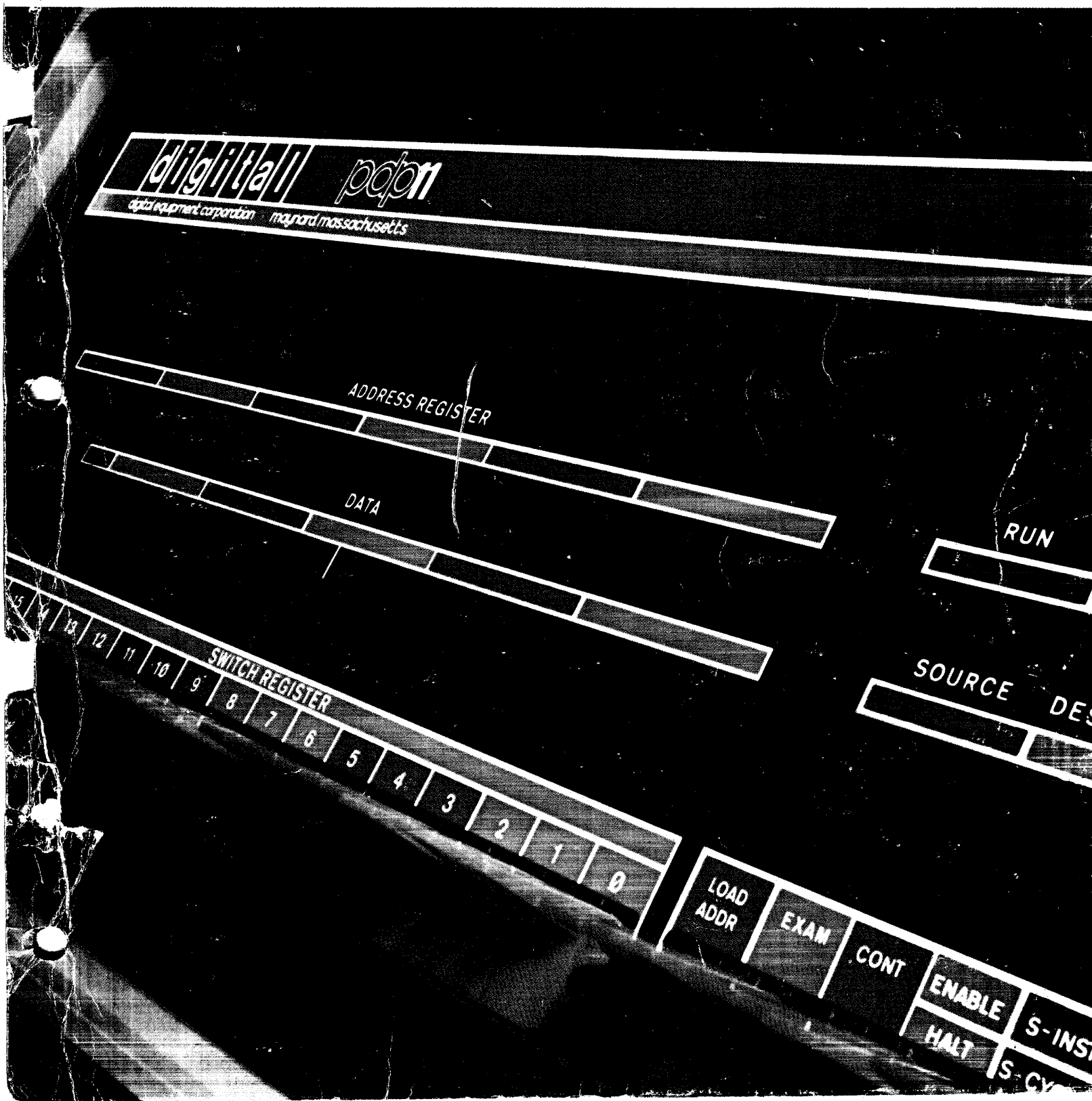
digital

pdp11

disk operating system monitor

programmer's handbook



PDP-11
Disk Operating System Monitor
Programmer's Handbook

SOFTWARE SUPPORT CATEGORY

The software described in this document is supported by DEC under Category I, as defined on page iv of this document.

For additional copies, order No. DEC-11-MWDC-D from Direct Mail
Bldg. 1-1, Digital Equipment Corporation, Maynard, Mass. 01754

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

First Printing, May 1971
Revised, August 1971
Revised, February 1972

Your attention is invited to the last two pages of this document. The "How to Obtain Software Information" page tells you how to keep up-to-date with DEC's software. The "Reader's Comments" page, when filled in and mailed, is beneficial to both you and DEC; all comments received are acknowledged and are considered when documenting subsequent documents.

This document is for information purposes and is subject to change without notice

Associated Documents:

PDP-11 FORTRAN IV
Programmer's Manual, DEC-11-KFDA-D

PDP-11 PAL-11R Assembler,
Programmer's Manual, DEC-11-ASDC-D

PDP-11 Edit-11 Text Editor,
Programmer's Manual, DEC-11-EEDA-D

PDP-11 ODT-11R Debugging Program,
Programmers Manual, DEC-11-OODA-D

PDP-11 Link-11 Linker and Libr-11 Librarian
Programmer's Manual, DEC-11-ZLDB-D

PDP-11 PIP, File Utility Package,
Programmer's Manual, DEC-11-PIDB-D

PREFACE

This document contains a comprehensive description of the PDP-11 Disk Operating System Monitor. The document is written for the PDP-11 programmer -- it assumes familiarity with the contents of the PDP-11 Handbook 1971 and the PAL-11R Assembler (see document number DEC-11-ASDB-D). Previous experience with monitor or executive systems would be helpful.

The document is separated into three chapters: Chapter 1 is an introduction to the DOS Monitor, and provides general information about the disk operating system. Chapter 2 describes the programmed requests that are available to the programmer through the Monitor. This chapter also explains the concepts and operation of each programmed request. Chapter 3 describes the keyboard commands available to the system operator through the Monitor; concepts and operation of each command are also explained. The entire document is summarized in the appendices. Appendices D (Monitor Commands) and E (Monitor Programmed Requests) should prove to be invaluable to the DOS user.

In addition to the DOS Monitor, the PDP-11 Disk Operating System software includes:

FORTRAN IV
PAL-11R Assembler
Edit-11 Text Editor
ODT-11R Debugging Program
PIP, File Utility Package
Link-11 Linker
Libr-11 Librarian

APPENDICES

APPENDICES (Cont)

APPENDICES (Cont)

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

1.1 THE DOS MONITOR

The PDP-11 Disk Operating System (DOS) Monitor supports the PDP-11 user throughout the development and execution of his program by:

- providing convenient access to system programs and utilities such as the FORTRAN, the DOS assembler, debugger, editor, file utility package, etc.;
- performing input/output transfers on three different levels, ranging from direct access of device drivers to full formatting capabilities;
- handling secondary storage management with two different kinds of file structure.

System programs and utilities can be called into core from disk or DECtape with Monitor commands issued from the keyboard. This feature eliminates the need to manipulate numerous paper tapes, and provides the user with an efficient and convenient programming tool.

All input/output (I/O) transfers are handled by the Monitor in any of three user-selected levels called READ/WRITE, BLOCK, and TRAN. READ/WRITE is a file-structured, formatted level of I/O in which the user can specify any one of nine modes. BLOCK is a file-structured, random access I/O level with no formatting. TRAN does basic I/O operations at the device driver level. All I/O is concurrent and interrupt driven.

The file system on secondary storage uses two types of file structures: linked and contiguous. Linked files can grow serially and have no logical limit on their size. Contiguous files must have their length specified but can be randomly accessed by BLOCK level I/O requests. Files can be deleted or created at any time, and are referred to by name. Table 1-1 summarizes the features and benefits of the DOS Monitor.

The user communicates with the Monitor in two ways: through programmed instructions called requests, and through keyboard instructions called commands.

Programmed requests are macros which are assembled into the user's program and through which the user specifies the operation to be performed. Some programmed requests are used to access input/output transfer facilities, and to specify where the data is, where it is going, and what format it is in. In these cases the Monitor will take care of bringing drivers in from disk, performing the data transfer, and notifying the user of the status of the transfer. Other requests access Monitor facilities to query system variables such as time of day, date, and system status, and to specify special functions for devices.

Keyboard commands enable the operator to load and run programs, load or dump data to or from core, start or restart programs at specific addresses, modify the contents of memory registers, and retrieve system information such as time of day, date, and system status.

Programs supported by DOS, and hence accessible through the Monitor, are listed in Table 1-2.

Table 1-1
PDP-11 DOS Monitor Features and Benefits

Feature	Benefits to User
Files are catalogued in multilevel file directories.	No file naming conflicts among users.
Files are referred to by name.	Files do not have to be remembered by number.
Files can grow serially.	Files can be created even when their final size is not known.
Files can be as large as the storage device can accept.	No logical limit on the size of files.
File storage is allocated dynamically from any bulk-storage device.	Files can be deleted or created even at run time for greater storage efficiency.
Monitor subroutines can be swapped into core when needed. Routines need not permanently tie up an area of core.	Much more efficient use of core space for user programs. Free core expands and contracts as Monitor subroutines are used. Space can be reclaimed for user programs. The user can determine which Monitor subroutines will be in core, and when.
Monitor subroutines can be made permanently core resident either before or during run time.	The user can tailor the Monitor for his particular needs.
The Monitor is divided into logical modules.	The user can easily and efficiently use the logical pieces of the Monitor for his own needs. He can also easily add his own specialized drivers to the system by following a simple set of rules, and still use the rest of the Monitor with these drivers.

Table 1-1 (Cont)
PDP-11 DOS Monitor Features and Benefits

Feature	Benefits to User
All I/O is interrupt driven.	Such specialized equipment as communications modems and A/D converters which must be interrupt driven can be run under the Monitor. Several I/O calls can be handled concurrently.
Device independence	Specific devices can be specified by the user in his program, and any device can be substituted by him when his program is being run.
Devices are assigned to one or more datasets.	The user may reassign a device which is used for one purpose (dataset) without changing its assignment for all other purposes (datasets).

Table 1-2
The DOS System Programs

System Program	Document Number
FORTRAN IV	DEC-11-KFDA-D
PAL-11R Assembler	DEC-11-ASDB-D
Edit-11 Text Editor	DEC-11-EEDA-D
ODT-11R Debugging Program	DEC-11-OODA-D
PIP, File Utility Package	DEC-11-PIDA-D
Link-11 Linker and Libr-11 Librarian	DEC-11-ZLDA-D

1.2 MONITOR CORE ORGANIZATION

Core memory is divided into:

- a user area where user programs are located;
- the stack where parameters are stored temporarily during the transfer of control between routines;
- the free core or buffer area which is divided into 16-word blocks assigned by the Monitor for temporary tables, for device drivers called in from disk, and for data-buffering between devices and user programs;
- the resident Monitor itself which includes all permanently resident routines and tables;
- the interrupt vectors.

Figure 1-1 is a map of core as organized by the Monitor.

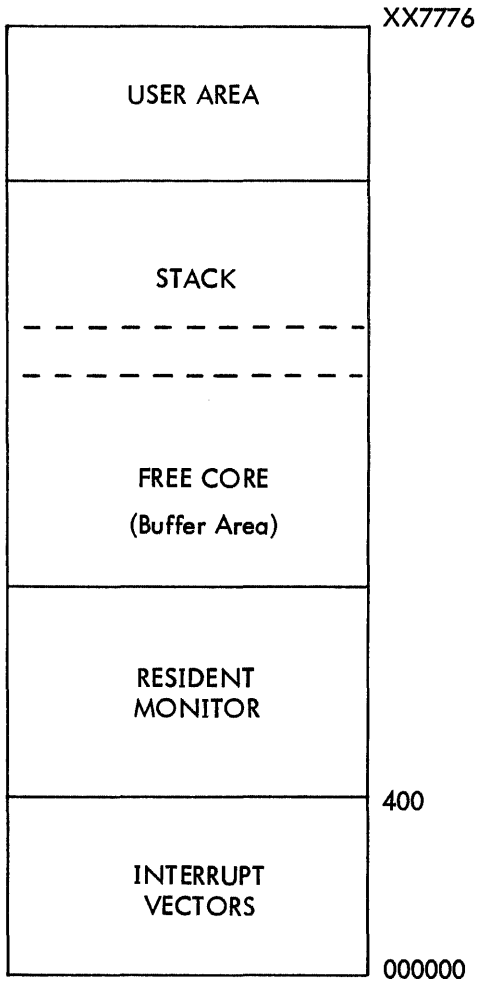


Figure 1-1 The Monitor Core Map

1.3 HARDWARE CONFIGURATIONS

Two possible minimum configurations required to run the PDP-11 DOS Monitor are:

Configuration A:	1	PDP-11/20 with 8K of core
	1	ASR-33 Teletype terminal
	1	RC11 disk controller
	1	RS64, 64K fixed head disk drive
	1	TC11 DECtape controller
	1	TU56 dual DECtape transport
	1	BM792-YB Bootstrap Loader

Configuration B:

1 PDP-11/20 with 8K of core
1 KSR-33 Teletype
1 RF11 disk controller
1 RS11, 256K fixed head disk drive
1 PC11 high-speed paper tape reader/punch
1 BM792-YB Bootstrap Loader

(The RF11/RS11 Disk in this configuration may be replaced by an RK11 disk controller with 1 RK02 or RK03 Disk Cartridge, provided that 12K of core is available.)

1.4 MONITOR MESSAGES

Monitor messages are stored on the disk. When a message-producing situation (such as a system error) occurs, Monitor calls the correct message into core and prints it on the teleprinter.

There are five types of Monitor messages:

- Informational
- Action required by the operator
- Warning to the operator
- Fatal
- System Program error

The type of message is identified by the letters I, A, W and F respectively. If the system disk should fail and the error message cannot be brought into core, the Monitor halts.

Monitor messages are described in detail in Section 2.10.

1.5 STARTING THE MONITOR

The monitor is called into core from disk by performing the following procedure:

1. Set the Switch Register to 173100 (the address of the ROM Bootstrap Loader)
2. Depress LOAD ADDRESS
3. Set the Switch Register to the address of the word count register for the disk on which the Monitor resides (177462 for RF/RS11, 177450 for RC11/RS64, 177406 for RK11/RK02-03)
4. Depress START.

The monitor will load into core and identify itself by printing:

MONITOR Vxxxx

on the teleprinter, where Vxxxx represents the version number of the Monitor being used. The Monitor is now ready to accept an operator command (see Chapter 3).

1.6 A GUIDE TO THIS HANDBOOK

1.6.1 Terminology

The reader should understand the following terms as they apply to the PDP-11 Disk Operating System. An expanded glossary, with abbreviations, can be found in Appendix I.

A dataset is a logical collection of data which is treated as an entity by a program. For example:

- All or part of a file on a file-structured device.
- A paper tape in a paper tape reader.
- Three physically different files which together constitute the source input to the assembler.

A device is any PDP-11 peripheral supported by the Monitor.
A device controller may support one or more device units.

A file is a physical collection of data which resides on a directory device (e.g., disk or DECtape) and is referenced by its name. A file consists of one or more blocks on a directory device.

A block is a group of adjacent words of a specified size on a device; it is the smallest addressable segment of the device. If the blocks comprising a file are adjacent to each other, the file is said to be contiguous; if the blocks of the file are not adjacent, the file is said to be linked.

A line is a string of ASCII* characters which is terminated by a LINE FEED, FORM FEED or VERTICAL TAB.

File structure refers to the manner in which files are organized. Specifically, each of a user's files is given a unique name by the user. Each user on a file-structured device is assigned a User File Directory (UFD) in which each of his files is listed by name and location. Each UFD is then listed in a Master File Directory (MFD) which is unique to a specific device unit.

Bulk storage devices containing directories are called directory devices or file-structured devices. Devices such as paper tape equipment and the teleprinter, which cannot support a file structure, are called non-directory devices or non-file-structured devices.

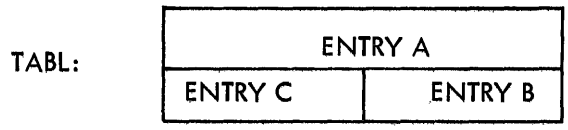
1.6.2 Standards for Tables

A table is a collection of data stored in sequential memory locations. A typical table as represented in this manual is shown below. This table is two words long, and is referenced by the symbolic address TABL:. The first entry is at location TABL and contains ENTRY A, which might be coded as .WORD

*ASCII stands for American Standard Code for Information Interchange.

AYE in the user's program. The second word of the table, at address TABL+2, is divided into two bytes. The low-order byte (address TABL+2) contains ENTRY B, and the high-order byte (address TABL+3) contains ENTRY C. They might be written into a program as .BYTE BEE,CEE.

a) Representation in manual:



b) Representation in program listing:

```
TABL:   .WORD AYE          ;ENTRY A
        .BYTE BEE,CEE      ;ENTRY B,ENTRY C
```

1.6.3 Standards for Numbers

Unless otherwise stated, all numbers in the text and examples are in octal.

CHAPTER 2
PROGRAMMED REQUESTS

2.1 INTRODUCTION

The user program calls for the services of the Monitor through programmed requests. These requests are macro calls which are assembled into the user program and interpreted by the Monitor at execution time. A programmed request consists of a one-word instruction followed, when appropriate, by one or more arguments. For example:

```
.WAIT LNKBLK
```

is a programmed request called .WAIT followed by an argument LNKBLK. The macro or request is expanded at assembly time by the DOS Assembler into a sequence of instructions which trap to and pass the arguments to the appropriate Monitor routine to carry out the specified function. The assembly language expansion for .WAIT LNKBLK is:

```
MOV #LNKBLK,-(SP)
EMT  1
```

The user may code a request in his program as either a macro call or as the equivalent assembly language program.

The request arguments are parameters or addresses of tables which contain the parameters of the request. These tables are also part of the user program, and are described in detail in Figures 2-5 to 2-12. Restrictions on argument names are found in the appropriate DOS Assembler Manual.

Services which the Monitor makes available to the user through programmed requests can be classified into three groups:

- requests for input/output and related services
- requests for directory management services
- requests for miscellaneous services

Table 2-1 summarizes the programmed requests available under the Monitor. They are described in general in Section 2.2.

Table 2-1
Summary of Monitor Requests

Mnemonic	Purpose
Requests for Input/Output and related services:	
.INIT	Associates a dataset with a device driver and sets up the initial linkage.
.RLSE	Removes the linkage between a device driver and a dataset, and releases the driver.
.OPENx	Opens a dataset.
.CLOSE	Closes a dataset.
.READ	Transfers data from a device to a user's line buffer.
.WRITE	Transfers data from a user's line buffer to a device.
.WAIT	Waits for completion of any action on a dataset.
.WAITR	Checks for completion of any action on a dataset, and provides a transfer address for a busy return.
.BLOCK	Transfers one block of a file between a device and a Monitor buffer.
.TRAN	Transfers data by absolute device block address between a device and a user buffer.
.SPEC	Performs special device functions.
.STAT	Obtains device characteristics.
Requests for Directory Management services:	
.ALLOC	Allocates a contiguous file.
.DELET	Deletes a file.
.RENAM	Renames a file.
.APPND	Appends one linked file to another.
.LOOK	Searches the directory for a particular file name and returns information about the file.
.KEEP	Protects a file against automatic deletion on FInish.

(Continued on next page)

Table 2-1 (Cont)
Summary of Monitor Requests

Mnemonic	Purpose
Requests for Miscellaneous services:	
.EXIT	Returns control to the Monitor.
.TRAP	Sets interrupt vector for the TRAP instruction.
.RSTRT	Sets the address used by the REstart command.
.CORE	Obtains address of highest word in core memory.
.MONR	Obtains address of first word above the resident Monitor.
.MONF	Obtains address of first word above the Monitor's highest allocated free core buffer.
.DATE	Obtains the date.
.TIME	Obtains the time of day.
.GTUIC	Gets current UIC.
.SYSDV	Gets Radix-50 name of System Device
.RADPK	Packs three ASCII characters into one Radix-50 word.
.RADUP	Unpacks one Radix-50 word into three ASCII characters.
.D2BIN	Converts five decimal ASCII characters into one binary word.
.BIN2D	Converts one binary word into five decimal ASCII characters.
.O2BIN	Converts six octal ASCII characters into one binary word.
.BIN2O	Converts one binary word into six octal ASCII characters.
.CSI1	Condenses a command string and checks for proper syntax.
.CSI2	Interprets one command string dataset specification.

2.2 TYPES OF PROGRAMMED REQUESTS

2.2.1 Requests for Input/Output and Related Services

All user I/O is handled by programmed requests, which provide three different levels of transfer:

- READ/WRITE
- BLOCK
- TRAN

Each level uses a sequence of requests to complete the transfer. Note the distinction between READ/WRITE, BLOCK, and TRAN as names of transfer levels, and .READ, .WRITE, .BLOCK, and .TRAN as specific requests within these levels.

Requests for I/O related services perform special device functions (such as rewinding a tape) and obtain device characteristics from device status words.

2.2.1.1 READ/WRITE Level Requests - This is the level at which the Monitor performs most of its services for the user. This is the most commonly used level of transfer. Among its users are the DOS Assembler and Edit-11 Text Editor programs, which input one line of ASCII characters at a time.

READ/WRITE I/O under the Monitor consists of transferring the contents of a dataset between a device and a line buffer. A line buffer is an area set up by the user in his program, into which he (or the Monitor) places data for output (or input). The line buffer may be preceded by the line buffer header, in which the user specifies the size and location of the line buffer and the mode (format) of the data.

The READ/WRITE user can specify nine different modes of transfer, in two categories: ASCII and Binary. Each mode is presented briefly here; more details are in Section 2.6.1 and Figure 2-10.

ASCII Modes:	Formatted ASCII Parity - Special
	Formatted ASCII Parity - Normal
	Formatted ASCII Nonparity - Special
	Formatted ASCII Nonparity - Normal
	Unformatted ASCII Parity - Normal
	Unformatted ASCII Nonparity - Normal
Binary Modes:	Formatted Binary - Special
	Formatted Binary - Normal
	Unformatted Binary - Normal

To implement a READ/WRITE transfer, the programmer follows the sequence of requests shown in Figure 2-1b. First, the programmer initializes the device to the dataset with the .INIT request. The argument of this request is the address of a table called the Link Block. Entries in this table specify the device involved in the approaching transfer so that the Monitor may eventually establish a link between that device and the dataset. The Link Block is described in detail in Figure 2-5. The .INIT calls the appropriate device driver into the free core buffer area, if it is not already there.

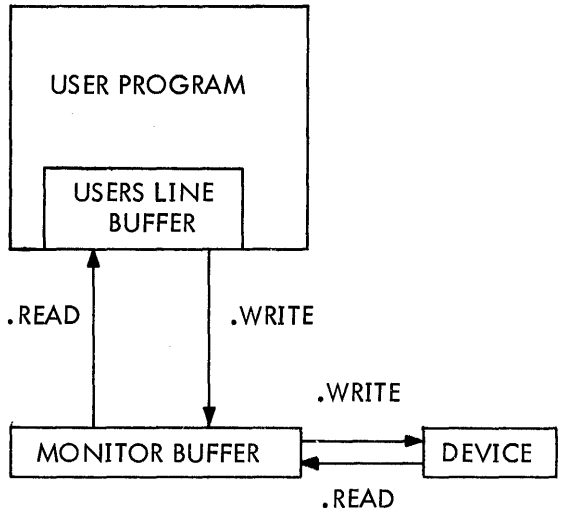
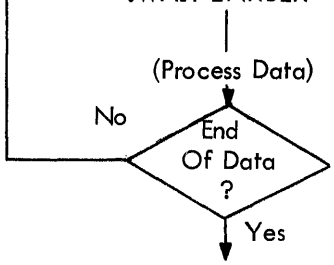


Figure 2-1a The Transfer Path

```
.INIT LNKBLK
.OPENI LNKBLK,FILBLK        ;FOR OUTPUT, REPLACE .OPENI
                            ; WITH .OPENO
.READ LNKBLK,BUFHDR         ;FOR OUTPUT, REPLACE .READ
                            ; WITH .WRITE
.WAIT LNKBLK                ;COULD BE REPLACED BY .WAITR
```




```
.CLOSE LNKBLK
.RLSE LNKBLK

LNKBLK:     (entries)
FILBLK:     (entries)     Tables in User's Program
BUFHDR:     (entries)
```

Figure 2-1b Sequence of Requests for READ/WRITE

Figure 2-1 .READ/.WRITE Input/Output Transfers

Following the .INIT request, the programmer opens a dataset with an .OPENx request. This need be done only if the device being used has a directory. However, it is advisable to use an .OPENx even for a nondirectory device to preserve the device independence of the program, i.e., the programmer may want to assign the transfer to a directory device later. The argument of this request is the symbolic address of a table called the Filename Block (Figure 2-6). Entries in this table specify the dataset involved in the transfer.

A dataset can be opened for input, for output, for update, or for extension. The last letter of the .OPENx request specifies which type of open is desired.

A .READ (for input) or a .WRITE (for output) follow the .OPENx. Either request causes a transfer to take place between the line buffer and the device via a buffer allocated by the Monitor in its free core area. The arguments of either request are the address of the Link Block for the dataset and the address of the Line Buffer Header (Figure 2-8). The Line Buffer Header specifies the area in the user's core area to or from which the dataset is to be transferred.

.READ or .WRITE are followed by .WAIT, which tests for the completion of the last transfer, and passes control to the next instruction. Typically, what follows a .WAIT on an input is a subroutine to process the portion of data just input. When the process has been completed, the program checks to see if it wants another portion of data; if it does, the program transfers control back to the .READ request and the process is repeated. If all data has been transferred, the .CLOSE request follows to complete any pending action, update any directories affected, and release to free core any buffer space the Monitor has allocated from free core. Finally, action on the dataset is formally terminated with the .RLSE request, which disassociates the device from the dataset, and releases the driver. Releasing the driver frees core provided there is no other claim to the driver from another dataset.

2.2.1.2 BLOCK Level Requests - BLOCK requests provide for random access of blocks in files stored on directory devices such as disk or DECtape. An example of a BLOCK user program is a Payroll Update Program which stores information about all employees on one file, with a set number of blocks assigned to each employee.

At this level, data is transmitted between a specified block of the file and the Monitor buffer (Figure 2-2a). The user program may directly access the data in the Monitor buffer, or may move it to its own area for further processing. BLOCK level requests require the use of the .INIT, .RLSE, .OPEN and .CLOSE requests, as in the READ/WRITE level requests.

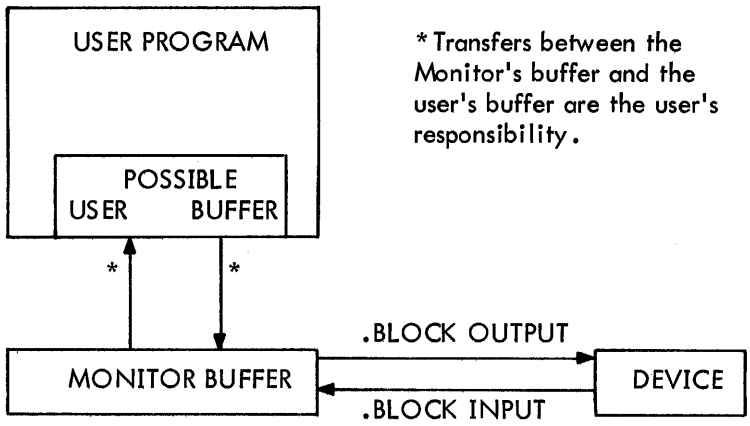


Figure 2-2a The Transfer Path

```
.INIT LNKBLK
.OPENU LNKBLK,FILBLK
.BLOCK LNKBLK,BLKBLK      ;INPUT DESIRED BLOCK
.WAIT LNKBLK              ;COULD BE REPLACED BY .WAITR
  |
(Process Data)            ;UPDATE DATA
  |
.BLOCK LNKBLK,BLKBLK      ;WRITE UPDATED BLOCK
.WAIT LNKBLK
  |
Yes <- More ? -> No
.CLOSE LNKBLK
.RLSE LNKBLK
LNKBLK:   (entries)  }
FILBLK:   (entries)  } Tables in User Program
BLKBLK:   (entries)  }
```

Figure 2-2b The Sequence of Requests For .BLOCK

Figure 2-2 .BLOCK Input/Output Transfers

To implement a BLOCK transfer, the programmer follows the sequence of requests shown in Figure 2-2b. Notice that the transfer must be INITed, OPENed, WAITed, CLOSEd, and ReLeaSEd following the same rules as the READ/WRITE level. The .BLOCK request has the address of the Link Block and the BLOCK block for its arguments. The BLOCK block specifies the block within the file that is to be transferred.

2.2.1.3 TRAN Level Requests - A TRAN level request is a basic input/output operation at the device driver level. Bulk storage devices are accessed by absolute block number without regard to file structure. For this reason, the user should be very careful not to destroy any files on the device on which he is performing TRAN level requests. He should allocate a contiguous file on the device for his purposes.

Data is transferred directly between the device and the user's line buffer (Figure 2-3a) with no formatting performed. TRAN level requests are generally used in two situations:

1. When the file structure does not allow the desired operation (for example, PIP uses TRAN to read a directory block).
2. When the user cannot afford the overhead of doing transfers by a READ/WRITE process, and the data is of a fixed format. (For example, a program to process data arriving at random intervals from an A/D converter might first dump the input data onto the disk via a .TRAN request as it arrives, and then read it back later for processing when time permits.)

To implement a TRAN transfer, the programmer follows the sequence of requests shown in Figure 2-3b. Notice that the transfer must be INITed and .RLSE'd, but is not .OPENed or .CLOSEd. The .TRAN request has the address of the TRAN Control Block as its argument. This block contains entries which specify the core starting address of the user's line buffer, the device block address, the number of words to be transferred, and the function to be performed. TRAN is therefore a device dependent request. A summary of the transfer levels which can be used on the various types of datasets is shown in Table 2-2.

Table 2-2
Transfer Levels for Types of Datasets

Type of Transfer	Type of Dataset: Linked File	Contiguous File	Nonfile-Structured Device
READ/WRITE	Yes	Yes	Yes
BLOCK		Yes	
TRAN	*	*	Yes

Yes indicates that the given transfer level will work on the given type of dataset.

* indicates that TRAN may be used on a file-structured device if the warnings mentioned are observed.

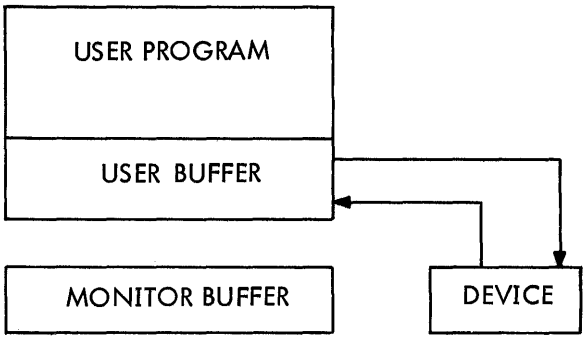


Figure 2-3a The Transfer Path

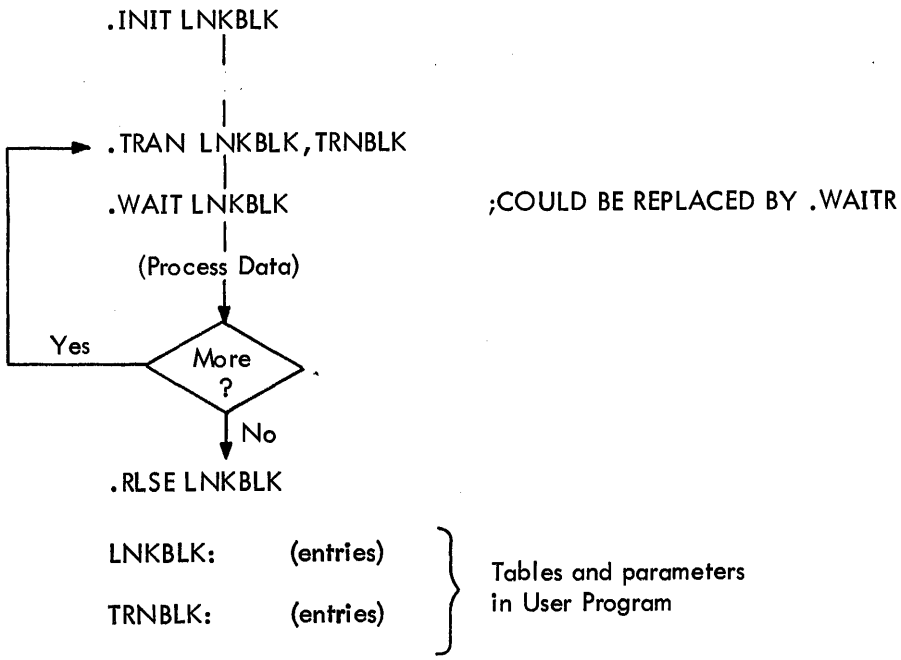


Figure 2-3b The Sequence of Requests For .TRAN

Figure 2-3 .TRAN Input/Output Transfers

2.2.2 Requests for Directory Management Services

Directory management requests are used to enter file names into directories, search for files, update file names, and protect files against deletion.

2.2.3 Requests for Miscellaneous Services

Requests for miscellaneous services include:

- Requests to return control to the Monitor from a running program.
- Requests to set Monitor parameters such as the TRAP vector or a program's restart address.
- Requests to obtain Monitor parameters such as the size of core, the size of the Monitor, the date, the time, and the current user's UIC.
- Requests to perform conversions between ASCII and Radix-50 packed ASCII, binary and ASCII decimal, and binary and ASCII octal.
- Requests to access the Command String Interpreter.

2.3 DEVICE INDEPENDENCE

Ordinarily, a programmer specifies input/output devices as he writes the program. However, there are circumstances when he will want to change the device specification when his program is run. For example:

- A device that the user specified when he wrote his program is not in operation at run time, but an alternate device is available.
- The programmer does not know the configuration of the system for which he is writing, or does not wish to specify it (i.e., he is writing a general purpose package).

The Monitor allows the programmer to write programs which are device independent in that the programmer can, but need not always, specify a device in his program. These facilities are:

- The programmer may specify a device for each dataset via a Link Block when he writes his program.
- A programmer can assign or reassign a device for a dataset through the keyboard with the ASsign command (Section 3.3.1.1) at run time. This command sets up a table entry in the Monitor which effectively overrides any entries in the Link Block.
- A general purpose program can dynamically request the device and filenames for each run via the keyboard and then obtain decoding and set-up via the Command String Interpreter.

Note that the substituted devices must be compatible. For example, the user may initially specify a BLOCK transfer from disk and later change the assignment to input from DECtape instead. But he cannot later specify paper tape reader as the input device, since BLOCK level requests do not apply to nonfile-structured devices.

It is important to note that a device is assigned in a program to a dataset logical name and that re-assigning a device at run time for one dataset logical name does not reassign that device for all dataset logical names to which it was originally assigned.

The only transfer request which is not device independent is .TRAN.

2.4 SWAPPING ROUTINES INTO CORE

Except for a small, permanently resident kernel, the Monitor routines which process most programmed requests are potentially swappable. They are normally disk resident and are swapped into core by the Monitor only when needed. The user may, however, specify that one or more of these potentially swappable routines be made permanently core resident or core resident just for the duration of his program's run. Making a potentially swappable routine core resident ties up core space, but speeds up operation on the associated request. The user may, for example, be collecting data via a .TRAN request in a real-time environment. In such a case, even the short time needed to swap in the .TRAN request processor could cause him to lose data.

Potentially swappable routines are made core resident by one of the following.

- Routines may be made permanently core resident in the Monitor by specifying the appropriate GLOBAL NAME at system generation time.
- Routines may be made resident for the duration of a program's run by declaring the appropriate GLOBAL NAME (as specified in the definition of each request in Sections 2.6 through 2.8) in a .TRAN processor directive in the program. For example, to make the .TRAN processor resident while program FROP is being run, the following directive would be included in program FROP:

```
.GLOBL  TRA.
```

- In the absence of either of the above specifications for a routine, the Monitor will swap in that routine whenever it is requested.

Any routine which services a programmed request (other than RWN) is potentially swappable; i.e., those given Global Names in this Chapter.

2.5 MONITOR RESTRICTIONS ON THE PROGRAMMER

In return for the services provided by the Monitor, the programmer must honor certain restrictions:

- The programmer should not use either the EMT or the IOT instructions for communication within his program.
- It is recommended that the user not raise his interrupt priority level above 3, since it might lock out a device that is currently trying to do input/output.

- HALTS are not recommended. If a HALT is executed during an I/O operation, most devices will stop, and only recovery from the console (pressing the CONTinue switch on the console) will be effective (recovery from the keyboard will not be immediately possible, since a HALT inhibits the keyboard interrupt). Some devices, such as DECtape, will not see the HALT and will continue moving, will lose their positions over the block under transfer, and may even run tape off the reel.
- The RESET instruction should not be used becduse it forces a hardware reset; clearing all buffers and flags and disabling all interrupts, including the keyboard's. Since all I/O is interrupt driven, RESET will disable the system.
- The user must be careful to avoid penetrating the Monitor when he is using the stack. The stack is set by the RUN time loader just below the lowest address of the program loaded. The Monitor checks to see that the stack is not overflowing each time it honors a request. The user can relocate the stack pointer and claim more space as follows:

  a. He can determine where the pointer is currently and where the current top of Monitor is located, then move the stack pointer down the correct amount.

  b. He can ask the Monitor for buffer space via the general utilities (see PDP-11 DOS Monitor, System Programmer's Manual).

- The user should be aware that certain instructions, such as .INIT, may change the amount of available free core, since they may call in drivers and establish data blocks. Such requests effect the results of the MONR or MONF requests.
- Certain requests return data to the user on the stack. The user must clear the stack himself before the stack is used again. The Monitor clears the stack after it honors requests that do not return data to the user on the stack.
- The user should not use GLOBAL names that are currently used by the Monitor. All these names are listed in Appendix E.

2.6 DEFINITION OF REQUESTS FOR INPUT/OUTPUT SERVICES

Each request has one or more arguments which are addresses of tables in the user's program. The tables specify the variables of the request. Table entries are explained in detail in Figures 2-5 to 2-12 at the end of this section.

2.6.1 READ/WRITE Level Requests

This is the level at which the Monitor performs most of its services for the user. The user can specify nine different modes of transfer, in two categories; ASCII and Binary. Each mode is discussed here, and is presented in detail in Figure 2-10.

ASCII Modes: Formatted ASCII Parity - Special
Formatted ASCII Parity - Normal
Formatted ASCII Nonparity - Special

(Continued on next page)

ASCII Modes (Cont)

Formatted ASCII Nonparity - Normal

Unformatted ASCII Parity - Normal

Unformatted ASCII Nonparity - Normal

Binary Modes: Formatted Binary - Special

Formatted Binary - Normal

Unformatted Binary - Normal

1. Formatted and Unformatted ASCII Modes:

   Data in formatted ASCII modes is assumed by the Monitor to be in strings of 7- or 8-bit ASCII characters terminated by LINE FEED, FORM FEED or VERTICAL TAB. PAL-11R Assembler source programs are in a formatted ASCII mode. In these modes, the Monitor manages all device-dependent conversions at the driver level. For example, LINE FEED is supplied after RETURN in character strings from keyboard terminals.

   Data in unformatted ASCII modes is also assumed to be in strings of 7- or 8-bit ASCII characters. Checks for terminators and device-dependent conversion are not performed by the Monitor, thus allowing the user to handle all characters in his own way.

2. ASCII Parity and Nonparity Modes:

   In ASCII nonparity modes, 7-bit ASCII characters are transferred.

   In formatted ASCII parity modes, even parity is generated in the 8th bit and is checked during the transfer. In unformatted ASCII parity mode, 8 bits are transferred, but no parity is generated or checked.

3. Normal and Special Modes:

   Special modes provide additional Monitor facilities over and above normal modes; normal modes are compatible with the PDP-11 I/O Executive (IOX).

4. Formatted and Unformatted Binary Modes:

   Data in formatted binary modes is transferred in 8-bit bytes as read from the device. The Monitor makes no assumptions about the nature of the data. A checksum is calculated during a WRITE request and transmitted with the data, as well as a count of the number of bytes. The checksum is verified during a READ. The binary output of the PAL-11R Assembler, for example, is in a formatted binary mode.

   Unformatted binary mode is the same as formatted binary except that no checksum or count is calculated or verified.

NOTE

A dataset can only support transfers in one direction, i.e., READ only or WRITE only. If the same device is to be used for both operations, separate datasets must be used for each.

2.6.1.3 .OPEN - Prepare .INITed device for usage and make a named file available if the device is directory oriented.

Macro Call: .OPENx LNKBLK,FILBLK

where x indicates the type of OPEN:

U for update
O for output
E for extension
I for input
C for create data in contiguous file
LNKBLK = address of Link Block
FILBLK = address of Filename Block

Assembly Language

Expansion:

```
MOVB #CODE,FILBLK-2          ;MOVE "HOW OPEN"
                             ;CODE TO FILENAME BLOCK
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  16
```

where CODE indicates the type of OPENx request:

OPEN0 = 2
OPENI = 4
OPENU = 1
OPENC = 13
OPENE = 3

Global Name: OPN. (See Appendix C for subsidiary routines.)

Description: In general, an .OPENx request causes the Monitor to allocate a data buffer and to make other necessary preparations for transferring to a dataset to or from a device. More specifically:

.OPENU	opens a previously created contiguous file for input and output by .BLOCK.
.OPENO	creates a new linked file and prepares it to receive output.
.OPENE	opens a previously created linked file to make it longer.
.OPENI	opens a previously created linked or contiguous file for input to the computer. It normally precedes all .READ operations.
.OPENC	opens a previously created contiguous file for output from the computer.

After the open request has been processed, control is returned to the user at the instruction following the assembly language expansion; the arguments are removed from the stack. At this time, however, the device concerned may still be completing operations required by the request. A summary of transfer requests which may legally follow OPEN requests is illustrated in Table 2-3.

Table 2-3
Transfer Requests Which May Follow Open Requests

Type of File / Type of Transfer / Transfer Request / Type of Open	Linked File		Contiguous File				File Already Exist?
	Input	Output	Input		Output		
	.READ	.WRITE	.READ	.BLOCK	.WRITE	.BLOCK	
.OPENU				YES		YES	must
.OPENO		YES					must not
.OPENE		YES					must
.OPENI	YES		YES	YES			must
.OPENC					YES		must

Rules:

a. General Rules for All .OPENx Requests

The user must set up a Filename Block in his program (Figure 2-6). If the dataset is a file, the Filename Block must contain a legal file name. A file name consists of up to six characters (A-Z, $ 0-9); the first character cannot be a digit (0-9), it may be followed by an extension of 3 characters. If the dataset is not a file, the Filename Block need not contain any FILENAME or EXTENSION entries.

All datasets must have been INITed before they are OPENed. Type of OPEN must be applicable to type of device (e.g., OPENI to line printer is illegal).

For datasets on directory devices, the User Identification Code (UIC) in the Filename Block (if specified) must be in the directory of the device. If the UIC is not specified, the user must have logged in with a UIC that appears on the device.

The .OPENx request must not violate the protect code of the file.

If a dataset is opened for any output, it cannot be opened again until it has been closed.

b. Rules for .OPENO

The .OPENO request is applicable only for outputs to nonfile-structured devices or to a linked file on a file-structured device.

The .OPENO request creates a linked file on a directory device; hence, the file referenced in the corresponding Filename Block cannot exist prior to the .OPENO request.

The .OPENO request will return an error if the directory is full.

c. Rules for .OPENI

.OPENI may be used for inputs from contiguous or linked files, or nondirectory devices.

The file referenced in the corresponding Filename Block must exist on the directory.

If a file is open for input (OPENI), it cannot be opened for output, but it can be opened for extension or update.

At any one time, a file can be opened for input to a maximum of 62_{10} or 76_8 datasets.

d. Rules for .OPENU, .OPENE and .OPENC

The file must exist and cannot currently be opened for output.

The file cannot currently be opened by .OPENU, .OPENE or .OPENC.

A contiguous file cannot be opened for extension.

A linked file cannot be opened with .OPENC, which is applicable only to contiguous files.

Errors: If any of the preceding rules are violated, the Monitor places an error code in the STATUS byte of the Filename Block (see Table 2-4) and transfers control to the pointer in the ERROR RETURN ADDRESS of the Filename Block. If this address is 0, a fatal error message is printed on the console. Fatal error messages are listed in Section 2.10.

Example: (See .CLOSE)

2.6.1.4 .CLOSE - Close a dataset.

Macro Call: .CLOSE LNKBLK

where LNKBLK is the address of the Link Block.

Assembly Language

Expansion:

```
MOV   #LNKBLK,-(SP)
EMT   17
```

Global Name: CLS. (See Appendix C for subsidiary routines.)

Description: The close request indicates to the Monitor that no more I/O requests will be made on the dataset. Close completes any outstanding processing on the dataset, updates any directories affected by the processing, and releases to free core any buffer space established for the processing. For example, if .CLOSE had been preceded by an .OPENE request to a file, the added portion is linked to the file, the directory entry for the file is updated to acknowledge the added portion, and buffers used for data and Monitor internal file information tables are released to free core. After the close request has been completed, control is returned to the user at the instruction following the assembly language expansion; the argument is removed from the stack. As with OPEN, some appropriate device action may still be in progress at this point.

Rules: The dataset to be closed must have previously been opened if it was a file on a directory device. As with .OPENx, a .CLOSE is not required if the dataset is not a file, but it is strongly recommended.

Errors: Dataset Not Inited - Fatal Error F000; Device Parity Error - Fatal Error F017.

Example: Open for input a dataset named IMP, which is file PROG1.BIN on DECtape unit 3. After the data transfer is complete, close the file.

```
.INIT  SET1
   :
   :
.OPENI  SET1,FILE1          ;OPEN SET1 FOR INPUT
   :
   :
(Input is performed here)
   :
   :
.CLOSE SET1                 ;CLOSE SET1
```

(Continued on next page)

```
        .RLSE   SET1
        .WORD   ERR1
SET1:   .WORD   0
        .RAD50  /IMP/
        .BYTE   1,3
        .RAD50  /DT/
        .WORD   ERF1            ;ADDR OF ERROR RTN
        .WORD   0               ;MACRO PUTS HOW-OPEN HERE
FILE1:  .RAD50  /PRO/           ;FILE NAME
        .RAD50  /G1/
        .RAD50  /BIN/           ;EXTENSION
        .BYTE   PROG,PROJ       ;USER ID CODE
        .BYTE   177             ;PROTECT CODE
```

2.6.1.5 .READ - Read from device.

Macro Call: .READ LNKBLK,BUFHDR

where LNKBLK is the address of the Link Block, and BUFHDR is the address of the line buffer header.

Assembly Language

Expansion:
```
MOV  #BUFHDR,-(SP)
MOV  #LNKBLK,-(SP)
EMT  4
```

Global Name: RWN.

Description: The .READ request transfers the data specified in the line buffer header from the device to the user's line buffer. The transfer is done via a buffer in the Monitor, into which an entire device block is read, and from which the desired data is transferred to the user's line buffer. (If the data requested traverses a device block boundary, a second device block is read.) After any I/O transfer has been started, control is returned to the user at the next instruction, with the arguments removed from the stack.

Rules: If the device is file structured, the .READ request must be preceded by an .OPENI.

The user must provide in his program a line buffer and line buffer header (see Figure 2-8).

Further actions on the dataset by the Monitor will be automatically postponed until the .READ processing has completed. The user program should, however, perform a .WAIT or .WAITR to ensure proper completion of transfer before attempting to use the data in the line buffer.

Errors: Specification of a transfer mode which is inappropriate for the device assigned to the dataset, and attempting to .READ from or .WRITE to a file-structured device for which no file has been .OPENed or the type of .OPEN is incorrect. These will be treated as a fatal error and result in a F010 message.

2.6.1.6 .WRITE - Write on a device.

Macro Call: .WRITE LNKBLK,BUFHDR

where LNKBLK is the address of the Link Block, and BUFHDR is the address of the line buffer header.

Assembly Language

Expansion:
```
MOV  #BUFHDR,-(SP)
MOV  #LNKBLK,-(SP)
EMT  2
```

Global Name: RWN.

Description: The .WRITE request initiates the transfer of data from the user's line buffer to the device assigned. The data is first transferred to a buffer in the Monitor, where it is accumulated until a buffer of suitable length for the device is filled. The data in the Monitor buffer is then transferred to the appropriate device block, and any data remaining in the user's line buffer is moved to the (emptied) Monitor buffer. After any I/O transfer to the device has been started, control is returned to the user at the next sequential instruction. The arguments are removed from the stack upon return.

Rules: If the requested device is file structured, the dataset must have been opened by an .OPENO or .OPENE for a linked file, or .OPENC for a contiguous file.

The user must provide a line buffer and its header in his program (Figure 2-8).

Further actions on the dataset by the Monitor after .WRITE will be automatically postponed until the .WRITE processing has been completed. Before refilling the line buffer, however, the user program should perform a .WAIT or .WAITR to insure proper completion of the transfer.

Errors: See .READ for errors.

2.6.1.7 .WAIT - Wait for completion of process on dataset.

Macro Call: .WAIT LNKBLK

where LNKBLK is the address of the Link Block.

Assembly Language

Expansion:
```
MOV #LNKBLK,-(SP)
EMT 1
```

Global Name: (Routine is permanently core resident.)

Description: .WAIT tests for completion of the last requested action on the dataset represented by the referenced Link Block. If the action is complete (that is, if the request has completed all its action), control is returned to the user at the next sequential instruction following the assembly language expansion; otherwise, the Monitor retains control until the action is complete. A .WAIT or .WAITR should be used to ensure the integrity of data transferred to or from a line buffer. The argument is removed from the stack.

Rules: The dataset must be INITed.

Errors: If the dataset is not INITed, a fatal error occurs and F000 is printed to the teleprinter.

2.6.1.8 .WAITR - Wait for completion of processing on dataset, or return.

Macro Call: .WAITR LNKBLK,ADDR

where LNKBLK is the address of the Link Block, and ADDR is the address to which control is transferred if the processing is not complete.

Assembly Language

Expansion:
```
MOV  #ADDR,-(SP)
MOV  #LNKBLK,-(SP)
EMT  0
```

Global Name: (Permanently Core Resident.)

Description: .WAITR tests for completion of the last requested action on the specified dataset. If all actions are complete, control is transferred back to the user at the next sequential instruction following the assembly language expansion. If all actions are not complete, control is given to the instruction at location ADDR. The arguments are removed from the stack. It is the user's responsibility to return to the .WAITR to check again.

Rules: The user should use a .WAIT or a .WAITR request to assure the completion of data transfer to the user's line buffer before processing the data in the buffer, or moving data into it. The dataset must be INITed.

Errors: If the dataset is not INITed, a fatal error occurs and F000 is printed on the teleprinter.

2.6.2 BLOCK Level Requests

BLOCK requests provide for the random access to the blocks of files stored on the disk or DECtape. In this mode, data is transmitted to or from a specified block in a file with no formatting performed. Transfers take place between the device block and the Monitor buffer. The user is responsible for transferring the block to and from his own area. BLOCK level requests require the use of the .INIT, .RLSE, .OPEN and .CLOSE requests discussed earlier.

2.6.2.1 .BLOCK - Transfer one physical block of a file.

Macro Call: .BLOCK LNKBLK,BLKBLK

where LNKBLK is the address of the Link Block, and BLKBLK is the address of the BLOCK block (see Figure 2-11).

Assembly Language

Expansion:
```
MOV  #BLKBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  11
```

Global Name: BLO.

Description: This request allows for random, relative block access to contiguous files. The user must specify one of three functions in the block called: INPUT, GET, or OUTPUT. After the transfer has started, control is returned to the user at the instruction following the assembly language expansion with arguments removed from the stack.

INPUT: During an INPUT request, the requested block of the requested file is read into a Monitor buffer, and the user is given in the BLOCK block (see Figure 2-11) the address of the buffer and the physical length of the block transferred.

GET: During a GET request, the Monitor gives the user the address and length of a buffer within the Monitor that he can fill for subsequent output. The user must be careful that he does not over-run the buffer. This request is unnecessary if an INPUT request has occurred.

OUTPUT: During an OUTPUT request, the contents of the buffer assigned is written on the device in the requested relative position of the requested file.

Rules: The associated file must be opened by .OPENI for input or .OPENU for input or output.

Access to linked files or nondirectory devices is illegal.

The user must set up the BLOCK block in his program according to the format of Figure 2-11.

Errors: Error processing causes a return to the user as usual, with the type of error indicated in the FUNCTION/STATUS word of the BLOCK block. The user should perform

```
TSTB  BLKBLK+1
BNE   ERROR
```

after a .WAIT to ensure that his request was error free.

2.6.3 TRAN Level Requests

TRAN requests provide for direct access to any device. Bulk storage or directory devices are accessed by absolute block without regard to the directory structure. For this reason, the user should be very careful not to destroy the file structure of a directory device to which he is requesting TRAN level transfers. Data is transferred directly between the device and the user's buffer. No formatting is performed.

TRAN requests require the use of the .INIT and .RLSE requests, discussed earlier.

2.6.3.1 .TRAN - Transfer absolute block.

Macro Call: .TRAN LNKBLK,TRNBLK

where LNKBLK is the address of the Link Block, and TRNBLK is the address of the TRAN block.

Assembly Language

Expansion:
```
MOV  #TRNBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  10
```

Global Name: TRA.

Description: .TRAN performs a direct transfer of data, by absolute block on the device (or next block on sequential devices), between the device and the user's area in core memory. No Monitor buffering or formatting occurs. After the transfer has started, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack. The user is warned that .TRAN provides no protection for files on a directory-oriented device.

Rules: .TRAN must be preceded by an .INIT request on the associated dataset.

For each .TRAN request, the user must provide a transfer control block, as shown in Figure 2-12.

Further actions on the dataset by the Monitor will be automatically postponed until the .TRAN processing has been completed. The user program should perform a .WAIT or .WAITR to ensure proper completion of the transfer before attempting to reference any location in the data buffer.

If file structured data shares the same device as the block(s) referenced by the .TRAN request, it is recommended that the user first allocate a contiguous file for .TRAN usage.

Errors: An invalid function code in the transfer control block will result in an error diagnostic message on the teleprinter at run time.

Errors in the transfer will be shown in the FUNCTION/STATUS word of the TRAN block; the last word of the block will be set to show how many data words have not been transferred.

Example: Transfer 200_8 words of ASCII data from DECtape unit 3, starting at block 100_8 to core starting at location 4000_8.

```
            .INIT  TAPE1
                :
                :
            .TRAN  TAPE 1,BIN40
                :
                :
            .RLSE TAPE 1
                :
                :
            .WORD ERR 1
TAPE 1:     .WORD 0
            .RAD50 /TP1/
            .BYTE 1,3
            .RAD50 /DT/
                :
BIN40:      .WORD 100           ;STARTING BLOCK #
            .WORD 4000          ;STARTING ADDRESS IN CORE
            .WORD 200           ;NUMBER OF WORDS
            .WORD 4             ;INPUT IN ASCII
            .WORD 0             ;FOR MONITOR USE
```

2.6.4 Requests for Input/Output Related Services

2.6.4.1 .SPEC - Special functions.

Macro Call: .SPEC LNKBLK,SPCARG

where LNKBLK is the address of the Link Block, and SPCARG may be either a special function code or the address of a special function block containing the code, depending upon the function.

Assembly Language

Expansion:
```
MOV   #SPCARG,-(SP)
MOV   #LNKBLK,-(SP)
EMT   12
```

Global Name: SPC.

Description: This request is used to specify a special function (action) to a device, such as rewind magnetic tape. A code identifies the function and must be in the range $0\text{-}255_{10}$. Where the function requires no supporting data, the code itself is the first parameter to be placed upon the processor stack in the assembly language call sequence. If, however, either the user must supply additional information or the function expects to return data to the user, the code is passed within a Special Function Block and the address of the block is the call parameter. The format of this block is shown in Figure 2-13.

If a .SPEC request is made to a device which has no special function code, an immediate return is made showing that the function has been complete. After the request has been started, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared.

Rules: The dataset must be INITed.

Errors: Fatal Error F000 is returned if the dataset has not been INITed.

2.6.4.2 .STAT - Obtain device status.

Macro Call: .STAT LNKBLK

where LNKBLK is the address of the Link Block.

Assembly Language

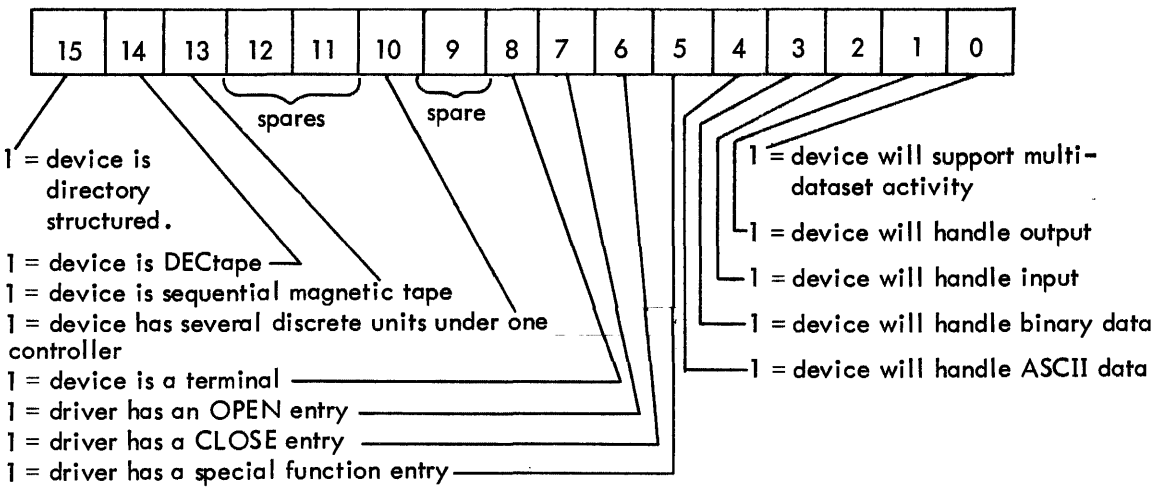
Expansion:
```
MOV  #LNKBLK,-(SP)
EMT  13
```

Global Name: STT.

Description: Determine for the user the characteristics of the device specified in the Link Block. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. This request returns to the user with the following information at the top of the stack:

SP	Driver Facilities Word
SP+2	Device Name
SP+4	Device Standard Buffer Size

where Driver Facilities Word has the following format:



Device Name is the Radix-50 packed ASCII standard mnemonic for the device (Appendix A).

Device Standard Buffer Size is the block size on a blocked device or an appropriate grouping size on a character device.

Rules: The dataset must be INITed. The user must clear the stack upon return.

2.7 DEFINITIONS OF REQUESTS OF DIRECTORY MANAGEMENT SERVICES

2.7.1 .ALLOC

Allocate (create a contiguous file).

Macro Call: .ALLOC LNKBLK,FILBLK,N

where LNKBLK is the address of the Link Block, FILBLK is the address of the Filename Block, and N is the number of 64-word segments requested.

Assembly Language

Expansion:

```
MOV   #N,-(SP)
MOV   #FILBLK,-(SP)
MOV   #LNKBLK,-(SP)
EMT   15
```

Global Name: ALO. (See Appendix C for subsidiary routines.)

Description: Searches the device for a free area equal to N 64-word segments, and creates a contiguous file in the area if it is found, by making an appropriate entry in the User File Directory. (Linked files are created by an .OPENO request.) Search begins at the high end of the device. The number of blocks allocated will be the minimum number required to satisfy N segments, i.e.,

$$\left|\frac{N}{B}\right|$$

where B is the number of segments per block. For example, if N = 9 for DECtape, and

$$B = \frac{256}{64} = 4 \qquad \text{therefore,} \qquad \left|\frac{N}{B}\right| = \left|\frac{9}{4}\right| = 3$$

After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack, and the top word of the stack will be set to -1 to indicate the successful completion of the request, or to the largest number of segments currently available if this is less than the called request. The value will be meaningless if the call cannot be met by reason of any other error.

Rules: Must be preceded by an .INIT request on the dataset. A Filename Block must be set up by the user in his program.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or to the console for an error message if it is not. Possible errors resulting from .ALLOC are:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	None	F000
Device Not Ready	None	A002
File Exists	2	F024
Illegal File Name	15	F024
UIC Not In Directory	13	F024
Directory Full	12	F024

If the error address in the Filename Block is taken, the top word of the stack is meaningless.

Example: Create a contiguous file of 1024_{10} words on DECtape unit 4. Name the file FREQ.DAT.

```
                    .
                    .
                    .
                .ALLOC    FRQ,FREQIN,20
                INC       @SP
                BNE       NOROOM
                    .
                    .
                    .
                .WORD     ERR1
FRQ:            .WORD     0
                .RAD50    /DTA/
                .BYTE     1,4
                .RAD50    /DT/
                    .
                    .
                    .
                .WORD     ERR2
                .WORD     0
FREQUIN:        .RAD50    /FRE/
                .RAD50    /Q/
                .RAD50    /DAT/
                .WORD     UIC,PROT1
                    .
                    .
                    .
ERR1:                                 ;TO HERE IF NO BUFFER AVAILABLE
                                      ;FOR DRIVER
ERR2:                                 ;TO HERE IF NOT ENOUGH CONTIGUOUS
                                      ;BLOCKS ON DEVICE
```

2.7.2 .DELET

Delete a file.

Macro Call: .DELET LNKBLK,FILBLK

where LNKBLK is address of Link Block, and FILBLK is address of Filename Block.

Assembly Language

Expansion:
```
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  21
```

Global Name: DEL. (See Appendix C for subsidiary routines.)

Description: Deletes from directory-oriented device the file named in the Filename Block. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack.

Rules: .DELET operates on both contiguous and linked files. If the file has been opened, it must be closed before it is deleted.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or to the console for an error message if it is not. Possible errors resulting from .DELET are:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	None	F000
Device Not Ready	None	A002
Non-existent File	2	F024
Protect Code Violate	6	F024
File Is Open	14	F024

2.7.3 .RENAM

Rename a file.

Macro Call: .RENAM LNKBLK,OLDNAM,NEWNAM

where LNKBLK is the address of the Link Block, OLDNAM is the address of the Filename Block representing the file, and NEWNAM is the address of the Filename Block containing the new information.

Assembly Language

Expansion:
```
MOV  #NEWNAM,-(SP)
MOV  #OLDNAM,-(SP)
MOV  #LNKBLK,-(SP)
EMT  20
```

Global Name: REN. (See Appendix C for subsidiary routines.)

Description: Allows the user to change the name and protection code of a file. After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack.

Rules: Dataset must be INITed, and file must not be opened. The user must specify two Filename Blocks; one contains the name and protection code of the file as it presently is before the .RENAM request, and the other contains the name and protection code of the file as it should be after the .RENAM request. The two file names must be different. To change just the protection for a file, two .RENAMs must be requested.

Only the owner of a file may rename it. The new file name must not already exist, and the new file name must be legal.

The old file must exist.

NOTE

Renaming a file assigned from the keyboard to the dataset will effectively be a NOP.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the offending Filename Block if it is specified and applicable, or to the Monitor for an error message if it is not. Possible errors resulting from .RENAM are:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
File Exists (New Name)	2	F024
File Does Not Exist (Old File)	2	F024
Dataset Not INITed	None	F000
File Is Open	14	F024
Protection Violation	6	F024
Illegal File Name	15	F024

2.7.4 .APPEND

Append one linked file onto another.

Macro Call: .APPEND LNKBLK,FIRST,SECOND

where LNKBLK is the address of the Link Block, FIRST is the address of the Filename Block for the first file, and SECOND is the address of the Filename Block for the second file.

Assembly Language

Expansion:

```
MOV  #SECOND,-(SP)
MOV  #FIRST,-(SP)
MOV  #LNKBLK,-(SP)
EMT  2
```

Global Name: APP. (See Appendix C for subsidiary routines.)

Description: Makes one linked file out of two by appending the SECOND to the FIRST. The directory entry of the SECOND file is deleted. When the request is completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack. No attempt is made to pack the two files together, the physical blocks are merely linked together.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the offending Filename Block if it is specified, or to the console for an error message if it is not. Possible errors resulting from .APPEND are:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	None	F000
First File Nonexistent	2	F024
Contiguous File	5	F024
Device Not Ready	None	A002
Protect Code Violated	6	F024
File Opened	14	F024

2.7.6 .KEEP

Protect file from automatic deletion.

Macro Call: .KEEP LNKBLK,FILBLK

where FILBLK is the address of the Filename Block of the file to be protected.

Assembly Language

Expansion:
```
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  24
```

Global Name: PRO.

Description: Protects the named file from being deleted by the Monitor upon a FInish command (Section 2.3.5.5). It does this by setting bit 7 of the PROTECT byte in the Filename Block.

2.8 DEFINITION OF REQUESTS FOR MISCELLANEOUS SERVICES

2.8.1 Requests to Return Control to the Monitor

.EXIT

2.8.1.1 .EXIT - Exit from program to Monitor.

Macro Call: .EXIT

Assembly Language

Expansion: EMT 60

Global Name: XIT.

Description: This is the last executed statement of a user's program. It returns control to the Monitor, insures that all of the program's data files have been closed and, in general, prepares for the next keyboard request. After the exit, all Monitor buffer space reserved for the program, such as Device Assignment Tables (DAT) established after the program was coded, are returned to free core.

2.8.2 Requests to Set Monitor Parameters

In addition to the above programmed requests, the user can provide the Monitor with data to be stored in Monitor Tables or can request information on the content of those tables via the EMT level 41 instruction. The user communicates his request to the Monitor by pushing the necessary parameters and an identifier code onto the stack. If the code is outside the ranges of those currently established, a fatal error will result (F002).

2.8.2.1 .TRAP - Set interrupt vector for the trap instruction.

Macro Call: .TRAP STATUS,ADDR

where ADDR is the address for trap, STATUS is the desired status for the trap.

Assembly Language

Expansion:

```
MOV  #ADDR,-(SP)
MOV  #STATUS,-(SP)
MOV  #1,-(SP)
EMT  41
```

Global Name: GUT.

Description: Sets the STATUS and ADDR into trap vector 34. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared. The user may then use the trap instruction.

2.8.2.2 .RSTRT - Sets address used by the REstart command.

Macro Call: .RSTRT ADDR

where ADDR is the restart address.

Assembly Language

Expansion:
```
MOV  #ADDR,-(SP)
MOV  #2,-(SP)
EMT  41
```

Global Name: GUT.

Description: Sets the address where the program should restart in response to the keyboard command REstart. This is the assumed address in the absence of an address in the REstart operator command. It can be reset as often as requested by the program. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The stack is cleared.

2.8.3 Requests to Obtain Monitor Parameters

.CORE

2.8.3.1 .CORE - Obtain address of the highest word in core memory.

Macro Call: .CORE

Assembly Language

Expansion:
```
MOV  #100,-(SP)      ;CODE
EMT  41
```

Global Name: GUT.

Description: Determines the address of the highest word in core memory (core size minus 2) and returns it to the top of the stack. For an 8K machine, it would return 37776. The user must clear the stack.

2.8.3.2 .MONR - Obtain the address of the first word above the Monitor.

Macro Call: .MONR

Assembly Language

Expansion:
```
MOV  #101,-(SP)
EMT  41
```

Global Name: GUT.

Description: Determines the first word above the top of the currently resident Monitor (see Figure 2-4) and returns it to the user at the top of the stack. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The user must clear the stack.

2.8.3.3 .MONF - Obtain the address of the first word above the Monitor's highest allocated free core buffer.

Macro Call: .MONF

Assembly Language Expansion:
```
MOV  #102,-(SP)
EMT  41
```

Global Name: GUT.

Description: The address of the first word above total Monitor area (in VØØ4A, last word of the present Monitor area) (see Figure 2-4), including the buffer and transient areas current at the time of the request, is returned to the user at the top of the stack. After the request is completed, control is returned to the user at the instruction following the assembly language expansion. The user must clear the stack.

Rules: Since buffers are allocated by the Monitor in its processing of certain requests, .MONF should be placed in the program at the point where the information is actually required.

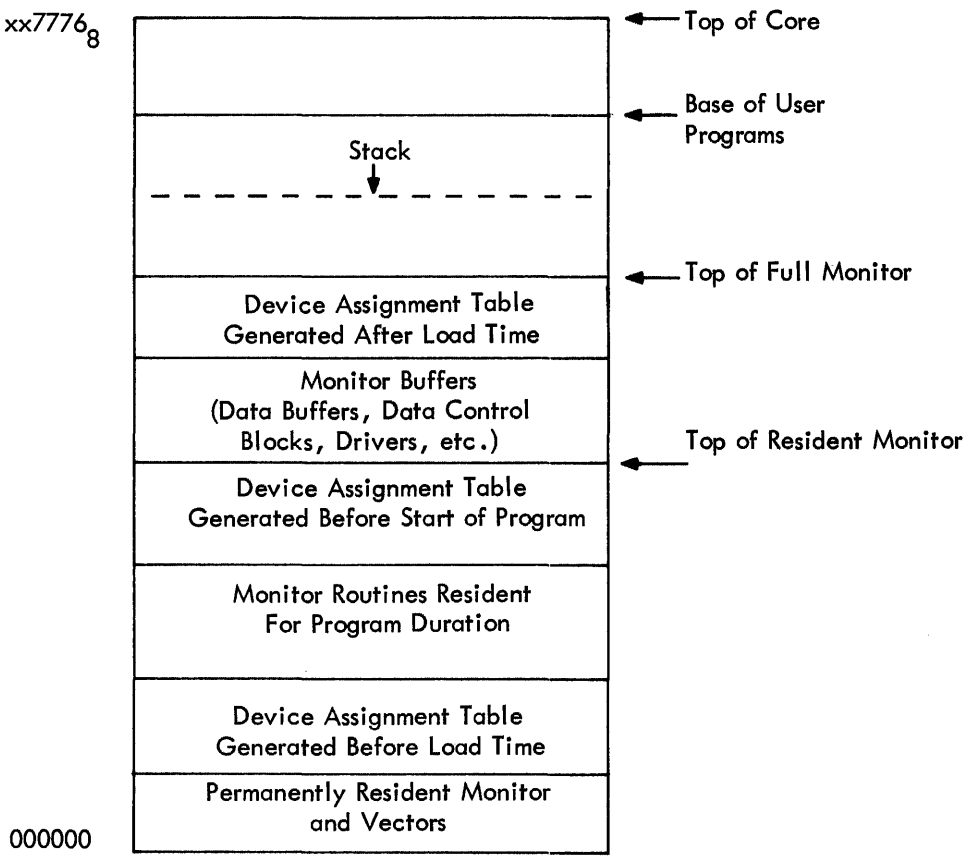


Figure 2-4 Core Map of Resident Monitor and Full Monitor.

2.8.3.4 .DATE - Obtain current date.

Macro Call: .DATE

Assembly Language

Expansion:
```
MOV  #103,-(SP)
EMT  41
```

Global Name: GUT.

Description: The current date word is returned to the user at the top of the stack. The user must clear the stack. The date format is Julian-$70,000_{10}$.

2.8.3.5 .TIME - Obtain current time of day.

Macro Call: .TIME

Assembly Language

Expansion:
```
MOV  #104,-(SP)
EMT  41
```

Global Name: GUT.

Description: The two current time words are returned to the user at the top of the stack.

LOW-ORDER TIME IN TICS	SP
HIGH-ORDER TIME	SP+2

where a TIC is 1/60 of a second (1/50 second for 50 cycle lines). The words are 15-bit unsigned numbers. The user must clear the stack.

2.8.3.6 .GTUIC - Get current user's UIC.

Macro Call: .GTUIC

Assembly Language

Expansion:
```
MOV  #105,-(SP)        ;CODE
EMT  41
```

Global Name: GUT.

Description: The current user's UIC is returned to the user at the top of the stack. The user must clear the stack.

2.8.3.7 .SYSDV - Get Name of System Device

Macro Call: .SYSDV

Assembly Language

Expansion:
```
MOV  #106,-(SP)
EMT  41
```

Global Name: GUT.

Description: The name of the System Device in Radix-50 notation is returned to the user on top of the stack.

2.8.4 Requests to Perform Conversions

Using the EMT level 42 instruction the user can request data conversions between binary and some external form, such as decimal ASCII or Radix-50. He communicates his request by pushing the necessary parameters and an identifier code onto the stack. If a code outside the range of those currently established is specified, a fatal error (F034) will result.

A note on Radix-50 packing, follows:

Because the characters allowed within names (e.g., file names or extensions, Assembler symbols, etc.) are restricted to letters, digits, and one or two specials, it is possible to store 3 characters at a time within a single word by using the formula:

$$((C_1 \times 50_8) + C_2) \times 50_8 + C_3$$

where C_1, C_2, and C_3 are the three characters converted from their original ASCII value to the one shown in the following table:

	ASCII	RAD-50 Format
Space	40	0
A-Z	101-132	1-32
$	44	33
.	56	34
Unused		35
0-9	60-71	36-47

(The maximum value is thus $47 \times 50^2 + 47 \times 50 + 47 = 174777$)

2.8.4.1 .RADPK - Pack three ASCII characters into one Radix-50 word.

Macro Call: .RADPK ADDR

where ADDR is the address of the first byte in the 3-byte string of ASCII characters to be converted.

Assembly Language

Expansion:

```
MOV  # ADDR,-(SP)
CLR  -(SP)          ;MOVE CALL CODE ONTO STACK
EMT  42
```

Global Name: CVT.

Description: The string of 7- or 8-bit ASCII characters in three consecutive bytes starting at ADDR is converted to Radix-50 packed ASCII using the algorithm in Section 2.8.4. The packed value is returned on the top of the stack, followed by the address of the byte following the last character converted.

Rules: ADDR may be set at any byte address (need not be at word boundary).

The stack must be cleared by the user after the Monitor returns control.

Errors: The conversion will be stopped if an error condition is encountered, and the user will be informed of the type of error via the condition codes in the Processor Status register:

> C-bit set means that an ASCII byte outside the valid Radix-50 set was encountered.

The value returned will be left-justified and correct up to the last valid byte, e.g. DT: = DT : The address returned will be that of the first invalid byte.

If no errors were encountered during the conversion, the condition codes will be cleared.

Example: Pack a string of 30_{10} ASCII characters, starting at UNPBUF, into a buffer starting at PAKBUF.

```
        MOV     #PAKBUF,R3          ;SET UP POINTER TO PACK-BUFFER
        MOV     #UNPBUF,-(SP)       ; .RADPK UNBUF
NEXT:   CLR     -(SP)
        EMT     42
        BCS     ERRC                ;INVLID ASCII CODE ENCOUNTERED
        MOV     (SP) +, (R3) +      ;MOV PACKED VALUE TO BUFFER
        CMP     R3, #PAKBUF+12      ;END OF STRING?
        BNE     NEXT                ;NO
        TST     (SP) +              ;YES - REMOVE POINTER FROM STACK
```

Note that this example takes advantage of the fact that the Monitor returns to the stack the address of the byte which follows the last character converted.

2.8.4.2 .RADUP - Unpack one Radix-50 word into three ASCII characters.

Macro Call: .RADUP ADDR,WORD

where ADDR is the pointer to the buffer into which the unpacked bytes are to be placed, and WORD is the Radix-50 word to be converted.

Assembly Language

Expansion:
```
MOV  WORD,-(SP)
MOV #ADDR,-(SP)
MOV #1,-(SP)          ;MOVE CALL CODE ONTO STACK
EMT 42
```

Global Name: CVT.

Description: WORD is converted into a string of 7-bit ASCII characters which are placed left-justified with trailing spaces in three consecutive bytes starting at location ADDR. The stack is returned cleared.

Errors: If an error is encountered, the user will be informed via the condition codes in the Processor Status register:

C-bit set means that (a) a value of WORD was outside the valid Radix-50 set, i.e., 174777, (see Section 2.8.4); (b) a Radix-50 byte value was found to be 35, which is currently not used.

Nevertheless, three bytes will be returned, with a : as the first of the three for error type (a), and a / for any of the three bytes for error type (b).

If the conversion is satisfactory, the condition codes are cleared.

2.8.4.3 .D2BIN - Convert five decimal ASCII characters into one binary word.

Macro Call: .D2BIN ADDR

where ADDR is the address of the first byte in the 5-byte string of decimal characters to be converted (can be on byte- or word-boundary).

Assembly Language

Expansion:
```
MOV  #ADDR,-(SP)
MOV  #2,-(SP)          ;MOVE CALL CODE ONTO STACK
EMT  42
```

Global Name: CVT.

Description: The 5-byte string of 7- or 8-bit ASCII characters which start at ADDR are converted into their binary equivalent. The converted value is returned to the top of the stack, right-justified, followed by the address of the byte which follows the last character converted. The largest decimal number that can be converted is 65,535 (2^{16}-1). The user must clear the stack.

Errors: The conversion will be stopped if an error condition is encountered. The user will be informed of the type of error via the condition codes in the Processor Status register:

C-bit set means that a byte was not a digit.
V-bit set means that the decimal number was too large, i.e. greater than 65535.

The value returned will be correct up to the last valid byte. The address returned will be that of the invalid byte. If the conversion is satisfactory, the condition codes will be cleared.

2.8.4.4 .BIN2D - Convert one binary word into five decimal ASCII characters.

Macro Call: .BIN2D ADDR,WORD

where WORD is the number to be converted, and ADDR is the address of the first byte of the buffer where the characters are to be placed.

Assembly Language

Expansion:

```
MOV  WORD,-(SP)
MOV  #ADDR,-(SP)
MOV  #3,-(SP)          ;MOVE CALL CODE ONTO STACK
EMT  42
```

Global Name: CVT.

Description: WORD is converted into a string of five decimal 7-bit ASCII characters which are placed into consecutive bytes starting at location ADDR. They are right-justified with leading zeros. The stack is cleared.

Errors: No errors are possible.

2.8.4.5 O2BIN - Convert six octal ASCII characters into one binary word.

Macro Call: .O2BIN ADDR

where ADDR is the address of the first byte in the 6-byte string of octal characters to be converted.

Assembly Language

Expansion:
```
MOV  #ADDR,-(SP)
MOV  #4,-(SP)          ;MOVE CALL CODE ONTO STACK
EMT  42
```

Global Name: CVT.

Description: The 6-byte string of octal 7- or 8-bit ASCII characters which start at ADDR are converted into the binary number equivalent. The converted value is returned to the top of the stack, right-justified, followed by the address of the byte which follows the last character converted. The largest octal number which can be converted is 177777. The stack must be cleared by the user.

Errors: The conversion will be stopped if an error condition is encountered, and the user will be informed of the type of error via the condition codes in the Processor Status register:

C-bit set means that a byte was not a digit.

V-bit set means that the octal number was too large, i.e., the first byte of six was greater than 1.

If the conversion has been satisfactory, the condition codes are cleared. Following C- or V-bit errors the value returned will be correct up to the last valid byte. The address returned will be that of the first invalid byte.

2.8.4.6 .BIN2O - Convert one binary word into six octal ASCII characters.

Macro Call: .BIN2O ADDR,WORD

where WORD is the binary number to be converted, and ADDR is the address of the buffer into which the six octal ASCII characters are to be placed.

Assembly Language

Expansion:
```
MOV  WORD ,-(SP)
MOV  #ADDR,-(SP)
MOV  #5,-(SP)
EMT  42
```

Global Name: CVT.

Description: The WORD is converted into a 6-byte string of octal 7-bit ASCII characters, right-justified with leading zeros, which are placed into the buffer addressed by ADDR. The stack is cleared.

Errors: No errors are possible.

2.8.5 Requests for Interfacing with the Command String Interpreter

A user program may obtain dataset specifications via keyboard input at run time by calling the Command String Interpreter (CSI) routine. This is the same routine used by many system programs; it accepts keyboard input at program run time in the format presented in Section 3.4.1.

The CSI is called in two parts, by two different requests: .CSI1 and .CSI2. .CSI1 condenses the command string and checks for syntactical errors. .CSI2 sets the appropriate Link Block and Filename Block parameters for each dataset specification in the command string. Each command string requires one .CSI1 request for the entire command string, and one .CSI2 request for each dataset specifier in the command string.

The user must first set up a line buffer in his program and read in the command string. Then he does a .CSI1, which condenses the string by eliminating spaces, horizontal TABs, nulls, and RUBOUTs, sets pointers in a table to be referenced by .CSI2, and checks the command string for syntactical errors. If there are no errors, the .CSI2 request may be given once for each dataset specification that the user expects to find in the command string. .CSI2 sets up the appropriate Link Block and Filename Block parameter according to the device name, file name, extension, UIC, and switch entries in the command string.

2.8.5.1 .CSI1 - Condense command string and check syntax.

Format: .CSI1 CMDBUF

where CMDBUF is the address of the command buffer header described under "Rules" below.

Assembly Language

Expansion:
```
MOV #CMDBUF,-(SP)
EMT 56
```

Global Name: CSX.

Description: Condenses the command string by removing spaces, horizontal TABs, nulls, and RUBOUTs, and checks the entire command string for syntactical errors. Control is returned to the user with a 0 at the top of the stack if the syntax was acceptable, or with the address (in the command string line buffer) of the data byte at which the scan terminated because the first error was encountered.

Rules: The .CSI2 request must be preceded by a .CSI1 request, because .CSI2 assumes it is getting a syntactically correct command; more than one CSI2 request can follow a single .CSI1 request.

The user must set up a line buffer and read in the command string before doing .CSI1.

It is the user's responsibility to print a # on the teleprinter to inform the operator that a CSI format is expected (Section 3.1).

The user must set up a seven-word command buffer header in his program immediately preceding the header of the line buffer into which the command is to be read. The user is not required at this time to set up anything in the command buffer header prior to calling .CSI1; it will be used as a work-and-communication area by the Monitor routines processing the .CSI1 and .CSI2 requests.

The user must clear the stack upon return from the Monitor. If the top of the stack $\neq 0$ (i.e., if there was a syntax error), .CSI2 must not be called.

Example: (See .CSI2.)

2.8.5.2 .CSI2 - Interpret one dataset specification of command string.

Format: .CSI2 CSIBLK

Assembly Language

Expansion:
```
MOV #CSIBLK,-(SP)
EMT 57
```

Global Name: CSM.

Description: Gets the next input or output dataset specification from the command string, and sets the PHYSICAL DEVICE NAME entry in the Link Block, the FILENAME, EXTENSION, and UIC entries in the Filename Block, and any switch entries in an extension of the Link Block.

Rules: Before calling .CSI2, the user must:

- Call CSI1 to condense the command string and check it for syntax errors. There must have been no syntax errors.
- Set up a CSI control block as follows:

CSIBLK:

POINTER TO CMDBUF
POINTER TO LNKBLK
POINTER TO FILBLK

where POINTER TO CMDBUF is the address of the 7-word work area preceding the command string line buffer header;

POINTER TO LNKBLK is the address of the Link Block of the dataset whose specification is being requested; and

POINTER TO FILBLK is the address of the Filename Block of the dataset whose specification is being requested (currently, CSI allows only one file per dataset specification).

- Set the first word of CMDBUF to either 0 or 2. 0 means "get next input dataset specification", and 2 means "get the next output dataset specification". CSI2 does not check the validity of the code word.
- Initialize the NUMBER OF WORDS TO FOLLOW entry in the Link Block to contain the number of words to follow. This must be at least one, because CSI2 will alter the following word, i.e., the PHYSICAL DEVICE NAME word. CSI2 does not check the validity of this byte.

  The user may specify any number from 1 to 255_{10} in this location. All words in excess of 1 are used for switch space (see the interface with respect to switches, described below).

Upon return from the .CSI2 request, the Monitor will have provided the following information:

- The top of the stack contains either:

  (a) 0, which means the dataset specification requested has been obtained, and there are still more dataset specifcations of the type requested (i.e., input or output); or

  (b) 1, which means the dataset specification requested has been obtained, and there are no further dataset specifications of the type requested; or

  (c) 2, which means (a), but this particular dataset specification included more switches than would fit in the space provided; or

  (d) 3, which means (b), but this particular dataset specification included more switches than would fit in the space provided.

- With respect to the Link Block (Figure 2-5):

  If the PHYSICAL DEVICE NAME word is zero, the user does not wish this particular output (input) dataset to be generated (read); i.e., this entry was omitted when the command string was typed in. If not zero, the PHYSICAL DEVICE NAME and UNIT NUMBER are appropriately set to the device and unit specified in the command string.

- Immediately following the PHYSICAL DEVICE NAME word in the Link Block are the switches specified in the command string. The interface for each switch is shown in the switch block below. These switch blocks are written in the area provided by the programmer in the Link Block.

NUMBER OF WORDS TO FOLLOW
POINTER TO FIRST CHARACTER OF Vn
POINTER TO FIRST CHARACTER OF Vn-1
:
POINTER TO FIRST CHARACTER OF V1
W(ASCII) S(ASCII)

If NUMBER OF WORDS TO FOLLOW is zero, there are no more switches. Note that the pointers are in reverse order. After the value pointers is an ASCII word which contains the first two characters of the switch. The first character is in the low byte, and the second is in the high byte. If the name of the switch contains only one character, the ASCII representation of that character will be in the low byte, and the high byte will contain a zero. Note that if the number of words to follow is not zero, it is the number of values +1. For example, if the switch /SWITCH:$12:AB is stored in memory beginning at location 1000 as:

1000	1001	1002	1003	1004	1005	1006
/	S	W	I	T	C	H
1007	1010	1011	1012	1013	1014	1015
:	$	1	2	:	A	B

then the completed interface appears as:

3	
1014	
1010	
127=S	123=W

- With respect to the Filename Block (Figure 2-6):

  (a) The FILE NAME occupies the two words at FILBLK and FILBLK+2. If the Monitor returns zero at FILBLK, no FILE NAME was specified in the dataset specification; if it returns 52_8 at FILBLK, * was specified as the FILE NAME. Otherwise, the Monitor returns at FILBLK and FILBLK+2 the first six characters of FILE NAME, in Radix-50 packed ASCII.

  (b) The EXTENSION occupies the word at FILBLK+4. If the Monitor returns zero at FILBLK+4, no EXTENSION was specified; if it returns 52_8, * was specified. Otherwise, the Monitor returns the first three characters of the extension specified, in Radix-50 packed ASCII.

  (c) The USER IDENTIFICATION CODE occupies the word at FILBLK+6. If the Monitor returns zero at FILBLK+6, no UIC was specified in the dataset specification (the I/O processors will assume the UIC of this user). If a UIC was typed in, the Monitor will set this word appropriately. The Monitor returns 377_8 in either high- or low-order byte of this word if * was specified.

  The user may restart at the beginning of the input dataset or output dataset side of the command string simply by recalling .CSI1 and issuing a 0 or 2 code, respectively. Note that he may not restart one without restarting the other.

Remark: There is no error checking with respect to magnitude when the UNIT or UIC values are converted from octal ASCII to binary.

2.8.6 User Program Tables

2.8.6.1 The Link Block (used for all input/output and directory requests)

LNKBLK:

ERROR RETURN ADDRESS	
000000 LINK POINTER (for Monitor use only)	
LOGICAL NAME OF DATASET -- Radix-50 Packed ASCII	
UNIT NUMBER	NUMBER OF WORDS TO FOLLOW
PHYSICAL DEVICE NAME -- Radix-50 Packed ASCII	

Figure 2-5 The Link Block

Each dataset in a user's program must have a Link Block associated with it. Entries in the Link Block which must be specified by the user can be written into his program or set by the program itself before the dataset is INITed. Each entry is explained below.

Address	Name	Function
LNKBLK-2	ERROR RETURN ADDRESS	This entry must be set by the user to contain the address where he wants control transferred in the event that any request associated with this dataset fails to obtain required buffer space from the Monitor. If no address is specified here, such an error will be treated as fatal. This address may be changed by the user's program at any time.
LNKBLK	LINK POINTER	This location must be set to zero by the user and must not be modified by him. The Monitor places a linking address here when the dataset is INITed. Before INITing a dataset, the Monitor tests this pointer for zero. If it is not zero, the Monitor assumes that the dataset was already INITed.
LNKBLK+2	LOGICAL NAME OF DATASET	The user can specify a name for the dataset in this entry. This name, which must be unique, is used to associate the dataset with a device which is specified by an ASSIGN from the keyboard. The name is stored in Radix-50 packed ASCII by the .RAD50 assembler directive. (A specification is required only when using an ASSIGN.)
LNKBLK+4	NUMBER OF WORDS TO FOLLOW	This byte contains the count of the number of words to follow in the Link Block. The user should set it to a 0 if he does not specify any PHYSICAL DEVICE NAME in the next word, or to a 1 if he does. Values greater than 1 may be used if the Command String Interpreter is to be called.
LNKBLK+5	UNIT NUMBER	This code specifies the unit number of the device linked to the dataset. For example, the TC11 Controller (DECtape) can drive up to eight tape drives (units), numbered 0-7.
LNKBLK+6	PHYSICAL DEVICE NAME	If the user specified 1 or greater LNKBLK+4, he must specify here the standard name (Appendix A) for the device associated with the dataset. If no name is specified here, the user must specify LOGICAL NAME OF DATASET and perform an ASsign command before he runs his program.

2.8.6.2 The Filename Block - Each file associated with a dataset must be described by the user in a Filename Block. If a dataset is not a file, the Filename Block must still be used, but FILENAME, EXTENSION, and PROTECT need not be specified. The Filename Block is used by OPEN and all directory management requests.

FILBLK:

ERROR RETURN ADDRESS	
ERROR CODE	HOW OPEN
FILE NAME	
FILE NAME	
EXTENSION	
USER ID CODE	
(spare)	PROTECT CODE

Figure 2-6 The Filename Block

Address	Name	Function
FILBLK-4	ERROR RETURN ADDRESS	The user must specify here the address to which he wants the Monitor to return control if one of the errors in Table 2-4 occurs during an operation involving the file. If no address is specified here, any such error will be treated as a fatal error.

Table 2-4
Filename Block Error Conditions

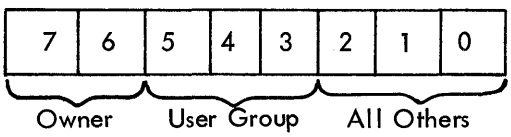
Error Code In File-name Block	Faulting Request	Cause	Remedy
00	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to open a dataset that was previously opened.	
01		unused	
02	.OPENO	An attempt was made to .OPENO a file which already exists.	Delete the file (with PIP) or change file name.
	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file for input, extension, or update which is currently opened for output, or which does not exist.	

Table 2-4 (Cont)
Filename Block Error Conditions

Error Code In File-name Block	Faulting Request	Cause	Remedy
03	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file which has already been opened the maximum number of times (76_8).	Close file.
04	.OPENC .OPENE .OPENU	An .OPENC, .OPENE, or .OPENU attempt was made to open a file which has already been opened for either .OPENC, .OPENE, or .OPENU.	.CLOSE the previous open.
05	.OPENE	Illegal request to a contiguous file.	
06	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to access a file which the protection code prohibits.	
07		unused	
10	.OPENC	Illegal OPEN request to a contiguous file.	
11	.OPENC .OPENE .OPENO .OPENU	File opened for output or extension is already on current DECtape unit.	Close offending file.
12	.ALLOC .OPENO	Directory full (DT).	Mount another DEC-tape.
13	.ALLOC .OPENO	The UIC was not entered into the device MFD.	Enter UIC via PIP.
14	.APPND .DELET .RENAM	An attempt was made to perform an illegal operation on an opened file.	Wait until file is closed.
15	.ALLOC .OPENO	An attempt was made to create a file with an illegal file name	Change file name.

Address	Name	Function
FILBLK-2	HOW OPEN	This is set when the .OPENx macro's assembly language expansion is executed. It tells the Monitor which kind of open is being requested: .OPENU = 1, .OPENO = 2, .OPENE = 3, .OPENI = 4, .OPENC = 13.
FILBLK-1	ERROR CODE	This entry should not be set by the user. It will be set by the Monitor to indicate the type of error (Table 2-4) which occurred. It will be cleared of any previous condition at each .OPEN call.
FILBLK+0 FILBLK+2	FILE NAME	This two-word entry must be specified by the user if this dataset, or portion thereof, is a file. It is the name of the file, in Radix-50 packed ASCII.
FILBLK+4	EXTENSION	This entry must be specified if the file named in the previous entry has an extension. It is Radix-50 packed ASCII.
FILBLK+6	USER I.D. CODE	The user may enter his USER ID CODE here in octal: GROUP NUMBER | USER'S NUMBER High-Order Byte Low-Order Byte If no entry is specified here, the current user's UIC is assumed.
FILBLK+10	PROTECT CODE	The user may specify here the protection to be given to the file at its creation or renaming (see following paragraph). If 0, a default protection 233 will be allotted.

2.8.6.3 The File Protection Codes



Owner: Bit 6 = 1 = Owner cannot write on or delete the file. This is a safeguard to prevent inadvertent deletion or over-writing.

Bit 7 = 1 = Protect the file from automatic deletion on FInish.

Figure 2-7 File Protection Codes

(continued on next page)

User Group and All Others:

Code	Function: Delete	Write	Read	Run
0	yes	yes	yes	yes
1		yes	yes	yes
2 or 3			yes	yes
4 or 5				yes
6 or 7				

Note: yes indicates that the operation is allowed. For example, if a file belongs to user [23,10], a protection code of 3 will allow user [12,4] to read or run but not delete or write on it.

Figure 2-7 File Protection Codes

2.8.6.4 The Line Buffer Header - (used by READ and WRITE requests)

BUFHDR:

MAXIMUM BYTE COUNT	
STATUS	MODE
ACTUAL BYTE COUNT	
POINTER (Dump Mode only)	

Figure 2-8 Line Buffer Header

Each element of the line buffer header table is as follows:

Address	Name	Function
BUFHDR	MAXIMUM BYTE COUNT	The count shows the size of the buffer, in bytes. It must be specified here by the user on all INPUT operations.
BUFHDR+2	MODE	The user specifies here the mode of the transfer. All modes are listed and explained in Figure 2-10.
BUFHDR+3	STATUS	The Monitor will place in this byte the status of the transfer when control is returned to the user. Figure 2-9 lists each bit and its meaning. Errors encountered executing an I/O transfer will be flagged in this byte. The user should always check its content after each transfer completes.
BUFHDR+4	ACTUAL BYTE COUNT	This count controls the number of bytes to be transferred on OUTPUT. It must be initialized by the user before any output transfer from the line buffer. After any transfer in or out, it will show how many bytes have been transmitted (or in some modes, see Section 2.8.6.6, would have been transferred had some error not been detected).

Address	Name	Function
BUFHDR+6	POINTER (dump mode)	If bit 2 of MODE is 1, the user specifies here the starting address of the line buffer. If bit 2 of MODE is 0, the line buffer header is only three words in length, and must immediately precede the line buffer itself. (Section 2.8.6.6 Note 9.)
		Note: The Monitor will return control to the program if a device transfer is needed to satisfy a READ or WRITE request. During this time, the header words will be used to store data relevant to the operation underway. The user should not, therefore, attempt to change this content until it is evident that the transfer has been completely effected, e.g., after a .WAIT return.

2.8.6.5 The Status Byte

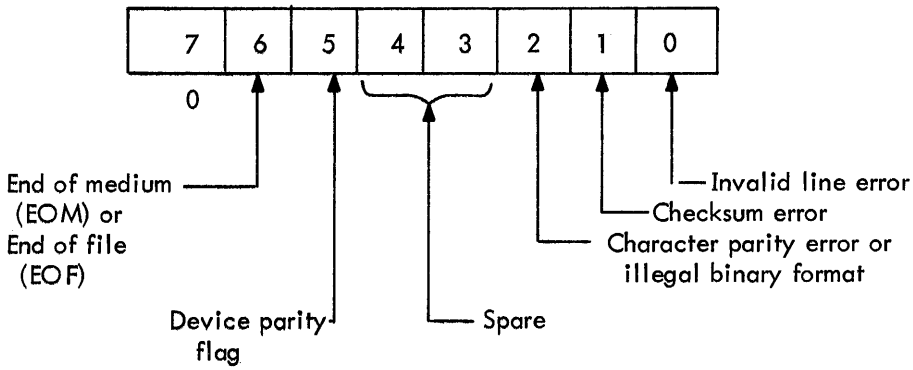


Figure 2-9 Status Format

The function of each status format bit is explained below.

Bit	Mode	Request	Condition
	ALL	.READ/WRITE	Appropriate BYTE COUNT = 0 at call.
0 (INVALID LINE)	FORMATTED ASCII NORMAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT ran out before a line terminator was seen. (Last byte has been overlaid until the terminator has been reached.)
		.WRITE	The last byte was not a terminator.
	FORMATTED ASCII SPECIAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT was reached before a line terminator was seen (excess data has not yet been read).
		.WRITE	The ACTUAL BYTE COUNT was reached before any terminator was seen.

Bit	Mode	Request	Condition
	FORMATTED BINARY NORMAL	.READ	The MAXIMUM BYTE ran out before the count stored with the data. (The last byte has been overlaid in order to verify the checksum.)
	FORMATTED BINARY SPECIAL	.READ	The MAXIMUM BYTE COUNT was reached before the count stored with the data. (The excess data still remains to be read and checksum has not been verified.)
1 (CHECKSUM ERROR)	FORMATTED BINARY	.READ	There was a discrepancy between the checksum accumulated during the .READ, and that stored with the incoming data.
2 (PARITY FORMAT)	FORMATTED ASCII PARITY NORMAL OR SPECIAL	.READ	A character was read which had odd parity. The eighth bit of the illegal character delivered is set to a 1.
2 (ILLEGAL BINARY FORMAT)	FORMATTED BINARY	.READ	This bit is set if a line processed in a binary mode does not have a 001 in the first word.
6 (EOM/EOF)	ALL MODES	.READ or .WRITE	An input device cannot supply any more data or an output device cannot accommodate more, i.e., the disk has no more storage space, or the paper tape reader has run out of paper tape.
5 (DEVICE PARITY)	ALL MODES	.READ or .WRITE	A hardware error has been detected on a bulk storage device. This could be either a parity error or a timing error. The driver will already have tried to READ or WRITE 8 or 9 times before setting this bit. (This flag is a warning that the data in this line or some subsequent line still using data from the same device block may be invalid. It will be returned for each transfer call using the same block.)

2.8.6.6 The Transfer Modes

1. Formatted ASCII Normal - Data in this mode is assumed by the Monitor to be in strings of 7-bit ASCII characters terminated by LINE FEED, FORM FEED, or VERTICAL TAB.

   READ: The line buffer is filled until either a terminator is seen or the number of bytes transferred becomes equal to the MAXIMUM BYTE COUNT. If the MAXIMUM BYTE COUNT is reached before the terminator is seen, the invalid line error bit in the Status Register of the buffer header is set, and each remaining character through to the terminator is read into the last byte of the line buffer, i.e., the surplus bytes are overlayed. After

the transfer, the actual byte count equals the number of bytes read (including the excess). RUBOUTs and NULLs are discarded. The terminator is transferred.

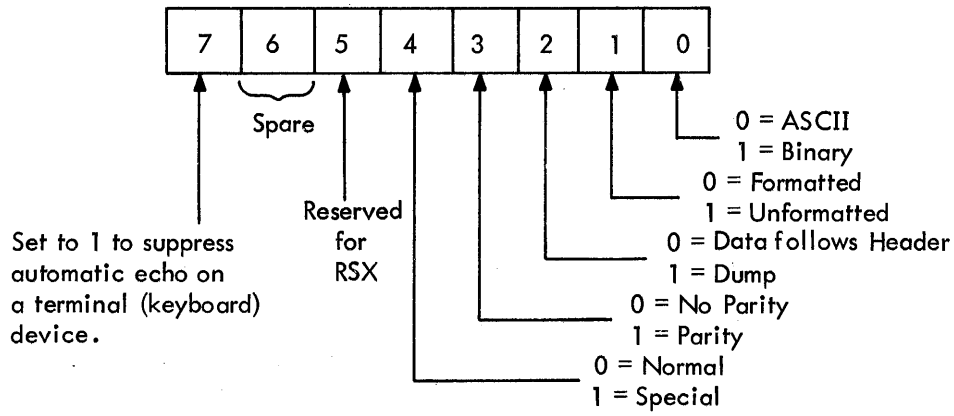


Figure 2-10 The Mode Byte

WRITE: The line buffer is output until the number of bytes transferred equals the ACTUAL BYTE COUNT. If the last character is not a terminator, an invalid line error bit is set in the STATUS BYTE of the buffer header. Previous terminators are output as normal characters.

TABs are followed by RUBOUTs; FORM FEEDs are followed by NULLs.

The READ/WRITE processor passes data to the device driver specified, and each driver will convert the information to meet its specific needs. Appendix G summarizes the characteristics of the device drivers.

2. Formatted ASCII Special -

READ: The same as formatted ASCII normal with this exception: if the MAXIMUM BYTE COUNT is reached before the terminator, the transfer is stopped. The remaining characters are not overlaid, but are retained for transfer at the next .READ. An invalid line error will be returned in the STATUS BYTE, and ACTUAL BYTE COUNT will equal MAXIMUM.

WRITE: The same as formatted ASCII normal with this exception: the line buffer is output until the first terminator; the ACTUAL BYTE COUNT will stop the transfer if it is reached before the terminator is seen. In this case, the invalid line error bit is set into the STATUS BYTE. Note that in this mode only one line of data can be output at once, but its byte count need not be exact, provided this is greater than the actual.

3. Formatted Binary Normal -

READ: This is an 8-bit transfer. Words 2 and 4, STATUS, MODE, and ACTUAL BYTE COUNT always accompany the data during formatted binary transfers. The counts are adjusted by the Monitor to include the extra words. On input, the line buffer is filled until the number of characters transferred equals the ACTUAL BYTE

READ (Cont)

COUNT read, or the MAXIMUM BYTE COUNT. If the MAXIMUM is reached before the ACTUAL, an invalid line error occurs and the remaining bytes are overlaid into the last byte until the checksum is verified. After the transfer, the ACTUAL BYTE COUNT contains the actual number of bytes read (including the excess).

WRITE: This is an 8-bit transfer. Words 2 and 4 of the line buffer are output until the number of characters transferred equal the ACTUAL BYTE COUNT and a checksum is calculated. The checksum is output at the end.

4. Formatted Binary Special -

READ: The line buffer is filled until the number of characters transferred equals the ACTUAL BYTE COUNT read. If the MAXIMUM COUNT is reached before the ACTUAL, the remainder of the line is retained by the Monitor. The MAXIMUM number is transferred to the line buffer and the ACTUAL BYTE COUNT is set to the full input count, rather than to the number of bytes actually transferred. The invalid line error will be set in the STATUS BYTE. The user can compare the MAXIMAL COUNT with the ACTUAL, determine how much data remains, and recover it by an unformatted binary read (allowing 1 extra byte for the checksum).

WRITE: Identical to formatted binary normal.

5. Unformatted ASCII Normal or Special - This mode is available to the user who wants to do his own formatting. Seven bits are transferred; the eighth is always set to zero. NULLs are discarded.

READ: Transfer stops when the number of bytes transferred reaches the MAXIMUM BYTE COUNT. Nulls are discarded but all other characters are treated as valid.

WRITE: All characters are transferred. The transfer stops when the ACTUAL BYTE COUNT is reached.

6. Unformatted Binary Normal or Special - This mode is identical to unformatted ASCII except that eight bits are transferred on both input and output. No checksum is calculated.

7. Formatted ASCII Parity - Identical to formatted ASCII (Special or Normal) except that even parity is generated in the eighth bit on OUTPUT; during INPUT it will be checked. Valid characters will be passed to the user as 7 bits; invalid characters will be marked by bit 8 = 1, and will cause the setting of the parity error bit in the STATUS BYTE (11).

8. Unformatted ASCII Parity - Identical to unformatted ASCII (Special or Normal) except that eight bits are transferred instead of seven. No parity generating or checking is performed.

9. Dump Modes - All modes can be specified as DUMP, which means that the word after the ACTUAL BYTE COUNT is considered to be a pointer to the beginning of the data rather than the beginning of the data proper. (Section 2.8.6.4.)

2.8.6.7 The BLOCK Block - (used by BLOCK request only)

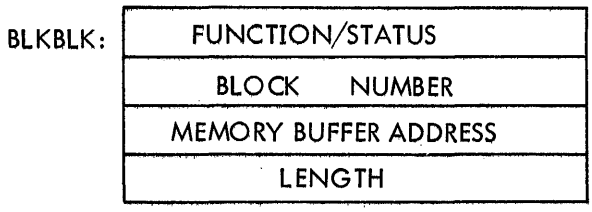


Figure 2-11 The BLOCK Block

Address	Name	Function
BLKBLK	FUNCTION/STATUS	User specifies here the function to be performed, and the Monitor returns to the user with the appropriate status bits set. Bit — Bit = 1 means: function 0: function is GET function 1: function is OUTPUT function 2: function is INPUT (3-8): spares (ignored by Monitor) status 9: illegal function status 10: file is linked, or device is not file structured status 11: block number does not exist in file, i.e., it is greater than the file length. status 12: file not open status 13: protect code violation status 14: end of data error status 15: device parity error
BLKBLK+2	BLOCK NUMBER	Requested block number to be transferred relative to the beginning of the file. First block of file is 0.
BLKBLK+4	Memory Buffer Address	The address and length of the Monitor buffer given by the Monitor on an INPUT or GET function.
BLKBLK+6	Length	

2.8.6.8 The TRAN Block (used by TRAN request only)

TRNBLK:	DEVICE BLOCK NUMBER
	MEMORY START ADDRESS
	WORD COUNT
	FUNCTION/STATUS
	NUMBER OF WORDS NOT TRANSFERRED

Figure 2-12 The TRAN Block

The user must set up a TRAN block for each .TRAN in his program.

Address	Name	Function
TRNBLK	DEVICE BLOCK NUMBER	User specifies here the absolute block number of the device, at which the transfer is to begin. If it is not a bulk storage device, specify block 0.
TRNBLK+2	MEMORY START ADDRESS	User specifies here the core memory address at which the dataset transfer is to begin.
TRNBLK+4	WORD COUNT	User specifies here the total number of 16-bit words to be transferred. Word count need not be block size.
TRNBLK+6	FUNCTION/STATUS	Bit: Bit = 1 means: 0 Binary, rather than ASCII* 1 Write = 1* 2 Read = 1* 3, 4, 5, 6, 7, 8, 9, 10 Reserved for Monitor's use 11 DECtape direction* (0 = forward, 1 = reverse) 12 spare 13 invalid call (improper function/no word count) 14 end of medium** 15 recoverable device** error such as parity or timing.
TRNBLK+10	NUMBER OF WORDS NOT TRANSFERRED	User leaves this entry blank. If an EOM occurs during the transfer, the Monitor will place in this entry the number of words not transferred.

*Must be specified by user.

**These bits are cleared before TRAN is carried out.

2.8.6.9 The Special Functions Block (used by SPEC request only)

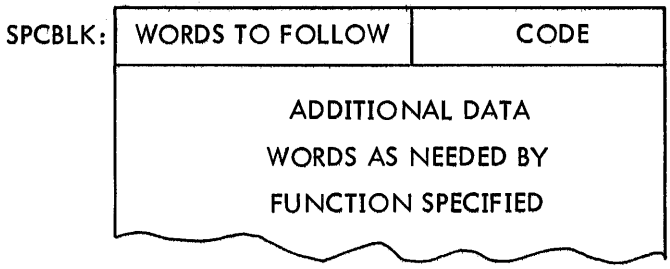


Figure 2-13

Where a special function requires supporting data the user must set up a Special Functions Block in his program.

Address	Name	Function
SPCBLK	CODE	The user identifies the function here by inserting the appropriate code in the range $0\text{-}255_{10}$.
SPCBLK+1	WORDS TO FOLLOW	The size of each Special Functions Block is dependent upon the Function. The user shows here how many more words belong to the particular block.
SPCBLK+2 ⋮	--	The user places in these words data to be passed to the function processor or the function processor will return here such items as status information etc. The format in each case is determined by the function.

2.9 PROGRAMMING TIPS

Swapping time can be kept to a minimum by placing like requests together in the coding. For example, method 1, below, will require the .INIT and the .OPEN processors to be swapped in only once each. However, method 2 requires that each be swapped in three times. The exception of course occurs if either are made core resident.

Method 1	Method 2
.INIT A	.INIT A
.INIT B	.OPEN A
.INIT C	.INIT B
⋮	.OPENO B
.OPENI A	⋮
.OPENO B	.INIT C
.OPENO C	⋮
	.OPENO C

Core can be used more efficiently if datasets which are to be used the longest (i.e., .RLSEd last) are .INITed first. Such action is efficient because free core is allocated from the bottom, and if the more

permanent routines are allocated first (i.e., at the bottom), larger areas of free core will become available as less permanent routines are released from the top. Thus, method 1 below is potentially more efficient than method 2.

Method 1	Method 2
.INIT C	.INIT A
.	.INIT B
.	.INIT C
.INIT B	.
.	.
.	.RLSE A
.INIT A	.RLSE B
.	.
.	.
.RLSE A	.RLSE C
.RLSE B	
.	
.	
.RLSE C	

.READ and .WRITE were designed to be used for sequential access to a linked file, but are legal for both linked and contiguous files.

Since .EXIT will cause the user's program to be effectively wiped out, if the programmer wishes his program to remain in core after it has finished (e.g., for debugging or for immediate reuse), he might, instead of .EXIT, use something like:

```
BR  .   or   LOC: BR LOC
```

The operator can then specify the next action by recalling the Monitor via a command at the keyboard (see Section 3.2).

In some cases the WAIT or WAITR instructions are not needed. This situation is called an implied WAIT, and occurs because the Monitor will only process one action on a dataset at a time. For example, if a program is written:

```
.READ  LNK1,BUF1
        .
        .
.READ  LNK1,BUF2
```

the second READ becomes an implied WAIT for the first, since the Monitor will not start the second until the first is finished with the dataset. This implies that when control returns to the user after the second READ, he may safely assume the data transferred by the first can now be processed. Similarly, if two different datasets reference one device in common, action on the second dataset will not proceed until action on the first is complete.

2.10 MONITOR MESSAGES

Monitor messages are typed on the teleprinter in the following format:

CNNN XXXXXX

where C is one of five letters identifying the type of message:

I	Informational
A	Action required by the operator
W	Warning to the operator
F	Fatal error
S	System program error

where NNNN is the message number, and XXXXXX gives appropriate additional information. Informational, Warning, and System program messages are printed and the program continues.

Action messages are printed and the program is suspended. The Monitor expects the operator to take some action such as "continue the program" (type COntinue), or "kill the program" (type KILL).

Fatal error messages are printed if possible, and the program is suspended. The Monitor will not allow the operator to continue the program, but expects to see either a BEgin, REstart or KILL command. If a fatal error is a system disk failure and the error message cannot be printed, the central processor halts. This is the only time that a halt occurs in the Monitor.

If the error has been caused by a stack overflow, the stack pointer is reset before the message is printed. All Monitor and system program error messages are summarized in Appendix F.

2.11 EXAMPLE PROGRAMS

The following are assembled listings of two simple programs written in and assembled using PAL-11R. The programs contain many of the Monitor's programmed requests.

Example Program #1

```
                        ;PROGRAM WHICH TYPES A MESSAGE ON THE TELETPE WHILE
                        ;ACCEPTING A MESSAGE FROM THE KEYBOARD. PROGRAM REPEATS

        000000          R0=%0
        000001          R1=%1
        000002          R2=%2
        000003          R3=%3
        000004          R4=%4
        000005          R5=%5
        000006          SP=%6
        000007          PC=%7
        000015          CR=15
        000012          LF=12
        000011          HT=11
        000107          EROR=107

000000 012746'BEGIN:    MOV #LNK1,-(SP) ;INIT LNK1
       000312
000004 104006           EMT 6
000006 012746'          MOV #LNK2,-(SP) ;INIT LNK2
       000324
000012 104006           EMT 6
000014 012746'          MOV #FIL1,-(SP) ;OPEN FOR OUTPUT
       000340
000020 012746'          MOV #LNK1,-(SP)
       000312
000024 104015           EMT 15
000026 012746'          MOV #FIL2,-(SP) ;OPEN FOR INPUT
       000356
000032 012746'          MOV #LNK2,-(SP)
       000324
000036 104016           EMT 16
000040 012746'          MOV #MSG1,-(SP) ;WRITE THE MESSAGE
       000370
000044 012746'          MOV #LNK1,-(SP)
       000312
000050 104002           EMT 2
000052 012700'          MOV #LIB1+6,R0  ;SET THE BUFFER POINTER
       000170
000056 005020 LOOP1:    CLR (R0)+       ;CLEAR THE ADDRESS AND INCREMENT
000060 020027'          CMP R0,#LIB1+80.        ;END OF BUFFER?
       000302
000064 103774           BLO LOOP1       ;NO, GO BACK & CONTINUE CLEARING
000066 012746'          MOV #LNK1,-(SP) ;YES,CONTINUE
       000312
000072 104001           EMT 1
000074 012746'          MOV #LIB1,-(SP) ;NO,READ LNK2,LIB1
       000162
000100 012746'          MOV #LNK2,-(SP)
       000324
000104 104004           EMT 4
000106 012746'          MOV #LNK2,-(SP) ;WAIT
       000324
```

```
000112 104001          EMT 1
000114 132767          BITB #EROR,LIB1+3       ;ANY ERRORS?
       000107
       000043
000122 001016          BNE ERR3                ;YES,GO TO THE ERROR#3 ADDRESS
000124 012746'         MOV #LNK1,-(SP) ;NO, .CLOSE LNK1
       000312
000130 104017          EMT 17
000132 012746'         MOV #LNK2,-(SP) ;.CLOSE LNK2
       000324
000136 104017          EMT 17
000140 012746'         MOV #LNK1,-(SP) ;.RLSE LNK1
       000312
000144 104007          EMT 7
000146 012746'         MOV #LNK2,-(SP) ;.RLSE LNK2
       000324
000152 104007          EMT 7
000154 000167          JMP BEGIN
       177620


               ERR1:
               ERR2:
               ERR3:
000160 104060          EMT 60                  ; EXIT ON ANY ERROR


000162 000120 LIB1:    .WORD 80.               ;MAX BYTE COUNT
000164    000          .BYTE 0,0               ;FORMATTED ASCII
000165    000
000166 000000          .WORD 0                 ;ACTUAL BYTE COUNT
       000310          .=.+80. ;RESERVE THE BUFFER SPACE


000310 000160'         .WORD ERR1              ;ERROR RETURN ADDRESS
000312 000000 LNK1:    .WORD 0 ;POINTER
000314 016027          .RAD50 /DS1/            ;LOGICAL NAME
000316    001          .BYTE 1,0               ;UNIT 0
000317    000
000320 042420          .RAD50 /KB/             ;KEYBOARD


000322 000160'         .WORD ERR2              ;ERROR RETURN ADDRESS
000324 000000 LNK2:    .WORD 0
000326 016030          .RAD50 /DS2/
000330    001          .BYTE 1,0
000331    000
000332 042420          .RAD50 /KB/             ;KEYBOARD

000334 000000          .WORD 0 ;GO TO FATAL ERROR MESSAGE
000336    002          .BYTE 2,0               ;OPEN FOR OUTPUT
000337    000
000340 000000 FIL1:    .WORD 0,0,0,0,0 ;NO NAME, EXT, UIC, OR PROTECT
000342 000000
000344 000000
```

```
000346 000000
000350 000000

000352 000000          .WORD 0 ;GO TO FATAL ERROR
000354     004          .BYTE 4,0        ;OPEN FOR INPUT
000355     000
000356 000000 FIL2:    .WORD 0,0,0,0,0 ;NO NAME, EXT, UIC, OR PROTECT
000360 000000
000362 000000
000364 000000
000366 000000


000370 000210 MSG1:    .WORD 210        ;MAX BYTE COUNTS
000372     000          .BYTE 0,0        ;FORMATTED ASCII
000373     000
000374 000205          .WORD MSGEND-MSG1-6      ;ACTUAL BYTE COUNT
000376     015          .BYTE CR,LF,HT
000377     012
000400     011
000401     040          .ASCII / SPEAK ROUGHLY TO YOUR LITTLE BOY /
000402     123
000403     120
000404     105
000405     101
000406     113
000407     040
000410     122
000411     117
000412     125
000413     107
000414     110
000415     114
000416     131
000417     040
000420     124
000421     117
000422     040
000423     131
000424     117
000425     125
000426     122
000427     040
000430     114
000431     111
000432     124
000433     124
000434     114
000435     105
000436     040
000437     102
000440     117
000441     131
000442     040
000443     015          .BYTE CR,LF,HT,
000444     012
```

```
000445    011
000446    000
000447    040          .ASCII / AND BEAT HIM WHEN HE SNEEZES /
000450    101
000451    116
000452    104
000453    040
000454    102
000455    105
000456    101
000457    124
000460    040
000461    110
000462    111
000463    115
000464    040
000465    127
000466    110
000467    105
000470    116
000471    040
000472    110
000473    105
000474    040
000475    123
000476    116
000477    105
000500    105
000501    132
000502    105
000503    123
000504    040
000505    015          .BYTE CR,LF,HT
000506    012
000507    011
000510    040          .ASCII / HE ONLY DOES IT TO ANNOY /
000511    110
000512    105
000513    040
000514    117
000515    116
000516    114
000517    131
000520    040
000521    104
000522    117
000523    105
000524    123
000525    040
000526    111
000527    124
000530    040
000531    124
000532    117
000533    040
000534    101
```

```
000535     116
000536     116
000537     117
000540     131
000541     040
000542     015          .BYTE CR,LF,HT
000543     012
000544     011
000545     040          .ASCII / BECAUSE HE KNOWS IT TEASES /
000546     102
000547     105
000550     103
000551     101
000552     125
000553     123
000554     105
000555     040
000556     110
000557     105
000560     040
000561     113
000562     116
000563     117
000564     127
000565     123
000566     040
000567     111
000570     124
000571     040
000572     124
000573     105
000574     101
000575     123
000576     105
000577     123
000600     040
000601     015          .BYTE CR,LF
000602     012
        000603 MSGEND=.
        000604          .EVEN

        000001          .END
```

```
BEGIN     000000R     CR      = 000015     EROR    = 000107
ERR1      000160R     ERR2      000160R    ERR3      000160R
FIL1      000340R     FIL2      000356R    HT      = 000011
LF      = 000012      LIB1      000162R    LNK1      000312R
LNK2      000324R     LOOP1     000056R    MSGEND  = 000603R
MSG1      000370R     PC      =%000007     R0      =%000000
R1      =%000001      R2      =%000002     R3      =%000003
R4      =%000004      R5      =%000005     SP      =%000006
.       = 000604R
```

Example Program #2

```
                ; PROGRAM TO DUPLICATE A PAPER TAPE
                ; USING TRAN-LEVEL REQUESTS
                ;
        000000          R0=%0
        000006          SP=%6
        000007          PC=%7
        000015          CR=15
        000012          LF=12
        000011          HT=11
        000004          RD=04           ;TRANBLOCK FUNCTION CODE FOR .READ
        000002          WR=02           ;TRANBLOCK FUNCTION CODE FOR .WRITE
        000107          G=107           ;ASCII G
        040000          EOD=40000       ;TRANBLOCK FUNCTION/STATUS=EOD
        000107          EROR=107
000000 012746'BEGIN:    MOV #LNK1,-(SP)     ;.INIT LNK1
        000416
000004 104006           EMT 6
000006 012746'          MOV #LNK2,-(SP)     ;.INIT LNK2
        000430
000012 104006           EMT 6
000014 012746'          MOV #LNK3,-(SP)     ;INIT LNK3
        000346
000020 104006           EMT 6
000022 012746'          MOV #LNK4,-(SP)     ;.INIT LNK4
        000372
000026 104006           EMT 6
000030 005067 START:    CLR FLAG1               ;ZERO END FLAG
        000210
000034 012767    MOV #100.,BLK1+4     ;INITIALIZE BUFFER SIZE
        000144
        000344
000042 005067           CLR BUF1+6       ;INITIALIZE INPUT BUFFER
        000316
000046 005067           CLR BUF1+10      ;INITIALIZE INPUT BUFFER
        000314
000052 012746'          MOV #MSG1,-(SP)  ;.WRITE LNK3,MSG1
        000246
000056 012746'          MOV #LNK3, -(SP)     ;
        000346
000062 104002           EMT 2
000064 012746'          MOV #LNK3,-(SP)     ;.WAIT LNK3
        000346
000070 104001           EMT 1
000072 012746'          MOV #BUF1,-(SP)     ;.READ LNK4,BUF1
        000356
000076 012746'          MOV #LNK4,-(SP)
        000372
000102 104004           EMT 4
000104 012746'          MOV #LNK4,-(SP)     ;.WAIT LNK4
        000372
000110 104001           EMT 1
000112 132767           BITB #EROR,BUF1+3
        000107
        000241
```

```
000120 001050          BNE ERR6
000122 122767          CMPB #G,BUF1+6        ;G?
       000107
       000234
000130 001337          BNE START             ;NO
000132 112767 LOOPR:   MOVB #RD,BLK1+6       ;YES,SET UP READ
       000004
       000250
000140 012746'         MOV #BLK1,-(SP)       ;.TRAN LNK1,BLK1
       000402
000144 012746'         MOV #LNK1,-(SP)
       000416
000150 104010          EMT 10
000152 012746'         MOV #LNK1,-(SP)       ;.WAIT LNK1
       000416
000156 104001          EMT 1
000160 032767          BIT #EOD,BLK1+6       ;TEST FUNCTION FOR EOD
       040000
       000222
000166 001406          BEQ LOOPW
000170 166767 ENDM:    SUB BLK1+10,BLK1+4    ;RESET WORDCOUNT TO FINAL
       000216
       000210
                                            ;   BUFFER'S SIZE
000176 012767          MOV #1,FLAG1          ;SET EOD=FLAG
       000001
       000040
000204 112767 LOOPW:   MOVB #WR,BLK1+6       ;SET UP WRITE
       000002
       000176
000212 012746'         MOV #BLK1,-(SP)       ;.TRAN LNK2,BLK1
       000402
000216 012746'         MOV #LNK2,-(SP)
       000430
000222 104010          EMT 10
000224 012746'         MOV #LNK2,-(SP)       ;.WAIT LNK2
       000430
000230 104001          EMT 1
000232 005767          TST FLAG1             ;END OF DATA?
       000006
000236 001274          BNE START             ;YES,START OVER
000240 000734          BR LOOPR              ;NO, GET MORE
                ERR1:
                ERR2:
                ERR3:
                ERR4:
                ERR5:
                ERR6:
                ERR7:
000242 104060          EMT 60              ;EXIT ON ANY ERROR
000244 000000 FLAG1:    .WORD 0            ;1=>EOD RECEIVED ON READ
000246 000067 MSG1:   .WORD 55.
000250    000         .BYTE 0,0
000251    000
000252 000067         .WORD 55.
000254    015         .BYTE CR,LF,HT
```

```
000255     012
000256     011
000257     114          .ASCII /LOAD TAPE INTO READER/
000260     117
000261     101
000262     104
000263     040
000264     124
000265     101
000266     120
000267     105
000270     040
000271     111
000272     116
000273     124
000274     117
000275     040
000276     122
000277     105
000300     101
000301     104
000302     105
000303     122
000304     015          .BYTE CR,LF,HT
000305     012
000306     011
000307     120          .ASCII /PUSH    G, CR   WHEN READY/
000310     125
000311     123
000312     110
000313     040
000314     040
000315     040
000316     040
000317     107
000320     054
000321     040
000322     103
000323     122
000324     040
000325     040
000326     040
000327     127
000330     110
000331     105
000332     116
000333     040
000334     122
000335     105
000336     101
000337     104
000340     131
000341     015          .BYTE CR,LF
000342     012
        000344          .EVEN
000344 000242'          .WORD ERR3
```

```
000346 000000 LNK3:    .WORD 0
000350 016027          .RAD50 /DS1/
000352    001          .BYTE 1,0
000353    000
000354 042420          .RAD50 /KB/
000356 000004 BUF1:    .WORD 4
000360    000          .BYTE 0,0
000361    000
000362 000004          .WORD 4
       000370          .=.+4
       000370          .EVEN
000370 000242'         .WORD ERR4
000372 000000 LNK4:    .WORD 0
000374 016027          .RAD50 /DS1/
000376    001          .BYTE 1,0
000377    000
000400 042420          .RAD50 /KB/
000402 000000 BLK1:    .WORD 0
000404 000440'         .WORD BUF2
000406 000144          .WORD 100.
000410 000000          .WORD 0
000412 000000          .WORD 0
000414 000242'         .WORD ERR3
000416 000000 LNK1:    .WORD 0
000420 016031          .RAD50 /DS3/
000422    001          .BYTE 1,0
000423    000
000424 063320          .RAD50 /PR/
000426 000242'         .WORD ERR2
000430 000000 LNK2:    .WORD 0
000432 016032          .RAD50 /DS4/
000434    001          .BYTE 1,0
000435    000
000436 063200          .RAD50 /PP/
       000604 BUF2:    .=.+100.
       000001          .END
```

```
BEGIN     000000R     BLK1      000402R     BUF1      000356R
BUF2      000440R     CR      = 000015      ENDM      000170R
EOD     = 040000      ERDR    = 000107      ERR1      000242R
ERR2      000242R     ERR3      000242R     ERR4      000242R
ERR5      000242R     ERR6      000242R     ERR7      000242R
FLAG1     000244R     G       = 000107      HT      = 000011
LF      = 000012      LNK1      000416R     LNK2      000430R
LNK3      000346R     LNK4      000372R     LOOPR     000132R
LOOPW     000204R     MSG1      000246R     PC      =%000007
RD      = 000004      R0      =%000000      SP      =%000006
START     000030R     WR      = 000002      .       = 000604R
```

CHAPTER 3
OPERATOR COMMANDS

3.1 THE OPERATOR KEYBOARD INTERFACE

Through the operator keyboard, the user can communicate with

- the Monitor
- a program the user wrote to run under the Monitor
- a DOS system program (Assembler, PIP, Editor, etc.)

Rules which are common to all users of the operator keyboard under DOS are described in Section 3.2.

In communicating with the Monitor, the keyboard is used as a control device to allocate system resources, move programs into core, start and stop programs, and exchange information with the system. Commands which the user can type are described in detail in Section 3.3 and summarized in Appendix D.

For use in communicating with a system program or a user's program, the operator keyboard functions as a normal input device; the data from the keyboard may be transferred to a buffer in the program, or it may be preprocessed by a special routine called the Command String Interpreter (CSI), described in Section 3.4.

When the system requests input from the keyboard, a single character is printed on the teleprinter:

Character	Meaning
$	The system is idle and will remain idle awaiting an operator command. A command can be entered.
.	The Monitor has acknowledged a CTRL/C typed by the operator and is in listening mode, ready to accept a command from the operator.
#	A system program or user's program requests an operator reply through the CSI.
*	A system program requests an input message directly (i.e., not through CSI).

3.2 COMMUNICATING THROUGH THE KEYBOARD

Since the Monitor and any program operating under it must share the keyboard, the user must specify whether a given keyboard input is intended for the Monitor or for the operating program:

- All characters following a CTRL/C (typed by holding down the CTRL key while typing the C key) or following a $ output by the Monitor through the next RETURN are interpreted as Monitor commands and are passed to the Monitor for execution.
- All other characters are assumed to be for the operating program, provided one is currently in core and the keyboard device has been associated with one of its datasets. In this case, the characters will be buffered until required by the program. The characters will be ignored if no program has been loaded or if it is not using the keyboard as one of its data media.

Certain keys on the keyboard have special functions. These are listed in Table 3-1.

Table 3-1
Special Keyboard Functions

Keyboard Key	Function
RETURN	Pressing RETURN terminates an operator command to the Monitor or a line of input to a system or user program. The RETURN key produces a carriage return and LINE FEED on the teleprinter.
RUBOUT	This key permits the correction of typing errors. Pressing RUBOUT once causes the last character typed to be deleted. RUBOUT does not delete characters past the previous line terminator.[1] If the last remaining character has already been deleted, a RUBOUT will be ignored. The Monitor prints the deleted characters delimited by backslashes. For example, if you were typing .APPEND and typed .APPAM instead, the error could be corrected by typing two RUBOUTS and then the correct letters. The typeout would be: APPAM\MA\END Notice that the deleted characters are shown in reverse order, i.e., in the order in which they are deleted.
CTRL/C	When the CTRL and C keys are pressed, the Monitor is alerted to accept a command from the keyboard. CTRL/C is echoed as ↑C RETURN LINE FEED period.

[1] line terminator is a LINE FEED, FORM FEED, or VERTICAL TAB

(continued on next page)

Table 3-1 (Cont)
Special Keyboard Functions

Keyboard Key	Function
CTRL/C (Cont)	The operator can then type in a command to the Monitor; while the command is being typed, the interrupted program continues running normally except that any output to the teleprinter is interrupted until the command is terminated by the RETURN key. CTRL/C will interrupt teleprinter output or keyboard input in a user program. However, Monitor action on a CTRL/C is not taken until any current Monitor command is completed because the keyboard interrupt is turned "off". Except for DUmp and MOdify, however, it appears to the user that action on a CTRL/C is immediate. CTRL/C puts the Monitor in listening mode only. If it is desired to stop the function of the operating program, the STop command (Section 3.3.4.1) should be used. If a second CTRL/C is typed before the RETURN terminating a command, the input so far will be erased, a fresh ↑C RETURN LINE-FEED period will be printed and the Monitor will await a new command.
CTRL/U	CTRL/U is used if the user has completely mistyped the current line and wishes to start it over (CTRL/U deletes the entire line back to the last line terminator). When given in a command, it will act as a second CTRL/C and echo as ↑U RETURN LINE FEED period. CTRL/U given within a line of program input will echo as ↑U RETURN LINE FEED. CTRL/U may also be used to stop the printing of the current line of program output provided that no other input characters are still awaiting processing by the program. In this usage, it will not be echoed.
;	; causes all characters up to the line terminator within a command string to be treated as comments. It effectively puts the keyboard off-line--all characters following are echoed, but no Monitor action is taken. If a ; appears on a line and no ↑C has been issued, it is passed to the user program's buffer like any other character.

3.3 MONITOR COMMANDS

A command to the Monitor consists of two parts: a command name and possibly one or more command arguments. A command name is a string of two or more characters; all characters after the first two and up to a delimiter are ignored. The command formats are given in this section. The following conventions apply to examples in this chapter.

- Brackets [ ] are used to enclose elements of the command which are optional, i.e., they may or may not appear depending on the desired Monitor reaction.
- Braces { } are used to indicate that a choice must be made from the enclosed information.
- A comma , indicates that either one comma and/or one space must appear in that position.
- device name refers to a physical device name, as listed in Appendix A.
- dataset specifier may be represented by any portion of the expression:

  dev:filenam.ext,[uic]

  where

dev:	is a physical device name (as listed in Appendix A) and is followed by a colon.
filenam	is a file name of up to 6 characters (as described on Page 2-17).
.ext	is a period followed by a filename extension of up to 3 characters.
uic	is the user's identification code in the form: [Group No., User No.] (the uic must be typed within brackets)

- logical name is the name given to the dataset by the user in link block word LNKBLK + 2.

NOTE

To distinguish in the examples between the echo from an operator command on the teleprinter and the Monitor's solicited response, the Monitor's response will be underlined.

RETURN is represented by <CR> and is echoed by the Monitor as RETURN and LINE FEED.

If a command cannot be executed satisfactorily, an appropriate message will be printed at the teleprinter and the command will be ignored. The message will be one of the following.

Message	Meaning
ILL CMD!	Command requested does not exist
INV CMD!	Command cannot be accepted at this time (e.g., KILL with no program to kill)
SYN ERR!	Syntax of command is faulty
ILL DEV!	The device specified is illegal
NO FILE!	File specified does not exist
ILL ADR!	Address is illegal (not on word-bound or in core)
NO CORE!	Insufficient core capacity to execute command (SAVE)

3.3.1 Commands to Allocate System Resources

3.3.1.1 The ASsign Command

AS [SIGN] [,dataset specifier, logical name]

The ASsign command assigns a physical device (and, when the device is file structured, a file name) to the dataset specified by "logical name". The ASsign command overrides any assignment made in the dataset's Link Block. If no file name is specified in the "dataset specifier", the file name in the associated Filename Block is used. If no device name is specified, the device given in the Link Block stands (no default is assumed). Any file name specified for a nonfile- structured device is ignored.

Note that a device is assigned to a dataset, and that reassigning it for one dataset does not reassign it for all datasets.

he ASsign command can be given at any time the Monitor is in core:

- If ASsign is given before a program is loaded, the device assignment will remain in effect until another ASsign is given with no arguments, or until the Monitor itself is reloaded. ASsign given at this time enables the user to specify the same assignment for a set of programs to be run.
- If ASsign is given after a program is loaded, (i.e., after a GEt command), the assignment will remain in effect as long as the program is in core, or until the user performs a reassignment. As soon as the program disappears (by an .EXIT request or a KIll command), the assignment is released.
- ASsign may also be given after a program is running. For example, as recovery from a

  A003 message (Device not available)

  the user would do an ASsign followed by COntinue. The assignment will remain in effect as long as the program is in core, until the programmer reassigns, or restarts the program with a BEgin command.

  Doing an ASsign at this time is provided for such emergency situations, but is not recommended as standard practice because it causes an extra buffer to be allocated from free core and it will only be effective if the program has not already INITed the dataset to some other device.

For example, to assign DECtape file FREQ.BIN to dataset FRQ:

```
↑C
.AS,DT:FREQ.BIN,FRQ<CR>
```

3.3.2 Commands to Manipulate Core Images

3.3.2.1 The RUn Command

RU[N], dataset specifier

The RUn command loads into core the specified program from the specified device and starts its execution at the normal start address. The RUn command is equivalent to a GEt command followed by a BEgin command. RUn is valid only when there is no program already loaded.

- If a READ error occurs during the loading of the program, a fatal error message F021 xxxxxx is printed.
- If RUn calls a program which is not in the proper form (i.e., is not in formatted binary or does not have a start address), it produces a fatal error and the following message is printed:

  F022 xxxxxx
- If the program to be loaded is too large for available core, the fatal error message F023 (program size) is printed. Recovery from all these errors will be by way of a KIll command.

The user need not be currently logged in to use programs stored in the system area (UIC 1,1)--the RUN command processor will automatically search this area if the requested program does not appear in the user's own file area. If, however, the UIC is explicitly stated in the command string, only the relevant file area will be searched. The search order is: 1) user's area for file name as given; 2) UIC [1,1] for file as given; 3) user's area for file with extension LDA if no extension is given; and 4) UIC [1,1] for file .LDA. (Exact specification will, of course, reduce execution time particularly for devices such as DECtape for which search time can be lengthy.)

3.3.2.2 The GEt Command

GE[T], dataset specifier

The GEt command loads the specified dataset from the specified device. GEt is valid only when there is no dataset already loaded. Error reporting will be the same as for RUn. The user should use a BEgin or ODt command to commence execution.

3.3.2.3 The DUmp Command

DU[MP], LP: [, O][, {start addr / 0} [, end addr]]

This command prints on the Line Printer an absolute copy of the contents of the specified core area. The core image is not altered. O specifies a dump from core. An O is assumed on default, but the commas are required. 0 is assumed if no START ADDRESS is specified, and the highest word in memory is assumed if no END ADDRESS is specified. DUmp is valid at any time; if given while a program is running, it will merely suspend operations for the time required to effect the dump.

3.3.2.4 The SAve Command

SA[VE] [,dataset specifier]

SAve writes the program in core onto the device in loader format. The core image is not altered. SAve is valid only when a program is in core but not running, i.e., immediately after loading with a GET command or after being halted either by a STOP command or fatal error.

If no dataset specifier is given, the SAVE processor will automatically set up a file called SAVE.LDA on the system disk after it has deleted any current file of the same name. If the user wishes to retain the current file, he must first rename it using PIP. If the dataset specifier is given, the file named must not already exist or the command will be rejected. System disk is assumed by default if the dataset specifier contains only a filename.

Normally, it is expected that the user will only wish to save his program area. If this is the case, the range need not be given and the new file will begin from the program's low limit and extend to the top of core. If any other area is to be saved, the user should include the following at the end of the command:

/RA:low:high

/RA is the range switch, and low and high define the limits required (each being valid octal word-bound addresses) The saved image will be preceded by the same communication information as that for the original program loaded.

The SAVE processor will endeavor to get an extra 256-word buffer in order to satisfy the command. If this request cannot be granted because of insufficient free core, the command will be rejected. The user is therefore advised to use this facility only after he has released any datasets currently established.

Once the SAVE command has been syntactically verified, any errors will be handled by the SAVE processor, which will print a relevant message and recall Monitor listening mode:

DEVICE FULL	End of output medium reached
FILE ERROR XXX	File structures error as indicated by XXX = File Status Byte (see Section 2.8.6.2)

3.3.3 Commands to Start a Program

3.3.3.1 The BEgin Command

BE[GIN] [,address]

The BEgin command starts the execution of a program at the stated address. If no address is specified, the normal start address will be used. This command is valid only if a program is already in core. BEgin is used after a GEt, a STop, or following a fatal error condition. The GEt command followed by a BEgin command is equivalent to a RUn command. If given after a program has been started, a BEgin will clear all core allocations to buffers, device drivers, and assignments made dynamically, and the

stack will be cleared before control is passed back to the program. If any files are under creation at this time, they will be deleted.

To start a program at its normal start address, type:

BE <CR>

To start a program at absolute location 3446, type:

BE,3446 <CR>

3.3.3.2 The COntinue Command

CO[NTINUE]

This command is used after a WAit or a recoverable error condition (operator action message) to resume program operation at the point where it was interrupted. It is valid only if a program is already in core.

3.3.3.3 The REstart Command

RE[START] [,address]

This command restarts the program at the given address. If ADDRESS is not specified, the address set by the .RESTART programmed request (Section 2.8.2.2) is assumed. If neither address is specified, the command is rejected.

REstart is valid only if a program is already in core. Before the resumption of operations, the stack will be cleared, any current I/O will be stopped, and all internal busy states will be removed. However, buffers and device drivers set up for I/O operations will remain linked to the program for further use.

3.3.4 Commands to Stop a Program

3.3.4.1 The STop Command

ST[OP]

This is an emergency command to stop the program and kill any I/O in progress (by doing a hardware reset). The program may be resumed with either BEgin or REstart. STop is valid only if a program is in core.

3.3.4.2 The WAit Command

WA[IT]

This command suspends the current program and finishes any I/O in progress. Program can be resumed with either COntinue or REstart. WAit is valid only if a program is in core.

3.3.4.3 The KIll Command

KI[LL]

This command stops the execution of the current program after closing all open files and completing any outstanding I/O, and removes the program from core by returning control to the Monitor. It is valid only when a program is in core. To resume operations, the user must reload the program or load another by RUn or GEt.

3.3.5 Commands to Exchange Information with the System

3.3.5.1 The DAte Command

DA[TE] [,date]

This command sets the Monitor's date-word to the date specified in date, or if date is not specified, it prints the date previously entered. DAte is valid any time. (It should be noted that the date-word will not be updated internally; the operator must reset it daily if such information is needed.) Day is specified and output in the following format:

dd-mmm-yy

where dd = day, mmm = month, and yy = year. If the user input is an invalid date, e.g., 37-MAR-K4, 00-XXX-yy will be printed.

3.3.5.2 The TIme Command

TI[ME] [,time]

Sets the time-of-day entry in the Monitor to the time if time is specified; otherwise it prints the present content of the time-of-day. The format of time is:

hh:mm:ss

where hh = hours
mm = minutes
ss = seconds

The TIme command is valid at any time.

NOTE

The clock service routine does not automatically zero time at midnight; as with DATE, this must be set daily.

3.3.5.3 The LOgin Command

LO[GIN],uic

This command allows the user to give his user identification code to the Monitor. It is a valid command only when there is no program loaded in core and provided no other user has logged in and not FInished.

3.3.5.4 The MOdify Command

MO[DIFY], octal address
octal address/contents: [new contents]

This command allows the user to make changes in the contents of the absolute memory location specified by octal address. After the RETURN is typed at the end of the first line, the system responds by printing the contents of that address. At this point, the user can type one of the following (<CR> denotes the RETURN key; <LF> the LINEFEED key).

<CR>	will leave the contents unmodified
new contents <CR>	will change contents to new contents
Replacing <CR> by <LF>	will take similar action and then automatically print the contents of the next location.

This command is valid at any time. To change the contents of location 40000:

```
↑C
.MO,40000 <CR>
40000/164060:  104060 <CR>
```

Then to examine the contents of 40000:

```
↑C
.MO,40000 <CR>
40000/104060:  <CR>
```

To examine the contents of locations 40000 and 40002, the sequence would be:

```
↑C
.MO,40000 <CR>
40000/104060 <LF>
40002/000003 <CR>
```

NOTE

Entry of an address outside the available core memory as part of the original MOdify command will cause an error, and the command will be rejected. However, no check is made during line-feed sequence: if the user 'steps' outside memory, an illegal address trap will be taken.

3.3.5.5 The FInish Command

FI[NISH]

This command informs the Monitor that the current user is leaving the system. This command is valid only when no user program is in core. The Monitor deletes all files which do not have bit 7 on the protect byte set (Figure 2-11). This byte can be set at the file's creation, or by the .KEEP programmed request (Section 2.7.6). On completion, a completely new copy of the resident Monitor will be "booted" from the disk.

3.3.6 Miscellaneous Commands

3.3.6.1 The ECho Command

EC[HO]

This command suppresses teleprinter echo from the keyboard input to a user program. A subsequent ECho command turns the echo on again. The teleprinter as an output device for the program or the Monitor is not affected.

This command is valid only when a program is running in core and using the keyboard as a device.

3.3.6.2 The PRint Command

PR[INT]

This command suppresses teleprinter printing when the teleprinter is used as an output device to a user program. A subsequent PRint command turns the printing on again. PRint is valid only when a program is running in core and is using the teleprinter as a device.

3.3.6.3 The ENd Command

EN[D] [,{KB / PT}]

This command tells the Monitor "there is no more input from device KB (or PT)". It effectively generates an End-of-File from the keyboard (KB) or paper tape reader (PT). If no argument is specified, KB is assumed. If the program is expecting input at this time, it may be necessary to enter a second <CR> to ensure that the command is recognized.

ENd is valid only when a program is running in core.

3.3.6.4 The ODt Command

OD[T] [,{R / K}]

This command starts the execution of the ODT-11R Debugger Program. The argument specifies which ODT start-address is used:

(No argument)	starts at START +0	(clear ODT breakpoint table without resetting breakpoints)

(continued on next page)

R	starts at START +2	(clears ODT breakpoint table after replacing old instructions at breakpoints)
K	starts at START +4	(leaves breakpoints exactly as they are)

For example, to reset all breakpoint locations to their former instructions and restart ODT:

```
↑C
.ODT,R
```

ODT is valid only when ODT is linked to a program and both are in core.

3.4 THE COMMAND STRING INTERPRETER (CSI)

The one common format for input and output dataset specifications to a system program is provided through a single Monitor routine, the Command String Interpreter (CSI). This routine preprocesses the specification for whatever system program it was called by.

The CSI may also be called by a user's program. The user's software interface with CSI is described in Section 2.8.5.

3.4.1 CSI Command Format

Whenever a system program requests input through the CSI, a # will be printed on the teleprinter and the program will wait for the operator's reply. Generally, a CSI command consists of one or more output dataset specifications, followed by <, followed by one or more input dataset specifications. Spaces, horizontal TABs, and nulls may appear anywhere in the string and are ignored. A command is terminated by a FORM FEED, LINE FEED, or VERTICAL TAB. If RETURN appears within a command, the character which immediately follows must be a space, horizontal TAB, null, RUBOUT, or one of the command terminators; otherwise, an error will result. It should also be noted that typing the RETURN key causes RETURN and LINE FEED to be passed to the program, hence terminating the input.

< need not occur. If it does, at least one input file specification must appear. Only one < per command is allowed. Commands can not be continued from line to line.

A dataset specification must be delimited by a comma. If no items appear before the comma, it is interpreted as "this particular positional field will not be used". For example, suppose a program requires three (output) data specifications. Then the syntax:

```
Dataset Specification,,Dataset Specification
```

indicates that the second (output) dataset specified will not be generated.

Each dataset specification is a field which describes a dataset. It generally contains information as to where to find the dataset, the file name and extension if the dataset is a file, the user identification code associated with the file, and one or more switches which request various actions to be performed. A dataset specification containing all of the above elements would appear as:

$$dev:filnam.ext[uic]/sw_1:v_1:\ldots:v_n/sw_2:v_1:\ldots:v_n,$$

where dev = The device specification consisting of two or three letters (and often an octal digit) followed by a colon. The letters identify the device and the digit identifies the unit. Units must be given in octal. The colon delimits this field. Only physical names as listed in Appendix A may be specified. For example, DTA1: is the correct specification for DECtape, controller A, unit 1.

If no digit appears, unit 0 is assumed. If the device specification itself does not appear, the current device is assumed to be the device last specified, if there is one; otherwise, the system disk unit 0 is assumed.

Assumptions (defaults) do not carry across the <, i.e., from output to input.

filnam = The file name specification consists of one or more letters or digits, or exactly one asterisk. The first six letters or digits specify the name. The first character must be a letter. All letters and digits in excess of six are ignored.

The file name need not appear. No system-wide default file name is assumed.

.ext = The extension specification consists of a period, followed by one or more letters or digits, or followed by exactly one asterisk. The first three letters or digits specify the extension. All letters or digits in excess of three are ignored.

The extension need not appear.

The asterisk is used to specify "all". For example:

*.EXT specifies all files with extension .EXT,
FIL.* specifies all files with name FIL, and
. specifies all files and all extensions.

[uic] = The User Identification Code (UIC) specification consists of a left square bracket, followed by one or more octal digits or exactly one asterisk, followed by a comma, followed by one or more octal digits or exactly one asterisk, followed by a right square bracket. The field to the left of the comma specifies the user's group and the field to the right of the comma specifies the user within the group. Both fields must be given in octal, and the largest valid octal number is 376 in both cases (0 is invalid). For example, [12,136] is the correct specification for user number 136 of user group 12.

NOTE

The left and right square brackets are not visible on some keyboard keys; however, they are typed using SHIFT/K and SHIFT/M, respectively.

As in filnam and .ext, the asterisk specifies "all". For example:

[*,136] specifies all users whose number is 136

[12,*] specifies all members of user group 12, and

[*,*] specifies all users.

The user identification code need not appear, in which case the default is the identification entered by the user currently entering the command.

/sw:v_1:...:v_m = A switch specification consists of a slash (/), followed by one or more letters or digits, and optionally followed by one or more value specifications. A value specification is initially delimited by a colon. The value itself can be null, or consist of one or more letters, digits, periods, or dollar signs. Other characters are illegal. The digits 8 and 9 are legal.

For examples: /DATE:12:20:69 might be a switch to enter December 20, 1969 in a date field.

/DATE:12::69 might enter December, 1969 in a date field.

Switches need not appear. If a switch does appear, it need not contain more than one letter or digit after the slash. For example:

/S and /SWITCH2 are both legal.

The first two characters after the slash uniquely identify the switch. For example:

/S is treated as if it were /S null.
/SWITCH1 and /SWITCH2 are both treated as /SW.

Table 3-2 summarizes the legal command syntax.

Table 3-2
.CSI Command String Syntax Rules

Item Which Last Appeared	Item Immediately Following							
	,	DEV:	FILNAM	.EXT	UIC	/SWITCH	<	Terminator
blank[1]	*	*	*	E	*	*	*	*
,	*	*	*	E	*	*	*	*
DEV:	*	E	*	E	*	*	*	*
FILNAM	*	E	E	*	*	*	*	*
.EXT	*	E	E	E	*	*	*	*
UIC	*	E	E	E	E	*	*	*
/SWITCH	*	E	E	E	E	*	*	*
<	*	*	*	E	*	*	E	E

Legend: E indicates error. * indicates legal.

Note: [1] The next item encountered is the first item in the command string.

For example, a device specification immediately followed by an extension specification is an error, whereas a file name specification immediately followed by a comma is legal.

3.4.2 CSI Command Example

An example of a complete command is:

```
F1.E1,,DTA1:F2.E2/S:1<F3.E3[11,123],DTB:F4.E4/ABC,F5.E5
```

which is interpreted as explained below.

a. The first positional output dataset is to be a file named F1 and will have extension E1. It is to be put on disk unit 0, and catalogued under the ID of the user who entered the command. No switches are associated with this dataset.

b. The second positional output dataset will not be generated.

c. The third positional output dataset is to be in a file named F2 and will have extension E2. It is to be put on the DECtape which is mounted on unit 1 of controller A. This file is to be catalogued under the ID of the user who entered the command. The action indicated by switch S with value 1 is to be performed on this dataset.

d. The fourth and subsequent positional output dataset will not be generated.

e. The first positional input dataset is a file named F3, and its extension is E3. It can be found on disk unit 0, catalogued under the user number 123 of user group 11. No switches are associated with this dataset.

f. The second positional input dataset is a file named F4 and its extension is E4. It can be found on the DECtape currently mounted on controller B, unit 0. Associate the ID of the user who entered the command with this dataset. Perform the action indicated by switch AB (not ABC) on this dataset. No values are associated with the switch.

g. The third positional input dataset is a file named F5 and its extension is E5. It can be found on the DECtape currently mounted on controller B, unit 0. Associate the ID of the user who entered the command with this dataset. No switches are associated with this dataset.

h. The fourth and subsequent input datasets are not required.

APPENDIX A
PHYSICAL DEVICE NAMES

Mnemonic	Device	Radix-50 Equivalence
DC	RC11 Disk	014570
DF	RF11 Disk	014760
DK	RK11 Disk	015270
DT	DECtape (TC11)	016040
KB	ASR-33 Keyboard/Teletype	042420
LP	Line Printer (LP11)	046600
MT	Magtape (TM11)	052140
PP	High-Speed Paper Tape Punch	063200
PR	High-Speed Paper Tape Reader	063320
PT	ASR-33 Paper Tape Device	063440
CR	Card Reader (CR11)	012620

NOTE

a. Device mnemonics may be three letters on a particular system. The third letter is assigned if there is more than one controller, e.g.:

DTA for DECtape controller "A"
DTB for DECtape controller "B"

b. The device name may be followed by an octal number to identify a particular unit when the controller has several device units associated with it, e.g.:

DT1 indicates unit 1 under a single DECtape control.

DTA1 indicates unit 1 under controller A in a multicontrol situation.

APPENDIX C
SUBSIDIARY ROUTINE ASSIGNMENTS

The routines associated with the GLOBAL NAMES specified below are called by the REQUEST processor as indicated:

(blank) = subsidiary routine is never called

X = subsidiary routine is called only when a file structured device is referenced

L = subsidiary routine is called only when a linked file is referenced

C = subsidiary routine is called only when a contiguous file is referenced

D = subsidiary routine is called only when DECtape is referenced

M = subsidary routine is called only if Magtape is referenced

For example, if a user wants all .OPENI processing routines core resident, he would put the following assembler directive in his program:

```
.GLOBL    OPN.,FOP.,LUK.,CKX.
```

	Global Name												
Request	FOP.	FCR.	FCL.	LUK.	LBA.	GMA.	CBA.	CKX.	DLN.	DCN.	AP2.	GNM.	MTO.
.READ/WRITE												X	
.OPENU	X			X				X					M
.OPENO		X		X	X	X		X					M
.OPENE	X			X	X	X		X					M
.OPENI	X			X				X					M
.OPENC	X			X				X					
.CLOSE			X										
.ALLOC				X			X	X					
.DELET				X				X	L	C			
.RENAM				X				X					
.APPND				X				X			D		
.LOOK				X				X					
.KEEP				X				X					

APPENDIX D
SUMMARY OF MONITOR COMMANDS

Command	Usage
Commands to Allocate System Resources	
ASsign	Assign a physical device to a logical device name
Commands to Manipulate Core Images	
RUn	Load and begin a program
GEt	Load a program
DUmp	Write a specified core area onto a device as a core image
SAve	Write a program onto a device in loader format
Commands to Start a Program	
BEgin	Start execution of a program
COntinue	Resume execution of a halted program
REstart	Restart execution of a previously operating program
Commands to Stop a Program	
STop	Halt the current program, including any I/O in progress
WAit	Halt current program after finishing any I/O in progress
KIll	Halt the current program, finish any I/O in progress, close all open files, and pass control back to the Monitor
Commands to Exchange Information with the System	
DAte	Fetch/Specify date
TIme	Fetch/Specify time

(continued on next page)

Command	Usage
Commands to Exchange Information with the System (Cont)	
LOgin	Enter User Identification Code
MOdify	Modify contents of memory location
FInish	Log off system
Miscellaneous Commands	
ECho	Disable/enable keyboard echo to user program
PRint	Disable/enable teleprinter output from user program
ENd	End input from a device
ODt	Begin operation of Octal Debugger (ODT)

Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page

.ALLOC	Allocate a Contiguous File	.ALLOC LNKBLK,FILBLK,N	MOV #N,-(SP)
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 15
				2-32

.APPND	Append to a Linked File	.APPND LNKBLK,FIRST,SECOND	MOV #SECOND,-(SP)
			MOV #FIRST,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 22
				2-37

.BIN2D	Convert Binary to Decimal ASCII	.BIN2D ADDR,WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #3,-(SP)
			EMT 42
				2-55

.BIN2O	Convert Binary to Octal ASCII	.BIN2O ADDR,WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 42
				2-57

.BLOCK	Transfer a Block	.BLOCK LNKBLK,BLKBLK	MOV #BLKBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 11
				2-25

.CLOSE	Close a Dataset	.CLOSE LNKBLK	MOV #LNKBLK,-(SP) EMT 17	2-19
.CORE	Obtain Core Size	.CORE	MOV #100,-(SP) EMT 41	2-44
.CSI1	CSI Interface - part 1	.CSI1 CMDBUF	MOV #CMDBUF,-(SP) EMT 56	2-58

(continued on next page)

APPENDIX E

SUMMARY OF MONITOR PROGRAMMED REQUESTS

Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.CSI2	CSI Interface - part 2	.CSI2 CSIBLK	MOV #CSIBLK,-(SP) EMT 57	2-59
.DATE	Obtain Date	.DATE	MOV #103,-(SP) EMT 41	2-47

.DELET	Delete a File	.DELET LNKBLK,FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 21
				2-34

.D2BIN	Convert Decimal ASCII to Binary	.D2BIN ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 42
				2-53

.EXIT	Exit to Monitor	.EXIT	EMT 60	2-41
.GTUIC	Get Current UIC	.GTUIC	MOV #105,-(SP) EMT 41	2-49
.INIT	Initialize a Dataset	.INIT LNKBLK	MOV #LNKBLK,-(SP) EMT 6	2-14

.KEEP	Protect a File	.KEEP LNKBLK,FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 24
				2-40

.LOOK	Directory Search	.LOOK LNKBLK,FILBLK (,)
		
		(,) = optional argument
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
			
			or
			
			MOV #FILBLK,-(SP)
			CLR -(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
				2-38

.MONF	Obtain Full Monitor Size	.MONF	MOV #102,-(SP) EMT 41	2-46

(continued on next page)

Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.MONR	Obtain size of resident Monitor	.MONR	MOV #101,-(SP) EMT 41	2-45

.OPENx	Open a Dataset	.OPENx LNKBLK,FILBLK	MOV #CODE,FILBLK-2
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
			CODE=1 for .OPENU
			2 for .OPENO
			3 for .OPENE
			4 for .OPENI
			13 for .OPENC
				2-16

.O2BIN	Convert Octal ASCII to Binary	.O2BIN ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 42
				2-56

.RADPK	Radix-50 ASCII Pack	.RADPK ADDR	MOV #ADDR,-(SP) CLR -(SP) EMT 42	2-51

.RADUP	Radix-50 ASCII Unpack	.RADUP ADDR,WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #1,-(SP)
			EMT 42
				2-53

.READ	Read from Device	.READ LNKBLK,BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 4
				2-21

.RENAM	Rename a file	.RENAM LNKBLK,OLDNAM,
		NEWNAM
			MOV #NEWNAM,-(SP)
			MOV #OLDNAM,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 20
				2-35

(continued on next page)

Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.RLSE	Release a Dataset	.RLSE LNKBLK	MOV #LNKBLK,-(SP) EMT 7	2-15

.RSTRT	Set REstart address	.RSTRT ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 41
				2-43

.SPEC	Special Function	.SPEC LNKBLK,SPCARG	MOV #SPCARG,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 12
				2-29

.STAT	Obtain Device Status	.STAT LNKBLK	MOV #LNKBLK,-(SP) EMT 13	2-30
.SYSDV	Obtain System Device Name	.SYSDV	MOV #106, -(SP) EMT 41	2-50
.TIME	Obtain Time of Day	.TIME	MOV #104,-(SP) EMT 41	2-48

.TRAN	Transfer absolute block	.TRAN LNKBLK,TRNBLK	MOV #TRNBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 10
				2-27

.TRAP	Set TRAP vector	.TRAP STATUS,ADDR	MOV #ADDR,-(SP)
			MOV #STATUS,-(SP)
			MOV #1,-(SP)
			EMT 41
				2-42

.WAIT	Wait for Completion	.WAIT LNKBLK	MOV #LNKBLK,-(SP) EMT 1	2-23

.WAITR	Wait for Completion; Return to ADDR	.WAITR LNKBLK,ADDR	MOV #ADDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 0
				2-24

.WRITE	Write on a Device	.WRITE LNKBLK,BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2
				2-22

(continued on next page)

NOTES:		
	ADDR	a memory address
	BLKBLK	address of BLOCK Block
	BUFHDR	address of Line Buffer Header
	CMDBUF	address of Command String Buffer
	CSIBLK	address of Command String Interpreter Control Block
	FILBLK	address of Filename Block
	FIRST	address of Filename Block of file which is to be appended to
	LNKBLK	address of Link Block
	N	number of 64-word segments requested
	NEWNAM	address of Filename Block containing the file's new name
	OLDNAM	address of Filename Block containing the file's old name
	SECOND	address of Filename Block of file which is appended
	SP	Stack Pointer (register R6)
	SPCARG	code for Special Function or Address of Special Function Block as determined by Function called.
	TRNBLK	address of TRAN Block

APPENDIX F
SUMMARY OF DOS ERROR MESSAGES

Following is a complete summary of all error messages which can appear when using the DOS Monitor and system programs.

F.1 ACTION MESSAGES

Action messages are printed and the program is suspended. The Monitor expects the operator to take some action such as "continue the program" (type COntinue), or "kill the program" (type KILL).

Error Code	Additional Information	Meaning
A001	User Call Address	Disk address error.
A002	Device (RAD50)	Device not ready (Appendix A). Make device ready and type CO.
A003	Link Block Address	The Link Block contains either an illegal device code or no device code at all. Use the MODIFY command to display the contents of Link Block +2, which is the device name (RAD50), and then use the ASSIGN command to assign a device and/or file (Appendix A); then type CO.
A004	User Call Address	DECtape error. Try adjusting the tape; type CO to continue.
A005	Pause Number	A PAUSE was encountered in a FORTRAN program. Type CO to continue.
A006	Correct Module Name	Loading paper tape out of order on Pass 2 of Linker. Load correct module and type CO to continue.
A007	Call Address	Magtape. The name of the output file being created is the same as that of an existing file. Type CO to write over the old file.
A010	0	Magtape. A parity error occurred when trying to open a file. Type CO to continue searching. If the file being sought has a parity error, it cannot be found.

Error Code	Additional Information	Meaning
A043	Disk Pack Block Number	This is the block that is bad; issued by the RK11 pack initializer to provide a list of bad blocks and to permit job termination if too many are bad. Type CO if number of bad blocks thus far is tolerable.

F.2 INFORMATION MESSAGES

Informational and Warning messages are printed and the program generally continues.

I350	STOP Number	A STOP statement was executed in a FORTRAN program.
I351	0	More errors of a specified type occurred than were allowed. The program is terminated.
I352	Address of DEVTB entry	The logical device specified is not available. (See FORTRAN device table, DEVTB, for a layout.)
I353	Error Class Number	No logging device. The command input device was in use when a run-time diagnostic message was to be issued. Because of a device conflict the normal message could not be issued.
I354	0	Illegal response to CONFIRM: when attempting to zero an RK11 disk cartridge. The disk was not zeroed.

F.3 WARNING MESSAGES

W002	Device Name (RAD50)	Device time out (Appendix A)
W043	Block Number	Transfer error while using ITRAN to zero the disk.
W101	No. of Task Called	Task called by number not present or call number illegal. Request ignored. (RSX)
W102	Addr. in Call Sequence	Delay units not correct in call start. Request ignored. (RSX)
W103	Addr. in Call Sequence	Delay time too large in call start. Request ignored. (RSX)
W104	Addr. in Call Sequence	No time slot available. Request ignored. (RSX)
W105	Current Run-Time	A level 1 task has exceeded its maximum run time. Task continues. (RSX)

Error Code	Additional Information	Meaning
W106		Illegal or unrecognized console command. Command ignored. (RSX)
W107	Report Number	Illegal system report number in system command. Command ignored.
W110	Addr. in Call Sequence	Attempted to start a background task while the background is busy. Request ignored.
W111	Addr. in Call Sequence	Attempted to clock a background task. Request ignored.
W112		Symbolic task name not found. Request ignored.
W113		Command syntax error. Command ignored.
W114	Addr. in Call Sequence	Illegal clock (call TRNON) time. Request ignored.
W300	Module Name	Non-unique object module name.
W301	Addr. of Byte Error	Byte relocation error; Linker automatically continues.
W302	Symbol and Module Names	Multiple definitions of global symbol. Second definition is ignored and linking continues.
W303		Buffer overflow. Overflow of one of the following Editor buffers: Command Input Buffer Save Buffer Page Buffer
W304		Macro overflow. The command string as stored in the Save Buffer was too long to execute, when requested to do so by an EM (Execute Macro) command.
W305		Recursive macro. The command string as stored in the Save Buffer contains an EM command.
W306		Empty Save Buffer. An EM or U (Unsave) command was issued with nothing in the Save Buffer.
W307		Search failure. The n^{th} occurrence of the search object was not found in the available text.
W310		Unsave failure. Insufficient room to copy the contents of the Save Buffer into the Page Buffer at dot.
W311		End-of-data detected. The end of the input file or the end of the input medium was reached during the last Read of text into the Page Buffer. Last page read was last in the file.

Error Code	Additional Information	Meaning
W312		Illegal line feed. A line feed character was encountered in the command string.
W313		Illegal negative argument. A negative argument was used with a command that does not accept them. Negative arguments are not permitted by the specified command.
W314		Arguments not permitted. The command specified does not permit any argument with it.
W315		Illegal argument. The given argument was not acceptable to the specified command.
W316		Illegal text string. Usually caused by the lack of a second delimiter.
W317		Illegal command. The Editor was unable to execute the specified command. The command may be an illegal character, one that is not an EDIT-11 command character.
W320		Page buffer almost full. The Page Buffer was within 128 characters of being full.
W321		File closed. An attempt to Read from or Write to a primary file after an EF (End-of-File) command was issued.
W322		Undefined global symbols in load module. Linking continues.
W323		Illegal size of named .CSECT. or illegal entry in named .CSECT or task's named .CSECT size too large (RSX).
W324		Too many entries in task's named .CSECT (RSX).
W325		Illegal priority specification in real-time header (RSX).
W350	Number of Failures	Power fail (RSX).

F.4 FATAL MESSAGES

Fatal error messages are printed, if possible, and the program is suspended. The Monitor will not allow the operator to continue the program, but expects to see either a BEgin, REstart or KILL command. If a fatal error is a system disk failure and the error message cannot be printed, the central processor halts. This is the only time that a halt occurs in the Monitor.

Error Code	Additional Information	Meaning
F000	Request Address	Dataset not INITed. Program must issue .INIT before any other requests to a dataset. page 2-19, -23)
F001	Request Address	Stack overflow. Once loaded, a program requires additional space for its stack, buffers, and control blocks. These are allocated as they are needed. Reduce the size of the program.
F002	Request Address	Invalid EMT call (Appendix B). The EMT code issued by the program has not been assigned
F003	Request Address	Invalid .TRAN function (page 2-27 and 2-72).
F005	Request Address	.RLSE error. If a file has been OPENed, it must be CLOSEd before a .RLSE can be issued. (page 2-15)
F006	Request Address	Device full. No more space exists on the device being referenced by the request. For a file-structured device, use PIP to look at the number of free blocks and delete any files which are not needed.
F007	Request Address	No buffer space (see F001).
F010	Request Address	Illegal .READ/.WRITE. Incorrect mode for device or file not opened correctly.
F011	Request Address	Illegal OPEN. Unused code or type unsuitable for device.
F012	Request Address	(See table below.)
F014*	Request Address	Device error on trying to read bit map.
F015	Request Address	DECtape error. Nonexistent memory addressed or end-zone reached during transfer.
F016	Block Number	DECtape search failure. Block requested cannot be found.
F017*	Device (RAD50)	Parity error on file-structured device. (Page 2-19 and App. A)
F020	Irrelevant	Too many datasets using low-speed paper tape. A maximum of one for each direction is allowed. Restart your job and use the ASSIGN command to reassign the excess datasets.
F021*	Irrelevant	Program loader read failure.
F022	Irrelevant	Program loader format error. File being loaded is not a load module.
F023	Program Size	Program too large for core available.

Error Code	Additional Information	Meaning
F024	Request Address	(See table below.)
F025	Device (RAD50)	Master directory full when attempting to add UIC. No more UIC's can be added. (Appendix A.)
F026*	Disk Control Status Register	Disk (RF11 or RC11) transfer failure. Hardware error or persistent parity failure. (See Peripherals & Interface Handbook, page 67.)
F027*	Error Register	Disk (RK11) transfer failure. (See Peripherals & Interface Handbook, page 84.)
F030	Err Class, Number	FORTRAN system error.
F031	Addr. of Log Device	No more room on FORTRAN logging device, or illegal end-of file was encountered while a FORTRAN READ was in progress.
F032*	Status Register	Magtape hardware error. (See Peripherals & Interface Handbook, page 44.)
F033	Special Function Block Address	Invalid special function block. (page 2-73)
F034	Call Address	The call code passed to a conversion request was invalid (e.g., '5 means binary-to-octal, but 6 is not defined).
F035	Block Number	Illegal block number (RK 11)
F036	Lowest Slot Used by Tasks	No slot available (RSX loader) .
F037	Lowest Slot Used by Tasks	Illegal slot specified (RSX loader).
F040	Low Address of Task Code	Attempted to overlay the executive or another task (RSX loader).
F041	Load Address of Binary Block	Attempted to load outside limits defined in the command (RSX loader).
F042*	Error Register	Disk (RP11) transfer failure (reserved).
F043*	Block Number	Illegal block number (RP11) (reserved).
F050	Request Address	Illegal I/O to batch stream.
F051	Request Address	Too many successive batch stream read errors.

Error Code	Additional Information	Meaning
F100	Address in Call Sequence	Insufficient arguments in call sequence or in console command (RSX).
F240	Irrelevant	An attempt was made to allocate a contiguous file, but not enough contiguous blocks are free.
F275	0	Incorrect argument to link subroutine.
F276	0	Transfer address of overlay not specified.
F277	Contents of PC	Overlay could not be brought into core.
F300	0	FORTRAN overlays cannot be executed -- FORTRN.OVR may be nonexistent or improperly constructed.
F342**	Contents of PC	Error trap. Probably caused by a reference to a byte boundary or to nonexistent memory or to a nonexistent device. Could also be caused by movement of the stack pointer below 400 or by executing JMP or JSR with register mode destination.
F344**	Contents of PC	Reserved instruction trap. The instruction just executed is not a valid PDP-11 instruction. Perhaps you jumped to a point outside your program or perhaps you have stored information over an instruction.
F346**	Contents of PC	Trace trap. Bit 4 of the Processor Status Register is on. Look for traps in the PDP-11 Processor Handbook.
F350**	Contents of PC	Power fail trap.
F352**	Contents of PC	Trap instruction trap. A trap instruction was issued by your program and you did not previously specify a trap address with the .TRAP request.
F356***	Contents of PC	Unexpected device interrupt. Either a new device has been added to your system without initializing the interrupt vector or a hardware failure has occurred.

* This is most likely a hardware error. If it persists, call a service engineer.

** The PC is pointing at the instruction following the erroneous instruction; subtract 2, 4, or 6 to get the address of the incorrect instruction. Consult TRAPS in the PDP-11 Processor Handbook for further information.

*** Both * and ** notes above.

Are you logged in?	LOgin
Is your UIC entered?	Enter it
Are you attempting to create a file which already exists ?	Run PIP and DELETE
Does the input file you are accessing exist?	?
Are you attempting to delete a nonexistent file?	?
Are you attempting to delete a locked file? (The command to delete is correct, and the file exists.)	Run PIP and UNlock
Are you attempting to access another user'sfile illegally? (Ask PIP for his DIrectory to find out.)	?

F.5 SYSTEM PROGRAM MESSAGES

System program messages are printed and the program generally continues.

Error Code	Additional Information	Meaning
S200	0	Too many .CSECT directives
S201	0	Conditionals nested too deeply
S202	Error Status Byte	EOD or device error on .WRITE or .READ. The disk may have filled up.
S203	0	Illegal switch, or too many switches, or illegal switch value, or switch value not given
S204	0	Too many or too few output files
S205	0	Too many or too few input files
S206	0	Input file not specified in command string
S207	Error Status Byte	EOD or device error on .TRAN
S210	0	Unrecognized symbol table entry
S211	0	An RLD references a global name which cannot be found in the symbol table
S212	0	An RLD contains a location counter modification command which is not last
S213	0	Object module does not start with a GSD
S214	0	The first entry in the module is not the module name

Error Code	Additional Information	Meaning
S215	0	An RLD references a section name which cannot be found
S216	0	The TRA specification references a nonexistent module name
S217	0	The TRA specification references a nonexistent section name
S220	0	An internal jump table index is out of range
S221		Unassigned
S222		Unassigned
S223	0	No more room for CSI input buffer or Monitor's file manager routine, or Monitor's Library search buffer
S224		Unassigned
S225	0	Program too large or top too low (program has been linked below zero in memory)
S226	0	An open angle bracket, <, is present in a line other than the first.
S227	1:No Primary Output 2:Sec In = Sec Out 3:Sec In = Pri Out 4:Pri In = Sec Out 5:Pri In = Sec In 6:Pri Out= Sec Out	Illegal file combination; arguments 2-6 for Editor-type commands. Pri = Primary File Sec = Secondary File
S230	Error Status Byte	Error on .BLOCK I/O
S231		Illegal command, file-structured device required
S232		No more than one action switch permitted
S233		Specified UIC not found in MFD
S234		Null file name given where file name required
S235		No files found in UFD
S236		Operation applicable to DECtape only
S237		File not found during file recovery operation
S240		No space for file allocate
S241		MFD is full
S242		Meaningless command; no action taken
S243	0	No < in first line of command
S244	0	Already past requested position
S245	0	Object module not found, could be out of order
S246	0	Illegal library format

Error Code	Additional Information	Meaning
S247	0	Listing requested, but unable to read output library from specified output device
S250	0	Core library symbol table not specified first or consecutively
S251	0	No files found for * request
S252	0	File name given when none allowed
S253	0	Linker error
S254	0	It is illegal to ZEro the system resident disk

APPENDIX G
I-O DRIVERS WITHIN THE DISK OPERATING SYSTEM

The principal function of an I/O driver is to satisfy the requirement of a Monitor processing routine for the transfer of a block of data in a standard format to or from the device it represents. This will involve both setting up the device hardware registers to cause the transfer and its control under the interrupt scheme of PDP-11, making due allowance for peculiar device characteristics (e.g., conversion to or from ASCII if some special code is used).

It may also include routines for handling device start-up or shut-down such as punching leader or trailer, and for making available to the user certain special features of the device, such as rewinding magtape.

G.1 Driver Structure

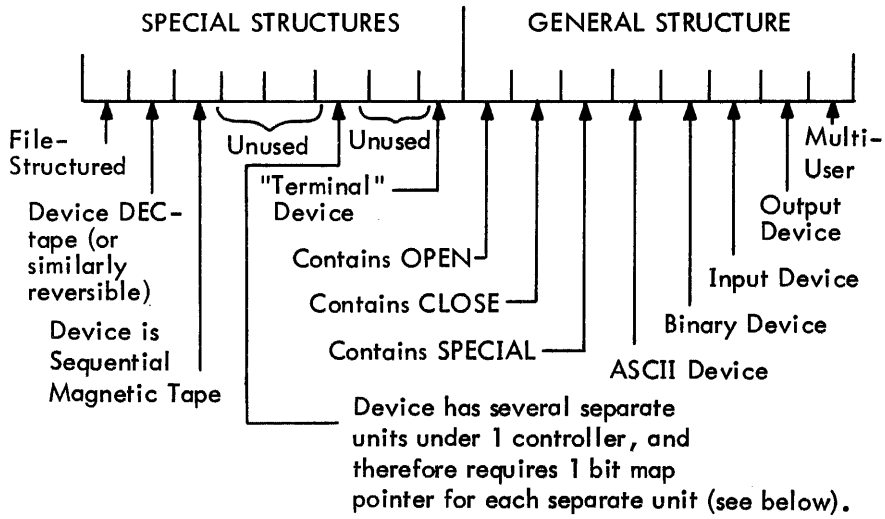
In order to provide a common interface to the Monitor, all drivers must begin with a table of identifying information as follows:

DVR:

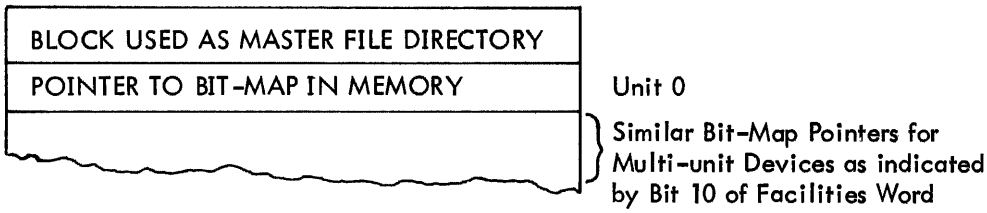
BUSY FLAG (initially 0)	
FACILITY INDICATOR (expanded below)	
Offset to Interrupt Routine*	Standard Buffer Size in 16-word Units.
Offset to OPEN Routine *	Priority for Interrupt Service
Offset to CLOSE Routine *	Offset to Transfer Routine *
Space	Offset to Special Functions*
DEV	NAME (Packed Radix-50)

Offsets marked * will enable calling routine to indicate routine required. They will be considered as an unsigned value to be added to the start address of the driver. This may mean that with a 256 maximum, the instruction referenced by the offset will be JMP or BR (routine).

Bits in the Facility Indicator Word define the device for Monitor reference:



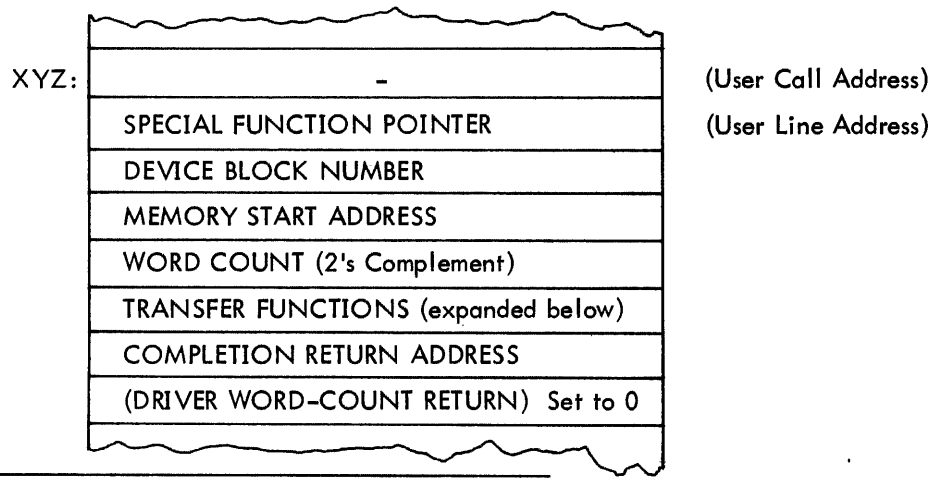
The table should be extended as follows if the device is file-structured:



The driver routines to set up the transfer and control it under interrupt, and possibly for OPEN, CLOSE, and SPECIAL, follow the table. Their detailed operation will be described later.

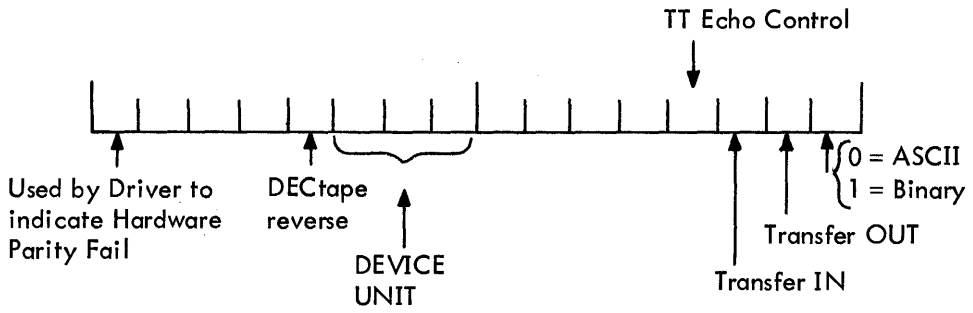
G.2 Monitor Calling

When a Monitor I/O processing routine needs to call the driver, it first sets up the parameters for the driver operation in relevant words of the appropriate DDB[1], as follows:



[1] Dataset Data Block - in full, a 16-word table which provides the main source of communication between the Monitor drivers and a particular set of data being processed on behalf of a user program.

The relevant content of the Transfer Function word is as follows:



Provided that the Facility Indicator in the Driver Table described above shows that the driver is capable of satisfying the request, both from the point of view of direction and mode and of the service required, the Monitor routine places in Register 0 the relative byte address of the entry in the Driver Table containing the offset to the routine to be used (e.g., for the Transfer routine, this would be 10). It then calls the Driver Queue Manager, using JSR PC,S.CDB.

The Driver Queue Manager ensures that the driver is free to accept the request, by reference to the Busy Flag (Word 0 of the driver table). If this contains 0, the Queue Manager inserts the address of the DDB from Register 0 and jumps to the start of the routine in the driver using Register 1 content to evaluate the address required. If the driver is already occupied, the new request is placed in a queue linking the appropriate DDB's for datasets waiting for the driver's services. It is taken from the queue when the driver completes its current task. (This is done by a recall to the Queue Manager from the routine just serviced, using JSR PC,S.CDQ.)

On entry to the Driver Routine, therefore, the address following the Monitor routine call remains as the "top" element of the processor stack. It can be used by the driver to make an immediate return to the Monitor (having the function requested), using RTS PC. It should also be noted that the Monitor routine will have saved register contents if it needs them after the device action. The driver may thus freely use the registers for its own operations.

When the driver has completely satisfied the Monitor request, it should return control to the Monitor using the address set into the DDB. On such return, Register 0 must be set to contain the address of the DDB just serviced and, since the return will normally follow hardware interrupt, Registers 0-5 at the interrupt must be stored on top of the stack.

G.3 Driver Routines

G.3.1 TRANSFER

The sole purpose of the TRANSFER routine is to set the device in motion. As indicated above, the information needed to load the hardware registers is available in the DDB, whose address is contained in the first word of the driver. Conversion of the stored values is, of course, the function of the routine. It must also enable the interrupt; however, it need not take any action to set the interrupt vectors as these will have been preset by the Monitor when the driver is brought into core. Having then given the device GO, an immediate retum to the calling processor should be made by RTS PC.

G.3.2 Interrupt Servicing

The form of this routine depends upon the nature of the device. In most drivers it will fall into two parts, one for handling the termination of a normal transfer and the other to deal with reported error conditions.

For word- or byte-oriented devices, the routine must provide for individual word or byte transfers, with appropriate treatment of certain characters (e.g., TAB or Null) and for their conversion between ASCII or binary and any special device coding scheme until either the word count in the DDB is satisfied or an error prevents this. On these devices, the most likely cause for such error is the detection of the end of the physical medium; its treatment will vary according to whether the device is providing input or accepting output. The calling program will usually need to take action in the former case and the driver should merely indicate the error by returning the unexpired portion of the word count in DDB Word 7 on exit to the Monitor. Output End-of-Data, however, will, in general, require operator action. To obtain this, the driver should call the Error Diagnostic Print routine within the Monitor by:

```
MOV    DEVNAM,-(SP)        ;SHOW DEVICE NAME
MOV    #402,0(SP)          ;SHOW DEVICE NOT READY
IOT                        ;CALL E.D.P.
```

On the assumption that the operator will reset the device for further output and request continuation, the driver must follow the above sequence with a Branch or Jump to produce the desired resumption of the transfer.

Normal transfer handling on blocked devices (or those like RF11 Disk which are treated as such) is probably simpler since the hardware takes care of individual words or bytes and the interrupt only occurs on completion. Errors may arise from many more causes, and their handling is, as a result, much more complex and device dependent. In general, those which indicate definite hardware malfunctions must lead to the situation in which the operator must be informed by a diagnostic message and the only recourse after rectification will be to start the program over.

At the other end of the scale there are errors which the driver itself can attempt to overcome by re-starting the transfer--device parity failure on input is a common example. If a retrial, or several, still does not enable a satisfactory conclusion, the driver should normally allow programmed recovery and merely indicate the error by Bit 17 of DDB Word 5. Nevertheless, because the program may wish to process the data despite the error, the driver should attempt to transfer the whole block requested if this has not already been effected. Between these two extremes, the remaining forms of error must be processed according to the type of recovery deemed desirable.

Whether the routine uses processor registers for its operation or not will naturally depend on considerations of the core space saved against the time taken to save the user's content. However, on completion (or error return) to the Monitor, as indicated in an earlier paragraph, the calling routine expects the top of the stack to contain the contents of all Registers 0-5 and Register 0 to be set to the address of the DDB just serviced. The drive must, therefore, provide for this.

G.3.3 OPEN

This routine need be provided only for those devices for which some hardware initialization is required by the user. It should not normally appear in drivers for devices used in a file-oriented manner. Its presence must be indicated by the appropriate bit (Bit 7) in the driver table Facility Indicator.

The routine itself may vary according to the transfer direction of the device. For output devices, the probable action required is the transmission of appropriate data (e.g., CR/LF at a keyboard terminal, form feed at a printer, or null characters as punched leader code), and for this a return interrupt is expected. The OPEN routine should then be somewhat similar to that for TRANSFER in that it merely starts the device and makes an interim return via RTS PC, waiting until completion of the whole transmission before taking the final retum address in the DDB.

On the other hand, an input OPEN will likely consist of just a check on the readiness of the device to provide data when requested. In this case, the desired function can be effected without any interrupt wait. The routine should, therefore, take the completion return immediately. Nevertheless, it must ensure that the saved PC value on top of the stack from the call to S.CDB is appropriately removed before exit. In the case of drivers which can only service one dataset at a time (i.e., Bit 0 of their Facility Pattern word is set to 0) and can never, therefore, be queued, it will be sufficient merely to use TST (SP) + to effect this. A multi-user driver, however, must allow for the possibility that it may be recalled to perform some new task already waiting in a queue. This is shown by the byte at DDB-3 being nonzero. In this case, the intermediate return to the routine originally requesting the new task has already been made directly by S.CDB. The address now on top of the stack is the return to the routine, whose task the driver has just completed and which has called S.CDQ to dequeue the driver.

This return must be taken when the first routine has performed its Completion Return processing. Moreover, this first routine expects to exit as from an interrupt. When a driver is recalled from a queue, it must simulate this interrupt. A possible sequence might be:

```
        MOV   DRIVER, R0        ;PICK UP DDB ADDRESS
        MOV   (SP)+, R5         ;SAVE INTERIM RETURN
        TSTB  -3(R0)            ;COME FROM QUEUE?
        BEQ   EXIT
        MOV   @#177776,-(SP)    ;IF SO, STORE STATUS
        MOV   R5,-(SP)          ; ...& RETURN
        SUB   #14,SP            ;DUMMY SAVE REGS
EXIT:   JMP   @14(R0)
```

G.3.4 CLOSE

As with OPEN, this routine should provide for the possibility of some form of hardware shut down such as the punching of trailer code and is not necessary for file-structured devices. Moreover, it is likely to be a requirement for output devices only. If it is provided, Driver Table Facility Indicator (Bit 6) must be set.

Again, the probable form is initialization of the hardware action required, with immediate return via RTS PC and eventual completion return via the DDB-stored address.

G.3.5 SPECIAL

This routine may be included if either the device itself contains the hardware to perform some special function or there is a need for software simulation of such hardware on other devices, e.g., tape rewind. It should not be provided otherwise. Its presence must be indicated by Bit 5 of the Facility Indicator.

The function itself is stored as a code, in the range $0\text{-}255_{10}$, in the first word of a table with a pointer being set by the Monitor in DDB+2. The upper byte of the same word gives the length of the following table. Depending upon the function, the remainder of the table may contain supporting information from the user program or will be used to return data to the program. When called, the driver routine must determine whether the function is appropriate in its case. If not, the completion return should be taken immediately with prior stack clearance as discussed under OPEN. For a recognized function, the necessary routine must be provided and this must decode the information table as required. Again, its exit method will depend upon the necessity for an interrupt wait or otherwise.

G.4 Drivers for Terminals

The rate of input from terminal devices is normally dictated externally by the operator, rather than being program-driven; moreover, for both input and output, the amount of data to be transferred on each occasion may be a varying value, i.e., a line rather than a block of standard size. Furthermore, there

may be problems with the conflict between echo of input during output. As a result, drivers for such devices will demand special treatment.

Normal output operation, i.e., .WRITE by the program, is handled by the Monitor Processor. On recognizing that the device being used is a terminal, as shown by Bit 8 of the Facility Indicator, this routine always causes a driver transfer at the end of the user line, even though the internal buffer has not been filled. The driver, however, is given the whole of a standard buffer, padded as necessary with nulls. Provided the driver can ignore these, the effect is that of just a line of output.

Input control on the other hand, must remain driver responsibility. Overcoming the rate problem will, in most cases, require circular buffering within the driver until demanded by the Monitor. At this point, transfer of data already in should occur. If this is sufficient to fill the Monitor buffer, the driver can await the next request before further transfer onward. If insufficient, it should operate as any other device and use subsequent interrupts to continue to satisfy the Monitor request. It must, nevertheless, stop any transfer at the end of a line in normal operation. In order to allow the Monitor to continue, the driver must simulate the filling of the buffer by null padding (of no consequence, since terminals are by nature character-based). (Normal operation, of course, means response to user .READ's and is indicated by the size of the buffer to be filled, namely the driver standard. Should the user be requesting .TRANs, the buffer size will vary from the standard in all likelihood and the driver may then assume he requires operation as a normal device--complete buffer fill-up before return.)

Where input echo is a further complexity, there will doubtless be other requirements. If the echo is made immediately after the input, it may be desirable to have a second buffer to cater for the likely situation that the echo will not exactly match its origin. On the other hand, if the echo is held for any length of time, perhaps to provide correct relations between program-driven output and the echo, the second buffer could be too expensive. A larger input buffer and routines to allow for several outputs to one input character while sitting on that character might be more convenient. The conflict between such echo and program-driven output will require controlled switching within the driver input and output handlers.

APPENDIX H
USING DEVICE DRIVERS OUTSIDE DOS

H.1 INTRODUCTION

Subroutines to handle I/O transfers between a PDP-11 and each of its peripheral devices are developed as required for use within the Disk Operating System (DOS). These subroutines are made available within an I/O Utilities Package for the benefit of PDP-11 users who have configurations unable to support DOS or who wish to run programs outside DOS control.

All the subroutines associated with one peripheral device together form an entity which is known as a Driver. The Device Driver Package (DEC-11-NIZA-D) provides a general description of a driver and shows how it may be used in a stand-alone environment. The unique properties of each driver are discussed in separate documents issued as supplements to the Device Driver Package. The I/O Utilities Package for any system is determined by the peripherals of that system. Thus the full documentation for a particular package consists of the Device Driver Package and applicable supplements.

Within this appendix, Section H.2 consists of an outline of the established driver structure and its interface to the program using it. Section H.3 then illustrates how a stand-alone program can match this interface in order to make immediate use of each driver as supplied within the package.

H.2 DRIVER FORMAT

H.2.1 Structure

The basic principle of all drivers under the DOS Monitor is that they must present a common interface to the routines using them in order to provide for device-independent operation. The subroutines are structured to meet this end. Moreover, the driver may be loaded anywhere in memory under Monitor control. Its code must always, therefore, be position-independent.

The detailed description of a driver is found in Appendix G. This section is concerned with driver interfaces.

H.2.1.1 Driver Interface Table - The first section of each driver consists of a table which contains, in a standard format, information on the nature and capabilities of the device it represents and entry pointers to each of its subroutines. The calling program may then use this table as required, regardless of the device being called.

H.2.1.2 Setup Routines - Each driver is expected to handle its device under the PDP-11 interrupt system. When called by a program, therefore, a driver subroutine merely initiates the action required by setting the device hardware registers appropriately. It then returns to the calling program by a standard subroutine exit.

The main setup routine prepares for a data transfer to or from the device, using parameters supplied by the calling program. Normally, blocks of data will be moved at each transfer. The driver will only return control to the program when the whole block has been actioned or when it is unable to continue because there is no more data available.

The driver may also contain subroutines by which the calling program may request start-up or shut-down action, such as leader or trailer code at a paper tape punch, or some special function provided by the device hardware (or a software simulation of that for some similar device), e.g., rewind of a magnetic tape (or DECtape).

H.2.1.3 Interrupt Servicing - The nature of the driver routine to service device interrupts is particularly dependent upon the extent of the hardware provisions of the device for controlling transfers. In general, the driver determines the cause of the interrupt and checks whether the last action was performed correctly or was prevented by some error condition. If more device action is needed to satisfy the program, the driver again initiates that action and takes a normal interrupt exit. If the program request has been fully met, control is returned to the program at an address supplied at the time of the call.

H.2.1.4 Error Handling - Device errors may be handled in two ways. There are some errors for which recovery can be programmed; the driver will, if appropriate, attempt this itself (as in the case of parity or timing failure on a bulk-storage device) or will recall the program with the error condition flagged (as at the end of a physical paper tape). Other errors will normally require action externally, perhaps by an operator. For the latter, the driver calls a common error handler based on location 34 (IOT call) with supporting information on the processor stack.

H.2.2 Interface to the Driver

H.2.2.1 Control Interface - The principal link between a calling program and any driver subroutine is the first word of the driver table. In order to provide the control parameters for a device operation, the calling program prepares a list in a standardized form and places a pointer to the list in the driver link. The called driver then uses the pointer to access the parameters. If the driver need then return status information, it may again place this in the list area via the link-word.

The first word of the driver may also act as an indicator in that while it remains 0 the driver is not already busy upon some task, whereas when the word contains a list-pointer the driver is assumed to be busy. Since most drivers can support only one job at a time, the link-word state can be significant.

H.2.2.2 Interrupt Interface - Although the driver will always expect to use the interrupt system, it does not itself ensure that its interrupt vector in the memory area below 400 has been set up correctly; the Monitor under DOS takes care of this. However, the Driver Table contains the necessary information to allow the vector to be set correctly.

H.3 STAND-ALONE USAGE

Because each driver is designed for operation within the device-independent framework of DOS Monitor, it may be similarly used in other applications. Possible methods will be discussed later. However, since the easiest way to use the driver is to assembly it with the program requiring it, this will be described first.

H.3.1 Driver Assembled with Program

H.3.1.1 Setting Interrupt Vector - As noted in Section H.2.2.2, the calling program must first correctly set the device transfer vector within memory locations 0-377. The address of the driver's interrupt entry point can be identified on the source listing by the symbolic name which appears as the content of the Driver Table Byte, DRIVER+5. The priority level at which the driver expects to process the interrupt is at byte DRIVER+6. For a program which can use position-dependent code, the setup sequence may be:

```
MOV     #DVRINT, VECTOR    ;SET INT. ADDRESS
MOVB    DRIVER+6, VECTOR+2 ;SET PRIORITY
CLRB    VECTOR+3           ;CLEAR UPPER STATUS BYTE
```

(where the Driver Table shows at DRIVER +5: .BYTE DVRINT-DRIVER).

If the program must be position-independent, it may take advantage of the fact that the Interrupt Entry address is actually stored as an offset from the start of the driver, as illustrated above. In this case, a sample sequence might be:

```
MOV     PC,R1               ;GET DRIVER START
ADD     #DRIVER-.,R1
MOV     #VECTOR,R2          ;...& VECTOR ADDRESSED
CLR     @R2                 ;SET INT. ADDRESS
MOVB    5(R1),@R2           ;...AS START ADDRESS+OFFSET
ADD     R1, (R2)+
CLR     @R2                 ;SET PRIORITY
MOVB    6 (R1),@R2
```

H.3.1.2 Parameter Table for Driver Call - For any call to the driver, the program must provide the list of control arguments mentioned in Section H.2.2.1. This list must adhere in general to the following format:[1]

```
[SPECIAL FUNCTION POINTER][2]
[BLOCK NO.][3]
STARTING MEMORY ADDRESS FOR TRANSFER
NO. OF WORDS to be transferred (2's complement)
STATUS CONTROL showing in Bits:

    0-2:    Function (octally 2=WRITE, 4=READ)[4]
    8-10:   Unit (if Device can consist of several, e.g., DECtape)
    11:     Direction for DECtape travel (0=Forward)

ADDRESS for RETURN ON COMPLETION
[RESERVED FOR DRIVER USE][5]
```

The list itself may be assembled into the required format if its content will not vary. The driver may return information in the area as described in a later paragraph; however, this will not corrupt the program data and it is removed by the driver before it begins its next operation.

On the other hand, most programs will probably wish to use the same area for the lists for several tasks or even between different drivers. In this case, the program must contain the necessary routine to set

---

[1] In some cases, it may be further extended as discussed in later paragraphs.

[2] Required only if Driver is being called for Special Function; addresses a Special Function Block as described in Section 2.6.4.1.

[3] Required only if the Device is bulk storage (e.g., Disk or DECtape).

[4] Most devices transfer words regardless of their content, i.e., ASCII or Binary. Some devices, e.g., Card Reader, may be handled differently for the two modes; for these, Bit 0 must also be set to indicate ASCII=0, Binary=1. (In these cases, the driver always produces or accepts ASCII even though the device itself uses some other code.)

[5] This word may be omitted if the device is bulk storage (see below).

up the list for each task before making the driver call, perhaps as illustrated in the next paragraph. It must be noted, however, that the driver may wish to refer to the list again when it is recalled by an interrupt or to return information to the calling program. Therefore, the list must not be changed until any driver has completed a function requested; for concurrent operations, different list areas must be provided.

H.3.1.3 Calling the Driver - To enable the driver to access the parameter list, the program must set the first word of the driver to an address six bytes less than that of the word containing MEMORY START ADDRESS. It may then call the driver subroutine required directly by a normal JSR PC,xxxx call.

As an example, the following position-independent code might appear in a program which wishes to read Blocks #100-103 backward from DECtape unit 3 into a buffer starting at address BUFFER:

```
        MOV     PC,R0               ;GET TABLE ADDRESS
        ADD     #TABLE+12-.,R0
        MOV     PC,@R0              ;GET & STORE...
        ADD     #RETURN-.,@R0       ;...RETURN ADDRESS
        MOV     #5404,-(R0)         ;SET READ REV. UNIT 3
        MOV     #-1024.,-(R0)       ;4 BLOCKS REQUIRED
        MOV     PC,-(R0)            ;GET & STORE
        ADD     #BUFFER-.,@R0       ;...BUFFER ADDRESS
        MOV     #103,-R0)           ;START BLOCK
        CMP     -(R0),-(R0)         ;SUBTRACT 4 FROM POINTER
        MOV     R0,DT               ;SET DRIVER LINK
        JSR     PC,DT.TFR           ;GOTO TRANSFER ROUTINE
WAIT:                               ;RETURNS HERE WHEN
RETURN:                             ;...TRANSFER UNDERWAY
                                    ;RETURNS HERE WHEN
                                    ;...TRANSFER COMPLETE
TABLE:  .WORD 0                     ;LIST AREA SET
        .WORD 0                     ;...BY ABOVE SEQUENCE
        .WORD 0
        .WORD 0
        .WORD 0
```

H.3.1.4 User Registers - During its setup operations for the function requested, the driver assumes that Processor Registers 0-5 are freely available for its purpose. If their contents are of value, the program must save them before the driver is called.

While servicing intermediate interrupts, the driver may need to save or restore these registers. It expects to have available two subroutines for the purpose (provided by the Monitor under DOS). It accesses them via addresses in memory locations 44 (SAVE) and 46 (RESTORE) using the sequence:

```
        MOV     @# 44,-(SP)         ;OR 'MOV     @# 46,-(SP)
        JSR     R5,@(SP)+
```

The program must, therefore, contain these subroutines. They might, for example, be as follows:

```
SAVE:   MOV     R4,-(SP)        ;SAVE R0-4
        MOV     R3,-(SP)        ;...R5 SAVED BY CALL
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        MOV     R0,-(SP)
        MOV     R5,PC           ;EXIT TO CALLER

RESTOR: INC     (SP)+           ;FORGET CALL R5
        MOV     (SP)+,R0        ;RESTORE R0-4
        MOV     (SP)+,R1
        MOV     (SP)+,R2
        MOV     (SP)+,R3
        MOV     (SP)+,R4
        RTS     R5              ;R5  RESET ON EXIT
```

It must also ensure that their start addresses are set into the correct locations.

At its final interrupt, the driver always saves the contents of Registers 0-5 before returning control to the calling program completion return.

H.3.1.5 Returns From Driver - As shown in the example in Section H.3.1.3, the driver returns control to the calling program immediately after the JSR as soon as it has set the device in motion. The program may then wait or carry out some alternative operations until the driver signals completion by returning at the address supplied, i.e., RETURN above. Prior to this, the program should not attempt to access the data being read in, or to refill a buffer being written out.

The program routine beginning at address RETURN will vary according to the device in use. In general, the driver has given control to the routine for one of two reasons, namely, the function has been satisfactorily performed, or it cannot be carried out due to some hardware failure with which the driver is unable to cope, though the program may. If the latter, the driver uses the STATUS word in the program list to show the cause:

Bit 15 = 1	indicates that a device parity or timing failure has occurred and the driver has not been able to overcome this, perhaps after several attempts.
Bit 14 = 1	shows that the end of the data available has been reached.

The driver places in R0 the content of its first word as a pointer to the list concerned.

In addition, the driver may have transferred only some of the data required. In this case, it will show, in the RESERVED word of the program list, a negative count of the words not transferred in addition to setting Bit 14 of the STATUS. As mentioned in the note in Section H.3.1.2, this applies only to non-bulk storage devices. The drivers for DECtape or Disks[1] always endeavor to complete the full transfer, even beyond a parity failure, or they take more drastic action (see Section H.3.1.6).

It is thus the responsibility of the program RETURN routine to check the information supplied by the driver in order to verify that the transfer was satisfactory and to handle the error situations accordingly.

In addition, the routine must contain a sequence to take care of the Processor Stack, Registers, etc. As noted earlier, the driver takes the completion return address after an interrupt and has saved Registers 0-5 on the stack above the Interrupt Return Address and Status. The program routine should, therefore, contain some sequence to restore the processor to its state prior to such interrupt, e.g., using the same Restore subroutine illustrated earlier:

```
MOV     @#46,-(SP)          ;CALL REGISTER RESTORE
JSR     R5,@(SP)+
  .
  .
  .
RTI                         ;RETURN TO INTERRUPTED PROG.
```

H.3.1.6 Irrecoverable Errors - All hardware errors other than those noted in the previous paragraph are more serious in that they cannot normally be overcome by the program or the driver on its behalf. Some of these could be due to an operator fault, such as an omission to turn a paper tape reader on or to set the correct unit number on a DECtape transport. Once the operator has rectified the problem, the program could continue. Other errors, however, will require hardware repair or even software repair, e.g., if the program asks for Block 2000 on a device having a maximum of 1000. In general, all these errors will result in the driver placing identifying information on the processor stack and calling IOT to produce a trap through location 34.

Under DOS, the Monitor provides a routine which prints a teleprinter message when this occurs. In a stand-alone environment, the program using the driver must itself contain the routine to handle the trap (unless the user wishes to modify the driver error exits before assembly). The handler format will depend upon the program. Should it wish to take advantage of the information supplied by the driver, the format is as follows:

[1] This includes RF11 Disk; although this is basically word-oriented, it is assumed to be subdivided into 64-word blocks.

(SP):	Return Address	Stored by IOT Call
2 (SP):	Return Status	
4 (SP):	Error No. Code	generally unique to driver
5 (SP):	Error Type Code:	1 = Recoverable after Operator Action 3 = No recovery
6 (SP):	Additional Information	such as content of Driver, Control Register, Driver Identity, etc.

As a rule, the driver will expect a return following the IOT call in the case of errors in Type 1 but will contain no provision following a return from Type 3.

H.3.1.7 General Comment - The source language of each driver has been written for use with the DOS version of the Assembler which requires certain statements which will not be accepted by the Paper Tape Software PAL-11A, in particular: .TITLE & .GLOBL. These should be edited out before the source is used. Similarly, an entry in the driver table gives the device name as .RAD50 'DT' to obtain a specially packed format used internally by DOS. If the user wishes to keep the name, for instance for identification purposes as discussed in Section H.3.3, .RAD50 might easily be changed to .ASCII without detrimental effect, or it can be replaced with .WORD 0.

H.3.2 Drivers Assembled Separately

Rather than assemble the driver with every program requiring its availability, the user may wish to hold it in binary form and attach it to the program only when loaded. This is readily possible; the only requirement is that the start address of the driver should be known or can be determined by the program.

The example in Section H.3.1.2 showed that the Interrupt Servicing routine can be accessed through an offset stored in the Driver Table. The same technique can be used to call the setup subroutines, as these also have corresponding offsets in the Table, as follows:

DRIVER+7	Open[1]
+10	Transfer
+11	Close[1]
+12	Special Functions[1]

The problem, of course, is the start address. There is always the obvious solution, that of assembling the driver at a fixed location so that each program using it can immediately reference the location chosen. This, however, ceases to be convenient when the program itself has to avoid the area given to the driver. A more general method is to relocate the driver as dictated by the program using it, thus taking advantage of the position-independent nature of the driver. The Absolute Loader,

[1] If the routine is not provided, these are 0.

described in the Paper Tape Software Handbook (DEC-11-GGPB-D), Chapter 6, provides the capability of continuing a load from the point at which it ended. Using this facility to enter the driver immediately after the program, the program itself might contain the following code to call the subroutine to perform the transfer illustrated in Section H.3.1.3:

```
        MOV     PC,R1           ;GET DRIVER START ADDRESS
        ADD     #PRGEND-.,R1
        MOV     PC,R0           ;GET TABLE ADDRESS
        ADD     #TABLE+12-.,R0  ;& SET UP AS SHOWN
          .                     ;...IN SECTION H.3.1.3
          .
          .
        CMP     -(R0),-(R0)     ;FINAL POINTER ADJUSTMENT
        MOV     R0,@R1          ;STORE IN DRIVER LINK
        CLR     -(SP)           ;GET BYTE SHOWING...
        MOVB    10(R1),@SP      ;...TRANSFER OFFSET
        ADD     (SP)+,R1        ;COMPUTE ADDRESS
        JSR     PC,@R1          ;GO TO DRIVER
          .
          .
          .
PRGEND:
        .END
```

This technique may be extended to cover situations in which several drivers are used by the same program, provided that it takes account of the size of each driver (this being already known because of prior assembly) and that the drivers themselves are always loaded in the same order.

For example, to access the second driver, the above sequence would be modified to:

```
        MOV     PC,R1           ;GET DRIVER 1 ADDRESS
        ADD     #PRGEND-.,R1
        ADD     #DVR1SZ,R1      ;STEP TO DRIVER 2
          .
          .
          .
DVR1SZ=
PRGEND:
        .END
```

An alternative method may be to use the Relocatable Assembler PAL-11S in association with the Linker program LINK-11S, both of which are available through the DECUS Library. The start address of each driver is identified as a global. Any calling program need, therefore, merely include a corresponding .GLOBL statement, e.g., .GLOBL DT.

H.3.3 Device-Independent Usage

As mentioned earlier, the drivers are designed for use in a device-independent environment, i.e., one in which a calling program need not know in advance which driver has been associated with a table for a particular execution run. One application of this type might be to allow line printer output to be diverted to some other output medium because the line printer itself is not currently available. Another might be to provide a general program to analyze data samples although these on one occasion might come directly from an Analog-to-Digital converter and on another be stored on a DECtape, because the sampling rate was too high to allow immediate evaluation.

As a rule, programs of this type should be written to cater for all the facilities that any one device might offer, but not necessarily all of them. For instance, the program should ask for start-up procedures because it may sometime use a paper tape punch which provides them, even though it may normally use DECtape which does not. As noted in Section H.2.1.1, the driver table contains an indication of its capabilities to cater for this situation. The program can thus examine the appropriate item before calling the driver to perform some action. As an example, the code to request start-up procedures might be (assuming R0 already set to List Address):

```
        MOV     #DVRADD,R1      ;GET DRIVER ADDRESS
        TSTB    2 (R1)          ;BIT 7 SHOWS...
        BPL     NOOPEN          ;...OPEN ROUTINE PRESENT
        MOV     R0,@R1          ;STORE TABLE ADDRESS
        CLRB    -(SP)           ;BUILD ADDRESS
        MOVB    7(R1),@SP       ;...OF THIS ROUTINE
        ADD     (SP)+,R1
        JSR     PC,CR1          ;...& GO TO IT
                                ;FOLLOWED POSSIBLY BY
                                ;WAIT AND COMPLETION
                                ;PROCESSING
NOOPEN:                         ;RETURN TO COMMON OPERATION
```

Similarly, the indicators show whether the device is capable of performing input or output or both, whether it can handle ASCII data or binary data, whether it is a bulk storage device capable of supporting a directory structure or is a terminal-type device requiring special treatment and so on. Other table entries show the device name as identification and how many words it might normally expect to transfer at a time (in 16-word units). All of the information may readily be examined by the calling program, thus enabling the use perhaps of a common call sequence for any I/O operation, as for example:

```
        MOV     #DVRADR, R5     ;SET DRIVER START
        JSR     R5, IOSUB       ;CALL SET UP SUB
        BR      WAIT            ;SKIP TABLE FOLLOWING ON RETURN
        .WORD   10              ;TRANSFER REQUIRED
        .WORD   103             ;BLOCK NO.
        .WORD   BUFFER          ;BUFFER ADDRESS
        .WORD   -256.           ;WORD COUNT
        .WORD   404             ;READ FROM UNIT 1
        .WORD   RETURN          ;EXIT ON COMPLETION
        .WORD   0               ;RESERVED
WAIT:                           ;CONTINUE HERE...
        .                       ;WHILE TRANSFER IN PROGRESS
        .
        .
        .
IOSUB:  MOV     @SP,R0          ;PICK UP DRIVER ADDR
        MOV     R5,R1           ;SET POINTER TO LIST
        TST     (R1)+           ;BUMP TO COLLECT CONTENT
        .                       ;ROUTINE CHECKS ON DEVICE..
        .                       ;..CAPABILITY USING R1
        .                       ;..TO ACCESS LIST &
        .                       ;..R0 THE DRIVER TABLE
        .                       ;IF O.K...
        MOV     @R1,R1          ;GET ROUTINE OFFSET
        ADD     R0,R1
        CLR     -(SP)           ;USE IT TO BUILD
        MOVB    @R1,@SP         ;...ENTRY POINT
        ADD     R0,@SP
        JSR     PC,@(SP)+       ;CALL DRIVER
        RTS     R5              ;EXIT TO CALLER
```

The calling program, or a subroutine of the type just illustrated, may also wish to take advantage of a further feature mentioned earlier: the fact that when a driver is already occupied its first word must be nonzero. The driver itself does not clear this word except in special cases shown in the description for the driver concerned. If the program itself always ensures that it is set to zero between driver tasks, this word forms a suitable driver-busy flag. Under DOS, in fact, the program parameter list is extended to allow additional words to provide linkage between lists as a queue of which the list indicated in the driver first word is the first link.

The preceding paragraphs are intended merely to indicate possible ways of incorporating the drivers available into the type of environment for which they were designed. The user will probably find others. However, he should read carefully the more detailed description of the driver structure in Appendix G and the individual driver specifications before determining the final form of his program.

In particular, one general word of warning is appropriate here. Although most drivers normally set up an operation and then wait for an interrupt to produce a completion state, there are some cases in which the driver can finish its required task without an interrupt, e.g., "opening" a paper tape reader involves only a check on its status. Moreover, where "Special Functions" are concerned, the

driver routine may determine from the code indicated that the function is not applicable in its case and will, therefore, have nothing to do. In those cases, the driver clears the intermediate return address from the processor stack and takes the completion return immediately. Special problems may arise, however, if the driver concerned may be covering several tasks, any of which may cause a queue for the driver's services under DOS. To overcome these problems, the driver expects to be able to refer to flags outside the scope of the list described so far. This may mean that a program using such a driver may also need to extend the list range to cover this possibility. Extreme care will then be needed.

APPENDIX I
GLOSSARY AND ABBREVIATIONS

Bit Map	A table describing the availability of space. Each bit in the table indicates the state (occupied or free) or one segment of storage, for example a block on a bulk storage device.
Buffer	A storage area
Buffer Use Table	A bit map in the permanently resident monitor, which describes the availability of buffers in the free core area.
Contiguous File	A file consisting of physically contiguous blocks on a bulk storage device.
Core Bit Map	That portion of a Permanent Bit Map which happens to be in core. Not to be confused with the Buffer Use Table.
Core Image	A copy of what a program or other data would look like if it were in core.
CSI	Command String Interpreter.
DAT	Device Assignment Table.
Dataset	A logical collection of data which is treated as an entity by a program.
DDB	Dataset Data Block.
Default Device	The device specified in the Link Block of a dataset, and which is used for I/O operations on that dataset if there is no other device assigned in a DAT entry for the dataset.
Device Driver	The minimal routine which controls physical hardware activities on a peripheral device. The device driver is the interface between a device and the common, device-independent I/O code in the monitor.
Fatal Error	An error from which a user's program cannot recover.
File	A physical collection of data which resides on a directory-structured device and is referenced through its name.
FBM	File Bit Map - A device-resident bit map with bits flagged for the blocks used for a single file. Used on DECtape to aid in the deletion process.
FIB	File Information Block

File Structure	The manner in which files are organized on a bulk storage device. Each of the files of a user is referenced through an entry in that user's User File Directory. Each User File Directory on the device is, in turn, referenced through an entry in the Master File Directory.
Interleave Factor	The optimal minimum distance, measured in number of physical device blocks, between logically adjacent blocks of a linked file. Presently it is four on all PDP-11 bulk storage devices. For example, if physical block N is assigned to block 1 of a linked file, then physical block N+4 would be the closest device block that could be assigned to block 2 of that file.
Julian Date	A 5-digit (decimal) numerical representation of the date, in which the two high-order digits give the year (1900=00, 1999=99) and the three low-order digits give the day within the year (January 1 = 001, December 31 = 365 (366 for leap year)). For example, January 28, 1971 is represented as 71028.
KSB	Keyboard Swap Buffer
Linked File	A file consisting of a set of blocks within which an ordering is specified through the use of a link word imbedded within each block.
Linker	A systems program which creates a load module to be loaded into core memory. The linker relocates and links internal and external symbols to provide communication between independently assembled programs.
Load Module	The output of the linker. A program in absolute binary form ready for loading and executing on a PDP-11.
MFD	Master File Directory
MRT	Monitor Residency Table
MSB	Monitor Swap Buffer
Object Module	The relocatable binary output of an assembler or compiler.
Operator	A user communicating directly with the Monitor through the keyboard.
Parity Bit	A binary digit appended to an array of bits to make the sum of all the bit values always odd or always even.
PBM	Permanent Bit Map - A bit map which describes the availability of space on a DECtape or disk. It resides on the device it describes, and can be read into core in segments, called Core Bit Maps, for reference or updating.
Radix-50 packed ASCII	A format in which 3 ASCII characters (from a subset of all ASCII characters) are packed into a single 16-bit word.
SAM	Swap Area Manager
SAL	A friend of SAM.

Swapping	The movement of programs or program sections from secondary storage to core.
Table	A collection of data in a form suitable for ready reference.
UFD	User File Directory.
UIC	User Identification Code
User	The person who is using the Monitor. He may use the Monitor as an operator, or via a program.
User Program	Any program written by a user to run under the Monitor.

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 & PDP-12
Digital Software News for the PDP-11
Digital Software News for the PDP-9/15 Family

These newsletters contain information applicable to software available from Digital's Program Library, Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's Software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problem to:

Software Information Service
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

These forms which are provided in the software kit should be fully filled out and accompanied by teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software and manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. U.S.A. customers may order directly from the Program Library in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

PDP-11 Disk Operating System Monitor
Programmer's Handbook
DEC-11-MWDC-D
February 1972

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — Fold Here — — — — — — — — — — — — — — — — — —

— — — — — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

digital equipment corporation

PREFACE

This document contains a comprehensive description of the PDP-11 Disk Operating System Monitor. The document is written for the PDP-11 programmer -- it assumes familiarity with the contents of the PDP-11 Handbook 1971 and the PAL-11R Assembler (see document number DEC-11-ASDB-D). Previous experience with monitor or executive systems would be helpful.

The document is separated into three chapters: Chapter 1 is an introduction to the DOS Monitor, and provides general information about the disk operating system. Chapter 2 describes the programmed requests that are available to the programmer through the Monitor. This chapter also explains the concepts and operation of each programmed request. Chapter 3 describes the keyboard commands available to the system operator through the Monitor; concepts and operation of each command are also explained. The entire document is summarized in the appendices. Appendices D (Monitor Commands) and E (Monitor Programmed Requests) should prove to be invaluable to the DOS user.

In addition to the DOS Monitor, the PDP-11 Disk Operating System software includes:

FORTRAN IV
PAL-11R Assembler
Edit-11 Text Editor
ODT-11R Debugging Program
PIP, File Utility Package
Link-11 Linker
Libr-11 Librarian

SOFTWARE SUPPORT CATEGORIES

Digital Equipment Corporation (DEC) makes available four categories of software. These categories reflect the types of support a customer may expect from DEC for a specified software product. DEC reserves the right to change the category of a software product at any time. The four categories are as follows:

CATEGORY I
Software Products Supported at no Charge

This classification includes current versions of monitors, programming languages, and support programs provided by DEC. DEC will provide installation (when applicable), advisory, and remedial support at no charge. These services are limited to original purchasers of DEC computer systems who have the requisite DEC equipment and software products.

At the option of DEC, a software product may be recategorized from Category I to Category II for a particular customer if the software product has been modified by the customer or a third party.

CATEGORY II
Software Products that Receive Support for a Fee

This category includes prior versions of Category I programs and all other programs available from DEC for which support is given. Programming assistance (additional support), as available, will be provided on these DEC programs and non-DEC programs when used in conjunction with these DEC programs and equipment supplied by DEC.

CATEGORY III
Pre-Release Software

DEC may elect to release certain software products to customers in order to facilitate final testing and/or customer familiarization. In this event, DEC will limit the use of such pre-release software to internal, non-competitive applications. Category III software is only supported by DEC where this support is consistent with evaluation of the software product. While DEC will be grateful for the reporting of any criticism and suggestions pertaining to a pre-release, there exists no commitment to respond to these reports.

CATEGORY IV
Non-Supported Software

This category includes all programs for which no support is given.

CONTENTS

CONTENTS (Cont)

APPENDICES

ASCII Modes (Cont)
- Formatted ASCII Nonparity - Normal
- Unformatted ASCII Parity - Normal
- Unformatted ASCII Nonparity - Normal

Binary Modes:
- Formatted Binary - Special
- Formatted Binary - Normal
- Unformatted Binary - Normal

1. Formatted and Unformatted ASCII Modes:

   Data in formatted ASCII modes is assumed by the Monitor to be in strings of 7- or 8-bit ASCII characters terminated by LINE FEED, FORM FEED or VERTICAL TAB. PAL-11R Assembler source programs are in a formatted ASCII mode. In these modes, the Monitor manages all device-dependent conversions at the driver level. For example, LINE FEED is supplied after RETURN in character strings from keyboard terminals.

   Data in unformatted ASCII modes is also assumed to be in strings of 7- or 8-bit ASCII characters. Checks for terminators and device-dependent conversion are not performed by the Monitor, thus allowing the user to handle all characters in his own way.

2. ASCII Parity and Nonparity Modes:

   In ASCII nonparity modes, 7-bit ASCII characters are transferred.

   In formatted ASCII parity modes, even parity is generated in the 8th bit and is checked during the transfer. In unformatted ASCII parity mode, 8 bits are transferred, but no parity is generated or checked.

3. Normal and Special Modes:

   Special modes provide additional Monitor facilities over and above normal modes; normal modes are compatible with the PDP-11 I/O Executive (IOX).

4. Formatted and Unformatted Binary Modes:

   Data in formatted binary modes is transferred in 8-bit bytes as read from the device. The Monitor makes no assumptions about the nature of the data. A checksum is calculated during a WRITE request and transmitted with the data, as well as a count of the number of bytes. The checksum is verified during a READ. The binary output of the PAL-11R Assembler, for example, is in a formatted binary mode.

   Unformatted binary mode is the same as formatted binary except that no checksum or count is calculated or verified.

NOTE

A dataset can only support transfers in one direction, i.e., READ only or WRITE only. If the same device is to be used for both operations, separate datasets must be used for each.

2.6.1.1 .INIT - Associate or initialize a dataset with a device driver and set up the initial linkage between them.

Macro Call: .INIT LNKBLK

where LNKBLK is the address of the Link Block.

Assembly Language

Expansion:
```
MOV #LNKBLK,-(SP)
EMT 6
```

Global Name: INR.

Description: Assigns a device to a dataset and makes sure that the appropriate driver exists and is in core. If the driver is not in core, it is swapped in. The device assigned is that specified in the associated Link Block, unless assignment has been made to the logical name specified in the Link Block with the ASsign command. After the INIT has been completed, control is returned to the user at the instruction following the assembly language expansion. The argument is removed from the stack.

Rules: The user must set up a Link Block of the format shown in Figure 2-5 in his program for each dataset to be INITed. Another .INIT on a dataset for which no .RLSE has been given will effectively be a .RLSE followed by an .INIT except that no form of close will be performed.

Errors: A non-fatal error message, A003, is printed on the teleprinter if no assignment has been made through the ASsign command, and the DEFAULT DEVICE is either not specified in the Link Block or has been specified illegally (i.e., no such device on the system). The user may type in an assignment (ASsign) and give the command CO (continue) to resume operation.

Control is transferred to the address specified by the Link Block if at any time during an operation there is not enough space in free core for the necessary drivers, buffers, or tables. If no address (i.e., a zero) is specified in the Link Block's ERROR RETURN ADDRESS, a fatal error (F007) is printed and the program stops.

Example: (see .RLSE).

2.6.1.2 .RLSE - Remove the linkage between a device driver and a dataset, and release the driver.

Macro Call: .RLSE LNKBLK

where LNKBLK is the address of the Link Block previously INITed.

Assembly Language

Expansion:
```
MOV #LNKBLK,-(SP)
EMT 7
```

Global Name: RLS.

Description: Dissociates the device from the dataset and releases the dataset's claim to the driver. Releasing the driver frees core provided no other dataset has claimed the driver.

Rules: The device to be released must have been previously INITed to the dataset.

If the dataset has been opened on a directory device, it must be closed before the device is released. On a nondirectory device, RLS will ensure that any data remaining in the Monitor buffer for output is dispatched to the device and will return any buffer still associated with the dataset to free core.

After the release has been completed, control is returned to the user at the instruction following the assembly language expansion; the argument is removed from the stack.

Errors: If the dataset has been .OPENed to a file-structured device, a .RLSE not preceded by a .CLOSE will be treated as a fatal error, F005. A .RLSE error (F005) may also occur if dataset link is invalid (DDB or driver not correct).

Example:

```
            .
            .
            .
        .INIT   LNK1        ;ASSOCIATE A DATASET WITH A DEVICE
            .
            .
            .
        .RLSE   LNK1
            .
            .
            .
        .WORD   ERR1        ;ERROR RETURN ADDRESS
LNK1:   .WORD   0           ;POINTER FOR MONITOR
        .RAD50  /DSI/       ;LOGICAL NAME OF DATASET
        .BYTE   1, 0        ;DEVICE SPECIFIED, UNIT
        .RAD50  /KB/        ;SPECIFY KEYBOARD
            .
            .
            .
ERR1:   ERROR
        PROCESSING
```

2.6.1.3 .OPEN - Prepare .INITed device for usage and make a named file available if the device is directory oriented.

Macro Call: .OPENx LNKBLK,FILBLK

where x indicates the type of OPEN:

U for update
O for output
E for extension
I for input
C for create data in contiguous file
LNKBLK = address of Link Block
FILBLK = address of Filename Block

Assembly Language

Expansion:

```
MOVB #CODE,FILBLK-2          ;MOVE "HOW OPEN"
                             ;CODE TO FILENAME BLOCK
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  16
```

where CODE indicates the type of OPENx request:

OPEN0 = 2
OPENI = 4
OPENU = 1
OPENC = 13
OPENE = 3

Global Name: OPN. (See Appendix C for subsidiary routines.)

Description: In general, an .OPENx request causes the Monitor to allocate a data buffer and to make other necessary preparations for transferring to a dataset to or from a device. More specifically:

.OPENU	opens a previously created contiguous file for input and output by .BLOCK.
.OPENO	creates a new linked file and prepares it to receive output.
.OPENE	opens a previously created linked file to make it longer.
.OPENI	opens a previously created linked or contiguous file for input to the computer. It normally precedes all .READ operations.
.OPENC	opens a previously created contiguous file for output from the computer.

2.7.4 .APPEND

Append one linked file onto another.

Macro Call: .APPEND LNKBLK,FIRST,SECOND

where LNKBLK is the address of the Link Block, FIRST is the address of the Filename Block for the first file, and SECOND is the address of the Filename Block for the second file.

Assembly Language

Expansion:
```
MOV  #SECOND,-(SP)
MOV  #FIRST,-(SP)
MOV  #LNKBLK,-(SP)
EMT  2
```

Global Name: APP. (See Appendix C for subsidiary routines.)

Description: Makes one linked file out of two by appending the SECOND to the FIRST. The directory entry of the SECOND file is deleted. When the request is completed, control is returned to the user at the instruction following the assembly language expansion. The arguments are removed from the stack. No attempt is made to pack the two files together, the physical blocks are merely linked together.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the offending Filename Block if it is specified, or to the console for an error message if it is not. Possible errors resulting from .APPEND are:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	None	F000
First File Nonexistent	2	F024
Contiguous File	5	F024
Device Not Ready	None	A002
Protect Code Violated	6	F024
File Opened	14	F024

2.7.5 .LOOK

Search the directory for a particular file name.

Macro Call: .LOOK LNKBLK,FILBLK (,1)

where LNKBLK is the address of the Link Block, and FILBLK is the address of the Filename Block.

Assembly Language

Expansion:

a. If the optional argument is not specified:

```
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  14
```

b. If the optional argument is specified:

```
MOV  #FILBLK,-(SP)
CLR  -(SP)
MOV  #LNKBLK,-(SP)
EMT  14
```

Global Name: DIR. (See Appendix C for subsidiary routines.)

Description: The primary purpose of this routine is to search through a specified directory for a specified file and return with the current parameters of the file. However, this routine can also be used to return the characteristics of a non-directory device. By specifying the optional argument, the user can indicate whether he requires two or three parameters returned.

The device searched is specified through the Link Block, and the file through the Filename Block. The request returns to the user with the top elements of the stack as follows.

	2 Arg. Call	3 Arg. Call
START BLOCK		SP
# OF BLOCKS	SP	SP+2
INDICATOR WORD	SP+2	SP+4

where # OF BLOCKS is the number of binary blocks in the file, and the INDICATOR WORD is coded as follows:

INDICATOR WORD		
bit 0 = 1	...	OPENC allowed
bit 1 = 1	...	OPENI allowed
bit 2 = 1	...	OPENE allowed
bit 3 = 1	...	OPENU allowed
bit 4 = 0	...	file is not in use
= 1	...	file is being used by another dataset
bit 5 is 1	...	dataset already has a file open (no search has been performed)
bit 6 = 0	...	file is linked
= 1	...	file is contiguous
bit 7 = 0	...	file does not exist (OPENO allowed)
= 1	...	file exists
bits 15-8	...	protection code

After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The stack must be cleared by the user. If a file is protected against READ access, it will be signaled as nonexistent to a caller other than the owner.

Rules: The dataset must be INITed.

Errors: Control is returned either to the ERROR RETURN ADDRESS in the Filename Block if it is specified, or the console for an error message if it is not. Possible errors resulting from .LOOK are:

Error Condition	Error Code Returned To Filename Block	Error Message
Device Not Ready	None	A002
A File Is Open on Requesting Dataset	14	F024
Illegal File Name	15	F024

2.7.6 .KEEP

Protect file from automatic deletion.

Macro Call: .KEEP LNKBLK,FILBLK

where FILBLK is the address of the Filename Block of the file to be protected.

Assembly Language

Expansion:
```
MOV  #FILBLK,-(SP)
MOV  #LNKBLK,-(SP)
EMT  24
```

Global Name: PRO.

Description: Protects the named file from being deleted by the Monitor upon a FInish command (Section 2.3.5.5). It does this by setting bit 7 of the PROTECT byte in the Filename Block.

APPENDIX A
PHYSICAL DEVICE NAMES

Mnemonic	Device	Radix-50 Equivalence
DC	RC11 Disk	014570
DF	RF11 Disk	014760
DK	RK11 Disk	015270
DT	DECtape (TC11)	016040
KB	ASR-33 Keyboard/Teletype	042420
LP	Line Printer (LP11)	046600
MT	Magtape (TM11)	052140
PP	High-Speed Paper Tape Punch	063200
PR	High-Speed Paper Tape Reader	063320
PT	ASR-33 Paper Tape Device	063440
CR	Card Reader (CR11)	012620

NOTE

a. Device mnemonics may be three letters on a particular system. The third letter is assigned if there is more than one controller, e.g.:

DTA for DECtape controller "A"
DTB for DECtape controller "B"

b. The device name may be followed by an octal number to identify a particular unit when the controller has several device units associated with it, e.g.:

DT1 indicates unit 1 under a single DECtape control.

DTA1 indicates unit 1 under controller A in a multicontrol situation.

APPENDIX B
EMT CODES

Code	Usage	Described on Page
0	.WAITR	2-24
1	.WAIT	2-23
2	.WRITE	2-22
3	**	
4	.READ	2-21
5	**	
6	.INIT	2-14
7	.RLSE	2-15
10	.TRAN	2-27
11	.BLOCK	2-25
12	.SPEC	2-29
13	.STAT	2-30
14	.LOOK	2-38
15	.ALLOC	2-32
16	.OPENx	2-16
17	.CLOSE	2-19
20	.RENAM	2-35
21	.DELET	2-34
22	.APPND	2-37
24	.KEEP	2-40
25-27	**	
30-31	*	
32	Diagnostic Print	2-75, -78
33,34	*	
35-37	**	
40	*	
41	General Utilities	2-42, -50
42	General Conversions	2-51, -57
43-55	*	
56,57	Command String Interpreter	2-58, -61
60	.EXIT	2-41
61-63	*	
64-77	**	
100-117	(reserved for Communications Executive, COMTEX-11)	
120-137	(reserved for Real Time Monitor, RSX-11)	
140-167	(reserved for user-implemented routines)	

*Reserved for Monitor internal communication
**Reserved for future Monitor expansion