UPDATE NOTICE No. 1

OS/78 User's Manual

Order No. AD-5748B-T1

September 1979

NEW AND CHANGED INFORMATION

This update contains the Table of Contents and revised information for the OS/78 User's Manual.

INSTRUCTIONS

Place the following pages in the OS/78 User's Manual as replacements for, or additions to, current pages. The changes made on replacement pages are indicated in the outside margin by change bars (▌) for additions and by bullets (•) for deletions.

Old Page	New Page
Title Page/Copyright	Title Page/Copyright
--	iii through xv
2-9/2-10	2-9/2-10
2-15/2-16	2-15/2-16
2-17/2-18	2-17/2-18
2-23/2-24	2-23/2-24
2-25/2-26	2-25/2-26
2-27/2-28	2-27/2-28
2-29/2-30	2-29/2-30
2-31/2-32	2-31/2-32
3-33/3-34	3-33/3-34
3-35/3-36	3-35/3-36
6-15/6-16	6-15/6-16
6-25/6-26	6-25/6-26
6-61/6-62	6-61/6-62
--	6-62.1/Blank
6-67/6-68	6-67/6-68
--	6-68.1/Blank
6-83/6-84	6-83/6-84
--	6-84.1/Blank
Index-1/Index-15	Index-1/Index-15

OS/78
User's Manual

Order No. AD-5748B-T1

September 1979

This document is the user's manual for the OS/78 V3 operating system. Its purpose is to acquaint new users with the operating system designed for the DECstation 78/88 minicomputers. This manual presents the background material for getting on the air, and a detailed description of the OS/78 commands that are used to direct computer operations. It also describes the OS/78 Editor, PAL8 assembly language, and the two high-level languages supported by the system, BASIC and FORTRAN IV. Command examples and demonstration programs are contained throughout the manual.

SUPERSESSION/UPDATE INFORMATION:	This manual supersedes previous editions, Order Numbers DEC-S8-OS78A-A-D, published 1977, and DEC-S8-OS78A-A-DN1, published 1978. This manual includes Update Notice No. 1.
OPERATING SYSTEM AND VERSION:	OS/78 V3

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard. massachusetts

First Printing, August 1979
Revised: September 1979

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

FIGURES

TABLES

3. Type the following commands in response to the monitor's period (.) prompting symbol:

```
.FORMAT RL01
.ZERO RL1A:/Y
.ZERO RL1B:
.ZERO RL1C:
.COPY RL1A:<RL0A:*.*
```

   The system will format the disk pack, zero each device and copy all the system files from unit A of Drive 0 to unit A of Drive 1. When the operation is complete, the monitor takes control as indicated by the period (.).

4. Remove both disk packs and label the disk pack taken from Drive 1 appropriately. This is the working system disk pack.

5. Store the master disk pack in a safe place.

6. Place the working system disk pack in Drive 0.

7. Press the system's BOOT button, and you are ready to use OS/78.

2.2.3 DECstations With RX02 and RL01 Disk Systems

The following steps describe how to make a backup copy of your master disk pack.

1. Place the master disk pack in RL01 Drive 0. Make sure that the drive is not write protected (WRITE PROT indicator lamp is extinguished).

2. Press the BOOT button.

3. Place a blank diskette in RX02 Drive 0.

4. Type the following command to initiate the backup operation:

```
.SUBMIT BUILDX
```

5. When the monitor again displays the period (.), remove the disk pack from its drive, and place a blank disk pack in the drive.

6. Type the following command to complete the backup operation:

```
.SUBMIT BACKRL
```

7. When the monitor again displays the period (.), the disk pack in Drive 0 now contains a complete copy of the contents of the OS/78 system software that resides on Device A (RL0A) of your master disk pack. If you want to make an additional backup copy of the system disk, type the following command:

```
.BO/RX
```

   then perform steps 5 through 7 again.

8. Store the master disk pack in a safe place.

2.3 DEMONSTRATION PROGRAMS

A series of programs are provided to demonstrate some of the capabilities of OS/78. These programs are run by a self-explanatory BATCH file called DEMO.BI. To initiate this demonstration, start the system and type the following command in response to the monitor's period (.):

```
.SUBMIT DEMO/H/T
```

The batch stream controlling the demonstration repeats it every three minutes until you type a CTRL/C.

Once you become familiar with the system, you may wish to remove the files comprising the demonstration to make room for other programs. To do so, use the DELETE command and type:

```
.DEL DEMO??.*
```

2.4 DEVICE AND FILE NAMES

2.4.1 Devices

The system recognizes each of its devices by one of the logical names listed in Table 2-2. You use these names in command strings to the system programs to specify the devices that you want to use. Although the system can recognize all the names, no one DECstation can have all listed physical devices. The SET HANDLER command, described in Chapter 3, allows you to specify which devices are available.

If you refer to a device that does not physically exist, the system will enter a loop condition and will not print any error message. You must restart the system by pressing the START or BOOT button to resume OS/78 operation.

Table 2-2
OS/78 Permanent Logical Device Names

Name	Description
DSK	Default output device; usually same as SYS
SYS	The diskette or disk pack inserted in Drive 0 where the monitor and system programs reside.
RL0A	Unit A of RL01 Drive 0
RL0B	Unit B of RL01 Drive 0
RL0C	Unit C of RL01 Drive 0
RL1A	Unit A of RL01 Drive 1
RL1B	Unit B of RL01 Drive 1
RL1C	Unit C of RL01 Drive 1
RXA0	The diskette inserted in Drive 0
RXA1	The diskette inserted in Drive 1
RXA2	The diskette inserted in Drive 2
RXA3	The diskette inserted in Drive 3
TTY	Keyboard/screen

(continued on next page)

2. If the command is recognized, execution will begin immediately. If you see an error in the command line, press CTRL/C quickly to terminate the action and return control to the monitor. Whether this step prevents execution of the incorrect command depends on the type of command and the file(s) involved.

   You can destroy important files if you incorrectly issue any of the following commands:

   COPY SAVE SQUISH DELETE ZERO

   In fact, the error may actually destroy part of the software. Therefore, be extremely careful when using these commands. Any specific cautions that should be exercised when using these commands are fully described in Chapter 3. Always be sure to make a back-up copy of the system and all other important files that you have created (see Section 2.2).

Section 2.5.3 describes how to correct any errors that have been made in the command line prior to executing the command.

2.5.3 Correcting Errors

Until the RETURN key is pressed, the system will not process the command line. Therefore, the command line is still available for correction. Always check the command lines before pressing the RETURN key since a line with errors may cause undesirable results.

To correct typing mistakes, use the DELETE key. This key erases the last character typed. Each use of the DELETE key causes one more character to be erased. The correct character or characters can then be typed.

A command line may be deleted completely before it is entered by typing CTRL/U (produced by holding down the CTRL key and pressing the U key). This echoes as ^U and returns control to the monitor without processing the current line. The monitor's period (.) indicates that the system is ready to accept a new command. You can then retype the command line. You can use the LINE FEED key to verify the contents of a command that is being entered. The system will display the portion of the command line that you have entered.

2.5.4 Using Input/Output Options

In addition to specifying output and input files in the command line, you can specify various options to perform select certain functions. Options are numbers, alphanumeric characters, or their full-word equivalents. Numbers used as options are generally contained in the command line with the equal sign (=) or square brackets ([]) construction. The alphanumeric option characters are set off from the I/O specifications by the slash (/) character for single character options and parentheses for a string of single characters. The usage of the slash, parentheses, equal sign, and square brackets is explained below.

You will find an explanation of the single character options in the command descriptions in Chapter 3. Appendix I lists the full-word equivalents to the single character options. A group of general-purpose options known as dash options that you can use in most command lines are described below.

2.5.4.1 **Equal Sign Construction** - An equal sign (=) followed by an octal number may occur only once in a command line and must be followed by a separator character (comma or left-angle bracket), other options, or a line terminator (RETURN or ESCape). For example,

```
.DIRECT SYS:=3
```

will list the directory of the system device on the screen in three columns.

2.5.4.2 **Slash Construction** - A single alphanumeric character is preceded by a slash and can occur anywhere in the command line (even in the middle of a file name) although the usual position is at the end of the line. For example:

```
.COPY RXA1:SECOND.EX<RXA0:FIRST.EX/T
```

means the same as

```
.COPY RXA1:SECOND.EX/T<RXA0:FIRST.EX
```

The option /T instructs the monitor to assign the current date to the output file.

2.5.4.3 **Parentheses Construction** - When two or more letter options are used, you can group them together inside parentheses. This construction is valid anywhere in the command line. For example,

```
.COPY RLIB:OUTPUT.EX<SYS:INPUT.EX(QT)
```

means the same as

```
.COPY RLIB:OUTPUT.EX<SYS:INPUT.EX/Q/T
```

2.5.4.4 **Square Bracket Construction** - The square bracket construction can only occur immediately after an output file name and consists of a left square bracket ([), a decimal number between 1 and 255, and a right square bracket (]). The square bracket construction allows you to optimize file storage by specifing an upper limit on the number of blocks required by your output file. For example:

```
.PAL BINARY[19],LISTIN[200]<SOURCE
```

The output files are a file named BINARY on device DSK: having a maximum length of 19 blocks, and a file named LISTIN (only six characters are significant) on the device DSK: with a maximum length of 200 blocks. The input file is SOURCE on device DSK.

2.5.4.5 **General-Purpose Dash Options** - These options provide a shorthand method for specifying devices and files with certain commands. The form is:

```
-ex
```

where:

-ex is one of the options specified in Table 2-4.

In the following example, typing

```
*PAL TEST-T
```

will assemble TEST.PA, store the resulting binary program in TEST.BN on DSK, and display the program listing on the terminal. The -T option saves you from typing the specification:

```
,TTY:<
```

Table 2-4
General-Purpose Dash Options

Option	Meaning
-FT	Selects the FORTRAN IV Compiler if you do not use the default file extension .FT (used with the COMPILE and EXECUTE commands).
-L	Send output to LPT. For example, typing .DIR-L will list the director of the system device on the line printer (LPT).
-LS	Generate a listing file (used with the COMPILE, EXECUTE, and PAL commands). The listing file is written onto SYS: if no output device is specified and is given a .LS extension. The listing file name is the same as the file name that immediately preceded the -LS option in the command string.
-MP	Generate a memory map (used with the COMPILE, EXECUTE, and PAL commands).
-NB	Do not create a binary file (used with the COMPILE, EXECUTE, and PAL commands).
-PA	Selects the PAL8 Assembler if you do not use the default file extension .PA (used with the COMPILE and EXECUTE commands).
-T	Send output to terminal.

2.5.5 Remembering Previous Arguments

The system remembers the arguments that you use with the CREATE, LOAD, PAL, EDIT, and EXECUTE commands by storing them in a temporary file. Thereafter, when you use any of these commands without arguments, the system will use the arguments that it stored. When you again use these commands with arguments, the system replaces the currently stored arguments with the new arguments. The storage area for the EDIT command is separate from the others and its contents will not change unless you issue an EDIT command with arguments.

The following example illustrates this feature. If you entered the command:

```
.EXECUTE TEST1.PA
```

you could compile TEST.PA by issuing the command:

```
.COMPILE
```

NOTE

> The system does not remember command arguments typed if you restart the system with the START button or the BOOT command.

2.5.6 Using Wildcards

The wildcard construction is a feature that allows you write generalized file specifications when manipulating files. Wildcards allow a file name or the extension specified in a command to be replaced totally with an asterisk or partially with a question mark. This allows you to designate classes of file names or extensions. The wild characters are particularly useful when doing multiple file transfers.

You can use the wildcard construction in the file name specifications of the COPY, DELETE, DIRECT, LIST, RENAME, and TYPE commands.

The asterisk (*) is used as a wild field to designate the entire file name or extension. This is illustrated in the following examples:

TEST1.*	All files with the name TEST1 and any extension.
*.BN	All files with a BN extension and any file name.
.	All files (except when used with the DELETE command).

The question mark (?) is used as a wildcard character to designate part of the file name or extension. A question mark is used for each unspecified character that is to be matched. For example, PR?? matches all files beginning with PR that are two to four characters long. Other examples are as follows:

TEST2.B?	All files with the name TEST2 and any extension beginning with B.
TES??.PA	All files with a PA extension and any file name from three to five characters long beginning with TES.
??.??	All files with file names of two characters or less.

The asterisk and the question mark can be specified together in the same command line:

???.*	All files with file names of three characters or less.

2.8 FILE-STRUCTURED DEVICES

A file-structured device is a random-access mass storage device that is logically divided into a number of blocks. This kind of device can read and write any desired block. A diskette and disk pack are file-structured devices, but a terminal is not.

All OS/78 file-structured devices are logically divided into 256-word blocks. Hence, blocks of 256 words are considered the standard size for OS/78. A file consists of one or more sequential (consecutively numbered) blocks. A minimum of one block per file is required, although a single file could occupy all of the blocks on a device.

By convention, OS/78 block numbers are specified in octal radix, while file lengths (number of blocks per file) are specified in decimal radix.

There are two types of file-structured devices, system devices and non-system (or files only) devices. A system device contains a copy of the OS/78 Monitor (also called the system head) on its medium and the OS/78 system programs; a non-system device does not.

The system device is also the disk drive that is accessed when you press the DECstation's START or BOOT button or use the BOOT command. If your system has an RX01 or RX02 diskette drive, the system device is Drive 0 (RXA0). If your system has RL01 cartridge disk drives only, the system device is device A of RL01 Drive 0 (RL0A).

Table 2-5 lists the OS/78 file-structured devices and their storage capacities.

Table 2-5
Storage Capacity of OS/78 File-Structured Devices

Device	Type	Size (in blocks)
RLxA	System	4018
	Non-system	4074
RLxB	Non-system	4074
RLxC	Non-system	2018
RX01	System	431
	Non-system	487
RX02	System	932
	Non-system	981
VXA0*	Non-system	128

* DECstation 88/97 systems only

2.8.1 Using RX01 And RX02 Diskettes

The RX01 and RX02 diskettes are physically identical. The RX01 is a single-density diskette and the RX02 is a double-density diskette. This means that an RX02 diskette has approximately twice the storage capacity of an RX01 diskette (see Table 2-4).

The RX01 diskette is supplied already formatted for single-density operation. You must, however, create a directory area on it with the ZERO command before copying files to it. If you are creating backup copies with the DUPLICATE command, the ZERO command is not necessary.

The RX02 diskette is a single-density RX01 diskette that you reformat for dual-density operation with the DUPLICATE command's /D option. You must, however, create a directory area on it with the ZERO command before copying files to it. If you are creating backup copies with the DUPLICATE command, the ZERO command is not necessary.

You should observe the following rules when using diskettes:

- An RX01 disk drive can use single-density diskettes only.
- An RX02 disk drive can use either single-density or double-density diskettes. The system automatically determines the density of the diskette.
- A system diskette that contains an old version of OS/78 can run on an RX01 drive only. You can use it on an RX02 drive if you want to use it for file storage only.

2.8.2 Using the RL01 Cartridge Disk Pack

The RL01 cartridge disk pack is a high-density mass storage device that has a storage capacity that is approximately ten times that of the RX02 double-density diskette (see Table 2-4).

Each disk pack consists of three logical OS/78 devices: device A, device B, and device C. Device A and B are the same size. Transfers between these devices are faster than transfers between either one of them and device C. This is because the device C is smaller than the other two and consequently occupies less of each track on the disk than the others. This increases the search time required for device C blocks.

You must format and zero each new blank disk pack with the FORMAT and ZERO commands before using it with OS/78. If you do not do this, OS/78 cannot successfully use the disk pack.

Each disk pack incorporates a bad block mapping scheme that allows OS/78 software to ignore disk blocks that become defective. Whenever a bad block error occurs, you can use the FORMAT command to make the block unavailable to OS/78. This allows you to prolong the life of your disk packs.

2.8.3 Using the VXA0 Extended-Memory Device

The VXA0 device enables you to use the extended memory above 32K provided in DECstation 88/97 systems as though it were a separate file-structured device. The VXA0 device provides high speed I/O performance similar to that of a fixed-head disk type of storage device. You refer to it in the same way as the other OS/78 devices. For example, this command

```
.COPY VXA0:SAMPLE<RXA0:SAMPLE
```

copies a program called SAMPLE into the area of memory above 32K. The command

```
.DIR VXA0:
```

produces a directory listing of the device.

2.9 FILES

A file is the fundamental storage unit of the OS/78 system. It is a uniquely named collection of data that is treated as a unit. The format of this data is unimportant; OS/78 can manipulate several standard formats, including ASCII files, binary files, and memory-image files. The important consideration is that the data forms a single and uniquely identifiable unit within the system.

2.9.1 File Directories

To maintain records of the files on a device, blocks 1 through 6 are reserved for the file directory of that device. Thus, file structured devices are also called directory devices. Entries in this directory inform the system of the name, size, anad location of each file, including all empty files and the tentative output file, if one exists. Six blocks are always allocated, though all are not necessarily active at any given time. Block zero is unused, except on the system device, in which case it contains the system bootstrap, a program that reads in the OS/78 Monitor and the system device handler (SYS).

2.9.2 File Types

Three types of files exist in the OS/78 system:

1. Empty File - An empty file is a contiguous area of unused blocks. Empty files are created when permanent files are deleted or when the ZERO command is used.

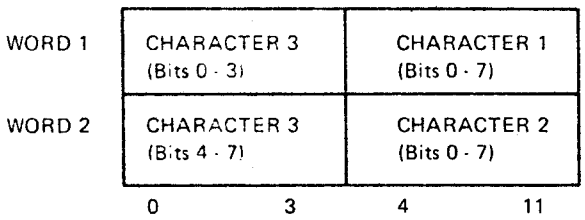
2. Tentative File - A tentative file is a file that is open to accept output and has not yet been closed. Only one tentative file can be open on any single device at one time. Tentative files never appear in directory listings.

3. Permanent File - A permanent file is a file that has been given a fixed size and is no longer expandable. A tentative file becomes permanent when it is closed.

To further understand file types, consider what occurs when a file is created. Normally, in creating a tentative file, the system first locates the largest empty file available and creates a tentative file in that space. That establishes the maximum space into which the file can expand. The user program then writes data into the tentative file. At the end of the data, the program tells the system to close the tentative file, making it a permanent file. When the file is closed, whatever space remains at the end of the file is available to contain a new file.

2.9.3 ASCII File Format

ASCII files are packed as three 8-bit characters into two words as follows:



2.10 OS/78 COMMAND SUMMARY

Table 2-6 is a summary of OS/78 commands. For each command, the table shows the command and its abbreviation printed in red ink, the format, and a brief functional description.

Table 2-6
OS/78 Command Summary

Command	Format Example	Function
ASSIGN	AS permdev user-name	Associates a new user-defined device name with the permanent logical device name.
BASIC	BAS	Requests execution of the BASIC language editor to allow for program creation and modification.
BOOT	BO /dv	Initializes the system residing on the specified system device RL or RX.
CANCEL	CA	Terminates the symbiont task (see Appendix H) that is currently running and returns OS/78 to 16K, single task operation (DECstation 78 systems only).
COMPARE	COMPA dev:output.ex< dev:file1.ex,dev:file2.ex	Compares two source files line by line and displays all their differences.
COMPILE	COM file.ex<file1.ex,.../options	Produces binary files and/or listings for specified program files. This command will chain to PAL8, BASIC, or FORTRAN IV depending on the file name extension. Under FORTRAN IV, only one input file can be specified.
COPY	COPY dev:file.ex<dev:file.ex,...	Transfers files from one OS/78 device to another. Up to five input files can be specified.
CREATE	CREA file.ex	Opens a new file for editing.
CREF	CREF file.ex	Assembles and produces a cross-reference listing from the source file. Defaults to LPT, thus the -T option must be used to output to terminal.

(continued on next page)

Table 2-6 (Cont.)
OS/78 Command Summary

Command	Format Example	Function
DATE	DA dd-mmm-yy	Enters the date into the system. If no argument is given, print current date on terminal or NONE if no date was specified.
DEASSIGN	DE	Eliminates all previous user-defined names.
DELETE	DEL dev:file.ex,.../options	Deletes one or more files. Up to five input files can be specified, separated by commas. All files must reside on the same device.
DIRECT	DIR dev:/options	Produces a listing of an OS/78 device directory.
DUPLICATE	DUP outdev:<indev:/options	Reproduces the contents of the input diskette onto the output diskette.
EDIT	ED file or ED file<file	Opens an already existing file for editing. Using the input/output construction allows a copy of the original file to be retained, while naming a new output file.
EXECUTE	EX file.ex,file.ex/options or EX file.ex<file.ex	Produces binary files and/or listings for the specified program files, loads the binary file, and executes the program. This command will chain to PAL8, BASIC or FORTRAN IV depending on the file name extension. Under FORTRAN IV, only one file can be specified.
FORMAT	FORMAT RL01 /options	Formats RL01 disk pack for OS/78 use.

(continued on next page)

Table 2-6 (Cont.)
OS/78 Command Summary

Command	Format Example	Function
GET	GE dev:file.ex	Loads memory image files (.SV format) into memory from a device.
HELP	HE command	Displays instructional information on the specified OS/78 command.
LIST	LI dev:file.ex,...	Lists the contents of the specified files on LPT.
LOAD	LO dev:file.ex	Runs one of the OS/78 loaders for either PAL8, BASIC or FORTRAN IV depending upon the extension of the filename.
MAP	MAP dev:file.ex<dev:file.ex	Produces a memory map of the specified input file.
MEMORY	MEM n	Sets the number of memory fields available.
	MEM	Displays both the number of fields actually being used by OS/78 and the number of fields available in the hardware.
ODT	OD	Loads and starts the ODT (Octal Debugging Technique) system.
PAL	PAL dev:binfile,dev:listfile, dev:creffile<dev:infile,...	Runs the PAL8 assembler and assembles the specified source file.
QUEUE	Q filename-list/options	Line printer spooler symbiont command (see Appendix H) to queue files to be printed (DECstation 78 system only).

(continued on next page)

Table 2-6 (Cont.)
OS/78 Command Summary

Command	Format Example	Function
R	R file.ex	Loads memory image files (except OS/78 system programs) from the system device (making it equivalent to a RUN SYS file.ex command except that the system scratch area is not affected) and starts it at the starting address. If .ex is omitted, .SV is assumed.
RENAME	REN dev:new.ex<dev:old.ex/options	Changes the name of the file from the input name to the output name.
REQUEST	REQ symbiont-name	Starts symbiont task (DECstation 78 systems only. See Appendix H).
RUN	RU dev:file.ex	Loads a memory image file (except OS/78 system programs), moves its Core Control Block to the system scratch area, and starts the program at its starting address. This command is equivalent to a GET and START command, and must be used to run a program that does not reside on SYS:.
SAVE	SA dev:file.ex options	Saves an image of the program currently in memory on the device specified. If .ex is omitted, .SV is assumed.
SET		Changes device handler characteristics. ter: = TTY:, SLUx, VLUx any: = any device NO = optional modifier to inhibit effect
	SET any: [NO] READONLY	Makes device read-only device.
	SET LPT: [NO] LC	Prints lower-case characters.

(continued on next page)

Table 2-6 (Cont.)
OS/78 Command Summary

Command	Format Example	Function
SET (Cont.)	SET LPT:WIDTH n	Sets column width to n.
	SET LPT:COL n	Sets number of columns displayed by DIRECT (ter: also).
	SET ter: [NO] ARROW	Sets terminal so that CTRL characters map as ^char rather than the actual code.
	SET ter:COL n	Sets number of columns displayed by DIRECT (LPT: also).
	SET ter: [NO] ECHO	Enables character echoing on input.
	SET TTY: [NO] ESC	Sets console terminal so that ESC code maps to a $ rather than 033(8).
	SET ter:HEIGHT m	Sets screen height to m.
	SET SYS: [NO] INIT	Executes command in INIT.CM when system is bootstrapped.
	SET SYS: [NO] INIT cmd	Executes command (cmd) when system is bootstrapped.
	SET ter: [NO] PAGE	Enables CTRL/Q and CTRL/S.
	SET ter: [NO] PAUSE n	Enables pause of n seconds during output.
	SET ter: [NO] SCOPE	Deletes character from screen rather than printing a \ when DELETE key is used.
	SET ter:WIDTH n	Sets column width to n.
	SET HANDLER old<new	Replaces handler currently in system (old) with new handler (new) called: handlername.HN.

(continued on next page)

Table 2-6 (Cont.)
OS/78 Command Summary

Command	Format Example	Function
SET (Cont.)	SET HANDLER/L	Lists names of handlers currently in system.
SQUISH	SQ dev:<dev:	Eliminates all embedded empty files on the device.
START	ST ffnnnn	Starts the program currently in memory at location nnnn in field ff.
	ST	Starts the program currently in memory at the starting address specified in the Core Control Block.
SUBMIT	SU dev:file.ex<dev:file.ex	Runs the BATCH program where the output is the optional spooling file and the input is the BATCH input file.
TERMINATE	TER	Causes OS/78 to terminate its operation and to run a user-created stand-alone program called TERMIN.SV.
TYPE	TY dev:file.ex,...	Displays the specified files on the terminal.
UA	UA (argument)	Remembers the argument where the argument must be a legal OS/78 command. UA with no argument executes the last remembered argument.
UB	UB (argument)	Similar to UA.
UC	UC (argument)	Similar to UA.
ZERO	ZERO dev:/options	Zeros the specified device directory, deleting any files that may exist or the device.

The above example shows a sequence of commands in creating and editing of file MATCH.PA. The EDIT command is given without any argument since it remembers the file MATCH.PA specified with the CREATE command.

If the date has changed and a command line is typed containing the EDIT command without any arguments, the error message BAD RECOLLECTION will be displayed.

Example 2:

```
.EDIT A2.FT<A1.FT
*R
 .
 .
 .
*E
 .
 .
 .
.EDIT
```

Since the second EDIT command had no arguments, the previous argument up to the left-angle bracket (<) is recalled. Thus, the second EDIT command is equivalent to:

```
.EDIT A2.FT
```

This command executes the CCL.SV and EDIT.SV programs.

EXECUTE

3.16 EXECUTE COMMAND

The EXECUTE command performs the following functions:

1. assemble or compile, link, load and execute a BASIC, FORTRAN, or PAL8 source program
2. link, load and execute an already assembled or compiled program
3. execute an already linked and loaded program

Format:

```
EXECUTE indev:file.ex
```

To use the EXECUTE command with a source file, the input device must be specified. Then that source file is assembled or compiled, linked and loaded into memory and executed. The file extension used determines the compiler or assembler called.

If the file name extension is not a standard extension, specify the correct assembler or compiler by using one of general-purpose dash options in the -ex portion of the command line (see Table 2-4).

Example:

```
.EXE RXA1:NABS.IN-PA
```

If the EXECUTE command is used with an already assembled or compiled program, the input device must be specified. When the EXECUTE command is processed, the binary file is loaded into memory and executed. If the file name extension is not specified in the command line but is specified in the directory, the correct extension is automatically added.

The EXECUTE command can remember arguments from a previous COMPILE, LOAD, PAL, or EXECUTE command. When you restart the system with the START or BOOT button or with the BOOT command, the arguments are lost.

NOTE

For options and expanded command strings acceptable to the EXECUTE command, refer to the appropriate language/loader section.

This command executes CCL.SV and one or more of the following programs: PAL8.SV and ABSLDR.SV, F4.SV, FRTS.SV and LOAD.SV, or BCOMP.SV and BLOAD.SV.

FORMAT

3.17 FORMAT COMMAND

The FORMAT command verifies and formats RL01 cartridge disk packs. You must use FORMAT on each new disk pack before you can store any OS/78 files on it. You should also use FORMAT to reformat any disk pack that has blocks that the system cannot access to make them invisible to the system. (After using FORMAT, you must also use the ZERO command to create a new directory.)

NOTE

FORMAT does not alter the contents of a disk pack that has no bad blocks. If there are bad blocks, the data residing in the good blocks that follow a bad block will become unusable after reformatting, so be sure to make copies of the files that you want to save before using FORMAT.

Format:

```
FORMAT RL01/n/P
```

where:

/n specifies the drive number, 0 or 1. The default value is 1.

/P causes the system to pause before and after the operation so you can change disk packs. If you select drive 0, the pause is automatic. Type a CTRL/C to return to the monitor. If you are running under BATCH, FORMAT does not pause after it completes the operation.

Example:

```
.FORMAT RL01
```

This command formats the disk pack on drive 1.

The FORMAT command executes the FORMAT.SV program.

NOTE

FORMAT.SV is available on RL01 and RX02 media only.

Table 3-13
FORMAT Error Messages

Message	Meaning
BAD COMMAND LINE	The command line contains a syntax error. Retype the line correctly.
BAD DISK	Disk pack cannot be reformatted. There are more than 63 bad blocks or the system area (blocks 0 - 70 octal) contains bad blocks.
CAUTION - THIS COMMAND DESTROYS CONTENTS OF SYSTEM DISK. CHANGE DISKS BEFORE PROCEEDING OR CTRL/C.	Drive 0 was selected for formatting. Be sure to replace your system disk pack with the one that you want to format.
READY. STRIKE CARRIAGE RETURN TO CONTINUE	The /P option was selected. Mount the disk pack to be formatted and type a carriage return to begin formatting.

For example, the following statement stores the value 37 in the variable N.

```
10 LET N=37
```

If N already contains a value, the new value replaces it.

Once you have assigned a value to a variable, BASIC will use the value in any expression in which the variable appears, for example:

```
10 LET B=N*2
```

This statement evaluates the expression N*2 and stores the result in the variable B.

You may instruct BASIC to change the value of a variable any number of times during the execution of a program. BASIC always uses the most recently assigned value of a variable when performing calculations. The following section describes numeric variables, string variables, and subscripted variables.

6.3.2.1 **Numeric Variables** - A numeric variable name consists of a letter or a letter followed by a digit, for example:

Acceptable Variables	Unacceptable Variables
M	2C (A variable cannot begin with a digit.)
R2	AB (A variable may contain only one letter.)

Unless you specify otherwise, BASIC automatically sets all variables to zero before executing a program. However, if you wish to assign zero, it is good programming practice to do the initializing yourself at the beginning of the program. You can do this with a series of LET statements or by using READ and DATA statements. For example, this statement

```
10 LET A=0\B=0\C=5
```

assigns 0 to A and B and 5 to C.

6.3.2.2 **String Variables** - A string variable name consists of a letter -- or a letter and a digit -- followed by a dollar sign. A$ and A2$ are both legitimate string variable names; 2A$ and AA$ are not.

You may assign up to 18 characters to a string variable. To assign more you must first dimension the variable with a DIM statement. (See Section 6.7.)

6.3.2.3 **Subscripted Variables** - A subscripted variable consists of a string or numeric variable followed by a subscript in parentheses. A subscript may be a number, a numeric variable, or an expression or any two such elements separated by a comma. The following are all legal subscripted variables.

```
A(2)

A(K)

M(3,4)

M$(I,J)

G(K-1)

F(3.4)
```

If the subscript is not a whole number, BASIC uses only the whole number part. In the example above F(3.4) is the same as F(3). BASIC permits a subscript value of zero.

One subscript after a numeric variable serves as a pointer to a location in a one-dimensional array. For example, this subscripted numeric variable

```
X(3)
```

indicates the fourth position in the array.

```
X(0), X(1), X(2), X(3), X(4), X(5)
```

Note that all subscripted variables in an array share the same variable name.

Two subscripts appended to a numeric variable (and separated by commas) indicate a row and column number in two-dimensional array. This variable

```
A(3,5)
```

points to row 3 column 5.

One subscript appended to a string variable indicates the length of the string. Two subscripts indicate its position in a list and its length. For example, this variable

```
R$(35)
```

will hold a string constant 35 characters long. And this variable

```
R$(5,35)
```

indicates that the sixth position in a list (beginning with 0) holds a string 35 characters long.

You cannot create a two-dimensional array of string variables in a BASIC program.

To specify the length of a string, use a DIM statement in the form:

```
(line number) DIM X$(n)
```

where:

X$	is a string variable
n	is the length of the string. A string may contain no more than 132 characters. All strings that exceed 18 characters in length must be dimensioned with a DIM statement.

To create a one-dimensional array of subscripted string variables, use the format

```
(line number) DIM X$(n,m)
```

where:

X$	is a string variable name
n	specifies the number of strings in the array. (The number of strings is n + 1.)
m	is the length of each string -- up to 132 characters

For example, this DIM statement describes one string 12 characters long.

```
10 DIM C$(12)
```

This statement describes 4 strings, each 20 characters long.

```
10 DIM D$(3,20)
```

This program will fill variables from a DATA list.

```
10 DIM D$(3,20)
20 FOR Y=0 TO 3
30 READ D$(Y)
40 NEXT Y
50 FOR Z=0 TO 3
60 PRINT D$(Z)
70 NEXT Z
80 DATA "ZERO","ONE","TWO","THREE"
```

Keep in mind the following features and rules concerning the DIM statement.

- Arrays are limited in size only by the amount of available memory -- that is, space not used by the monitor or the program statements.
- Subscripts n and m must be integer numbers. They may not be variables.
- A variable may not appear in a program with subscripts higher than the ones you have described in the DIM statement.
- BASIC assumes a string length of 18 characters or less unless you define the string variable with a DIM statement. If you wish to assign a string that is more than 18 characters long, you must DIMension the string variable.

- BASIC will not accept two-dimensional string variables.
- BASIC assigns a subscript of 0 to the first element in every array. Therefore, the number of elements in a one-dimensional array is n + 1, and the number of elements in a two-dimensional array is (n + 1)*(m + 1).
- You may define more than one array with a single DIM statement. For example, this statement dimensions both the one-dimensional array A and the two-dimensional array B.

```
10 DIM A(20), B(4,7)
```

6.8 THE COMMENT STATEMENT -- REMARK AND EXCLAMATION POINT

The REM statement or an exclamation point lets you document your source program with notes and comments, for example:

```
10 REM SUBROUTINE SWAPS VALUES A AND B
```

or

```
10 ! SUBROUTINE SWAPS A AND B
```

Use the exclamation point to add a comment to an executable line, for example:

```
100 A(I,J) = B(I,J) !  COPY B TO A
```

6.9 INPUT AND OUTPUT

BASIC provides you with three ways to supply a program with data.

- The INPUT statement lets you type in data at runtine.
- The READ, DATA, and RESTORE statements let you include data in a source program.
- BASIC file and record statements make it possible for you to store data in files and retrieve it under program control.

This section describes only the first two methods. See Sections 6.12 and 6.13 for information on file and record input and output.

The BASIC PRINT statement causes BASIC to display strings and the results of computations on the terminal.

6.9.1 INPUT

The INPUT statement allows you to enter data from the terminal during program execution.

Format:

```
(line number) INPUT x1, x2,...,xn
```

where:

x1 through xn represent numeric variables or string variables.

6.11.12.1 **The TAB Function** - he TAB function enables you to position characters on a line.

Format:

```
PRINT TAB(n)
```

where:

n is the number of a column.

For example, to print an upper-case "X" in column 10 of your screen, type

```
5 PRINT TAB(10);"X"
```

6.11.12.2 **The PNT Function** - The PNT function accepts an ASCII code number in decimal and returns an ASCII character. PNT is useful for transmitting non-printing characters and control codes to the terminal.

PNT works only in combination with a PRINT (or PRINT#) statement.

Format:

```
PRINT PNT(n)
```

where:

n is an ASCII code number expressed in decimal.

Use the PNT function to transmit control codes to the video terminal. For example, code 07 decimal (which you can specify by typing CTRL G) rings the bell on the terminal. To call for this action in a BASIC program, use the PNT function in a PRINT statement, specifying code 07 as the argument.

```
10 PRINT PNT(07)
20 END
```

This complete BASIC program rings the terminal bell.

The following control codes move the cursor on the screen.

PNT(08) moves cursor one space to the left

PNT(09) moves cursor to next tab stop

PNT(10) moves cursor down one line and scrolls if necessary

PNT(13) moves cursor to left margin of current line

To call for an escape sequence in a BASIC program, enter the ESCape code (27 decimal) with the PNT function, followed by the letter code for the sequence you wish to generate.

Format:

```
PRINT PNT(27);"X"
```

where:

27 is the code for ESCape

X is any capital letter that generates an escape sequence.

For example, the following statement transmits the sequence ESC H to the terminal. ESC H moves the cursor to the upper left corner of the screen.

```
10 PRINT PNT(27);"H"
```

For a list of additional ESCape Sequences that you can call for with the PNT function, see Section 6.14.

For a complete discussion of control codes, including ESCape Sequences, see the user's manual for your terminal: the DECscope User's Manual (EK-VT5X-OP-001) or the VT100 User Guide (EK-VT100-UG-PRE).

6.12 FILE STATEMENTS

BASIC file statements -- which are distinguished from other BASIC statements by the number sign (#) -- let you store data on peripheral devices for later use in any BASIC program. They include

- FILE# describes the file, assigns it a channel number from 1 to 5 (the number of files which BASIC can handle at one time), and opens it.
- PRINT# writes data on the file.
- PRINT# USING defines format of string numbers in a file.
- INPUT# reads data from the file.
- RESTORE# resets the pointer to the beginning of the file.
- CLOSE# closes the file and removes the channel number.
- IF END# tests for end-of-file.
- IF OPEN# determines if the file you specify is currently open.

In most operations, you open a file (FILE#) for output or input, write on it (PRINT#) or read from it (INPUT#), and close it (CLOSE#). You can open five files at a time -- excluding the terminal, which is always open and available for use -- on the system device plus 3 non-system devices. The ability to open and close files under program control gives you access to an unlimited number of files. That is, when you close a file, you may re-assign its channel number to a newly opened file.

BASIC treats files in the same way it treats terminal input and output. The INPUT statement causes BASIC to read a value that you enter on the terminal and assign it to a variable; the INPUT#

statement causes it to read a value from a file. The PRINT statement instructs BASIC to display data on the terminal; the PRINT# statement tells it to write data in a file.

will display:

```
1
2
3
4
5
6
```

6.12.2.3 Formatting Output to a File -- PRINT# USING - The PRINT# USING statement lets you define the format of string numbers that you want to write to a file.

The difference between PRINT# USING and PRINT USING is the same as the difference between PRINT# and PRINT: instead of causing BASIC to display formatted data on the terminal, PRINT# USING causes BASIC to place it in a file.

For a complete description of the procedure you follow to define the format of a string number, see the PRINT USING statement (Section 6.9.4).

Format:

```
(line number) PRINT #n: USING format string, list
```

where:

n	is the channel number of a file opened to receive string output
format string	is a string or string variable describing the format of the string numbers you want to write to the file
list	is a series of string numbers, string variables, or string functions separated by commas or semicolons.

For example, the following statement defines the format for any string number that you assign to the string variable A$ and places the formatted value in file #1.

```
20 PRINT #1: USING "$****.##",A$
```

6.12.2.4 Resetting a File -- RESTORE# - The RESTORE# statement resets the file back to the beginning so that the next INPUT# statement will cause BASIC to read the first item in the series.

Format:

```
(line number) RESTORE# n
```

where:

n	is the channel number of the file to be reset or a variable representing the channel number.

If n is 0, BASIC resets the DATA list to the beginning.

In the following program, RXA1:FILB.LM is a numeric input file containing the numbers 1 through 9. These instructions

```
100 FILEN#3:"RXA1:FILB.LM"
110 FOR I=1 TO 3
120 INPUT #3:Z
130 PRINT Z
140 NEXT I
150 RESTORE#3
160 INPUT#3 Z
170 PRINT Z
999 END
```

will display:

```
1
2
3
1
```

6.12.2.5 Checking for Open Files -- IF OPEN# - This statement lets you determine if a FILE statement has successfully opened a file or if a CLOSE statement has closed all files at the conclusion of a program. You can also use the IF OPEN statement to determine if a file exists.

Format:

(line number) IF OPEN #n THEN ln

where:

n is the number of the file you wish to test; ln is the line number of the statement to which the program will jump if the file is open.

For example, the following program

```
10 DIM J$ (15)
20 FILEV #2: "CHECKF.FI"
30 PRINT #2: "CHECK FILE"
40 CLOSE #2
50 IF OPEN #2 THEN 80
60 PRINT "FILE NO.2 CLOSED"
70 GO TO 99
80 PRINT "FILE NO.2 OPEN"
99 END
```

will display

```
FILE NO.2 CLOSED
```

Without the CLOSE statement in line 40, BASIC would execute line 80, displaying the information that the file was open.

This statement provides a way to protect a data file in a long program, where you might accidentally omit the CLOSE statement.

You can use the IF OPEN statement to determine if a file that is called for actually exists. For example in the following sequence of statements:

```
10 FILE #1 "PROG.PA"
20 IF OPEN #1 THEN 50
30 PRINT "FILE NOT THERE"
40 STOP
50 ....
```

line 10 opens file PROG.PA. If PROG.PA does not exist, the program proceeds from the IF OPEN statement to line 30 and displays the message. If PROG.PA is open, and therefore exists, the program jumps to line 50.

6.12.2.6 **Checking for End-of-File -- the IF END# Statement** - The IF END# statement lets you detect the end of a string file.

Format:

(line number) IF END#n THEN m

where:

n	is the channel number of the file in question or a variable representing the number.
m	is the number of the line in the program which BASIC will jump to if it has reached the end of the file.

The IF END# statement works only on string files and must immediately follow the PRINT# or INPUT# statement for that file.

When you use the IF END# statement, you are asking BASIC to check if its last attempt to execute a PRINT# or INPUT# statement was successful. If it was unsuccessful -- if nothing was written or read -- BASIC jumps to line m.

Statement	Function
PRINT#	Writes data to a file Example: 180 PRINT#1:J
PRINT USING	Allows formatting of string numbers displayed on terminal Example: 10 PRINT USING "##","3.7"
PRINT# USING	Allows formatting of string numbers written to a file Example: PRINT #1: USING "$****.##",A$
PUT#	Writes file records Example: PUT#1:O,A$,B$,C$
RANDOMIZE	Causes the RND function to produce a different set of numbers each time the program is run Example: 10 RANDOMIZE
READ	Sets variables equal to the values in DATA statements Example: 50 READ A$,B
REM	Inserts comments into the program Example: 30 REM COMPUTE EARNINGS
RESTORE	Sets program READ statements back to the beginning of the DATA list Example: 85 RESTORE
RESTORE#	Resets a file pointer back to the beginning of that file Example: 130 RESTORE#3
RETURN	Returns control from a subroutine (used with GOSUB) Example: 115 RETURN
STOP	Terminates program execution Example: 40 STOP

6.22 SUMMARY OF BASIC FUNCTIONS

Command	Function
ABS(X)	Returns the absolute value of an expression Example: 10 LET X=ABS(-66) will assign X a value of 66

Command	Function
AND(A,B)	Performs a bitwise AND on two numbers entered in decimal
ASC(X$)	Converts a one character string to its code number Example: 20 PRINT ASC("B") will display 2
ATN(X)	Calculates the angle (in radians) whose tangent is given as the argument Example: 30 LET X=ATN(.57735) will assign X a value of 0.523598
CAP$(X$)	Converts lower case to upper case
CCL(X$)	Causes BASIC to exit from currently running program, chain to CCL.SV, and pass the argument assigned to the function Example: 100 X=CCL("SUBMIT BJOB.BI/T") terminates BASK program and submits BATCH job BJOB.BI
CHR$(X)	Converts a code number to its equivalent character Example: 40 PRINT CHR$(1) will display A
COL(N)	Returns the column number where the print head is positioned
COS(X)	Returns the cosine of an angle specified in radians Example: 50 LET Y=COS(45*3.14159)/180 will assign Y a value of 0.707108
CUR$(V,H)	Outputs a VT52 escape sequence
DAT$(X)	Returns the current system date Example: 60 PRINT DAT$(X) will display the system date, such as 07/20/77
EXP(X)	Calculates the value of e raised to a power, where e is equal to 2.71828 Example: 30 IF Y>EXP(1.5) GOTO 70 will go to line 70 if Y is greater than 4.48169
FIX$(X$)	Accepts a string number as an argument and returns the integer part Example: 10 PRINT FIX$ ("4"/"3") prints the value 1

Command	Function
INT(X)	Returns the value of the nearest integer not greater than the argument

Example: 60 LET X=INT(34.67)

will assign X the value 34

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS ML 5-5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

Do Not Tear - Fold Here