Disk Operating System Monitor
Programmer's Handbook

DEC-11-OMONA-A-D

PDP-11

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

P R O G R A M M E R ' S   H A N D B O O K

Monitor Version V08-02

October 1972

SOFTWARE SUPPORT CATEGORY

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

For additional copies, order No. DEC-11-OMONA-A-D from DEC,
Software Distribution Services, Maynard, Mass. 01754

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

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

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

NOTE

Associated Documents:

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

PDP-11 MACRO-11 Assembler,
Programmer's Manual, DEC-11-OMACA-A-D

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

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

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

PDP-11 PIP, File Utility Package,
Programmer's Manual, DEC-11-UPUPA-A-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 MACRO-11 Assembler (see document number DEC-11-OMACA-A-D). Previous experience with monitor or executive systems would be helpful.

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

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

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

SOFTWARE SUPPORT CATEGORIES

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

CATEGORY I
Software Products Supported at no Charge

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

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

CATEGORY II
Software Products that Receive Support for a Fee

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

CATEGORY III
Pre-Release Software

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

CATEGORY IV
Non-Supported Software

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

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 THE DOS MONITOR

The PDP-11 Disk Operating System (DOS) Monitor is a powerful, keyboard-oriented, program development system designed for use on PDP-11 computers. The DOS Monitor facilitates use of a wide range of peripherals available for use with the PDP-11.

The DOS Monitor supports the PDP-11 user throughout the development and execution of his program by:

- providing convenient access to system programs and utilities such as the FORTRAN Compiler[1], the MACRO-11 Assembler[1], a Linker, a debugging package, an Editor, a file utility package, etc.;
- performing input/output transfers at four different levels, ranging from direct access of device drivers to full formatting capabilities, while providing the convenience of complete device independence;
- providing a file system for management of secondary storage; and
- providing a versatile set of keyboard commands for use in controlling the flow of programs.

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

DOS gives the user program the capability of complete device independence. Programs can be written without concern for specific I/O devices. When the program is run, the user can select the most effective or convenient I/O device available for the function to be performed. In addition, if the system configuration is altered, many programs can take advantage of the new configuration without being rewritten. Logical names can be assigned to devices

[1] Available only on 12K or larger systems. The 8K assembler does not support macros.

within the system enabling symbolic referencing of any device. No concern need be given to I/O buffer size within the user program yet the user can alternatively retain direct control of I/O buffers.

All input/output (I/O) transfers are handled by the Monitor in any of three user-selected levels called READ/WRITE, RECORD/BLOCK, and TRAN. READ/WRITE is a formatted level of I/O in which the user can specify any one of nine options. RECORD/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 files: linked and contiguous. Linked files can grow serially and have no logical limit on their size. Contiguous files must have their lengths declared before use but can be randomly accessed by RECORD or BLOCK level I/O requests. All blocks in a contiguous file are physically adjacent, while blocks in a linked file are typically not adjacent (the first word of each block contains the address of the next block). Files can be deleted or created at any time, and are referenced by name. Table 1-1 summarizes the features and benefits of the DOS Monitor.

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

Keyboard commands enable the user to load and run programs; assign I/O devices or files; start or restart programs at specific addresses; modify the contents of memory locations; retrieve system information such as time of day and date; and dump core. Users with more than 8K of memory[1] can utilize programmed requests, which are macros assembled into the user's program and through which the user specifies the operation to be performed by the Monitor. 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.

---

[1]8K users must include the code generated by such an assembly (the assembly language expansion shown in Appendix E and in the explanation of each programmed request in Chapter 3) in their programs to utilize the Monitor functions. See the MACRO manual (DEC-11-OMACA-A-D) for other differences in the 8K Assembler.

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 on any bulk-storage device.	Files can be deleted or created even at run time for maximum 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 before or during run time.	The user can tailor the Monitor for his particular needs.
The Monitor is divided into logical modules.	The user can easily and efficiently use the logical pieces of the Monitor for his own needs. He can also easily add his own specialized drivers to the system by following a simple set of rules, and still use the rest of the Monitor with these drivers.
All I/O is interrupt-driven.	Such specialized equipment as communications modems and A/D converters which must be interrupt driven can be run under the Monitor. Several I/O calls can be handled concurrently.
Device independence.	Any device can be specified by the user in his program, and another device can be substituted by him when his program is being run.
Devices are assigned to one or more datasets.	The user may reassign a device which is used for one purpose (dataset) without changing its assignment for all other purposes (datasets).

Other requests access Monitor facilities to query system variables such as time of day, date, and system status, and to specify special functions for devices.

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

Table 1-2

Principal DOS System Programs

System Program	Document Number
FORTRAN IV	DEC-11-LFIVA-A-D
MACRO-11 Assembler	DEC-11-OMACA-A-D
EDIT-11 Text Editor	DEC-11-EEDA-D
ODT-11R Debugging Program	DEC-11-OODA-D
PIP, File Utility Package	DEC-11-UPUPA-A-D
Link-11 Linker and Libr-11 Librarian	DEC-11-ULLMA-A-D

1.2 MONITOR CORE ORGANIZATION

Core memory is divided into:

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

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

The DOS Monitor dynamically acquires and releases core on the basis of system requirements.

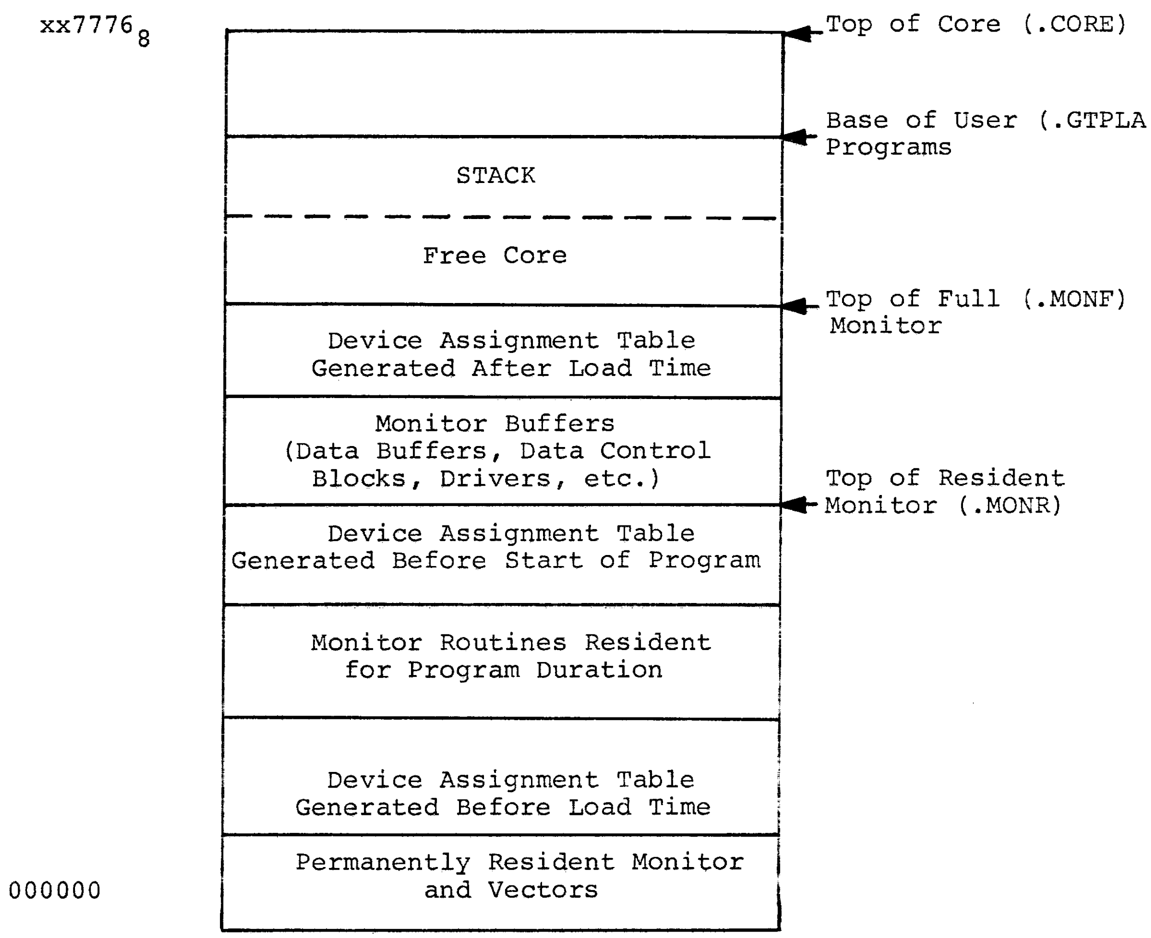


Figure 1-1 The Monitor Core Map

1.3 HARDWARE CONFIGURATIONS

Many minimum hardware configurations for use by the disk operating system may be derived by choosing one item from each of the five following sets.

- PDP-11 System Building Block with 900 nsec. Core Memory and a Terminal (DECwriter [LA30], Alphanumeric CRT [VT05-B], or Teletype[1] [LT33]).
- Cabinets and all Mounting Hardware.
- Bootstrap Loader (BM792-YB or MR-11).
- Choice of Disks (Control Logic Included)

  64K word Fixed Head Disk (RS64/RC11)
  256K word Fixed Head Disk (RF11/RS11)
  1.2 word Interchangeable Cartridge Disk (RK05/RK11)

- Choice of Tape Devices (Control Logic Included)

  Dual Drive DECtape (TU56/TC11)
  7- or 9-track Industry Standard Magnetic Tape (TU10/TM11)
  High-Speed Paper Tape Reader/Punch (PC11)

Specific details are available from a sales representative. Note that 12K of core is required with the RK disk and DECtape is required with the RC disk.

1.4 MONITOR MESSAGE

When a message-producing situation (such as a system error) occurs, an error code and an additional word of information are displayed on the teleprinter. There are five types of messages:

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

The type of message is identified by being preceded by the letter I, A, W, F or S 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 Appendix F.

[1]Teletype is a registered trademark of the Teletype Corporation.

1.5 STARTING THE MONITOR

The Monitor is called into core from disk by performing the following procedure for systems with the BM792YB:

1. If the system device is an RK11 Disk, turn WRITE ENABLE off;
2. Move HALT/ENABLE switch to HALT position;
3. Load the processor switch register with 173100;
4. Depress LOAD ADDRESS processor switch;
5. Load the switch register with,

    177462 if the system device is RF11 disk,
    177406 if the system device is RK11 disk,
    177450 if the system device is RC11 disk;

6. Move HALT/ENABLE processor switch to ENABLE position;
7. Depress START processor switch.

With the MR11 Bootstrap Loader, the procedure is:

1. Load the processor switch register with:

    173100 if the Monitor storage device is RF11 disk,
    173110 if the Monitor storage device is RK11 disk,

2. Move HALT/ENABLE switch to HALT position;
3. Move HALT/ENABLE switch to ENABLE position;
4. Depress LOAD ADDRESS processor switch;
5. Depress START processor switch.

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

```
DOS  Vxx
```

on the teleprinter, where Vxx represents the version number of the Monitor being used. The Monitor is now ready to accept an operator command (see Chapter 2).

1.6 A GUIDE TO THIS HANDBOOK

1.6.1 Terminology

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

A dataset is a logical collection of data which is treated as an entity by a program. Typically, the items in a dataset have a realtionship to each other which simultaneously binds them together and distinguishes them from items in other datasets. For example, the records in the Object dataset produced by the assembler are clearly related to each other and are clearly distinct from the listing dataset produced by the same assembler. A parameter file and a source file, when presented successively to the assembler, might be viewed as a single dataset, however.

Typically, each dataset is associated with exactly one link block (see section 3.8.6.1), although a link block can be associated (successively, not simultaneously) with more than one dataset. For example, when the assembler finishes processing one dataset and returns for another command, the new input will constitute a new dataset, but the same link block will be used.

Examples of datasets are:

- all or part of a file on a file-structured device;
- one or more paper tapes in a paper tape reader;
- a deck of cards, terminated by an EOF card;
- three lines of keyboard data, a disk file, and a paper tape; which are read in sequence by the assembler and are viewed as the source input dataset.

A device is any PDP-11 peripheral supported by the Monitor.

A device controller can support one or more device units.

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

On a directory device it is possible to store data by name, rather than simply physical location; it is also called a file-structured device.

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

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

A line is a string of ASCII[1] characters which is terminated by a LINE FEED, FORM FEED or VERTICAL TAB.

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

1.6.2 Standards for Tables

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

---

[1]ASCII represents American Standard Code for Information Interchange.

a) Representation in manual

	ENTRY A	
TABL:	ENTRY C	ENTRY B

b) Representation in program listing:

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

Note that the first byte specified is stored at the rightmost available byte.

1.6.3 Standards for Numbers

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

CHAPTER 2
MONITOR KEYBOARD COMMANDS

2.1 INTRODUCTION

This Chapter shows how the Disk Operating System (DOS) Monitor looks to the user as he sits at the terminal (i.e., the Teletype, DECwriter, etc.). The user is communicating with the DOS Monitor while running system, utility, and user programs.

Since DOS is an interactive operating system, the primary input and output device is the user's terminal or teleprinter (keyboard and printer). Through the terminal keyboard, the user can communicate with

- the Monitor,
- a system or utility program (Macro, PIP, Editor, etc.), or
- a user program written to run under DOS.

The terminal printer is used to record user input and system output.

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. Data from the keyboard may be transferred to a buffer in the user program or it may be processed immediately by the DOS Command String Interpreter (CSI) as explained in Appendix I. In this Chapter, the CSI is described only as it applies to the formatting of Monitor keyboard commands.

When the system is ready for input from the keyboard, a single character is printed on the teleprinter. The following conventions apply:

Character	Meaning
$	The system is idle, waiting for a Monitor command.
.	The Monitor is waiting to continue or abort a task.
#	A system, utility, or user's program requests a command through the CSI.
*	A system program requests direct input, i.e., not through the CSI.

Command String Interpreter

In this Chapter, we are concerned only with the $ and . characters. The # and * characters are explained in the individual system and utility programmer's manuals.

The $ and . indicate that the Monitor is waiting for a keyboard command from the user. Note, however, that some commands may be issued only to a $ and some only to a ., and that each command has different limitations; these are discussed with each command in Section 2.8.

2.1.1 Monitor Commands by Function

A number of keyboard commands are provided for communication with the DOS Monitor. These commands are briefly identified by function in Table 2-1 and are fully described in Section 2.8.

Table 2-1

Monitor Commands by Function

Function	Command
Establish identity of user	LOGIN
Terminate a session before leaving the system	FINISH
Enter or retrieve date	DATE
Enter or retrieve the time-of-day	TIME
Load and execute a program	RUN
Load a program	GET
Start a program which has been loaded	BEGIN
Resume a program that is waiting for user action	CONTINUE

(continued on next page)

Table 2-1 (Cont'd)

Monitor Commands by Function

Function	Command
Assign an I/O device or a file at run-time	ASSIGN
Inspect or modify individual memory locations	MODIFY
Save a program in core for later use	SAVE
Dump memory data on the teleprinter	DUMP
Suppress or resume echoing of keyboard input	ECHO
Suppress or resume teleprinter output	PRINT
Start the program just loaded at its ODT entry point	ODT
Stop a program	STOP
Suspend a program	WAIT
Restart a program that has been running	RESTART
Terminate a keyboard or paper tape dataset	END

2.1.2 When Monitor Commands are Legal

Each command performs a specific function, is legal to use under specific conditions, and often alters the state of the system, as shown in the following table.

Command	Legal When:	State Induced
ASSIGN	any time	no change
BEGIN	program loaded and stopped	program running
CONTINUE	program loaded and waiting	program running
DATE	any time	no change
DUMP	any time	no change
ECHO	program running	no change

(continued on next page)

Command	Legal When:	State Induced
END	program running	no change
FINISH	no program loaded	logged out
GET	no program loaded	program loaded and stopped
KILL	program loaded	program stopped and unloaded
LOGIN	not logged in	logged in
MODIFY	any time	no change
ODT	program loaded and stopped	program running under ODT
PRINT	program running	no change
RESTART	program loaded and stopped/waiting	program running
RUN	no program loaded	program loaded and running
SAVE	program loaded and stopped	no change
STOP	program running	program stopped
TIME	any time	no change
WAIT	program running	program waiting

A program is loaded if you have typed RUN or GET but not KILL, and as long as the program has not executed a .EXIT call (see Chapter 3).

A program is running if you have typed RUN or if it has been loaded and you have typed BEGIN, CONTINUE, RESTART, or ODT.

A program is loaded and stopped if GET but not BEGIN was typed, if it was running and a STOP was typed, or after issuing a fatal error message (see Appendix F).

A program is waiting if it was running and you typed CTRL/C followed by WAIT, or after the system issues an action error message (see Appendix F).

A program is stopped and unloaded (from core) if you have typed KILL or if the program issued an .EXIT call (see Chapter 3).

2.2 MONITOR MODE AND USER MODE

From the user's point of view, his terminal is in either Monitor mode or user mode. In Monitor mode, each line the user types is sent to the Monitor's Command String Interpreter (CSI). The execution of certain commands places the terminal in user mode. When the terminal is in user mode, it becomes simply an input/output (I/O) device for that user. In addition, user programs use the terminal for two purposes: to accept user command strings (user mode) or as a direct I/O device (data mode).

2.3 COMMAND STRING INTERPRETER (CSI)

When the terminal is in Monitor mode the user communicates with the Monitor's Command String Interpreter (CSI). The commands described in this Chapter are processed by the CSI (see Appendix I).

The CSI makes several checks before processing commands from the user. For example, if a user who has not logged in types a command that requires him to be logged in, the system responds with the message:

ILL CMD!

meaning the command was illegal and was not executed. The commands discussed in this Chapter require that the user be logged in except where explicitly stated otherwise. When a command is issued that requires the job to use more core than is available, the system responds with the message:

NO CORE!

and the user's command is not executed.

All Monitor messages are shown in Appendix F.

2.4 USER IDENTIFICATION CODE (UIC)

Each user of the system is normally assigned a User Identification Code (UIC) by the system or installation manager. The UIC is first used when logging in to the system, as explained in Section 2.7. The format of the UIC is:

nnn,nnn

where nnn represents a string of two or three octal digits, from 11 to 376 (Ø-1Ø and 377 are reserved for the system). The value to the left of the comma represents the user-group number, while the value to the right represents the user's number within the group.

For example:

    67,123

specifies user group 67 and user number 123.

NOTE

> Except when logging in, the UIC is always delimited by the left and right square brackets, as shown in the examples of various commands in this Chapter.

2.5 FILENAMES AND FILENAME EXTENSIONS

User program files are named with a certain convention, much the same as a person is named. For example, the first name is the filename and the second name is the filename extension. By convention, the filename and extension are separated by a period. For example:

    GEORGE.DOE

could be a legal filename and extension. Note that the filename and extension cannot have embedded blanks (spaces) because a space will be interpreted as a delimiter,

Filenames can consist of from one to six alphanumerics; all after the sixth are ignored. The filename extension can consist of from one to three alphanumerics. The extension is generally used to indicate the type of information in the file. For example:

File	Could be:
MAIN.F4	a FORTRAN file named MAIN
SAMPLE.MAC	a Macro source file named SAMPLE
TEST1.TMP	a temporary file named TEST1
NAME.REL	a relocatable binary file named NAME

A list of standard extensions are shown in Appendix O

User program files are identified by the filename.extension and the UIC. Thus, different users may use the same filename.extension, and as long as they are created under different UIC's the files would remain distinct and separate.

2.6 SPECIAL KEYBOARD CHARACTERS

There are several special keyboard characters recognized by the Monitor's CSI that cause specific functions to be performed. These keyboard characters are explained below.

2.6.1. The RETURN Key

The RETURN key is used to terminate a keyboard command and to advance the teleprinter paper one line. Typing the RETURN key produces a carriage return and line feed action on the teleprinter.

As characters are typed, they are transferred into a buffer where they are stored until the RETURN key (or another special keyboard character(s)) is typed. When the RETURN key is typed, the data on that line is transferred to and processed by the CSI.

All legal command strings are terminated by the RETURN key.

2.6.2 The RUBOUT Key

The RUBOUT key is used to correct typing errors. Typing the RUBOUT key once causes the last character typed to be deleted; typing it twice causes the last two characters to be deleted; etc. The Monitor prints the deleted characters delimited by backslashes. For example, if you meant to type ASSIGN but typed ASIS instead, the error could be corrected by typing two RUBOUTs and then the correct characters. The printout would be:

```
ASIS\SI\SIGN
```

Notice that the deleted characters are shown in reverse order, i.e., in the order in which they are deleted.

2.6.3 The CTRL/C Keys

The CTRL/C key combination is typed by holding down the CTRL key while typing the C key. When CTRL/C is typed, the Monitor is alerted to accept a command from the keyboard. CTRL/C is echoed on the teleprinter as ↑C, carriage return, line feed, and period.

CTRL/C interrupts teleprinter output or keyboard input in a user program. Monitor action on a CTRL/C is not taken until any current Monitor command is completed because the keyboard interrupt is turned off. However, except for DUMP and MODIFY, 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 should be used.

If a second CTRL/C is typed before the RETURN key terminating a command, the input so far will be erased, a fresh ↑C will be printed, and the Monitor will await a new command.

2.6.4 The CTRL/U Keys

The CTRL/U key combination is typed by holding down the CTRL key while typing the U key. When CTRL/U is typed, the line on which it is typed is deleted; the system responds with a carriage return and line feed so that the line (command) may be typed again.

CTRL/U is echoed on the teleprinter as ↑U, carriage return, and line feed.

2.6.5 The Semicolon Key

When the Monitor is in listening mode (i.e., following a CTRL/C), the semicolon (;) key causes subsequent characters on the line to be treated as a comment. It effectively puts the keyboard off-line so that all characters following the semicolon are printed on the teleprinter but no Monitor action is taken.

2.6.6 The ESCAPE Key

The ESCAPE key (ASCII Ø33 octal) may be used to pass special keyboard characters to a running user program. When the CSI detects the ESC key it passes the next character directly to the user program. The use of this feature is under programmer control.

2.6.7 How Keyboard Characters are Processed

As characters are typed they are stored in the keyboard buffer (about 85 characters capacity) pending termination of the line with a RETURN, CTRL/C, or CTRL/U, which transfers the line of characters to the Monitor buffer.

When a RUBOUT is processed, it remains in the keyboard buffer and the character which it deletes is replaced with another RUBOUT. Since RUBOUTs are not removed until the line is transferred to the user, the capacity of the keyboard buffer may be exceeded if the sum of normal characters plus RUBOUTs is greater than 85. When this occurs, only RETURN, CTRL/C, or CTRL/U is accepted; all other characters are discarded and not echoed. This is done to maintain economy of core and to ensure that characters such as CTRL/C and CTRL/U can be processed correctly, even when they appear at the end of a very long line.

CTRL/C and CTRL/U characters are processed immediately.

2.7 GETTING ON THE SYSTEM

In order to gain access to the system, the user must log in with the LOGIN command (see section 2.8.11). First, ensure that the terminal is connected to the system (see Appendix H). The LOGIN command is issued in response to the Monitor's \$. If none exist on the teleprinter paper, type the RETURN key and a \$ will be printed by the Monitor; if not, a new Monitor must be loaded as described in the Batch/DOS-11 System Manager's Guide.

In response to \$, the user should issue the LOGIN command with his User Identification Code (UIC) (see section 2.4). For example:

```
$LOGIN 2ØØ,2ØØ
DATE:-2Ø-OCT-72
TIME:-1Ø:41:16
$
```

NOTE

> In the examples, underscoring is used to designate system printout, whereas user input is not underscored.

In response to the LOGIN command, the Monitor prints the current calendar date and time-of-day followed by the \$, indicating that the system is ready for a Monitor command from the user.

Only one user can be logged in at a time. The LOGIN command will be rejected when it is given before the previous user has logged out with the FINISH command.

2.8 MONITOR KEYBOARD COMMANDS

A keyboard 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 letters; all letters after the first two and up to a command name delimiter (space or comma) are ignored.

Monitor keyboard commands are typed in response to a dollar sign ($) or a period (.), which is printed by the system. Generally speaking, the $ indicates that the Monitor is waiting for a new task, and the . indicates that the Monitor is waiting to continue or abort a previously assumed task.

Although the commands are arranged in alphabetical order for ease of reference, they can be divided into functional groups for ease of learning. These groups with their associated commands are as follows:

- Command to allocate system resources:

  ASSIGN

- Commands to manipulate core images:

RUN	GET
DUMP	SAVE

- Commands to start a program:

BEGIN	CONTINUE
RESTART	

- Commands to stop a program:

STOP	WAIT
KILL	

- Commands to exchange information with the system:

DATE	TIME
LOGIN	MODIFY
FINISH	

- Miscellaneous commands:

ECHO	PRINT
END	ODT

The following conventions apply to all Monitor commands:

- All commands are terminated with the RETURN key.
- The command name is separated from its argument (dataset specifier, etc.) with a space.
- All characters in a command are interpreted by the CSI; thus, no embedded blanks are allowed.
- The UIC is always enclosed within square brackets, [ ], except when used with the LOGIN command.

The proper format for each command is given in the discussion of each command in this section. The following conventions apply to the command formats shown in this section.

- Brackets [ ] are used to enclose optional elements
- Braces { } are used to indicate that a choice must be made from the enclosed elements
- The symbol Δ indicates that a space must appear there.
- dev: refers to a device mnemonic (see Appendix A).
- dataset specifier may be represented by any portion of the expression:

  dev:filename.ext,[uic]

  where

dev:	is a legal device mnemonic and colon
filename	is a filename of up to six alphanumerics
.ext	is a period and filename extension of up to three alphanumerics
[uic]	is the user's identification code in the form: [group no., user no.]

- logical name is the name given by the user to the dataset in Link Block word LNKBLK+2 (see Chapter 3).

If for any reason a command cannot be executed satisfactorily, an appropriate message will be printed on the teleprinter and the command will be ignored. These messages are shown in Appendix F.

2.8.1 The ASSIGN Command

Format:

AS[SIGN]△dev:[dataset specifier, logical name]

Purpose:

This command assigns a physical device (and a filename when the device is file-structured) to the dataset identified by "logical name". The format of "dataset specifier" is:

filenam.ext[uic]

which designates the name, extension, and uic, if any, to be assigned to the file.

Any filename specified for a nonfile-structured device is ignored.

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

The ASSIGN command overrides any assignment made in the program's internal control blocks (Link and Filename Blocks). The ASSIGN command is not needed if the program makes its own provisions for obtaining this information; e.g., by specifying defaults in its control blocks or by requesting a command string, as is done with the # symbol in the DOS system programs.

An ASSIGN with no argument releases (deassign) all ASSIGNments previously made by the current user, i.e., since the last LOGIN command

The ASSIGN command can be given at any time the Monitor is in core. Consider the following:

- If ASSIGN is given before a program is loaded, the device assignment will remain in effect until another ASSIGN is given with the same logical name or with no arguments, or until the Monitor itself is reloaded. ASSIGN, given at this time, enables the user to specify an assignment which will apply to several programs.

- If ASSIGN is given after a program is loaded and before it has started running (i.e., after a GET command), the assignment will remain in effect as long as the program is in core, or until another ASSIGNment is performed. When 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 a recovery from an

  A∅∅3 (device not ready)

  message, the user would do an ASSIGN followed by a CONTINUE. The assignment will remain in effect as long as the program is in core, or until the programmer reassigns the dataset, or until he restarts the program with a BEGIN command.

  Doing an ASSIGN in this manner 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 be effective only if the program has not already INITed the dataset to some other device.

2.8.2 The BEGIN Command

Format:

    BE[GIN]△[address]

Purpose:

The BEGIN command starts the execution of an already loaded program at the stated address. If no address is specified, the normal start address will be used. This command is valid only if a program is already in core.

BEGIN is used after a GET, a STOP, or following a fatal error condition. The GET command followed by a BEGIN command is equivalent to a RUN command. If given after a program has been started, a BEGIN will restore core to the state which existed immediately after the program was loaded. It will rename all core allocations to buffers, device drivers, and assignments made dynamically, and the stack will be cleared before control is passed back to the program. If any files are under creation at this time, they will be deleted (see section 2.8.15).

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

    BE

To start a program at absolute address 3446, type:

    BE△3446

After a Program Crash:

The BEGIN Command is provided not only as a means of starting a program loaded by GET but also to enable the user to try again after a program crash, hopefully with a clean slate. At the time of the crash, the program may already have opened but not closed output files and the subsequent request to reopen after a restart could then lead to other failure because these files now exist. To prevent this, the BEGIN processor tries to delete the files, but not by the normal Monitor process since this could mean writing out bit-maps which are currently in core and must be suspect because of the crash. Instead, it merely removes the names of the files from the appropriate device directory, and if these are on disk, unlinks any blocks so far allocated; for safety it does not touch the bit-maps already stored on the device. In almost all cases, this procedure suffices. However, the following implications should be noted.

1. This automatic deletion by BEGIN will not suit a user who has already amassed considerable data in one of his output files and cannot replace it if he starts over. In this case, KILLing the program to save his data under a different filename might be a more appropriate action. However, he should then realize that he might be transmitting the effects of his program failure to the device concerned.

2. It is possible that by the time of the crash the program may have produced a fairly long file. On a DECtape for which there is only one bit-map, this is no problem. A disk, however, requires several bit-maps and the allocation of some of the blocks for the file may already be permanently recorded because the appropriate bit-map has been filled and has been replaced in core by another. Since BEGIN does not change the maps, these blocks will not be freed for further use. A series of situations such as this can, after a time, result in the disk becoming full even though the known files are not seen to occupy the whole capacity. The user should in this case consider whether or not he should chance disk-corruption and use KILL rather than BEGIN. The user can then delete the file by using PIP-11 to avoid the build-up of the nonavailable blocks described.

3. Some programs cannot be restarted with BEGIN (i.e., after having been started, they cannot be restarted with BEGIN.) A FORTRAN program is an example. In general, a program must be self-initialized if BEGIN is to be used in this way. Also, since the Monitor will try to clean up core and delete files, reBEGINning a program which was badly out of control may lead to undesirable results. Thus, use BEGIN <u>only</u> if there is no other alternative.

2.8.3 The CONTINUE Command

Format:

CO[NTINUE]

Purpose:

This command is used after a WAIT command or a recoverable error condition (operator action message) to resume program operation at the point where it was interrupted.

CONTINUE is valid only if a program is already in core.

2.8.4 The DATE Command

Format:

    DA[TE]△[date]

Purpose:

The DATE command may be used to obtain the current calendar date and to enter a date value from the keyboard; the date is printed in the dd-mmm-yy format.

To obtain the current calendar date, simply type the DATE command followed by the RETURN key. For example:

```
$DATE
20-OCT-72
$
```

The current calendar date is entered by the system or installation manager, and need not be reentered except when loading a new DOS Monitor.

To enter a date value from the keyboard, type the DATE command, the desired date value, and then the RETURN key. For example:

```
$DATE △ dd-mmm-yy
```

putting the desired date value in place of dd-mmm-yy. The entered date value is returned in response to subsequent DATE commands until another date is given. If the desired date value is an invalid date, e.g., 42-BOB-A1, subsequent responses to DATE will be meaningless, e.g., ØØ-XXX-YY.

DATE is valid at any time.

DUMP

2.8.5 The DUMP Command

Format:

$$\mathrm{DU[MP]}\triangle\mathrm{LP}\genfrac{}{}{0pt}{}{:}{;}\mathrm{[O]},\ \left[\left\{\begin{matrix}\text{start addr}\\ \emptyset\end{matrix}\right\}\right]\text{[,end addr]}$$

Purpose:

The DUMP command is used to print on the Line Printer an absolute copy of the contents of the specified core area, formatted in octal. The core image is not altered.

The argument O specifies the dump to be output from core. An O is assumed on default, but the comma is required.

The argument Ø is assumed if no "start address" is specified and the highest word in core is assumed if no "end address" is specified.

DUMP is valid at any time. If given while a program is running, the operation of the program will be suspended for the time required to effect the dump.

The syntax of the DUMP command was chosen to facilitate later expansion and flexibility of the command.

2.8.6 The ECHO Command

Format:

EC[HO]

Purpose:

The ECHO command may be used to suppress and restore keyboard echo, i.e., characters typed by the user will not appear on the terminal printer. A subsequent ECHO command turns the echo feature on again. The teleprinter as an output device for the program or the Monitor is not affected by this command.

ECHO is valid only when a program is running in core and using the keyboard as an input device.

END

2.8.7 The END Command

Format:

$$\text{EN[D]}\,\Delta \begin{Bmatrix} \text{KB} \\ \text{PT} \end{Bmatrix}$$

Purpose:

The END command is used to terminate using the console as an input device, i.e., the keyboard or low-speed paper tape reader. The command tells the Monitor "there is no more input from the device". The command effectively generates an end-of-file (EOF) from the keyboard.

When no device is specified in the command, KB is assumed.

The following actions are required with this command

1. Type CTRL/C to obtain the Monitor's attention. Since the console is being used for program input (data mode), the Monitor is not expecting a command.

2. Issue the END command (with appropriate argument).

3. Type the RETURN key twice; yes, two RETURNs. The two RETURNs are required to return to the Monitor.

For example: (where ↑C = CTRL/C, and (CR) = RETURN)

```
↑C
.END,KB    (CR)    (CR)
```

END is valid only when the console is being used as an input device.

2.8.8 The FINISH Command

Format:

FI[NISH]

Purpose:

The FINISH command informs the Monitor that the current user is leaving the system. The Monitor deletes all files which are not protected against automatic deletion on FINISH (see Section 3.9.2.2), and a new copy of the resident Monitor is "booted" into core.

FINISH is valid only when no user program is in core. Therefore, unless the last character on the teleprinter is a $, the user should precede a FINISH with CTRL/C followed by KILL. For example, the printout might be:

```
↑C
.KILL
$FINISH
TIME:-16:42:00
MONITOR VØ8-Ø2
$
```

In response to a FINISH, the Monitor prints the time and then the newly booted Monitor identifies itself. The system is now ready for a user to log in.

2.8.9 The GET Command

Format:

GE[T]Δdataset specifier

Purpose:

The GET command loads the specified file from the specified device. When a device is not specified, the system device is assumed.

GET is valid only when no program is in core.

The user should use a BEGIN or ODT command to commence execution.

2.8.10 The KILL Command

Format:

KI[11]

Purpose:

The KILL command stops the execution of the current program after closing all open files and completing any outstanding I/O. It then removes the program from core by returning control to the Monitor.

KILL is valid only when a program is in core.

To resume operations, the user must reload the program or load another with RUN or GET.

2.8.11 The LOGIN Command

Format:

LO[GIN]△uic

Purpose:

The LOGIN command enables a user to gain access to the system. LOGIN requires a UIC as its argument (see section 2.4). The UIC indicates which directory (of several possible), on each file-structured device, will be directly available to the user.

Here the UIC is not enclosed within the square brackets; its format is simply

nnn,nnn

specifying group, user numbers respectively.

LOGIN is valid only when there is no program loaded in core and provided no user has logged in and not logged out (FINISHed).

2.8.12 The MODIFY Command

Format:

MO[DIFY] △ octal address

octal address/contents: [new contents]

Purpose:

This command allows the user to display and make changes to the contents of the absolute memory location specified by "octal address" in the command line. When the RETURN key is typed at the end of the command line, the system responds by printing the contents of that address. At this point, the user can type one of the following ((CR) = RETURN key; (LF) = LINE FEED key):

(CR)	will leave the contents unmodified.
new contents (CR)	will change contents to new contents.
(LF)	will take similar action as CR and then automatically print the contents of the next memory location.

To change the contents of location 40000:

```
$MODIFY△40000    (CR)
40000/16406:  10406    (CR)
```

Then to examine the contents of 40000:

```
$MO△40000    (CR)
40000/10406:    (CR)
```

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

```
$MO△40000    (CR)
40000/104060:    (LF)
40002/000003:
```

Entry of an address outside the available core memory as part of the original MODIFY command will cause an error, and the command will be rejected.

MODIFY is valid at any time.

2.8.13 The ODT Command

Format:

OD[T]Δ [{R / K}]

Purpose:

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

Argument	Starts at	Action
(none)	START+Ø	Clears ODT breakpoint table without resetting breakpoints.
R	START+2	Clears ODT breakpoint table after replacing old instructions at breakpoints.
K	START+4	Leaves breakpoints exactly as they are.

This command begins execution at the ODT entry point of the user's load module. The user must have linked ODT-11R with his program and must have identified his program to the Linker with the /OD switch.

To reset all breakpoint locations at their former instructions and restart ODT, the user would type:

.ODΔR

ODT is valid only when ODT-11R is linked to a program and both are in core.

2.8.14 The PRINT Command

Format:

PR[INT]

Purpose:

The PRINT command may be used to suppress and restore teleprinter printing when the printer is used as an output device to a user program. A subsequent PRINT command turns the printing feature on again.

PRINT is valid only when a program is running in core and is using the teleprinter as an output device.

2.8.15 The RESTART Command

Format:

RE[START]△[address]

Purpose:

The RESTART command permits a program to be restarted. As shown, the user may optionally supply an address at which the program is to be restarted. If no address is specified, the address set by the .RESTART programmed request is assumed if a .RSTRT request has been issued by the program (see Section 3.8.3.2).

If neither address is specified, the command is rejected.

RESTART is valid only when a program is already in core.

Before the program is restarted, the stack is cleared, any current I/O is stopped, and all internal busy states are removed. Buffers and device drivers set up for I/O operations will, however, remain linked to the program for future use.

2.8.16 The RUN Command

Format:

```
RU[N] Δdataset specifier
```

Purpose:

The RUN command loads into core the specified program from the specified device and starts its execution at the normal start address. RUN is equivalent to a GET command followed by a BEGIN command.

The dataset specifier is of the form:

```
dev:filenam.ext[uic]
```

When no device is specified, the system device (disk) is assumed.

The sequence in which the Monitor performs its search for the specified program depends on the existence and type of filename extension and on the UIC. Various forms of the RUN command are shown below with the search sequence performed by the Monitor.

- RUNΔFILE

```
Attempt 1 -- FILE.LDA [current uic]
Attempt 2 -- FILE.LDA [1,1]
Attempt 3 -- FILE [current uic]
Attempt 4 -- FILE [1,1]
```

- RUNΔFILE.EXT

```
Attempt 1 -- FILE.EXT [current uic]
Attempt 2 -- FILE.EXT [1,1]
```

- RUN△FILE[x,x]

```
Attempt 1 -- FILE.LDA [x,x]
Attempt 2 -- FILE [x,x]
```

- RUN△FILE.EXT[x,x]

```
Attempt 1 -- FILE.EXT [x,x]
```

If all attempts fail to find the file, a NO FILE message is printed on the teleprinter.

Searching for the LDA extension first exploits the fact that both the Linker and the SAVE command produce LDA extensions, unless the user specifies otherwise.

RUN is valid only when there is no program in core.

2.8.17 The SAVE Command

Format:

SA[VE]Δ[dataset specifier][/RA:low:high]

Purpose:

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

If no dataset specifier is given, the SAVE processor will automatically set up a file called SAVE.LDA on the system disk after it has deleted any current file of the same name. If the user wishes to retain the current file, he must first rename it using PIP-11. If the dataset specifier is given, the file named must not already exist or the command will be rejected. System disk is assumed by default if the dataset specifier contains only a filename. When the filename is specified, the extension should also be specified.

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

/RA:low:high

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

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

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

DEVICE FULL	End of output medium reached
FILE ERROR xxx	File structures error as indicated by xxx = file status byte

STOP

2.8.18 The STOP Command

Format:

ST[OP]

Purpose:

This is an emergency command to stop the program and to abort any I/O in progress (by doing a hardware reset). The program may be resumed with either the BEGIN or RESTART command.

STOP is valid only if a program is in core.

STOP differs from KILL in that KILL terminates the program in an orderly manner and removes the program from core.

2.8.19 The TIME Command

Format:

```
TI[ME]Δ[time]
```

Purpose:

The TIME command may be used to obtain the current time-of-day and to enter a time value from the keyboard. The time is printed in the following format.

```
hh:mm:ss
```

meaning hours:minutes:seconds.

To obtain the current time-of-day, simply type the TIME command followed by the RETURN key. For example:

```
$TIME
10:43:27
$
```

The current time-of-day is entered by the system or installation manager, and need not be reentered except when loading a new DOS Monitor.

To enter a time value from the keyboard, type the TIME command, the desired time value, and then the RETURN key.
For example:

```
$TIMEΔhh:mm:ss
```

putting the desired time value in place of hh:mm:ss. The entered time value is returned in response to subsequent TIME commands until another time value is given.

TIME is valid at any time.

2.8.20 The WAIT Command

Format:

WA[IT]

Purpose:

The WAIT command suspends the current program and allows any I/O in progress to finish. The program may be resumed with either the CONTINUE or RESTART command.

WAIT is valid only if a program is in core.

CHAPTER 3
PROGRAMMED REQUESTS

3.1 INTRODUCTION

The Monitor provides a number of services which are available to any user or system program. The most prominent of these are input/output (I/O) services. Other services include directory management, retrieval and modification of system parameters, various conversion routines, and a command string interpreter. The I/O services provide for linkage to device drivers, access to files in the file structure, and transfer of data to or from each device.

The user program calls for the services of the Monitor through programmed requests. Programmed requests are macro calls[1] which are assembled into the user program and interpreted by the Monitor at execution time. A programmed request consists of a macro call 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 request is expanded at assembly time by the MACRO Assembler[1] into a sequence of instructions which trap to and pass the arguments to the appropriate Monitor service routine to carry out the specified function. The assembly language expansion for .WAIT #LNKBLK is:

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

To use the macro call, it is necessary to tell the assembler that you want the system definition for the macro. This is accomplished via the .MCALL assembler directive (Macro-11 Assembler Programmer's Manual), e.g.,

```
.MCALL    .WAIT
```

which must appear in the source prior to the first use of .WAIT. When .MCALL is encountered, the MACRO Assembler will get the definition of .WAIT from the system macro file (SYSMAC.SML) which is searched for, first in the current user's disk area, then under user identification code [1,1].

The system macros will accept most addressing modes as arguments. They will detect and announce potentially troublesome (e.g. X(SP)) or unlikely (e.g. SP) modes to protect the user.

---

[1]Users with less than 12K of core cannot run MACRO and consequently must include the assembly language expansion of the programmed request in their programs instead of the request itself.

All legal addressing modes will appear without alteration in the expansion. Since the monitor expects the address of the Link Block on top of the stack at .WAIT time, any of the following macro calls might be appropriate:

```
.WAIT #LNKBLK
.WAIT RØ          ;ADDRESS OF LNKBLK
                  ;IS IN REGISTER RØ
.WAIT POINTR      ;ADDRESS OF LNKBLK IS
                  ;IN MEMORY LOCATION POINTR
```

Refer to the MACRO-11 Assembler Programmer's Manual (Order Number DEC-11-OMACA-A-D) for further details.

The programmed request arguments are parameters or addresses of tables which contain the parameters of the request. These tables are part of the user program, and are described in detail in Figures 3-6 to 3-18.

3.2 TYPES OF PROGRAMMED REQUESTS

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 3-1 summarizes the programmed requests available under the Monitor. Detailed descriptions of each request can be found in the sections cited in Table 3-1.

Table 3-1
Summary of Monitor Requests

Mnemonic	Purpose	Section
Requests for Input/Output and Related Services:		
.INIT	Associates a dataset with a device driver and sets up the initial linkage.	3.6.1
.RLSE	Removes the linkage between a device driver and a dataset, and releases the driver.	3.6.2
.OPEN	Opens a dataset.	3.6.3
.CLOSE	Closes a dataset.	3.6.4
.READ	Transfers data from a device to a user's line buffer.	3.6.5
.WRITE	Transfers data from a user's line buffer to a device.	3.6.6
.RECRD	Transfers one logical record of a file between a device and a user buffer.	3.6.7
.BLOCK	Transfers one physical block of a file between device and a Monitor buffer.	3.6.8
.TRAN	Transfers data between a device and a user buffer, independent of any file structure.	3.6.9
.WAIT	Waits for completion of any action on a dataset.	3.6.10
.WAITR	Checks for completion of any action on a dataset, and provides a transfer address for a busy return.	3.6.11
.SPEC	Performs special device functions.	3.6.12
.STAT	Obtains device characteristics.	3.6.13
Requests for Directory Management Services:		
.ALLOC	Allocates a contiguous file.	3.7.1
.DELET	Deletes a file.	3.7.2
.RENAM	Renames a file. Changes a protection code.	3.7.3
.APPND	Appends one linked file to another.	3.7.4
.LOOK	Searches the directory for a particular filename and returns information about the file.	3.7.5
.KEEP	Protects a file against automatic deletion on FINISHing.	3.7.6
Requests for Miscellaneous Services:		
.RUN	Loads programs and overlays.	3.8.1.1
.EXIT	Returns control to the Monitor.	3.8.2.1

(Continued on next page)

Table 3-1
Summary of Monitor Requests (Cont.)

Mnemonic	Purpose	Section
.TRAP	Sets interrupt vector for the TRAP instruction.	3.8.3.1
.RSTRT	Sets the address used by the RESTART command.	3.8.3.2
.CORE	Obtains address of highest word in core memory.	3.8.4.1
.MONR	Obtains address of first word above the resident Monitor.	3.8.4.2
.MONF	Obtains address of first word above the Monitor's highest allocated free core buffer.	3.8.4.3
.DATE	Obtains the date.	3.8.4.4
.TIME	Obtains the time of day.	3.8.4.5
.CVTDT	Converts internal date or time to ASCII.	3.8.4.6
.GTUIC	Gets current UIC.	3.8.4.7
.SYSDV	Gets Radix-50 name of the system device.	3.8.4.8
.GTPLA	Gets the current program load address.	3.8.4.9
.STPLA	Sets the program low address.	3.8.4.10
.GTCIL	Gets the base disk address of the CIL.	3.8.4.11
.GTSTK	Gets the current stack base address.	3.8.4.12
.STSTK	Sets the current stack base address.	3.8.4.13
.STFPU	Sets the floating point exception vector.	3.8.4.14
.RADPK	Packs three ASCII characters into one Radix-50 word.	3.8.5.1
.RADUP	Unpacks one Radix-50 word into three ASCII characters.	3.8.5.2
.D2BIN	Converts five decimal ASCII characters into one binary word.	3.8.5.3
.BIN2D	Converts one binary word into five decimal ASCII characters.	3.8.5.4
.O2BIN	Converts six octal ASCII characters into one binary word.	3.8.5.5
.BIN2O	Converts one binary word into six octal ASCII characters.	3.8.5.6
.CSI1	Condenses a command string and checks for proper syntax.	3.8.6.1
.CSI2	Interprets one command string dataset specification.	3.8.6.2

3.2.1 Requests for Input/Output and Related Services

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

- READ or WRITE
- RECORD or BLOCK
- TRAN

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

Each request related to I/O services is described in Section 3.6.

3.2.1.1 READ or WRITE Level Requests - Most input and output is done at this level. Processing is sequential, in that each read or write is applied to the next record or line in the file. Records may be in either ASCII or binary mode, and a number of formats are handled by the monitor. Records may also be of variable length: ASCII records usually contain line terminators while formatted binary records contain byte counts.

READ or WRITE I/O under the Monitor consists of transferring the contents of a dataset between a device and a line buffer via a buffer in the Monitor (Figure 3-1a). 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 is usually 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.

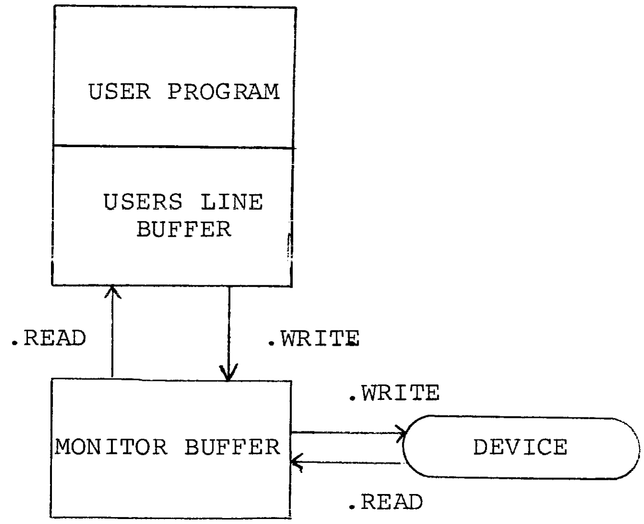


Figure 3-1a The Transfer Path

```
        .INIT    #LNKBLK

        MOV      #FILBLK,RØ

        .OPENI   #LNKBLK,RØ               ;FOR OUTPUT, REPLACE .OPENI
                                          ; WITH .OPENO
                                          ;ADDRESS OF FILBLK IS IN RØ

   ---> .READ    #LNKBLK,#BUFHDR          ;FOR OUTPUT, REPLACE .READ
  |                                       ; WITH .WRITE
  |     .WRITE   #LNKBLK                  ;COULD BE REPLACED BY .WAITR
  |                 :
  |      (Process Data)
  | No
   ----- End Of Data ?
              |
              | Yes
              v
        .CLOSE   #LNKBLK
        .RLSE    #LNKBLK

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

Figure 3-1b Sequence of Requests for READ/WRITE

Figure 3-1 .READ/.WRITE Input/Output Transfers

When using READ or WRITE one can specify nine different modes of transfer, in two categories: ASCII and Binary. Details are presented in Section 3.6.1 and Figure 3-11.

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

To implement a READ or WRITE transfer, the programmer follows the sequence of requests shown in Figure 3-1b. First, the programmer associates the device with the dataset via 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 3-6. The .INIT request loads the appropriate device driver into the Monitor's free core 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 is a file-structured device. However, it is advisable to use an .OPENx even for a non-file-structured device to preserve the device independence of the program, since it may be desirable to assign the transfer to a file-structured device later. The arguments of this request are the address of the Link Block and a register into which the user has moved the address of a table called the Filename Block (Figure 3-7). Entries in this table describe the file involved in the transfer.

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

A .READ (for input) or a .WRITE (for output) follows 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 3.9). The Line Buffer Header specifies the area in the user's core area to or from which the data is to be transferred. During the transfer, the Monitor formats the data according to the transfer mode and formatting characters in the data itself. In most modes, terminating characters indicate the end of a line.

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

3.2.1.2 RECORD Level Requests - The Record Level request is used for random access to the records in a file. A program which uses Read or Write Level requests can only read or write the next record in the dataset being processed. When Record Level requests are used, the program always has access to any record in the file.

Record Level requests may be used only with file-structured devices and only with contiguous files (not with linked files). Each of the records in the file must contain the same numbers of bytes. No formatting is done and no line terminating characters are needed. The length of a record is independent of the block size of the device (may be the same or smaller or larger; neither record length nor block size need divide the other, but processing may be faster if this is arranged, since it can reduce the number of multi-block transfers).

Some consideration must be given to the manner in which a Record Level file is created. Perhaps the most common way to create such a file is by doing an .OPENC (after the file has been allocated) and using the .WRITE request to enter data. Unformatted ASCII and unformatted binary are the suggested transfer modes, since they do not require terminators and do not perform formatting; recall that all records must be the same length. When such a file is .CLOSED, a logical end-of-file is established following the last record written. Subsequent processing of the file by .READ or .RECRD will be confined to the area just written. At some later time, the file may be opened for extension (.OPENE) and more data can be written (.WRITE), provided the original space allocated to the file is sufficient to contain it. A second way to create a Record Level file is to start with .OPENU (again the file must have been allocated previously) and to use .RECRD to do the writing. In this mode, the logical end-of-file corresponds to the end of the allocated area). Note also that, unless the program writes in every record of the file, that some records will be left with meaningless contents.

Before issuing Record Level requests, the program must issue an .INIT request to associate the dataset with a file-structured device. It must then open the dataset; .OPEN is not optional as with .READ and .WRITE. The dataset may be opened in two ways:

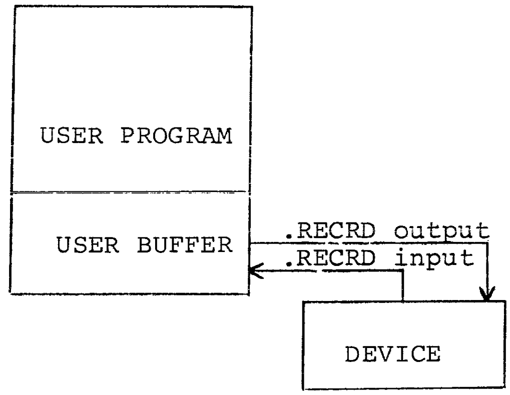


Figure 3-2a The Transfer Path

```
               .INIT    #LNKBLK            ;INIT THE DATASET
               MOV      #FILBLK,R1
               .OPENU   #LNKBLK,R1         ;OPEN THE FILE
Request next  →.RECRD   #LNKBLK,#RECBLK    ;INPUT THE RECORD
record for     .WAIT    #LNKBLK
update if      (Process Data)
more
               .RECRD   #LNKBLK,#RECBLK    ;OUTPUT THE RECORD
               .WAIT    #LNKBLK
          Yes    More
                  ?
                    No
               .CLOSE   #LNKBLK            ;CLOSE THE FILE
               .RLSE    #LNKBLK            ;RELEASE THE DATASET

       LNKBLK:
       FILBLK:  }  Tables in User Program
       RECBLK:
```

Figure 3-2b Sequence of Requests for .RECRD

Figure 3-2 .RECRD Input/Output Transfers.

- OPENU - This mode is used if the program will write in the dataset. Reading is also permitted. In fact, quite often the program will read a record, update it, and write it back.
- OPENI - This mode is used if no writing will be done. Only reading will be permitted.

The dataset may then be processed using .RECRD requests. If updating is being done, there will generally be two such requests in each cycle. Otherwise, there will be only one. Each .RECRD request should be followed by a .WAIT (or .WAITR) request. When processing is completed, a .CLOSE request should be issued to ensure that the last record is actually written to the device (for output) and that the directory is updated (if necessary). A .RLSE request is also required, so that the driver can be removed from core (if not still in use by another dataset). The .RECRD request has a Link Block and a Record Block as arguments. The Record Block specifies function (input/output), buffer address, record length, and record number (see Figure 3-12).

3.2.1.3. BLOCK Level Requests - The Block Level request is used for random access to the physical blocks in a file. The Block Level is similar to the Record Level. However, at the Block Level, each request always reads or writes exactly one physical block of data instead of a user-defined quantity of data, as is true at the Record Level. In addition, data transfer is to and from a buffer provided by the monitor, rather than a buffer provided by the user. The user may do his processing in the monitor buffer or he may transfer data to his own area. As with Record Level requests, Block Level requests may be used only with file-structured devices and only with linked files (~~not with~~ contiguous files).

To implement a BLOCK transfer, the programmer follows the sequence of requests shown in Figure 3-3b. Notice that the transfer must use .INIT, .OPEN, .WAIT, .CLOSE and .RLSE following the same rules as the READ or WRITE level. The .BLOCK request has the address of the Link Block and the BLOCK block for its arguments.

The BLOCK block specifies the function (INPUT, GET, or OUTPUT), the relative number of the block being transferred to or from, the Monitor buffer address (supplied by the Monitor), and the length of the Monitor buffer (supplied by the Monitor). See section 3.6.8.

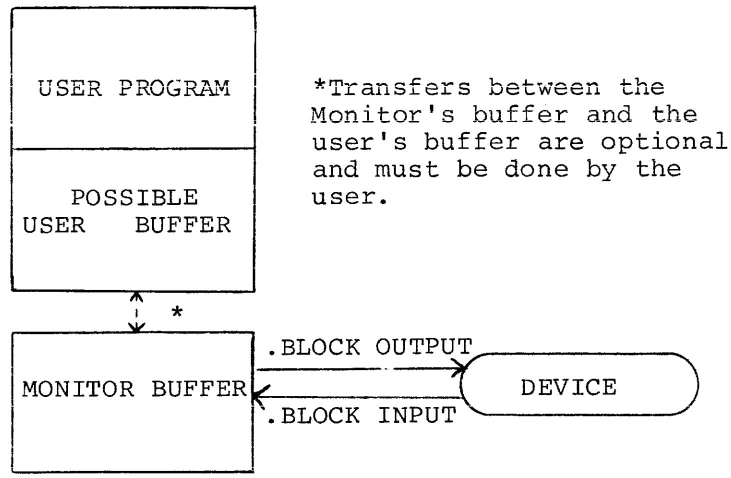


Figure 3-3a The Transfer Path

```
        .INIT    #LNKBLK
        MOV      #FILBLK,R1
        .OPENU   #LNKBLK,R1
   ---> .BLOCK   #LNKBLK,#BLKBLK     ;INPUT DESIRED BLOCK
   |    .WAIT    #LNKBLK             ;COULD BE REPLACED BY .WAITR
   |
   |    (Process Data)               ;UPDATE DATA
   |
   |    .BLOCK   #LNKBLK,#BLKBLK     ;WRITE UPDATED BLOCK
   |    .WAIT    #LNKBLK
   |
  Yes <---- More ?
              |
              v  No
        .CLOSE   #LNKBLK
        .RLSE    #LNKBLK
        LNKBLK:          (entries)
        FILBLK:          (entries)
        BLKBLK:          (entries)
```

Figure 3-3b The Sequence of Requests For .BLOCK

Figure 3-3 .BLOCK Input/Output Transfers

3.2.1.4 TRAN Level Requests - A TRAN level request is a basic input/output operation. No services are provided for the user other than to pass his request to the appropriate driver. .TRAN ignores any file-structure on the device. .TRAN does not operate within a particular file as do .READ, .WRITE, .RECRD, and .BLOCK; hence no .OPEN or .CLOSE is used. Because .TRAN does not respect file structures, the user is strongly cautioned against using it with file-structured devices, since he can easily do irreparable damage to information on such a device. Omitting the dataset name from the Link Block will prevent a file-structured device from being assigned.

Data is transferred directly between the device and a buffer provided by the user (Figure 3-4a), with no formatting performed.

.TRAN is generally used in 2 situations:

1. When the file structure does not allow the desired operation (e.g., PIP uses .TRAN to read a directory block for the directory listing operation).
2. When one does not need or cannot afford the overhead of doing READ/WRITE processing on a non-file structured device (e.g, a program to read data arriving at random intervals from an A/D converter might use .TRAN to read the data and .BLOCK to buffer the data on a disk for processing as time permits.

To implement a TRAN transfer, the programmer follows the sequence of requests shown in Figure 3-4b. Notice that the transfer must use .INIT and .RLSE, but must not use .OPEN or .CLOSE. The .TRAN request has the address of the TRAN Control Block (TRNBLK) as its argument. This block contains entries which specify the core starting address of the user's 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.

Table 3-2

Transfer Levels for Types of Datasets

Type of Transfer	Type of Data: Linked File	Contiguous File	Nonfile-Structured Device
READ/WRITE	Yes	Yes	Yes
RECORD	No	Yes	No
BLOCK	No	Yes	No
TRAN	*	*	Yes

* indicates that TRAN may be used on a file-structured device if the warnings mentioned are observed. Usage in these cases is not advised.

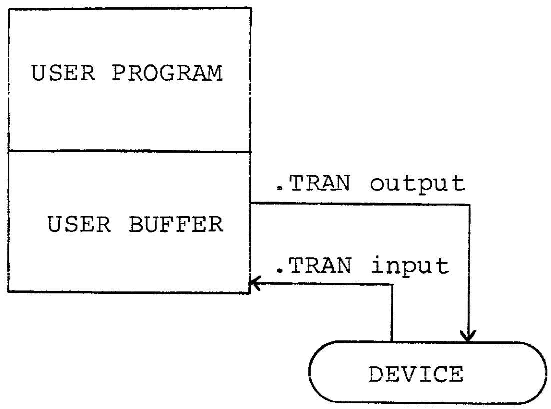


Figure 3-4a The Transfer Path

```
        .INIT #LNKBLK

    --> .TRAN #LNKBLK,#TRNBLK
   |
   |    .WAIT #LNKBLK                  ;COULD BE REPLACED BY .WAITR
   |
   |     (Process Data)
   |
Yes <--- More
          ?
          | No
          v
        .RLSE #LNKBLK

        LNKBLK:         (entries)
                                       Tables and parameters
        TRNBLK:         (entries)      in User Program
```

Figure 3-4b The Sequence of Requests For .TRAN

Figure 3-4 .TRAN Input/Output Transfers

3.2.2 Requests for Directory Management Services

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

Each directory management request is described in Section 3.7.

3.2.3 Requests for Miscellaneous Services

Requests for miscellaneous services include:

- Requests to Load programs and overlays.
- Requests to return control from a running program to the Monitor.
- 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 the Monitor, the date, the time, and the current user's UIC.
- Requests to perform conversions between ASCII and Radix-50 packed ASCII, binary and ASCII decimal, and binary and ASCII octal.
- Requests to access the Command String Interpreter.

Each miscellaneous service request is described in Section 3.8.

3.3 DEVICE INDEPENDENCE

It is generally preferable to write programs so that each dataset may be associated with the widest possible variety of devices. This makes it easier to move a program from one configuration to another. It also makes it possible to use the program with a variety of different media. For example, the Assembler accepts input from disk, paper tape, DECtape, and other devices.

The monitor makes it relatively easy to achieve this objective. Most I/O operations are completely device independent (i.e., no special actions by the user are required to accommodate the operation to the device, specifically .READ, .WRITE, .OPEN, .CLOSE, .WAIT, .WAITR, .INIT, and .RLSE. In addition, .RECRD and .BLOCK require only that the device be file structured. Only .TRAN and .SPEC are typically device dependent.

In all cases, no device is associated with a dataset until an .INIT request is made. The device name may be specified in any of the following ways:

- the programmer may specify the name in his Link Block;
- the program can obtain a device name by requesting the user to enter a command string (section 3.8.6);this will override any device specified in the Link Block;

- the user can use the ASSIGN command (see Chapter 2) to associate a device (and file name) with the dataset, this option overrides both preceding options.

Note that when a command string is solicited by the program, it will always override the link block specification, no matter what is entered. However, when ASSIGN is not solicited but is entered at the operator's discretion, it will override the Link Block only when specified. In the latter case, it is best to supply a default in the Link Block.

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

It is important to note that a device is assigned in a program to a dataset logical name and that 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 requests which are not device independent are .TRAN and .SPEC.

3.4 SWAPPING ROUTINES INTO CORE

Except for a small, permanently resident portion, 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 only 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.

Any routine which services a programmed request (other than .READ or .WRITE) may be made core resident by one of the following methods:

- Routines may be made permanently core resident at Monitor Generation time (see the DOS System Manager's Guide).

- Routines may be made core 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 3.6 through 3.8) in a .GLOBL assembler directive in the user program. For example, to make the .TRAN processor resident while program FROP is being run, the following directive would be included in program FROP:

  .GLOBL TRA

Device drivers are swapped into the Monitor's free core area on an .INIT call and are freed from core on the occurrence of a .RLSE, provided no other dataset is INITed to that device.

3.5 MONITOR RESTRICTIONS ON THE PROGRAMMER

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

- The programmer should not use either the EMT or the IOT instructions for communication within his program.
- It is recommended that the user not raise his interrupt priority level above 3, since it might lock out a device that is currently trying to do input/output.
- HALTS are not recommended. If a HALT is executed during an I/O operation, most devices will stop, and only recovery from the console (pressing the CONTinue switch on the console) will be effective (recovery from the keyboard will not be immediately possible, since a HALT inhibits the keyboard interrupt). Some devices, such as DECtape, will not see the HALT and will continue moving, will lose their positions over the block under transfer, and may even run the tape off the reel.
- The RESET instruction should not be used because it forces a hardware reset; clearing all buffer registers, and status flags and disabling all interrupts, including keyboard interrupts. Since all I/O is interrupt driven, RESET will disable the system.
- The user must not penetrate 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 may allocate temporary storage areas on the stack by simply subtracting the size of the area needed from the current stack pointer value. When doing so, he should use a .MONF (Section 3.8.4.3) to determine the highest address being used by the Monitor. It is generally wise to leave some space for future Monitor expansion (as a consequence of programmed requests) and for stack extension (as a consequence of subroutine calls, Monitor requests, device interrupts, etc.). Consult Figure 3-5 for more information about monitor core usage.

- The user should be aware that certain requests, such as .INIT, may change the amount of available free core, since the instructions may call in drivers and establish data blocks. Such requests affect the result of 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 listed in Appendix E.
- The Link pointer in the Link Block is set by the Monitor and must not be altered by the user.

3.6 REQUEST FOR INPUT/OUTPUT SERVICES

3.6.1 .INIT - Associate a dataset with a device driver and set up the initial linkage.

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 assures that the appropriate driver exists and is in core. If the driver is not in core, it is loaded. The device assigned is that specified in the associated Link Block, unless assignment has been made to the logical name specified in the Link Block with the ASSIGN command or via the Command String Interpreter. 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 within his program a Link Block of the format explained in section 3.9.1 for each dataset to be INITed. A dataset which has been .INITed should be .RLSEd prior to any further .INIT request for any Link Block.

Errors: A nonfatal error message, A003, is printed on the teleprinter if no assignment has been made through the ASSIGN command, and the DEFAULT DEVICE is either not specified in the Link Block or has been specified illegally (i.e., no such device on the system). The user may type in an assignment (ASSIGN) and give the CONTINUE console command to resume operation.

Control is transferred to the address specified by the error return address in the Link Block if at any time during an operation there is not enough space in free core for the necessary drivers, buffers, or tables. If no address (i.e., a zero) is specified in the Link Block's ERROR RETURN ADDRESS, a fatal (F007) error is printed and the program stops.

Example: (see .RLSE).

3.6.2 .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, and provided that the driver is not permanently core resident.

Rules: The device to be released must have been previously INITed to the dataset.

If the dataset has been OPENed on a directory device, it must be CLOSEd before the device is released. On a nondirectory device, a .RLSE will ensure that any data remaining in the Monitor buffer for output is dispatched to the device and will return any buffer still associated with the dataset to free core.

After the release has been completed, control is returned to the user at the instruction following the assembly language expansion; the argument is removed from the stack.

Errors: If the dataset has been OPENed to a file-structured device, a .RLSE not preceded by a .CLOSE will be treated as a fatal error, F005. A .RLSE error (F005) may also occur if the link pointer in the Link Block is invalid, indicating probable corruption of the Monitor or its control blocks.

Example:

```
                :
                .INIT   #LNK1     ;ASSOCIATE A DATASET WITH A DEVICE
                :
                .RLSE   #LNK1
                :
                .WORD   ERR1      ;ERROR RETURN ADDRESS
        LNK1:   .WORD   0         ;POINTER FOR MONITOR
                .RAD50  /DSI/     ;LOGICAL NAME OF DATASET
                .BYTE   1,0       ;DEVICE SPECIFIED, UNIT
                .RAD50  /KB/      ;SPECIFY KEYBOARD
                :
        ERR1:                     ;ERROR PROCESSING LOGIC
```

.OPEN

3.6.3 .OPEN - Prepare a device (which has been .INITed) for data transfer and associate the dataset with a file (if the device is file-structured).

Macro Call: .OPEN #LNKBLK,#FILBLK

This form assumes that the File Block contains a code indicating how the file is to be opened (see Description below).

Assembly Language Expansion:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 16
```

Alternate Form of Macro Call:

.OPENx #LNKBLK,Rn

where Rn is a register containing the address of the File Block and x indicates the type of .OPEN (see Description below).

Assembly Language Expansion:

```
MOVB #CODE,-2(Rn)       (see Description below)
MOV  Rn,-(SP)
MOV  #LNKBLK,-(SP)
EMT  16
```

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

Description: When used, .OPEN follows .INIT or .CLOSE (if more than one file is to be opened on the same dataset). When the device being used is file-structured, .OPEN associates a specific file with the dataset. .OPEN also acquires a data buffer and prepares the device or the file for the ensuing data transfers. See Appendix C for details about specific .OPEN actions for particular devices. .OPEN has five forms; the desired form may be specified by inserting the proper HOW OPEN code in the File Block (see Figure 3-7) or by selecting one of the alternate forms of the Macro Call. The different .OPEN forms are described below:

Form	HOW OPEN Code	Description
.OPENU	1	opens a previously created contiguous file for input and output by .RECRD or .BLOCK request; .OPENU is rejected if the device is not file-structured.
.OPENO	2	a. creates a new linked file and prepares it for output via .WRITE; the file must not already exist. b. prepares a nonfile-structured device for output via .WRITE (see Appendix C).

Form	HOW OPEN Code	Description
.OPENE	3	opens a previously created linked or contiguous file to make it longer via .WRITE; note that a contiguous file may only be extended within the area already allocated; although additional blocks may be added to a linked file, no additional blocks may be added to a contiguous file (see .CLOSE); .OPENE is treated like .OPENO if the device is not file-structured.
.OPENI	4	a. opens a previously created linked or contiguous file for input via .READ, .RECRD, or .BLOCK. b. prepares a nonfile-structured device for input via .READ (see Appendix C).
.OPENC	13	opens a previously created contiguous file for output via .WRITE; when a contiguous file is first opened for writing (via .WRITE), .OPENC must be used; subsequent opens for output (via .WRITE) must be .OPENE's; .OPENC is treated like .OPENO if the device is not file-structured.

At this point, the user should note the difference between linked files and contiguous files. A linked file has records allocated to it one at a time, as they are needed. Each record in the file contains a pointer to its successor, the User File Directory (UFD) points to the first record. Because records are allocated as needed, the user need not concern himself at all with the size of the file nor with the allocation of any records. Furthermore, a linked file can easily be extended in the future. However, because records are scattered about on the disk and because the system must read all intermediate records to move from one record to another (forward only), linked files can only be used for sequential processing (.READ or .WRITE).

A contiguous file has all of its records allocated at once in a contiguous area of the disk which is reserved for the file. Since any record in the file can easily be located relative to the first record in the file, random (or direct) access (.RECRD or .BLOCK) is possible in addition to sequential access. However, it is now necessary to know in advance how much space will be needed, since no more space can be added later. Since this may be difficult, one often has to guess and space is often wasted. Note, however, that a contiguous file can be extended within the space already allocated, i.e., if the area was not filled when the file was first written (or extended), more data can be added. Because the user is responsible for determining the size of a contiguous file, he is required to allocate it before opening it (compare .OPENC and .OPENO). This may be done with PIP, using the ALLOCATE command or with the .ALLOC programmed request.

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 3-3.

Table 3-3
Transfer Requests Which May Follow Open Requests

Type of File	Linked File		Contiguous File				
Type of Transfer	Input	Output	Input		Output		File Already Exist ?
Transfer Request / Type of Open	.READ	.WRITE	.READ	.RECRD .BLOCK	.WRITE	.RECRD .BLOCK	
.OPENU				Yes		Yes	Must
.OPENO		Yes					Must Not
.OPENE		Yes					Must
.OPENI	Yes		Yes	Yes			Must
.OPENC					Yes		Must

<u>Rules</u>: a. <u>General Rules for All .OPENx Requests</u> - The user must set up a Filename Block in his program (see Figure 3-7). If the dataset is a file, the Filename Block must contain a legal filename (see Section 2.3). If the dataset is not a file, or if it will be specified by an .ASsign or via the Command String Interpreter, the Filename Block need not contain any FILENAME or EXTENSION entries.

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

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

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

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

b. Rules for .OPENO - The .OPENO request is applicable only for outputs to nonfile-structured devices or to a linked file on a file-structured device. It is not applicable to contiguous files.

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

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

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

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

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

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

d. Rules for .OPENU, OPENE, and .OPENC - The file must exist and cannot currently be opened for output.

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

A contiguous file can be opened for extension, provided that the area already allocated to the file does not need to be enlarged, which is not possible.

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 3-7) and transfers control via the pointer in the ERROR RETURN ADDRESS of the Filename Block. If this address is 0, a fatal error message is printed on the teleprinter. Fatal error messages are listed in Appendix F.

Example: (See .CLOSE)

3.6.4 .CLOSE - Close a dataset.

Macro Call: .CLOSE #LNKBLK

where LNKBLK is the address of the Link Block (see Figure 3-7).

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 (e.g., on output, it writes the last buffer; on extension, it links the extension to the old file; etc.), updates any directories affected by the processing, and releases to free core any buffer space established for the processing. When a file which has been opened for output is closed, the last block written and the last byte written are recorded in the directory to indicate end-of-data. This eliminates the need to pad out blocks with nulls and allows the written data within a contiguous file to be extended at a later time.

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 (see Appendix C).

Rules: The dataset to be closed must have previously been opened if it was a file on a file-structured device.

As with .OPENx, a .CLOSE is not required if the dataset is not a file, but it is strongly recommended in order to maintain device independence.

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

All error messages are explained in Appendix F.

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

```
            .INIT #SET1
               :
            .OPEN   #SET1,#FILE1   ;OPEN SET1 FOR INPUT (OPEN CODE
               :                   ;IS IN FILE BLOCK)
            (Input is
             Performed
             Here)
               :
            .CLOSE #SET1           ;CLOSE SET1
               :
            .RLSE   #SET1
               :
            .WORD   ERR1
SET1:       .WORD   0
            .RAD50 /IMP/           ;DATASET NAME
            .BYTE   1,3
            .RAD50 /DT/            ;PHYSICAL DEVICE NAME
               :
            .WORD   ERF1           ;ADDR OF ERROR RTN
            .WORD   4              ;OPEN FOR INPUT
FILE1:      .RAD50 /PRO/           ;FILENAME
            .RAD50 /G1/
            .RAD50 /BIN            ;EXTENSION
            .BYTE    PROG,PROJ
            .BYTE    177
            .EVEN
               :
ERR1:                              ;HERE FOR .INIT, .OPENI, .CLOSE,
                                   ;OR .RLSE ERRORS (DEVICE)
               :
ERF1:                              ;HERE FOR .OPENI ERRORS
                                   ;(DATA FILE)
               :
```

3.6.5 .READ - Read the next record in the dataset.

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 (Routine is permanently core resident).

Description: The .READ request transfers the data from the device to the user's line buffer as specified in the line buffer header. 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. Each read causes the user to receive the next record in the data set. Block boundaries are ignored and new blocks are read as needed. 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.

Refer to Section 3.9.3.2 for more details on transfer modes.

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 3-9). 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. Otherwise, he might find that he is processing before the data he wants has arrived.

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

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

3.6.6 .WRITE - Write the next record in the dataset.

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 (Routine is permanently core resident).

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.[1] The data in the Monitor buffer is then transferred to the next device block, and any data remaining in the user's line buffer is moved to the (now 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.

Refer to Section 3.9.3.2 for more details on transfer modes and the like.

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 3-9).

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 ensure proper completion of the transfer. Otherwise, it might store new data on top of data which has not yet been written.

Errors: See .READ for errors.

---

[1]For terminal devices, data transfer also occurs when a line terminator is seen (see Section 3.9.3.2).

.RECRD

3.6.7 .RECRD - Read or write a specific record in a file.

Macro Call: .RECRD #LNKBLK,#RECBLK
where LNKBLK is the address of the Link Block, and RECBLK is the address of the Record Block (see Figure 3-12).

Assembly Language Expansion:

```
MOV #RECBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 25
```

Global Name: REC

Description: The .RECRD request causes a specific record to be transferred to (or from) the user's record buffer. Each record in the file may be individually addressed, and the user is not restricted to reading or writing the next record. Data transfer is by way of a buffer in the Monitor which will contain exactly one physical block of information. There is no rule concerning the relative sizes of records and blocks; however, efficiency may be improved if one is a multiple of the other. The Record Block specifies record number (starting at Ø), buffer address and length, and transfer direction (read or write). .RECRD requests require the use of the .INIT, .RLSE, .OPEN, .CLOSE, and .WAIT (or .WAITR) requests. 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.

Rules: The requested device must be file-structured and the file must be contiguous.

The user must set up a Record Block in his program and must provide a buffer.

All records must have the same length.

The user should perform a .WAIT or .WAITR to ensure that processing has completed.

The associated file must have been opened with .OPENU or .OPENI.

Errors: An error causes a return to the user with the type of error indicated in the FUNCTION/STATUS word of the RECORD BLOCK. The user should perform the following test after his request to ensure that the request completed normally.

```
TSTB  RECBLK+1
BNE   ERROR
```

3.6.8 .BLOCK - Read or write a specific block in 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 3-13).

Assembly Language Expansion:

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

Global Name: BLO

Description: BLOCK requests provide for random access to the blocks of files stored on 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 a Monitor buffer. The user may process the data in the Monitor buffer or he may transfer the block to and from his own area. BLOCK requests require the use or the .INIT, .OPEN, .CLOSE and .WAIT (or .WAITR) requests.

The user must specify one of three functions in the BLOCK block: INPUT, GET, or OUTPUT (see Figure 3-13). 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 3-11) the address of the buffer and the physical length of the block transferred.

GET: During a GET request, the Monitor returns in the BLOCK Block the address and length of a buffer within the Monitor that he can fill for subsequent output. Only one GET is required for each time the file is OPENed and CLOSEd (i.e., once a buffer has been located, it may be used repeatedly). The user must assure that he does not over-run the buffer. This request is unnecessary if an INPUT request has occurred.

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

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

* Access to linked files or nondirectory devices is illegal.

The user must set up the BLOCK block in his program according to the format of Figure 3-13.

Errors: Error processing causes a normal return to the user, 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 assure that his request was error free.

3.6.9 .TRAN - Read or write the specified block (file-structured device) or the next block (non-file-structured device).

Macro Call: .TRAN #LNKBLK,#TRNBLK
where LNKBLK is the address of the Link Block, and TRNBLK is the address of the TRAN block (see Figure 3-14).

Assembly Language Expansion:

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

Global Name: TRA

Description: .TRAN provides nearly direct access to the device on which the dataset resides. No file processing is done and any file structure is ignored. Therefore, writing with .TRAN on a file-structured device is especially risky and many lead to the corruption of all data on the device. If .BLOCK request can be used instead of .TRAN, it is recommended. Each .TRAN will transfer one or more blocks, depending upon the word count in the TRAN Block. Blocks on file-structured devices are referenced by absolute block number, while blocks on non-file-structured devices are processed in sequence. .INIT, .RLSE and .WAIT (or .WAITR) must be used. .OPEN and .CLOSE must not. 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.

Rules: .TRAN must be preceded by an .INIT request on the associated dataset. .OPEN must not be used. For each .TRAN request, the user must provide a transfer control block, as shown in Figure 3-12. 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.

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

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

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

```
            .INIT #TAPE1
              .
              .
              .
            .TRAN #TAPE1, #BIN40
              .
              .
              .
            .RLSE #TAPE1
              .
              .
              .
            .WORD ERR1
TAPE1:      .WORD 0
            .RAD50 /TP1/
            .BYTE 1,3
            .RAD50 /DT/
              .
              .
BIN40:      .WORD 100         ;STARTING BLOCK #
            .WORD BUFFER      ;STARTING ADDRESS IN CORE
            .WORD 200         ;NUMBER OF WORDS
            .WORD 4           ;INPUT
            .WORD 0           ;FOR MONITOR USE
              .
              .
              .
ERR1:                         ;ERROR ROUTINE FOR DECTAPE
              .
              .
              .
BUFFER:     .WORD 0
BUFEND:     .BLKW 200
              .
              .
              .
            .END
```

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

Macro Call: .WAIT #LNKBLK

where LNKBLK is the address of the Link Block (see Figure 3-6).

Assembly Language Expansion:

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

Global Name: (Routine is embedded in the resident Monitor.)

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 on the teleprinter.

.WAITR

3.6.11 .WAITR - Check for completion of processing on dataset and return or transfer.

Macro Call: .WAITR #LNKBLK,#ADDR

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

Assembly Language Expansion:

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

Global Name: (Routine is imbedded in the resident Monitor.)

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

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

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

3.6.12 .SPEC - Special functions.

Macro Call: .SPEC #LNKBLK,#SPCARG

where LNKBLK is the address of the Link Block, and SPCARG may be either a special function code or the address of a special function block containing the code (see Figure 3-15), depending upon the function.

Assembly Language Expansion:

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

Global Name: SPC

Description: This request is used to specify a special function (action) to a device, such as rewind magnetic tape. A code identifies the function and must be in the range $0\text{-}255_{10}$. When the function requires no supporting data, the code itself is the first parameter to be placed upon the processor stack in the assembly language call sequence. However, if the user must supply additional information or if the function expects to return data to the user, the code is passed within a special function block and the address of the block is the call parameter. The format of this block is shown in Figure 3-15.

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

Rules: The dataset must be INITed.

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

.STAT

3.6.13 .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 (Packed Radix-50)
SP+4	Device Standard Buffer Size (in words)

where Driver Facilities Word has the following format;

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

spare spare

1 = device is directory structured
1 = device is DECtape
1 = device is sequential magnetic tape
1 = system disk driver
1 = device has multiple units under one controller
1 = device is a terminal
1 = driver has an OPEN entry

1 = device will support multi-dataset activity
1 = device will handle output
1 = device will handle input
1 = device will handle binary data
1 = device will handle ASCII data
1 = driver has a special function entry
1 = driver has a CLOSE entry

Device Name is the Radix-50 packed ASCII standard mnemonic for the device (Appendix A); and, Device Standard Buffer Size is the block size (in words) 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.

3.7 REQUESTS FOR DIRECTORY MANAGEMENT SERVICES

3.7.1 .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) or MOV #N+1ØØØØØ,-(SP)
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 15
```

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

Description: Searches the device for a free area equal to N 64-word segments, and creates a contiguous file in the area if it is found, by making an appropriate entry in the User File Directory (UFD). If the sign bit (bit 15) of N is set, the UFD pointer will point to the beginning of the allocated area thereby indicating that the file is empty. This enables partial filling of the file space and later extension of the file. If the sign bit of N is not set, the UFD pointer will point to the end of the allocated area and thereby indicate that the file area is full and may not later be extended. (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 contain N segments, i.e.,

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

where B is the number of 64-word segments per block. For example, if N=9 and the device specified is DECtape, then $B=\frac{256}{64} = 4$. Therefore, $\frac{N}{B} = \frac{9}{4} = 3$, and 3 blocks will be allocated.

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

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

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

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	A002
Dataset Not INITed	--	F000
File Exists	2	F024
Directory Full	12	F024
UIC Not In Directory	13	F024
Illegal Filename	15	F024

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

Example: Create a contiguous file of four 256_{10} word blocks on DECtape unit 4. Name the file FREQ.DAT.

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

3.7.2 .DELET - Delete a file.

Macro Call: .DELET #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 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 teleprinter for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	A002
Dataset Not INITed	--	F000
Nonexistent File	2	F024
Protect Code Violation	6	F024
File Is Open	14	F024

.RENAM

3.7.3 .RENAM - Rename a file. Change protection code.

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 (see Section 3.8.6.3) 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 filenames must be different. To change just the protection for a file, two .RENAMs must be requested.

The new filename must not already exist, and the new filename must be legal. The old file must exist.

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

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

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Dataset Not INITed	--	F000
File Exists (new name)	2	F024
File Nonexistent (old file)	2	F024
Protection Violation	6	F024
File Is Open	14	F024
Illegal Filename	15	F024

3.7.4 .APPND - Append one linked file to another.

Macro Call: .APPND #LNKBLK,#FIRST,#SECOND

where LNKBLK is the address of the Link Block, FIRST is the address of the Filename Block for the first file (file to be appended to), and SECOND is the address of the Filename Block for the second file (file to be appended).

Assembly Language Expansion:

```
MOV #SECOND,-(SP)
MOV #FIRST,-(SP)
MOV #LNKBLK,-(SP)
EMT 22
```

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 teleprinter for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message On Default
Device Not Ready	--	A002
Dataset Not INITed	--	F000
First File Nonexistent	2	F024
Contiguous File	5	F024
Protect Code Violated	6	F024
File Opened	14	F024

NOTE

Since the last block of a file is typically not full, there will be a gap (null characters) in the new file at the junction point. This causes no problem in ASCII files but might cause confusion in binary files.

3.7.5 .LOOK - Search the device directory for a specified filename.

Macro Call: .LOOK #LNKBLK,#FILBLK[,1]

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

Assembly Language Expansion:

a. If the optional argument is not specified:

```
MOV #FILBLK,-(SP)
MOV # LNKBLK,-(SP)
EMT 14
```

b. If the optional argument is specified:

```
MOV #FILBLK,-(SP)
CLR -(SP)
MOV #LNKBLK,-(SP)
EMT 14
```

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

Description: The primary purpose of this routine is to search through a specified directory for a specified file and return with the current parameters of the file. However, this routine can also be used to indicate (bits 0-3) the permissible functions for a nondirectory device (i.e., input, output, update, etc.). By specifying the optional argument, the user indicates whether he requires two or three parameters be returned.

The device to be searched is specified in the Link Block, and the file is specified in the Filename Block. The request returns to the user with the top elements of the stack as follows

	2 Arg. Call	3 Arg. Call
START BLOCK		SP
# OF BLOCKS	SP	SP+2
INDICATOR WORD	SP+2	SP+4

where # OF BLOCKS is the number of blocks in the file, and the INDICATOR WORD is coded as follows:

Bit 0=1	.OPENC allowed
Bit 1=1	.OPENI allowed
Bit 2=1	.OPENE allowed
Bit 3=1	.OPENU allowed
Bit 4=0	File is not in use
4=1	File is being used by another dataset
Bit 5=1	Dataset already has a file open (no search has been performed)
Bit 6=0	File is linked
6=1	File is contiguous
Bit 7=0	File nonexistent (OPENO allowed)
7=1	File exists or .OPENO not allowed
Bits 8-15	Protection Code

After the request has been completed, control is returned to the user at the instruction following the assembly expansion. The stack must be cleared by the user. If a file is protected against READ access, it will be signaled as nonexistent.

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 teleprinter for an error message if it is not. Possible errors are shown below:

Error Condition	Error Code Returned To Filename Block	Error Message
Device Not Ready	--	A002
A File Is Open On Requesting Dataset	14	F024
Illegal Filename	15	F024

Note that it is possible to .LOOK for a file and be told that it does not exist. A subsequent attempt to open the nonexistent file may lead to an OPEN error (code=2). Hence, it may be more efficient to simply attempt the .OPEN and check for an error (see Section 3.6.3).

3.7.6 .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 and LNKBLK is the address of the Link Block.

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 Keyboard command (see Chapter 2). It does this by setting bit 7 of the PROTECT byte in the Filename Block. Automatic deletion upon FInish is not currently implemented.

3.8 REQUESTS FOR MISCELLANEOUS SERVICES

3.8.1 Load a Program or an Overlay

3.8.1.1 .RUN

Macro Call: .RUN #RUNBLK

where RUNBLK is the address of the user's Run Block (see Figure 3-16).

Assembly Language Expansion:

```
MOV #RUNBLK,-(SP)     ;PUSH ADDRESS OF THE RUN BLOCK
EMT 65                ;ONTO THE STACK
```

Global Name: RUN

Description: The RUN request may be used to load an entire program or a program overlay. It has several options, among which are:

- load a program or load an overlay - when an overlay is loaded, the existing program environment is not disturbed; one section of the program is simply replaced by another. When a new program is loaded, the old program and its effects (except for data on the stack) are purged from core, and the new program takes over; for example, FORTRAN can use the RUN request to load LINK and LINK can use it to load and execute the user's program;
- load a core image or a load module;
- return of control:

  instruction following .RUN;

  transfer address of load module or core image;

  transfer address plus offset (word F);

  alternate return address (word G);
- stack movement:

  leave as is;

  move the stack down if it would otherwise be destroyed by the entity being loaded;
- load address:

  as specified in file,

  as specified by user.

The RUN request requires the following control blocks:

Run Block: A variable length control block whose address is passed on the stack. It contains a function word and various optional parameters. It is described in Section 3.9.8.

Link Block: The standard Link Block (section 3.9.1). It describes the device from which the entity is to be loaded. It is required unless bit 15 of the function word in the Run Block is 1.

File Block: The standard File Block (section 3.9.2). It describes the file from which the entity is to be loaded: either an .LDA file or a CIL. It is required unless bit 15 of the function word in the Run Block is 1.

The Link Block should not be .INITed, nor should the File Block be .OPENed, when .RUN is called. RUN will perform .OPEN, .CLOSE, .INIT and .RLSE processing. The lookup sequence is as follows:

First an extension of LDA is attempted, then no extension, unless an extension is specified, in which case it alone is used;

For each extension, the current UIC, then [1,1] is tried, unless a UIC is specified, in which case it alone is used;

The .RUN request always removes the Run Block address from the stack. If bit Ø is Ø, the following information will be returned upon the stack:

(SP) - transfer address of loaded module,
2(SP) - size of loaded module in bytes,
4(SP) - low address of loaded module.

Aside from this, the stack is not disturbed, although it may be moved. This means that the stack may be used for passing arguments.

Rules: The Link Block should not be .INITed.

The File Block should not be .OPENed.

If an overlay is being loaded, it must not extend above the bottom of the resident program section, nor below the top of the Monitor.

If a new program is to be loaded, all datasets used by the current program must be RLSEd.

The user must be sure that his stack is not inadvertently destroyed.

When options are requested through the function word, the appropriate supporting data must be present in the Run Block.

If the stack might be moved, it must not contain absolute pointers to locations within the stack. For example:

```
MOV   SP,RØ
MOV   RØ,-(SP)
```

produces a stack which should not be moved. The user can assure that such a stack will not be moved by setting bit 1 of the Function word in the RUN Block to Ø (see Section 3.9.8).

Errors: Errors F007, F012, F021, F022, F024, F045, F054, F274, F276, and F277 are all possible. All but F007 and F021 are nonfatal, provided that an error return is provided in the File Block (see Table 3-4).

3.8.2 Request to Return Control to the Monitor

3.8.2.1 .EXIT - Exit from a user program to Monitor.

Macro Call: .EXIT

Assembly Language Expansion:

EMT 60

Global Name: XIT

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

3.8.3 Requests to Set Monitor Parameters

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

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

Macro Call: .TRAP #STATUS,#ADDR

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

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #STATUS,-(SP)
MOV #1,-(SP)         ;1 is the identifier code for .TRAP
EMT 41
```

Global Name: GUT

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

Rules: STATUS must be a valid Status Byte.
ADDR must specify an address within the user's core area.

Errors: If an invalid code is specified, a fatal (FØØ2) error will result.

3.8.3.2 .RSTRT - Set the default address for use by the REstart keyboard command.

Macro Call: .RSTRT #ADDR

where ADDR is the restart address.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #2,-(SP)          ;2 is the identifier code for .RSTRT
EMT 41
```

Global Name: GUT

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

Rules: ADDR must be an address within the user's core area.

3.8.4 Requests to Obtain Monitor Parameters

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

Macro Call: .CORE

Assembly Language Expansion:

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

Global Name: GUT

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

Errors: No errors are possible.

3.8.4.2 .MONR - Obtain the address of the first word not within the resident 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 3-5) and returns it to the user at the top of the stack. This value does not reflect any area allocated by the Monitor for control blocks, device drivers, data buffers, etc. (see .MONF, Section 3.8.4.3). 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.

Errors: No errors are possible.

.MONF

3.8.4.3 .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 3-5), 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.

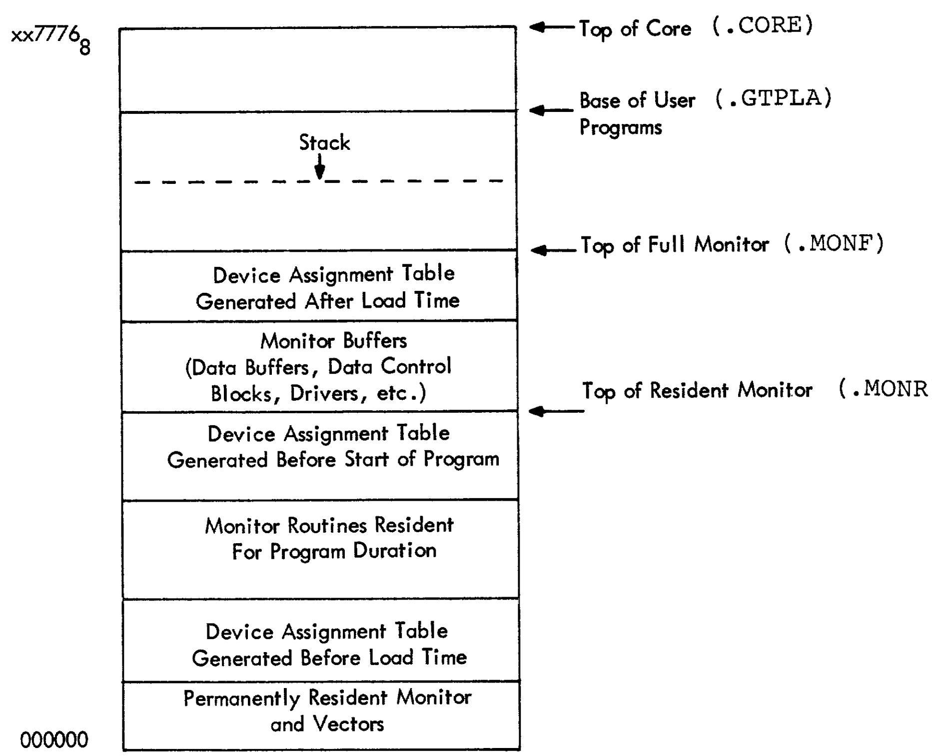


Figure 3-5 Core Map of Resident Monitor and Full Monitor.

3.8.4.4 .DATE - Obtain current date.

Macro Call: .DATE

Assembly Language Expansion:

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

Global Name: GUT

Description: The current date word is returned to the user at the top of the stack. The user must clear the stack. The date format is a binary number equal to Julian-$70,000_{10}$. If the user requires the ASCII representation of the date, he should use the .CVTDT request (see 3.8.4.6).

Errors: No errors are possible.

.TIME

3.8.4.5 .TIME - Obtain current time of day.

Macro Call: .TIME

Assembly Language Expansion:

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

Global Name: GUT

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

SP:	LOW-ORDER TIME IN TICKS
SP+2:	HIGH-ORDER TIME

where a TICK 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. See the CVTDT request for how to obtain the ASCII representation of current time value.

Errors: No errors are possible.

3.8.4.6 .CVTDT - Convert binary representation of date or time to ASCII character string.

Macro Call: .CVTDT #CODE, #ADDR[,VALUE]

where CODE identifies the conversion to be done;

```
CODE = Ø   Current date as stored by monitor,
CODE = 1   Current time as stored by monitor,
CODE = 2   Date supplied as VALUE,
CODE = 3   Time supplied as VALUE (and VALUE+2)
```

ADDR is the address of the first byte of the user buffer into which the ASCII string is to be stored, and VALUE is the address of user supplied Date or Time (used with CODEs 2 and 3 only).

Assembly Language Expansion:

```
MOV VALUE+2,-(SP): Code 3 only
MOV VALUE,-(SP); Codes 2 and 3 only

MOV #ADDR,-(SP)
MOV #CODE,-(SP)
EMT 66
```

Global Name: CDT

Description: This request converts either a date or a time from internal (binary) representation into an ASCII string suitable for display. The user may specify that the current system value (of date or time) is to be used for conversion or he may supply his own value. The string returned has the format of the Date and Time returned by the Keyboard DATE and TIME commands (see Chapter 2). Upon return, the call arguments have been removed from the stack and condition codes N, Z and V are cleared to Ø.

Rules:

1. The buffer area supplied by the user program (starting at ADDR) must provide sufficient room for the text returned as no check is made. Nine bytes are required for Date, eight bytes are required for Time.
2. User-supplied VALUEs for Date or Time must comply with the internal storage format of those values, that is:
   a. Date; 1 word containing (year-197Ø)*1ØØØ + day of the year (Julian).
   b. Time; 2 unsigned integer words for high-order and low-order time in clock ticks.

Errors: 1. Specification of an illegal CODE (i.e., > 3) causes fatal error message:

F∅34 Call address

2. If the currently stored Date or Time is out of range (i.e., Date > 366 (Modulo 1∅∅∅) or Time > 47:59:59), an operator action message

A∅11 CODE(∅ = Date, 1 = Time)

is printed. The operator should enter the desired value via the appropriate DAte or TIme keyboard command and type COntinue to proceed. If 23:59:59 < Time < 48:00:00, Date is incremented and Time is reduced by 24:00:00.

3. If a user supplied Date or Time is out of range as above, the conversion routine will return without attempting conversion and the condition code V will be set to 1. Thus the program should follow the .CVTDT request with the check:

BVS (error routine).

3.8.4.7 .GTUIC - Get the current user's UIC.

Macro Call: .GTUIC

Assembly Language Expansion:

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

Global Name: GUT

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

Errors: No errors are possible.

3.8.4.8 .SYSDV - Get name of the system device.

Macro Call: .SYSDV

Assembly Language Expansion:

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

Global Name: GUT

Description: The name of the system device in Radix-50 notation is returned to the user at the top of the stack.

Errors: No errors are possible.

3.8.4.9 .GTPLA - Return the current program low address.

Macro Call: .GTPLA

Assembly Language Expansion:

```
CLR -(SP)
MOV #5,-(SP)
EMT 41
```

Global Name: GUT

Description: The program low address is the address of the first (lowest) word of the current program. In the case of a program with overlays, the PLA is the address of the first word of the resident section. PLA is established when the keyboard RUN command is executed or when the .RUN request is used to load a new program (not an overlay, e.g., when MACRO calls CREF, which then replaces MACRO). Because the .RUN processor will not load an overlay which extends above this address, the PLA is also called the Protection Boundary.

.GTPLA allows the user to retrieve this value (see Figure 3-5), which is returned to the top of the stack. .STPLA allows the user to set it.

Rules: The user must clear the stack.

Errors: No errors are possible.

3.8.4.10 .STPLA - Set the program low address.

Macro Call: .STPLA #ADDR

where ADDR is the desired new program low address.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #5,-(SP)
EMT 41
```

Global Name: GUT

Description: This request allows the user to establish a new program low address. This is done if the user wants part of his resident code overlayed or if he wants to reserve additional space between his resident code and his overlays. Consult the .GTPLA description for more details.

The old program low address (or a zero) will be returned on top of the stack upon return from this macro call.

Rules: The user is required to clear the returned address from the stack.

Errors: The address returned on top of the stack will be zero when the call is unsuccessful. This occurs when the address is outside of available memory.

3.8.4.11 .GTCIL - Return the address of the first block of the Monitor core image library (CIL).

Macro Call: .GTCIL

Assembly Language Expansion:

```
MOV #111,-(SP)
EMT 41
```

Global Name: GUT

Description: This request returns the address of the first block of the Monitor core image library to the top of the stack.

Rules: The user is required to clear the disk address returned on the stack.

Errors: No errors are possible.

3.8.4.12 .GTSTK - Return the current stack base entry.

Macro Call: .GTSTK

Assembly Language Expansion:

```
CLR -(SP)
MOV #4,-(SP)
EMT 41
```

Global Name: GUT

Description: The stack base is the highest core address used for stack storage plus two. A RUN Keyboard command clears the stack and sets the stack base address to the program low address. A user .RUN request does not clear the stack (to allow inter-program communication via the stack) but the stack may be relocated. This request may be used to determine the stack base. Following the request the current stack base entry is returned on top of the stack.

Rules: The user is required to clear the returned value from the stack.

Errors: No errors are possible.

3.8.4.13 .STSTK - Modify the stack base entry.

Macro Call: .STSTK #ADDR

where ADDR is the desired new stack base address entry.

Assembly Language Expansion:

```
MOV #ADDR,-(SP)
MOV #4,-(SP)
EMT 41
```

Global Name: GUT

Description: This request is used when the stack is to be relocated. It does not relocate the stack, but it does record its new base (the address of the word immediately above the stack; see section 3.8.4.12), and it returns the old stack base on the stack. EXTREME CAUTION should be used when moving the stack; it is not recommended as a standard procedure. Note that the .RUN request may be used to move the stack when that is appropriate.

Rules: The user must clear the old base value from the stack when control is returned.

The user is responsible for moving the stack.

Caution should be used when moving the stack, since the new and old stack areas may overlap and since Monitor interrupt routines may use the stack while it is being moved. Let:

SB1 = old stack base (returned on stack)
SB2 = new stack base (supplied by user)
SP1 = old stack pointer (current value of SP)
SP2 = new stack pointer (SB2 - SB1 + SP1)

First, set SP=min (SP1,SP2) to protect against interrupts. Then if SB1< SB2, move the stack starting from the base (SB1 to SB2), If SB1>SB2, move the stack starting from the top (SP1 to SP2). This strategy prevents the stack from being corrupted during the move (since the two stack areas might overlap). Finally, set SP to SP2.

Errors: If the new stack base ADDR is outside available memory or inside the Monitor, the request is not honored and a zero is returned on the stack.

3.8.4.14 .STFPU - Initialize the floating-point exception vector.

Macro Call: .STFPU #PSW,#ADDR

Assembly Language Expansion:

```
MOV #ADDR,-(SP)     ;ADDRESS OF EXCEPTION ROUTINE
MOV #PSW,-(SP)      ;PROGRAM STATUS WORD FOR
                    ;EXCEPTION RTN
MOV #3,-(SP)        ;REQUEST CODE
EMT 41
```

Global Name: GUT

Description: This request initializes the exception interrupt vector for the floating-point processor on the PDP-11/4Ø or PDP-11/45. Any floating-point exception for which interrupt is enabled will cause a trap to location ADDR with a new program status word of PSW. The interrupt vector is at location 244_8.

Rules: None.

Errors: None.

3.8.5 Requests to Perform Conversions

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

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

Macro Call: .RADPK #ADDR

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

Assembly Language Expansion:

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

Global Name: CVT

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

Radix-50 is used by the Monitor to store in one word three characters for half a filename or an extension or other three-character sets of data.

Because the characters allowed within names (e.g., filenames or extensions, assembler symbols, etc.) are restricted to letters, digits, and a few special characters, it is possible to store three characters within a single word by using the formula:

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

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

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

The maximum value for three characters is thus:

$$47 \times 50^2 + 47 \times 50 + 47 = 174777$$

The Radix-50 representation for various peripheral devices is shown below:

Mnemonic	Device	Radix-50 Equivalence
CR	Card Reader (CR11)	012620
DC	RC11 Disk	014570
DF	RF11 Disk	014760
DK(A,B)	RK11 Disk	015270(+1,2)
DT(A)	DECtape (TC11)	016040(+1)
KB	ASR-33 Keyboard/Printer	042420
LP	Line Printer (LP11)	046600
MT	Magtape (TM11)	052140
PP	High-Speed Paper Tape Punch	063200
PR	High-Speed Paper Tape Reader	063320
PT	ASR-33 Paper Tape Device	063440

NOTES:

a. Device mnemonics may be three letters on some systems. The third letter is assigned if there is more than one controller. For example:

   DTA for DECtape controller A
   DTB for DECtape controller B

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

   DT1 for unit 1 under a single DECtape control
   DTA1 for unit 1 under controller A in a multi-controller situation.

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

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

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

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

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

```
         MOV #PAKBUF,R3       ;SET UP POINTER TO PACK BUFFER
         MOV #UNPBUF,-(SP)    ;.RADPK UNBUF
NEXT:    CLR -(SP)
         EMT 42
         BCS ERRC             ;INVALID 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 on the stack the address of the byte which follows the last character converted.

.RADUP

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

Macro Call: .RADUP #ADDR,WORD

where ADDR is the address of the first of three bytes into which the unpacked characters are to be placed, and WORD is the Radix-50 word to be converted.

Assembly Language Expansion:

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

Global Name: CVT

Description: WORD is converted into a string of 7-bit ASCII charactors which are placed left-justified with trailing spaces in three consecutive bytes starting at location ADDR. The stack is cleared. See section 3.8.5.1 for a definition of Radix-50.

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

C-bit set means:

a. a value of WORD was outside the valid Radix-50 set, i.e., >174777 (see Section 3.8.5.1).

b. a Radix-50 byte value was found to be 35, which is currently not used.

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

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

3.8.5.3 .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.

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

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

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

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

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

Macro Call: .BIN2D #ADDR,WORD

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

Assembly Language Expansion:

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

Global Name: CVT

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

Errors: No errors are possible.

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

Macro Call: .O2BIN #ADDR

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

Assembly Language Expansion:

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

Global Name: CVT

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

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

C-bit set means that a byte was not an octal digit.
V-bit set means that the octal number was too large,
i.e., the first byte was greater than 1.

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

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

Macro Call: .BIN2O #ADDR,WORD

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

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 7-bit octal ASCII characters, right-justified with leading zeros, which is placed into the buffer addressed by ADDR. The stack is cleared.

Errors: No errors are possible.

3.8.6 Requests for Interfacing with the Command String Interpreter

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

The CSI is called in two parts, by two different requests:

- .CSI1 condenses the command string and checks for syntactical errors.
- .CSI2 sets the appropriate Link Block and Filename Block parameters for each dataset specification in the command string.

Each command string requires one .CSI1 request for the entire command string, and one CSI2 request for each dataset specifier in the command string.

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

3.8.6.1 .CSI1 - Condense command string and check syntax.

Macro Call: .CSI1 #CMDBUF

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

Assembly Language Expansion:

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

Global Name: CSX

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

Rules: The .CSI2 request must be preceded by a .CSI1 request, because .CSI2 assumes it will get 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. Command Strings must not be read in dump mode.

It is the user's responsibility to print a # on the teleprinter to inform the operator that a CSI format is expected (Section 2.1). If VERTICAL TAB is used as the terminator, the # will be typed immediately without a carriage return or line feed.

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 which process the .CSI1 and .CSI2 requests.

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

Example: See .CSI2, Section 3.8.6.2.

3.8.6.2 .CSI2 - Interpret one dataset specification of a command string.

Macro Call: .CSI2 #CSIBLK

where CSIBLK is the CSI control block, described under "Rules" below.

Assembly Language Expansion:

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

Global Name: CSM

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

Rules: Before calling .CSI2, the user must:

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

CSIBLK:

POINTER TO CMDBUF
POINTER TO LNKBLK
POINTER TO FILBLK

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

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

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

- Set the first word (Code 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 two items of information. Bits 1-0 have the following meaning:

  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

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

  If there are no more dataset specifications and the user requests one anyway, a null specification will be returned.

  Bit 2, when set to one, indicates that the device name in the Link Block is a default supplied by the system (see Section 3.4.1).

- With respect to values returned in the Link Block (Figure 3-6):

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

- Immediately following the PHYSICAL DEVICE NAME word in the Link Block are the switches specified in the command string. The interface for each switch is shown in the switch block below. These switch blocks are written in the area provided by the programmer in the Link Block. Note that the number of words to follow in the switch block is not the same quantity as is specified in the LINK Block.

NUMBER OF WORDS TO FOLLOW		
POINTER TO FIRST CHARACTER OF Vn		
POINTER TO FIRST CHARACTER OF Vn-1		
. . .		
POINTER TO FIRST CHARACTER OF V1		
W(ASCII)	S(ASCII)	;for /SW

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

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

then the completed interface appears as:

3	
1014	
1010	
127=W	123=S

- With respect to the values returned in the Filename Block (Figure 3-7):

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

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

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

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

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

3.9 USER PROGRAM TABLES AND CONTROL BLOCKS

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

LNKBLK:

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

Figure 3-6 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 to transfer control in the event that any request associated with this dataset fails to obtain required buffer space from the the Monitor. If no address is specified here, such an error will be treated as fatal. This address may be changed by the user's program at any time.
LNKBLK	LINK POINTER	This location must be set to zero by the user and must not be modified by him. The Monitor places a linking address here when the dataset is INITed. Before INITing a dataset, the Monitor tests this pointer for zero. If it is not zero, the Monitor assumes that the dataset was already INITed.
LNKBLK+2	LOGICAL NAME OF DATASET	The user can specify a name for the dataset in this entry. This name, which must be unique, is used to associate the dataset with a device which is specified by an ASSIGN from the keyboard. The name is stored in Radix-50 packed ASCII by the .RAD50 assembler directive. This specification is optional, but if it is omitted, the ASSIGN command cannot be used.
LNKBLK+4	NUMBER OF WORDS TO FOLLOW	This byte contains the count of the number of words to follow in the Link Block. The user should set it to a 0 if he does not specify any PHYSICAL DEVICE NAME in the

Address	Name	Function
		next word, or to a 1 if he does. Values greater than 1 may be used if the Command String Interpreter is to be called.
LNKBLK+5	UNIT NUMBER	This code specifies the unit number of the device linked to the dataset. For example, the TC11 Controller (DECtape) can drive up to eight tape drives (units), numbered 0-7.
LNKBLK+6	PHYSICAL DEVICE NAME	If the user specified 1 or greater in byte LNKBLK+4, he may specify here the standard name (Appendix A) for the device associated with the dataset in Radix-50 format. If no name is specified here, the user must specify LOGICAL NAME OF DATASET and perform an ASsign command before he runs his program. If physical device name is specified both here and in an ASSIGN command, the device specified in the ASSIGN command overrides the value given here.
LNKBLK+8 through LNKBLK+n	OPTIONAL DATA	Present only if LNKBLK+4 is greater than 1. It is used to pass additional information such as switch information when using the Command String Interpreter or Resident EMT information when using .RUN, via the Link Block.

FILENAME Block

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

FILBLK:

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

Figure 3-7 The Filename Block

Address	Name	Function
FILBLK-4	ERROR RETURN ADDRESS	The user must specify here the address to which he wants the Monitor to return control if one of the errors in Table 3-4 occurs during an operation involving the file. If no address is specified here, any such error will be treated as a fatal error.

3.9.2.1 Error Condition Codes (FILBLK-1)

Table 3-4

Filename Block Error Conditions

Error Code In Filename Block	Faulting Request	Cause	Remedy
00	.OPENC .OPENE .OPENI .OPENO .OPENU	An attempt was made to open a dataset that was previously opened.	
01		unused	

(continued on next page)

Table 3-4 (Cont)

Filename Block Error Conditions

Error Code In File-name Block	Faulting Request	Cause	Remedy
02	.OPENO	An attempt was made to open a file which already exists.	If name of file was correct, delete the file (with PIP) or change file name.
	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file for input, extension, or update which is currently opened for output, or which does not exist.	
	.RUN	The file specified was already OPENed for output, or the file does not exist.	
03	.OPENC .OPENE .OPENI .OPENU	An attempt was made to open a file which has already been opened the maximum number of times (76_8).	Close file.
04	.OPENC .OPENE .OPENU	An .OPENC, .OPENE, or .OPENU attempt was made to open a file which has already been opened for either .OPENC, .OPENE, or .OPENU.	.CLOSE the previous open.
05	.OPENE	Illegal request to a contiguous file.	
06	.OPENC .OPENE .OPENI .OPENO .OPENU .RUN	An attempt was made to access a file which the protection code prohibits.	Resolve access problem with owner of the file.
07	.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.

(Continued on next page)

Table 3-4 (Cont)
Filename Block Error Conditions

Error Code In File-name Block	Faulting Request	Cause	Remedy
13	.ALLOC	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.
16	.RUN	All datasets were not released prior to issuing the request.	Release all datasets which were INITed.
17	.RUN	Load module format error.	File must be linked into a load module.
20	.RUN	Specified CIL entry not found.	Add proper entry to CIL or use correct name.
21	.RUN	No transfer address or illegal transfer address.	Check for END statement in source program, or use correct /TR when linking.
22	.RUN	Stack base entry in the System Vector Table (SVT) is below the Stack Pointer. Stack cannot be moved as requested in the call.	Probably a program error.
23	.RUN	Module is outside the boundaries of the allowable load area.	Relink to within boundaries. Ensure that resident portion of program is not being overlayed.

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

3.9.2.2 The File Protection Codes

7	6	5	4	3	2	1	0
Owner		User Group			All Others		

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

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

Figure 3-8 File Protection Codes

User Group and All Others

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

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

Figure 3-8 File Protection Codes

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

BUFHDR:

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

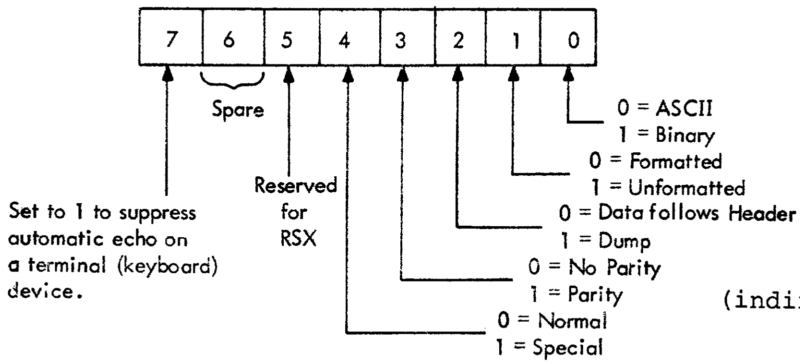
Figure 3-9 Line Buffer Header

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

Address	Name	Function
BUFHDR	MAXIMUM BYTE COUNT	The count shows the size of the buffer, in bytes. It must be specified here by the user on all INPUT operations.
BUFHDR+2	MODE	The user specifies here the mode of the transfer. All modes are listed and explained in Figure 3-10.
BUFHDR+3	STATUS	The Monitor will place in this byte the status of the transfer when control is returned to the user. Figure 3-11 lists each bit and its meaning. Errors encountered executing an I/O transfer will be flagged in this byte. The user should always check its content after each transfer completes.
BUFHDR+4	ACTUAL BYTE COUNT	This count controls the number of bytes to be transferred on OUTPUT. It must be initialized by the user before any output transfer from the line buffer. After any transfer in or out, it will show how many bytes have been transmitted (or in some modes, see Section 3.6, 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. (Section 3.9.6 Note 9.)

NOTE

The Monitor will return control to the program if a device transfer is needed to satisfy a READ or WRITE request. During this time, the header words will be used to store data relevant to the operation underway. The user should not, therefore, attempt to change this content until it is evident that the transfer has been completely effected, e.g., after a .WAIT return.



(indirect)

Figure 3-10 The Mode Byte

3.9.3.1 The Transfer Modes - The user can specify ASCII or binary data in nine different modes of transfer:

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

Formatted ASCII Nonparity - Special
Formatted ASCII Nonparity - Normal

Unformatted ASCII Parity - Special
Unformatted ASCII Nonparity - Normal

Binary Modes: Formatted Binary - Special
Formatted Binary - Normal

Unformatted Binary - Normal

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

READ: The line buffer is filled until either a terminator is seen or the number of bytes transferred becomes equal to the MAXIMUM BYTE COUNT. If the MAXIMUM BYTE COUNT is reached before the terminator is seen, the invalid line error bit in the Status Register of the buffer header is set, and each remaining character through to the terminator is read into the last byte of the line buffer, i.e., the surplus bytes are overlayed. After the transfer, the actual byte count is set to the number of bytes read (including the excess). RUBOUTs and NULLs are discarded. The terminator is transferred. LINE FEED is supplied after RETURN.

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, the invalid line error bit is set in the STATUS BYTE of the buffer header. Previous terminators are output as normal characters.

For non file-structured devices, TABs are automatically followed by RUBOUTs; FORM FEEDs are automatically followed by NULLs.

The READ/WRITE processor passes data to the device driver specified, and each driver will convert the information to meet its specific needs. Appendix G summarizes the characteristics of the device drivers. Normally, output is deferred until the current buffer is full or until a .CLOSE or .RLSE occurs. However, for terminal devices, the buffer is written when a line terminator is seen. VERTICAL TAB plays a special role here, since it is a terminator but does not cause a ~~carriage return or paper motion.~~

2. Formatted ASCII Special -

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

WRITE: The same as formatted ASCII normal with this exception: the line buffer is output until the first terminator; the ACTUAL BYTE COUNT will stop the transfer if it is reached before the terminator is seen. In this case, the invalid line error bit is set in 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 exactly specified, provided it is not greater than the ACTUAL BYTE COUNT.

3. Formatted Binary Normal -

READ: This is an 8-bit transfer. Words 2 and 3, 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 data bytes read (including the excess).

WRITE: This is an 8-bit transfer. Words 2 and 3 of the line buffer header are output, and data is transferred until the number of characters transferred is equal to the ACTUAL BYTE COUNT; then a checksum is calculated. The checksum is output at the end. The byte count is adjusted to reflect the presence of words 2 and 3 from the line buffer header.

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 BYTE COUNT is transferred to the line buffer and the ACTUAL BYTE COUNT is set to the full input count, rather than to the number of bytes actually transferred. The invalid line error will be set in the STATUS BYTE. The user can compare the MAXIMUM 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 and nulls are not discarded. No checksum is calculated.

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

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. Indirect Modes - All modes can be specified as indirect, which means that the word after the ACTUAL BYTE COUNT is considered to be a pointer to the beginning of the data rather than the beginning of the data proper. (Section 3.9.4.) This is referred to as DUMP mode.

3.9.3.2 The Status Byte

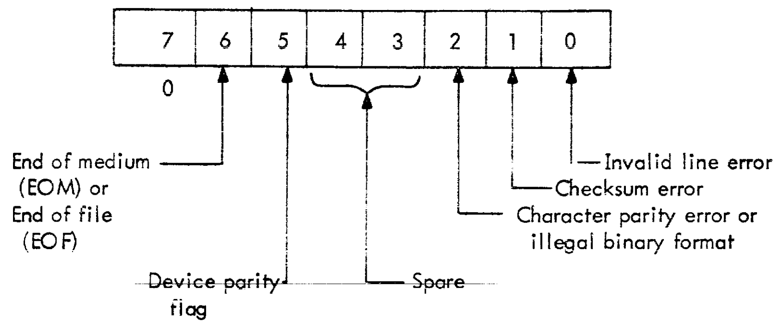


Figure 3-11 Status Byte Format

The function of each status format bit is explained below.

Bit	Mode	Request	Condition
	ALL	.READ/WRITE	Appropriate BYTE COUNT = 0 at call.
0 (INVALID LINE)	FORMATTED ASCII NORMAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT ran out before a line terminator was seen. (Last byte has been overlaid until the terminator has been reached.)
		.WRITE	The last byte was not a terminator.
	FORMATTED ASCII SPECIAL (parity or non-parity)	.READ	The MAXIMUM BYTE COUNT was reached before a line terminator was seen (excess data has not yet been read).
		.WRITE	The ACTUAL BYTE COUNT was reached before any terminator was seen.
	FORMATTED BINARY NORMAL	.READ	The MAXIMUM BYTE ran out before the count stored with the data. (The last byte has been overlaid in order to verify the checksum.)
	FORMATTED BINARY SPECIAL	.READ	The MAXIMUM BYTE COUNT was reached before the count stored with the data. (The excess data still remains to be read and checksum has not been verified.)
	ALL UNFORMATTED MODES	.READ	BYTE COUNT = the actual number of bytes transferred. The reason BYTE COUNT < MAXIMUM BYTE COUNT is that an EOF or EOM has been encountered before the buffer was full. Bit 6 will also be set.
1 (CHECKSUM ERROR)	FORMATTED BINARY	.READ	There was a discrepancy between the checksum accumulated during the .READ, and that stored with the incoming data.

2 (PARITY FORMAT)	FORMATTED ASCII PARITY NORMAL OR SPECIAL	.READ	A character was read which had odd parity. The eighth bit of the illegal character delivered is set to a 1. The transfer continues. If this bit is set the user need only check each character returned during processing of the buffer for bit 8 set to locate the character returned with wrong parity.
2 (ILLEGAL BINARY FORMAT)	FORMATTED BINARY	.READ	This bit is set if a line processed in a binary mode does not have a 001 in the first word. The first word is ignored, i.e., no data is returned to the buffer. Subsequent reads access successive lines and return error bits or data as appropriate.
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. No data is returned on .READs unless bit Ø is also set (see bit Ø). On .WRITEs an unspecified portion of the buffer may have been written (enough data to fill a partially filled monitor buffer may have been transferred to the buffer and written before the EOM or EOF was detected). Subsequent requests return to user with this bit set.
5 (DEVICE PARITY)	ALL MODES	.READ or .WRITE	A hardware error has been detected on a bulk storage device. This could be either a parity error or a timing error. The driver will already have tried to READ or WRITE 8 or 9 times before setting this bit. (This flag is a warning that the data in this line or some subsequent line still using data from the same device block may be invalid. It will be returned for each transfer call using the same block.)

3.9.4 The RECORD Block

RECBLK:

FUNCTION / STATUS
BUFFER ADDRESS
RECORD LENGTH
HI ORDER, RECORD #
LO ORDER, RECORD #

Figure 3-12 The Record Block

ADDRESS — FUNCTION

RECBLK — FUNCTION / STATUS WORD

BIT

- Ø - Not used
- 1 - Record Output - Set by user
- 2 - Record Input - Set by user
- 3-8 - Not used

(Following bits set by Monitor)

- 9 - Illegal Function
- 1Ø - File is linked or device is not File structured.
- 11 - Record requested lies outside the file.
- 12 - File not OPEN
- 13 - Protect code violation, Incorrect Open
- 14 - Not used
- 15 - Device parity error

The user may set only bits 1 or 2; error bits are set by the Monitor, and should be tested for by the user upon return from the request. The error bits are cleared by the Monitor when a .RECRD request is issued and are set as appropriate upon return from the Monitor.

RECBLK+2 — BUFFER ADDRESS

The address of the user's buffer. The buffer must be large enough to contain a record of the length indicated in the next word, as the Monitor assumes that sufficient space is available and will overlay data stored below a buffer of insufficient length.

RECBLK+4 — RECORD LENGTH

The number of bytes of a Record. This value, which must remain the same for all records in the file, is supplied by the user.

RECBLK+6 — High Order - Record Number
RECBLK+1Ø — Low Order - Record Number

This entry identifies the record to be read or written. Two words are provided in anticipation of files with more than 65,536 records.

First Record of File is number Ø.

BLOCK Block

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

BLKBLK:

FUNCTION/STATUS
BLOCK NUMBER
MEMORY BUFFER ADDRESS
LENGTH

Figure 3-13 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
	3-8	reserved
error	9	illegal function
	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
status	12	file not open
	13	protect code violation
	14	end of data error
	15	device parity error

Address	Name	Function
BLKBLK+2	BLOCK NUMBER	Requested block number to be transferred relative to the beginning of the file. First block of file is 0.
BLKBLK+4	MEMORY BUFFER ADDRESS	The address of the buffer (supplied by the Monitor on INPUT or GET functions).
BLKBLK+6	LENGTH	The length of the buffer in words. BLKBLK+6 is set by the Monitor on INPUT or GET functions.

3.9.6 The TRAN Block (used by TRAN request only)

TRNBLK:

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

Figure 3-14 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. Block Ø is the first block on bulk storage devices. If it is not a bulk storage device, specify block Ø.
TRNBLK+2	BUFFER ADDRESS	User specifies here the core memory address at which the data transfer is to begin.
TRNBLK+4	WORD COUNT	User specifies here the total number of 16-bit words to be transferred. Word count may be more or less than block size.
TRNBLK+6	FUNCTION/STATUS	

Bit	Bit Meaning
0	Binary = 1, as opposed to ASCII = 0
1	Write = 1*
2	Read = 1*
3 4 5 6 7 8 9 10	Reserved for Monitor's use
11	DECtape direction* 0 = forward 1 = reverse
12	Reserved for RSX-11
13	Invalid call (improper function/no word count)**
14	End of medium**

*Must be specified by user.
**This bit is cleared by the Monitor upon .TRAN request issue and is set as appropriate upon return.

Address	Name	Function	
		Bit	Bit = 1 means:
		15	Recoverable device error (such as parity, timing, or record length)**
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.	

**This bit is cleared by the Monitor upon .TRAN request issue and is set as appropriate upon return.

3.9.7 The Special Functions Block (used for SPEC request only)

SPCBLK:

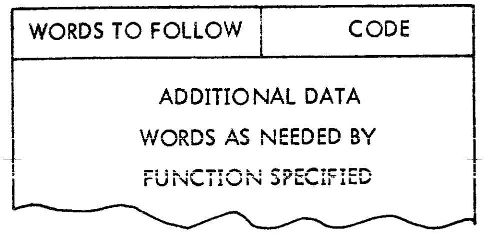


Figure 3-15

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

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

See Appendix J for a description of the special functions which may be performed for each device.

3.9.8 The RUN Block

The RUN Block is used exclusively with the .RUN request. It is a variable length control block containing a function word and several parameter words. The function word is always present; any of the parameter words may be omitted, depending upon the settings of the function word.

NOTE

Omitting a parameter word does not mean setting it to zero, but rather leaving it out. Hence, no parameter word occupies a set position in the RUN Block and the block itself is of variable length. For reference, all words but the function word are referred to by a letter, not by a number.

Word*	Parameter	Present If:
1	FUNCTION WORD	always
A	FILE BLOCK POINTER	Bit 15=0
B	LINK BLOCK POINTER	Bit 15=0
C	NAME	Bit 15=1 or Bit 13=1
D	NAME	Bit 15=1 or Bit 13=1
E	LOAD ADDRESS	Bit 3=1
F	TRANSFER ADDRESS OFFSET	Bit 4=1
G	RETURN ADDRESS	Bit 5=1

* Words A through G are so designated because any of them might be omitted under certain conditions.

Figure 3-16 The RUN Block

Address	Name	Function
RUNBLK	FUNCTION	User specifies here the function to be performed (see below).
RUNBLK+A	FILE BLOCK	Address of the File Block describing the file which contains the load module or core image to be loaded.
RUNBLK+B	LINK BLOCK	Address of the Link Block which describes the device from which the entity is to be loaded. Sufficient room must be provided in the Link Block to contain the EMT numbers of all Monitor modules which are to be loaded (these are contained in the load module, if there are any).

Address	Name	Function
RUNBLK+C and RUNBLK+D	NAME	Two Radix-50 words containing either the name of the specific core image to be loaded from a CIL (bit 13=1) or the name of the file to be loaded if no File Block was given (bit 15=1).
RUNBLK+E	LOAD ADDRESS	Specifies an address at which the entity is to be loaded, without regard to the load address in the load module or CIL. The entity should be position independent.
RUNBLK+F	TRANSFER ADDRESS OFFSET	Specifies a value to be added to the transfer address obtained from the load module or CIL. Provides for alternate entry points to the module.
RUNBLK+G	TRANSFER ADDRESS	Specifies an address to which control must be passed when loading is completed. This address may or may not be in the loaded entity.

3.9.8.1 The Function Word

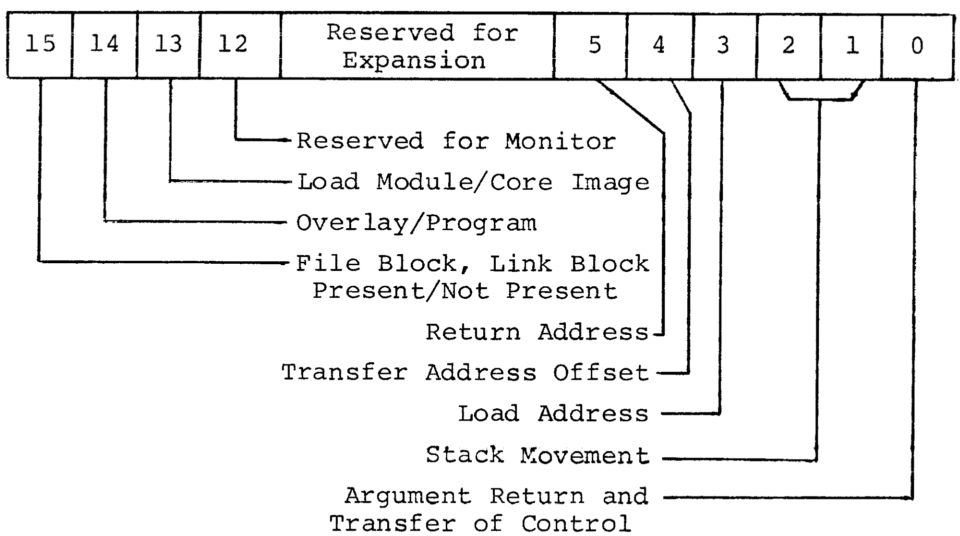


Figure 3-17 The Function Word

Bit Ø — Argument Return and Transfer of Control

=Ø Indicates control is to be returned to the instruction following the .RUN request after completing the requested actions, unless bit 5=1. Regardless of the setting of bit 5, the load module's transfer address, size in bytes, and low address will be on top of the stack when bit Ø=Ø (see Section 3.8.1.1).

=1 Indicates control is to be switched to the transfer address of the loaded module after completion of the load, unless bit 5=1. Regardless of the setting of bit 5, no information is returned on the stack when bit Ø=1, but information may be passed by the call to the loaded module either on the stack or in the general registers.

Bit 1 — Stack Movement

=Ø Indicates that the stack is not to be moved from its present position under any condition.

=1 Indicates that stack relocation may be necessary and that bit 2 of this word must be tested to determine under what conditions relocation will be necessary.

Bit 2 — Movement Condition

=Ø Indicates that the stack is to be unconditionally moved to the area directly below the module to be loaded. In this position the stack base entry in the System Vector Table (SVT) will be the same as the low address of the loaded module.

=1 Indicates that the stack is to be conditionally moved, based on the relative positions of the stack base and low address of the module to be loaded. If the stack base entry in the SVT is higher than the low address of the module to be loaded, then the stack should be relocated as described above. If the stack base entry in the SVT is lower in core or equal to the low address of the module to be loaded, then the stack will not be relocated.

Bit 3 — Load Address

=Ø Indicates that no optional load address is specified in the RUN Block. The load address information in the load module will be used.

=1 Indicates that the address specified in the RUN Block is to be used as the load address for the requested module. This entry overrides the load module information.

Bit 4 — Transfer Address Offset

=Ø Indicates that no offset from the module's transfer address is included in the RUN Block.

=1 Indicates that the user desires an offset, specified in the RUN Block, to be added to the loaded module's transfer address. This offset is added to the transfer address regardless of the setting of bit 0 of the action word.

Bit 5 — Return Address

=Ø Indicates that no alternate return address is included in the RUN Block. Return of control will thus be determined by the setting of bit 0.

=1 Indicates that an alternate return address has been specified in the RUN Block and that this address will receive control instead of the address following the .RUN request or the transfer address of the load module. The setting of bit 0 will still determine whether information will be returned on the stack.

Bit 12 — Reserved for Monitor

This bit should always be zero.

Bit 13 — Load Module/Core Image

=Ø Indicates that the entity being loaded is a load module. If the file identified by the File Block is a CIL, the first member of the CIL will be loaded.

Bit 13 (continued)

=1 Indicates that the entity to be loaded is a member of Core Image Library. The File Block identifies the CIL, while words 4 and 5 of the RUN Block contain the name of the CIL member.

<u>Bit 14</u> <u>Overlay/Program</u>

=Ø Indicates that an overlay is being loaded. Since this is a continuation of the current program, datasets may be left open across this call. The overlay may not extend above the low address of the resident module, nor may it extend below the top of the Monitor area. System control tables are not refreshed as a consequence of this call. No additional Monitor modules may be made resident.

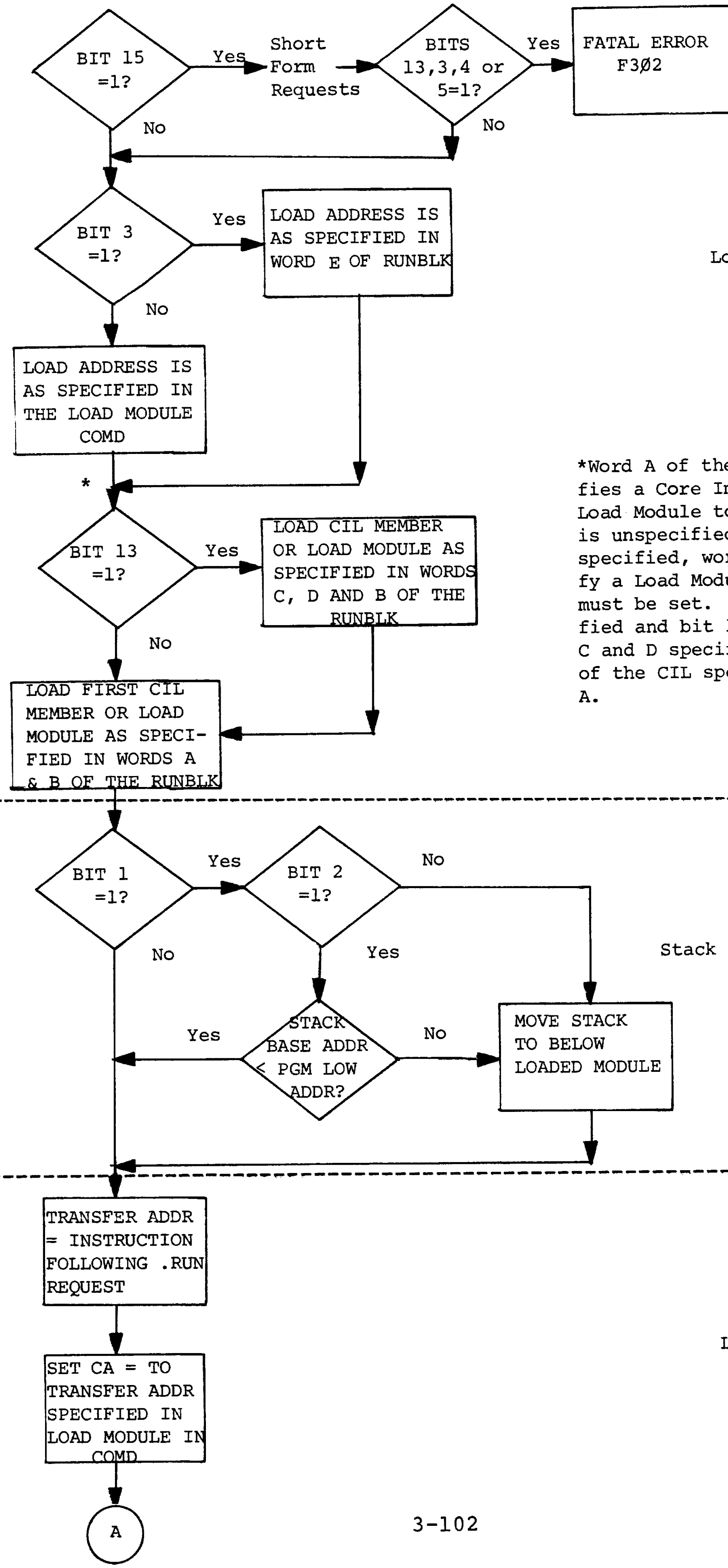
=1 Indicates that a new program is being loaded. This is as if a new program were being RUN from the keyboard. Although all datasets must be released by the program which called RUN, RUN itself will do several things to refresh the environment. This includes releasing Monitor modules made resident by the previous program, undoing dataset assignments made specifically for the previous program, loading any Monitor modules which should be resident for this program, and changing any program-related values in the SVT.

<u>Bit 15</u> <u>File Block, Link Block</u>

=Ø Indicates that a Link Block and a File Block pointer are in the RUN Block.

=1 Indicates that the caller has provided a short form of the RUN Block; the short form contains only a function word and a six-character filename. The Link Block and File Block are created by the .RUN request itself. The entity to be loaded must be either in the current user's area or in the [1,1] UIC area and must have an extension of LDA or null. All other function bits are ignored. The load module or core image (first member of CIL) is loaded at its normal load address, as if it were an overlay, and receives control at its normal transfer address. The stack is not moved.

The following flowchart illustrates the effects of the various function word bits and their interrelationships.

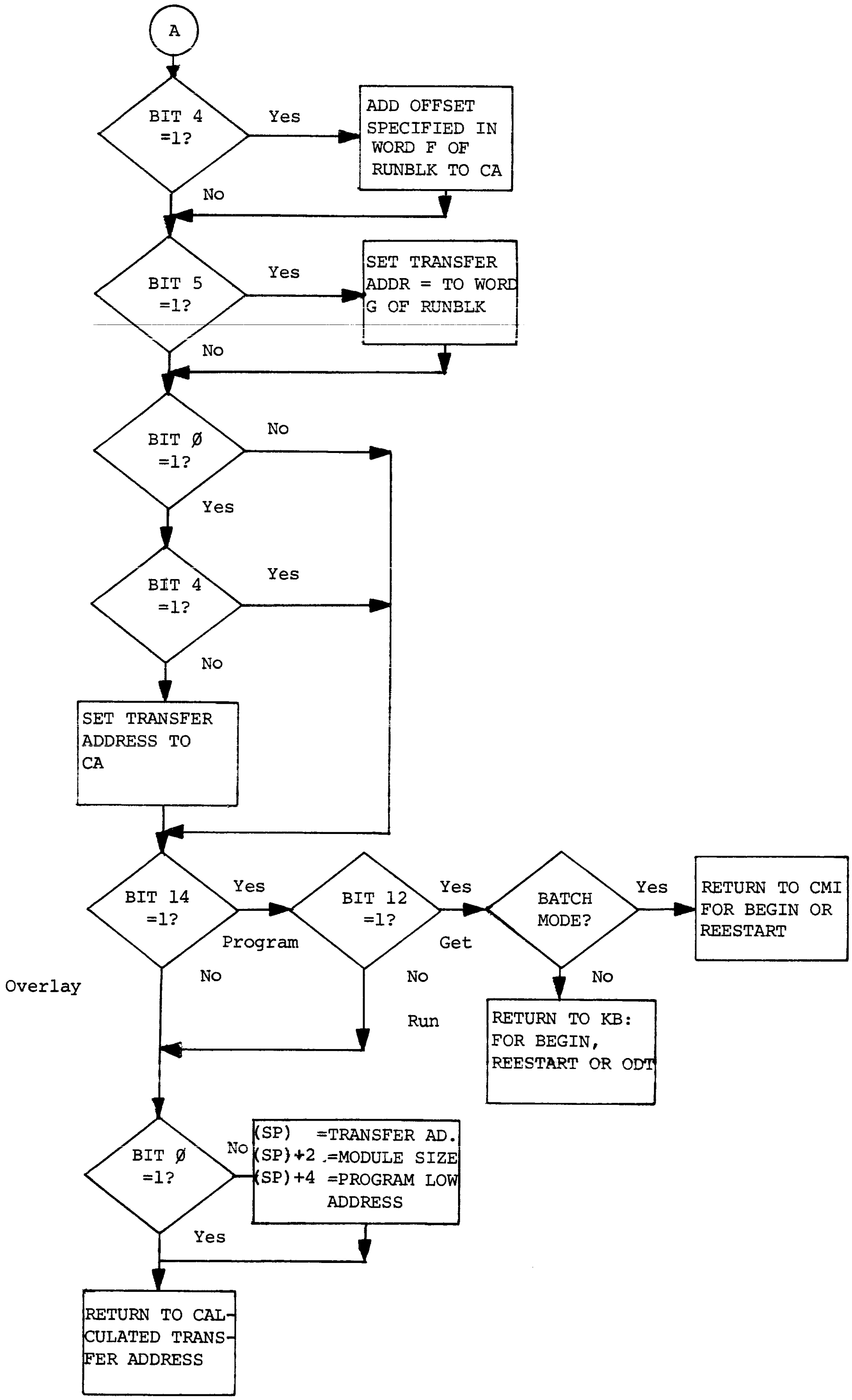


Load new module

*Word A of the RUNBLK specifies a Core Image Library or Load Module to be loaded or is unspecified. If it is unspecified, words C and D specify a Load Module and bit 13 must be set. If it is specified and bit 13 is set, words C and D specify a CIL member of the CIL specified in word A.

Stack Movement

Load new module



A
BIT 4
=1?
Yes
ADD OFFSET SPECIFIED IN WORD F OF RUNBLK TO CA
No
BIT 5
=1?
Yes
SET TRANSFER ADDR = TO WORD G OF RUNBLK
No
BIT Ø
=1?
No
Yes
BIT 4
=1?
Yes
No
SET TRANSFER ADDRESS TO CA
BIT 14
=1?
Yes
Program
No
Overlay
BIT 12
=1?
Yes
Get
No
Run
BATCH MODE?
Yes
RETURN TO CMI FOR BEGIN OR REESTART
No
RETURN TO KB: FOR BEGIN, REESTART OR ODT
BIT Ø
=1?
No
(SP) =TRANSFER AD.
(SP)+2 =MODULE SIZE
(SP)+4 =PROGRAM LOW ADDRESS
Yes
RETURN TO CALCULATED TRANSFER ADDRESS

APPENDIX A
PHYSICAL DEVICE NAMES

Mnemonic	Device	
DC	RC11 Disk	014570
DF	RF11 Disk	014760
DK	RK11 Disk	015270
DT	DECtape (TC11)	016040
KB	ASR-33 Keyboard/Teletype	042420
LP	Line Printer (LP11)	046600
MT	Magtape (TM11)	052140
PP	High-Speed Paper Tape Punch	063200
PR	High-Speed Paper Tape Reader	063320
PT	ASR-33 Paper Tape Device	063440
CR	Card Reader (CR11)	012620
SY	System Residence Device (DC, DF, or DK)	075250

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

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

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

DT1 indicates unit 1 under a single DECtape control.

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

The Radix-50 equivalence is derived in accordance with the following formula:

$$C_1 \times 50_8{}^2 + C_2 \times 50_8 + C_3$$

where C_n is a character (legal characters are space A-Z, $, period, and 1-9. These characters are assigned values from Ø (for space) through 47_8 (for 9).

The following program may be used to print the octal representation of any 3-character set Radix-5Ø equivalence. To exit type an illegal character.

APPENDIX B
EMT CODES

EMT Code	Programmed Request	Described on Page
0	.WAITR	3-36
1	.WAIT	3-35
2	.WRITE	3-29
3	[2]	
4	.READ	3-28
5	[2]	
6	.INIT	3-20
7	.RLSE	3-21
10	.TRAN	3-33
11	.BLOCK	3-31
12	.SPEC	3-37
13	.STAT	3-38
14	.LOOK	3-44
15	.ALLOC	3-39
16	.OPENx	3-22
17	.CLOSE	3-22
20	.RENAM	3-26
21	.DELET	3-41
22	.APPND	3-43
24	.KEEP	3-46
25	.RECRD	3-30
26-27	[2]	
30-31	[1]	
32	Diagnostic Print	
33-35	[1]	
36-37	[2]	
40	[1]	
41	General Utilities	3-50,-66
42	General Conversions	3-67,-74
43-55	[1]	
56,57	Command String Interpreter	3-76,77
60	.EXIT	3-49
61-63	[1]	
64	[1]	
65	.RUN	3-47
66	.CVTDT	3-57
67	[2]	
68-76	[2] (70, reserved for Multi-User Operation)	
77	[1]	
100-117	(reserved for Communications Executive, COMTEX-11)	
120-137	(reserved for Real-Time Monitor, RSX-11)	
140-167	(reserved for user-implemented routines)	

[1]Reserved for Monitor internal communication.

[2]Reserved for future Monitor expansion.

APPENDIX C
SUBSIDIARY ROUTINES AND OVERLAYS

With the exception of .READ/.WRITE and .WAIT, all Monitor code for performing programmed requests is potentially non-resident. Since non-resident modules are limited to a size of 256 words (the size of the swap buffer) and since many common functions are required, many of the programmed request modules must make use of subsidiary routines. The table given below can be used in two ways:

- when assessing the number of disk accesses required to satisfy a request, the table shows how many modules (in addition to the primary module) may be loaded;
- when making certain functions resident, one must not only make the primary module resident, but must also make resident each of the subsidiary modules which may be called. For example, if one wants all .OPENI processing routines (except for magtape) resident, he would put the following assembler directive in his program:

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

The following summary explains the codes used in the table.

(blank) = subsidiary routine is never called

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

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

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

D = subsidiary routine is called only when DECtape is referenced

M = subsidiary routine is called only if magtape is referenced

		Name of Subsidiary Routine														
Global Name of Primary Module	Request	FOP Open Existing File	FCR Creating New Linked File	FCL Close File	LUK Directory Search	LBA Allocate Block, Linked File	GMA Get Bit Map Segment	CBA Allocate Contiguous Blocks	CKX Check Access	DLN Delete Linked File	DCN Delete Contiguous File	AP2 Append DECtape	GNM[2] Get Next Bit Map Segment	MTO Magtape Open	LDR[4] Loader	LD2[4] Loader
RWN	.READ/WRITE[1]												X			
OPN	.OPENU	X			X				X					M		
OPN	.OPENO[3]		X		X	X	X		X					M		
OPN	.OPENE	X			X	X	X		X					M		
OPN	.OPENI[4]	X				X			X					M		
OPN	.OPENC	X				X			X							
CLS	.CLOSE[4]			X												
ALO	.ALLOC				X			X	X							
DEL	.DELET				X				X	L	C					
REN	.RENAM				X				X							
APP	.APPND				X				X			D				
DIR	.LOOK				X				X							
PRO	.KEEP				X				X							
RUN	.RUN[4]	X		X	X				X					M	X	X
INR	.INIT[5]															
RLS	.RLSE[4]															

[1]Always resident.

[2]Should never be made resident.

[3]The .OPENO module requires a second section if a dataset other than CMO is being opened on the device assigned to CMO.

[4]The .RUN EMT calls the following routines:

.INIT	
.OPENI	(once for each combination of filename and UIC)
.LDR	(three sections if LDA file; two if CIL file)
.LD2	
.CLOSE	(once for each .OPENI)
.RLSE	

[5]The .INIT module has two sections, but the second has no name. It is resident automatically if .INIT is resident.

APPENDIX D

SUMMARY OF MONITOR COMMANDS

Command	Usage
Commands to Allocate System Resources	
ASSIGN	Assign a physical device to a logical device name
Commands to Manipulate Core Images	
RUN	Load and begin a program
GET	Load a program
DUMP	Write a specified core area onto a device as a core image
SAVE	Write a program onto a device in loader format
Commands to Start a Program	
BEGIN	Start execution of a program
CONTINUE	Resume execution of a halted program
RESTART	Restart execution of a previously operating program
Commands to Stop a Program	
STOP	Halt the current program, including any I/O in progress
WAIT	Halt current program after finishing any I/O in progress
KILL	Halt the current program, finish any I/O in progress, close all open files, and pass control back to the Monitor
Commands to Exchange Information with the System	
DATE	Fetch/Specify date
TIME	Fetch/Specify time

(continued on next page)

Optional characters are underlined. If any optional character appears, all must appear.

Command	Usage
Commands to Exchange Information with the System (Cont)	
LOGIN	Enter User Identification Code
MODIFY	Modify contents of memory location
FINISH	Log off system
Miscellaneous Commands	
ECHO	Disable/enable keyboard echo to user program
PRINT	Disable/enable teleprinter output from user program
END	End input from a device
ODT	Begin operation of Octal Debugger (ODT)

Optional characters are underlined. If any optional character appears, all must appear.

APPENDIX E

SUMMARY OF MONITOR PROGRAMMED REQUESTS

Global 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
				3-39

.APPND	Append to a Linked File	.APPND #LNKBLK,#FIRST,#SECOND	MOV #SECOND,-(SP)
			MOV #FIRST,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 22
				3-43

.BIN2D	Convert Binary to Decimal ASCII	.BIN2D #ADDR,WORD	MOV WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #3,-(SP)
			EMT 42
				3-72

.BIN2O	Convert Binary to Octal ASCII	.BIN2O #ADDR,WORD	MOV WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 42
				3-74

.BLOCK	Transfer a Block	.BLOCK #LNKBLK, #BLKBLK	MOV #BLKBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 11
				3-31

.CLOSE	Close a Dataset	.CLOSE #LNKBLK	MOV #LNKBLK,-(SP) EMT 17	3-26
.CORE	Obtain Core Size	.CORE	MOV #100,-(SP) EMT 41	3-52
.CSI1	CSI Interface - part 1	.CSI1 #CMDBUF	MOV #CMDBUF,-(SP) EMT 56	3-76
.CSI2	CSI Interface - part 2	.CSI2 #CSIBLK	MOV #CSIBLK,-(SP) EMT 57	3-77

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

.CVTDT	Convert Binary Date or Time to ASCII character string	.CVTDT #CODE,#ADDR[,VALUE]
		VALUE is an optional argument specified with Codes 2 and 3 only.
			If Code = 3
			MOV VALUE+2,-(SP)
			If Code = 2 or 3
			MOV VALUE,-(SP)
			All codes
			MOV #ADDR,-(SP)
			MOV #CODE,-(SP)
			EMT 66
				3-57

.DATE	Obtain Date	.DATE	MOV #103,-(SP) EMT 41	3-55

.DELET	Delete a File	.DELET #LNKBLK,#FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 21
				3-41

.D2BIN	Convert Decimal ASCII to Binary	.D2BIN #ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 42
				3-71

.EXIT	Exit to Monitor	.EXIT	EMT 60	3-49
.GTCIL	Get disk address of Core Image library	.GTCIL	MOV #111,-(SP) EMT 41	3-63
.GTUIC	Get Current UIC	.GTUIC	MOV #105,-(SP) EMT 41	3-59
.GTPLA	Get Program Low Address	.GTPLA	CLR -(SP) MOV #5,-(SP) EMT 41	3-61
.GTSTK	Get the Stack Base Address	.GTSTK	CLR -(SP) MOV #4,-(SP) EMT 41	3-64

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.INIT	Initialize a Dataset	.INIT #LNKBLK	MOV #LNKBLK,-(SP) EMT 6	3-20

.KEEP	Protect a File	.KEEP #LNKBLK,#FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 24
				3-46

.LOOK	Directory Search	.LOOK #LNKBLK,#FILBLK[,1]
		
		,1 is an optional argument
			MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
			or when optional argument is specified:
			
			MOV #FILBLK,-(SP)
			CLR -(SP)
			MOV #LNKBLK,-(SP)
			EMT 14
				3-44

.MONF	Obtain Full Monitor Size	.MONF	MOV #102,-(SP) EMT 41	3-54
.MONR	Obtain Size of Resident Monitor	.MONR	MOV #101,-(SP) EMT 41	3-53

.OPEN	Open a Dataset	.OPEN #LNKBLK,#FILBLK	MOV #FILBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
				3-22

.OPENx	Open a Dataset	.OPENx #LNKBLK,R	MOV #CODE,-2(R)
			MOV R,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 16
			
			CODE=1 for .OPENU
			2 for .OPENO
			3 for .OPENE
			4 for .OPENI
			13 for .OPENC
				3-22

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

.O2BIN	Convert Octal ASCII to Binary	.O2BIN #ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 42
				3-73

.RADPK	Radix-50 ASCII Pack	.RADPK #ADDR	MOV #ADDR,-(SP) CLR -(SP) EMT 42	3-67

.RADUP	Radix-50 ASCII Unpack	.RADUP #ADDR,WORD	MOV WORD,-(SP)
			MOV #ADDR,-(SP)
			MOV #1,-(SP)
			EMT 42
				3-70

.READ	Read from Device	.READ #LNKBLK,#BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 4
				3-28

.RECRD	Read or Write a Specified Record in a File	.RECRD #LNKBLK,#RECBLK	MOV #RECBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 25
				3-30

.RENAM	Rename a File	.RENAM #LNKBLK,#OLDNAM,#NEWNAM	MOV #NEWNAM,-(SP)
			MOV #OLDNAM,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 20
				3-42

.RLSE	Release a Dataset	.RLSE #LNKBLK	MOV #LNKBLK,-(SP) EMT 7	3-21

.RSTRT	Set REstart address	.RSTRT #ADDR	MOV #ADDR,-(SP)
			MOV #2,-(SP)
			EMT 41
				3-51

.RUN	Load a program or Overlay	.RUN #RUNBLK	MOV #RUNBLK,-(SP) EMT 65	3-47

.SPEC	Special Function	.SPEC #LNKBLK,#SPCARG	MOV #SPCARG,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 12
				3-37

Global Mnemonic	Function	Macro Call (see notes)	Assembly Language Expansion (see notes)	Refer to Page
.STAT	Obtain Device Status	.STAT #LNKBLK	MOV #LNKBLK,-(SP) EMT 13	3-38

.STPLA	Set Program Low Address	.STPLA #ADDR	MOV #ADDR,-(SP)
			MOV #5,-(SP)
			EMT 41
				3-62

.STFPU	Initialize the Floating Point exception vector (11/45)	.STFPU #PSW,#ADDR	MOV #ADDR,-(SP)
			MOV #PSW,-(SP)
			MOV #3,-(SP)
			EMT 41
				3-66

.STSTK	Set the Stack Base Address	.STSTK #ADDR	MOV #ADDR,-(SP)
			MOV #4,-(SP)
			EMT 41
				3-65

.SYSDV	Obtain System Device Name	.SYSDV	MOV #106,-(SP) EMT 41	3-60
.TIME	Obtain Time of Day	.TIME	MOV #104,-(SP) EMT 41	3-56

.TRAN	Transfer Absolute Block	.TRAN #LNKBLK,#TRNBLK	MOV #TRNBLK,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 10
				3-33

.TRAP	Set TRAP Vector	.TRAP #STATUS,#ADDR	MOV #ADDR,-(SP)
			MOV #STATUS,-(SP)
			MOV #1,-(SP)
			EMT 41
				3-50

.WAIT	Wait for Completion	.WAIT #LNKBLK	MOV #LNKBLK,-(SP) EMT 1	3-35

.WAITR	Wait for Completion; Return to ADDR	.WAITR #LNKBLK,#ADDR	MOV #ADDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 0
				3-36

.WRITE	Write on a Device	.WRITE #LNKBLK,#BUFHDR	MOV #BUFHDR,-(SP)
			MOV #LNKBLK,-(SP)
			EMT 2
				3-29

NOTES:

ADDR	a memory address
BLKBLK	address of BLOCK Block
BUFHDR	address of Line Buffer Header
CMDBUF	address of Command String Buffer
CSIBLK	address of Command String Interpreter Control Block
FILBLK	address of Filename Block
FIRST	address of Filename Block of file which is to be appended to
LNKBLK	address of Link Block
N	number of 64-word segments requested
NEWNAM	address of Filename Block containing the file's new name
OLDNAM	address of Filename Block containing the file's old name
PSW	program status word for an exception routine
R	register from RØ through R5 containing address of Filename Block
RECBLK	address of RECORD Block
SECOND	address of Filename Block of file which is appended
SP	Stack Pointer (register R6)
SPCARG	code for Special Function or Address of Special Function Block as determined by Function called.
TRNBLK	address of TRAN Block

APPENDIX F
SUMMARY OF DOS ERROR MESSAGES

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

F.1 Keyboard Command Messages

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

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

F.2 Error Messages

Error messages are printed on the teleprinter in the following format.

CNNN XXXXXX

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

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

NNN is the message number; and XXXXXX gives appropriate additional information. Information, Warning, and System program messages are printed and the program continues.

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

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

F.2.1 Action Message

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).

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
AØØ1 DOS	User Call Address Disk address error.
AØØ2 DOS	Device (RAD5Ø) Device not ready. For example, the desired device/unit may be off-line or it may not be write-enabled. For DECtape or magtape, the proper unit may not have been selected. Make the device ready and type CO.
AØØ3 DOS	Link Block Address The Link Block contains either an illegal device code or no device code at all. Use the MODIFY command to display the contents of Link Block+2, which is the dataset name (RAD5Ø), and then use the ASSIGN command to assign a device and/or file; type CO when ready.
AØØ4 DOS	User Call Address DECtape error. Try adjusting the tape; type CO to retry the operation.
AØØ5 OTS	Pause Number A PAUSE was encountered in a FORTRAN program. Type CO to continue.
AØØ6 LINK	Correct Module Name Paper tape loaded out of order on Pass 2 of Linker. Load correct module and type CO to continue.
AØØ7 DOS	Call Address The name of the output file being created on magtape is the same as that of an existing file. Type CO to write over the old file or mount another tape and then type CO.
AØ1Ø DOS	Ø A parity error occurred when trying to open a file on magtape. Type CO to continue searching. If the file being sought has a parity error in its label, it cannot be found.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
AØ11 DOS	Ø = Date is Bad, 1 = Time is Bad System date or time is not valid. Re-enter date or time via the console keyboard and type CO to continue.
AØ12 DOS	Status Register Magtape error. After having made 15 entries on a WRITE or WRITE EOF, the operation is still unsuccessful. Type CO to ignore the error and proceed, or type KI to stop the program and start over with a good tape.
AØ43 PIP	Disk Pack Block Number This is the block that is bad; issued by the RP11 pack initializer to provide a list of bad blocks and to permit job termination if too many are bad. Type CO if number of bad blocks thus far is tolerable.
AØ5Ø BATCH	Ø Batch Stream Wait. Type CO to continue.
A35Ø DOS	Ø Power has come up following a power failure. Any I/O in progress has been lost, but information in core and in the registers has been retained. If you wish to continue, type CO. Note, however, that if I/O was in progress, the driver(s) may have been left in a state which will not permit your program to be continued.

F.2.2 Information Messages

Information messages are printed and the program generally continues.

CODE/ISSUER		ADDITIONAL INFORMATION/MEANING
I350	OTS	STOP Number A STOP statement was executed in a FORTRAN program.
I351	FORTRN	Ø More errors of a specified type occurred than were allowed. The program is terminated.
I352	FORTRN	Address of DEVTB Entry The logical device specified is not available, (See FORTRAN device table, DEVTB, for a layout.)
I353	OTS	Error Class Number No logging device. The command input device was in use when a run-time diagnostic message was to be issued. Because of a device conflict the normal message could not be issued.
I354	PIP	Ø Illegal response to CONFIRM; when attempting to zero an RK11 disk cartridge. The disk was not zeroed. Legal responses are: H for high-density disks (RKØ3/Ø5) L for low-density disk (RKØ2).

F.2.3 Warning Messages

Warning messages are printed and the program generally continues.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
WØØ2	Device Name (RAD5Ø) Device time out.
WØ43	Block Number Transfer error while using .TRAN to zero the disk.
W1Ø1 RSX	Number of Task Called Task called by number not present or call number illegal. Request ignored.
W1Ø2 RSX	Addr. in Call Sequence Delay units not correct in call start. Request ignored.
W1Ø3 RSX	Addr. in Call Sequence Delay time too large in call start. Request ignored.
W1Ø4 RSX	Addr. in Call Sequence No time slot available. Request ignored.
W1Ø5 RSX	Current Run-Time A level 1 task has exceeded its maximum run time. Task continued.
W1Ø6 RSX	Illegal or unrecognized console command. Command ignored.
W1Ø7 RSX	Report Number Illegal system report number in system command. Command ignored.
W11Ø RSX	Addr. in Call Sequence Attempted to start a background task while the background is busy. Request ignored.
W111 RSX	Addr. in Call Sequence Attempted to clock a background task. Request ignored.
W112 RSX	Symbolic task name not found. Request ignored.
W113 RSX	Command syntax error. Command ignored.
W114 RSX	Addr. in Call Sequence Illegal clock (call TRNON) time. Request ignored.
W3ØØ LINK	Ø, Module Name Non-unique object module detected in first pass. Second and subsequent occurrences of the module are ignored.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
W3Ø1 LINK	Addr. of Byte Error Byte relocation error. Linker automatically continues.
W3Ø2 LINK	Ø, Symbol and Module Names Multiple definitions of global symbol. Second definition is ignored and linking continues.
W3Ø3 EDIT	Buffer overflow. Overflow of one of the following Editor buffers: Command Input Buffer Save Buffer Page Buffer
W3Ø4 EDIT	Macro overflow. The command string as stored in the Save Buffer was too long to execute, when requested to do so by an EM (Execute Macro) command.
W3Ø5 EDIT	Recursive macro. The command string as stored in the Save Buffer contains an EM command.
W3Ø6 EDIT	Empty Save Buffer. An EM or U (Unsave) command was issued with nothing in the Save Buffer.
W3Ø7 EDIT	Search failure. The n^{th} occurrence of the search object was not found in the available test.
W31Ø EDIT	Unsave failure. Insufficient room to copy the contents of the Save Buffer into the Page Buffer at dot.
W311 EDIT	End-of-data detected. The end of the input file or the end of the input medium was reached during the last read of text into the Page Buffer, last page read was last in the file.
W312 EDIT	Illegal line feed. A line feed character was encountered in the command string.
W313 EDIT	Illegal negative argument. A negative argument was used with a command that does not accept negative arguments.

CODE/ISSUER ADDITIONAL INFORMATION/MEANING

W314
EDIT
Arguments not permitted. The command specified does not permit any argument with it.

W315
~~EDIT~~
~~Illegal argument. The given~~ argument was not acceptable to the specified command.

W316
EDIT
Illegal text string.

W317
EDIT
Illegal command. The Editor was unable to execute the specified command. The command may be an illegal character, one that is not an EDIT-11 command character.

W32Ø
EDIT
Page Buffer almost full. The Page Buffer was within 128 characters of being full. Write out part or all of the Page Buffer and then delete from the Buffer the part that was written.

W321
EDIT
File closed. An attempt to Read from or Write to a primary file after an EF (End-of-File) command was issued.

W322 Ø
LINK
Undefined global symbols in load module. Linking continues.

W323
RSX
Illegal size of named .CSECT or illegal entry in named .CSECT or task's named .CSECT size too large.

W324
RSX
Too many entries in tasks named .CSECT.

W325
RSX
Illegal priority specification in real-time header.

W35Ø Number of Failures
RSX
Powerfail interrupt occurred.

W352 Disk Error Code
RSX
Disk error detected by RSX.
Codes are:
3 transmission error
5 illegal error
6 undefined file
7 illegal file, i.e., linked
8 block of file out of range

F.2.4 Fatal Messages

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

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
FØØØ DOS	Request Address Dataset not INITed. Program must issue .INIT before any other requests to a dataset.
FØØ1 DOS	Request Address Stack overflow. Once loaded, a program requires additional space for its stack, buffers and control blocks. These are allocated as they are needed. Reduce the size of the program. If the error has been caused by a stack overflow, the stack pointer is reset by bytes before the message is printed. This allows the monitor to proceed (since it needs the stack) and leaves the top of the stack intact (though not pointed to by SP). (See F.2.)
FØØ2 DOS	Request Address Invalid EMT call. The EMT code issued by the program has not been assigned.
FØØ3 DOS	Request Address Invalid .TRAN function or .TRAN to an open file.
FØØ4 DOS	Error Code Incorrect OPEN on industry compatible magnetic tape. Caused by program error or improperly assigning devices via datasets. Defined error code values: Ø - another file currently opened on tape, 1 - attempt to READ or WRITE to unopened file.
FØØ5 DOS	Request Address .RLSE error. If a file has been OPENed, it must be CLOSEd before a .RLSE can be issued.
FØØ6 DOS	Request Address Device full. No more space exists on the device being referenced by the request. For a file-structured device, use PIP to look at the number of free blocks and delete any files which are not needed.
FØØ7 DOS	Request Address No buffer space available. Insufficient space for completion of required operation. Reduce program size or close open files.

CODE/ISSUER		ADDITIONAL INFORMATION/MEANING
FØ1Ø	DOS	Request Address Illegal .READ/.WRITE. Incorrect mode for device or file not opened correctly.
FØ11	DOS	Request Address Illegal OPEN. OPEN code is not used or is unsuitable for device.
FØ12		Request Address File access violation. You are trying to OPEN a file that cannot be opened for the requested purpose. See Table 1 below for details. Assure that the name of the file requested was correct.
FØ14	DOS	Request Address Device error on trying to read bit map. The system cannot proceed if it cannot read the bit map. New files cannot be created on the device nor can old files be extended. Existing files may be copied to a backup medium for recovery.
FØ15	DOS	Request Address DECtape error. Nonexistent memory addressed or end-zone reached during transfer.
FØ16	DOS	Block Number DECtape search failure. Block requested cannot be found.
FØ17	DOS	Device (RAD5Ø) Parity error on file-structured device.
FØ2Ø	DOS	Irrelevant Too many datasets using low-speed paper tape. A maximum of one each for input or output is allowed. Restart your job and use the ASSIGN command to reassign the excess datasets.
FØ21	DOS	Irrelevant Checksum error or device parity error while typing to load a program. Type KILL then try again. If that doesn't work, try relinking the program. Try recreating the file. If the error persists, hardware may be faulty. Call field service.
FØ22	DOS	Irrelevant An attempt was made to load for execution a dataset which is not formated binary or which has no start address. Typically this means that the dataset being loaded is not a load module.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
FØ23 DOS	Program Size Program too large for core available. Try to overlay the program or make it smaller.
FØ24 DOS	Request Address File access violation. You are trying to perform an operation that violates the monitor's user and file protection scheme. See Table 1 below for details. Resolve access problems with owner.
FØ25 PIP	Device (RAD5Ø) Master directory full when attempting to add UIC. No more UIC's can be added.
FØ26 DOS	Disk Control Status Register Disk (RF11 or RC11) transfer failure. Hardware error or persistent parity failure.
FØ27 DOS	Error Register Disk (RK11) transfer failure.
FØ3Ø OTS	Error Class, Number FORTRAN system error. An illegal call to the FORTRAN Error Processor was made.
FØ31 OTS	Addr. of Log Device No more room on FORTRAN logging deivce, or illegal end-of-file was encountered while a FORTRAN READ was in progress.
FØ32 DOS	Status Register Magtape hardware error.
FØ33 DOS	Special Function Block Address Invalid special function block.
FØ34 DOS	Call Address The call code passed to a conversion request was invalid, e.g., 5 means binary-to-octal, but 63 is not defined.
FØ35 DOS	Block Number Illegal block number (RK11).
FØ36 RSX	Lowest Slot Used by Tasks No slot available.
FØ37	Lowest Slot used by Tasks Illegal slot specified.
FØ4Ø RSX	Low Address of Task Code Attempted to overlay the executive for another task.

CODE/ISSUER		ADDITIONAL INFORMATION/MEANING
FØ41	RSX	Load address of Binary Block Attempted to load outside limits defined in the command.
FØ42	DOS	Error Register Disk (RP11) transfer failure.
FØ43	DOS	Block Number Illegal block number (RP11).
~~FØ44~~	LINK	~~Ø~~ Error in command string passed by a Compiler via the .RUN request.
FØ45	DOS	Request Address The RUN EMT cannot find the requested entry in the specified core image library. Add proper entry to CIL or use correct name.
FØ5Ø	BATCH	Request Address Illegal I/O to batch stream. Either an illegal mode (e.g., unformatted binary when not in "OWN" mode) or a byte count less than 83, on formatted read.
FØ51	BATCH	Request Address Too many successive read errors or EOF's while reading the batch stream.
FØ52	BATCH	PC Illegal Open to one of the Batch Datasets. OPENO and OPENI are the only legal OPEN's and OPENO (OPENI) to an input (output) dataset is also illegal.
FØ53	BATCH	PC Illegal request to the BATCH stream flush EMT. Request code must be Ø, 1, or 2.
FØ54	DOS	Address of DDB An attempt was made to load a new program via the RUN request (EMT) before releasing all of the datasets INITed by the current program. Correct the program by releasing all INITed datasets before the RUN request is issued.
FØ55	BATCH	PC The time limit for the current job has expired. The current job has been aborted.
F1ØØ	RSX	Address in Call Sequence Insufficient arguments in call sequence or in console command.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
F24Ø DOS	Irrelevant An attempt was made to allocate a contiguous file, but not enough contiguous blocks are free.
F274 DOS	Irrelevant The stack base address has not properly set. Thus the stack could not be moved by the RUN EMT as requested. This is probably a program error. The .STSTK request may be used to set the stack base prior to issuing the .RUN request.
F275 OTS	Ø Incorrect argument to link subroutine.
F276 DOS	Request Address The transfer address of the program or overlay to be loaded (by the RUN or GET commands or by the .RUN request) was not specified or is not legal. Specify a transfer address in your source program (END statement) or correct the /TR specification in your linking procedure.
F277 DOS	Request Address The program or overlay could not be loaded because it was outside the legal load area (on top of the Monitor or the main program or outside actual memory). Relink the program to conform to allowable boundries. Assure that the section being improperly loaded does not overlay the resident portion of your program.
F3ØØ FORTRN	Ø FORTRAN Compiler overlays cannot be executed. FORTRN.OVR may be nonexistent or improperly constructed.
F3Ø1 FORTRN	Ø No output file specified for the "/GO" options.
F3Ø2 DOS	Action Word. Illegal options requested in short form of RUN△EMT.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
F340 DOS	PC at Time of IOT The DOS error routine was called with an invalid error code. This might happen if the program branched into a data area since the integer 4 would be ~~executed as an IOT instruction~~ (the error routine is called via an IOT).
F342 DOS	Contents of PC Error trap. Probably caused by a reference to a byte boundary or to nonexistent memory or to a nonexistent device. Could also be caused as a consequence of the stack pointer being below 400 or by executing JMP or JSR with register mode destination.
F344 DOS	Contents of PC Reserved instruction trap. The instruction just executed is not a valid PDP-11 instruction. Perhaps you jumped to a point outside your program or perhaps you have stored information over an instruction.
F346 DOS	Contents of PC Trace trap. Bit 4 of the Processor Status Register is on. Look for traps in the PDP-11 Processor Handbook.
F352 DOS	Contents of PC Trap Instruction trap. A trap instruction was issued by your program and you did not previously specify a trap address with the .TRAP request.
F356 DOS	Contents of PC Unexpected device interrupt. Either a new device has been added to your system without initializing the interrupt vector or a hardware failure has occurred.

Table F-1

Recovery from FØ12 or FØ24 File Access Violations

CONDITION	ACTION
Are you logged in?	LOgin
Is your UIC entered?	Enter it with PIP.
Are you attempting to create a file which already exists?	Run PIP and DELETE
Does the Input file you are accessing exist?	Use PIP with /BR or /DI switch to check
Are you attempting to delete a non-existent file?	Use PIP with /BR or /DI switch to check
Are you attempting to delete a locked file? (The command to delete is correct, and the file exists.)	Run PIP and UNlock
Are you attempting to access another user's file illegally?	Ask PIP to list the user's directory and see if an access error results

F.2.5 System Program Messages

System program messages are printed and the program continues. This class of error may be issued by a variety of system programs. If an ISSUER is specified, the error is unique to the indicated program. See the appropriate program manual for greater detail.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
SØØ1 FORTRN	Ø FORTRAN Compiler has exhausted symbol table space during the assembly phase of compilation.
S2ØØ	Ø Too many .CSECT directives.
S2Ø1	Ø Conditionals nested too deeply.
S2Ø2	Error Status Byte. Dev: file, ext. EOD or device error on .WRITE or .READ; the disk may have filled up.
S2Ø3	Relative address of error call Illegal switch, or too many switches, or illegal switch value, or switch value not given, or switch in output field.
S2Ø4	Relative address of error call Too many or too few output files.
S2Ø5	Ø Too many or too few input files.
S2Ø6	Relative address of error call No input files specified.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
S2Ø7	Error Status Byte EOD or device error on .TRAN.
S21Ø	Ø , dev:file.ext Unrecognized symbol table entry in indicated file.
S211	Ø , dev:file.ext An RLD of the given file references a global name which cannot be found in the symbol table.
S212	Ø , dev:file.ext An RLD of the given file contains a location counter modification command which is not last.
S213	Ø , dev:file.ext Object module does not start with a GSD in the indicated file.
S214	Ø , dev:file.ext The first entry in the module is not the module name of the indicated file.
S215	Ø , dev:file.ext An RLD of the given file references a section name which cannot be found.
S216	Ø The TRA specification references a nonexistent module name.
S217	Relative address at error call. Insufficient core.
S22Ø	Ø An internal jump table index is out of range.
S223	Ø No more room for CSI input buffer or Monitor's file manager routine, or Monitor's library search buffer.
S225	Ø Program too large or top too low (program has been linked below zero in memory).
S226	Ø An open angle bracket, <, is present in a line other than the first.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
S227	Error Code Illegal file combinations due to name conflicts. Defined error codes are: 1 No Primary File (PRI) output, 2 Secondary File (SEC) input = SEC output, 3 SEC input = PRI output, 4 PRI input = SEC output, 5 PRI input = SEC input, 6 PRI output = SEC output.
S23Ø	Error Status Byte Error on.BLOCK I/O.
S231	Illegal command, file-structured device required.
S232	No more than one action switch permitted.
S233	Specified UIC not found in MFD.
S234	Null filename of "*" given where filename required.
S235	No files found in UFD.
S236	Operation applicable to DECtape only.
S237	File not found during file recovery operation.
S24Ø	No space for file allocate.
S241	MFD is full.
S242	Meaningless command, no action taken.
S243	Ø An open angle bracket, < , is not present in the first line.
S244	Ø Already past requested position.
S245	Ø Object module not found, could be out of order.
S246	Ø Illegal library format.
S247	Ø Listing requested, but unable to read output library from specified output device.

CODE/ISSUER	ADDITIONAL INFORMATION/MEANING
S25Ø	Ø Core library symbol table not specified first or consecutively.
S251	Ø No files found for "*" request.
S252	Ø Filename given when none allowed.
S253	Ø Linker error.
S254	Ø It is illegal to zero the system resident disk.
S255	Ø Match found in third of later binary block in a paper tape library.
S256	Ø Illegal input device.
S257	File Block Error Code, dev:file.ext Illegal file operation. For example, protect code does not allow transfer of file; UIC different from Login UIC, thus making certain "wildcard" operations illegal. The operation in question is not performed.
S26Ø	Ø Same device needed for input and output in fast copy operation
S262	Ø Record size too big for buffer.
S263	File Number File record sizes do not agree on verify, "/V".
S264	Ø Conflict in standard file name extension which determines mode of transfer. Use explicits to resolve.
S265	Ø Operation attempted on device which is not legal for non-privileged user, for example, /PK PIP switch attempted by a user not logged in under [1,1].

APPENDIX G

LISTING OF SYSMAC.SML (SYSTEM MACRO FILE)

```
;	PDP-11 DOS SYSTEM MACROS	V005A

	.MACRO	.PARAM
R0=%^O0
R1=%^O1
R2=%^O2
R3=%^O3
R4=%^O4
R5=%^O5
R6=%^O6
R7=%^O7
SP=%^O6
PC=%^O7
PSW=^O177776
SWR=^O177570
	.ENDM
	.MACRO	.INIT	.LBLCK
	.MCALL	.AMODE
	.AMODE	.LBLCK
	EMT <^O6>
	.ENDM

	.MACRO	.RLSE	.LBLCK
	.MCALL	.AMODE
	.AMODE	.LBLCK
	EMT <^O7>
	.ENDM

	.MACRO	.CLOSE	.LBLCK
	.MCALL	.AMODE
	.AMODE	.LBLCK
	EMT <^O17>
	.ENDM

	.MACRO	.READ	.LBLCK,.LBUFF
	.MCALL	.AMODE
	.AMODE	.LBUFF
	.AMODE	.LBLCK
	EMT <^O4>
	.ENDM
```

```
.MACRO  .WRITE  .LBLCK,.LBUFF
.MCALL  .AMODE
.AMODE  .LBUFF
.AMODE  .LBLCK
EMT <^O2>
.ENDM

.MACRO  .OPENO  .LBLCK,.FBLCK
.MCALL  .CODE,.OPEN
.CODE   .FBLCK,<^O2>
.OPEN   .LBLCK,.FBLCK
.ENDM

.MACRO  .OPENI  .LBLCK,.FBLCK
.MCALL  .CODE,.OPEN
.CODE   .FBLCK,<^O4>
.OPEN   .LBLCK,.FBLCK
.ENDM

.MACRO  .OPENU  .LBLCK,.FBLCK
.MCALL  .CODE,.OPEN
.CODE   .FBLCK,<^O1>
.OPEN   .LBLCK,.FBLCK
.ENDM

.MACRO  .OPENC  .LBLCK,.FBLCK
.MCALL  .CODE,.OPEN
.CODE   .FBLCK,<^O13>
.OPEN   .LBLCK,.FBLCK
.ENDM

.MACRO  .OPENE  .LBLCK,.FBLCK
.MCALL  .CODE,.OPEN
.CODE   .FBLCK,<^O3>
.OPEN   .LBLCK,.FBLCK
.ENDM

.MACRO  .OPEN   .LBLCK,.FBLCK
.MCALL  .AMODE
.AMODE  .FBLCK
.AMODE  .LBLCK
EMT <^O16>
.ENDM

.MACRO  .WAIT   .LBLCK
.MCALL  .AMODE
.AMODE  .LBLCK
EMT <^O1>
.ENDM

.MACRO  .WAITR  .LBLCK,.ADDR
.MCALL  .AMODE
.AMODE  .ADDR
.AMODE  .LBLCK
EMT <^O0>
.ENDM

.MACRO  .BLOCK  .LBLCK,.BBLCK
.MCALL  .AMODE
.AMODE  .BBLCK
.AMODE  .LBLCK
EMT <^O11>
.ENDM
```

```
.MACRO  .TRAN   .LBLCK,.TBLCK
.MCALL  .AMODE
.AMODE  .TBLCK
.AMODE  .LBLCK
EMT <AC10>
.ENDM

.MACRO  .SPEC   .LBLCK,.SARG
.MCALL  .AMODE
.AMODE  .SARG
.AMODE  .LBLCK
EMT <AC12>
.ENDM

.MACRO  .STAT   .LBLCK
.MCALL  .AMODE
.AMODE  .LBLCK
EMT <AC13>
.ENDM

.MACRO  .ALLOC  .LBLCK,.FBLCK,.N
.MCALL  .AMODE
.AMODE  .N
.AMODE  .FBLCK
.AMODE  .LBLCK
EMT <AC15>
.ENDM

.MACRO  .DELET  .LBLCK,.FBLCK
.MCALL  .AMODE
.AMODE  .FBLCK
.AMODE  .LBLCK
EMT <AC21>
.ENDM

.MACRO  .RENAM  .LBLCK,.OFB,.NFB
.MCALL  .AMODE
.AMODE  .NFB
.AMODE  .OFB
.AMODE  .LBLCK
EMT <AC20>
.ENDM

.MACRO  .APPND  .LBLCK,.1FB,.2FB
.MCALL  .AMODE
.AMODE  .2FB
.AMODE  .1FB
.AMODE  .LBLCK
EMT <AC22>
.ENDM

.MACRO  .LOCK   .LBLCK,.FBLCK,.OP
.MCALL  .AMODE
.AMODE  .FBLCK
.IIF NB,.OP,CLR -(SP)
.AMODE  .LBLCK
EMT <AC14>
.ENDM

.MACRO  .KEEP   .LBLCK,.FBLCK
.MCALL  .AMODE
.AMODE  .FBLCK
.AMODE  .LBLCK
EMT <AC24>
.ENDM
```

```
.MACRO  .EXIT
EMT <^O60>
.ENDM

.MACRO  .TRAP   .STUS,.ADDR
.MCALL  .AMODE
.AMODE  .ADDR
.AMODE  .STUS
MOV     #^O1,-(SP)
EMT <^O41>
.ENDM

.MACRO  .STFPU  .STUS,.ADDR
.MCALL  .AMODE
.AMODE  .ADDR
.AMODE  .STUS
MOV     #^O3,-(SP)
EMT <^O41>
.ENDM

.MACRO  .RECRD  .LBLCK,.RBLCK
.MCALL  .AMODE
.AMODE  .RBLCK
.AMODE  .LBLCK
EMT <^O25>
.ENDM

.MACRO  .DUMP   .LOW,.HIGH,.CDE
.MCALL  .AMODE
.AMODE  .LOW
.AMODE  .HIGH
.AMODE  .CDE
EMT <^O64>
.ENDM

.MACRO  .RSTRT  .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
MOV     #^O2,-(SP)
EMT <^O41>
.ENDM

.MACRO  .CORE
MOV     #^O100,-(SP)
EMT <^O41>
.ENDM

.MACRO  .MONR
MOV     #^O101,-(SP)
EMT <^O41>
.ENDM

.MACRO  .MONF
MOV     #^O102,-(SP)
EMT <^O41>
.ENDM

.MACRO  .DATE
MOV     #^O103,-(SP)
EMT <^O41>
.ENDM
```

```
.MACRO  .TIME
MOV     #^O1Ø4,-(SP)
EMT <^O41>
.ENDM

.MACRO  .GTUIC
MOV     #^O1Ø5,-(SP)
EMT <^O41>
.ENDM

.MACRO  .SYSDV
MOV     #^O1Ø6,-(SP)
EMT <^O41>
.ENDM

.MACRO  .RADPK   .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
CLR     -(SP)
EMT <^O42>
.ENDM

.MACRO  .RADUP   .ADDR,.WRD
.MCALL  .AMODE
.AMODE  .WRD
.AMODE  .ADDR
MOV     #^O1,-(SP)
EMT <^O42>
.ENDM

.MACRO  .D2BIN   .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
MOV     #^O2,-(SP)
EMT <^O42>
.ENDM

.MACRO  .BIN2D   .ADDR,.WRD
.MCALL  .AMODE
.AMODE  .WRD
.AMODE  .ADDR
MOV     #^O3,-(SP)
EMT <^O42>
.ENDM

.MACRO  .O2BIN   .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
MOV     #^O4,-(SP)
EMT <^O42>
.ENDM

.MACRO  .BIN2O   .ADDR,.WRD
.MCALL  .AMODE
.AMODE  .WRD
.AMODE  .ADDR
MOV     #^O5,-(SP)
EMT <^O42>
.ENDM
```

```
.MACRO  .CST1   .CMDBF
.MCALL  .AMODE
.AMODE  .CMDBF
EMT <^O56>
.ENDM

.MACRO  .CST2   .CSBLK
.MCALL  .AMODE
.AMODE  .CSBLK
EMT <^O57>
.ENDM

.MACRO  .DTCVT  .ADDR
.MCALL  .CVTDT
.CVTDT  #^O0,.ADDR
.ENDM

.MACRO  .TMCVT  .ADDR
.MCALL  .CVTDT
.CVTDT  #^O1,.ADDR
.ENDM

.MACRO  .CVTDT  .CDE,.ADDR,.VAL1,.VAL2
.MCALL  .AMODE
.IF     NB,.VAL2
.AMODE  .VAL2
.ENDC
.IF     NB,.VAL1
.AMODE  .VAL1
.ENDC
.AMODE  .ADDR
.AMODE  .CDE
EMT <^O66>
.ENDM

.MACRO  .GTPLA
CLR     -(SP)
MOV     #^O5,-(SP)
EMT <^O41>
.ENDM

.MACRO  .STPLA  .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
MOV     #^O5,-(SP)
EMT <^O41>
.ENDM

.MACRO  .GTCIL
MOV     #^O111,-(SP)
EMT <^O41>
.ENDM

.MACRO  .GTSTK
CLR     -(SP)
MOV     #^O4,-(SP)
EMT <^O41>
.ENDM

.MACRO  .STSTK  .ADDR
.MCALL  .AMODE
.AMODE  .ADDR
```

```
        MOV     #^O4,-(SP)
        EMT <^O41>
        .ENDM

        .MACRO  .RUN    .RNBLK
        .MCALL  .AMODE
        .AMODE  .RNBLK
        EMT <^O65>
        .ENDM

        .MACRO  .FLUSH  .CDE
        .MCALL  .AMODE
        .AMODE  .CDE
        EMT <^O67>
        .ENDM

; THE MACRO .AMODE ACCEPTS ONE ARGUMENT AND
; AS A FUNCTION OF THE ADDRESSING MODE OF
; THE ARGUMENT GENERATES THE APPROPRIATE
; MOV TO -(SP).
; ADDRESS MODES THAT ARE TROUBLESOME (E.G.
; X(SP)) OR UNLIKELY (E.G. SP) WILL RESULT
; IN A .ERROR TO CMO INCLUDING THE
; VALUE OF THE ADDRESS MODE (E.G. X(SP)
; IS REPRESENTED AS 000066), THE ARGUMENT ITSELF
; AND THE TEXT "ADDRESSING MODE ILLEGAL AS SYSTEM
; MACRO ARGUMENT".
;
        .MACRO  .AMODE  .ARG
SP=%^O6
        .NTYPE  .SYM,.ARG               ;.SYM=ADDRESS MODE.

        .IF LE,.SYM-^O5
        MOV     .ARG,-(SP)              ;R0 TO R5
        .MEXIT
        .ENDC

        .IF EQ,.SYM&^O70-^O10
        .IF LE,.SYM&^O7-^O6
        MOV     .ARG,-(SP)              ;@R0 TO @R6
        .MEXIT
        .ENDC
        .ENDC

        .IF EQ,.SYM&^O60-^O20
        MOV     .ARG,-(SP)              ;[@](R0)+ TO [@](R7)+
        .MEXIT                          ; #N,@#ADDR
        .ENDC

        .IF EQ,.SYM&^O40-^O40
        .IF LE,.SYM&^O7-^O5
        MOV     .ARG,-(SP)              ;[@]-(R0) TO [@]-(R5)
        .MEXIT                          ;[@]X(R0) TO [@]X(R5)
        .ENDC
        .ENDC

        .IF EQ,.SYM&^O67-^O67
        MOV     .ARG,-(SP)              ;ADDR AND @ADDR
        .MEXIT
        .ENDC
```

```
        .ERROR .SYM                 ;'.ARG ADDRESSING MODE ILLEGAL
        .PRINT                      ;AS SYSTEM MACRO ARGUMENT'.
        .ENDM

; THE MACRO .CODE SETS UP THE FILEBLOCK
; WITH THE NEW OPEN CODE.
; THE ADDRESS OF THE FILEBLOCK MUST
; BE IN A REGISTER (R0 TO R5)

        .MACRO  .CODE   .FBLK,.N
        .NTYPE  .SYM,'.FBLK

        .IF LE,.SYM-^O5
        MOVB    #'.N,-^O2('.FBLK) ;R0 TO R5
        .MEXIT
        .ENDC

        .ERROR '.SYM                ;'.FBLK ADDRESSING MODE ILLEGAL
        .PRINT                      ;FOR '.OPEN FILE BLOCK
        .ENDM
```

APPENDIX H
PERIPHERAL DEVICES

H.1 OPERATING THE TELETYPE

The ASR-33 Teletype is the basic input/output device for PDP-11 computers. It consists of a printer, keyboard, paper tape reader, and paper tape punch, all of which can be used either on-line under program control or off-line. The Teletype controls (Figure H-1) are described as they apply to the operation of the computer.

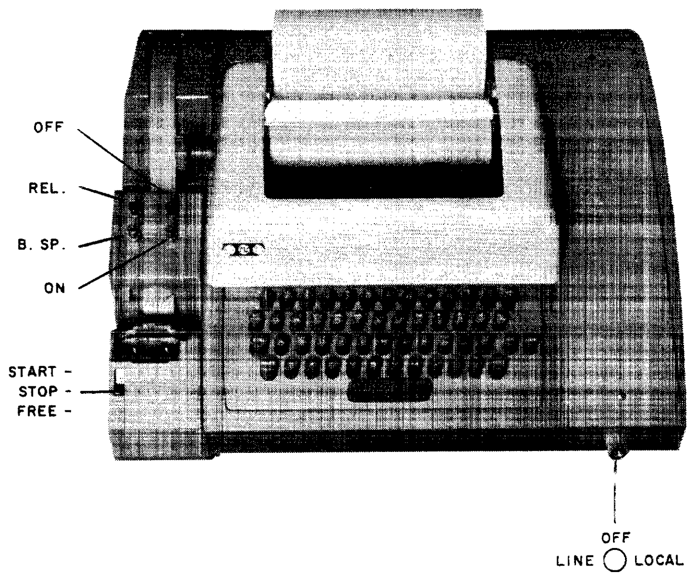


Figure H-1 ASR-33 Teletype Console

H.1.1 Power Controls

LINE - The Teletype is energized and connected to the computer as an input/output device, under computer control.

OFF - The Teletype is de-energized.

LOCAL - The Teletype is energized for off-line operation.

H.1.2 Printer

The printer provides a typed copy of input and output at 10 characters per second, maximum.

H.1.3 Keyboard

The Teletype keyboard is similar to a typewriter keyboard. However, certain operational functions are shown on the upper part of some of the keytops. These functions are activated by holding down the CTRL key while depressing the desired key. For example, when using the Text Editor, CTRL/U causes the current line of text to be ignored.

Although the left and right square brackets are not visible on the keyboard keytops, they are shown in Figure H-2 and are generated by typing SHIFT/K and SHIFT/M, respectively. The ALT MODE key is identified as ESC (ESCape) on some keyboards.

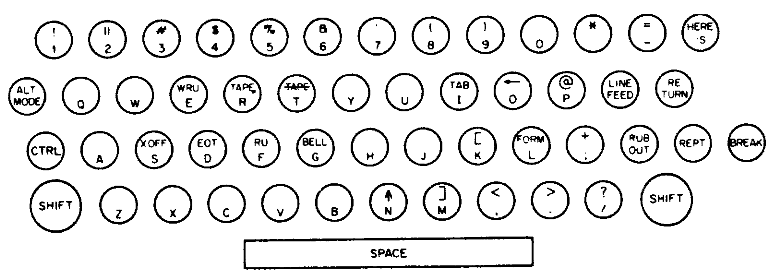


Figure H-2 ASR-33 Teletype Keyboard

H.1.4. Paper Tape Reader

The paper tape reader is used to read data punched on eight-channel perforated paper tape at a rate of 10 characters per second, maximum. The reader controls are shown in Figure H-1 and described below.

START	Activates the reader; reader sprocket wheel is engaged and operative.
STOP	Deactivates the reader; reader sprocket wheel is engaged but not operative.
FREE	Deactivates the reader; reader sprocket wheel is disengaged.

The following procedure describes how to properly position paper tape in the low-speed reader.

a. Raise the tape retainer cover.

b. Set reader control to FREE.

c. Position the leader portion of the tape over the read pens with the sprocket (feed) holes over the sprocket (feed) wheel and with the arrow on the tape (printed or cut) pointing outward.

d. Close the tape retainer cover.

e. Make sure that the tape moves freely.

f. Set reader control to START, and the tape will be read.

H.1.5 Paper Tape Punch

The paper tape punch is used to perforate eight-channel rolled oiled paper tape at a maximum rate of 10 characters per second. The punch controls are shown in Figure H-1 and described below.

RELease	Disengages the tape to allow tape removal or loading.
B.SP	Backspaces the tape one space for each firm depression of the B.SP button.
ON (LOCK ON)	Activates the punch.
OFF (UNLOCK)	Deactivates the punch.

Blank leader/trailer tape is generated by:

1. Turning the TTY switch to LOCAL
2. Turning the low speed punch on (depress ON button)
3. Typing the HERE IS key
4. Turning the low speed punch off (depress OFF button)
5. Turning the TTY switch to LINE.

H.2 OPERATING THE HIGH-SPEED PAPER TAPE READER AND PUNCH UNITS

A high-speed paper tape reader and punch unit is pictured in Figure H-3 and descriptions of the reader and punch units follow.

H.2.1 Reader Unit

The high-speed paper tape reader is used to read data from eight-channel fan-folded (non-oiled) perforated paper tape photoelectrically at a maximum rate of 300 characters per second. Primary power is applied to the reader when the computer POWER switch is turned on. The reader is under program control. However, tape can be advanced past the photoelectric sensors without causing input by pressing the reader FEED button.

H.2.2 Punch Unit

The high-speed paper tape punch is used to record computer output on eight-channel fan-folded paper tape at a maximum rate of 50 characters per second. All characters are punched under program control from the computer. Blank tape (feed holes only, no data) may be produced by pressing the FEED button. Primary power is available to the punch when the computer POWER switch is turned on.

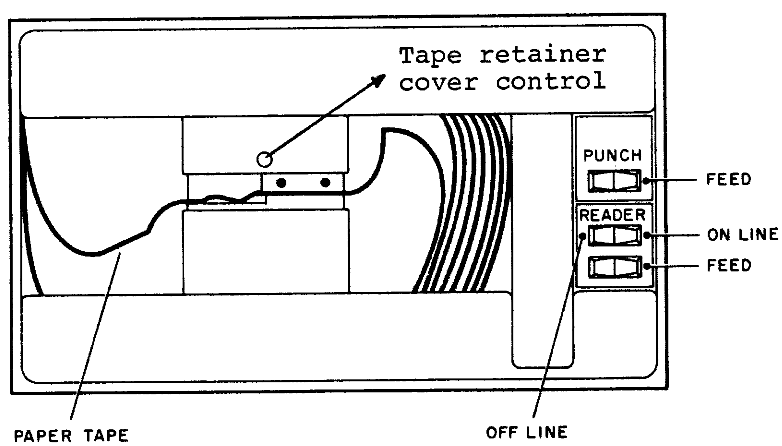


Figure H-3 High-Speed Paper Tape Reader/Punch

Paper tape is loaded into the reader as explained below.

1. Raise tape retainer cover.
2. Put tape into right-hand bin with channel one of the tape toward the rear of the bin.
3. Place several folds of blank tape through the reader and into the left-hand bin.

4. Place the tape over the reader head with feed holes engaged in the teeth of the sprocket wheel.
5. Close the tape retainer cover.
6. Depress the tape feed button until the leader tape is over the reader head.

CAUTION

Oiled paper tape should not be used in the high-speed reader or punch - oil collects dust and dirt which can cause reader or punch errors.

H.3 THE LP11 LINE PRINTER

The LP11 is a line printer with 80 column capacity, capable of printing more than 300 lines per minute at a full 80 columns, and more than 1100 lines per minute at 20 columns. The print rate is dependent upon the data and the number of columns to be printed.

Characters are loaded into the printer memory via the Line Printer Buffer (LPB) serially. When the memory becomes full (20 characters) the characters are automatically printed. This continues until the 80 columns have been printed or a carriage return, line feed, or form feed character is recognized.

H.3.1 Printer Control Panel

Figure H-4 illustrates the printer control panel on which are mounted three indicator lights and three toggle switches.

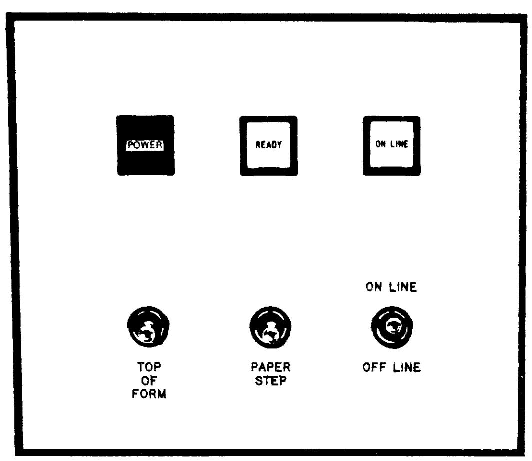


Figure H-4 Line Printer Control Panel

Operation of the lights and switches is as follows:

POWER light	Glows red to indicate main power switch (located inside cabinet) is at ON position and power is available to the printer.
READY light	Glows white, shortly after the POWER light goes on to indicate that internal components have reached synchronous state and the printer is ready to operate.
ON LINE light	Glows white to indicate that ON LINE/ OFF LINE toggle switch is in ON LINE position.
TOP OF FORM switch	This switch is tipped toward the front of the cabinet to roll up the form to the top of the succeeding page. It is spring-returned to center position, and produces a single top-of-form operation each time it is actuated. The switch is effective only when the printer is off line.
PAPER STEP switch	Operates similarly to TOP OF FORM but produces a single line step each time it is actuated. It is only effective with printer off line.
ON LINE/OFF LINE switch	This two-position toggle switch is spring-returned to center. When momentarily positioned at ON LINE it logically connects the printer to the computer and causes the ON LINE light to glow. Positioned momentarily at OFF LINE, the logical connection to the computer is broken, the ON LINE light goes off, and the TOP OF FORM and PAPER STEP switches are enabled.

H.3.2 Maintenance Panel

The maintenance panel contains controls used for the line printer's initial set-up and maintenance. It is accessible only by opening the front cabinet door, located beneath the control panel.

This panel contains three switches, and three indicators.

1. Main AC power switch;
2. PRINT INHIBIT switch - must be off (down) to enable printing;

3. DRUM GATE indicator - if lit, drum gate not properly locked;
4. PAPER FAULT - if lit, check for no paper, or torn paper;
5. PRINT INHIBIT indicator - if lit, turn PRINT INHIBIT switch off;
6. MASTER CLEAR switch - spring-loaded to off (down); if toggled to on (up), resets printer logic, turns off READY and ONLINE indicators.

H.3.3 Adjustment Controls

Controls are provided as listed in Table H-1.

Table H-1 Adjustment Controls

Control	Location	Function
Drum gate latch	Gearshift type knob near right-hand side of maintenance panel.	Unlocks drum gate which can then be swung open for access to components on back.
Tractor paper width adjustment	Setscrew at far right of tractor pressure plate behind drum gate.	Adjusts right tractor for various paper widths; left tractor is factory adjusted.
Tractor horizontal tension adjustment	Next to left side of tractor paper width adjustment.	Adjusts horizontal tension of paper.
COPIES CONTROL lever	Extreme upper right-hand corner of cabinet just above drum gate hinge.	Adjusts the distance between hammer bank and character drum for different numbers of printed copies. Settings are: 1-2, 3-4 and 5-6.
Paper vertical adjustment control	Knob at upper left of cabinet, directly above right-hand side of maintenance panel	Adjusts vertical alignment of printing so that it prints on lined paper. Can be adjusted to plus or minus one line and may be adjusted while the printer is in operation.
Top-of-form indicators	Red arrows visible when drum gate is swung open one on each side of paper directly below tractor pressure plates	Aligns paper during loading.

H.3.4 Loading Paper

Follow the steps listed below to load paper into the printer.

Step	Procedure
1.	Open front door of cabinet to gain access to maintenance panel and turn main AC power switch on. Verify that control panel POWER indicator lights.
2.	Lift control panel TOP OF FORM switch and release to move tractors to correct loading position.
3.	Open the drum gate by moving the drum gate latch knob to the left and up. Swing drum gate open.
4.	Adjust right-hand tractor paper width adjustment for proper paper width. This is accomplished by loosening the set screw on the 8Ø-column model or by using the easy release mechanism on the 12Ø column model. Make certain that the right-hand tractor is tightened in place after it is adjusted.
5.	Open spring-loaded pressure plates on both tractors.
6.	Load paper so that a perforation is pointed to by the two red arrows (top-of-form indicators). Paper should lie smoothly between tractors without wrinkling or tearing the feed holes.
7.	Close spring-loaded pressure plates on both tractors.
8.	Adjust the COPIES CONTROL lever to the proper number for the number of copies to be made. For example, set to 1-2 for single forms, set to 5-6 for six-part forms.
9.	Close drum gate and lock into position with drum gate latch. After approximately 1Ø seconds the control panel READY indicator should light. If it does not, check to see if any error is indicated. An error is indicated if one of the following lights is on: DRUM GATE, PAPER FAULT, or PRINT INHIBIT.
10.	Lift TOP OF FORM switch several times to ensure paper is feeding properly.
11.	Set system to on-line mode by lifting ON LINE/OFF LINE switch and verifying that ON LINE indicator lights. At this point, printed matter can be aligned with the paper lines by rotating the paper vertical adjustment knob.

For further details on the LP11, refer to the LP11 Line Printer Manual, DEC-11-ODLPA-A-D.

H.4 THE TU10 MAGTAPE DRIVE

The TU10 is a magnetic tape drive which may be a 7- or 9-track unit and which will record data in densities of 200, 556 or 800 bits per inch.

Figure H-5 shows the magnetic tape drive control panel and its schematic representation. Table H-2 shows the meaning assigned to each indicator light and Table H-3 explains the function of each switch.

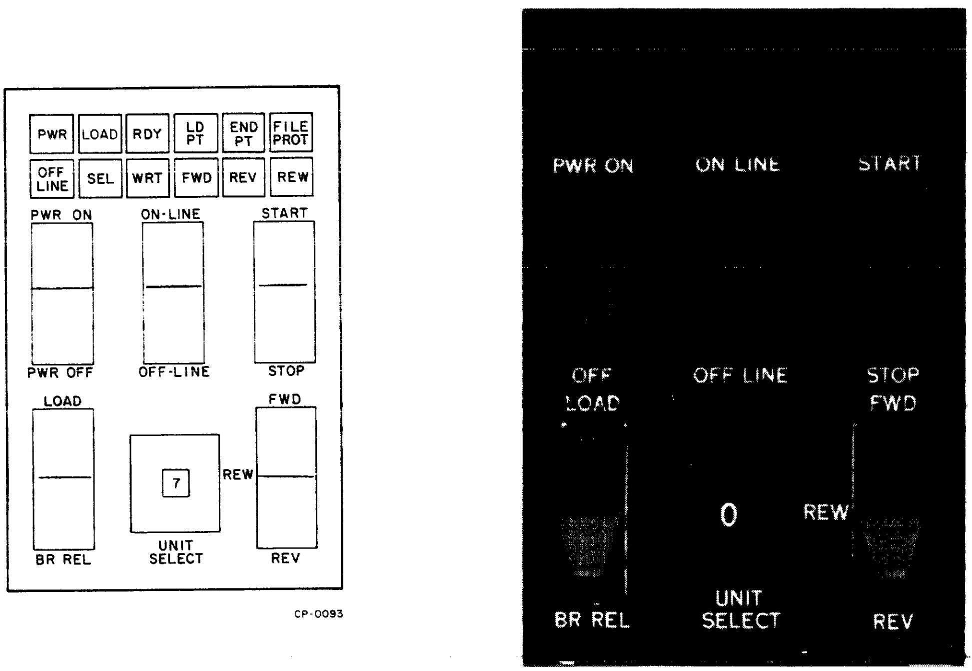


Figure H-5 Magnetic Tape Drive Control Panel

Table H-2
Status Indicators

Indicator	Procedure
PWR	Indicates that power is being supplied to the drive unit.
OFF-LINE	Indicates local operation by the control box.
LOAD	Indicates that the vacuum system has been enabled and the unit is prepared to accept on-line or off-line commands.
SEL	Indicates the tape transport has been selected by the controller (program).
RDY	Indicates that the drive is ready to accept requests for operation (provided the SEL light is also lit).
WRT	Indicates that the program has initiated a write operation in the tape transport.
LD PT	Indicates that the tape mounted on this unit is at its Load point (BOT marker is being sensed). REW command is disabled.
FWD	Indicates that a forward command has been issued.
END PT	Indicates that the tape mounted on this unit is at its end point (EOT marker is being sensed). FWD command is disabled.
REV	Indicates that a reverse command has been issued.
FILE PROT	Indicates that the tape may not be written on (No Write ring in tape reel).
REW	Indicates that a rewind command has been issued.

Table H-3
Switch Functions

Switch	Function
PWR ON/OFF	Controls power to the drive.
ONLINE/OFFLINE	Transfers drive control to processor (ON LINE) or enables local control box control by operator (OFF LINE).
START/STOP	Initiates or terminates tape movement.
LOAD/BR REL	LOAD position causes tape to be drawn into vacuum columns. Center position applies reel motion brakes. BR REL position releases reel motion brakes.
UNIT SELECT	Assigns a logical unit number (zero through seven) to this drive.
FWD/REW/REV	Selects tape motion direction to be controlled by START/STOP switch. FWD position indicates transfer to take-up reel until EOT (end of tape) marker is sensed, REV position indicates transfer to file reel until BOT (beginning of tape) marker is sensed, REW position indicates transfer as in REV at a higher tape speed; when the tape stops at BOT, depressing the start switch again causes tape to unload.

H.4.1 Operating Procedures

H.4.1.1 Loading and Threading Tape - Use the following procedure to mount and thread the tape:

Step	Procedure
1	Apply power to the transport by depressing PWR ON switch.
2	Ensure the LOAD/BR REL switch is in the center position (this applies the brakes).
3	Place a write enable ring in the groove on the file reel if data is to be written on the tape. Ensure there is no ring in the groove if data on the tape is not to be erased or written over.
4	Mount the file reel onto the lower hub with the groove facing towards the back. Ensure that the reel is firmly seated against the flange of the hub.
5	Install the take-up reel (top) as described in Step 4.
6	Place LOAD/BR REL switch to the BR REL position.
7	Unwind tape from the file reel and thread the tape over the tape guides and head assembly as shown in Figure H-6.
8	Wind about five turns of tape onto the take-up reel.
9	Set the LOAD/BR REL switch to the LOAD position to draw tape into the vacuum columns.
10	Select FWD and press START to advance the tape to Load Point. When the BOT marker is sensed, tape motion stops, the FWD indicator goes out, and the LOAD PT indicator comes on.

NOTE

If tape motion continues for more than 10 seconds, press STOP, select REV (reverse) and press START. The tape should move to the BOT marker (Load Point) before stopping.

H.4.1.2 Unloading Tape - To unload the tape proceed as follows:

Step	Procedure
1	Press OFF-LINE switch if the transport has been operating in the on-line mode.
2	Press STOP switch and select REW.
3	Press START switch. The tape should rewind until the BOT marker is reached.
4	Press the LOAD/BR REL switch to release the brakes.
5	Gently hand wind the file reel in a counterclockwise direction until all of the tape is wound onto the reel.

CAUTION

When handwinding the tape, do not jerk the reel. This can stretch or compress the tape which could cause irreparable damage.

6	Remove the file reel from the hub assembly.

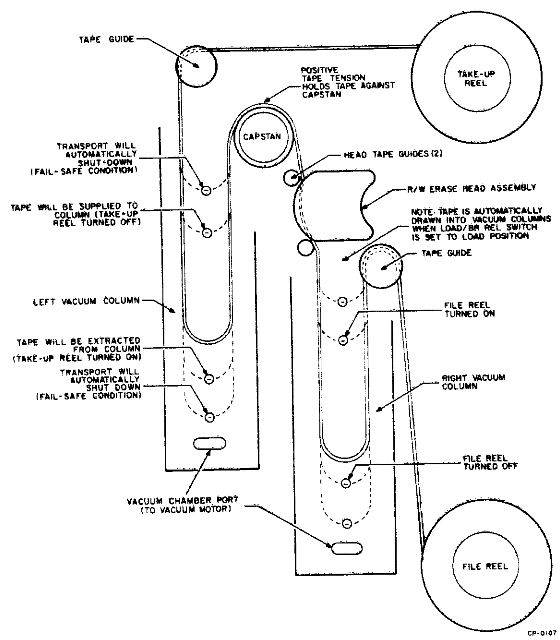


Figure H-6 Tape Transport Mechanism

H.4.1.3 Restart After Power Failure - In the event of a power failure, the DECmagtape automatically shuts down and tape motion stops without damage to the tape. Return of power is indicated when the PWR indicator lights. To restart the transport proceed as follows:

Step	Procedure
1	Press the LOAD/BR REL switch to release the brakes.
2	Manually wind the reels to take up any slack in the tape.
3	Set the LOAD/BR REL switch to the LOAD position to draw tape into the vacuum columns.
4	Set ON-LINE/OFF-LINE switch to the desired position and continue operation.

H.4.1.4 Restart After Fail-Safe - If the tape loop in either buffer column exceeds the limits shown in Figure H-6, the vacuum system automatically shuts down and tape motion stops without damage to the tape. When this fail-safe condition occurs, the DECmagtape does not respond to on-line or off-line commands. To restart the transport, perform Steps 1 through 4 in Paragraph H.4.1.3.

H.4.1.5 - Tape Handling - Observe the following precautions when handling magnetic tape:

a. Always handle a tape reel by the hub hole; squeezing the reel flanges can cause damage to the tape edges when winding or unwinding tape.

b. Never touch the portion of tape between the BOT and EOT markers. Oils from fingers attract dust and dirt. Do not allow the end of the tape to drag on the floor.

c. Never use a contaminated reel of tape. This spreads dirt to clean tape reels and can affect tape transport operation.

d. Always store tape reels inside their containers. Keep empty containers closed so dust and dirt cannot get inside.

e. Inspect tapes, reels, and containers for dust and dirt. Replace take-up reels that are old or damaged.

f. Do not smoke near the transport or tape storage area. Tobacco smoke and ash are especially damaging to tape.

g. Do not place the DECmagtape near a line printer or other device that produces paper dust.

h. Clean the tape path frequently as described in Paragraph 5.2.1.

H.5 THE TC11 DECTAPE DRIVE

Figure H-7 pictures the TC11 DECtape drive unit. Table H-4 shows the meaning of each indicator lamp and Table H-5 shows the function of each switch.

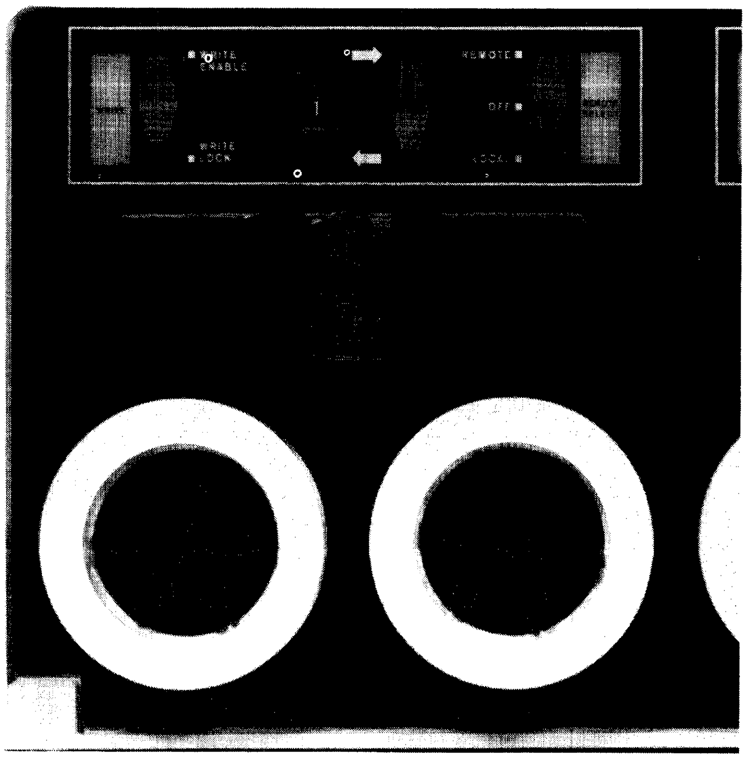


Figure H-7 TC11 DECtape Drive

To mount a DECtape on the TC11:

1. Move the LOCAL/REMOTE/OFF switch to the OFF position.
2. Mount a DECtape by centering it over the left band hub and pushing it firmly onto the spring loaded hub.
3. Wind sufficient tape to wrap around the recording head guides and the empty DECtape reel which should be mounted on the right hand hub.
4. Take up a few inches to tape on the right hand hub by hand.
5. Move the LOCAL/REMOTE/OFF switch to LOCAL position.
6. Depress the DECtape motion switch to the LOAD position until about 6 feet of tape are on the right hand hub.

7. Depress the WRITE PROTECT switch or write enable as appropriate.

8. Assure that the unit number showing for this drive does not show on any other drive.

9. Move the LOCAL/OFF/REMOTE switch to the remote position.

To dismount a DECtape from the TC11:

1. Move the LOCAL/OFF/REMOTE switch to the LOCAL position.

2. Depress the tape motion switch in the rewind direction (←) until all the tape is on the left hand reel.

3. Move LOCAL/OFF/REMOTE switch to OFF position.

4. Pull the DECtape reel from the left hand hub.

APPENDIX I
COMMAND STRING INTERPRETER

Also 7.4 of
RSX-11C

I.1 SYSTEM PROGRAM/USER PROGRAM COMMAND STRINGS

There is a single, general format for all system program command strings. All system programs use it, and any user program may also do so. These command strings are all processed by a Monitor routine, the Command String Interpreter (CSI) which is in Section 3.8.6. Any program expecting such a command first types # on the console to indicate the fact to the operator. The general format is

ds-spec [,[ds-spec] ...] ... [<ds-spec] [,[ds-spec]] ...

where "ds-spec" represents a dataset specifier (described in the next section), brackets indicate optional items, and elipsis (...) indicates that the preceding item may appear zero or more times. Items preceding the < (if any) describe output datasets; those which follow describe input datasets.

I.2 CSI COMMAND FORMAT

Whenever a system program requests input through the CSI, a # will be printed on the teleprinter (exception, ODT-11R prints an *) and the program will wait for the operator's reply. A CSI command may consist of one or more output dataset specifications, followed by <, followed by one or more input dataset specifications. Spaces, horizontal TABs, and nulls may appear anywhere in the string and are ignored. A command is terminated by typing the RETURN key, which causes both carriage return and line feed characters to be passed to the program. The line-feed character terminates the input. < need not occur. If it does, at least one input file specification must appear. Only one < per command is allowed. <u>Commands can not be continued from line to line.</u>

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

Dataset Specification,,Dataset Specification

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

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

dev:filnam.ext[uic]/sw$_1$:v$_1$:...:v$_n$/sw$_2$:v$_1$:...:v$_n$,

where: dev = The device specification consisting of two or three letters (and often an octal digit) terminated by a colon. The letters identify the device and the digit identifies the unit. Units must be given in octal. The colon delimits this field with one exception; only physical names as listed in Appendix A may be specified. For example, DTA1: is the correct specification for DECtape, controller A, unit 1. The exception is SY: which is a generic name for the system residence device (e.g., on an RK system SY: is equivalent to DK:). If no digit appears, unit 0 is assumed. If the device specification itself does not appear, the device is assumed to be the device last specified, on the current side of the <, if there is one; otherwise, the system disk (SY:) unit 0 is assumed.

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

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

The file name need not appear if the device is not file-structured or if the program can supply a name.

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

The extension need not appear.

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

*.EXT specifies all files with extension .EXT,

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

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

NOTE

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

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

[*,136] specifies all users whose number is 136
[12,*] specifies all members of user group 12, and
[*,*] specifies all users.

The user identification code need not appear, in which case the default is the identification entered with the LOGIN command.

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

For examples: /DATE:12.20.69 might be a switch to enter December 20, 1969 in a date field.

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

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

/S and /SWITCH2 are both legal.

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

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

Table I-1 summarizes the legal command syntax.

Table I-1

.CSI Command String Syntax Rules

Item Which Last Appeared	Item Immediately Following								
	,	DEV:	FILNAM	.EXT	UIC	/SWITCH	<	Terminator	*
blank[1]	*	*	*	E	*	*	*	*	*
,	*	*	*	E	*	*	*	*	*
DEV:	*	E	*	E	*	*	*	*	*
FILNAM	*	E	E	*	*	*	*	*	E[2]
.EXT	*	E	E	E	*	*	*	*	E
UIC	*	E	E	E	E	*	*	*	E
/SWITCH	*	E	E	E	E	*	*	*	E
<	*	*	*	E	*	*	E	E	*

Legend: E indicates error. * indicates legal.

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

[2] .* is legal following FILNAM.

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. Note that a /SWITCH specification is always legal even alone. In such a case, the system device SY: and a null filename are assumed.

I.3 CSI COMMAND EXAMPLE

An example of a complete command is:

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

which is interpreted as explained below.

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

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

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

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

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

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

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

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

APPENDIX J
SPECIAL I/O FUNCTIONS

Certain I/O functions are sufficiently device-dependent that they are beyond the scope of the File System. The .SPEC request (see Section 3.6.12) is provided as a means of accommodating such functions. A special function request requires one argument, which must be either a code in the range 0-255 or a pointer to a special function block. When a special function block is used, it must contain a code.

In general, special function codes will have similar meanings from device to device. When a code has no meaning for a device, it is treated as a no-op. Currently, special functions are defined only for magtape.

J.1 MAGTAPE FUNCTIONS

J.1.1 Special Function Block

The magtape driver requires a special function block to perform the special function requests. The following is the calling sequence for magtape special functions and the special function block format:

```
               .SPEC   #LNKBLK, #SFBLK
               .
               .
               .
    SFBLK:     .BYTE   Special function code
               .BYTE   Words to follow (must be 3 or larger)
               .WORD   Tape unit status (returned by driver)
               .WORD   User specified count or control information
               .WORD   Residue count (returned by driver)
```

J.1.2 Functions

Code	Function
1	Offline (rewind and unload)
2	Write End-of-File
3	Rewind
4	Skip Record(s)
5	Backspace Record(s)
6	Set Density and Parity
7	Obtain Status

J.1.2.1 OFFLINE (Rewind and Unload) - function Code 1

This request causes the magtape to be rewound to the beginning-of-tape (BOT) marker and SELECT REMOTE status to go off. If the last command to the driver for this device was a WRITE, three EOF's are written before rewinding. Thus, this function could cause data to be lost if it is issued before a CLOSE during READ/WRITE processing.

J.1.2.2 WRITE END-OF-FILE - function Code 2

This request writes an end-of-file (EOF) record on magtape. It may cause data to be lost as described under OFFLINE.

J.1.2.3 REWIND - function Code 3

The REWIND request performs the same function as OFFLINE except that the SELECT REMOTE status does not go off.

J.1.2.4 SKIP RECORD(S) - function Code 4

Skips forward over the requested number of records (SFBLK+4) until either the SKIP count is exhausted or until an EOF record is encountered, in which case the EOF is spaced over and counted, but the operation terminates and a residue count (SFBLK+6) is returned (if any).

J.1.2.5 BACKSPACE RECORD(S) - function Code 5

This request skips backwards over the requested number of records until either the SKIP count is exhausted or an EOF or the BOT marker is encountered. If an EOF is encountered it is spaced over and counted, but the operation terminates and a residue count is returned (if any). If the BOT marker is encountered, it is not skipped or counted. Instead, the operation is terminated and a residue count is returned.

J.1.2.6 SET DENSITY AND PARITY - function Code 6

This request is ignored for 9-track tapes; it sets density and parity as follows for 7-track tapes:

DENSITY (SFBLK+5)	PARITY (SFBLK+4)
ø = 2øø BPI	ø = ODD
1 = 556 BPI	1 = EVEN
2 = 8øø BPI	
3 = 800 BPI Dump Mode	

The default density and parity are 8øø BPI Dump Mode, ODD. In this mode, one byte from core is represented as two bytes on 7-track magtape. Changing from this default causes one byte from core to be represented by one byte on tape with a loss of the two high order bits (6-7) of the byte.

J.1.2.7 TAPE UNIT STATUS - function Code 7

This request returns the current status of the tape unit in SFBLK+2 in the following form:

Bits	Content
ø - 2	Last command was: ø = OFFLINE 1 = READ 2 = WRITE 3 = WRITE EOF 4 = REWIND 5 = SKIP RECORD 6 = BACKSPACE RECORD
3 - 6	Unused.
7	1 = TAPE AFTER EOF (BEFORE EOF IF LAST COMMAND WAS BACKSPACE)
8	1 = TAPE AT BOT MARKER
9	1 = TAPE AFTER EOT MARKER
1ø	1 = WRITE LOCK ON
11	PARITY: ø = ODD 1 = EVEN (DEFAULT = ODD)
12	ø = 9 TRACK 1 = 7 TRACK
13 - 14	DENSITY: ø = 2øø BPI 1 = 556 BPI 2 = 8øø BPI 3 = 8øø BPI DUMP MODE
15	1 = LAST COMMAND CAUSED ERROR

Tape unit status is returned in SFBLK+2 for all special functions.

APPENDIX K
PROGRAMS

K.1 The two following example program listings illustrate methods for utilizing DOS monitor services. Note that the assembly language expansions of the programmed requests are used. Users with less than 12K of core should code their programs as illustrated and assemble the resultant code with the 8K assembler. Users with 12K of core or more may replace the assembly language expansion code with appropriate programmed requests and assemble with MACRO-11.

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

Example Program #2

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

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

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

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

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

APPENDIX M
CHARACTER CODES

N.1 CARD CODES

CARD CODES
(ANSI X3.26–1970)

Zone / Digit	12	11	0		12 0	12 11	11 0		12 9	11 9	0 9	9	12 0 9	12 11 9
	&	–	0	space	{	|	}							
1	A	J	/	1	a	j	~		SOH	DC1				
2	B	K	S	2	b	k	s		STX	DC2		SYN		
3	C	L	T	3	c	l	t		ETX	DC3				
4	D	M	U	4	d	m	u							
5	E	N	V	5	e	n	v		HT		LF			
6	F	O	W	6	f	o	w			BS	ETB			
7	G	P	X	7	g	p	x		DEL		ESC	EOT		
8	H	Q	Y	8	h	q	y			CAN				
9	I	R	Z	9	i	r	z							
8-1				grave						EM			NUL	DLE
8-2	[	]	\	:										
8-3	.	$	,	#					VT					
8-4	<	*	%	@					FF	FS		DC4		
8-5	(	)	_	'					CR	GS	ENQ	NAK		
8-6	+	;	>	=					SO	RS	ACK			
8-7	!	^	?	"					SI	US	BEL	SUB		

NOTES

To determine the card punch for a particular character, locate the character in the table and read the corresponding zone punch and then digit punch. For example, the card punch for a % is 0-8-4.

To obtain the character corresponding to a particular card punch, locate the junction of the zone punch and the digit punch. For example, the character corresponding to the card punch 12-11-9 is r.

Slots that do not contain characters represent card punches for which there are no ASCII equivalents.

M.2 ASCII CHARACTER SET

ASCII CHARACTER SET
ASCII–1968 (ANSI X3.4–1968)

To obtain octal or decimal ASCII representation of a character, add the row value to the column value.

Column Value / Row Value	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120	decimal ASCII
	000	010	020	030	040	050	060	070	100	110	120	130	140	150	160	170	octal ASCII
0	NUL	BS	DLE	CAN	space	(	0	8	@	H	P	X	grave	h	p	x	
1	SOH	HT	DC1	EM	!	)	1	9	A	I	Q	Y	a	i	q	y	
2	STX	LF	DC2	SUB	”	*	2	:	B	J	R	Z	b	j	r	z	
3	ETX	VT	DC3	ESC	#	+	3	;	C	K	S	[	c	k	s	{	
4	EOT	FF	DC4	FS	$	,	4	<	D	L	T	\	d	l	t	|	
5	ENQ	CR	NAK	GS	%	-	5	=	E	M	U	]	e	m	u	}	
6	ACK	SO	SYN	RS	&	.	6	>	F	N	V	^ (↑)	f	n	v	~ (ESC)	
7	BEL	SI	ETB	US	’ apos	/	7	?	G	O	W	_ (←)	g	o	w	DEL	

Differences in the ASCII Standard

Octal	(ASCII 1963)	ASCII 1968
136	↑	^ (circumflex)
137	←	_ (underline)
176	ESC	~

NUL	NULL
SOH	START OF HEADING (↑A)
STX	START OF TEXT (↑B)
ETX	END OF TEXT (↑C)
EOT	END OF TRANSMISSION (↑D)
ENQ	ENQUIRY (↑E)
ACK	ACKNOWLEDGE (↑F)
BEL	BELL (↑G)
BS	BACKSPACE (↑H)
HT	HORIZ. TABULATION (↑I)
LF	LINE FEED (↑J)
VT	VERT. TABULATION (↑K)
FF	FORM FEED (↑L)
CR	CARRIAGE RETURN (↑M)
SO	SHIFT OUT (↑N)
SI	SHIFT IN (↑O)
DLE	DATA LINK ESCAPE (↑P)
DC1	DEVICE CONTROL 1 (↑Q)
DC2	DEVICE CONTROL 2 (↑R)
DC3	DEVICE CONTROL 3 (↑S)
DC4	DEVICE CONTROL 4 (STOP) (↑T)
NAK	NEGATIVE ACKNOWLEDGE (↑U)
SYN	SYNCHRONOUS IDLE (↑V)
ETB	END OF TRANSMISSION BLOCK (↑W)
CAN	CANCEL (↑X)
EM	END OF MEDIUM (↑Y)
SUB	SUBSTITUTE (↑Z)
ESC	ESCAPE (↑[)
FS	FILE SEPARATOR (↑\)
GS	GROUP SEPARATOR (↑])
RS	RECORD SEPARATOR (↑↑)
US	UNIT SEPARATOR (↑←)
DEL	DELETE (RUBOUT)

The ↑x character is produced by depressing the CTRL key and at the same time depressing the x character key.

NOTES

1. Teleprinters manufactured by Teletype Corporation, Skokie, Illinois, have used codes 175 (ALT) and 176 for ESC. Programs may forgo the use of } (175) and ~ (176) in order to use these codes as ESC on older teleprinters.

2. ASCII is a seven bit character code with an optional odd parity bit (200) added for many devices. Programs normally use just seven bits internally; the 200 bit is either stripped or added so the program will operate with either parity or non-parity generating devices.

   ISO Recommendation R646 and CCITT Recommendation V.3 (International Alphabet No. 5) is identical to ASCII except that number sign (043) is represented as £ instead of # and certain characters are reserved for national use.

APPENDIX N

GLOSSARY AND ABBREVIATIONS

ABS	Absolute
A/D	Analog-to-digital
ADC	Add Carry
ADRS	Address
ASCII	American Standard Code for Information Interchange
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right Automatic Send/Receive
B	Byte
BAR	Bus Address Register
BBSY	Bus Busy
BCC	Branch if carry clear
BCS	Branch if carry set
BEQ	Branch if equal
BG	Bus Grant
BGE	Branch if greater or equal
BGT	Branch if greater than
BHI	Branch if higher
BHIS	Branch if higher or same
BIC	Bit Clear
BIS	Bit Set
BIT	Bit Test
Bit Map	A table describing the availability of space. Each bit in the table indicates the state (occupied or free) of one segment of storage, for example a block on a bulk storage device.
BLE	Branch if less or equal
BLOS	Branch if lower or same
BLT	Branch if less than
BMI	Branch if minus
BNE	Branch if not equal
BPL	Branch if plus
BR	Branch

BRD	Bus Register Data
BRX	Bus Request
BSP	Back Space
BSR	Bus Shift Register Back Space Record
BSY	Busy
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.
BVC	Branch if overflow clear
BVS	Branch if overflow set
CBR	Console Bus Request
CIL	Core Image Library
CILUS	Core Image Library Update & Save Program
CLC	Clear Carry
CLK	Clock
CLN	Clear Negative
CLR	Clear
CLV	Clear Overflow
CLZ	Clear Zero
CMP	Compare
CNPR	Console Nonprocessor Request
CNTL	Control
COM	Complement
COND	Condition
CONS	Console
CONT	Contents Continue
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.
CP	Central Processor
CSI	Command String Interpreter
CSR	Control and Status Register

D	Data
D/A	Digital-to-analog
DAR	Device Address Register
DAT	Device Assignment Table. Contains the specifications from ASSIGN commands.
Dataset	A logical collection of data which is treated as an entity by a program. For a more detailed description see Section 1.6.1.
DATI	Data In
DATIP	Data In, Pause
DATO	Data Out
DATOB	Data Out, Byte
DBR	Data Buffer Register
DCDR	Decoder
DDB	Dataset Data Block. Contains Monitor control information for a dataset.
DE	Destination effective address
DEC	Decrement Digital Equipment Corporation
Default Device	The device specified in the Link Block of a dataset, and which is used for I/O operations on that dataset if there is no other device assigned in a DAT entry for the dataset.
DEL	Delay
DEP	Deposit
DEPF	Deposit Flag
Device Driver	The minimal routine which controls physical hardware activities on a peripheral device. The device driver is the interface between a device and the common, device-independent I/O code in the monitor.
DIV	Divide
DMA	Direct Memory Access
DSEL	Device Select
DST	Destination
DSX	Display, X-deflection Register
EMT	Emulator Trap
ENB	Enable
EOD	End-of-data
EOF	End-of-file
EOM	End-of-medium
ERR	Error

EX	External
EXAM	Examine
EXAMF	Examine Flag
EXEC	Execute
EXR	External Reset
F	Flag (part of signal name)
Fatal Error	An error from which a user's program cannot recover.
FBM	File Bit Map - A device-resident bit map with bits flagged for the blocks used for a single file. Used on DECtape to aid in the deletion process.
FCTN	Function
FIB	File Information Block. Contains (in core) information from the UFD and other sources when a file is open.
File	A physical collection of data which resides on a directory-structured device and is referenced through its name.
FILO	First in, last out
FLG	Flag
GEN	Generator
INC	Increment Increase
INCF	Increment Flag
IND	Indicator
INDIVR	Integer Divide Routine
INH	Inhibit
INIT	Initialize
INST	Instruction
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.
INTR	Interrupt
INTRF	Interrupt Flag
I/O	Input/Output
IOT	Input/Output Trap

IOX	Input/Output Executive Routine
IR	Instruction Register
IRD	Instruction Register Decoder
ISR	Instruction Shift Register
JMP	Jump
JSR	Jump to subroutine
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. The non-resident routines which process keyboard commands are brought into the keyboard swap buffer.
LIFO	Last In, First Out
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.
LKS	Line time clock status register
Load Module	The output of the linker. A program in absolute binary form ready for loading and executing on a PDP-11.
LOC	Location
LP	Line Printer
LSB	Least Significant Bit
LSBY	Least Significant Byte
LSD	Least Significant Digit
MA	Memory Address
MAR	Memory Address Register
MBR	Memory Buffer Register
MEM	Memory
MFD	Master File Directory. Contains the names and locations of all UFDs on a file-structured device.
ML	Memory Location
MOV	Move

MRT	Monitor Residency Table. Contains the address (on disk or in core) of all non-resident Monitor modules.
MSB	Most Significant Bit
MSB	Monitor Swap Buffer. The non-resident routines which process requests to the Monitor are brought into the main swap buffer.
MSBY	Most Significant Byte
MSD	Most Significant Digit
MSEL	Memory Select
MSYN	Master Sync
ND	Negative Driver
NEG	Negate
NOR	Normalize
NPG	Nonprocessor Grant
NPR	Nonprocessor Request
NPRF	Nonprocessor Request Flag
NS	Negative Switch
Object Module	The relocatable binary output of an assembler or compiler.
ODT	Octal Debugging Technique
OP	Operate Operation
Operator	A user communicating directly with the Monitor through the keyboard.
OPR	Operator Operand
OTS	OBJECT TIME SYSTEM
PA	Parity Available
PAL	Program Assembly Language
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.
PB	Parity Bit
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.
PC	Program Counter
PD	Positive Driver
PDP	Programmed Data Processor
PERIF	Peripheral

PGM	Program
PP	Paper Tape Punch
PPB	Paper Tape Punch Buffer Register
PPS	Paper Tape Punch Status Register
PR	Paper Tape Reader
PRB	Paper Tape Reader Buffer Register
PROC	Processor
PRS	Paper Tape Reader Status Register
PS	Processor Status Positive Switch
PTR	Priority Transfer
PTS	Paper Tape Software System
PUN	Punch
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.
RD	Read
RDR	Reader
REG	Register
REL	Release
RES	Reset
ROL	Rotate Left
ROM	Read-only Memory
ROR	Rotate Right
R/S	Rotate/Shift
RTI	Return from Interrupt
RTS	Return from Subroutine
R/W	Read/Write
R/WSR	Read/Write Shift Register
S	Single
SACK	Selection Acknowledge
SAL	A friend of SAM.
SAM	Swap Area Manager
SBC	Subtract Carry
SC	Single Cycle
SE	Source Effective Address

SEC	Set Carry
SEL	Select
SEN	Set Negative
SEV	Set Overflow
SEX	Sign Extend
SEZ	Set Zero
SI	Single Instruction
SP	Stack Pointer Spare
SR	Switch Register
SRC	Source
SSYN	Slave Sync
ST	Start
STPM	Set Trap Marker
STR	Strobe
SUB	Subtract
SVC	Service
SVT	System Vector Table
SWAB	Swap Byte
Swapping	The movement of programs or program sections from secondary storage to core.
TA	Trap Address Track Address
Table	A collection of data in a form suitable for ready reference.
TEMP	Temporary
TK	Teletype Keyboard
TKB	Teletype Keyboard Buffer Register
TKS	Teletype Keyboard Status Register
TP	Teletype Printer
TPS	Teletype Printer Status Register
TRT	Trace Trap
TSC	Timing State Control
TST	Test
UFD	User File Directory. Contains the names and locations of all files created under a UIC. (See MFD.)
UIC	User Identification Code. A code which associates a user with one of the UFDs on a device.

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.
UTR	User Trap
VEC	Vector
WC	Word Count
WCR	Word Count Register
XDR	X-line Driver
XRCG	X-line Read Control Group
XWCG	X-line Write Control Group
YDR	Y-line Driver
YRCG	Y-line Read Control Group
YWCG	Y-line Write Control Group

APPENDIX O
FILENAME EXTENSIONS

Extension	Attribute
ALG	ALOGL source file
BAS	BASIC source file
BAK	Backup file
BLI	BLISS source file
CBL	COBOL source file
CIF	Core Image File
CIL	Core Image Library
CMD	Command file
CRF	Input to cross-referencing program
DAT	DATA file for FORTRAN job
DDT	Reserved for DDT
DGN	Diagnostic message file
FTN	FORTRAN source file
FCL	FOCAL source list
LBO	Library of object modules (other types of libraries may also be implemented)
LCL	Linked core image library
LDA	Load module, Absolute
LDR	Load module, Relocatable
LOG	Logging file
LSP	LISP source file
LST	Listing file
MAC	MACRO assembler source file
MAP	MAP file
MFD	Master file directory
OBJ	Object module
OPR	Program generation information
OVR	Overlay
PAL	PAL assembler source file
PL1	PL/1 source file
RNO	Reserved for RUNOFF program
ROL	Reserved for ROLLIN program
RPG	RPG source file
SNO	SNOBOL source file
SPC	SPEC format text
STB	Symbol Table (Link-11 output)
SYM	File of symbols
SYS	System management
TMP	Temporary scratch file
UFD	User File Directory

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

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

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

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

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

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

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

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

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

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-4
Maynard, Massachusetts 01754

PDP-11 Disk Operating System
Monitor Programmer's Handbook
DEC-11-OMONA-A-D
October 1972

READER'S COMMENTS

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

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

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

How can this manual be improved?

Other comments?

Please state your position. ______________________ Date: __________

Name: ______________________ Organization: ______________________

Street: ______________________ Department: ______________________

City: ______________ State: ______________ Zip or Country ______________

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

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

digital

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754