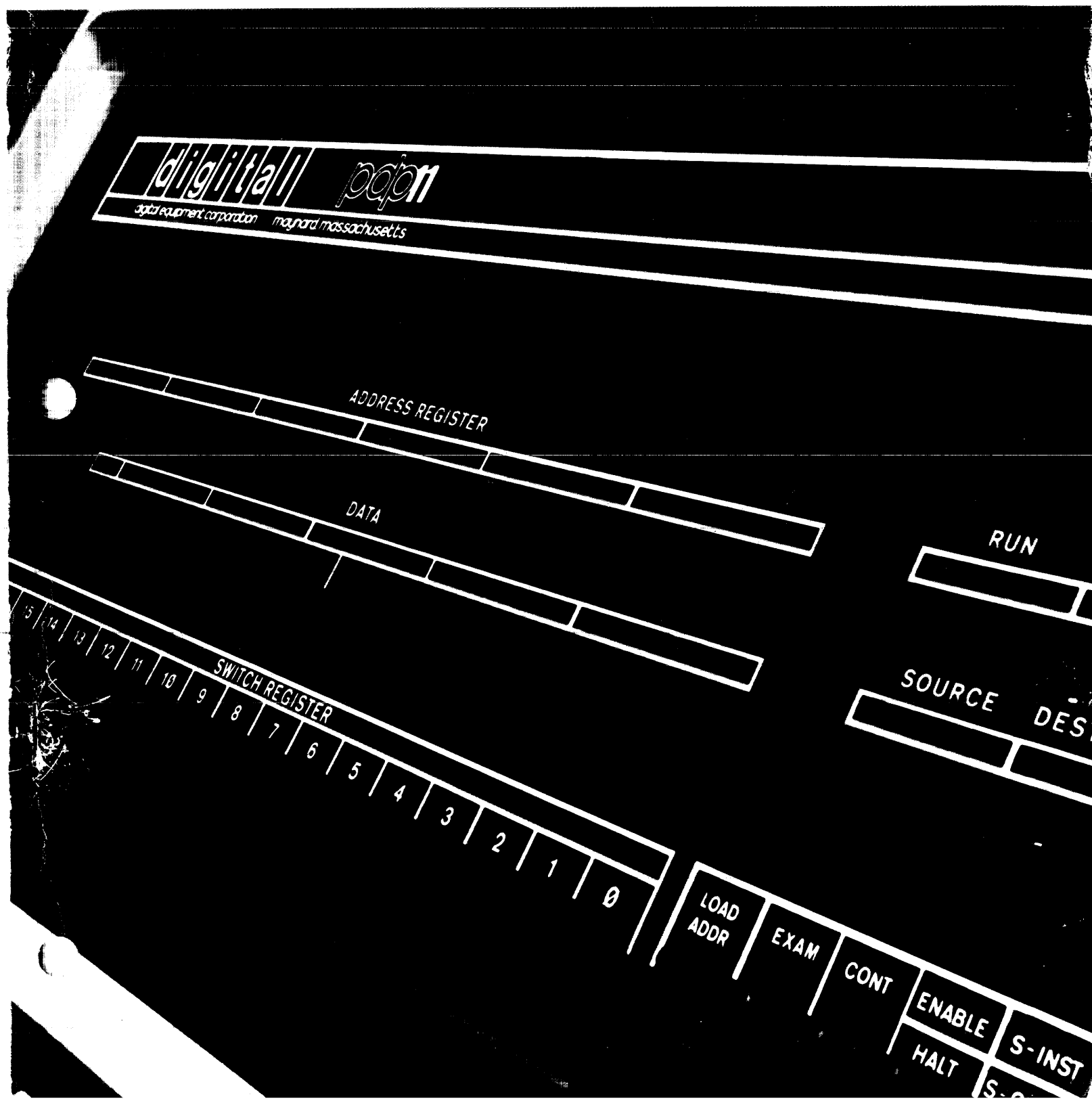
digital

pdp11

disk operating system monitor

programmer's handbook



DEC-11-SERA-D

P D P - 1 1

D I S K O P E R A T I N G S Y S T E M

(DOS)

M O N I T O R

PROGRAMMER'S HANDBOOK

For additional copies, order No. DEC-11-SERA-D from Digital Equipment Corporation, Direct Mail, Bldg. 6A-3, Maynard, Massachusetts 01754

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

First Printing, February 1971

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 only, and describes the first release of the Disk Operating System Monitor. It is subject to change without notice.

Associated Documents:

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

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

PDP-11 ODT-11R Debugging Program,
Programmer's Manual, DEC-11-ODDA-D

PDP-11 Link-11 Linker,
Programmer's Manual, DEC-11-LLDA-D

PDP-11 PIP, File Utility Package,
Programmer's Manual, DEC-11-PIPA-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-ASDA-D). Previous experience with monitor or executive systems would be helpful, but is not necessary.

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 is also explained.

The entire document is summarized in the appendixes. Appendixes D (Monitor Commands) and E (Monitor Programmed Requests) should prove to be invaluable to the DOS user.

The PDP-11 Disk Operating System software consist of, in addition to the DOS Monitor:

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

C O N T E N T S

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

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 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 the 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 Monitor.

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

TABLE 1-1

PDP-11 DOS MONITOR FEATURES AND BENEFITS

Feature	Benefits to User
Files are catalogued in multi-level 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 cheaply use the logical pieces of the Monitor for his own needs. The user 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.
All I/O is interrupt driven.	Such specialized equipment as communications modems and A/D converters which must be interrupt driven can be run through the Monitor. Several I/O calls can be handled concurrently.

(continued on next page)

Table 1-1 (continued)

Feature	Benefits to User
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 Datasets.	User may reassign a device which is used for one purpose, (Dataset) without changing its assignment for all other purposes (Datasets).

Programmed requests are macros which are assembled into the user's program and through which the programmer 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 to 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 under the DOS,and hence accessible through the Monitor, are listed in Table 1-2.

TABLE 1-2. THE DOS SYSTEM PROGRAMS.

System Program	Document Number
PAL-11R Assembler	DEC-11-ASDA-D
Edit-11 Text Editor	DEC-11-EDDA-D
ODT-11R Debugging Program	DEC-11-ODDA-D
PIP, File Utility Package	DEC-11-PIPA-D
Link-11 Program Linker	DEC-11-LLDA-D

1.2 MONITOR CORE ORGANIZATION

Core is divided into:

- a user area where user programs and buffers 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 shows a map of core as organized by the Monitor.

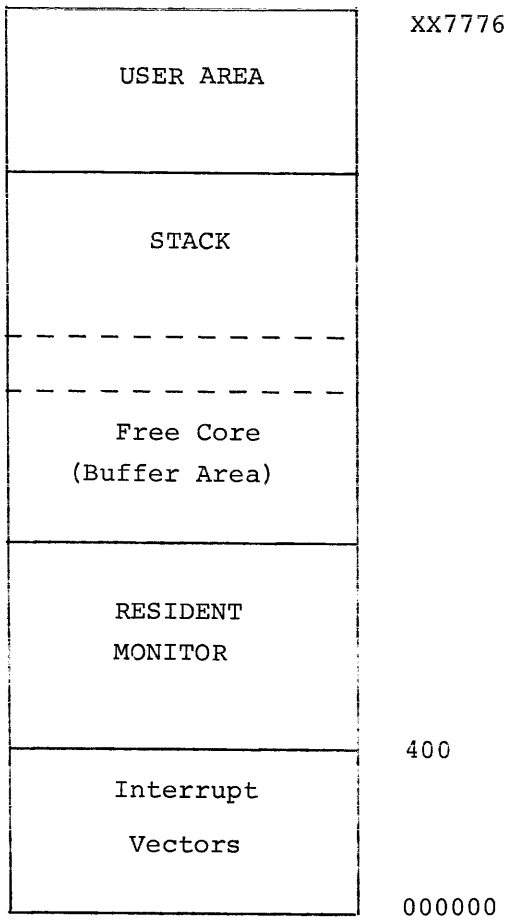


Figure 1-1. The Monitor Core Map.

1.3 HARDWARE CONFIGURATIONS

The smallest configurations adequate 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

1.4 MONITOR MESSAGES

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

There are four types of Monitor messages:

- Informational
- Action required by the operator
- Warning to the operator
- Fatal.

These message types are identified with 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.

The user may also store messages on the disk and call the appropriate Monitor routines to print them. The procedure for doing this is covered in Section D.

1.5 HOW TO START THE MONITOR

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

1. Place the address of the ROM Bootstrap Loader into the switches (773100)
2. Depress Load Address switch
3. Place the address of the word count register for the disk upon which the monitor resides (usually 177462 for RF/FS11)
4. Depress START switch.

The monitor will be loaded into core and identify itself by printing:

```
MONITOR  Vxxxx
```

on the teleprinter, were Vxxxx represents the version number of the Monitor being used. The Monitor is now ready to accept on 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 Disk Operating System. An expanded glossary, with abbreviations, can be found in Appendix G.

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 phusical collection of data which resides on a directory device and is referenced through its name. A file consists of one or more blocks on a directory device.

A block is a broup of adjacent words of a specified size; it is the smallest addressable segment. 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, VERTICAL TAB, or FORM FEED.

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.

---

*ASCII stands for American Standard Code for Information Interchange.

Bulk storage devices containing directories are called directory devices or file-structured devices. Devices such as paper tape equipment and the Teletype,* 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. Figure 1-2 shows how a typical table is represented in this manual. 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 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:

TABL:

ENTRY A	
ENTRY C	ENTRY B

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.

*Teletype is a registered trademark of the Teletype Corporation.

CHAPTER 2

PROGRAMMED MONITOR 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 at execution time by the Monitor. 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 PDP-11 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 filename and returns information about the file.
.KEEP	Protects a file against automatic deletion on Finish.
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.

(continued on next page)

Mnemonic	Purpose
.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.
.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 MONITOR REQUESTS

2.2.1 Requests for Input/Output and Related Services

Input/Output (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.

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. It 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.

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 - Special

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

1. Formatted and Unformatted Modes: Data in formatted ASCII modes is assumed by the Monitor to be in strings of ASCII characters terminated by LINE FEED, FORM FEED, or vertical TAB. DOS Assembler source programs use the formatted ASCII mode. In these modes, the Monitor manages all device-dependent conversions such as converting a TAB character to spaces when it is output to the line printer.

Data in unformatted ASCII modes is also assumed to be in strings of ASCII characters, but the Monitor does not check for terminators or make device-dependent conversions.

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: All normal modes are compatible with their counterparts in the PDP-11 I/O Executive (IOX). Special modes provide additional facilities to these normal modes.

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, but calculates and verifies a checksum and byte-count. The binary output of the DOS Assembler 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.

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.

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 non-directory 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.

There are several ways that a dataset can be opened. It can be opened for input, for output, for update, or for extension. The last letter of the .OPENx request specifies which type of open desired a

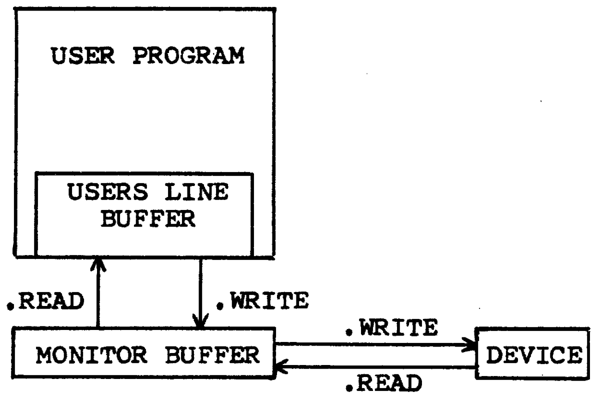


Figure 2-1a. The Transfer Path.

```
      .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
               |
         (Process Data)
               |
    No        End
            Of Data
               ?
               |   Yes
      .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.

.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 the data has all 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 dissociates 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 in directory devices such as the 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 his 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.

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.

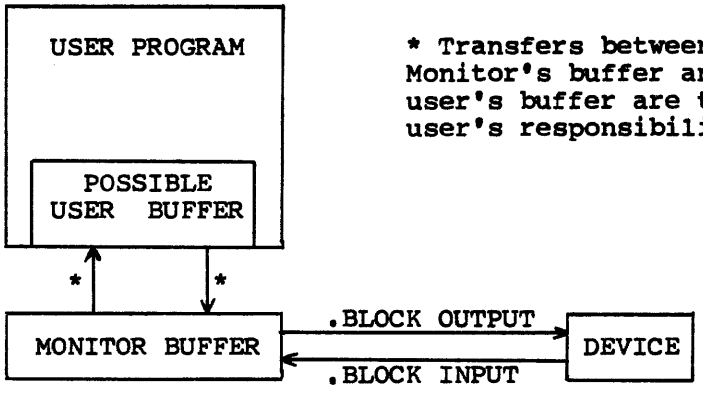


Figure 2-2a. The Transfer Path.

```
.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

.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.

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:

- when the file structure does not allow the desired operation (for example, PIP uses TRAN to read a directory block)
- 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	Non-File-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 earlier are observed.

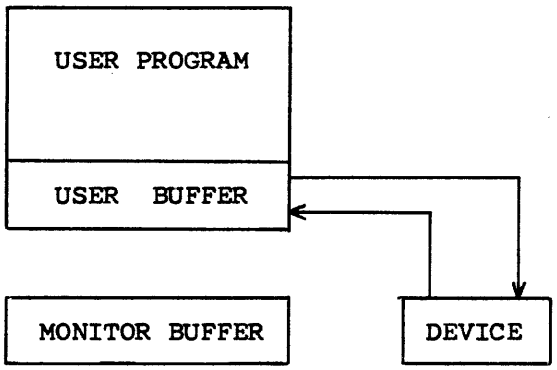


Figure 2-3a. The Transfer Path.

```
.INIT LNKBLK
   |
.ALLOC LNKBLK,FILBLK,N    ;MAKE SPACE AVAILABLE FOR TRAN
   |
.TRAN LNKBLK,TRNBLK
   |
.WAIT LNKBLK              ;WOULD BE REPLACED BY .WAITR
   |
  (Process Data)
   |
.RLSE LNKBLK

LNKBLK:     (entries)   ]
FILBLK:     (entries)   } Tables and parameters
N:          (entries)   } in User Program
TRNBLK:     (entries)   ]
```

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 data, the time, and the current user's UIC.
- Requests to perform conversions between ASCII and Radax-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.

Note of course 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 non-file structured devices.

It is important to note that a device is assigned in a program to a dataset logical name and that reassigning 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 listed in Appendix E.

- 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 .GLOBL assembler directive in the program. For example, to make the .READ processor resident while program FROP is being run, the following directive would be included in the program FROP:

```
.GLOBL RWN.
```

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

2.5 MONITOR RESTRICTIONS ON THE PROGRAMMER

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

- The programmer should not use either the EMT or the IOT instructions for communication within his program.
- It is recommended that the user does 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 because 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
	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 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.

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 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, number AØØ3, 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 type 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, then a fatal error, number FØØ7, 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 previously been INITed to the dataset.

If the dataset has been opened on a directory device, it must be closed before the device is released.

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.

Example:

```
                .
                .
                .
        .INIT LNK1          ;ASSOCIATE A DATASET WITH A DEVICE
                .
                .
                .
        .RLSE LNK1
                .
                .
                .
        .WORD ERR1          ;ERROR RETURN ADDRESS
LNK1:   .WORD Ø             ;POINTER FOR MONITOR
        .RAD5Ø /DSI/        ;LOGICAL NAME OF DATASET
        .BYTE 1,Ø           ;DEVICE SPECIFIED, UNIT
        .RAD5Ø /KB/         ;SPECIFY KEYBOARD
                .
                .
                .
ERR1:   ERROR
        PROCESSING
```

2.6.1.3 .OPEN

Prepares .INITed device for usage and makes a named file available if the device is directory oriented.

Macro Call: .OPENx LNKBLK,FILBLK

Where x identifies type of OPEN.

U for update
O for output
E for extension
I for input
C for contiguous create
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: General Rules for All .OPENx Requests

a. 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, Ø-9); the first character cannot be a digit (Ø-9). If the dataset is not a file, the Filename Block need not contain any FILENAME or EXTENSION entries.

b. All datasets must have been INITed before they are OPENed.

c. 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.

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

e. If a dataset is OPENed for any output, it cannot be OPENed again until it has been CLOSEd.

Rules for .OPENO

a. On a directory device, the .OPENO request is applicable only on outputs to a linked file or a non-directory device.

b. 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.

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

Rules for .OPENI

a. .OPENI may be used for inputs from contiguous or linked files, or non-directory devices.

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

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

d. At any one time, a file can be opened for input a maximum of 62_{10} or 76_{8} times.

Rules for .OPENU, .OPENE and .OPENC

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

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

c. A contiguous file cannot be opened for extension.

d. 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 the File Information Block (FIB) is 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 FØØØ; Device Parity Error - Fatal Error FØ17.

Example: Open for input a dataset named IMP, which is a 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)
```

(continued on next page)

```
        :
        :
      .CLOSE SET1,FILE1          ;CLOSE SET1
        :
        :
      .RLSE SET1
        :
        :
      .WORD ERR1
SET1: .WORD Ø
      .RAD5Ø /IMP/
      .BYTE 1,3
      .RAD5Ø /DT/
        :
        :
      .WORD ERF1                 ;ADDR OF ERROR RTN
      .WORD Ø                    ;MONITOR PUTS HOW-OPEN HERE
FILE1: .RAD5Ø /PRO/              ;FILE NAME
      .RAD5Ø /G1/
      .RAD5Ø /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 his 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, then 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 a .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 fatal and will result in a F010 message.

Example: (See .WRITE.)

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, then 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 Ø
```

Global Name: (Permanently Core Resident.)

Description: .WAITR tests for completion of the last requested action on the specified dataset. If all actions are completed, 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 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, this is a fatal error, and the message 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 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. (See Appendix C for subsidiary routines.)

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, and 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.

OUTPUT: During an OUTPUT request, 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 non-directory 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 random 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 line 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, 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 actioned.

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

```
            .INIT TAPE1
              :
            .TRAN TAPE 1,BIN4Ø
              :
            .RLSE TAPE 1
            .WORD ERR 1
TAPE 1:     .WORD Ø
            .RAD5Ø /TP1/
            .BYTE 1,3
            .RAD5Ø /DT/
              :
BIN4Ø:      .WORD 1ØØ              ;STARTING BLOCK #
            .WORD 4ØØØ             ;STARTING ADDRESS IN CORE
            .WORD 2ØØ              ;NUMBER OF WORDS
            .WORD 4                ;INPUT
            .WORD Ø                ;FOR MONITOR USE
```

2.6.4 Requests for Input/Output Related Services

2.6.4.1 .SPEC

Special functions.

Macro Call: .SPEC LNKBLK,CODE

where LNKBLK is the address of the link block, and CODE is the special function code.

Assembly Language Expansion:

```
MOV #CODE,-(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. The code identifies the special function. Available codes are listed below. If a SPEC request is made to a device which has no special function, then an immediate return is made showing that the function has been completed. 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:

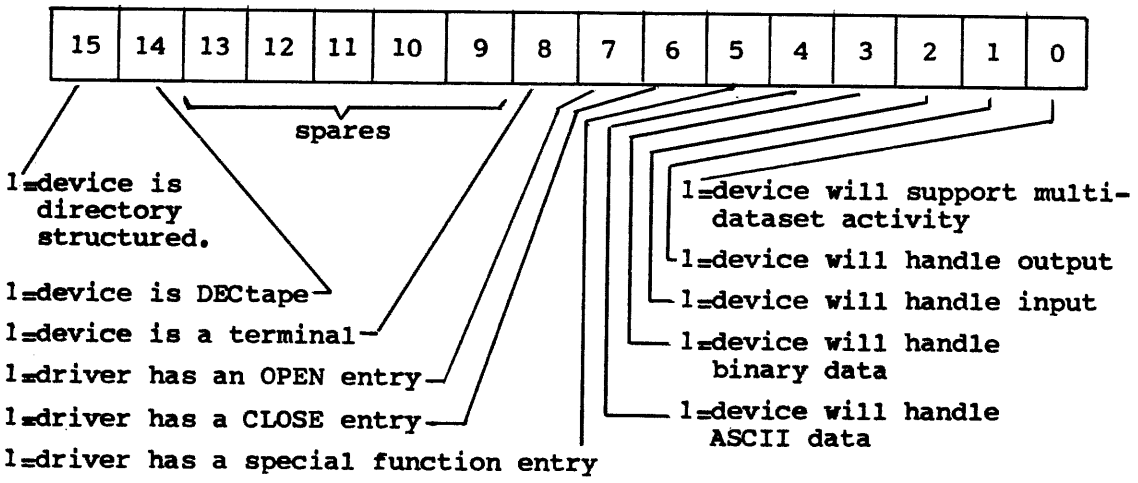
SP	Driver Facilities Word
SP+2	Device Name
SP+4	Device Standard Buffer Size

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 #FLKBLK,-(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.,

$$\left\lceil \frac{N}{B} \right\rceil$$

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

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

After the request has been completed, control is returned to the user at the instruction following the assembly language expansion. The segments are removed from the stack.

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

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

```
        .
        .
        .
        .ALLOC FRQ,FREQIN,20
        .
        .
        .
        .WORD ERR1
FRQ:    .WORD 0
        .RAD50 /DTA/
        .BYTE 1,4
        .RAD50 /DT/
        .
        .
        .
```

(continued on next page)

(continued from preceding page)

```
                    .
                    .
                    .
                .WORD ERR2
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 file-name 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.

The user must be authorized to access the file. The new file name must not already exist, and the new file name must be legal.

The old file must exist.

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

2.7.5 .LOOK

Searches the directory for a particular filename.

Macro Call: .LOOK LNKBLK,FILBLK

where LNKBLK is the address of the link block, and FILBLK is the address of the filename block.

Assembly Language Expansion:

```
MOV #FILBLK,-(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 will also be used to return the characteristics of a non-directory device.

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 two elements of the stack as follows -

Length	SP
Indicator Word	SP+2

where Length is the number of blocks in the file in binary, and the Indicator Word is encoded as follows:

```
bit Ø=1      ...    OPENC allowed
bit 1=1      ...    OPENI allowed
bit 2=1      ...    OPENE allowed
bit 3=1      ...    OPENU allowed
bit 4=Ø      ...    file is closed
     =1      ...    file is open
bit 5 is Ø
bit 6=Ø      ...    file is linked
     =1      ...    file is contiguous
bit 7=Ø      ...    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.

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 to 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
Dataset Not Inited	None	F000
Device Not Ready	None	A002
File Is Open	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.

2.8 DEFINITION OF REQUESTS FOR MISCELLANEOUS SERVICES

2.8.1 Requests to Return Control to the Monitor

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), are returned to free core.

2.8.2 Requests to Set Monitor Parameters

In addition to the above programmed requests, the user has some utility and conversion routines which he accesses via the EMT levels 41 and 42 instruction. The user communicates his request to the monitor by pushing the necessary parameters and an identifier code onto the stack.

2.8.2.1 .TRAP

Sets 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. Control is returned to the user at the instruction following the assembly language expansion, after the request is completed. 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 default address in the absence of an address in the REstart operator command. 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

2.8.3.1 .CORE

Obtains 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

Obtains 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. Control is returned to the user at the instruction following the assembly language expansion after the request is completed. 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 (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.

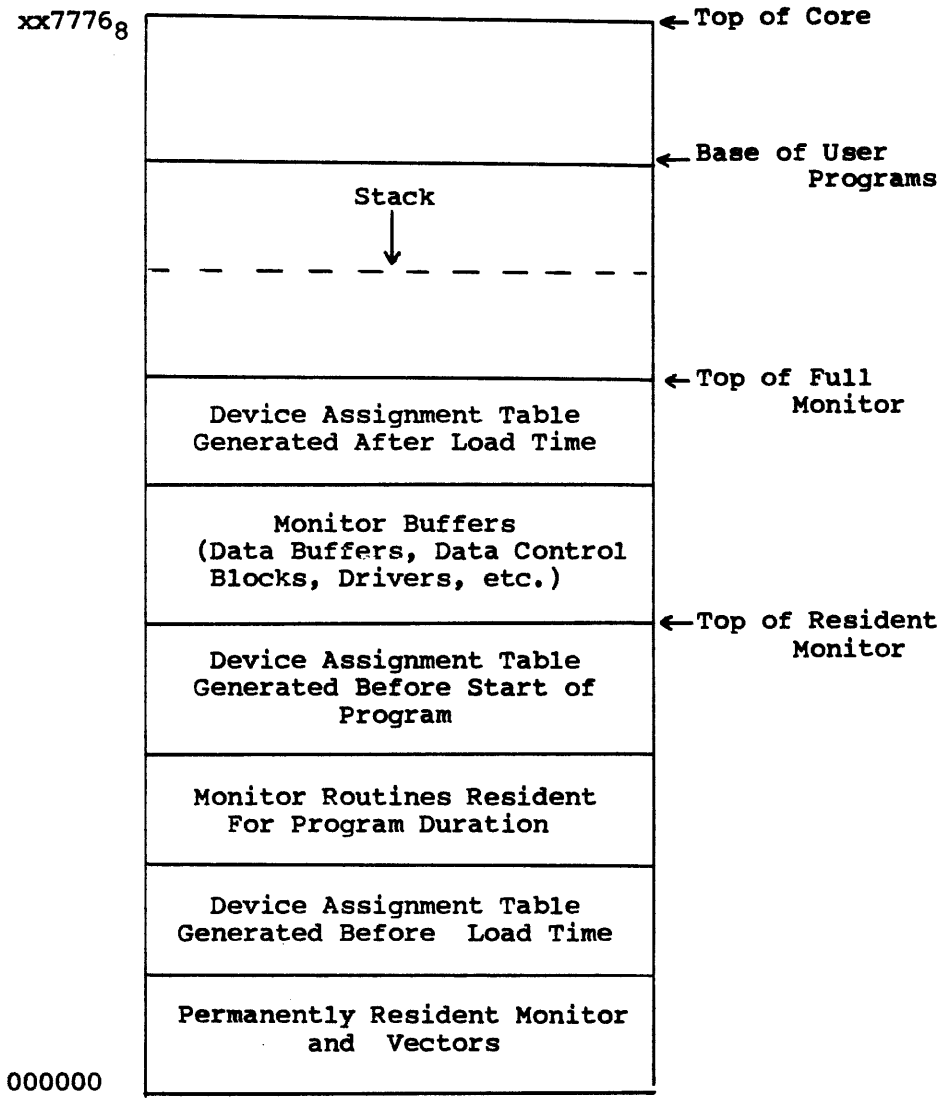


Figure 2-4. Core Map of Resident Monitor and Full Monitor.

2.8.3.4 .DATE

Obtain 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 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	PC
HIGH ORDER TIME	PC+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 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.

2.8.4. Requests to Perform Conversions

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)
MOV #0,-(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 Appendix D. 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:

N-bit set means an illegal call code (greater than 5) was used.

C-bit set means that an ASCII byte was used which was outside the valid RADIX-50 set. (See Appendix D.)

In the latter case, the returned value will be left-justified and correct up to the last valid byte. The address returned will be that of the first invalid byte. No conversion will follow recognition of an N error.

If there were no errors encountered during the conversion, the condition codes will be cleared, e.g., DT:=DT :

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:   MOV     #Ø,-(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.

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 con-

secutive bytes starting at location ADDR. The stack is returned cleared.

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:

N-bit set means a call code greater than 5 was used. No conversion has been attempted as a result.

C-bit set means that (a) a value of WORD was outside the valid RADIX-50 set, i.e., 174777, (see Appendix D); (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 any of the three a / 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.

N-bit set means a call code greater than 5 was used (no conversion). C-bit set means that a byte was not a digit. V-bit set means that the decimal number was too large.

The returned value 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 to five decimal ASCII characters.

Macro Call: .BIN2D ADDR,WORD

where WORD is the number to be converted, and ADDR is the address 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: 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:

N-bit set means a call code greater than 5 was used.

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

2.8.4.5 .02BIN

Convert six octal ASCII characters to one binary word.

Macro Call: .02BIN 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.

N-bit set means that a call code greater than 5 was used. 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. In either case, the returned value will be correct up to the last valid byte. The returned address will be that of the first invalid byte.

2.9.4.6 .BIN20

Convert one binary word to six octal ASCII characters.

Macro Call: .BIN20 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: 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:

N-bit set means a call code greater than 5 was used.

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

2.8.5 Requests for Interfacing to the Command String Interpreter

A user program may obtain I/O device specifications via keyboard input at run time by calling the Command String Interpreter (CSI) Monitor 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 device that the user expects to find in the command string. .CSI2 sets up the appropriate link block and filename 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 string buffer.

Assembly Language Expansion:

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

Global Name: CSX.

Description: Collapses 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 where the scan encountered the first error.

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 ≠ 0, (i.e., if there was a syntax error), then .CSI2 must not be called.

Example: (See .CSI2.)

2.8.5.2 .CSI2

Interpret one dataset specification of command string.

Format: .CSI2 CS2BLK

Assembly Language Expansion:

```
MOV #CS2BLK,-(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 buffer 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.

POINTER TO FILBLK is the address of the filename block of the dataset whose specification is being requested (currently, CSI allows just 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 specifications 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 that would fit in the space provided.

- With respect to the link block (Figure 2-5):

(a) If the PHYSICAL DEVICE NAME word is 0, 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 as follows:

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)

Assuming the switch was /SWITCH:V1:V2...Vn. If NUMBER OF WORDS TO FOLLOW is 0, there are no more switches. Note that the pointers are in reverse order. After the value pointers is a word which contains the first two characters of the switch in ASCII. The first character is in the low byte, and the second is in the high byte. If the name of the switch only contains 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 | 123

- With respect to the filename block (Figure 2-6):

(a) Recall that 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 the Monitor 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) Recall that EXTENSION occupies the word at FILBLK+4. If the Monitor returns zero at FILBLK+4, no EXTENSION was specified; if it returns 52_8, then * was specified; otherwise, the Monitor returns the first three characters of the extension specified, in RADIX-50 packed ASCII.

(c) Recall that 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 appropriated. 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 re-calling .CSI1 and issuing a 0 or 2 code, respectively. Note that he may not restart one without restarting the other, unless he saves and restores the appropriate information from CMDBUF.

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

2.8.5.3 The Link Block

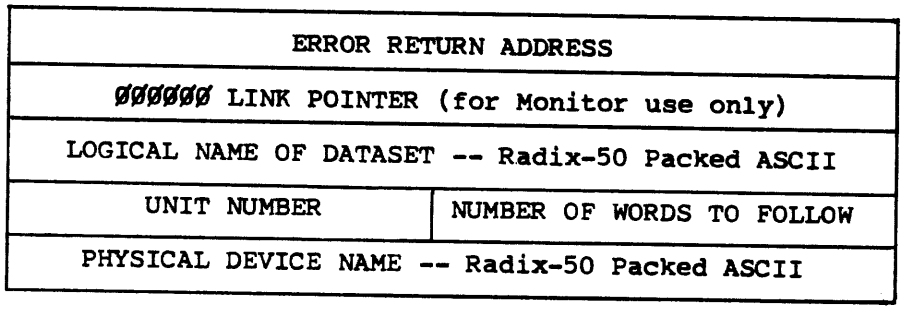


Figure 2-5. The Link Block.

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 which is 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 is the name that will be used if the ASsign command is to be called after the program is loaded. If this is possible, the name must be unique from that for other datasets. The

ADDRESS	NAME	FUNCTION

name is stored in RADIX-50 packed ASCII by the .RAD50 assembler directive.

LNKBLK+3	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+4	UNIT NUMBER	This code specifies the unit number of the device linked to the dataset. For example, the control

type TC11 (DECtape) can drive up to eight tape drives (units), numbered 0-7.

LNKBLK+6	PHYSICAL DEVICE NAME	If the user specified a 1 at LNKBLK+3, 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 a LOGICAL NAME OF DATASET and perform an ASsign command before he runs his program.

2.8.5.4 The filename Block

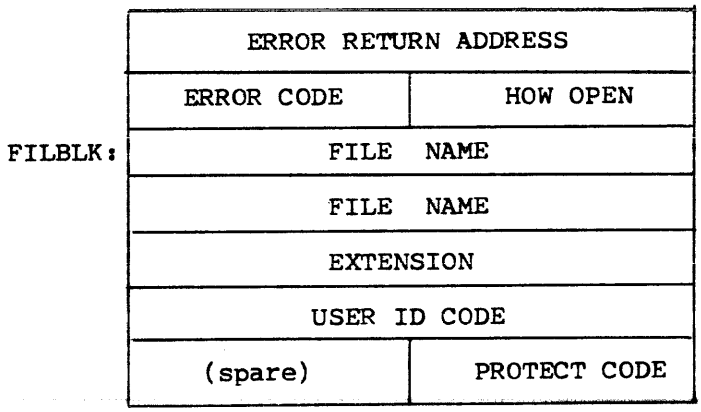


Figure 2-6. The Filename Block.

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 FILE NAME, EXTENSION, and PROTECT need not be specified. Each entry is explained below.

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 Section 2.10 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.

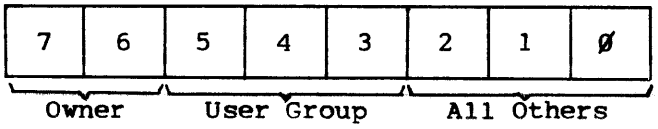
Error Code In File-name Block	Faulting Request	Cause	Remedy
ØØ	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to open dataset that was previously opened.	
Ø1		unused	
Ø2	.OPENO	An attempt was made to .OPENO a file which already exists.	Delete the file (with PIP) or change file name.
	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file for input, extension for update which is currently opened for output, or which does not exist.	
Ø3	.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.
Ø4	.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.
Ø5	.OPENE	Illegal request to a contiguous file.	
Ø6	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to access a file which the protection code prohibits.	
Ø7		unused	
1Ø	.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 DECtape.
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 requests 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 (Section 2.10) which occurred.
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+10	PROTECT 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.

2.8.5.5 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.

User Group and All Others:

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

Figure 2-7. File Protection Codes. **(continued)**

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. (concluded)

2.8.5.6 The Line Buffer Header

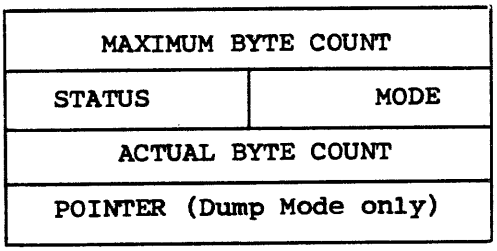


Figure 2-8. Line Buffer Header.

Each element of the Line Buffer Header table is explained.

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+1	MODE	The user specifies here the mode of the transfer. All modes are listed and explained in Figure 2-10.
BUFHDR+2	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 [g.u.] would have been transferred had some error not been detected).
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.

2.8.5.7 The Status Byte

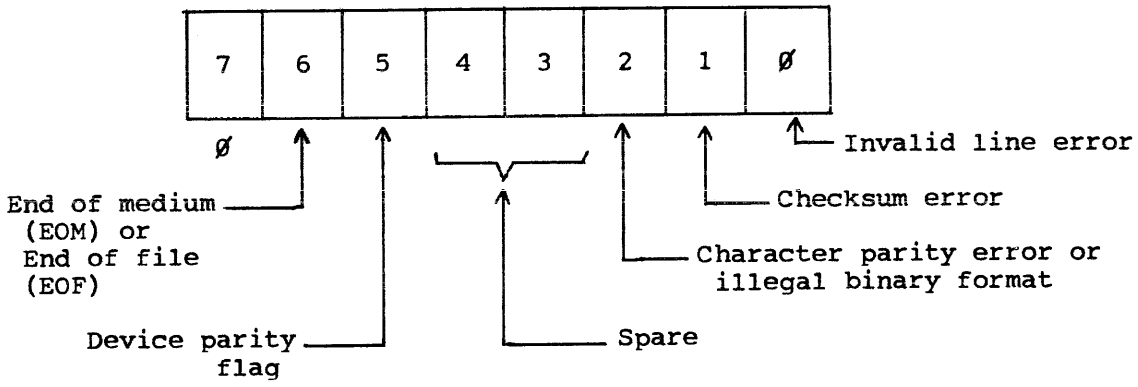


Figure 2-9. Status Format.

The function of each status format bit is explained below.

BIT	MODE	REQUEST	CONDITION
	ALL	.READ/WRITE	Appropriate BYTE COUNT=0 at call.
ø (INVALID LINE)	FORMATTED ASCII NORMAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT was reached 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 was reached 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 (CHECK-SUM 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 is delivered set to a 1.
2 (IL-LEGAL 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 intended as 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.)

2.8.5.8 The Transfer Modes

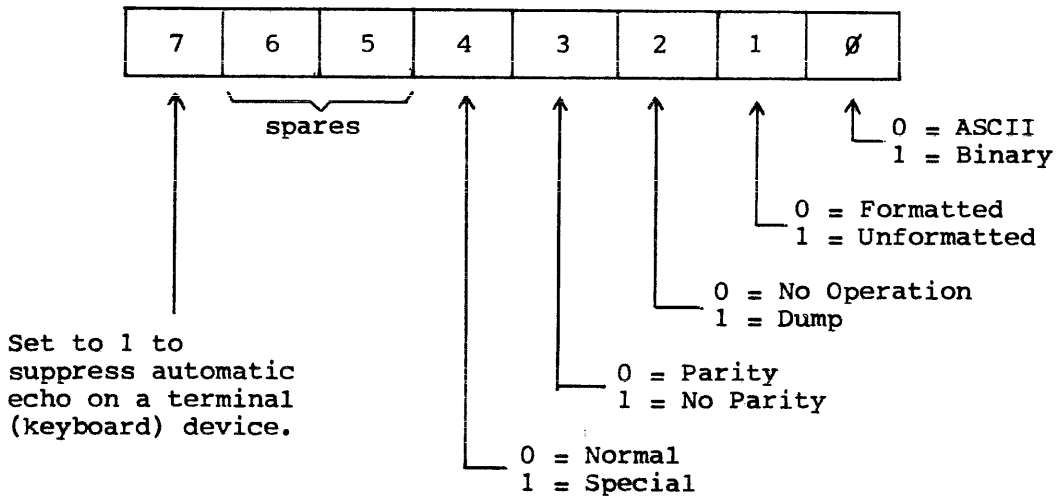


Figure 2-10. The Mode Byte.

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 thrown away. 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.

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 hands the data over to the device driver specified, and each driver will convert the information to meet its specific needs. Appendix F summarizes the characteristics of the drivers.

2. Formatted ASCII Special --

READ: The same as formatted ASCII normal with one 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 reset .READ. The 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 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 accumulated. 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 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 and checked on INPUT.

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.

2.8.5.9 The BLOCK Block

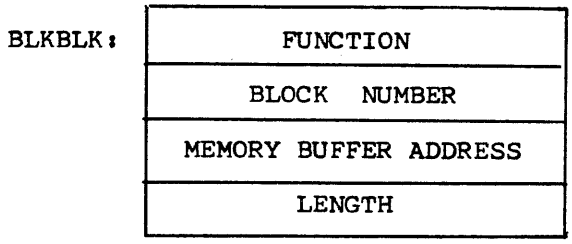


Figure 2-11. The BLOCK Block.

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 1 — function is OUTPUT 2 — function is INPUT (3-9) — spares (ignored by Monitor) error status: 10 — file is linked, or device is not file structured 11 — block number does not exist in file, i.e., it is greater than the file length. 12 — file not open 13 — protect code violation 14 — end of data error 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 Ø.
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	

2.8.5.10 The TRAN Block

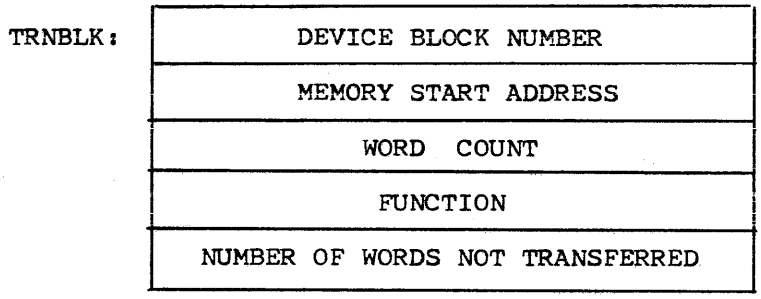


Figure 2-12. The TRAN Block.

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.
TRNBLK+6	FUNCTION/STATUS	Bit: — Bit = 1 means: 0 — spare 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

*Must be specified by user.

(continued on next page)

ADDRESS	NAME	FUNCTION
		12 } 13 } spare 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.

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

```
.INIT A
.INIT B
.INIT C
   :
   :
.OPENI A
.OPENO B
.OPENO C
```

Method 2

```
.INIT A
.OPEN A
.INIT B
.OPENO B
   :
   :
.INIT 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 routings are allocated first (i.e., at the bottom) then 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

```
.INIT C
   :
   :
.INIT B
   :
   :
```

Method 2

```
.INIT A
.INIT B
.INIT C
   .
   .
   .
```

*These bits are cleared before TRAN is carried out.

Method 1	Method 2
.INIT A	.RLSE A
:	.RLSE B
:	.
.RLSE A	.
.RLSE B	.
:	.RLSE C
:	
.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 re-use), he might, instead of.EXIT, use something like

```
BR .
```

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 on of four letters identifying the type of message:

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

where NNNN is the message number, and XXXXXX gives appropriate additional information. Informational and Warning 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 processorhalts. 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. Monitor's error messages are listed below.

The following are action messages:

Error Message	Additional Information	Remarks
A001	USER CALL ADDRESS	Disk address error
A002	DEVICE NAME (RAD50)	Device not ready
A003	LINK BLOCK ADDR.	No Device or Illegal Device or INIT
A004	USER CALL ADDRESS	DECtape error. Command CO will retry the operation.

The following are fatal error messages:

Error Message	Additional Information	Remarks
F000	Request Address	Dataset not Inited
F001	Request Address	Stack overflow

Error Message	Additional Information	Remarks
F002	Request Address	Invalid call
F003	Request Address	Invalid .TRAN function
F005	Request Address	.RLSE error (no .CLOSE after .OPEN on file structured device)
F006	Request Address	Device full (new file cannot be .OPENed)
F007	Request Address	No buffer space
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	Invalid open (no program return provided for failure)
F014	Request Address	Bit map failure (device error on trying to read bit map)
F015	Request Address	DECtape error (nonexistent memory addressed or end-zone reached dummy transfer)
F017	Device (RAD50)	File structures device parity error
F020		Too many datasets using low speed paper tape (a maximum of one for each direction is allowed).
F021	Irrelevant	Program Loader read failure
F022	Irrelevant	Program Loader format error (program is not in absolute load format).
F023	Program Size	Program too large for core available.
F024	Request Address	Illegal file structures operation, e.g., Delete or Rename of an open file.
F025	Device	Master Directory full (when attempting to add UIC).
F026	Request Address	Disk transfer failure.
F342		Error trap.

Error Message	Additional Information	Remarks
F344		Reserved instruction trap.
F346		Trace trap.
F350		Power fail trap.
F352		Trap instruction trap.
F356		Unexpected device interrupt.

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 104016           EMT 16
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     ERROR    = 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?
       000005
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    000300R     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 summarized in Appendix D and described in detail in Section 3.3.

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.
.	The Monitor has acknowledged a CTRL/C typed by the operator and is in listening mode, ready to accept a message 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) through the next RETURN are interpreted as Monitor commands and are passed to the Monitor for execution.

- All other characters are assumed to be intended 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.

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:

- Brackets [ ] are used to enclose elements of the command which are optional, i.e., they may appear or 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 a space must appear in that position.

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. RETURN is generated on the teleprinter as a carriage return and LINE FEED.
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 character of a Command has already been deleted, a RUBOUT will echo as RETURN, LINE-FEED, and the user must type CTRL/C to input the command anew. A RUBOUT given in the same circumstance within a line of program input will produce no response. The Monitor prints the deleted characters delimited by backslashes. For example, if one were typing .APPEND and typed .APPAM instead, the error can 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.

1 = a line terminator is a LINE FEED, FORM FEED, or VERTICAL TAB.

(continued on next page)

TABLE 3.1 (Cont'd)

Keyboard Key	Function
CTRL/C	When the CTRL key and C key are pressed, the Monitor is alerted to accept a command from the keyboard. CTRL/C is echoed as ↑C RETURN LINE FEED period. 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.

(continued on next page)

Table 3-1 (Cont'd)

Keyboard Key	Function
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 echo as ↑U RETURN LINE-FEED and the user must type CTRL/C to enter the command anew. CTRL/U given within a line of program input will merely echo as ↑U. 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. (concluded)

(Section 3.3. Continued from page 3-2.)

- DEVICE NAME refers to a physical device name as listed in Appendix A.

- DATASET SPECIFIED may be represented by any portion of the expression:

  DEV:FILENAM.EXT,[UIC]

  where

DEV:	is a physical device name as listed in Appendix and is followed by a colon.
FILENAM	is a file name of up to 6 characters.
.EXT	is a period followed by a file-name extension of up to 3 characters.
UIC	is the user's identification code in the form Group No., User No.

- 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 ↵ and is echoed by the Monitor as RETURN LINEFEED.

NOTE: An invalid command causes the error message

BAD CMD - IGNORED

to be typed on the teleprinter, and the command is ignored.

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, then the file name in the associated filename block is used. Any FILE NAME specified for a non-file-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.

The 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 suite 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 programmer performs a re-assignment. 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 re-assigns, or restarts the program by 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
```

3.3.1.2 The OTher Command

OT[HER],DATASET SPECIFIER

The stated DATASET is made the Monitor command device. The keyboard may recover control as command device with ↑C from its keyboard or by command OTher KB from the other device. OTher is valid only when no program is running. This command may be used to provide a form of batch-processing, for example.

3.3.2 Commands to Manipulate Core Images

3.3.2.1 The RUn Command

RU[N],DATASET SPECIFIER

The RUn command loads 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:

  FØ22 XXXXXX

- If the program to be loaded is too large for the available core, the fatal error message F023 [PROGRAM SIZE] is printed. Recovery from all these errors will be by way of a KIll command.

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 command to commence execution.

3.3.2.3 The DUmp Command

DU[MP],DEVICE NAME:$\left[,\left\{\begin{matrix}O\\I\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}START\ ADDR\\0\end{matrix}\right\}[,END\ ADDR]\right]$

This command writes an absolute copy of the specified core area to or from a fixed area on the specified device. I in the second argument specifies dump to core, and O specifies a dump from core. The core image is not altered. An O is assumed on default. Ø 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 taken to effect the dump.

3.3.2.4 The SAve Command

SA[VE]

SAve writes the program in core on the system disk in loader format. The core image is not altered. SAve is valid only when a program is in core but is not running.

The file which is created is always named SAVE.BIN, on dataset SAV, device DF. Before this file is created the Monitor deletes any file in the user's area on the disk with the same name. Thus, if it is desired to retain the saved file for any length of time, it should be renamed or copied to another area and given a new name, by means of PIP.

The SAVE file may be directed to any device other than DF by ASSIGNning the dataset SAV to the desired device.

The segment of core which is saved starts at the initial program load address and ends with the last word in memory. The saved image will be preceded by the same communication information as that for the original program loaded.

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 address used will be the normal start address. 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 a buffer, devices and assignments made dynamically and the stack will be cleared before control is passed back to the program.

To start a program at its normal start address, type

```
BE
```

To start a program at location 3446, type

```
BE,3446
```

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 drives set up for I/O operations will remain linked to the program for further usage.

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 a COntinue or a REstart command. 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 completingany 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. The user must reload the program or load another by RUn or GEt, to resume operations.

3.3.5 Commands to Exchange Information With the System

3.3.5.1 The DAte Command

DA[TE][,DAY]

This command sets the Monitor's date-word to the date specified in DAY, or if DAY is not specified, it prints the date previously specified. 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 in Julian, i.e., 71026 for January 26, 1971

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 types 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.

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 RETURN is typed at the end of the first line, the system responds by typing the CONTENTS of that address. At this point, the user can type one of the following (↵ denotes the RETURN key):

↵	will leave the CONTENTS unmodified
NEW CONTENTS↵	will change CONTENTS to NEW CONTENTS

This command is valid at any time. To change the contents of location 4ØØØØ:

```
↑C
.11040000↵
$40000/164060: 104060↵
```

Then to examine the contents of 4ØØØØ:

```
↑C
.l1040000)
$40000/104060: )
```

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 (Figure 2-11) set. This byte can be set at the file's creation, or by the .KEEP programmed request (Section 2.7.6).

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][, $\left\{\begin{matrix}KB\\PT\end{matrix}\right\}$]

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.

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

3.3.6.4 The ODt Command

$$OD[T][,\begin{Bmatrix}R\\K\end{Bmatrix}]$$

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 +Ø	(clear ODT breakpoint table without resetting breakpoints)
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
.OD,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. 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. A command under CSI consists of one or more output dataset specifications, followed by <, followed by one or more input dataset speci-

fications. 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.

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

A dataset specification must be delimited by a comma. If no items at all appear before the comma, then this 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 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]/SW1:V1:...:Vm/SW2:V1:...:Vm,

where DEV = The device specification which consists of one or more letters, with a maximum of three, followed by a colon. The letters identify the device and the digits identify 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 digits appear, then unit Ø is assumed. If the device specification itself does not appear, then the current device is defaulted to the device last specified, if there is one; otherwise, it is defaulted to disk (DF).

Defaults do not carry across the < , i.e., from output to input.

FILNAM = The file name specification, consisting of one or more letters or digits, or exactly one asterisk. The first six letters or digits specify the name. All letters and digits in excess of six are ignored.

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

.EXT = The extension specification which 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. No system-wide default extension is assumed.

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 which 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 within the group. Both fields must be given in octal. The largest useable 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.

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.

/SWm:V1:...:Vm = A switch specification which 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, ir consists of one or more letters, digits, periods, or dollar signs. Other characters are illegal. The digits 8 and 9 are legal.

For examples: /DATE: 12:2Ø: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, then 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	E
,	*	*	*	E	*	*	E[2]	E[2]
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.

Notes: 1 The next item encountered is the first item in the command string.
2 This is an error because the following command is meaningless, e.g.,

.<terminator

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 and its interpretation is:

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

which is interpreted as:

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

The second positional output dataset will not be generated.

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 is to be performed assuming value 1 on this dataset.

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

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

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 Ø. 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.

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 Ø. Associate the ID of the user who entered the command with this dataset. No switches are associated with this dataset.

The fourth and subsequent input datasets are not required.

A command of the following form:

```
Dataset Specification, < Dataset Specification
                         terminator
```

could be interpreted to mean "do not generate the second and subsequent datasets", but this is accomplished by:

```
Dataset Specification  < Dataset Specification
                         terminator
```

as well.

APPENDIX A

PHYSICAL DEVICE NAMES

MNEMONIC	DEVICE
DC	RC11 Disk
DF	RF11 Disk
DK	RK11 Disk
DT	DECtape (DT11)
KB	ASR-33 Keyboard/Teletype
LP	Line Printer (LP11)
PP	High Speed Paper Tape Punch
PR	High Speed Paper Tape Reader
PT	ASR-33 Paper Tape Device

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

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

APPENDIX B

EMT CODES

Code	Usage
Ø	.WAITR
1	.WAIT
2	.WRITE
3	**
4	.READ
5	**
6	.INIT
7	.RLSE
10	.TRAN
11	.BLOCK
12	.SPEC
13	.STAT
14	.LOOK
15	.ALLOC
16	.OPENx
17	.CLOSE
20	.RENAM
21	.DELET
22	.APPND
23	**
24	.KEEP
25-27	**
30-31	*
32	Diagnostic Print

* Reserved for Monitor internal communication

** Reserved for future Monitor expansion

EMT CODES

Code	Usage
33,34	*
35-37	**
40	*
41	General Utilities
42	General Conversions
43-55	*
56,57	Command String Interpreter
60	.EXIT
61,62	*
63-77	**
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

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 when and only when a file structured device is referenced

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

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

D = subsidiary routine is called when and only when DECtape 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.
.READ/WRITE												X
.OPENU	X			X				X				
.OPENO		X		X	X	X		X				
.OPENE	X			X	X	X		X				
.OPENI	X			X				X				
.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
OTher	Pass Monitor control to another device
Commands to Manipulate Core Images	
RUn	Load and begin a program
GEt	Load a program
DUmp	Write a specified core area onto a device in absolute binary
SAve	Write a program onto a device in loader format
Commands to Start a Program	
BEgin	Starts execution of a program
COntinue	Resumes execution of a halted program
REstart	Restarts execution of a previously operating program
Commands to Stop a Program	
STop	Halt the current program, including any I/O in progress
WAit	Halts current program after finishing any I/O in progress
KIll	Halts the current program, finishes any I/O in progress, closes all open files, and passes control back to the Monitor.
Commands to Exchange Information with the System	
DAte	Fetch/Specify date
TIme	Fetch/Specify time
LOgin	Enter User Identification Code

Command	Usage
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
				2-30

.APPND	Append to a Linked File	.APPND LNKBLK,FIRST,SECOND	MOV #SECOND,-(SP)
			MOV #FIRST,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 22
				2-34

.BIN2D	Convert Binary to Decimal ASCII	.BIN2D ADDR,WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #3,-(SP)
			EMT 42
				2-45

.BIN20	Convert Binary to	.BIN20 ADDR,WORD	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 42
				2-46

.BLOCK	Transfer a Block	.BLOCK LNKBLK,BLKBLK	MOV #BLKBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 11
				2-7
				2-25
				2-62

.CLOSE	Close a Dataset	.CLOSE LNKBLK	MOV #LNKBLK,-(SP) EMT 17	2-21
.CORE	Obtain Core Size	.CORE	MOV #1ØØ,-(SP) EMT 41	2-38
.CSI1	CSI Interface - part 1	.CSI1 CMDBUF	MOV #CMDBUF,-(SP) EMT 56	2-47
.CSI2	CSI Interface - part 2	.CSI2 CSIBLK	MOV #CSIBLK,-(SP) EMT 57	2-48

Mnemonic	Function	Macro Call	Assembly Language Expansion	Refer to Page
.DATE	Obtain Date	.DATE	MOV #1Ø3,-(SP) EMT 41	2-41

.DELET	Delete a File	.DELET LNKBLK,FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 21
				2-32

.D2BIN	Convert Decimal ASCII to Binary	.D2BIN ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 42
				2-44

.EXIT	Exit to Monitor	.EXIT	EMT 6Ø	2-37
.GTUIC	Get Current UIC	.GTUIC	MOV #1Ø5,-(SP) EMT 41	2-41
.INIT	Initialize a Dataset	.INIT LNKBLK	MOV #LNKBLK,-(SP) EMT 6	2-15

.KEEP	Protect a File	.KEEP LNKBLK,FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 24
				2-36

.LOOK	Directory Search	.LOOK LNKBLK,FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
				2-35

.MONF	Obtain full Monitor size	.MONF	MOV #1Ø2,-(SP) EMT 41	2-39
.MONR	Obtain size of resident Monitor	.MONR	MOV #1Ø1,-(SP) EMT 41	2-39

.OPENx	Open a Dataset	.OPENx LNKBLK,FILBLK	MOV #CODE,FILBLK-2
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
				2-17

Mnemonic	Function	Macro Call	Assembly Language Expansion	Refer to Page
			(CODE = 1 for .OPENU 2 for .OPENO 3 for .OPENE 4 for .OPENI 13 for .OPENC	

.02BIN	Convert Octal ASCII to Binary	.D2BIN ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 42
				2-44

.RADPK	RADIX-50 ASCII Pack	.RADPK ADDR	MOV #ADDR,-(SP)
			MOV #Ø,-(SP)
			EMT 42
				2-42

.RADUP	RADIX-50 ASCII Unpack	.RADUP ADDR,	MOV #WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #1,-(SP)
			EMT 42
				2-43

.READ	Read from Device	.READ LNKBLK,BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 4
				2-4
				2-14
				2-22

.RENAM	Rename a file	.RENAM LNKBLK,OLDNAM,NEWNAM	MOV #NEWNAM,-(SP)
			MOV #OLDNAM,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2Ø
				2-33

.RLSE	Release a Dataset	.RLSE LNKBLK	MOV #LNKBLK,-(SP) EMT 7	2-16

.RSTRT	Set REstart address	.RSTRT ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 41
				2-38

.SPEC	Special Function	.SPEC LNKBLK,CODE	MOV #Code,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 12
				2-28

Mnemonic	Function	Macro Call	Assembly Language Expansion	Refer to Page
.STAT	Obtain Device Status	.STAT LNKBLK	MOV #LNKBLK,-(SP) EMT 13	2-29 2-57
.TIME	Obtain Time of Day	.TIME	MOV #1Ø4,-(SP) EMT 41	2-41

.TRAN	Transfer absolute block	.TRAN LNKBLK,TRNBLK	MOV #TRNBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 1Ø
				2-9
				2-27
				2-63

.TRAP	Set TRAP vector	.TRAP STATUS,ADDR	MOV #ADDR,-(SP)
			MOV #STATUS,-(SP)
			MOV #1,-(SP)
			EMT 41
				2-37

.WAIT	Wait for Completion	.WAIT LNKBLK	MOV #LNKBLK,-(SP) EMT 1	2-24

.WAITR	Wait for Completion; Return to ADDR	.WAITR LNKBLK,ADDR	MOV #ADDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT Ø
				2-25

.WRITE	Write on a Device	.WRITE LNKBLK,BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2
				2-4
				2-14

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 appended to

NOTES: (Cont'd)

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)
TRNBLK	address of TRAN Block

APPENDIX F

DEVICE DRIVERS

Device drivers are classified by the types of devices they serve. There are:

- Terminal devices such as keyboards or displays that interface directly to human operators.
- Bulk storage devices such as disks or DECtapes which are usually file or directory structured.
- Others

Terminal Device Drivers

Drivers for terminal devices are usually the most complex because:

- People are unpredictable and demanding. they require flexibility and understanding.
- Terminals are usually inexpensive and depend on software to perform what other devices manage with hardware.

A terminal driver must be designed to respond not only to valid commands from the knowledgeable programmer, but also to a variety of errors from less experienced people Such a driver must give the operator the opportunity to correct any mistakes he may discover as he uses the device, must not depend on any particular time between inputs, and cannot require a particular length of input. Rather, a terminal must accept commands to ignore characters or lines at the leisure of the operator, of whatever length he so desires, subject only to the limitation of the driver's buffer.

The ASR-33 Teleprinter Driver

The Keyboard

When an operator enters data from the keyboard, the data is stored in a buffer within the driver until requested by the program. As the data is processed for transfer to the program, it is echoed back to the teleprinter. While the data is in the buffer, it can be added to, deleted, or Modified by the operator. Once passed to the Monitor, the data cannot be directly changed. Keyboard keys which are recognized as special functions by the driver are:

RUBOUT — which erases the preceding character from the buffer and echoes deleted characters between \....\

CTRL/U — which deletes the current line and echoes as ↑U

RETURN — which places a carriage return followed by a line feed into the buffer, and echoes the same to the teleprinter

TAB — is simply echoed as spaces up to the next stop position. The TAB code is placed into the buffer.

*ESC — which notifies the Monitor that it is not to take action on the next character. This allows the programmer to pass to the user program or systems program a code which is normally interpreted by the monitor in a special way. For example, ESC CTRL C will not cause the Monitor to expect a command. Similarly, preceding CTRL U by ESC will remove its erasing facility.

The Teleprinter

TAB — is replaced with spaces up to the next stop position

RUBOUT — is discarded

---

*On some Teletypes, this appears as ALTMOD.

APPENDIX G

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 linkblock 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.
Driver, device	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 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 (a core image) 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.

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 DEC 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 available without charge from the Program Library, 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.

New and revised software and manuals, Software Performance Report forms, and software price lists are available from the Program Library. When ordering, include the document number and a brief description of the program or manual requested. Revisions of programs and documents will be announced in the newsletters. Direct all inquiries and requests to:

Program Library
Digital Equipment Corporation
146 Main Street, Bldg. 1-2
Maynard, Massachusetts 01754

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-SERA-D
February 1971

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