR
0205  00103 150216        CPA EQT12,I   FUNCTION INDEX = 0, THEN
0206  00104 026155R       JMP IO3       OPERATION COMPLETED.
0207*
0208  00105 004010        SLB
0209  00106 026133R       JMP D11       - CONTROL FUNCTION -
0210*
0211  00107 150217        CPA EQT13,I  IF CURRENT CHARACTER INDEX =
0212  00110 026137R       JMP D12       0, THEN OUTPUT END OF RECORD.
0213*
0214  00111 164216        LDB EQT12,I  GET CURRENT CHAR. BUFFER ADDRESS.
0215  00112 134216        ISZ EQT12,I  ADD 1 FOR NEXT CHARACTER.
0216  00113 004065        CLE,ERB      CONVERT TO WORD ADDRESS.
0217  00114 160001        LDA B,I     GET WORD AND
0218  00115 002041        SEZ,RSS       POSITION PROPER
0219  00116 001727        ALF,ALF       CHARACTER IN A(07-00).
0220  00117 010074        AND MASK3    REMOVE UPPER POSITION DATA.
0221*
0222  00120 066222R       LDB TEMP1    PUT MODE IN B(15).
0223  00121 134217        ISZ EQT13,I  INDEX CHARACTER COUNTER.
0224  00122 026127R       JMP IO1       - NOT LAST CHARACTER.
0225*
0226  00123 006020        SSB          IF BINARY MODE,
0227  00124 026127R       JMP IO1       WRITE LAST CHARACTER.
0228*
0229  00125 052220R       CPA ARROW    IF CHAR = < ↑ >, THEN OMIT IT
0230  00126 026155R       JMP IO3       AND R/LF ON ASCII RECORD.
0231*
0232* OUTPUT CHARACTER TO PUNCH UNIT.
0233*
0234  00127 102600  IO1   OTA 0         OUTPUT CHARACTER TO INTERFACE
0235  00130 102700  IO2   STC 0         TURN DEVICE ON.
0236  00131 036070R       ISZ C.02     ADJUST RETURN TO (P+2).
0237  00132 126070R       JMP C.02,I    -EXIT-.
0238*
0239* CONTROL FUNCTION OUTPUT
0240*
0241  00133 160217  D11   LDA EQT13,I  (A) = LINE-FEED OR FEED FRAME.
0242  00134 134216        ISZ EQT12,I  INDEX OUTPUT COUNT FOR LEADER/
```

```
0243  00135 000000         NOP             TRAILER OR LINE SPACING.
0244  00136 026127R        JMP I01        GO TO OUTPUT CHARACTER.
0245*
0246* END OF RECORD PROCESSING
0247*
0248  00137 062222R D12     LDA TEMP1      CHECK MODE OF TRANSFER.
0249  00140 002020          SSA
0250  00141 026155R         JMP I03         - BINARY -
0251*
0252  00142 164220          LDB EQT14,I    *ASCII* RECORD
0253  00143 062217R         LDA RETN        OUTPUT FIRST A
0254  00144 056217R         CPB RETN         RETURN AND THEN A
0255  00145 060065          LDA LINF         LINE-FEED.
0256  00146 170220          STA EQT14,I    SET EQT11 FOR LINE-FEED CHECK.
0257  00147 056217R         CPB RETN       IF LINE-FEED IS BEING OUTPUT,
0258  00150 026152R         JMP D14         GO TO SET COMPLETION FLAG.
0259  00151 026127R         JMP I01          - OUTPUT  RETURN -
0260*
0261*
0262  00152 006400  D14     CLB            SET BUFFER ADDRESS = 0
0263  00153 174216          STB EQT12,I     TO INDICATE LAST CHARACTER.
0264  00154 026127R         JMP I01
```

```
0266*
0267* STATUS AND TRANSMISSION COMPLETION SECTION
0268*
0269  00155 102500  IO3   LIA 0         GET DEVICE STATUS.
0270  00156 070001        STA B
0271  00157 160206        LDA EQT4,I    REMOVE PREVIOUS
0272  00160 010075        AND MASK2      STATUS.
0273  00161 030001        IOR B         SET NEW
0274  00162 170206        STA EQT4,I     STATUS WORD.
0275  00163 002400        CLA           IF LOW TAPE
0276  00164 006002        SZB            SUPPLY, SET
0277  00165 060055        LDA .2         A = 2 FOR  *EOT*.
0278  00166 164207        LDB EQT5,I     GET "CLEAR" FLAG.
0279  00167 006020        SSB            CLEAR?
0280  00170 026176R       JMP IO5        YES.
0281*
0282  00171 164215        LDB EQT11,I   SET  (B) = TRANSMISSION
0283  00172 006020        SSB            LOG AS POSITIVE # OF WORDS
0284  00173 007004        CMB,INB        OR CHARACTERS.
0285*
0286  00174 106700  IO4   CLC 0          TURN DEVICE OFF.
0287*
0288  00175 126070R       JMP C.02,I     AND EXIT FOR COMPLETION.
0289*
0290  00176 006400  IO5   CLB
0291  00177 174207        STB EQT5,I     RESET "CLEAR" FLAG.
0292  00200 126070R       JMP C.02,I     RETURN
0293*
0294* SUBROUTINE:  <SETIO>
0295*
0296*  PURPOSE: TO CONFIGURE THE I/O INSTRUCTIONS
0297*           IN THE DRIVER TO REFERENCE THE
0298*           SUBJECT PAPER TAPE PUNCH.
0299*
0300*  CALL:       (A)05-00 CONTAINS I/O ADDRESS
0301*        (P)    JSB  SETIO
0302*        (P+1)  -RETURN- (REGISTERS MEANINGLESS)
0303*
0304  00201 000000  SETIO NOP
0305  00202 032221R       IOR LIA       COMBINE <LIA> WITH I/O ADDRESS
0306  00203 072155R       STA IO3        AND SET.
0307*
0308  00204 040067        ADA .100      CONSTRUCT <OTA> INSTRUCTION
0309  00205 072127R       STA IO1
0310*
0311  00206 042215R       ADA .1100     CONSTRUCT  <STC,C> INSTRUCTION
0312  00207 072130R       STA IO2
0313*
0314  00210 032216R       IOR .4000     CONSTRUCT  <CLC> INSTRUCTION
0315  00211 072174R       STA IO4
0316  00212 072024R       STA CLEAR
0317*
0318  00213 126201R       JMP SETIO,I
```

```
0320*
0321* CONSTANT AND VARIABLE STORAGE AREA
0322*
0323  00000          A      EQU 0          DEFINE SYMBOLIC REFERENCE FOR
0324  00001          B      EQU 1           A AND B REGISTERS.
0325*
0326  00214 000040   .40    OCT 40
0327  00215 001100   .1100  OCT 1100
0328  00216 004000   .4000  OCT 4000
0329*
0330*
0331  00217 000015   RETN   OCT 15
0332  00220 000137   ARROW  OCT 137
0333*
0334  00221 102500   LIA    LIA 0
0335*
0336  00222 000000   TEMP1  NOP
0337*
0338  00223 000061R  IEXTA  DEF IEXIT-1
0339  00224 177610   N100   DEC -120
0340*
```

```
0342*
0343*** SYSTEM BASE PAGE  COMMUNICATION  AREA ***
0344*
0345  00053          ..     EQU 53B
0346  00047          N4     EQU ..-4
0347  00054          .1     EQU ..+1
0348  00055          .2     EQU ..+2
0349  00056          .3     EQU ..+3
0350  00057          .4     EQU ..+4
0351  00061          .6     EQU ..+6
0352  00063          .10B   EQU ..+8
0353  00064          .11B   EQU ..+9
0354  00065          LINF   EQU ..+10
0355  00067          .100   EQU ..+12
0356  00073          MASK1  EQU ..+16
0357  00074          MASK3  EQU ..+17
0358  00075          MASK2  EQU ..+18
0359  00100                 EQU 100B     ESTABLISH ORIGIN OF AREA
0360*
0361*
0362* I/O MODULE/DRIVER COMMUNICATION
0363*
0364  00203          EQT1   EQU .+67
0365  00204          EQT2   EQU .+68
0366  00205          EQT3   EQU .+69
0367  00206          EQT4   EQU .+70
0368  00207          EQT5   EQU .+71
0369  00210          EQT6   EQU .+72
0370  00211          EQT7   EQU .+73
0371  00212          EQT8   EQU .+74
0372  00213          EQT9   EQU .+75
0373  00214          EQT10  EQU .+76
0374  00215          EQT11  EQU .+77
0375  00216          EQT12  EQU .+78
0376  00217          EQT13  EQU .+79
0377  00220          EQT14  EQU .+80
0378  00221          EQT15  EQU .+81
0379  00222          EQT16  EQU .+82
0380  00223          EQT17  EQU .+83
0381                 END
**  NO ERRORS*
```

```
.       00359   00364   00365   00366   00367   00368   00369
        00370   00371   00372   00373   00374   00375   00376
        00377   00378   00379   00380

..      00345   00346   00347   00348   00349   00350   00351
        00352   00353   00354   00355   00356   00357   00358

.1      00347   00112

.100    00355   00308

.10B    00352   00125

.1100   00327   00311

.11B    00353   00127

.2      00348   00114   00132   00277

.3      00349   00110

.4      00350   00184

.40     00326

.4000   00328   00314

.6      00351

.A      00323

ARROW   00332   00229

B       00324   00199   00217   00270   00273

C.02    00191   00008   00175   00236   00237   00288   00292

CLEAR   00134   00124   00316

D01     00139   00126

D02     00146   00128

D03     00154   00142

D04     00159   00115

D05     00174   00155

D10     00198   00176

D11     00241   00209

D12     00248   00212
```

```
 D14     00262    00258

*EQT1    00364

 EQT10   00373    00146    00159

 EQT11   00374    00162    00282

 EQT12   00375    00140    00151    00161    00205    00214    00215
         00242    00263

 EQT13   00376    00154    00167    00211    00223    00241

 EQT14   00377    00170    00183    00252    00256

*EQT15   00378

*EQT16   00379

*EQT17   00380

*EQT2    00365

*EQT3    00366

 EQT4    00367    00271    00274

 EQT5    00368    00194    00278    00291

*EQT6    00369

*EQT7    00370

*EQT8    00371

 EQT9    00372    00109    00119    00198

 I.02    00105    00008    00113    00133    00180    00185

 I.A.4   00182    00178

 I.A.6   00184    00135

 IEXIT   00179    00338

 IEXTA   00338    00174

 IO1     00234    00224    00227    00244    00259    00264    00309

 IO2     00235    00312

 IO3     00269    00196    00206    00230    00250    00306

 IO4     00286    00315

 IO5     00290    00280
```

LIA	00334	00305		
LINF	00354	00152	00255	
MASK1	00356	00122		
MASK2	00358	00272		
MASK3	00357	00220		
N100	00339	00139		
*N4	00346			
RETN	00331	00253	00254	00257
SETIO	00304	00107	00193	00318
TEMP1	00336	00202	00222	00248

SECTION XIV
Privileged Mode

Certain situations may arise where a user wishes to process his own errors instead of having the operating system handle them for him. In addition, there may be cases where he wishes to determine when an I/O operation (initiated without wait) is complete.

Both of these options are available with use of the system's privileged mode flag (MDFLG = location 133_8). In order to operate in this privileged mode (i.e., user processing of I/O errors and/or determining I/O completions) the user

- must be programming in Assembly language
- is responsible for setting MDFLG properly

  Bit 0 set - user error processing

  Bit 15 set - I/O completion processing

DOS-III uses MDFLG as follows:

1. After an I/O initiation (performed by an EXEC call) MDFLG bit 0 is checked, and if it is equal to one, control returns to the user program with the A register set as follows:

Contents (decimal)	Meaning
0	Operation initiated
1	Read or write illegal
2	Control request ignored
3	Device down
4	Immediate completion
5	DMA busy
6	Driver busy
7	Driver overlay area busy
8	EXEC overlay area busy
9	Operation rejected
10	Memory protect error
11	Request code error
12	Execution time exceeded
13	Spare

Contents (decimal)	Meaning
14	Illegal logical unit
15	Unassigned logical unit
16	Illegal buffer address
17	Memory wrap around
18	Illegal track address
19	File cannot be found

2. After an I/O completion, MDFLG bit 15 is checked, and if it is equal to one, control is passed to a user subroutine which must immediately follow the EXEC call. Upon entry to the routine, the B register contains the driver transmission log and the A register contains the device status as follows:

A register Contents	Meaning
0	I/O completed without errors
–1	Device was not ready
–2	End-of-tape
–3	Parity error
–4	Batch input detected a colon (:)

   If the I/O completion resulted from an I/O error (not ready, parity, or end-of-tape) and the device is not the system console or the disc, bit 14 of EQT4 (the fourth word of the current Equipment Table entry) is set to indicate that the device is down.

   MDFLG bit 0 is then checked, and if it is equal to one, control returns to user (thus bypassing system processing of the error).

3. During a FILE NAME SEARCH EXEC call (RCODE = 18) where the search is requested without wait, no subsequent EXEC calls are allowed. If a second EXEC call is requested during execution of a file search, the system will wait for the search to complete before processing the second EXEC call. If the user does not want the system to wait, he should set Bit 1 of MDFLG. If Bit 1 of MDFLG is set and the above condition is encountered, control will be returned to the user following the second EXEC call with the A register = 8 (EXEC busy).

4. The system clears all bits of MDFLG following any program completion.

5. An I/O calling sequence operating in privileged mode might look something like this:

```
        JSB   EXEC
        DEF   END
        DEF   RCODE
        DEF   CONWD     (must be an I/O without wait)
          .
          .
          .
END     JMP   INIT
COMP    NOP             } If present, the completion routine must be located here.
          .             } Executed following I/O completion, and should include
          .             } a check for completion errors. This routine must not
          .             } use any routine that is not re-entrant.
        JMP   COMP,I    }
          .
          .
          .
INIT                    } Executed following an I/O initiation, and should
          .             } include a check for initiation errors.
          .
          .
```

SECTION XV
Halt Codes and Error Messages

This section describes the error conditions which can occur while DOS-III is being generated, loaded and operated. Error conditions are reported to the user by one of the following:

- a computer halt; the halt code is displayed in the DISPLAY register
- an error message; the message is displayed on the system console
- an error message (displayed on the system console) followed by a computer halt (halt code displayed in the DISPLAY register)
- an error code returned to a user program (by EFMP); the error code is also returned in the A register

This section contains halt code and error message tables, including corrective action (when applicable) for the following:

- DSGEN ERROR CONDITIONS
  - DSGEN Error Halts
  - DSGEN Error Messages
- DOS-III BOOTSTRAP ERROR HALTS
- DOS-III ERROR CONDITIONS
  - DOS-III Error Halts
  - DOS-III Error Messages
- EFMP ERROR CODES

Note: The ALGOL, FORTRAN and Assembler subsystems also print error messages. These subsystem error messages are documented in the SOFTWARE OPERATING PROCEDURE module "Assembler, FORTRAN and ALGOL Error Messages" (5951-1377). FORTRAN IV error messages are described in HP FORTRAN IV (5951-1321).

Table 15-1. DSGEN Error Conditions

DSGEN ERROR HALTS

Halt Code	Cause	Recovery Action
102000	Follows an irrecoverable error message. Generator unable to find $STRT in DISCM. DISCM is probably missing.	Irrecoverable Irrecoverable
102002	Follows ERR02.	See ERR02 in error messages.
102003	Follows ERR03.	See ERR03 in error messages.
102004	Follows ERR04.	See ERR04 in error messages.
102007	Normal halt. Disc initialization of sub-channel has completed.	Start the computer executing to initialize another subchannel or to generate a system.
102022	Disc error after ten attempts. Disc address in A, disc status in B.	Start execution to retry ten more times. When preceded by ERR12 continues to next track.
102032	Disc not ready or disc should be unprotected. Disc address in A and disc status in B.	Ready or unprotect the disc. Start the computer executing.
102077	Normal halt. Ready to receive another program tape.	Continue generation. Enter next tape and start the computer executing.
102000	If DSGEN is above 10000_8 an impossible condition has occurred.	Either a hardware/software failure has occurred or DSGEN has overflowed its work area because the system was too large.

DSGEN ERROR MESSAGES

Messages During Initialization and Input Phases

Message	Meaning	Action
ERR01	Invalid response to initialization request.	Request is repeated. Enter valid reply.
ERR02	Checksum error on program input.	Computer halts; to try again, reposition tape to beginning of program and start the computer.
ERR03	Record out of sequence.	Same as ERR02.

Table 15-1. DSGEN Error Conditions (continued)

Message	Meaning	Action
ERR04	Illegal record type.	Same as ERR02. If input is from disc, error is irrecoverable; remove non-relocatable files from disc.
ERR05 name	Duplicate entry point.	The current entry point replaces the previous entry point.
ERR06	Invalid base page length in BCS-produced relocatable tape (must be zero).	Base page area is ignored, but memory protect error will occur if program is executed.
ERR07	Program name or entry point table overflow of available memory.	Irrecoverable error. Revise or delete programs.
ERR08 name	Duplicate program name.	The current program replaces the previous program.
Messages During the Parameter Phase		
ERR09	Parameter name error (no such program).	Enter valid parameter statement.
ERR10	Parameter type error.	Same as ERR09.
General Messages		
ERR11	System directory track overflow.	Irrecoverable.Regenerate system and reduce the value of the response to the "FIRST SYSTEM SECTOR?" message.
ERR12	Disc error during disc initialization.	Start the computer executing to bypass the faulty tracks.
ERR13	User segment precedes user main program.	Irrecoverable.
ERR14	Absolute code overlays relocatable code in the disc scratch area.	Irrecoverable. Regenerate the system and select one of the following two options: 1. Reduce number of programs being loaded 2. Load the library after all other programs are loaded. If this is not successful, increase the size of the system disc and/or lower the starting track/sector of the system.
ERR15	More than 63 subprograms called by a main program.	Revise main program (subsequent calls to subprograms are ignored).

Table 15-1. DSGEN Error Conditions (continued)

Message	Meaning	Action
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 word).	Current word is ignored (the address is printed).
ERR20	Binary DBL record overflow of internal table.	Records overlay previous DBL records (diagnostic printed for each overflow record). Revise program.
ERR21	Module containing entry point $CIC not loaded.	Irrecoverable error. Regenerate the system; include DISCM.
ERR22	Read parity/decode disc error. A register bits 8-14 show track number; bits 0-7 show sector number.	After ten attempts to read or write the disc sector, the computer halts. To try ten more times, start the computer executing.
ERR23	EQT not entered for disc-resident I/O module.	Restart at 100_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 number.	Enter valid DRT statement.
ERR28	Invalid channel number.	Enter valid INT statement.
ERR29	Channel number decreasing.	Same as ERR28.
ERR30	Invalid INT mnemonic.	Same as ERR28.
ERR31	Invalid EQT number.	Same as ERR28.
ERR33	Invalid entry point.	Same as ERR28.
ERR34	Invalid absolute value.	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.

Table 15-2. DOS-III Bootstrap Error Halts

Halt Code	Cause	Recovery Action
102011	Disc error status is in the A register. If A register contains 0, the subchannel did not contain a system.	Check that the device is ready and the proper disc cartridge is being used; then call maintenance.
102031	Same as above.	Occurs during execution of disc-resident part of Bootstrap. Check that the disc is ready; then call maintenance.

Table 15-3. DOS-III Error Conditions

DOS-III ERROR HALTS

Halt Code	Location	Cause	Recovery Action
102002 102003	location 2_8 location 3_8	Possible memory wrap-around when memory protect is not present.	Program error. Bootstrap DOS-III from the disc and correct the program.
102004	DISCM	Power has gone up or down with powerfail option present.	Bootstrap DOS-III from disc and restart.
102011	$EX20	Disc parity error. Halt occurs after a message is printed giving location of error.	Unprotect the disc and start the computer executing. DOS-III assigns next spare track.
102031	DVR31	Trying to write on disc cylinder that is flagged "protected" without first unprotecting the disc.	Start the computer executing to exit DVR31 with no action taken.
102077	$EX20	Follows message telling operator to protect the disc after spare track assignment.	Protect the disc and start the computer executing. DOS-III aborts the job that was running.

DOS-III ERROR MESSAGES

During the operation of DOS-III certain messages may be output on the system console. These messages may be error reports or simply informative; they are generated by various parts of DOS-III. The messages are listed 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.

Message	Source	Description
BAD CONTROL STATE	JOBPR	Directive just entered is not acceptable in DOS III. Enter correct directive on system console.[1]
BEGIN 'DEBUG' OPERATION	DEBUG	Any legal DEBUG operations may now be entered. Enter any legal DEBUG operations.
BP BND [L,U]?	LOADR	Specify the base page bounds desired for the program being loaded by the Loader. The bounds should be entered as two octal constants separated by a comma.
CHECKSUM ERROR	JOBPR	Checksum error in input to ST,R,file or ST,X,file directive. Correct tape.[1]

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
CW nnnnn	DISCM	In an I/O READ/WRITE EXEC call at nnnnn, buffer extends beyond memory bounds. Correct program.
DEVICE #nn DOWN	JOBPR	EQT #nn is unavailable (down). Use the UP,nn directive to make the device available. (Then use the GO directive if needed.)
DICTIONARY OVERFLOW	JOBPR	No room is left for entries in the user file dictionary. Put file on another disc or remove some of the files.
??? DISC	DISCM	Informs user that disc is not recognizable by DOS-III. Must be labeled or unlabeled with :IN, or formatted with DSGEN, before using in DOS-III.
DISC GEN CODE nnnn NOT SYS GEN CODE mmmm ERR POSS	DISCM	Informs the user that the disc being requested was initialized (labeled) by a system with a different system generation code. Generation code on disc may be updated by labeling or unlabeling using :IN.
DISC NOT ON SYSTEM	DISCM	No disc pack with the currently requested label can be found on the system. Mount disc pack with correct label or ready drive containing disc.
DONE?	JOBPR	Thirty feed frames (paper tape) or an end-of-file (magnetic tape) have occurred during input. Enter YES for end of input; NO for more input.
??? LABEL xxxxxx DOS LABEL xxxxxx TSB LABEL xxxxxx OK TO PURGE?	DISCM	Attempting to label (or unlabel) an already labeled disc pack. Enter YES to relabel the disc pack or NO to drop the request to relabel the disc pack.
DUPLICATE FILE NAME	JOBPR	Doubly defined file name found in a STORE directive (other than STORE,P); an EDIT directive with a new file name; on DD,U; or on a RENAME directive. Remove file or rename file.[1]
$END ALGOL	ALGOL	End of ALGOL compilation. No response required.
$END ASMB	ASMB	Assembly has completed. No response required.
$END ASMB CS	ASMB	Assembly has ended because of an error in the assembler control statement. Correct the control statement.
$END ASMB NPRG	ASMB	Assembly has terminated because no JFILE was found when required. Define the file using a JFILE directive.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
$END ASMB PASS	ASMB	Another pass of the source program through the input device is required. Printed on the system console after Pass 1. Replace the program in the input device and type :GO.
$END ASMB XEND	ASMB	Assembly stops. An EOF occurred in the source program before an END statement. Add an END statement to the program.
END FILE	JOBPR	During an EDIT, (1) the master file ended before completion of editing or (2) a triple colon occurred in the first 3 columns of a source statement. Check input to the EDIT program.[1]
$END,FTN[4]	FTN[4]	Compilation has completed. No response required.
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 of the job are printed on the system console and standard list device if a Time-base Generator is present.
ENTER FILE NAME(S) OR /E	LOADR	Enter list of relocatable program files. To terminate list of file names type "/E".
ENTRY ERROR	DEBUG	DEBUG operation entered was illegal. Correct entry.
EQT xx CH xx DVRxx D R Ux Sx	JOBPR	Equipment table entry output by the EQ directive. No action required.
EXTRA PARAMETERS	JOBPR	More than 15 parameters in a directive. Excess parameters are not processed.
FI nnnnn	DISCM	In a FILE READ/WRITE EXEC call (1) the file requested at nnnnn cannot be found. If nnnnn is not present, enter the file. (2) the length of the buffer requested at nnnnn extends beyond the end of the file. Correct the buffer length. Either case causes calling program to abort.
HPAL??	ALGOL	Control statement error. Correct control statement.
IB nnnnn	DISCM	Illegal buffer address in EXEC call at location nnnnn. Program is aborted. Correct buffer program address.
IE nnnnn	DISCM	If a colon occurs in the first column of input entered through the batch device during a program execution, the program is aborted, control is given to the JOBPR and the input is processed as a directive. nnnnn is the memory location of the input request.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
IGNORED	DISCM	Input from system console during program execution cannot be processed. Correct input.
*IGNORED	JOBPR	All directives following EJOB and before next JOB except BATCH, TYPE, TRACKS, and OFF are ignored. Enter acceptable directive.
file ILLEGAL	JOBPR	On a source file LIST directive, the requested file was not a source file. Retype LIST directive using source file.[1] A file name begins with a non-alphabetic character. Rename the file.[1]
ILLEGAL DIGIT	JOBPR	In a decimal number, character is other than 0-9. Enter correct decimal number. In an octal number, digit is other than 0-7. Enter correct octal number.[1]
ILLEGAL LUN	JOBPR	Logical unit requested is equal to zero, greater than the number of logical units in the system, not the correct type (i.e., input type for output device), etc. Enter a correct logical unit.[1]
ILLEGAL PROGRAM RUN LIMITS	DISCM	Attempt to run a user main or segment whose user area limits or base page limits will not fit within the limits of the current system. Recreate user mains or segments on current system using LOADR.
ILLEGAL PROGRAM TYPE	JOBPR	Program requested in a RUN or PROG is not legal. Enter correct name.[1]
INPUT ERROR	DISCM	Equipment table entry number or logical unit number in :EQ, :LU, :UP or :DN is illegal. Enter correct equipment table or logical unit entry number.
INPUT :DATE, XXXXXXXXXX[,H,M]	DISCM	When system is initiated from the disc, DOS-III requires a DATE directive. The [,H,M] is ignored in DOS-III if a Time-base Generator is not in the system. Enter a DATE directive.
I/O ERR ET EQT #mm	DISCM	End-of-tape on device #mm. EQT #mm is unavailable. To make the device available (up), use the UP,mm directive.
I/O ERR NR EQT #mm	DISCM	The device #mm is not ready. To make the device available (up), use the UP,mm directive.
I/O ERR PE EQT mm	DISCM	Parity error on device #mm returns to program return address with A set to status, B set to 0. Call maintenance.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
I/O ERR {PE / NR} USER DISC	DISCM	A parity error or device not ready occurred when attempting to assign a user disc. Disc may not be formatted; format it with DSGEN.
I/O ERR {PE / NR} USER DISK	DISCM	Disc error in completion section of DVR31. Retry previous operation.
IT nnnnn	DISCM	Illegal disc track or sector address in EXEC call from location nnnnn. Program is aborted. Correct the track or sector address in EXEC call.
JBIN OVF	FTN [,4], ASMB, ALGOL	Overflow of Job Binary Area during assembly or compilation. Reduce size of job or purge user files.
JOB ABORTED!	JOBPR	Correct problem and start new job.
JOB xxxxx ddddddddd [TIME = xxxx MIN. xx.x SECS EXEC = xxxx MIN. xx.x SEC.]		
	JOBPR	Message output at the beginning of each job. The time information is deleted in DOS-III if a Time-base Generator is not included in the system. Start job.
L01	LOADR	Checksum error on tape.
L02	LOADR	Illegal record.
L03	LOADR	Memory overflow.
L04	LOADR	Base page overflow.
L05	LOADR	Symbol table overflow.
L06	LOADR	Duplicate main or segment name (may be caused by attempting to run the Loader twice in one job).
L07	LOADR	Duplicate entry point.
L08	LOADR	No main or segment transfer address.
L09	LOADR	Record out of sequence.
L10	LOADR	Insufficient directory work area or user area space.
L11	LOADR	Program table overflow.
L12	LOADR	User file specified cannot be found.
L13	LOADR	Program name duplication.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
L14	LOADR	Non-zero base page length.
L15	LOADR	Segment occurred before main.
L16	LOADR	Program overlay (illegal ORG).
L17	LOADR	Illegal library record.
L18	LOADR	Illegal octal digit in base page bounds specification; or the lower base page bound is greater than the upper base page bound; or the lower or upper base page bound is greater than 2000_8. In keyboard mode, re-enter new base page bounds. In batch mode, Loader aborts.
L19	LOADR	Illegal octal digit in main memory bounds specification; or the lower program bound is greater than the upper program bound. In keyboard mode, re-enter new program bounds. In batch mode, Loader aborts.
LBL = 111111	DISCM	Disc subchannel referenced is labeled 111111. If attempting to change user disc subchannel, enter :UD with correct label.
LIMIT ERROR	JOBPR	In a directive, source statement numbers are out of order (:EDIT), dump limits are incompatible (:PDUMP,:ADUMP), sector numbers are illegal (:DUMP), or beginning source statement number is greater than final statement number (:EDIT). Correct directive and re-enter.[1]
xxxx LINES	JOBPR	Total number of statements stored by a STORE,S directive. No response required.
****LIST END****	JOBPR	Terminates list of source statements generated by a LIST directive. No response required.
LN nnnn	DISCM	Logical unit requested by an EXEC call at nnnnn is unassigned. Program is aborted. Assign logical unit.
LOADR COMPLETE	LOADR	Loading has completed. No responses required.
LOADR SUSP	LOADR	Loader has suspended (usually at EOT). Type :GO,n to restart the Loader with proper parameter value.
LOADR TERMINATED	LOADR	Loader has terminated because of an error. Correct input.
LOAD TAPE	LOADR	In conjunction with LOADR SUSP, this message requests that next relocatable tape be loaded before :GO. Load the next relocatable tape and enter :GO to read next tape or :GO,1 to indicate that all tapes are read in.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
LU nnnnn	DISCM	Illegal logical unit in EXEC call at nnnnn. Program is aborted. Enter correct logical unit number.
LUxx EQTyy	JOBPR	Logical unit table entry; EQT #yy assigned to LU #xx. No response required.
LUN UNASSIGNED	JOBPR	Logical unit requested in a directive is unassigned. Assign logical unit number requested in the directive.[1]
xxxxx MISSING	DISCM	Segment xxxxx requested by an EXEC call is not in system or user directory. Job is aborted. Correct job.
MISSING PARAMETER	JOBPR	A parameter is missing in a directive. Retype the directive correctly.[1]
MP nnnnn	DISCM	Memory protect violation at location nnnnn. Program is aborted. Correct the program.
NAME *IGNORED	JOBPR	Illegal JOB name; numeric first character. Retype correct job name.
NEXT AVAIL TRACK=tt BAD=n	JOBPR	In TRACK directive, tt = first track beyond end of current user area; n = number of bad tracks. "BAD=n" returned only if bad tracks do exist. tt = "NONE" if no tracks are available.
NO BIN END	JOBPR	No END record detected when storing a relocatable binary program.[1]
NO PROGRAMS LOADED	LOADR	No programs were loaded by the Loader. Loading terminates.
NO SOURCE	JOBPR	No source statements following a /R or /I in an EDIT directive. Enter source statements after the /R or /I.[1]
NO SOURCE	ALGOL	Source file from disc not pre-set.
NUMBER OVERFLO	JOBPR	An integer is too large.[1]
OR nnnnn	DISCM	I/O operation requested by EXEC call at nnnnn is rejected. Program is aborted. Check program.
OVERFLOW JBIN	JOBPR	There is not enough room in the JBIN for storing the relocatable binary output from the Assembler or compilers.[1]
PARAMETER ILLEGAL	JOBPR	A parameter of a directive is illegal. Re-enter directive.[1]
PARITY ERROR SC=m,TRK=ttt,SCTR=sss	JOBPR	Parity error during disc read or write. Call maintenance.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
PAUSE xxxx	LIBR (Formatter)	Program has temporarily suspended itself. xxxx is an octal number acting as an identifier. Restart program using the GO directive.
PROG BND [L,U]?	LOADR	Enter the program bounds for the program being loaded by the Loader. The bounds consist of two octal numbers separated by a comma.
RE-ENTER STATEMENT ON TTY	JOBPR	Follows most error messages that do not cause abort. Type in the correct statement.
RQ nnnnn	DISCM	Illegal request code in EXEC call at nnnnn. Program is aborted. Correct the program.
SPARE TRK OVERFLOW	JOBPR	Defective cylinder detected and no spare tracks available for reassignment.
STOP xxxxx: nnnnn	LIBR	Program xxxxx has terminated at location nnnnn.
SUBCHAN = n	DISCM/ JOBPR	Given in response to :UD information request or when :SS makes new subchannel assignment. No response required.
xxxxx SUSP	DISCM	Program xxxxx suspended by EXEC call or PAUSE directive. Restart program using the GO directive.
TAPE END	JOBPR	EOT flag set on magnetic tape or paper tape device during output via JOBPR directives DUMP and LIST or output of a JOB or EJOB statement. If a magnetic tape, it is rewound with standby; if paper tape, a trailer is punched. The JOBPR will then pause to allow new tape to be set up. Mount a new magnetic tape. Enter :GO to continue the output.
TM nnnnn	DISCM	Maximum execution time exceeded. The program is currently at nnnnn and is aborted. Increase execution time.
#TRACKS UNAVAILABLE	DISCM	There are not enough word tracks for the compiler. Enter :OFF then purge disc of unnecessary files.
TRAC #TOO BIG	JOBPR	Track requested is higher than last available disc track (track may be in JBIN area). Redefine the track request or purge files or use different disc.[1]
TSB DISC	DISCM	Informs user that the user disc was labeled by a non-DOS-III system. May be made DOS-III disc by labeling or unlabeling with :IN.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

Table 15-3. DOS-III Error Conditions (continued)

Message	Source	Description
TURN {ON / OFF} DISC PROTECT OVERRIDE SWITCH	DISCM	Unprotect [ON] or protect [OFF] the disc.
UD nnnnn	DISCM	Unable to find user disc requested by EXEC call at nnnnn. Mount required disc and type :GO; or terminate program with :ABORT or :OFF.
UNLBL	DISCM	User disc specified in :UD is unlabeled. If trying to change user disc assignment, enter :UD,*[,n].
file name UNDEFINED	JOBPR	Undefined file name as a parameter of a directive. Retype correct file name on the system console.[1]
UNDEFINED EXTS	LOADR	Undefined external references exist in programs loaded. The external references are listed one per line. To load additional programs from paper tape, type :GO,0[,n].
WRONG INPUT	JOBPR	Relocatable binary input furnished for a source file request or vice-versa. Enter correct input.[1]
name: nn xx	ERR0	Library routine error code, where name is the name of the user's program, nn is the routine identifier and xx is the error type.
@	JOBPR/ DISCM	Directives may be entered. Enter desired directive.
*	DISCM	Operator attention directives may be entered. Enter desired directive.

[1] This error causes a batch abort if the command is entered in batch mode. See "Batch Abort" in Section 1.

DOS-III EFMP ERROR CODES

These error numbers are returned to the user program (in ERRNO) by the EFMP. The error numbers are also returned in the A register.

Error No.	Description
0	No errors.
1	Invalid EFMP function number.
2	Duplicate file name.
3	File name not in directory.
4	File too long for this pack.
5	Invalid record length.
6	Pack number not available (or name not in directory if a search was made on all available pack directories).
7	Invalid security code.
8	A temporary file must be opened with a CREATE function. An OPEN function can only change the Temporary Record Buffer number of the starting record number for a temporary file.
9	Buffer area specified in Exec call is not valid.
10	Invalid Record Number.
11	File not open.
12	DEFINE not previously executed or Opened-File table used in previous DEFINE has been altered. Issue a new DEFINE.
13	Backspaced beyond "start-of-file."
14	No pack space available.
15	Invalid pack number.
16	No pack number entry is available in Opened-File table.
17	Work Area space not sufficient.
18	No Opened-File table space available.
19	Invalid temporary record buffer number.
20	Invalid number of EXEC call parameters.

Error No.	Description
21	End-of-File.
22	COPY terminated.
23	Invalid argument(s).
24	Maximum number of files exceeded.
25	File already OPEN.
26	Record size larger than one-half of a temporary record buffer.
27	Pack number previously initialized.
28	Pack number not initialized.
29	Directory requested is too large.

APPENDIX A
System Tables

This appendix contains figures and tables which represent the structure of the following

- Main-memory layout, including
  - main memory allocations in DOS-III
  - DOS-III base page constants
  - DOS-III base page communication area
- Disc layout, including
  - disc structure in DOS-III
  - disc directory entry format
  - disc labels
- System I/O tables, including
  - the equipment table
  - the logical unit table
  - the interrupt table

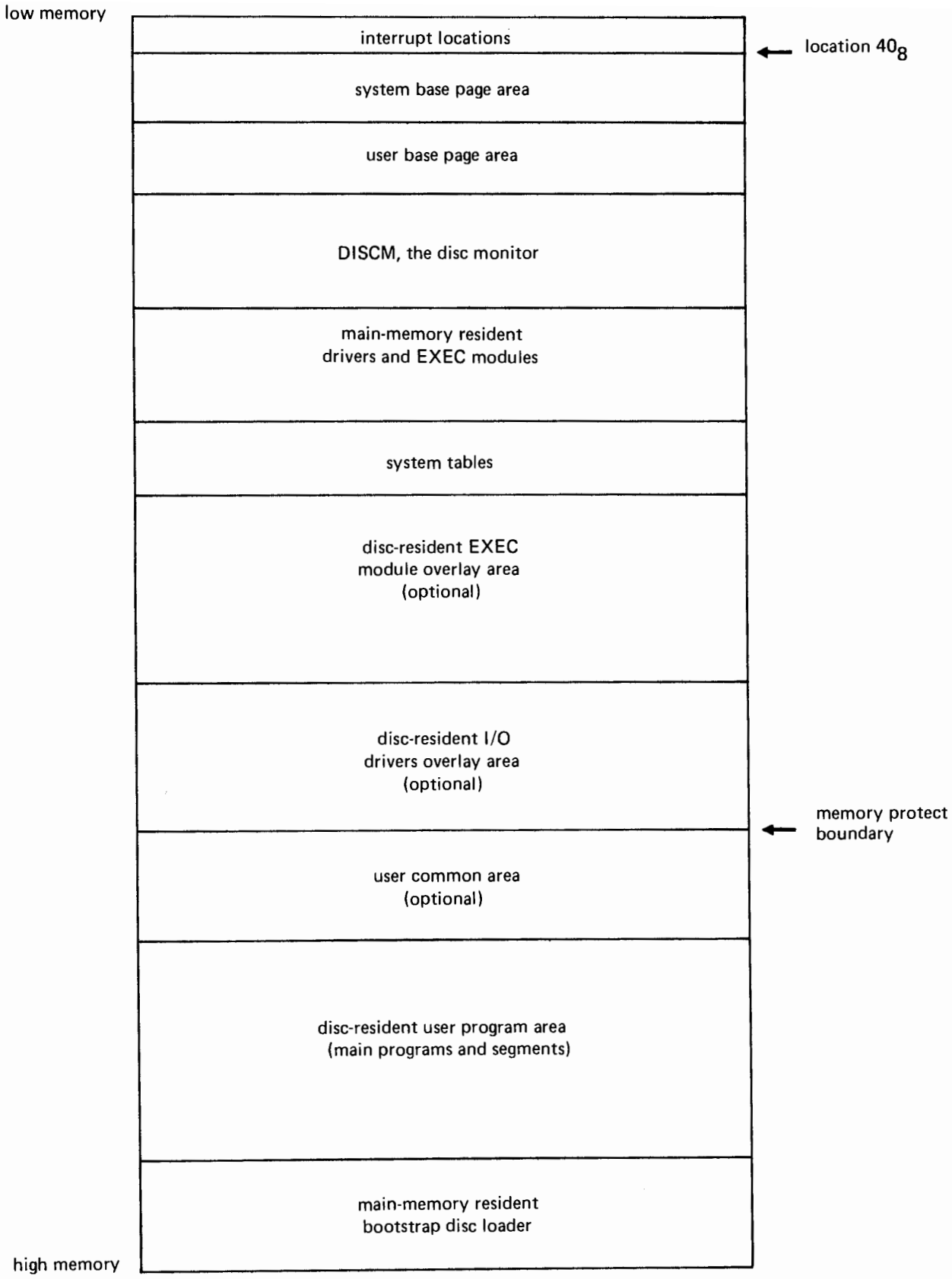

Figure A-1. Main Memory Allocations in DOS-III

Table A-1. DOS-III Base Page Constants

Location	Type	Value
40	DEC	-64
41	DEC	-10
42	DEC	-9
43	DEC	-8
44	DEC	-7
45	DEC	-6
46	DEC	-5
47	DEC	-4
50	DEC	-3
51	DEC	-2
52	DEC	-1
53	DEC	0
54	DEC	1
55	DEC	2
56	DEC	3
57	DEC	4
60	DEC	5
61	DEC	6
62	DEC	7
63	DEC	8
64	DEC	9
65	DEC	10
66	DEC	17
67	DEC	64
70	OCT	17
71	OCT	37
72	OCT	77
73	OCT	177
74	OCT	377
75	OCT	177400
76	OCT	3777
77	OCT	177700

Table A-2. DOS-III Base Page Communication Area

Location	Name	Contents
100	UMLWA	Last word address of user available memory
101	JBINS	Start track/sector of Job Binary Area
102	JBINC	Current track/sector of Job Binary Area
103	TBG	Time-base Generator I/O channel address
104-5	CLOCK	Current system clock time (2 words)
106-7	CLEX	Execution clock time (2 words)
110	CXMX	Maximum allowable execution time
111	BATCH	Logical unit # of batch input device
112	SYSTY	Logical unit # of system console
113	DUMPS	Abort/Post Mortem dump flag
114	SYSDR	System directory track/sector
115	SYSBF	System buffer track/sector
116	SECTR	Number of sectors/disc track
117	EQTAB	First word address of equipment table
120	EQT#	Number of equipment entries
121	LUTAB	First word address of logical unit table
122	LUT#	Number of logical unit entries
123	JBUF	Job input buffer address
124	JFILS	Source file starting track/sector
125	JFILC	Source file current track/sector
126-32	RONBF	Parameter buffer (5 words)
133	MDFLG	Mode flag for privileged I/O
134	DISP	(Reserved for System use)
135	AEPF	Alternate entry point flag
136	SGRTN	Segment return address
137	XIRT	System transfer address for interrupt-completion routine
140	SVEQT	EQT address for I/O operations
141-53	EXPG	Directory entry for current program (11 words)
154	DISCO	Disc I/O channel/last track on disc
155	SYSSC	System subchannel

Table A-2. DOS-III Base Page Communication Area (continued)

Location	Name	Contents
156	SCCNT	Number of subchannels on system minus 1
157	UDNTS	Next user disc track/sector
160	SYNTS	Next system disc track/sector
161	CUDSC	Current user disc subchannel
162	CRFLG	Current disc request flag: 0 for system, non-0 for user
163	CUDLA	Current user disc last access
164	FSFLG	File search flag
165	CUMID	Computer identification
166-70	DBUFR	System disc triplet parameter buffer (3 words)
171-73	UBUFR	User disc triplet parameter buffer (3 words)
174	TSONE	Last track/sector referenced +1
175	GUDSC	Default user disc subchannel
176	SYSCD	System generation code
177	JFLSC	Source file subchannel
200	DISCL	User label track/sector
201	INTAB	First word address of interrupt table
202	INT#	Number of interrupt entries
203	EQT1	EQT1-EQT17 are addresses of current equipment table entry
204	EQT2	
205	EQT3	
206	EQT4	
207	EQT5	
210	EQT6	
211	EQT7	
212	EQT8	
213	EQT9	
214	EQT10	
215	EQT11	
216	EQT12	
217	EQT13	

Table A-2. DOS-III Base Page Communication Area (continued)

Location	Name	Contents
220	EQT14	EQT1-EQT17 are addresses of current equipment table entry
221	EQT15	
222	EQT16	
223	EQT17	
224	RQCNT	Number of request parameters
225	RQRTN	Current request return address
226	RQP1	RQP1-RQP8 are addresses of current request parameters
227	RQP2	
230	RQP3	
231	RQP4	
232	RQP5	
233	RQP6	
234	RQP7	
235	RQP8	
236	NABRT	Illegal request code abort/no abort option
237	XA	A register contents at time of interrupt
240	XB	B register contents at time of interrupt
241	XEO	E and O register contents at time of interrupt
242	XSUSP	Point of suspension at time of interrupt
243	EXLOC	Address of Exec module doublet table
244	EX#	Number of Exec module doublet table entries
245	EXMOD	Exec module # currently in Exec module overlay area
246-47	EXMAN	Exec module low and high main memory addresses (2 words)
250-51	EXBAS	Exec module low and high base page memory addresses (2 words)
252	IODMN	First word address of I/O driver module main area
253	IODBS	First word address of I/O driver module base page area
254	UMFWA	First word address of user main area
255	UBFWA	First word address of user base page area
256	UBLWA	Last word address of user base page area

Table A-2. DOS-III Base Page Communication Area (continued)

Location	Name	Contents
257	CHAN	Current DMA channel number
260	OPATN	Operator/keyboard attention flag
261	OPFLG	Operator communication flag
262	SWAP	Job processor resident flag
263-64	JOBPM	Job processor disc address/number of words in main (2 words)
265	JOBPB	Job processor base page number of words
266	EJOBF	End-job flag
267	RTRK	Real time simulation track number
270	DUMMY	Reserved for system use
271	MPTFL	Memory protect flag
272	$GOPT	Point of suspension continuation address
273	$IDCD	Input request code check
274-75	$MDBF	Exec module data buffer (2 words)
276-304	TEMP	Reserved for data communications (7 word buffer)
305	TEMP0	Reserved for System use
306	TEMP1	
307	TEMP2	
310	UTMP0	
311	UTMP1	
312	UTMP2	
313	MSECT	Negative number of sectors/track
314	VADR	Address of instruction causing memory protect violation
315	IODMD	Current resident I/O driver module flag
316	RCODE	Current request code value
317	SXA	Operator attention restore A register value
320	SXB	Operator attention restore B register value
321	SXEO	Operator attention E and O register value
322	SXSUS	Operator attention return address
323	EFMP	Extended File Management Package flag

Table A-2. DOS-III Base Page Communication Area (continued)

Location	Name	Contents
324	DSCLB	Disc track/sector of Relocatable Library
325	DSCL#	Number of Relocatable Library routines
326	LSTCH	Last disc referenced
327	TRAC#	User file table validity flag
330	XFLG	Entry address for disc not ready
331	SSFLG	System search flag
332	CHARC	Batch input character count
333	TYEQT	System console EQT4 address

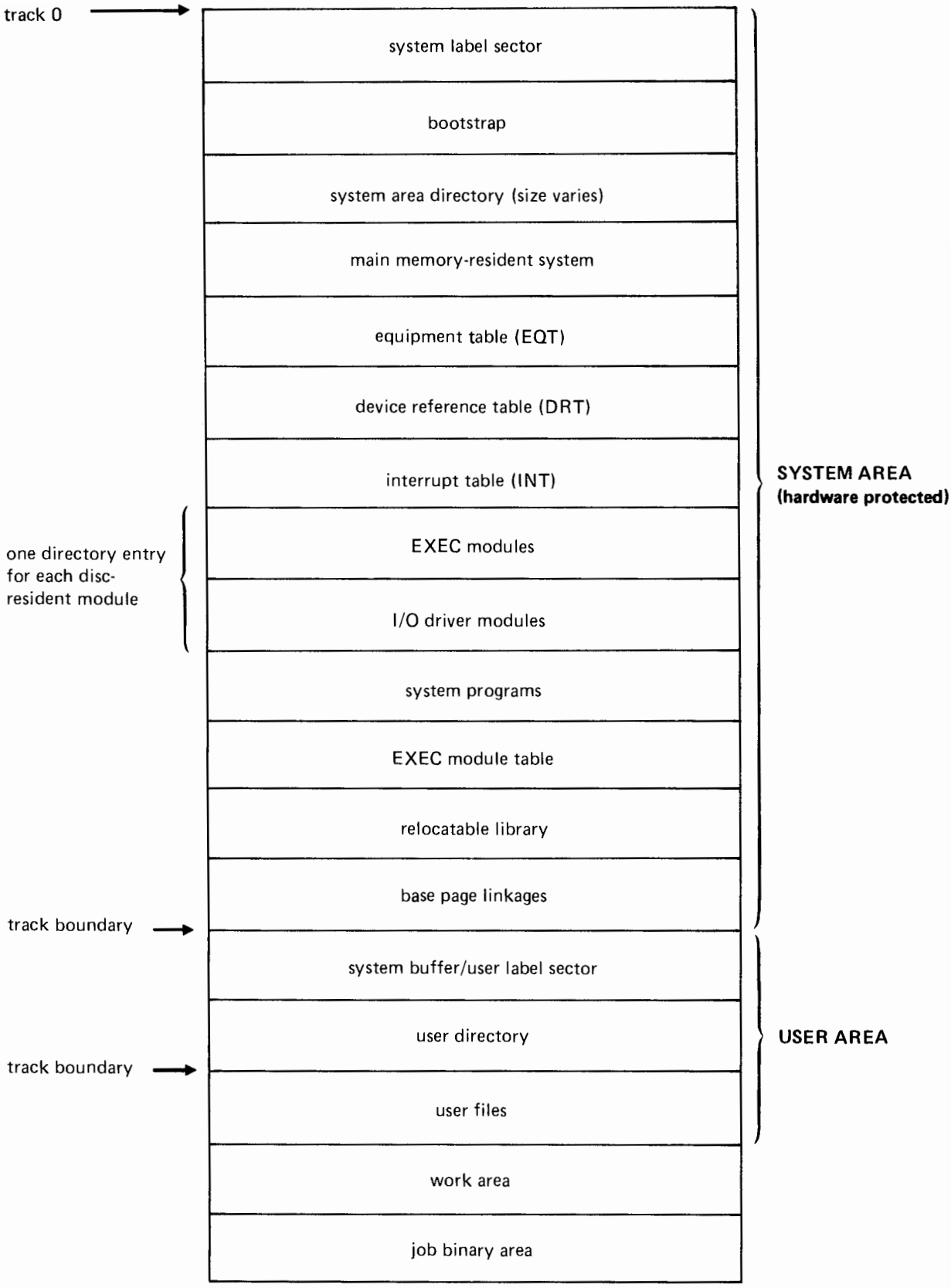

Figure A-2. Disc Structure in DOS-III

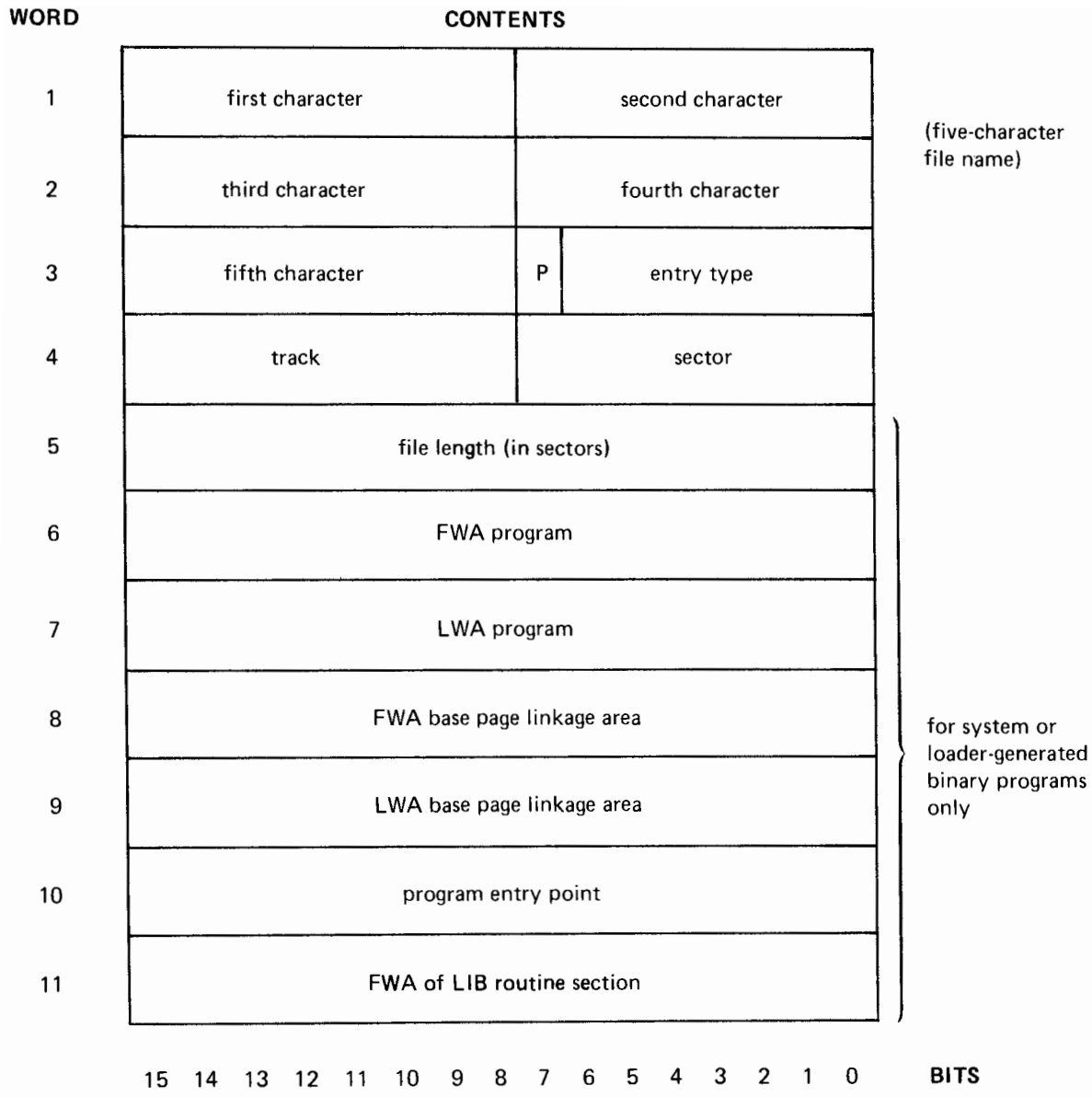

Figure A-3. Disc Directory Entry Format

'P' Bit

0 = Permanent file—no action is taken at end-of-job.

1 = Temporary file—purge this entry at end-of-job.

This bit is set by the Relocating Loader and cleared by a STORE,P directive.

Entry Type

Type	File
0	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
14_8	Absolute binary

Note: 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 (0).

DISC LABELS

Sector 0 of track 0 of each disc is used for label information. In addition, if the user area is on the system disc, a label also exists in Sector 0 of the first track after the system area. The first 64 words (words 0-63) are reserved for label information. Word 64 contains the next available track and sector. Words 65 and 66 contain the number of bad tracks and the next available spare track.

The contents of the label include:

Word 0: Label presence code (ASCII "LB" for labeled, zero for unlabeled)

Word 1: System proprietary code:

1. "DO" for DOS-III

2. "TS" for Time-shared BASIC

Word 2: System generation code assigned at system generation time. The code can be any four decimal digits.

Words 3-5: A six-character disc label. If the first character equals * the disc is unlabeled. This label can only be set using :IN (for user areas) or by DSGEN (set to "SYSTEM" for system discs).

Word 31: Checksum of words 0-30.

THE EQUIPMENT TABLE

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

D	= 1 if DMA channel required.	
R	= 1 if driver type is main-memory resident.	
Unit #	May be used for subchannel addressing.	
Channel #	I/O select code for device (lower number if multiboard interface).	
Av	= 0	Unit not busy and available
	= 1	Unit disabled (down)
	= 2	Unit busy

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

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

00-07	Paper Tape Devices
00	Teleprinter
01	Punched Tape Reader
02	High Speed Punch
05	System Console
10-17	Unit Record Devices
10	Plotter
11	Card Reader
12	Line Printer
15	Mark Sense Card Reader
20-37	Magnetic Tape/Mass Storage and other devices capable of both input and output
23	7970 Magnetic Tape
26	2762A Terminal Printer
31	Moving-Head Disc
33	Writable Control Store

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

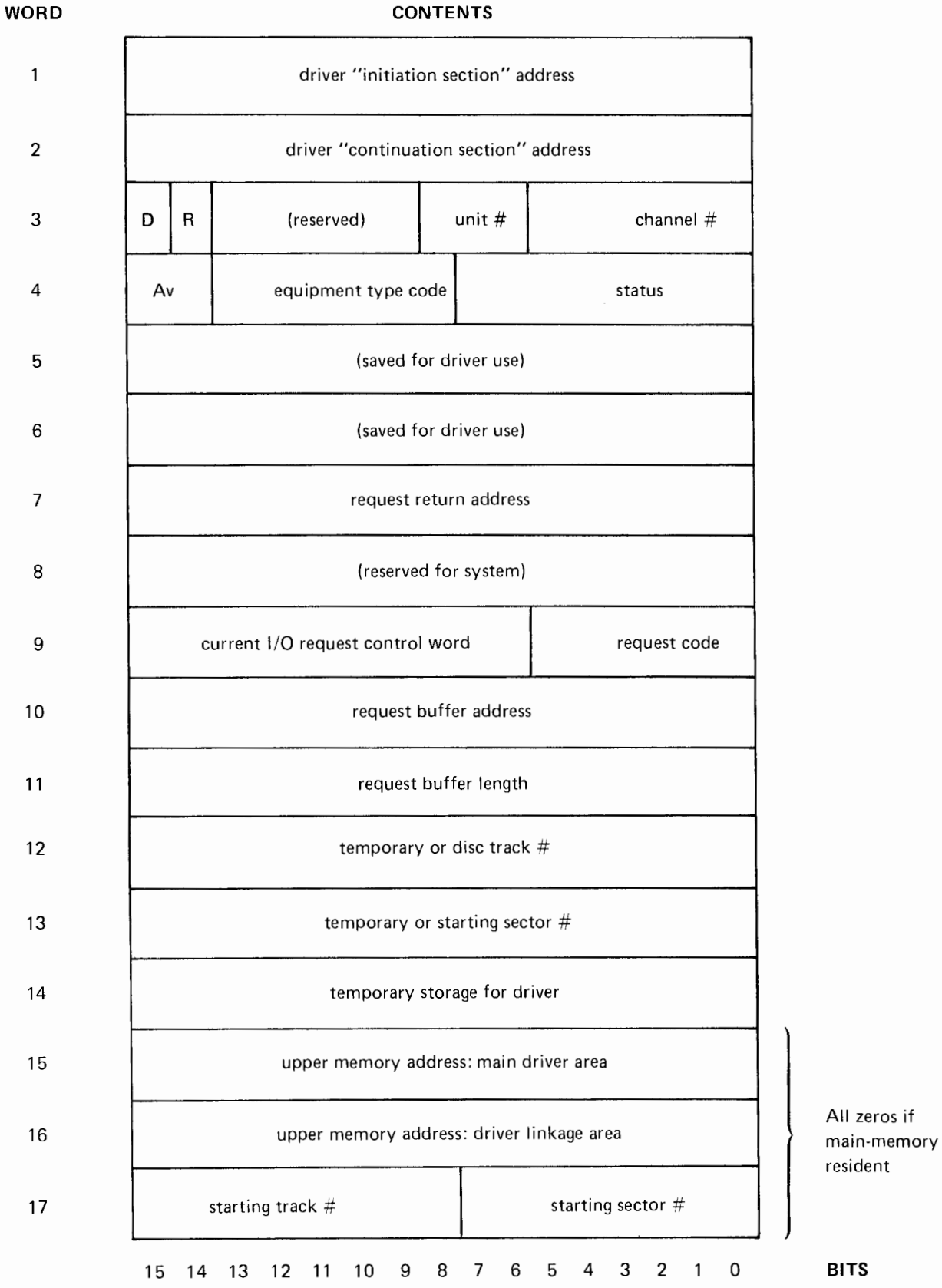

Figure A-4. The Equipment Table

THE LOGICAL UNIT TABLE

The logical unit table (LUT) has an entry for each logical unit defined at system generation time (maximum number is 63). These entries provide logical addressing of the physical devices defined in the EQT. Logical unit numbers 4-63 may be modified within a job by using the LU directive. At end-of-job, logical unit number 1-9 are restored to their original system generation values. The LUT entries reside in the permanent main-memory resident part of the system and have the following format:

Word	Contents
1	Device EQT number
.	
.	
.	
n	Device EQT number

THE INTERRUPT TABLE

The interrupt table (INT) contains an entry, established at system generation time, for each I/O channel which can cause an interrupt (beginning with I/O channel 6). The INT entries reside in the main-memory resident portion of the system and have the following format:

Word	Contents
1	Address of device's EQT entry
.	
.	
.	
n	Address of device's EQT entry

INDEX 1
Summary of Directives

Directive	Description	Page
:EJOB	Terminate the current batch and/or job normally	2-22
:EQ[,n]	List the complete equipment table, or just one line	2-23
:GO[,P_1,P_2, . . . ,P_5]	Continue processing a suspended program	2-25
:IN,label	Label or unlabel ("*") the current user disc	2-26
:JFILE,file	Specify a source file on the disc for the Assembler or a compiler	2-43
:JOB[,name]	Initiate a user job	2-28
:LIST,S,logical unit,file[,m[,n]]	List all or part of a source statement file	2-29
:LIST,U,logical unit[,$file_1$, . . .]	List all or part of the user directory	2-29
:LIST,X,logical unit[,$file_1$, . . .]	List all or part of the system directory	2-29
:LU[,n_1[,n_2]]	Assign or list logical unit assignments	2-33
2		
:OFF	Abort the currently executing program or operation without terminating the job	2-35
:PAUSE [comment string]	Suspend the current job or program (optionally, output a comment on the system console)	2-36
:PDUMP[,FWA[,LWA]] [,B] [,L]	Dump a program after normal completion	2-13
:PROG,name[,P_1,P_2, . . . ,P_5]	Turn on a system or user program	2-37
:PURGE[,$file_1$,$file_2$ . . .]	Delete all temporary file or specified user files	2-38
:RNAME,oldname,newname[,type]	Rename a specified user file and optionally, change its program type	2-40
:RWND[,logical unit]	Rewind a magnetic tape	2-41
:RUN,name[,time] [,N]	Run a user program	2-42
:SA,track,sector[,number]	Dump disc in ASCII to standard list device	2-15
:SO,track,sector[,number]	Dump disc in octal to standard list device	2-15
:SS	Set up system search for file names over all subchannels	2-49
:SS,n_1,n_2 . . .	Set up system search for file names over specified subchannels	2-49

INDEX 2

Summary of EXEC Calls

Consult Section III for the complete details on each EXEC call.

RCODE	Name	Function	Page
-19	BASE PAGE STORE	Store values into base page memory locations (Value to be stored in the A register, absolute location address in the B register)	3-6
1, 2	I/O READ/WRITE	Transfer input or output (1 = read or 2 = write)	3-14
3	I/O CONTROL	Carry out control operations	3-11
6	PROGRAM COMPLETION	Signal end of program	3-19
7	PROGRAM SUSPENSION	Suspend calling program	3-22
8	SEGMENT LOAD	Load segment of calling program	3-24
10	PROGRAM LOAD	Transfer a main program into main memory	3-20
11	TIME REQUEST	Request the time-of-day	3-27
13	I/O STATUS	Request device status	3-17
14, 15	FILE READ/WRITE	Read or write a user data file (14 = read or 15 = write)	3-9
16	WORK AREA STATUS	Ascertain if *n* contiguous work tracks are available	3-30
17	WORK AREA LIMITS	Ascertain first and last tracks of work area	3-28
18	FILE NAME SEARCH	Ascertain if a file name exists in the directory	3-7

RCODE	Name	Function	Page
23	USER DISC CHANGE	Change the current user disc subchannel	3-32
24	EFMP CALLS	Execute EFMP functions	Section VII
27, 28	USER EXEC CALLS	Execute user EXEC modules $EX36 or $EX37 (RCODE = 27 for $EX36; RCODE = 28 for $EX37; up to five words of parameter information)	Section XII
29	SEGMENT RETURN	Return from a segment to the main program at the instruction immediately following the segment load call	3-26
30	MEMORY PROTECT CONTROL	Control memory protect from a user program	3-18

INDEX 3
Index of Terms

02100-90136