CHAPTER 8

BATCH

OS/78 BATCH processing is ideally suited to lengthy or after-hour jobs consisting of fixed sequences of OS/78 system commands not requiring operator intervention. BATCH allows you to create a file that contains the OS/78 commands. Output can be line printer listings and files. These files contain program output or comprehensive summaries (log) of all action taken by the program, OS/78 BATCH, and the operator.

BATCH provides optional spooling of output files. Spooling allows printed output to be diverted to a fast file-structured device, such as a diskette, for later transferral to the slower device. This feature serves to increase perceived throughput on the system. A line printer, although optional, makes the use of BATCH more effective.

With a few exceptions, BATCH executes standard OS/78 commands.

8.1 BATCH PROCESSING UNDER OS/78

OS/78 BATCH maintains an input file and an output log. The BATCH input file consists of a series of BATCH commands. The input file must reside on the system device (disk pack or diskette). Its default extension is .BI. Each command in the BATCH input file generally occupies one line.

The BATCH output file is a line printer listing (log) on which BATCH prints job headers, certain messages that result from conditions within the input file, an image of each line in the input file, and certain types of user output. BATCH supports only the LA78 line printer for the output of the BATCH log. Listings, however, can be output on the LQP78 line printer. If a line printer is not present in the system, the output file is displayed on the terminal.

BATCH accepts program and data files from any input device in the system. User output files may be directed to any output device in the system.

BATCH also permits optional spooling of output files. When spooling is requested, every output request to a non-file-structured device output file is assigned a file name from a list of names maintained by BATCH and directed to a file-structured spool device instead of the user-specified device. Spooling of output files increases BATCH throughput, permitting slow output operations to be postponed until a more favorable time. For example, a batch processing run that generates many output listings may be initialized to reroute all listings to a specified diskette. The listings on the diskette may then be dumped onto the appropriate hard copy device after the run, when more time is available. The spool device may be any OS/78 file-structured device.

BATCH is called via the SUBMIT command. The format for a BATCH command string is:

```
SUBMIT spool-dev:<dev:input/options
```

where:

spool-dev: is the device on which to spool output. If not specified, no spooling is performed. Note that spooling applies only to non-file-structured output.

dev:input is the input device and file. The default extension for BATCH input files is .BI. The default input device is DSK.

/options are listed in Table 8-1.

Table 8-1
BATCH Run-Time Options

Option	Meaning
/E	Treat OS/78 Monitor errors as non-fatal errors. If /E is not specified, monitor errors cause the current BATCH job to be terminated.
/H	Do not output a BATCH log.
/Q	Output an abbreviated BATCH log, consisting of $JOB and $MSG lines.
/T	Output the BATCH log to the terminal. This option need be specified only if a line printer is available. If a line printer is not available, the BATCH log is automatically output to the terminal.
/U	BATCH will not pause for operator response to $MSG lines. Any attempt to use TTY: as an input device to an unattended BATCH stream will cause the current job to be aborted.

8.2 BATCH COMMANDS

A BATCH command is a character or string of characters that begins with the first character of a line in the BATCH input file. Each BATCH command must be followed by a RETURN. The files may contain form feed characters, but form feed characters are ignored by BATCH on input.

BATCH recognizes four monitor level commands. These commands allow routine housekeeping operations in a multi-job, batch processing environment and provide communication between the BATCH user and the operator. Table 8-2 lists the BATCH commands, which may be considered as an extension of the OS/78 Monitor command set. Note that the first character of the $JOB, $MSG and $END commands is a dollar sign (SHIFT/4).

Table 8-2
BATCH Commands

Command	Meaning
$JOB	Initialize for a new job and display a job header on the output file. The remainder of the $JOB record is included in the job header but ignored by BATCH. It should be used for job identification, to provide correlation between terminal output, line printer output and spool device output.
$MSG text	Sound the terminal buzzer and print the message specified by the argument text at the terminal. If the /U option was not specified, implying that an operator is present, BATCH will pause until any key is struck at the keyboard. If the /U option was specified, processing continues uninterrupted.
$END	Terminate batch processing and exit to the monitor. A $END command should be the last line of every BATCH input file.
/	Copy the line onto the output log, then ignore it. BATCH assumes that every line beginning with a slash is a comment.

Any record that begins with a dollar sign character but is not one of the BATCH commands listed above is copied onto the output file and ignored by BATCH.

A BATCH processing job consists of a $JOB command line and all of the commands that follow it up to the next $JOB or $END command. Normally, all the commands submitted by one user are processed as a single job, and all output from these commands appears under one job header.

After BATCH encounters a $JOB command, it scans the input file until the next OS/78 command is read. Any lines that follow the $JOB command and precede the first command are written onto the log and ignored by BATCH.

The first character of every command is a period (.). An exception to this is the use of an asterisk (*) whose function is to accept a command string that indicates input/output files. It is further described in the example program given in Section 8.3. The rest of the line contains a command, which should appear in standard OS/78 format. However, commands that would be terminated with an ESCape under interactive OS/78 should be terminated with a dollar sign under BATCH. Most OS/78 commands are legal input to BATCH except the MEMORY and SQUISH SYS: commands. ODT will go to the terminal for input instead of the BATCH file. It is not usually meaningful to invoke ODT, EDIT, or other interactive programs under BATCH.

BATCH executes an OS/78 command by stripping off the initial period (.) character and passing the remainder of the line to the monitor. BATCH then passes control to the monitor, which executes the command as though it had been typed at the keyboard. Monitor commands that return control to the monitor level should be followed by a BATCH command or another monitor command. The SUBMIT command can be used to chain from one BATCH stream to another.

The general rules and conventions associated with BATCH processing are as follows:

1. The dollar sign ($) is always in the first character position of the BATCH command lines $JOB, $MSG and $END.

2. Each job must have a $JOB and $END command.

3. OS/78 commands can be spelled out entirely or in accordance with the accepted abbreviations. Also, a period (.) must precede every command.

4. Wildcards can be specified only for the OS/78 commands COPY, DELETE, DIRECT, LIST, RENAME and TYPE.

5. Comments may only be included as separate comment lines.

6. Only 80 characters per control statement are allowed.

7. Continuation file specification lines (where necessary) are specified by an asterisk at the beginning of the line:

```
.LOAD PROG,SUBA,SUBB$
*SUBC,SUBD,SUBE$
```

8.3 THE BATCH INPUT FILE

The following file is an example of a BATCH input file.

```
$JOB
.DATE 13-MAY-79
/LIST ANY HELP FILES AND STARTING BLOCKS
.DIR *.HL/B
$MSG INSERT DISKETTE INTO DRIVE 1 - TYPE ANY KEY TO CONTINUE
/LIST DIRECTORY OF DISKETTE 2 - SPOOL TO FILE BTCHA1
.DIR TTY:<RXA1:
/COPY FILE FROM DISKETTE 2 DISKETTE 1
.COPY RXA0:<RXA1:POWER.FT
/COMPILE FORTRAN PROGRAM POWER.FT
.COMPILE POWER.FT
/LOAD FORTRAN PROGRAM AND CHAIN TO RUN TIME SYSTEM
.LOAD POWER.RL/G$
/STORE RESULTS OF FORTRAN PROGRAM IN FILE 'HOLD'
*HOLD.TM</4$
/EXECUTE FORTRAN PROGRAM AND DISPLAY RESULTS ON SCREEN
.EXE POWER.LD
/END OF JOB NO.1 - START NEXT JOB
$JOB
/ASSEMBLE FILE SAMPLE AND PRODUCE CREF LISTING
.PAL SAMPLE/C-LS
/EXECUTE PROGRAM SAMPLE
.EXE SAMPLE.BN
/END OF EXAMPLE AND BATCH INPUT FILE
$END
```

The file was created using the Editor and is named BATSAM.BI. For the example used, both the PAL8 and FORTRAN IV system programs were on diskette RXA0.

BATCH is started using the SUBMIT command and typing

```
.SUBMIT RXA0:<BATSAM
```

when spooling is desired, or

```
.SUBMIT BATSAM
```

when spooling is not desired. The default extension for the BATCH input file is .BI.

BATCH begins processing by printing a job header and executing the DATE command. BATCH next executes the DIRECT command, which displays any Help file names and their starting blocks. Wildcards are used in this command.

BATCH continues to scan the input file for the next line. BATCH processes the $MSG command, sounds the terminal buzzer, and copies the $MSG record onto the terminal. Assuming that an operator is present, processing is suspended until any key is typed at the terminal.

The operator performs the task specified by the $MSG command (that of placing diskette 2 into Drive 1) and types any key, allowing BATCH to continue reading the input file.

BATCH then executes the DIRECT command, which lists the directory of RXA1 on the line printer. Since spooling is active because RXA0 was specified as the spooling device in the SUBMIT command line and a non-file-structured device is specified as output in the DIRECT command, BATCH intercepts this output and stores it in a temporary file on the spool device. The output is stored in a file named BTCHA1. BATCH then outputs the message

```
#SPOOL TO FILE BTCHA1
```

on both the console terminal and the line printer, if available. If another file is routed to the spool device, it will be assigned the file name BTCHA2, and any successive files will be named in the following sequence:

```
BTCHA3
  .
  .
  .
BTCHA9
BTCHB0
  .
  .
  .
BTCHZ9
```

A total of 260 spool files are permitted. If output to a spool file is generated by a program that appends a default extension to output file names, the spool file will be assigned a standard default extension. All of the spool files may then be transferred to the terminal or line printer by using the TYPE or LIST command with the input file specification dev:BTCH??.*.

You may type CTRL/C at any time during a batch processing run. Typing CTRL/C at the program level causes an effective jump to location 07600, which recalls the BATCH program. BATCH program then recognizes the CTRL/C and terminates the BATCH run. Sometimes two CTRL/C's are required to be typed in succession to return to the monitor.

Continuing with our example program, the next command copies the FORTRAN program POWER.FT from Drive 1 to Drive 0. The program then is compiled creating a module POWER.RL that is used by the LOAD command to generate a loader image file. The LOAD command is given the /G option, which chains to the run-time system for program execution.

Note that the ESCape function, specified via a dollar sign ($), calls a system program called the Command Decoder. This program (which normally displays an asterisk when it is running) allows the run-time system to accept file input/output specifications. Similarly, an asterisk is used for specifying input/output files. In this case, a file HOLD.TM is designated to store the output of the executed FORTRAN program. The file specification line is then followed by a dollar sign (again serving as an ESCape key function) to execute the LOAD command. After the execution of this command, the loader image file POWER.LD has been created, and its executable results stored in the HOLD.TM file. The EXECUTE command executes the program POWER.FT, displaying the results on the terminal.

BATCH then encounters the second $JOB command. The first job is terminated and a new header is printed. The second job calls CREF to assemble a source program named SAMPLE and produce a CREF listing. The results of this command is to produce a binary file called SAMPLE.BN and a CREF listing stored in the file SAMPLE.LS. Typing the TYPE or LIST commands will print this file on the terminal or line printer, respectively. The program is then executed. The BATCH job is terminated upon encountering the $END command and control returns to the monitor.

The BATCH $END command always must appear as the last record in the input file to terminate batch processing and cause BATCH to recall the monitor and re-establish interactive processing under OS/78.

8.4 BATCH ERROR MESSAGES

BATCH generates two types of error messages. They are BATCH error messages and system error messages. BATCH level error messages appear in the form:

```
#BATCH ERR
```

System error messages are generated by OS/78 system programs. When these occur, BATCH will append a "#" character to the beginning of the message, so that it appears in the form:

```
#SYSTEM ERROR
```

Any occurrence of an error normally causes BATCH to terminate the current job and scan the input file for the next $JOB command. If the /E option was specified, BATCH treats errors as non-fatal and continues the BATCH run.

Table 8-3 lists the BATCH error messages, their meanings, and the probable cause for the error.

Table 8-3
BATCH Error Messages

Message	Meaning
#BAD LINE JOB ABORTED	The BATCH program detected a line in the input file that did not have one of the characters period (.), slash, dollar sign or asterisk as the first character of the record. The record is ignored, and BATCH scans the input file for the next $JOB command.
%BATCH SQUISHING SYS:!	BATCH is running and attempting to execute a SQUISH on the system device. This may cause BATCH.SV or the BATCH input file to be moved.
BATCH.SV NOT FOUND	A copy of BATCH.SV must exist on the system device. Control returns to the monitor.
DEV NOT IMPLEMENTED	BATCH cannot accept input from the specified input device because its handler is not permanently resident. Only input from SYS: is permitted. Control returns to the Command Decoder (See Appendix D). In most cases, type CTRL/C and then retype the command, using the correct parameters.
#ILLEGAL INPUT	A file specification designated TTY as an input device when the /U option indicated that an operator is not available. The current job is terminated, and BATCH scans the input file for the next $JOB command.
ILLEGAL SPOOL DEVICE	The device specified as a spooling output device must be file-structured. Control returns to the Command Decoder (See Appendix D). In most cases, type CTRL/C and then retype the command, using the correct parameters.
#INPUT FAILURE	Either a hardware problem prevented BATCH from reading the next line of the input file, or BATCH read the last line of the input file without encountering a $END command line. If a hardware problem exists, correct the problem and type any character at the terminal to resume processing.

(continued on next page)

Table 8-3 (Cont.)
BATCH Error Messages

Message	Meaning
#MANUAL HELP NEEDED	BATCH is attempting to operate an I/O device, such as TTY, that will require operator intervention. If the /U option was specified to indicate that an operator is not present, this message is suppressed, the current job is terminated, and BATCH scans the input file for the next $JOB command record. If an operator is present, the $MSG command provides notification of the action that should be taken.
#MONITOR OVERLAYED	The BATCH program was called to accept and transmit a file specification, but found that a user program had overlaid part or all of BATCH. BATCH then executes the next command.
#SPOOL TO FILE BTCHA1	Where the "A" may be any character of the alphabet and the "1" may be any decimal digit. This message indicates that BATCH has intercepted a non-file-structured output file and routed it to the spool device. This in not, generally, an error condition. Spool device file names are assigned sequentially, beginning with file BTCHA1. Standard default extensions may be assigned by some system programs.
#SYS ERROR	A hardware problem prevented BATCH from performing an I/O operation. Program execution halts, and the system must be restarted by pressing the START pushbutton.

8.5 RESTRICTIONS UNDER OS/78 BATCH

OS/78 BATCH is unprotected from user errors. The BATCH program resides in locations 5000 to 7577 in the highest memory field available. BATCH also uses the following locations in field 0 and the highest memory field available.

LOCATION	USED AS:
07777	Batch processing flag.
ff7774-ff7777	Internal pointers (ff = field number).

Both the monitor and the Command Decoder (see Appendix D) check the batch processing flag (bit 1 of 07777) whenever they are entered from the program level. Any user program that modifies location 07777 may cause batch processing to be terminated prematurely before the next line of the BATCH input file is read unless bit 1 is left alone.

When the monitor is entered from the program level (JMP to 07600 or 07605), it checks the batch processing flag and reads a new copy of the BATCH program into memory if batch processing is in progress. The Command Decoder, however, does NOT perform this operation. Thus, the Command Decoder must not be called unless the BATCH program is already in memory.

Therefore, large user programs may be loaded over the BATCH program as long as they do not modify the last four locations in field 3. However, once a user memory load has overwritten the BATCH program, execution must remain at the program level until the monitor has been re-entered and a new copy of the BATCH program is read into memory. The Command Decoder must not be called after a user program has been loaded over the BATCH program.

In general, this restriction applies only to loader programs and only when the loader calls the Command Decoder more than once while building a large memory load. Multiple calls to the Command Decoder may be avoided when loading large programs during batch processing if the memory load is first built in the monitor environment and then saved with the SAVE command for subsequent execution under BATCH.

However, the memory image of any program that overlays the BATCH program cannot be saved with the SAVE command. After the load operation but before the save is executed, BATCH will be read back into memory, destroying part of the user program. Thus, the monitor SAVE operation will cause part of the BATCH program to be saved instead of that part of the user program which originally overlaid the BATCH program.

A BATCH job must never move or delete either BATCH.SV or the BATCH input file. Also, SYS: should never be SQUISHed while BATCH is running since this may cause these files to be moved to another region on the system device.

CHAPTER 9

OCTAL DEBUGGING TECHNIQUE (ODT)

ODT is an interactive system program that makes debugging PAL8 programs easy by providing selective execution and memory examination, modification, and searching facilities.

9.1 ODT FEATURES

ODT features include examination and modification of memory locations, and the use of instruction breakpoints to return control to ODT. ODT makes no use of the program interrupt facility and, with few restrictions, is invisible to a user program.

The breakpoint is one of ODT's most useful features. To accomplish this, ODT acts as a monitor to the user program and permits you to insert ODT-controlled halts (breakpoints) in your program. You decide how far the program is to run and instruct ODT to insert a breakpoint in the program which, when encountered, transfers control to transfer back to ODT. ODT immediately preserves in its internal storage locations the contents of the AC and L at the breakpoint. It then displays the location at which the breakpoint occurred, as well as the contents of the AC at that point. ODT will allow examination and modification of any location of the program (or those ODT locations containing the AC and L). You can move the breakpoint or delete it, and request that ODT continue running the program. This causes ODT to restore the AC and L, execute the "trapped" instruction and continue in the program until the breakpoint is again encountered, or the program is terminated.

9.2 CALLING AND USING ODT

When ODT is being used, a complete assembly listing of the program should be available for the program that is being debugged.

Before calling ODT, place the program that is to be debugged in memory as the "current program". For example, if it is a memory image file, type

```
.GET SYS SAMPLE

.ODT
```

If it is a binary file, type

```
.LOAD SAMPLE

.ODT
```

Little of the memory that is being used is disturbed by the running of ODT, because the sections of the program which ODT may occupy when in memory are preserved on the system device and swapped back into memory as necessary. ODT uses the Job Status Word of the particular program to determine whether or not swapping should occur. If the program does not use locations 0-1777 in field 0, less swapping occurs during use of the breakpoint feature.

If any amount of a program is typed directly into memory (in octal), the Core Control Block of the program may not reflect the true extent of the program. If octal additions are made below location 2000 in field 0, ODT may give erroneous results. This condition is corrected by changing the Job Status Word, which is stored in location 7746 of field 0, and which can be examined and changed using ODT as explained later in this chapter. Location 7745 of field 0 is the 12-bit starting address of the program in memory and location 7744 contains the field designation in the form 62n3, where n is the field designation of the starting address.

When using the breakpoint feature of ODT, keep certain operating characteristics in mind:

1. If a breakpoint is inserted at a location which contains an auto-indexed instruction, the auto-index register is incremented immediately after the breakpoint. Thus, when control returns to the user in ODT, the register will have been increased by one. The breakpoint instruction is executed properly, but the index register, if examined, will be one greater than it should.

2. ODT keeps track of the terminal display I/O flag and restores the terminal flag when it continues from a breakpoint.

3. The breakpoint feature uses and does not restore locations 4, 5, and 6 in the memory field in which the breakpoint is set.

4. The breakpoint feature of ODT uses the table of user-defined device names as scratch storage, destroying any device names that may have been created and creating garbage entries. Therefore, it is advisable not to use user-defined device names in programs being debugged with ODT breakpoints. After a session with ODT in which breakpoints are used, give a DEASSIGN command to clear out the user-device name table.

5. Breakpoints must not be set in the monitor, in the device handlers, or between a CIF and the following JMP or JMS instruction.

6. ODT should not be used to debug programs which use interrupts.

If an operation is attempted in non-existent memory, ODT ignores the command and types "?". Thus, attempting to examine locations in field 4 and above, ODT responds with ?.

Typing CTRL/C returns control to the monitor. You can then save your program on any file-structured device with the SAVE command. If you want to do this, do not issue any other commands before the SAVE command.

9.3 ODT COMMANDS

9.3.1 Special Characters

These characters will be illustrated using the SAMPLE.PA program created in Chapter 4.

9.3.1.1 **Slash (/) - Open This Location** - The location examination character (/) causes the location addressed by the octal number preceding the slash to be opened for modification and its contents are displayed in octal. The open location can then be modified by typing the desired octal number and closing the location by using the RETURN key. Any octal number from 1 to 6 digits in length is legal input for addresses and from 1 to 4 digits for data. If the number is not octal or if the DELETE key is typed, a question mark (?) is displayed and the number is ignored. If more than the maximum number of digits are entered, ODT accepts only the last 6 entered (for addresses) or the last 4 (for data). Typing / with no preceding argument reopens the most recently opened location. For example,

```
200/7300 (RET)
000201/6046 (RET)
000201/6046 6048?
000201/6046 2345 (RET)
/2345
```

The above example modifies locations in field 0. If locations are examined in other fields, specify the location as ffnnnn/ where ff is the field (0-37) and nnnn is the location in octal.

9.3.1.2 **RETURN - Close Location** - If you have typed a valid octal number after the content of a location is displayed by ODT, pressing the RETURN key causes that number to replace the original contents of the opened location and the location to be closed. If you type nothing, the location is closed but the contents of the location are not changed. For example,

```
201/6046 (RET)          location 201 is unchanged.
000201/6046 2345 (RET)  location 201 is changed to contain 2345.
/2345 6046              place 6046 back in location 201.
```

Typing another command will also close an opened register.

9.3.1.3 **LINE FEED - Close Location, Open Next Location** - The LINE FEED key has the same effect as the RETURN key, but, in addition, the next sequential location is opened and its contents displayed. For example,

```
200/7300 (LF)           location 200 is closed unchanged and 201
                        is opened. Type change.
000201 /2346 6046 (LF)  201 is closed (containing 6046) and 202
                        is opened.
000202 /1216
```

9.3.1.4 **Semicolon(;) - Change Location, Close It and Open Next** - The semicolon inserts the octal value (if any) that follows it into the currently opened location, closes that location and opens the next sequential location for modification. For example,

```
202/1216 1234;5670 (RET)
000202/1234
000203/5670
```

A series of octal values can be deposited sequentially using the semicolon character. Typing multiple semicolons skips a memory location for each semicolon typed and will accept an octal value at the location skipped to. For example,

```
202/1234 1216;;;0000 (RET)
000202/1216
000203/5670
000204/6041
000205/0000
```

9.3.1.5 **nnnn+ and nnnn- - Open Location Relative to Another** - The nnnn+ and nnnn- commands open the current location plus or minus an offset value for modification and prints the content of that location. If n is omitted, it is assumed to be 1. For example,

```
200/7300 3+
000203/5670 3217
```

or

```
207/6046 2-
000205/0000 5204
```

9.3.1.6 **Circumflex(^) - Close Location, Take Contents as Memory Reference Instruction and Open Effective Address** - The circumflex will close an open location just as will the RETURN key. Further, it will interpret the contents of the location as a memory reference instruction, open the location referenced and display its contents. For example,

```
202/1216 ^              1216 symbolically is "TAD, this page,
                         relative location 16," so ODT opens
                         location 216.
000216 /0220
```

The indirect bit is used in determining the effective address.

9.3.1.7 **Underline(_) - Close Location, Open Indirectly** - The back arrow will close the currently open location and then interpret its contents as the address of the location (in the same field) whose contents it is to print and open for modification. For example,

```
202/1216 _
000216 /0220
000220 /0215
```

9.3.2 Illegal Characters

Any character that is neither a valid control character nor an octal digit causes the current line to be ignored and a question mark printed. For example,

2:? 2U?	ODT opens no location.
206/1617 67K?	ODT ignores a partial number and closes location 206.
/1617	

9.3.3 Control Commands

9.3.3.1 **ffnnnnG - Transfer Control to Program at Location nnnn of Field ff** - Clear the AC and L then go to the location ffnnnn. The breakpoint, if any, will be inserted. Typing G alone will cause a start at location 00000.

9.3.3.2 **ffnnnnB - Set Breakpoint at Location nnnn of Field ff** - Instructs ODT to establish a breakpoint at the location ffnnnn. A breakpoint may be changed to another location whenever ODT is in control, by simply typing ffnnnnB where ffnnnn is the new location. Only one breakpoint may be in effect at a time; therefore, requesting a new breakpoint removes any previously existing one. A breakpoint may be established at location 000000.

The breakpoint (B) command does not make the actual exchange of ODT instruction for user instruction, it only sets up the mechanism for doing so. The actual exchange occurs when a G or a C command is executed.

When, during execution, the program encounters the location containing the breakpoint, control passes immediately to ODT (via locations 0004-0006 of the instruction field). The contents of the accumulator and contents of the link at the point of the interruption are saved in special locations accessible to ODT. The user instruction that the breakpoint was replacing is restored, before the address of the trap and the content of the AC are displayed. The restored instruction has not been executed at this time, and will not be executed until the "continue from breakpoint" command is given. Any location used, including those containing the stored accumulator and Link, can now be modified. The breakpoint can also be moved or removed at this time.

9.3.3.3 **B - Remove Breakpoint** - Typing B alone removes any previously established breakpoint and restores the actual contents of the breakpoint location. B should be typed before saving the current program if a breakpoint was set. Typing B where no breakpoint is set has no effect.

NOTE

If a breakpoint set by ODT is not encountered while ODT is running the object (user's) program, the instruction which causes the break to occur will not be removed from the user's program.

9.3.3.4 **A - Open (AC) Location** - When the breakpoint is encountered the contents of the accumulator and the contents of the link are saved for later restoration. Typing A after having encountered a breakpoint opens for modification the location in which the AC was saved and prints its contents. This location may now be modified in the normal manner (see Slash) and the modification will be restored to the accumulator when the C or G command is given (contains either 0000 or 0001).

9.3.3.5 **L - Open C(L) Location** - Typing L opens the link storage location for modification and prints its contents. The link location may now be modified as usual (see Slash) and that modification will be restored to the Link when the C or G command is given.

9.3.3.6 **C - Continue From a Breakpoint** - Typing C, after having encountered a breakpoint, causes ODT to restore the contents of the accumulator, link, and breakpoint location, and transfer control to the breakpoint location. The user program then runs until the breakpoint is again encountered or the program halts or exits to the monitor.

9.3.3.7 **nnnnC - Continue nnnn+1 Times from Breakpoint** - A breakpoint may be established at some location within a loop of a program. Since loops often run to many iterations, some means must be available to prevent a break from occurring each time the break location is encountered. This is the function of nnnnC (where nnnn is an octal number). The "continue" operation is done nnnn+1 times. If nnnn is omitted, 0 is assumed. Thus, 2C will allow a loop in which the breakpoint is embedded to execute three times.

Given the following program ADD1, which increases the value of the accumulator by increments of 1, the use of the Breakpoint and Continue commands may be illustrated.

```
*200

            0200      *200
000200      7300              CLA CLL
000201      1206      A,      TAD ONE
000202      2207      B,      ISZ CNT
000203      5202              JMP B
000204      5201              JMP A
000205      7402              HLT
000206      0001      ONE,    1
000207      0000      CNT,    0
```

Assemble the program and call ODT by typing the following:

```
.PAL ADD1/L

.ODT
000201B
000200G
000201 (0;0000
C
000201 (0;0001
C
000201 (0;0002
4C
000201 (0;0006
```

ODT has been loaded and started. A breakpoint is inserted at location 0201 and execution stops here showing the accumulator initially set to 0000. The use of the Continue command (C) executes the program until the breakpoint is again encountered (after one complete loop) and shows the accumulator to contain a value of 0001. Again execution continues, incrementing the AC to 0002. At this point, the command 4C is used, allowing execution of the loop to continue five more times before stopping at the breakpoint. The contents of the accumulator have now been incremented to 0006.

9.3.3.8 **D - Open Data Field** - Typing D opens for modification the internal ODT location containing the data field which was in effect at the last breakpoint. Contents of D always appears as a value between 0 and 37.

9.3.3.9 **F - Open Current Field** - Typing F opens for modification the internal ODT location containing the field used by ODT in the W (search) command, in the and ^ (indirect addressing) commands, or in the last breakpoint (depending upon which was used most recently). A value between 0 and 37.

9.3.3.10 **M - Open Search Mask** - Typing M causes ODT to open for modification the internal ODT location containing the current value of the search mask and print its contents. Initially the mask is set to 7777. It may be changed by opening the mask location and typing the desired value after the value printed by ODT, then closing the location.

9.3.3.11 **M (LF) - Open Lower Search Limit** - The word immediately following the mask storage location contains the location at which the search is to begin. Pressing the LINE FEED key to close the mask location causes the lower search limit to be opened for modification and its contents displayed. Initially the lower search limit is set to 0000. It may be changed by typing the desired lower limit after that printed by ODT, then closing the location.

9.3.3.12 **M (LF)(LF) - Open Upper Search Limit** - The next sequential word after the lower search limit contains the location with which the search is to terminate. Pressing the LINE FEED key to close the lower search limit causes the upper search limit to be opened for modification and its contents printed. Initially, the upper search limit is 7577. It may be changed by typing the desired upper search limit after the one printed by ODT, then closing the location with the RETURN key.

9.3.3.13 **nnnnW - Word Search** - The command nnnnW (where nnnn is an octal number) will cause ODT to conduct a search of a defined section of memory, using the mask and the lower and upper limits that the user has specified, as indicated above. The word searching operation determines if a given quantity is present in any of the locations of that defined section of memory. A search never alters the contents of any location.

The search is conducted as follows. ODT masks each location within the limits specified and compares the result to the quantity for which it is searching. If the two quantities are identical, the address and the actual unmasked contents of the matching location are displayed, and the search continues until the upper limit is reached. The search includes the upper limit.

For example, locations 3000 through 3777 are to be searched for all ISZ instructions, regardless of what location they refer to (that is, search for all locations beginning with an octal 2, which is the operation code for an ISZ instruction).

M/7777 7000(LF)	Change the mask to 7000; open lower search limit.
000041/5273 3000(LF)	Change the lower limit to 3000; open upper limit.
000042/1335 3777(RET)	Change the upper limit to 3777; close location.
2000W	Initiate the search for ISZ instructions.
0000005 /2331	There are four ISZ instructions in this section of memory. Note that these might also be data values that happen to start with an octal 2.
0000006 /2324	
0000011 /2222	
0000033 /2575	

9.4 ERRORS

The only legal inputs are command characters and octal digits. Any other character will cause the character or line to be ignored and a question mark to be printed by ODT. When G is used, it must be preceded by an address to which control will be transferred. If G is not preceded by an address, a question mark will not be displayed, but control will be transferred to location 0.

9.5 PROGRAMMING NOTES

ODT will not turn on the program interrupt. It does, however, turn off the interrupt when a breakpoint is encountered, to prevent spurious interrupts.

Breakpoints are fully invisible to "open location" commands; however, breakpoints must not be placed in locations which the user program will modify in the course of execution or the breakpoint may be destroyed. Caution should be used in placing a breakpoint between a call to USR function code 10 (USRIN) and the following call to USR function code 11 (USROUT).

If a trap set by ODT is not encountered by your program, the breakpoint instruction will not be removed. If the SAVE command is to be used, the B command must be used to remove the breakpoint before typing CTRL/C. ODT should not be run under BATCH.

9.6 ODT COMMAND SUMMARY

Table 9-1 presents a brief summary of the ODT commands.

Table 9-1
ODT Command Summary

Command	Meaning
ffnnnn/	Opens location designated by the octal number ffnnnn, where ff represents the memory field (0-37). ODT displays the contents of the location, a space, and waits for the user to enter a new value for that location or close the location. If you omit ff, field 0 is assumed.
/	Reopens the latest opened location.
nnnn; or nnnn (LF)	Deposits nnnn in the currently opened location, close that location and open the next location in sequence for modification. The semicolon (;) lets you deposit a series of octal values in sequential locations. To skip locations in the sequence, type a semicolon for each location that you want to skip.
RETURN key	Closes the currently open location, if any.
LINE FEED key	Closes the currently open location, opens the next sequential location for modification, and displays the contents of that location.
nnnn+	Opens the current location plus n and displays the contents of that location.
nnnn-	Opens the current location minus n and displays its contents.

(continued on next page)

Table 9-1 (Cont.)
ODT Command Summary

Command	Meaning
^(Circumflex)	Closes the current location, reads its contents as a memory-reference instruction and opens the location it points to, displaying its contents. • ODT makes no distinction between instruction op-codes when you use this command. It treats all op-codes as memory-reference instructions. • Take care when you use this command with indirectly referenced auto-index registers. If you use the command in this way, the contents of the auto-index register is incremented by one. Check to see that the register contains the proper value before proceeding.
_(Underline)	Closes the current location, takes the contents of the current location as a 12-bit address and opens that address for modification, displaying its contents.
ffnnnnG	Transfers control of the program to location ffnnnn, where ff represents the memory field.
ffnnnnB	Establishes a breakpoint at location ffnnnn, where ff represents the memory field. Only one breakpoint is allowed at any given time.
B	Removes the breakpoint, if any.
A	Opens for modification the location in which the contents of the accumulator were stored when the breakpoint was encountered.
L	Opens for modification the location in which the contents of the link were stored when the breakpoint was encountered.
C	Continues from a breakpoint.
nnnnC	Continues from a breakpoint nnnn+1 times before interrupting the user's program at the breakpoint location.
M	Opens the search mask, initially set to 7777, which can be changed by typing a new value.

(continued on next page)

Table 9-1 (Cont.)
ODT Command Summary

Command	Meaning
M (LF)	Opens the lower search limit. Type in the location (four octal digits) where the search will begin.
M (LF) (LF)	Opens the upper search limit. Type in the location (four octal digits) where the search will terminate.
nnnnW	Searches the portion of memory as defined by the upper and lower limits for the octal value nnnn. Search can only be done on a single memory field at a time. See the F command.
D	Opens for modification the word containing the data field (0-37) that was in effect at the last breakpoint. To change the field, enter a field number in the range 0-37.
F	Opens for modification the word containing the field (0-37) used by ODT in the W (search) command, in the ^ and (indirect addressing) commands, or in the last breakpoint (depending upon which was used most recently). To change the field, enter a field number in the range 0-37.
CTRL/O	Interrupts a long search output and waits for a new ODT command.
DELETE key	Cancels previous number typed, up to the last non-numeric character entered. ODT responds with a question mark, after which you enter the correct location value.

APPENDIX A

ASCII CHARACTER SET

The following table shows the standard ASCII character set used by OS/78 software. Control codes marked with an asterisk (*) have no significance to OS/78.

7-Bit Decimal	8-Bit Octal	6-Bit Octal	Character/ Control Code	Remarks
000	200		NUL	Fill character - ignored on input
001	201		SOH *	
002	202		STX *	
003	203		ETX	CTRL/C - Return control to monitor
004	204		EOT *	
005	205		ENQ *	
006	206		ACK *	
007	207		BEL	CTRL/G - Sound audible signal
008	210		BS *	
009	211		HT	CTRL/I - Horizontal tab
010	212		LF	CTRL/J - LINE FEED (new line)
011	213		VT	CTRL/K - TAB (vertical)
012	214		FF	CTRL/L - Form feed
013	215		CR	Cursor or carriage RETURN (CTRL/M)
014	216		SO *	
015	217		SI	CTRL/O - Abort current terminal output
016	220		DLE *	
017	221		DC1	CTRL/Q - Disable terminal output
018	222		DC2 *	
019	223		DC3	CTRL/S - Enable terminal output
020	224		DC4 *	
021	225		NAK	CTRL/U - Delete current input line
022	226		SYN *	
023	227		ETB *	
024	230		CAN *	
025	231		EM *	
026	232		SUB	CTRL/Z - End of file
027	233		ESC	ALTMODE
028	234		FS *	
029	235		GS *	
030	236		RS *	
031	237		US *	
032	240	40	SP	Space

ASCII CHARACTER SET

7-Bit Decimal	8-Bit Octal	6-Bit Octal	Character/ Control Code	Remarks
033	241	41	!	Exclamation point
034	242	42	"	Quotation mark
035	243	43	#	
036	244	44	$	Dollar sign
037	245	45	%	Percent sign
038	246	46	&	Ampersand
039	247	47	'	Apostrophe
040	250	50	(	Left parenthesis
041	251	51	)	Right parenthesis
042	252	52	*	Asterisk
043	253	53	+	Plus sign
044	254	54	,	Comma
045	255	55	-	Hyphen or minus sign
046	256	56	.	Period
047	257	57	/	Slash (right)
048	260	60	0	
049	261	61	1	
050	262	62	2	
051	263	63	3	
052	264	64	4	
053	265	65	5	
054	266	66	6	
055	267	67	7	
056	270	70	8	
057	271	71	9	
058	272	72	:	Colon
059	273	73	;	Semicolon
060	274	74	<	Left angle bracket or less than sign
061	275	75	=	Equals sign
062	276	76	>	Right angle bracket or greater than sign
063	277	77	?	Question mark
064	300	00	@	At sign (null in 6-bit)
065	301	01	A	
066	302	02	B	
067	303	03	C	
068	304	04	D	
069	305	05	E	
070	306	06	F	
071	307	07	G	
072	310	10	H	
073	311	11	I	
074	312	12	J	
075	313	13	K	
076	314	14	L	
077	315	15	M	
078	316	16	N	
079	317	17	O	
080	320	20	P	
081	321	21	Q	
082	322	22	R	
083	323	23	S	
084	324	24	T	
085	325	25	U	
086	326	26	V	
087	327	27	W	
088	330	30	X	
089	331	31	Y	
090	332	32	Z	
091	333	33	[	Left square bracket

ASCII CHARACTER SET

7-Bit Decimal	8-Bit Octal	6-Bit Octal	Character/ Control Code	Remarks
092	334	34	\	Backslash
093	335	35	]	Right square bracket
094	336	36	^	Circumflex
095	337	37	--	
096	340	40	`	Grave accent
097	341	41	a	
098	342	42	b	
099	343	43	c	
100	344	44	d	
101	345	45	e	
102	346	46	f	
103	347	47	g	
104	350	50	h	
105	351	51	i	
106	352	52	j	
107	353	53	k	
108	354	54	l	
109	355	55	m	
110	356	56	n	
111	357	57	o	
112	360	60	p	
113	361	61	q	
114	362	62	r	
115	363	63	s	
116	364	64	t	
117	365	65	u	
118	366	66	v	
119	367	67	w	
120	370	70	x	
121	371	71	y	
122	372	72	z	
123	373	73	{	
124	374	74	|	
125	375	75	}	
126	376	76	~	Tilde
127	377	77	DEL	RUBOUT

APPENDIX B

USEFUL MATHEMATICAL SUBROUTINES

Since the DECstation hardware does not contain built-in multiplication or division instructions, you must incorporate these functions explicitly in your assembly language programs. The subroutines listed in this appendix illustrate efficient ways to do these calculations. These routines perform single-precision, unsigned calculations. this means that all numbers are positive values represented by 12-bit words in the range from 0 to 7777 (octal), inclusive. Another important point is that single-precision, unsigned numbers may be regarded as integers or as fractions, depending on whether the binary point (analogous to the decimal point in the decimal number system) is assumed to be located to the right of the right-most bit (in the case of integers) or to the left of the left-most bit (in the case of fractions).

B.1 UNSIGNED INTEGER MULTIPLICATION SUBROUTINE

```
/CALCULATES A*B, WHERE A AND B ARE UNSIGNED INTEGERS.  IF A*B>7777
/A JMP IS MADE TO LOCATION "ERROR".

/CALLING SEQUENCE:
/       JMS ML
/       A
/       B
/CONTROL RESUMES HERE WITH (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION) WITH ACCUMULATOR(AC)=A*B.  A AND B ARE PRESERVED.
/LINK=0 ON RETURN.

/NOTE:  USE REPETITIVE ADDITION INSTEAD OF THIS SUBROUTINE
/IF EITHER ARGUMENT WILL ALWAYS BE LESS THAN ABOUT 50 (OCTAL).

/NOTE:  STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

/NOTE:  REMOVING ALL OVERFLOW TEST INSTRUCTIONS GIVES A SUBROUTINE
/WHICH TRUNCATES THE RESULT, RETURNING A*B MOD 10000.

ML,         0
            CLA             /* IGNORE AC
            TAD I ML        /GET MULTIPLIER
            ISZ ML
            DCA ML2
            TAD KM14
            DCA COUNT
ML1,        CLL RAL         /ACCUMULATE LSB(PRODUCT)
            DCA PROD
            SZL             /* OVERFLOW TEST
            JMP ERROR       /*
```

```
                TAD ML2         /GET NEXT BIT OF MULTIPLIER
                RAL
                DCA ML2
                SZL             /IF SET, ADD MULTIPLICAND INTO
                                /THE PARTIAL PRODUCT
                TAD I ML        /MULTIPLICAND
                CLL             /* OVERFLOW TEST
                TAD PROD        /PARTIAL PRODUCT
                ISZ COUNT       /LOOP FOR 12 BITS
                JMP ML1
                ISZ ML
                SNL             /* OVERFLOW TEST
                JMP I ML        /RETURN
                CLA             /*
                JMP ERROR       /*

/VARIABLES

ML2,            0               /MULTIPLIER
PROD,           0               /PARTIAL PRODUCT
COUNT,          0
KM14,           -14
```

B.2 UNSIGNED FRACTIONAL MULTIPLICATION SUBROUTINE

```
/CALCULATES A*B/10000, WHERE A AND B ARE UNSIGNED INTEGERS.

/CALLING SEQUENCE:
/       JMS ML
/       A
/       B
/CONTROL RESUMES HERE WITH AC=A*B/10000.
/A AND B ARE PRESERVED.

/NOTE: THERE ARE NO ERROR CONDITIONS.

ML,             0
                CLA             /* MAY BE REMOVED IF AC=0 ON ENTRY
                TAD I ML        /GET MULTIPLIER
                ISZ ML
                DCA ML2
                TAD KM14
ML1,            DCA PROD
                TAD ML2         /GET NEXT BIT OF MULTIPLIER
                CLL RAR
                DCA ML2
                SZL             /IF SET, ADD MULTIPLICAND INTO
                TAD I ML
                CLL
                TAD PROD        /THE PARTIAL PRODUCT
                RAR             /KEEP MSB (PROD) ONLY
                ISZ COUNT       /LOOP FOR 12 BITS
                JMP ML1
                ISZ ML
                JMP I ML        /RETURN

/VARIABLES

ML2,            0               /MULTIPLIER
PROD,           0               /PARTIAL PRODUCT
COUNT,          0
KM14,           -14
```

B.3 UNSIGNED INTEGER DIVISION SUBROUTINE

```
/CALCULATES A/B, WHERE A AND B ARE UNSIGNED INTEGERS SUCH THAT
/B IS NOT E.O. IF B=0 THEN A JMP IS MADE TO LOC "ERROR".
/QUOTIENT IS RETURNED IN AC, REMAINDER IS AVAILABLE IN
/LOCATION "REMAIN".

/CALLING SEQUENCE:
/       JMS DV
/       A
/       B
/CONTROL RESUMES HERE (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION) WITH AC=QUOTIENT(A/B), REMAIN=REMAINDER(A/B).
/A AND B ARE PRESERVED.  LINK=0 ON RETURN.

/NOTE: STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

DV,         0
            CLA             /* IGNORE AC
            TAD I DV        /GET ARGS
            ISZ DV
            DCA DA          /DIVIDEND
            TAD I DV
            ISZ DV
            SNA             /* B = 0 TEST
            JMP ERROR       /*
            CLL CIA         /SUBTRACTION WILL BE DONE BY ADDING
            DCA DB          /-DIVISOR
            DCA REMAIN      /CLEAR MSB (DIVIDEND)
            CLL CLA CMA RAL /SET INITIAL VALUE OF QUOTIENT TO 7776. AS
                            /QUOTIENT IS SHIFTED LEFT, THE 0 BIT WILL SHI
                            /LEFT, SERVING AS A FLAG TO STOP THE DIVISION
                            /LOOP AFTER ALL 12 BITS ARE COMPUTED.
DV1,                        /MAIN LOOP: COMPUTE NEXT BIT OF PARTIAL
            DCA QUOT        /QUOTIENT INITIALIZE OR STORE PARTIAL
            TAD DA          /QUOTIENT SHIFT REMAINDER LEFT TO NEXT BIT
            CLL RAL
            DCA DA
            TAD REMAIN      /ONE BIT INTO MSB(DIVIDEND)
            RAL
            DCA REMAIN
            TAD REMAIN      /TRIAL SUBTRACTION
            TAD DB          /SUBTRACT DIVISOR FROM DIVIDEND
            SZL             /STORE BACK REMAINDER ONLY IF .GE. 0
            DCA REMAIN
            CLA             /LINK=1 IFF SUBTRACT SUCCEEDED
            TAD QUOT
            RAL             /ROTATE LINK INTO PARTIAL QUOTIENT
            SZL             /DO LOOP ONCE FOR EACH BIT
            JMP DV1
            JMP I DV        /RETURN
/VARIABLES

DA,         0               /LSB(DIVIDEND)
REMAIN,     0               /MSB(DIVIDEND), REMAINDER
DB,         0               /-DIVISOR
QUOT,       0               /PARTIAL QUOTIENT
```

B.4 UNSIGNED FRACTIONAL DIVISION SUBROUTINE

```
/CALCULATES 10000*A/B, WHERE A AND B ARE UNSIGNED INTEGERS SUCH THAT
/B .NE. 0 AND A<B. IF B=0 OR A .GE. B THEN A JMP IS MADE TO LOCATION *ER

/CALLING SEQUENCE:
/       JMS DV
/       A
/       B
/CONTROL RESUMES HERE (I.E., AT THE THIRD LOCATION AFTER THE
/JMS INSTRUCTION) WITH AC=QUOTIENT (10000*A/B).
/A AND B ARE PRESERVED.  LINK=0 ON RETURN.

/NOTE: STARRED INSTRUCTIONS MAY BE REMOVED IF THEIR EFFECTS
/ARE NOT NEEDED.

DV,         0
            CLA                 /* IGNORE AC
            TAD I DV            /GET ARGS
            ISZ DV
            DCA DA              /DIVIDEND
            TAD I DV
            ISZ DV
            SNA                 /* B = 0 TEST
            JMP ERROR           /*
            CLL CIA             /SUBTRACTION WILL BE DONE BY ADDING
            DCA DB              /-DIVISOR
            TAD DA              /* OVERFLOW TEST
            TAD DB              /*
            SZL CLA             /*
            JMP ERROR           /*
            CLL CLA CMA RAL     /SET INITIAL VALUE OF QUOTIENT TO 7776. AS QU
                                /IS SHIFTED LEFT, THE 0 BIT WILL SHIFT LEFT,
                                /SERVING AS A FLAG TO STOP THE DIVISION LOOP
                                /AFTER ALL 12 BITS ARE COMPUTED.
DV1,                            /MAIN LOOP: COMPUTE NEXT BIT OF PARTIAL
            DCA QUOT            /QUOTIENT INITIALIZE OR STORE PARTIAL
            TAD DA              /QUOTIENT SHIFT REMAINDER LEFT TO NEXT BIT
            CLL RAL
            DCA DA
            TAD DA              /TRIAL SUBTRACTION
            TAD DB              /SUBTRACT DIVISOR FROM DIVIDEND
            SZL                 /STORE BACK REMAINDER ONLY IF .GE. 0
            DCA DA
            CLA                 /LINK=1 IFF SUBTRACT SUCCEEDED
            TAD QUOT
            RAL                 /ROTATE LINK INTO PARTIAL QUOTIENT
            SZL                 /DO LOOP ONCE FOR EACH BIT
            JMP DV1
            JMP I DV            /RETURN

/VARIABLES

DA,         0                   /DIVIDEND (REMAINDER)
DB,         0                   /-DIVISOR
QUOT,       0                   /PARTIAL QUOTIENT
```

APPENDIX C

USER SERVICE ROUTINE

The User Service Routine, or USR, is a collection of subroutines that perform the operations of opening and closing files, loading device handlers, program chaining, and calling the Command Decoder. The USR provides these functions not only for the system itself but for any programs running under the OS/78 system. A summary of USR functions is given in Table C-1.

USR is used only by PAL8 programs. Users who code in BASIC or FORTRAN IV can skip this appendix and Appendix F, "Using Device Handlers".

The USR resides on the system device and, when called, is swapped into memory. Typically, a series of calls is involved in any I/O operation and it is inefficient to repeat the swap every time. Therefore, the calling program can lock USR in memory in the first 2000 (octal) words of field 1, through the use of USRIN function. To perform any operations on files, it is necessary to have the relevant device handlers available.

This is accomplished with the FETCH function which loads a device handler into memory. Since the handlers reside in the calling program, space must be reserved for them. Special characteristics of OS/78 device handlers and their use are described in Appendix F.

An attempt to call the USR with a code greater than 13 (octal) will cause a Monitor Error 4 message to be printed and the program to be aborted.

C.1 CALLING THE USR

Performing any USR function is done by issuing a JMS instruction followed by the proper arguments. Calls to the USR take a standardized calling sequence. This standard call should be studied before progressing to the operation of the various USR functions.

C.1.1 Standard USR Call

In the remainder of this chapter, the following calling sequence is referenced:

TAD VAL	The contents of the AC is applicable in some cases only.
CDF N	Where N is the value of the current program field multiplied by 10 (octal).

CIF 10	The instruction field must be set to 1.
JMS I (USR)	Where USR is either 7700 or 0200, (see Section C.1.2).
FUNCTION	This word contains an integer from 1 to 13 (octal) indicating which USR operation is to be performed as described in the USR Functions Summary Table (Table C-1).
ARG(1), ARG(2), ARG(3)	The number and meaning of these argument words varies with the particular USR function to be performed.
error return	When applicable, this is the return address for any errors.
normal return	The operation was successful. The AC is cleared and the data field is set to current field.

Table C-1
Summary of USR Functions

Function Code	Name	Operation
1	FETCH	Loads a device handler into memory. Return the entry address of the handler.
2	LOOKUP	Searches the file directory on any device to locate a specified permanent file.
3	ENTER	Creates and opens for output a tentative file on a specified device.
4	CLOSE	Closes the currently open tentative file on the specified device, making it a permanent file. Also, any previous permanent file with the same file name and extension is deleted.
5	DECODE	Calls the Command Decoder. The function of the Command Decoder is described in Appendix D.
6	CHAIN	Loads a specified memory image file from the system device and starts it.
7	ERROR	Prints an error message of the form USER ERROR n AT LOCATION xxxxx.
10	USRIN	Loads the USR into memory. Subsequent calls to the USR are by an effective JMS to location 10200.
11	USROUT	Dismisses the USR from memory and restores the previous contents of locations 10000 to 11777.

(continued on next page)

Table C-1 (Cont.)
Summary of USR Functions

Function Code	Name	Operation
12	INQUIRE	Ascertains whether a given device exists and, if so, whether its handler is in memory.
13	RESET	Resets system tables to their initial cleared state.
14 - 17		Not currently used, these request numbers are reserved for future use.

This calling sequence can change from function to function. For example, some functions take no value in the AC and others have fewer or greater numbers of arguments. However, this format is generally followed.

The value of the data field preceding the JMS to the USR is exceedingly important. The data field MUST be set to the current field, and the instruction field MUST be set to 1. Note that a CDF is not explicitly required if the data field is already correct. When a doubt exists as to the data field setting, an explicit CDF should be executed.

Three other restrictions apply to all USR calls, as follows:

1. The USR can never be called from any address between 10000 and 11777. Attempting to do so results in the:

   MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)

   message and termination of program execution. The value of xxxxx is the address of the calling sequence (in all such MONITOR ERROR messages).

2. Several USR calls take address pointers as arguments. These pointers always refer to data in the same memory field as the call.

3. When calling the USR from field 1, these address pointers must never refer to data that lies in the area 10000 to 11777.

C.1.2 Direct and Indirect Sequence

A user program can call the USR in two ways. First, by performing a JMS to location 17700. In this case, locations 10000 to 11777 are saved on a special reserved area on the system device, and the USR is then loaded into 10000 to 11777. When the USR operation is completed, locations 10000 to 11777 are restored to their previous values.

NOTE

By setting bit 11 of the Job Status Word to a 1, you can avoid this saving and restoring of memory for programs that do not use locations 0000 to 1777 in field 1.

Alternatively, a program can keep the USR permanently resident in memory at locations 10000 to 11777 by using the USRIN function (see Section C.2.8). Once the USR has been brought into memory, a USR call is made by performing a JMS to location 10200. This is the most efficient way of calling the USR. When USR operations have been completed, the program restores locations 10000 to 11777 to their original state by executing the USROUT function, if necessary (see Section C.2.9).

C.2 USR FUNCTIONS

C.2.1 FETCH Device Handler (Function Code = 1)

Device handlers must be loaded into memory to make them available to the USR and user program for I/O operations on that device. Before performing a LOOKUP, ENTER, or CLOSE function on any device, the handler for that device must be loaded by FETCH.

The FETCH function takes two forms:

1. Load a device handler corresponding to a given device name.

   First, the following is an example of loading a handler by name from memory field 0:

```
CLA             /AC MUST BE CLEAR
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
1               /FUNCTION CODE = 1
DEVICE RXA1     /GENERATES TWO WORDS: ARG(1)
                /AND ARG(2)
6001            /ARG(3)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

   ARG(1) and ARG(2) contain the device name in standard format. If the normal return is taken, ARG(2) is changed to the device number corresponding to the device loaded. ARG(3) contains the following information:

   Bits 0 to 4 contain the page number into which the handler is to be loaded (handlers are always loaded and used in field 0).

Bit 11 is 0 if the user program can only accept a 1-page handler for this FETCH operation.

Bit 11 is 1 if there is room for a 2-page handler.

Notice that in the example above, the handler for RXA1 is to be loaded into locations 6000 to 6377. After a normal return, ARG(3) is changed to contain the entry point of the handler.

2. Load a device handler corresponding to a given device number.

   A different set of arguments is used to fetch a device handler by number. The following is an example of this form:

```
TAD VAL          /AC IS NOT ZERO
CDF 0            /DF = CURRENT FIELD
CIF 10           /IF = 1
JMS I (USR)
1                /FUNCTION CODE = 1
6001             /ARG(1)
JMP ERR          /ERROR RETURN
.                /NORMAL RETURN
.
.
```

   On entry to the USR, the AC contains the device number in bits 8 to 11 (bits 0 to 7 are ignored). The format for ARG(1) is the same as that for ARG(3) in the previous example. Following a normal return ARG(1) is replaced with the entry point of the handler.

The conditions that can cause an error return to occur in both cases are as follows:

1. There is no device corresponding to the given device name or device number, or

2. An attempt was made to load a two-page handler into one page. If this is an attempt to load the handler by name, the contents of ARG(2) have been changed already to the internal device number.

In addition, one of the following Monitor errors can be printed, followed by a return to the Monitor:

Error Message	Meaning
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if bits 8 to 11 of the AC are zero (and bits 0 to 7 are non-zero).
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	Results if a read error occurs while loading the device handler.

The FETCH function checks to see if the handler is in memory, and if it is not, then the handler and all co-resident handlers are loaded. While the FETCH operation is essentially a simple one, the following points should be noted:

1. Device handlers are always loaded into memory field 0.

2. The entry point that is returned may not be on the page desired. This would happen if the handler were already resident.

3. Never attempt to load a handler into page 37 or into page 0. Never load a two page handler into page 36 since this will corrupt the OS/78 resident monitor.

For more information on using device handlers, see Appendix F.

NOTE

Two or more device handlers are "co-resident" when they are both included in the same memory pages, for example, RXA0 and RXA1.

C.2.2 LOOKUP Permanent File (Function Code = 2)

This request locates a permanent file entry on a given device, if one exists. An example of a typical LOOKUP would be:

```
        TAD VAL         /LOAD DEVICE NUMBER
        CDF 0           /DF = CURRENT FIELD
        CIF 10          /IF = 1
        JMS I (USR)
        2               /FUNCTION CODE = 2
        NAME            /ARG(1), POINTS TO FILE NAME
        0               /ARG(2)
        JMP ERR         /ERROR RETURN
        ...             /NORMAL RETURN
NAME,   FILENAME PROG.PA
```

This request looks up a permanent file with the name PROG.PA. The device number on which the lookup is to be performed must be in AC bits 8 to 11 when the call to USR is made. ARG(1) must contain a pointer to the file name. Note that the file name block must be in the same memory field as the call, and that it cannot be in locations 10000 to 11777. The device handler must have been previously loaded into memory. If the normal return is taken, ARG(1) is changed to the starting block of the file and ARG(2) is changed to the file length in blocks as a negative number. If the device specified is a readable, non-file structured device (for example, the terminal), then ARG(1) and ARG(2) are both set to zero.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

1. The device specified is a write-only device.

2. The file specified was not found.

In addition, specifying illegal arguments can cause one of the following monitor errors, followed by a return to the Keyboard Monitor:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN CORE)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if bits 8 to 11 of the AC are zero.

The LOOKUP function is the standard method of opening a permanent file for input.

C.2.3 ENTER Output (Tentative) (File Function Code = 3)

The ENTER function is used to create a tentative file entry to be used for output. An example of a typical ENTER function is as follows:

```
        TAD VAL         /AC IS NOT ZERO
        CDF 0           /DF = CURRENT FIELD
        CIF 10          /IF = 1
        JMS I (USR)
        3               /FUNCTION CODE = 3
        NAME            /ARG(1) POINTS TO FILE NAME
        0               /ARG(2)
        JMP ERROR       /ERROR RETURN
        .               /NORMAL RETURN
        .
        .
NAME,   FILENAME TEST.PA
```

Bits 8 and 11 of the AC contain the device number of the selected device; the device handler for this device must be loaded into memory before performing an ENTER function. If bits 0 to 7 of the AC are non-zero, this value is considered to be a declaration of the maximum length of the file. The ENTER function searches the file directory for the smallest empty file that contains at least the declared number of blocks. If bits 0 to 7 of the AC are zero, the ENTER function locates the largest available empty file.

On the normal return, the contents of ARG(1) are replaced with the starting block of the file. The 2's complement of the actual length of the created tentative file in blocks (which can be equal to or greater than the requested length) replaces ARG(2).

NOTE

If the selected device is not file structured but permits output operations (for example, a line printer), the ENTER operation always succeeds. In this case, ARG(1) and ARG(2) are both zeroed on return.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

1. The device specified by bits 8 to 11 of the AC is a read only device.

2. No empty file exists which satisfies the request length requirement.

3. Another tentative file is already active on this device (only one output file can be active at any given time).

4. The first word of the file name was 0 (an illegal file name).

In addition, one of the following monitor errors can occur, followed by a return to the Keyboard Monitor:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN MEMORY)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if AC bits 8 to 11 are zero.
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	Read error on the system device while bringing in the overlay code for the ENTER function.
MONITOR ERROR 6 AT xxxxx (DIRECTORY OVERFLOW)	Results if a directory overflow occurred (no room for tentative file entry in directory).

C.2.4 The CLOSE Function (Function Code = 4)

The CLOSE function has a dual purpose: first, it is used to close the current active tentative file, making it a permanent file. Second, when a tentative file becomes permanent, it is necessary to remove (delete) any permanent file having the same name; this operation is also performed by the CLOSE function. An example of CLOSE usage follows:

```
        TAD VAL         /GET DEVICE NUMBER
        CDF 0           /DF = CURRENT FIELD
        CIF 10          /IF = 1
        JMS I (USR)
        4               /FUNCTION CODE = 4
        NAME            /ARG(1)
        15              /ARG(2)
        JMP ERR         /ERROR RETURN
        .               /NORMAL RETURN
        .
NAME,   FILENAME TEST.PA
```

The device number is contained in AC bits 8 to 11 when calling the USR. ARG(1) is a pointer to the name of the file to be closed and/or deleted and ARG(2) contains the number of blocks to be used for the new permanent file.

The normal sequence of operations on an output file is:

1. FETCH the device handler for the output device.

2. ENTER the tentative file on the output device, getting the starting block and the maximum number of blocks available for the file.

3. Perform the actual output using calls to the device handler, keeping track of how many blocks are written, and checking to insure that the file does not exceed the available space.

4. CLOSE the tentative file, making it permanent. The CLOSE operation would always use the same file name as the one used for ENTER performed in step 2. The closing file length would have been computed in step 3 and used in the CLOSE call.

After a normal return from CLOSE, the active tentative file is permanent and any permanent file having the specified file name already stored on the device is deleted. If the specified device is a non-file structured device that permits output (the lineprinter, for example) the CLOSE function will always succeed.

NOTE

The user must be careful to specify the same file names to the ENTER and the CLOSE functions. Failure to do so can cause several permanent files with identical names to appear in the directory. If CLOSE is intended only to be used to delete some existing file, then the number of blocks (ARG(2)) should be zero.

The following conditions cause the error return to be taken:

1. The device specified by bits 8 to 11 of the AC is a read-only device.

2. There is neither an active tentative file to be made into a permanent file nor a permanent file with the specified name to be deleted.

In addition, one of the following Monitor errors can occur:

Error Message	Meaning
MONITOR ERROR 1 AT xxxxx (CLOSE ERROR)	Results if the length specified by ARG(2) exceeded the allotted space.
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading or writing the device directory.

Error Message	Meaning
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN MEMORY)	Results if the device handler for the specified device is not in memory.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if AC bits 8 to 11 are zero.

C.2.5 Call Command Decoder (DECODE) (Function Code = 5)

The DECODE function causes the USR to load and execute the Command Decoder. The Command Decoder accepts (from the terminal) a list of input and output devices and files, along with various options. The Command Decoder performs a LOOKUP on all input files, sets up necessary tables in the top page of field 1, and returns to the user program.

A typical call to the Command Decoder looks as follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
5               /FUNCTION CODE = 5
2001            /ARG(1), ASSUMED INPUT EXTENSION (.PA)
0               /ARG(2), ZERO TO PRESERVE ALL
                /TENTATIVE FILES
.               /NORMAL RETURN
.
.
```

ARG(1) is the assumed input extension; in this example it is ".PA". On return from the Command Decoder, information is stored in tables located in the last page of memory field 1. The DECODE function also resets all system tables as in the RESET function (see RESET function, Section C.2.11); if ARG(2) is 0, all currently active tentative files remain open; if ARG(2) is non-zero,1 all tentative files are deleted, and the normal return is to ARG(2) instead of ARG(2)+1.

The DECODE function has no error return (Command Decoder error messages are given in Appendix D). However, the following Monitor error can occur:

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	I/O error occurred while reading or writing on the system device.

C.2.6 CHAIN Function (function Code = 6)

The CHAIN function permits a program to run another program with the restriction that the program chained to must be a memory image (.SV) file located on the system device. A typical implementation of the CHAIN function follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
6               /FUNCTION CODE = 6
BLOCK           /ARG(1), STARTING BLOCK NUMBER
```

There is no normal or error return from CHAIN. However, the following monitor error can occur:

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	I/O error occurred while reading or writing on the system device.
CHAIN ERR	If an attempt is made to CHAIN to a file which is not a memory image (.SV) file. Control returns to the Monitor.

The CHAIN function loads a memory image file located on the system device beginning at the block number specified as ARG(1) (which is normally determined by performing a LOOKUP on the desired file name). Once loaded, the program is started at an address that is one greater than the starting address specified by the program's Core Control Block.

CHAIN automatically performs a USROUT function (see Section C.2.9), if necessary, to dismiss the USR from memory, and a RESET to clear all system tables (see Section C.2.11), but CHAIN does not delete tentative files. Normally, it is best to set bit 11 of the Job Status Word before chaining to an OS/78 system program.

The areas of memory altered by the CHAIN function are determined by the contents of the Core Control Block of the memory image file loaded by CHAIN. The Core Control Block for the file is set up by ABSLDR or LOADER programs. It can be modified by performing a SAVE command with specific arguments. Every page of memory in which at least one location was saved is loaded. If the page is one of the "odd numbered" pages (pages 1, 3, etc.; locations 0200 to 0377, 0600 to 0777, etc.), the previous page is always loaded. In addition, CHAIN always alters the contents of locations 07200 to 07577.

NOTE

CHAIN destroys a necessary part of the ODT resident breakpoint routine. Thus an ODT breakpoint should never be attempted across a CHAIN.

With the above exceptions, programs can pass data back and forth in memory while chaining. For example, FORTRAN programs normally leave the COMMON area in memory field 1 unchanged. This COMMON area can then be accessed by the program activated by the CHAIN.

C.2.7 Signal User ERROR (Function Code = 7)

The USR can be called to print a user error message for a program. The following is a possible ERROR call:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
7               /FUNCTION CODE = 7
2               /ARG(1), ERROR NUMBER
```

The ERROR function causes a message of the form:

```
USER ERROR n AT xxxxx
```

to be printed. Here n is the error number given as ARG(1); n must be between 0 and 11 (octal), and xxxxx is the address of ARG(1). If ARG(1) in the sample call above was at location 500 in field 0, the message:

```
USER ERROR 2 AT 00500
```

would be printed. Following the message, the USR returns control to the Keyboard Monitor, preserving the user program intact.

The error number is arbitrary. Two numbers have currently assigned meanings:

Error Message	Meaning
USER ERROR 0 AT xxxxx	During a RUN, GET, or R command, this error message indicates that an error occurred while loading the memory image.
USER ERROR 1 AT xxxxx	During a FORTRAN program execution, this error indicates that a call was made to a subroutine that was not loaded.

C.2.8 Lock USR in Memory (USRIN) (Function Code = 10)

When a program must make a number of calls to the USR, it is advantageous to have the program avoid reloading the USR each time a USR call is made. The USR can be brought into memory and kept there by the USRIN function. The calling sequence for the USRIN function is as follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (7700)
10              /FUNCTION CODE = 10
.               /NORMAL RETURN
.
.
```

The USRIN function saves the contents of locations 10000 to 11777 in a reserved area on SYS: (provided the calling program loads into this area as indicated by the current Job Status Word), then loads the USR, and finally returns control to the user program.

Once the USR is locked in memory, you can call it at location 10200.

NOTE

If bit 11 of the current Job Status Word is a 1, the USRIN function will not save the contents of locations 10000 through 11777.

C.2.9 Dismiss USR From Memory (USROUT) (Function Code = 11)

When a program has loaded the USR into memory with the USRIN function and no longer wants or needs the USR in memory, the USROUT function restores the original contents of locations 10000 to 11777. The calling sequence for the USROUT function is as follows:

```
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (200)     /DO NOT JMS TO 17700!!
11              /FUNCTION CODE = 11
.               /NORMAL RETURN
.
.
```

Subsequent calls to the USR must be made by performing a JMS to location 7700 in field 1.

NOTE

If bit 11 of the current Job Status Word is a 1, the contents of memory are not changed by the USROUT function. In this case USROUT is a redundant operation since memory was not preserved by the USRIN function.

C.2.10 Ascertain Device Information (INQUIRE) (Function Code = 12)

On some occasions a user may wish to determine what internal device number corresponds to a given device name or whether the device handler for a specified device is in memory, without actually performing a FETCH operation. INQUIRE performs these operations for the user. The function call for INQUIRE closely resembles the FETCH handler call.

INQUIRE, like FETCH, has two forms:

1. Obtain the device number corresponding to a given device name and determine if the handler for that device is in memory.

   An example of the INQUIRE call is shown below:

```
CLA             /AC MUST BE CLEAR

CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR)
12              /FUNCTION CODE = 12
DEVICE RLOB     /GENERATES TWO WORDS:
                /ARG(1) AND ARG(2)
0               /ARG(3)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

   ARG(1) and ARG(2) contain the device name in standard format. When the normal return is taken, ARG(2) is changed to the device number corresponding to the given name, and ARG(3) is changed to either the entry point of the device handler if it is already in memory, or zero if the corresponding device handler has not yet been loaded.

2. Determine if the handler corresponding to a given device number is in memory.

   A slightly different set of arguments is used to inquire about a device by its device number:

```
TAD VAL          /AC IS NON-ZERO
CDF 0            /DF = CURRENT FIELD
CIF 10           /IF = 1
JMS I (USR)
12               /FUNCTION CODE = 12
0                /ARG(1)
JMP ERR          /ERROR RETURN
.                /NORMAL RETURN
.
.
```

   On entry to INQUIRE, AC bits 8 to 11 contain the device number.

NOTE

If AC bits 0 to 7 are non-zero, and bits 8 to 11 are zero (an illegal device number), the

MONITOR ERROR 4 AT xxxxx

message is displayed and program execution is terminated.

On normal return ARG(1) is set to the entry point of the device handler if it is already in memory, or zero if the corresponding device handler has not yet been loaded. The error return in both cases is taken only if there is no device corresponding to the device name or number specified.

C.2.11 RESET System Tables (Function Code = 13)

Resetting the system tables effectively removes from memory all device handlers except the system handler. An example of the RESET function is shown below:

```
CDF 0            /DF = CURRENT FIELD
CIF 10           /IF = 1
JMS I (USR)
13               /FUNCTION CODE = 13
0                /0 PRESERVES TENTATIVE FILES
.                /NORMAL RETURN
.
.
```

RESET removes all device handlers, other than that for the system device, from memory. This operation should be done anytime a user program modifies any page in which a device handler was loaded.

RESET also deletes all currently active tentative files (files that have been entered but not closed). This results in zeroing bits 9 through 11 of every entry in the Device Control Word Table (see Section B.3.5 of OS/8 Software Support Manual). If RESET is to be used to delete all active tentative files, then ARG(1) must be non-zero and the normal return is ARG(1) rather than to ARG(1)+1. For example, the following call would serve this purpose.

```
CDF 0           /DF:CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
13              /FUNCTION CODE = 13
CLA CMA         /NON-ZERO INSTRUCTION
```

The normal return would execute the CLA CMA, and all active tentative files on all devices would be deleted. The Keyboard Monitor currently does not reset the Monitor tables. If user programs which do not call the Command Decoder are used, it is wise to do a RESET operation before loading device handlers. The RESET will ensure that the proper handler will be loaded into memory.

APPENDIX D

THE COMMAND DECODER

OS/78 provides a powerful subroutine called the Command Decoder for use by all system and user programs. The Command Decoder is normally called when a program starts running. When called, the Command Decoder displays an asterisk (*) and then accepts a command line from the console terminal that includes a list of I/O devices, file names, and various option specifications. The Command Decoder validates the command line for accuracy, performs a LOOKUP on all input files, and sets up various tables for the calling program (refer to Appendix C, Section C.2.2, for more information on LOOKUP).

D.1 COMMAND DECODER CONVENTIONS

The command line has the following general form in response to the Command Decoder asterisk:

```
*output files <input files/(options)
```

There can be 0 to 3 output files and 0 to 9 input files specified.

Output File Examples	Meaning
EXPLE.EX	Output to a file named EXPLE.EX on device DSK (the default file storage device).
LPT:	Output to the LPT. This format generally specifies a non-file-structured device.
RXA2:EXPLE.EX	Output to a file named EXPLE.EX on device RXA2.
RL0A:EXPLE.EX[99]	Output to a file named EXPLE.EX on device RL0A. A maximum output file size of 99 blocks is specified.
(null)	No output specified.

An input file specification has one of the following forms:

Input File Format	Meaning
RXA2:INPUT	Input from a file named INPUT.df on device RXA2. where df is the assumed input file extension specified to the Command Decoder.
RXA2:INPUT.EX	Input from a file named INPUT.EX on device RXA2. In this case .EX overrides the assumed input file extension.

Input File Format	Meaning
INPUT.EX	Input from a file named INPUT.EX. If there is no previously specified input device, input is from device DSK, the default file storage device; otherwise, the input device is the same as the last specified input device.
TTY:	Input from terminal; no file name is needed for non-file structured devices.
(null)	Repeats input from the previous device specified (must not be first in input list, and must refer to a non-file structured device).

NOTE

Whenever a file extension is left off an input file specification, the Command Decoder first performs a LOOKUP for the given name appending a specified assumed extension. If the LOOKUP fails, a second LOOKUP is made for the file appending a null (zero) extension.

The Command Decoder verifies that the specified device names, file names, and extensions consist only of the characters A through Z and 0 through 9. If not, a syntax error is generated and the command line is considered to be invalid.

Two kinds of options can be specified: alphanumeric and numberic. Alphanumeric option switches are denoted by a single alphanumeric character preceded by a slash (/) or a string of characters enclosed in parentheses. Numeric options can be specified in octal numbers from 1 to 37777777 preceded by an equal sign (=). These options are passed to the user program and are interpreted differently by each program.

Finally, the Command Decoder permits the command line to be terminated by either the RETURN or ESCape key. This information is also passed to the user program.

D.2 COMMAND DECODER ERROR MESSAGES

If an error in the command line is detected by the Command Decoder, one of the following error messages is displayed. After the error message, the Command Decoder starts a new line, prints an *, and waits for another command line. The erroneous command is ignored.

Error Message	Meaning
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified. (Or in special mode, more than one output file or more than five input files.)

Error Message	Meaning
device DOES NOT EXIST	The specified device name does not correspond to any permanent device name or any user assigned device name.
name NOT FOUND	The specified input file name was not found on the selected device.

D.3 CALLING THE COMMAND DECODER

The Command Decoder is initiated by the DECODE function of the USR. DECODE saves the contents of locations 0 to 1777 of field 0 on the system scratch blocks, and brings the Command Decoder into that area of memory and starts it. When the command line has been entered and properly interpreted, the Command Decoder exits to the USR, which restores the original contents of 0 to 1777 and returns to the calling program.

NOTE

By setting bit 10 of the Job Status Word to a 1; you can avoid this saving and restoring of memory for programs that do not occupy locations 0 to 1777.

The DECODE call can reside in the area between 00000 to 01777 and still function correctly. A typical call would appear as follows:

```
CDF 0               /SET DATA FIELD TO CURRENT FIELD
CIF 10              /INSTRUCTION FIELD MUST BE 1
JMS I (USR          /USR=7700 IF USR IS NOT IN MEMORY
                    /OR USR=0200 IF USRIN WAS PERFORMED
5                   /DECODE,FUNCTION = 5
2001                /ARG(1),ASSUMED INPUT EXTENSION
0                   /ARG(2), ZERO TO PRESERVE
                    /ALL TENTATIVE FILES
.                   /NORMAL RETURN
.
.
```

ARG(1) is the assumed input extension in SIXBIT notation. If an input file name is given with no specified extension, the Command Decoder first performs a LOOKUP for a file having the given name with the assumed extension. If the LOOKUP fails, the Command Decoder performs a second LOOKUP for a file having the given name and a null (zero) extension. In this example, the assumed input extension is ".PA".

DECODE performs an automatic RESET operation to remove from memory all device handlers except those equivalent to the system device. As in the RESET function, if ARG(2) is zero, all currently active tentative files are preserved. If ARG(2) is non-zero, all tentative files are deleted, and DECODE returns to ARG(2) instead of ARG(2)+1.

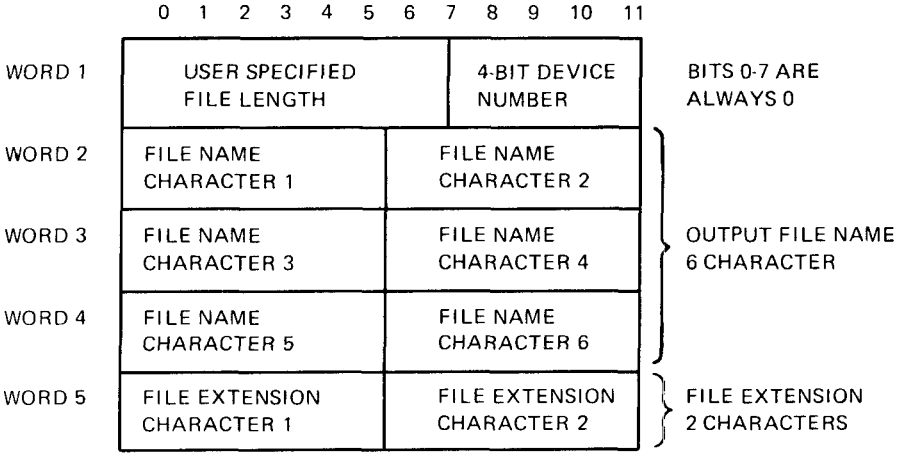
As the Command Decoder normally handles all of its own errors, there is no error return from the DECODE operation.

D.4 COMMAND DECODER TABLES

The Command Decoder sets up various tables in the top page of field 1 that describe the command line typed to the user program.

D.4.1 Output Files

The output file table that begins at location 17600 has room for three entries. Each entry is five words long and has the following format:



Bits 0 to 7 of word 1 in each entry contain the file length, if the file length was specified with the square bracket construction in the command line. Otherwise, those bits are zero.

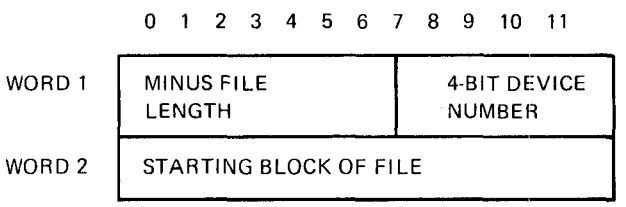
The entry for the first output file is in locations 17600 to 17604, the second is in locations 17605 to 17611, and the third is in locations 17612 to 17616. If word 1 of any entry is zero, the corresponding output file was not specified. A zero in word 2 means that no file name was specified.

Also, if word 5 of any entry is zero no file extension was specified for the corresponding file. It is left to the user program to take the proper action in these cases.

These entries are in a format that is acceptable to the ENTER function.

D.4.2 Input Files

The input file table that begins at location 17617 has room for nine entries. Each entry is two words long and has the following format:



Bits 0 to 7 of word 1 contain the file length as a negative number. Thus, 377 (octal) in these bits is a length of one block, 376 (octal) is a length of two blocks, etc. If bits 0 to 7 are zero, the specified file has a length greater than or equal to 256 blocks or a non-file structured device was specified.

NOTE

Restrictiing the actual specified size to 255 blocks can cause some problems if the program has no way of detecting end-of-file conditions. If this is a problem, your program probably should use the special mode of the Command Decoder described in Section D.5 and perform its own LOOKUP on the input files to obtain the exact file length.

The two-word input file list entries beginning at odd numbered locations from 17617 to 17637 inclusive. If location 17617 is zero, no input files were indicated in the command line. If less than nine input files were specified, the unused entries in the input file list are zeroed (location 17641 is always set to zero to provide a terminator even when no files are specified).

D.4.3 Command Decoder Option Table

Five words are reserved beginning at location 17642 to store the various options specified in the command line. The format of these five words is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11
17642	HIGH ORDER 11 BITS OF = N OPTIONS											
17643	A	B	C	D	E	F	G	H	I	J	K	L
17644	M	N	O	P	Q	R	S	T	U	V	W	X
17645	Y	Z	0	1	2	3	4	5	6	7	8	9
17646	LOW ORDER 12 BITS OF = N OPTIONS											

Each of the bits in locations 17643 and 17645 corresponds to one of the possible alphanumeric option switches. The corresponding bit is 1 if the switch was specified, 0 if not specified.

NOTE

If no = n option is specified, the Command Decoder zeroes location 17646 and bits 1 to 11 of 17642. Thus, typing = 0 is meaningless since the user program cannot tell that any option was specified.

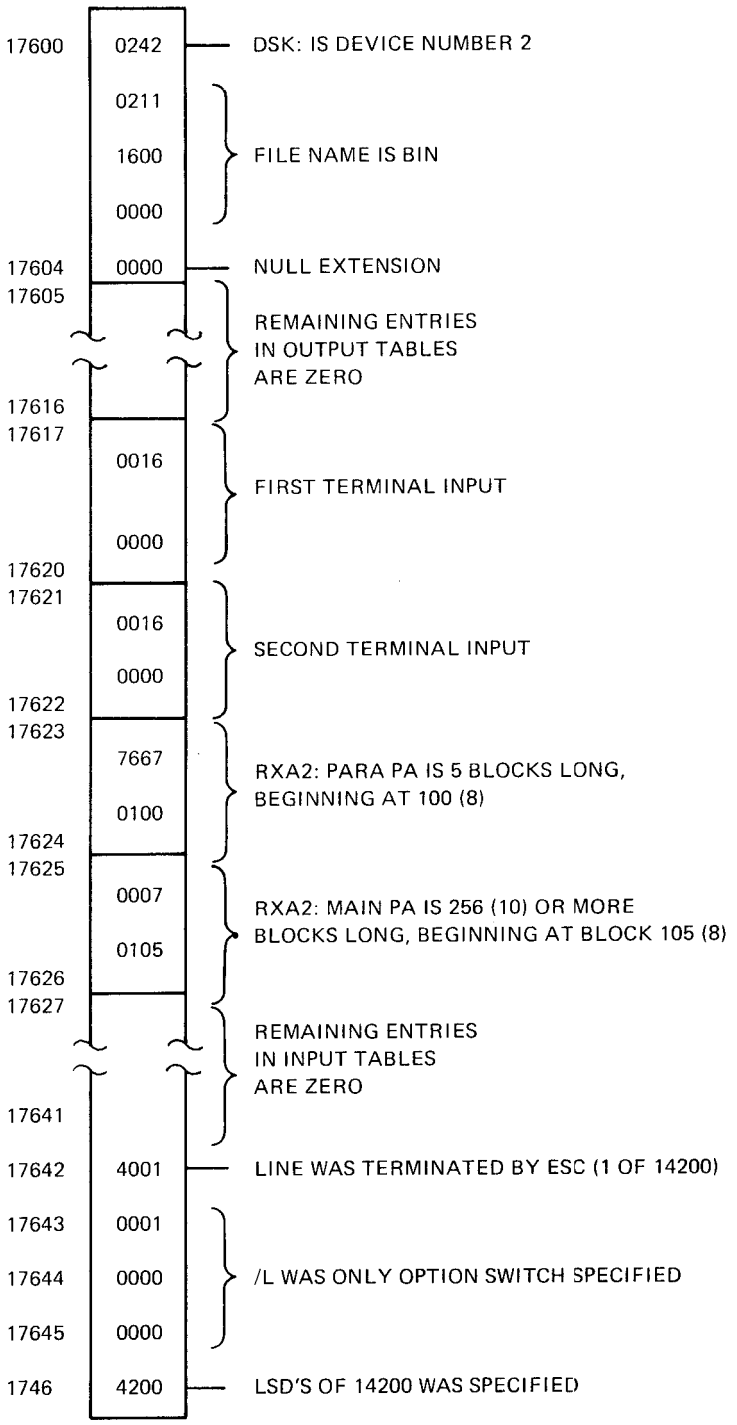
Bit 0 of location 17642 is 0 if the command line was terminated by a carriage return, 1 if it was terminated by an ESCape.

D.4.4 Example

To clarify some of the preceding, consider the interpretation of the following command line:

```
*BIN[10]<TTY:,,RXA2:PARA.PA,MAIN.PA/L=14200
```

If this command line is typed, the Command Decoder would return to the calling program with the following values in the system tables:


17600
0242
DSK: IS DEVICE NUMBER 2
0211
1600
0000
FILE NAME IS BIN
17604
0000
NULL EXTENSION
17605
REMAINING ENTRIES
IN OUTPUT TABLES
ARE ZERO
17616
17617
0016
0000
FIRST TERMINAL INPUT
17620
17621
0016
0000
SECOND TERMINAL INPUT
17622
17623
7667
0100
RXA2: PARA PA IS 5 BLOCKS LONG,
BEGINNING AT 100 (8)
17624
17625
0007
0105
RXA2: MAIN PA IS 256 (10) OR MORE
BLOCKS LONG, BEGINNING AT BLOCK 105 (8)
17626
17627
REMAINING ENTRIES
IN INPUT TABLES
ARE ZERO
17641
17642
4001
LINE WAS TERMINATED BY ESC (1 OF 14200)
17643
0001
17644
0000
/L WAS ONLY OPTION SWITCH SPECIFIED
17645
0000
1746
4200
LSD'S OF 14200 WAS SPECIFIED

NOTE

The entries for terminal (where no input file name is specified) have a starting block number and file size of zero. This is always true of the input table for a non-file structured device, or a file structured device on which no file name is given.

D.5 SPECIAL MODE OF THE COMMAND DECODER

Occasionally the user program does not want the Command Decoder to perform the LOOKUP on input files, leaving this option to the user program itself. Programs such as format conversion routines which access non-standard file structures could use this special format. If the input files were not OS/78 format, a command decoder LOOKUP operation would fail. The capability to handle this case is provided in the OS/78 Command Decoder. This capability is generally referred to as the "special mode" of the Command Decoder.

D.5.1 Calling the Command Decoder Special Mode

The special mode call to the Command Decoder is identical to the standard DECODE call except that the assumed input file extension, specified by ARG(1), is equal to 5200. The value 5200 corresponds to an assumed extension of ".*", which is illegal. Therefore, the special mode of the Command Decoder in no way conflicts with the normal mode.

D.5.2 Operation of the Command Decoder in Special Mode

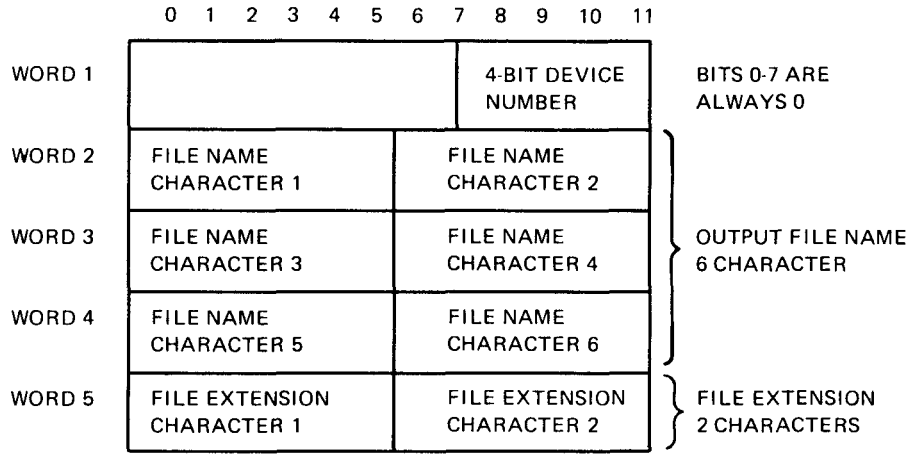
In special mode the Command Decoder is loaded and inputs a command line as usual. The appearance of the command line is altered by the special mode in these respects:

1. Only one output file can be specified.

2. No more than five input files can be specified, rather than the nine acceptable in normal mode.

3. The characters asterisk (*) and question mark (?) are legal in file names and extensions, both in input files and on output files. It is strongly suggested that these characters be tested by the user program and treated either as special options or as illegal file names. The user program must not ENTER an output file with an asterisk or a question mark in its name since the OS/78 system programs will have difficulty manipulating or deleting files named this way.

The output and option table set up by the Command Decoder is not altered in special mode. Entries in the input table are changed to the following format:



The table entry for the first input file is in locations 17605 to 17611; the second in locations 17612 to 17616; the third in location 17617 to 17623; the fourth in locations 17624 to 17630; and the fifth in locations 17631 to 17635. A zero in word 1 terminates the list of input files. If word 2 of an entry is zero, no input file name was specified.

APPENDIX E

OS/78 DEFAULT FILE NAME EXTENSIONS

This appendix lists the default file name extensions used in OS/78.

Extension	Meaning
.BA	BASIC source file (default extension for a BASIC input file).
.BI	Batch input file (input for BATCH).
.BN	Absolute binary file (default extension for ABSLDR and BITMAP input files; also used as default extension for PAL8 binary output file).
.CM	Command file.
.DI	Directory listing file.
.FT	FORTRAN language source file (default extension for FORTRAN input files).
.HL	Text file accessed by the HELP command.
.HN	Device handler save image file used by the SET HANDLER command.
.LD	FORTRAN load module file (default assumed by run-time system, FORTRAN IV loader).
.LS	Assembly listing Output file (default extension for PAL8).
.MP	File containing a loading map (used by the Linking Loader, MAP command).
.PA	PAL8 source file.
.RA	RALF assembly language file (FORTRAN IV).
.RL	Relocatable binary file (default extension for a Linking Loader input file).
.SV	Memory image file (SAVE file); default for the R, RUN, SAVE, and GET commands.
.TM	Temporary file generated by system.

APPENDIX F

USING DEVICE HANDLERS

A device handler is a system subroutine that is used by all parts of the OS/78 system and by all standard system programs to perform I/O transfers. All device handlers are called in the same way, and they all perform the same basic operation of reading or writing a specified number of 128 -word records beginning at a selected memory address. This appendix contains information relevant to assembly language (PAL8) programming only.

These subroutines effectively mask the unique characteristics of different I/O devices from the calling program; thus, programs that use device handlers properly are effectively "device independent". Changing devices involves merely changing the device handlers used for I/O.

Device handlers have another important feature. They are able to transfer a number of records as a single operation. On a file-structured device, a single operation could transfer an entire track or more. This capability significantly increases the speed of operation of programs that have large buffer areas.

All device handlers occupy two memory pages, and can run in any page of field 0, except in Pages 0, 36 and 37.

NOTE

A "record" is 128 words of data; thus, an OS/78 block consists of two 128-word records.

F.1 CALLING DEVICE HANDLERS

Device handlers are loaded into a user selected area in memory field 0 by the FETCH function of the USR (see Appendix C). This is possible because the SYS: handler is always resident. FETCH stores in ARG(1) the entry point of the handler loaded. The handler is called by executing a JMS to the specified entry point address. Handler calls have the following format:

```
CDF N           /WHERE N IS THE VALUE OF THE CURRENT
                /PROGRAM FIELD TIMES 10 (OCTAL)
CIF 0           /DEVICE HANDLER ALWAYS IN FIELD 0
JMS I ENTRY     /*ENTRY CONTAINS THE HANDLER ENTRY POINT
ARG(1)          /FUNCTION CONTROL WORD
ARG(2)          /BUFFER ADDRESS
ARG(3)          /STARTING BLOCK NUMBER
JMP ERR         /ERROR RETURN
```

```
        .               /NORMAL RETURN (I/O TRANSFER COMPLETE)
        .
        .
ENTRY,  0               /ENTRY CONTAINS ARG(3) FROM THE FETCH OPERATION
#
```

NOTE

The entry point for SYS is always 07607.

As with calls to the USR, it is important to set the data field is set to the current program field before the device handler is called. On exit from the device handler, the data field will remain set to the current program field. The accumulator need not be zero when calling a handler; it will be zero when the normal return is taken.

ARG(1) is the function control word, and contains the following information:

Bits	Contents
Bit 0	0 for an input operation (read), 1 for an output operation (write).
Bits 1 to 5	The number of 128 word records to be transferred. If bits 1-5 are zero and the device is non-file structured (i.e., TTY, LPT, etc.) the operation is device dependent. If the device is file structured, a read/write of 40 (octal) pages is performed.
Bits 6 to 8	The memory field in which the transfer is to be performed.
Bits 9 to 11	Unused (device dependent) bits should be left zero.

ARG(2) is the starting location of the transfer buffer.

ARG(3) is the number of the block at which the transfer is to begin. The user program initially determines this value by performing a LOOKUP or ENTER operation. After each transfer the user program should itself add to the current block number (this argument) the actual number of blocks transferred, equal to one-half the number of 128-word records specified, rounded up if the number of records was odd.

Error returns are either: fatal or non-fatal. When an error return occurs and the contents of the AC are negative, the error is fatal. A fatal error can be caused by a bad data checksum (CRC) or an attempt to write on a read-only device (or vice versa). The meaning can vary from device to device, but in all cases it is serious enough to indicate that the data transferred, if any, is invalid.

When an error return occurs and the contents of the AC are greater than or equal to zero, a non-fatal error has occurred. This error always indicates detection of the end-of-file character (CTRL/Z) on non-file-structured input devices. For example, when a CTRL/Z is typed during input from device TTY, the TTY handler inserts a CTRL/Z code in the buffer and takes the error exit with the AC equal to zero. While non-file-structured input devices can detect the end-of-file condition, file-structured devices cannot; furthermore, no device handler takes a non-fatal error return when doing output.

The following restrictions apply to the use of device handlers:

1. File-structured Handlers - If bits 1-5 of the function control word (ARG(1)) are zero, the handler transfers an entire memory page (40 (octal) blocks). Be caregul when using this type of transfer not to overlay the handler itself on the system that resides in the various parts of Fields 0, 1, and 2.

2. The user program must never specify an input into locations 07600 to 07777, 17600 to 17777, or 27600-27777, or the page(s) in which the device handler itself resides. In general, 7600-7777 in every memory field are reserved for use by system software. Those areas should be used with caution.

3. Note that the amount of data tranferred is given as a number of 128-word records, exactly one half of an OS/78 block. Attempting to output an odd number of records will change the contents of the last 128 words of the last block written.

4. The specified buffer address does not have to begin at the start of a page. The specified buffer cannot overlap fields; rather the address will "wrap around" memory. For example, a write of 2 pages starting at location 07600 would cause locations 07600-07777 and 00000-00177 of field 0 to be written.

5. If bits 1-5 of the function control word (ARG(1)) are zero, a non-file-structured device-dependent operation occurs. Users should not expect a 40-page (full field) transfer of data. The CLOSE operation of the USR calls the handler with bits 1-5 and 9-11 of the function control word 0. This condition means "perform any special close operation desired". Non-file structured handlers, which need no special handling on the conclusion of data transfers, should treat this case as a NOP. An example of usage of such special codes would be where the line printer would perform a line feed.

F.2 OS/78 DEVICE HANDLERS

This section describes briefly the operation of standard OS/78 device handlers, including normal operation, any special initialization operations for function control word=0, terminating conditions, and response to control characters typed at the keyboard.

F.2.1 Line Printer (LPT, LQP)

1. Normal Operation

   These handlers unpack characters from the buffer and print them on the line printer. The character horizontal tab (ASCII 211) causes sufficient spaces to be inserted to position the next character at a "tab stop" (every eighth column, by definition). The character vertical tab (ASCII 213) causes a skip to the next paper position for vertical tabulation if the line printer hardware provides that feature. The character form feed (ASCII 214) causes a skip to the top of the next page. Finally, the handler maintains a record of the current print column and starts a new line after the full width has been printed. This handler functions properly only on ASCII data.

2. Initialization for Function Control Word = 0

   Before printing begins, the line printer handler issues a form feed to space to the top of the next page.

3. Terminating Condition

   On detection or a CTRL/Z character in the buffer, the line printer handler issues a form feed and immediately takes the normal return. Attempting to input from the line printer forces a fatal error to be returned. A fatal error is also returned if the line printer error flag is set. There are no fatal errors associated with the line printer handler if the device is off line.

F.2.2 File-Structured Devices (SYS, DSK, RXAx, RLxx, VXA0)

1. Normal Operation

   These handlers transfer data directly between the device and the buffer.

2. Initialization for Function Control Word = 0

   None.

3. Terminating Conditions

   A fatal error is returned whenever the transfer sets one of the error flags in the device status register to be set. For example, a fatal error would result if a CRC error occurred while reading data from a diskette. The device handlers generally try to perform the operation three times before giving up and returning a fatal error. All errors for file-structured devices are fatal.

4. Terminal Interaction

   Typing CTRL/C forces a return to the Monitor except when using the system device handler (SYS).

NOTE

The system and non-system device handlers for the RX01 and RX02 when called whith a negative block number (4000-7777) take the error return with the complement of the device size in the AC (single density = 7022, double density = 6044). This allows you to determine whether the device is on RX01 or on RX02.

F.2.3 Terminal Handlers (TTY, SLUx, VLUx)

Listed are the features of the terminal handlers:

1. These handlers read a line at a time. Whenever the user types CR, they enter a CR/LF into the buffer, echo a CR/LF, pad the remainder of the buffer with nulls, and return to the calling program. The characters are inserted into the buffer one character per word. Thus every third character is a null so far as OS/78 is concerned.

2. The DELETE deletes the previous character from the buffer. The handler will echo a backslash (\) for a hard-copy device or delete the character from the screen for a video terminal. This can be altered by the SET term SCOPE command.

3. CTRL/U echoes as ^U and erases the current line, allowing the user to retype it. (It also starts a new line.) The buffer pointer is reset to the beginning of the buffer.

4. CTRL/Z echoes as ^Z (followed by CR, LF) and is used to signal end-of-file (end of input). The ^Z enters the buffer and the remainder of the buffer is padded with nulls. The error return is taken with a positive AC (non-fatal error).

5. Nulls are ignored.

6. CTRL/C echoes as ^C and returns control to the Monitor via location 07600.

On output: (either normal output or when echoing input)

1. CTRL/C on keyboard echoes as ^C and returns control to the Monitor.

2. CTRL/O on keyboard stops further echoing. All visible output ceases (although the buffer continues to be filled) until either the handler is reloaded into memory or the user types any character on the keyboard. Not operative during input.

3. CTRL/S causes the handler to suspend output to the terminal. No characters are lost, and output resumes when a CTRL/Q is typed. CTRL/S and CTRL/Q do not echo. These characters are operative only upon output. On input, they are treated like other input characters. This is useful on high-speed video terminals.

F.2.4 Multiple Input Files

There is a peculiarity associated with reusing Device Handler areas in OS/78. This is illustrated by the following example:

Assume a program has reserved locations 1000-1377 for its input handler and locations 7400-7577 for its output handler. If the program gives a USR FETCH command to load the RXA1 handler as an input device handler, both RX handlers will load into 1000-1377, since they are both co-resident. If another FETCH is issued to load the RXA0 handler as an output device handler, that handler will not be loaded, because it shares space with the RXA1 handler currently in memory. This is fine; however, if the user now switches input devices and FETCHes the terminal handler, as an input device handler, it will destroy the RXA0 handler and the next attempt to output using the RXA0 handler will produce errors. You can get around this problem in two ways.

1. Always assign first the handler which you expect to stay in memory the longest. Although most programs can process more than one input file per program step (for example, an assembly pass is one program step), they can process only one output file; therefore, they assign the output handler before any of the input handlers. In the above example, the problem would be eliminated if the RXA1 handler were assigned first.

2 Always give a USR RESET call before each FETCH. Obviously, this call should not delete any open output files. This means that the USR will always load the new handler, even if another copy is in memory. The user must FETCH the output handler again before issuing the USR CLOSE call; otherwise, the USR will determine that the output handler is not in memory and give a MONITOR ERROR 3 message.

APPENDIX G

OS/78 ERROR MESSAGE SUMMARY

Error messages generated by OS/78 programs are listed in alphabetic order and identified by the system program by which they are generated. Also, system halts that occur as a result of errors are listed. This appendix is only a summary. Refer to the appropriate chapters for more detailed information about specific error conditions.

G.1 SYSTEM HALTS

Errors that occur as a result of a major I/O failure on the system device can cause a system halt. These are as follows.

Value of PC	Meaning
00601	A read error occurred while attempting to load ODT.
07461	An error occurred while reading a program into memory during a CHAIN.
07605	An error occurred while attempting to write the Monitor area onto the system scratch blocks.
07702	A user program has performed a JMS to 7700 in field 0. This is a result of trying to call the USR without first performing a CIF 10.
07764	A read error occurred while loading a program.
07772	A read error occurred on the system scratch area while loading a program.
10066	An input error occurred while attempting to restore the USR.
10256	A read error occurred while attempting to load the Monitor.
17676	An error occurred while attempting to read the Monitor from the system device.
17721	An error occurred while saving the USR area.
17727	An error occurred while attempting to read the USR from the system device.
17736	An error occurred while reading the scratch blocks to restore the USR area.

After bootstrapping and retrying the operation that caused the failure, if the error persists, it is the result of a hardware malfunction or a parity error in the system area.

G.2 ERROR MESSAGES

Message	Program	Explanation
?0	SRCCOM	Insufficient memory -- this means that the differences between the files are too large to allow for effective comparison. Use of the /X option may alleviate this problem.
0	EDIT	Editor failed in reading a device. Error occurred in device handler; most likely a hardware malfunction.
?1	SRCCOM	Input error on file #1 or less than 2 input files specified.
1	EDIT	Editor failed in writing onto a device. Generally a hardware malfunction.
?2	SRCCOM	Input error on file #2.
2	EDIT	File close error occurred. The output file could not be closed; the file has been deleted.
2045 REFS	CREF	More than 2044 (decimal) references to one symbol were made.
?3	SRCCOM	Output file too large for output device. It has been deleted.
3	EDIT	File open error occurred. This error occurs if the output device is a read-only device or if no output file name is specified for a file-oriented output device.
?4	SRCCOM	Output I/O error.
4	EDIT	Device handler error occurred. The Editor could not load the device handler for the specified device. This error should never occur.
?5	SRCCOM	Could not create output file.
AA	F4	More than six subroutine arguments are arrays.
ALREADY EXISTS (filename)	FOTP	An attempt was made to rename an output file with the name of an existing output file.

Message	Program	Explanation
AMBIGUOUS SWITCH	CCL	Not enough characters of the command full word option were specified to make it unique. (See Appendix H.)
ARE YOU SURE?	PIP	Occurs when using the SQUISH command. A response of Y will compress the files.
AS	F4	Bad ASSIGN statement.
BAD ARG	FRTS	Illegal argument to library function.
BAD ARGS	Monitor	The arguments to the SAVE command are not consistent and violate restrictions.
BAD COMMAND LINE	FORMAT	The command line contains a syntax error. Retype the line correctly.
BAD DATE	Monitor	The date has not been entered correctly, or incorrect arguments were used, or the date was out of range.
BAD DEVICE	CCL	The device specified in a system command is not of the correct form.
BAD DIRECTORY ON DEVICE #n	PIP	PIP is trying to read the directory, but it is not a legal OS/78 directory.
BAD DISK	FORMAT	Disk pack cannot be reformatted. There are more than 63 bad blocks on the system area (blocks 0-70 octal) contains bad blocks.
BAD EXTENSION	CCL	Either an extension was specified without a file name or two extensions were specified.
BAD INPUT DIRECTORY	DIRECT FOTP	The directory on the specified input device is not a valid OS/78 directory.
BAD INPUT FILE	LOAD	An input file was not a RALF module.
BAD INPUT, FILE #n	ABSLDR	Attempt was made to load a non-binary file as file number n of the input file list; or a non-memory image with /I option.
BAD INPUT, FILE #n	BITMAP	A physical end of file was reached before a logical end of file, or extraneous characters were found in binary file n.

Message	Program	Explanation
#BAD LINE. JOB ABORTED	BATCH	BATCH detected a record in the input file that did not have one of the characters period, slash, dollar sign, or asterisk as the first character of the record. The record is ignored, and BATCH scans the input file for the next $JOB record.
BAD NUMBER	CCL	The =n option was used, where n was not in the correct octal format.
BAD OUTPUT DEVICE	FOTP	This message usually appears when a non-file structured device is specified as the output device for a file-oriented operation.
BAD OUTPUT DEVICE	LOAD	The loader image file device was not a directory device, or the symbol map file device was a read-only device. The entire command is ignored.
BAD OUTPUT DIRECTORY	FOTP	The directory on the specified output device is not a valid OS/78 device directory.
BAD RECOLLECTION	CCL	An attempt was made to use a previously remembered argument after the system date had been changed.
BAD SYSTEM HEAD	PIP	Attempt to copy an obsolete version of the system head.
BAD SWITCH OPTION	CCL	The character used with a slash (/) to indicate an option is not a legal option.
BATCH.SV NOT FOUND ON SYS:	BATCH	A copy of BATCH.SV must exist on the system device. Control returns to the OS/78 Monitor.
%BATCH SQUISHING SYS:!	BATCH	Batch is running and attempting to squish the system.
BD	F4	Bad dimensions (too big, or syntax) in DIMENSION, COMMON or type declarations.
BE	RALF	Illegal equate. The symbol had been defined previously.
BE	PAL8	PAL8 internal table has overflowed. Fatal error; assembly cannot continue.
BI	RALF	Illegal index register specification.
BO	FRTS	No more file buffer available.

Message	Program	Explanation
BS	F4	Non-declarative statement used BLOCK DATA program.
BX	RALF	Bad expression. Something in the expression is incorrect, or the expression is not valid in this context.
CANNOT CHANGE MEMORY CAPACITY WHILE RUNNING BATCH	CCL	A MEMORY command was issued while BATCH was running.
?CAN'T-DEVICE DOESN'T EXIST	SET	A nonexistent device was referenced.
?CAN'T-DEVICE IS RESIDENT	SET	No modifications are allowed to the system handler.
?CAN'T-HANDLER ALREADY RESIDENT	SET	Attempt to replace a handler with one already in the system head.
?CAN'T-OBSOLETE HANDLER	SET	Obsolete version of the handler.
CANT-TOO MANY LOGICAL DEVICES	SET	The system does not allow you to have more than 12 logical devices (excluding SYS, DSK, and TTY). The handler that you are attempting to insert will cause the number of logical devices to exceed 12.
?CAN'T-UNKNOWN VERSION OF THIS HANDLER	SET	This handler's version number is not recognized; possibly a newer version.
CAN'T OPEN OUTPUT FILE	PIP	Message occurs due to one of the following: 1. Output file is on a read-only device. 2. No name has been specified for the output file. 3. Output file has zero free blocks.
CAN'T READ IT	FRTS	I/O error on reading loader image file.
%CAN'T REMEMBER	CCL	The argument specified in a CCL command line is too long to be remembered or an I/O error occurred.
CAUTION - THIS COMMAND DESTROYS CONTENTS OF SYSTEM DISK. CHANGE DISKS BEFORE PROCEEDING OR CTRL/C.	RLFMT	Drive 0 was selected for formatting. Be sure to replace your system disk pack with the one that you want to format.
CF	PAL8	CREF.SV not on SYS:

Message	Program	Explanation
CH	PAL8	Chain to CREF error -- CREF.SV was not found on SYS:.
CHAIN ERROR	USR Monitor	Chain error
CL	F4	Bad COMPLEX literal.
CLOSE ERROR	USR Monitor	Close error
CLOSE FAILED	CREF	CLOSE on output file failed.
CO	F4	Syntax error in COMMON statement.
COMMAND LINE OVERFLOW	CCL	The command line specified with @ construction is more than 512 characters in length.
COMPARE ERROR	RXCOPY	A compare error has been detected at the track and sector specified.
CONTRADICTORY SWITCHES	CCL	Either two CCL processor switches were specified in the same command line or the file extension and the processor switch do not agree.
CORE IMAGE ERR	Monitor	Cannot run system program.
DA	F4	Bad syntax in DATA statement.
DE	F4	This type of statement illegal as end of DO loop.
DE	PAL8	Device error. An I/O failure was detected when trying to read or write a device. Fatal error-assembly cannot continue.
DELETES PERFORMED ONLY ON INPUT DEVICE GROUP 1 CAN'T HANDLE MULTIPLE DEVICE DELETES	FOTP	More than one input device was specified in the DELETE command when no output specification (device or filename) was included.
DEV LPT BAD	CREF	The default output device, LPT, cannot be used as it is not available on this system.
DEV NOT IMPLE-MENTED	BATCH	BATCH cannot accept input from the specified input device because its handler is not SYS: Control returns to the Command Decoder.
DEVICE DOES NOT HAVE A DIRECTORY	DIRECT	The input device is a non-directory device; for example, TTY. DIRECT can only read directories from file-structured devices.

Message	Program	Explanation
DEVICE FULL	HELP	Output device storage capacity exhausted.
DEVICE HANDLER NOT IN CORE	USR Monitor	Handler for device specified not in memory.
DEVICE IS NOT RX	RXCOPY	One or more of the devices specified to RXCOPY were not RX diskettes.
DF	PAL8	Device full. Fatal error -- assembly cannot continue.
DF	F4	Bad DEFINE FILE statement.
D.F. TOO BIG	FRTS	Product of number of records times number of blocks per record exceeds number of blocks in file.
DH	F4	Hollerith field error in DATA statement.
DIRECTORY I/O ERROR	USR Monitor	An I/O error occurred while attempting to read or write a directory block.
DIRECTORY OVERFLOW	USR Monitor	Directory overflow occurred.
DIVIDE BY	FRTS	Attempt to divide by zero. The resulting quotient is set to zero and execution continues.
DL	F4	Data list and variable list are not same length.
DN	F4	DO-end missing or incorrectly nested. It is followed by the statement number of the erroneous statement rather than the ISN.
DO	F4	Syntax error in DO or implied DO.
name DOES NOT EXIST	CCL	The device with the name given is not present on the OS/78 system.
DP	F4	DO loop parameter not integer or real.
EG	RALF	The preceding line contains extra code which could not be used by the assembler.
ENTER FAILED	CREF	Entering an output file was unsuccessful -- possibly output was specified to a read-only device.
EOF ERROR	FRTS	End of file encountered on input.
EQUALS OPTION BAD	DIRECT	The =n option is not in the correct range.

Message	Program	Explanation
ERROR CLOSING FILE	DIRECT	Output device was read-only.
ERROR IN COMMAND	CCL	A command not entered directly from the console terminal is not a legal CCL command. This error occurs when the argument of a UA, UB, or UC command was not a legal command.
ERROR ON INPUT DEVICE SKIPPING (filename)	FOTP	The file specified is not transferred (due to a hardware I/O error), but any previous or subsequent files are transferred and indicated in the new directory. Any file of the same name may have been deleted from the output device.
ERROR ON OUTPUT DEVICE	BITMAP	A hardware I/O error occurred while writing on output device.
ERROR ON OUTPUT DEVICE SKIPPING (filename)	FOTP	The file specified is not transferred, but any previous or subsequent files are ransferred and indicated in the new directory.
ERROR READING INPUT DIRECTORY	DIRECT FOTP	A hardware I/O error occurred while reading the directory.
ERROR WRITING FILE	DIRECT	A hardware I/O error occurred while writing the output file.
ERROR WRITING OUTPUT DIRECTORY	FOTP	A hardware I/O error has occurred. The output device has probably been corrupted (directory no longer describes current files).
ES	RALF	External symbol error.
EX	F4	Syntax error in EXTERNAL statement.
FETCH ERROR	HELP	Cannot initialize device handler.
FILE ERROR	FRTS	Any of the following: a. A file specified as an existing file was not found. b. A file specified as a non-existing file would not fit on the designated device. c. More than 1 nonexistent file was specified on a single device. d. File specification contained "*" as name or extension.
FILE OVERFLOW	FRTS	Attempt to write outside file boundaries.

Message	Program	Explanation
FL	RALF	An error has occurred in an integer-to-real conversion routine.
FORMAT ERROR	FRTS	Illegal syntax in FORMAT statement.
FP	RALF	A syntax error was encountered in a real constant.
FULL *	EDIT	The specified output device has become full. The file is closed; the user must specify a new output file.
GT	F4	Syntax error in GO TO statement.
GV	F4	Assigned or computed GO TO variable must be integer or real.
HO	F4	Hollerith field error.
IC	RALF	The symbol or expression in a conditional is improperly used, or left angle bracket is missing. The conditional pseudo-op is ignored.
IC	PAL8	Illegal character. The character is ignored and the assembly continued.
ID	PAL8	Illegal redefinition of a symbol.
IE	F4	A hardware I/O error on reading input file. Control returns to the Monitor.
IE	PAL8	Illegal equals -- an attempt was made to equate a variable to an expression containing an undefined term. The variable remains undefined.
IE	RALF	An entry point has not been defined, or is absolute, or also is defined as a common section, or external.
IF	F4	Logical IF statement cannot be used with DO, DATA, INTEGER, etc.
II	PAL8	Illegal indirect -- an off-page reference was made; a link could not be generated because the indirect bit was already set.
ILLEGAL *	DIRECT FOTP	An asterisk (*) was included in the output file specification or an illegal * was included in the input file name.

Message	Program	Explanation
ILLEGAL * OR ?	CCL	An * or ? was used in a CCL command that does not accept the wild card construction.
ILLEGAL ?	DIRECT FOTP	A question mark (?) was included in the output file specification.
ILLEGAL ARG	Monitor	The SAVE command was not expressed correctly; illegal syntax used.
#ILLEGAL INPUT	BATCH	A file specification designated TTY or LPT as an input device when the initial dialogue indicated that an operator is not available. The current job is aborted, and BATCH scans the input file for the next $JOB command record.
ILLEGAL ORIGIN	Loader	A RALF routine tried to store data outside the bounds of its overlay.
ILLEGAL SPOOL DEVICE	BATCH	The device specified as a spooling output device must be file-structured. Control returns to the Command Decoder.
ILLEGAL SYNTAX	CCL	The command line was formatted incorrectly.
?ILLEGAL WIDTH	SET	A width that was 0 or too large was specified; for the TTY, a width of 128 or one not a multiple of 8 was specified.
INCOMPATIBLE MONITOR - OS/78 V3.0 EXPECTED	SET	The system head contains an old version of the OS/78 Monitor. You cannot change handlers in a system head that has an old version of the Monitor.
INPUT DEVICE READ ERROR	RXCOPY	Bad sector(s), operation continues.
INPUT ERROR	CREF	A hardware I/O error occurred while reading the file.
	FRTS	Illegal character received as input.
INPUT ERROR READING INDIRECT FILE	CCL	CCL cannot read the file specified with the @ construction due to hardware I/O error.
#INPUT FAILURE	BATCH	Either a hardware problem prevented BATCH from reading the next line of the input file, or BATCH read the last record of the input file without encountering a $END command record.

Message	Program	Explanation
INSUFFICIENT MEMORY FOR BATCH RUN	BATCH	OS/78 BATCH requires 12K of memory to run. Control returns to the OS/78 Monitor. Type MEMORY 3 and try again.
I/O	RALF	I/O error (fatal error).
I/O ERROR	FRTS	Error reading or writing a file, tried to read from an output device, or tried to write on an output device.
I/O ERROR, FILE #n	ABSLDR BITMAP	An I/O error has occurred in input file number n.
IO ERROR IN (file name) -- CONTINUING	PIP	An error has occurred during a SQUISH transfer.
I/O ERROR ON SYS:	CCL	An error occurred while doing I/O to the system device. The system must be rebootstrapped.
?I/O ERROR ON SYS:	SET	An I/O error occurred while trying to read or rewrite the handler.
I/O ERROR TRYING TO RECALL	CCL	An I/O error occurred while CCL was trying to remember an argument.
I/O READ ERROR	SET	A hardware error occurred while reading from the system device.
I/O WRITE ERROR	SET	A hardware error occurred while writing to the system device.
IP	PAL8	Illegal pseudo-op -- a pseudo-op was used in the wrong context or with incorrect syntax.
IX	RALF	An index register was specified for an instruction which cannot accept one.
IZ	PAL8	Illegal page zero reference -- The pseudo-op was found in an instruction which did not refer to page zero. The Z is ignored.
LD	PAL8	The /L or /G options have been specified and ABSLDR.SV is not present on the system.
LG	PAL8	Link Generated -- only printed if the /E switch was specified to PAL8.
LI	F4	Argument of logical IF is not of type Logical.
LOADER I/O ERROR	LOAD	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.

Message	Program	Explanation
LT	F4	Input line too long, too many continuations.
LT	RALF	The line is longer than 128 characters. The first 127 characters are assembled and listed.
#MANUAL HELP NEEDED	BATCH	BATCH is attempting to operate an I/O device, such as a terminal, that will require operator intervention.
MD	RALF	The tag on the line has been previously encountered at another location or has been used in a context requiring an absolute expression.
MIXED INPUT	LOAD	The L option was specified on a line that contained some file other than a library file. The library file (if any) is accepted. Any other input file specification is ignored.
MK	F4	Misspelled keyword.
ML	F4	Multiply-defined line number.
MM	F4	Mismatched parenthesis.
MO	F4	Operand expected but not found.
MONITOR ERROR 5 AT xxxx (I/O ERROR ON SYS=)	Monitor	A hardware I/O error occurred while doing I/O to the system device.
MONITOR ERROR 6 AT xxxx (DIRECTORY OVERFLOW)	Monitor USR	A directory overflow has occurred (no room for tentative file entry in directory).
#MONITOR OVERLAYED	BATCH	The Command Decoder attempted to call the BATCH monitor to accept and transmit a file specification, but found that a user program had overlayed part of all of the BATCH monitor. Control returns to the monitor level, and BATCH executes the next Monitor command.
MORE CORE REQUIRED	FRTS	The space required for the program, the I/O device handlers (I/O buffers) and the resident Monitor exceeds the available memory.
MT	F4	Operand of mixed type or operator does not match operands.

Message	Program	Explanation
MULT SECT	Loader	Any combination of entry point, COMMON section (with the exception of multiple COMMONs) or program section of the same name causes this error.
NE	RALF	Number error. A number out of range was specified or an 8 or 9 occurred while in octal radix (internal error during FORTRAN assembly pass).
NO!!	Monitor	The user attempted to start (with .ST) a program that is no longer in memory.
NO CCL!!	Monitor	CCL.SV is not present on the system device or an I/O error occurred on reading it.
NO DEFINE FILE	FRTS	Direct access I/O attempted without a DEFINE FILE statement.
NO FILES OF THE FORM xxxx	FOTP	No files of the form (xxxx) specified were found.
NO HELP	HELP	No help information is present on the command given.
NO HELP FILE	HELP	No help file is on the system device.
NO/I	BITMAP	Cannot produce a bitmap of an image file.
NO/I!	ABSLDR	Use of /I is prohibited after the first file.
NO INPUT	ABSLDR BITMAP	No input or binary file was found on the designated device.
NO INPUT DEVICE	RXCOPY	The command line lacks the required parameters.
NO OUTPUT DEVICE	RXCOPY	The command line lacks the required parameters.
NO MAIN	LOAD	No RALF module contained section #MAIN.
NO NUMERIC SWITCH	FRTS	The referenced FORTRAN I/O unit was not specified to the run-time system.
NO ROOM FOR OUTPUT FILE	DIRECT PIP	Either room on device or room in directory is lacking.
NO ROOM IN (file name) -CONTINUING	PIP	Occurs during the SQUISH command. The output device cannot contain all of the files on the input device.

Message	Program	Explanation
NO ROOM, SKIPPING (filename)	FOTP	No space is available on the output device to perform the transfer. Predeletion may already have occurred.
NOT A LOADER IMAGE	FRTS	The first input file specified to the run-time system was not a loader image file.
name NOT AVAILABLE	Monitor	The device with the name given is not listed in any system table, or it is not available for use at the moment, or the user tried to obtain input from an output-only device.
NOT ENOUGH MEMORY	CCL	The number specified in a MEMORY command is greater than is available in the system.
name NOT FOUND	CCL Monitor	The file name designated in the command was not found in the device directory.
?NUMBER TOO BIG	SET	The number specified was out of range.
OF	F4	I/O error while writing output file. Control returns to the Monitor.
OLD HANDLER NOT FOUND IN MONITOR	SET	You attempted to replace a handler that is not in the system head.
OP	F4	Illegal operator.
OPTION UNKNOWN XXXXX	CCL	
OS78 ENTER ERROR	LOAD	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.
OT	F4	Type/operator use illegal (for example, A.AND.B where A and/or B not of type Logical).
OUT DEV FULL	CREF	The output device is full (directory devices only).
OUTPUT DEVICE READ ERROR	RXCOPY	Bad sector(s), operation continues.
OUTPUT DEVICE WRITE ERROR	RXCOPY	Bad sector(s), operation continues.
OVER CORE	LOAD	The loader image requires more memory than is available.
OVER IMAG	LOAD	Output file overflow in the loader image file.

Message	Program	Explanation
OVER SYMB	LOAD	Symbol table overflow. More than 253 (decimal) symbols in one FORTRAN job.
OVERFLOW	FRTS	Result of a computation exceeds upper bound for that class of variable. The result is set equal to zero and execution continues.
PARENS TOO DEEP	FRTS	Parentheses nested too deeply in FORMAT statement.
PD	F4	Compiler stack overflow; statement too big and/or too many nested loops.
PE	PAL8	Current non-zero page exceeded.
PH	F4	Bad program header line.
PH	PAL8	A conditional assembly bracket is still in effect at the end of the input stream -- this is caused by nonmatching < and > characters in the source.
QL	F4	Nesting error in EQUIVALENCE statement.
QS	F4	Syntax error in EQUIVALENCE statement.
RD	F4	Attempt to redefine the dimensions of an array.
RD	PAL8	A permanent symbol has been redefined using =. The new and old definitions do not match. The redefinition is allowed.
RE	RALF	Relocatability error. A relocatable expression has been used in a context requiring an absolute expression.
READ ERR	HELP	Cannot read input device.
READY. STRIKE CARRIAGE RETURN TO CONTINUE	RLFMT	The /P option was selected. Mount the disk pack to be formatted and type a carriage return to begin formatting.
RT	F4	Attempt to redefine the type of a variable.
RW	F4	Syntax error on READ/WRITE statement.
SAVE ERROR	Monitor	An I/O error has occurred while saving the program. The program remains intact in memory.

Message	Program	Explanation
SE	PAL8	Symbol table exceeded -- too many symbols have been defined for the amount of memory available for the symbol table. Fatal error -- assembly cannot continue.
SF	F4	Bad arithmetic statement function.
SN	F4	Illegal subroutine name in CALL.
SORRY -- NO INTERRUPTIONS	PIP	^C (CTRL/C) was typed during a SQUISH; the transfer continues.
#SPOOL TO FILE BTCHA1	BATCH	Where the "A" may be any character of the alphabet and the "1" may be any decimal digit. This message indicates that BATCH has intercepted a non-file structured output file and routed it to the spool device. This is not, generally, an error condition.
SS	F4	Error in subscript expression; i.e., wrong number, syntax.
ST	F4	Compiler symbol table full, program too big. Causes an immediate return to the Keyboard Monitor.
ST	RALF	User symbol table overflow (fatal error).
SWITCH NOT ALLOWED HERE	CCL	Either a CCL option was specified on the left side of the < or was used when not allowed.
SY	F4	System error; i.e., PASS20.SV or PASS2.SV missing, or no room for output file. Causes an immediate return to the Keyboard Monitor.
SYM OVERFLOW	CREF	More than 896 (decimal) symbols and literals were encountered. Try again using /M option.
?SYNTAX ERROR	SET	Incorrect format used in SET command or NO specified when not allowed.
#SYS ERROR	BATCH	A hardware I/O error occurred during BATCH operation.
SYSTEM DEVICE ERROR	FRTS	I/O failure on the system device.
SYSTEM ERR	Monitor	An error occurred while doing I/O to the system device. The system should be rebootstrapped.

Message	Program	Explanation
SYSTEM ERROR	LOAD	Fatal error message indicating that an error was detected by OS/78 while trying to perform a USR function.
SYSTEM ERROR -- CLOSING FILE	FOTP	Self-explanatory.
TD	F4	Bad syntax in type declaration statement.
TOO FEW ARGS	Monitor	An important argument has been omitted from a command.
TOO MANY FILES	CCL	Too many files were included in a CCL command.
TOO MANY HANDLERS	FRTS	Too many I/O device handlers are resident in memory, or files have been defined on too many devices.
TOO MANY RALF FILES	LOAD	More than 128 input files were specified.
?UNKNOWN ATTRIBUTE FOR DEVICE dev	SET	An illegal attribute was specified for the given device.
UNIT ERROR	FRTS	I/O unit not assigned, or incapable of executing tne requested operation.
UO	PAL8	Undefined origin -- an undefined symbol has occurred in an origin statement.
US	RALF PAL8	Undefined symbol in an expression.
USER ERROR	FRTS	Illegal subroutine call, or call to undefined subroutine. Execution continues only if the E option was specified.
USER ERROR 0 AT xxxx	Monitor	An input error was detected while loading the program. xxxx refers to the Monitor location where the error was generated.
VE	F4	Version error. One of the compiler programs is absent from SYS.
WRITE ERR	HELP	Cannot output to device.
WRONG FLOPPY TYPE	RXCOPY	Attempt to duplicate from an RX02 drive with a double-density diskette to an RX01 single-density diskette drive.

Message	Program	Explanation
xxxxxx.HN NOT FOUND ON SYS	SET	The handler that you want to insert into the system head does not reside on the system device (SYS), or does not have a .HN extension.
ZE	PAL8	Page 0 exceeded -- same as PE except with reference to page 0.
ZERO SYS?	PIP	If any attempt is made to zero the system device directory, this message occurs. Responding with Y causes the directory to be zeroed; any other character prevents destruction of the system directory.

APPENDIX H

OS/78 MULTIFUNCTION OPERATION

DECstation 78 series systems can perform multifunction operations; that is, these systems can run two tasks simultaneously. One task is the normal OS/78 job stream, that runs in the first 12K of memory. The second task (the symbiont) runs in the upper 4K of memory (field 3). The symbiont is an interrupt-driven task that you write in PAL8 assembly language. OS/78 and the symbiont share a common floppy disk and line printer.

BASIC, FORTRAN and PAL8 programs can be compiled, but only BASIC and PAL8 programs can run while a symbiont is active. You cannot run FORTRAN programs because the FORTRAN run time system performs interrupt-driven I/O, which would interfere with symbiont operation.

A demonstration symbiont, called spoolr comes with OS/78. You can use the SPOOLR to queue and print listings on the line printer while you do other OS/78 work at the terminal.

H.1 SYMBIONT COMMANDS

OS/78 commands that control symbiont operation are as follows:

REQEST symbiont-name

This command automatically sets memory for OS/78 to 12K (MEM 2) and starts up the named symbiont. The default extension for a symbiont save image is .SM; if .SM is not found, the auxiliary default extension is .SV.

CANCEL

This command cancels the currently running symbiont and returns OS/78 to operation in 16K memory.

CAUTION

Do not use the MEM command while a symbiont is running.

H.2 SPOOLER COMMANDS

Type the following command to start SPOOLR.

.REQ SPOOLR

This command starts the spooler.

You can use the following commands once the SPOOLR program is running.

.QUEue filename list, .../options,...

Enter the specified lists of files for printing on the line printer. Options for the QUEUE command are:

/L — Lists contents of queue and prints version number of QUEUE and SPOOLR

/C:n — Specifies the number (n) of copies to be printed (in octal)

/N — A block letter header will not be added to the files queued

/2 — Prints two pages of block letter header rather than one.

H.3 CUSP CODE CONVENTION

You should observe the following conventions when writing OS/78 programs if you want to run them while a symbiont is running.

1. If the program loads into page 0 of field 0, then it must contain the following code:

```
*1
CIF 30
JMP .-1
```

   Also, locations 0, 1, and 2 must not be used as scratch or as data.

2. The program must not use page 0 of field 0 as a buffer or data area. However, you may swap the OS/8 command decoder in this area.

3. The program must not turn interrupts on or off.

4 The program must not modify the software memory size.

5. The program must not use field 3.

H.4 WRITING A SYMBIONT

To write your own symbiont, you must know the PAL8 assembly language and the PDP-8 architecture. You must observe the following conventions when you write a symbiont.

1. The symbiont runs only in field 3.

2. The symbiont must turn on interrupts and must disable keyboard interrupts (KIE with AC=1).

3. The CANCEL command branches to the symbiont at location 30003.

4. The symbiont will receive interrupts at location 30001. After saving the AC, Link, etc., the symbiont should pick up location 0 in field 0 and save it in location 0 of field 3.

APPENDIX I

FULL WORD COMMAND SWITCHES

OS/78 allows you to enter complete words in place of most of the standard single-character command options. For example, this command

```
.DIR/BRIEF
```

is the same as

```
.DIR/F
```

This appendix lists the switch options and the full word equivalents that you can use in OS/78 commands. Keep in mind the following rules and features.

- The standard single-character slash options remain legal and are the only single-character switches you should attempt to use in a command.
- Only the first six characters of a full word are significant.
- Every character in a full word switch must be legal. For example, /QUIEX is an illegal switch and will produce the message UNKNOWN SWITCH.
- You may abbreviate any switch option to the number of characters (at least two) necessary to determine the switch uniquely for the command in question. For example, in the DIRECT command, /EXT is a valid abbreviation for the /EXTENDED switch. /EX, however, is not valid because the switch /EXCEPT also begins with these two characters.
- If you fail to type enough characters to identify a switch uniquely, you will receive an AMBIGUOUS SWITCH message.
- You may still enclose single-character switches in parentheses. (ABC) in a command line means the same as A/B/C. However, you may not enter full word switches in this manner.
- :n is the same as the standard =n and may optionally follow any full word switch. Thus /IMAGE:24 is the same as /I=24.

Command	Single-Character Switch	Full Word Switch
COMPARE	/B	/BLANKS
	/C	/NOCOMMENTS
	/S	/NO SPACES
	/T	/TABS
	/X	/NOPRINTCOMMENTS
COMPILE (see PAL and BASIC)	/G	/GO
	/N	/NOISN
	/Q	/OPTIMIZE
COPY	/C	/CURRENT
	/D	/NOCOPY
	/F	/FAILSAFE
	/N	/NOPREDELETE
	/O	/OTHER
	/Q	/QUERY
	/R	/RENAME
	/T	/TODAY
	/U	/INDEPENDENTLY
	/V	/EXCEPT
CREATE	/B	/SPACES
CREF	/E	KEEP
	/M	MAMMOTH
	/P	NOLIST
	/U	NOSYNTAB
	/X	NOLITERALS
DELETE	/C	/CURRENT
	/O	/OTHER
	/Q	/QUERY
	/T	/TODAY
	/U	/INDEPENDENTLY
	/V	/EXCEPT
DIRECT	/B	/BLOCKS
	/C	/CURRENT
	/E	/EXTENDED
	/F	/FAST
	/M	/EMPTIES
	/O	/OTHER
	/R	/REMAINDER
	/U	/INDEPENDENTLY
	/V	/EXCEPT
	=n	/COLUMNS:n
DUPLICATE	/M	/NOCOPY
	/N	/NOMATCH
	/P	/PAUSE
	/R	/READYONLY
EDIT	/B	/SPACES
	/D	/DELETE

FULL WORD COMMAND SWITCHES

Command	Single-Character Switch	Full Word Switch
FORMAT RL01	/0	/ZERO
	/1	/ONE
	/P	/PAUSE
LIST	/C	/CURRENT
	/O	/OTHER
	/Q	/QUERY
	/U	/INDEPENDENTLY
	/V	/EXCEPT
LOAD	/G	/GO
	/I	/IMAGE
	/S	/MULTIPLE
MAP	/S	/MULTIPLE
	/T	/INVERT
PAL	/B	/SHIFT
	/C	/CREF
	/E	/NOLINKS
	/F	/NOFILL
	/G	/GO
	/H	/NONPAGINATED
	/J	/NOCONDITIONALS
	/L	/LOAD
	/N	/NOLIST
	/O	/NOORIG
	/S	/NOSYMTAB
	/W	/NOREMEMBERLITERALS
QUEUE	/C=n	/COPIES:n
	/L	/LIST
	/N	/NOH
RENAME	/C	/CURRENT
	/O	/OTHER
	/Q	/QUERY
	/T	/TODAY
	/U	/INDEPENDENTLY
	/V	/EXCEPT
	/W	/VERSION
SQUISH	/O	/OK
SUBMIT	/E	/NONFATAL
	/H	/HUSH
	/Q	/QUIET
	/T	/TERMINAL
	/U	/UNATTENDED
	/V	/VERSION
TYPE	/C	/CURRENT
	/O	/OTHER
	/Q	/QUERY
	/T	/TODAY
	/U	/INDEPENDENTLY
	/V	/EXCEPT

APPENDIX J

DECstation Hardware Configuration Summary

MODEL	CPU	Console Terminal	Disk System	Line Printer (optional)	Serial I/O Ports*
78/40	VT78 (16K)	VT78	1 RX01	LA78 or LQP78	2
78/50	VT78 (16K)	VT78	1 RX02	LA78 or LQP78	2
78/60	VT78 (16K)	VT78	2 RX01	LA78 or LQP78	2
78/70	VT78 (16K)	VT78	2 RX02	LA78 or LQP78	2
88/50	8A205 (32K)	VT100 or LA36	1 RX02	LA180 or LQP78	2 (optional)
88/70	8A205 (32K)	VT100 or LA36	2 RX02	LA180	2 (optional)
88/80	8A205 (32K)	VT100 or LA36	2 RX02 1 RL01	LA180	2 (optional)
88/90	8A205 (32K)	VT100 or LA36	2 RL01	LA180	2 (optional)
88/92	8A425 (32K)	VT100 or LA36	2 RL01	LA180	2 (optional)
88/97	8A425 (64K)	VT100 or LA36	2 RL01	LA180	2 (optional)

* Can be connected to LA34/38, LA36, LA120, VT52, VT100 or equivalent.

GLOSSARY

ABSOLUTE ADDRESS
: A number that is permanently assigned as the address of a memory storage location.

ACCESS TIME
: The interval between the instant at which a data transfer is requested and the instant at which the data actually starts transferring.

ACCUMULATOR
: The register in which the hardware arithmetic operations are performed (abbreviated AC).

ADDRESS
: 1. A name or a number which identifies a location in memory, either within a field (12-bits wide) or within all available memory (15-bits wide with left-most bit 0).

  2. The part of an instruction that specifies the location of the operand of that instruction.

ALGORITHM
: A prescribed sequence of well-defined rules or processes for the solution of a problem.

ALPHANUMERIC
: Pertaining to the character set that contains only letters and numbers.

ARGUMENT
: A variable or constant which is given in the call of a subroutine as information to it; the independent variables of a function; the known reference factor necessary to find an item in a table or array (that is, the index).

ARRAY
: An ordered set of data values. An n-dimensional array is a table having n dimensions.

ASCII
: American Standard Code for Information Interchange. Established by American Standards Association to standardize a binary code for printing and control characters.

ASSEMBLE
: To translate from a source program to a binary program by substituting binary operation codes for mnemonic operation codes and absolute or relocatable addresses for symbolic addresses.

ASSEMBLER
A program which translates assembly language instructions into machine language and assigns memory locations for variables and constants.

ASSEMBLY LANGUAGE
A symbolic language that translates directly into machine language instructions. Usually there is a one to-one relation between assembly language instructions and machine language instructions.

AUTO-INDEXING
A property of the autoindex registers (locations 0010 through 0017. When one of these locations is addressed indirectly, the contents of that location are incremented by one, rewritten into that same location and then used as the effective address of the current instruction.

AUXILIARY STORAGE
Storage that supplements memory such as a diskettes and disk packs.

BASIC
A high-level easy to learn programming language for arithmetic and string computations Developed by Dartmouth College.

BATCH PROCESSING
A method of using OS/78 with no operator interaction.

BINARY
Pertaining to the number system with a base (or radix) of two. In this system numbers are represented by strings of 1's and 0's.

BINARY CODE
A code that makes use of two distinct values: 0 and 1.

BIT
Contraction of "Binary digit", a bit is the smallest unit of information in the binary system of notation.

BIT MAP
A method of keeping track of used and unused entities by assigning one bit in a table to each entity.

BLOCK
A set of consecutive machine words, characters or digits handled as a unit, particularly with reference to input and output; an OS/78 block is 400 octal contiguous words (two memory pages).

BOOTSTRAP
A program of instructions that are executed when the START pushbutton is pressed or you use the BOOT command. The purpose of a bootstrap is to load and start the OS/78 Monitor.

BREAKPOINT
A location in a program at which that program's execution may be suspended, so that partial results can be examined via ODT.

BUFFER
An area that is usually used for temporary storage. Buffers are often used to hold data being passed between processes or devices which operate at different speeds or different times.

BUG
A mistake in the design or implementation of a program resulting in erroneous results.

BYTE
A group of binary digits usually operated upon as a unit, especially one of six bits.

CALL
To transfer control to a specified routine; to invoke a system command.

CALLING SEQUENCE
A specified arrangement of instructions and data necessary to pass parameters and control to a given subroutine.

CARTRIDGE DISK PACK
A removable data storage medium consisting of a thin disk coated with a magnetic recording material and enclosed in a two-piece protective cover.

CCL(CONCISE COMMAND LANGUAGE)
A system program that simplifies the entry of OS/78 commands by calling the selected system program and decoding its command string.

CHAINING
A program technique which involves dividing a program into sections with each section terminated by a call to the next section.

CHARACTER
A single letter, number or symbol (printing or non-printing) used to represent information.

CLEAR
To erase or reset the contents of a memory location or hardware register.

CLOCK
A hardware device that generates periodic program interrupts when enabled.

COMPILE
To translate a source program written in a high-level language, such as BASIC or FORTRAN, into a binary-coded program. In addition to translating the source language, appropriate subroutines may be selected from a subroutine library and output in binary code along with the main program.

COMPILER
A program which compiles entire high-level language source programs into binary coded programs.

CONCATENATION
The adjacent joining of two strings of characters to produce a longer string.

CODE
To write instructions for a computer using symbols meaningful to the computer or to an assembler, compiler, or other language processor.

COMMAND
A directive from the user to the system entered from the keyboard or a BATCH input file.

COMMAND DECODER
A part of OS/78 that interprets file and option specification strings typed by the user.

CONDITIONAL ASSEMBLY
The processes of translating specific sections of an assembly language program into machine code only if certain conditions have been met during the assembly process.

CONFIGURATION
A particular selection of computer, peripherals and interfacing equipment that are to function together.

CONSTANT
Numeric data used but not changed by a program.

CONTROL
Memory address at which the next instruction will be executed. When "control is passed" to a location, the instruction contained in that location will be the next one to be executed.

CONVERSATIONAL PROGRAM
A program which interacts dynamically with on-line users, that is, an interactive program.

COUNTER
A variable or memory location used to control the number of iterations of a program loop.

CPU
Central Processing Unit, the portion of a computer that executes most instructions.

CRASH
Fail Totally. When a system crashes, it will not function at all and must be restarted.

CRC
Cycle Redundancy Check. An error detection technique used for detecting incorrectly read bits in file-structured devices.

CREATE
To open, write, and close a file for the first time.

CROSS REFERENCE PROGRAM (CREF)
A program that generates a sequence-numbered listing file and a table that contains line numbers of all referenced user-defined symbols and literals and their usage.

CYCLE TIME
A basic unit of time in a computer, usually equal to the memory read time plus memory write time. Computer instructions usually execute in multiples of the cycle time.

DATA
A general term used to denote any or all computer-represented facts, numbers, letters and symbols.

DEBUG
To detect, locate, and correct mistakes in a program.

DEFAULT
A parameter that is assumed by the system when none is explicitly specified.

DELIMITER
A character that separates and organizes elements of data, particularly the symbols of a programming language.

DEVICE
A peripheral hardwre I/O unit and/or a removable date volume mounted on it.

DEVICE CODES
Numbers assigned to each device in the system, used in the computer instructions for those devices.

DEVICE DRIVERS
See Device Handlers.

DEVICE HANDLERS
Routines that perform I/O for specific devices in a standard format. These routines also handle error recovery and provide device independence.

DEVICE INDEPENDENCE
The ability of a computer system to divert the input or output of an executing program from one device to another, either automatically if the specified device is out of order, or by a keyboard command to the Monitor.

DIAGNOSTIC
Pertaining to the detection and isolation of hardware malfunctions.

DIGITAL
Representation of information by discrete units.

DIRECT ACCESS
Same as Random Access.

DIRECT ADDRESS
A number that specifies the location of an instruction operand.

DIRECTORY
A reserved storage area on a mass storage device that describes the layout of the data on that device in terms of file names, length, location, and creation date.

DISKETTE
A removable data volume, consisting of a thin disk coated with a magnetic data-storage material. Also called a floppy disk.

DISK PACK
See cartridge disk pack.

DUMMY ARGUMENTS
Symbolic names used only as placeholders for other actual arguments that will be uniformly substituted for them at a later time. Used within programming language function definitions to represent the independent (supplied) variables.

ECHO
The displaying by the terminal of characters typed on the keyboard.

EDITOR
A program which interacts with the programmer or typist to enter new programs into the computer and edit them as well as modify existing programs.

EFFECTIVE ADDRESS
The address actually used to fetch an instruction operand, that is, the specified address modified by indexing or indirect addressing rules.

ENTRY POINT
A point in a subroutine to which control is transferred when the subroutine is called.

ERROR MESSAGE
A message from computer system to programmer that reports a hardware malfunction or an incorrect format or operation detected by software.

EXECUTE
To cause the computer to carry out an instruction; to run a program on the computer.

FIELD
1. A division of memory containing 4096 decimal (numbered 0-7777 octal) storage cells (locations).

2. An element of a format specification, particularly of a record.

FILE
A contiguous block of characters or computer words, particularly where stored on a mass storage device and entered in the device's directory.

FILE NAME
A name of one to six alphanumeric characters chosen by the user to identify a file.

FILE NAME EXTENSION
Two alphanumeric characters chosen by the programmer or provided by OS/78 to describe the format of information in the file.

FILE-STRUCTURED DEVICE
A device on which files may be stored; also contains a file directory.

FIXED POINT
A format in which one or more computer words (two in the case of OS/8 FORTRAN integers), represent a number with a fixed binary point.

FLAG
A variable or register used to record the status of a program or device. In the latter case it is sometimes called a device flag.

FLOATING POINT
A format in which one or more computer words (three in the case of OS/8 FORTRAN Real and OS/8 BASIC numbers) represent a number with a variable binary point, particularly when there is an exponent part and a mantissa part.

FLOPPY DISK
See diskette.

FORMAT
1. A description of a set of valid sequences of language or data elements; syntax.

2. A FORTRAN statement which specifies the arrangement of characters to be used to represent a piece of data.

FORTRAN
A high-level language developed for the scientific community, it stands for FORmula TRANslation.

GARBAGE
Undefined or random data or instructions.

HARD COPY
Computer output in the form of printing on paper and generally in readable form such as listings or other documents.

HEAD
A hardware component that reads, records, or erases data on a file-structured device.

HIGH-LEVEL LANGUAGE
A language in which single statements typically result in more than one machine language instruction, for example, BASIC, FORTRAN.

INDIRECT ADDRESS
An address in a computer instruction which indicates a location in memory where the address of the referenced operand is to be found.

INHIBIT
To prevent, suppress or disallow.

INITIALIZATION CODE
Code which sets counters, switches, and addresses to zero or other starting values at the beginning of or at prescribed points in a computer routine.

INPUT BUFFER
A section of memory used for storage of input data.

INPUT
The process of transferring data to memory from a mass storage device or from other peripheral devices into the AC.

INSTRUCTION
One unit of machine language, usually corresponding to one line of assembly language, which tells the computer what elementary operations to do next.

I/O
Input-output. Refers to transfers of data between memory and peripheral devices.

ITERATION
Repetition of a group (loop) of instructions.

INTERACTIVE
Referring to a mode of using a computer system in which the computer and the user communicate via a computer terminal.

INTERRUPT
A hardware facility that executes an effective JMS to location 0 of field 0 upon any of a particular set of external (peripheral) conditions. The processor state is saved so that the interrupted program can be continued following any desired interrupt-level processing.

INTERRUPT DRIVEN
Pertaining to software that uses the interrupt facility of the computer to handle I/O and respond to user requests.

JOB
A unit of code which solves a problem; a program or sequence of programs.

JUMP
A departure from the consecutive sequence of executing instructions. Control is passed to some location in memory which is specified by the jump instruction.

K
Two to the tenth power (1024 in decimal notation). For example, a 4K memory has 4096 words.

KEYBOARD
On a typing device, the array of buttons which causes character codes for letters, numbers, and symbols to be generated when pressed.

LABEL
One or more characters used to identify a source language statement on line or label.

LATENCY
On rotating storage devices, the delay between the instant the device is notified that a transfer is coming and the instant the device is ready to perform the transfer.

LEAST SIGNIFICANT DIGIT
The rightmost digit.

LIBRARY
A collection of standard routines which can be incorporated into other programs.

LINE
A string of characters terminated with a line feed, vertical tab, or form feed character (and usually also a carriage return). The terminator belongs to the line that it terminates.

LINE NUMBER
1. In source languages such as BASIC and FORTRAN, a number which begins a line for purposes of identification. A numeric label.

2. In editors and listings, the number of the line beginning at the first line of the program and counting each line.

3. An automatically-generated indirect address for an off-page reference (PAL8).

LINK

1. A one-bit register that is complemented when overflow occurs in the accumulator.

2. An address pointer to the next element of a list or next block of a file.

3. A PAL8 assembler-created indirect address.

LINKAGE

Code that connects two separately coded routines and passes values and/or control between them, particularly when the routines occupy separate fields.

LIST

To output out a listing on the terminal or line printer. The listing of a source program is a sequential copy of statements or instructions in the program. Also, a table of data in a program.

LITERAL

A constant that defines itself by requesting the assembler to reserve storage for it, even though no storage location is explicitly specified in the source prgram.

LOAD

To move data into a hardware register or into memory.

LOADER

A program which takes information in binary format and copies it into memory. An absolute loader loads binary information that has been prepared for the absolute addresses of memory. A relocatable or linking loader loads binary information specified in relative addresses by assigning an absolute address to every relative address.

LOCATION

A numbered or named cell in memory where a word of data or an instruction or address may be stored.

LOOP

A sequence of instructions that is executed repeatedly until a terminating condition occurs. Also, used as a verb meaning to execute this sequence of instructions while waiting for the ending condition.

MACHINE LANGUAGE

The language, particular to each kind of computer, that those computers understand. It is a binary code which contains an operation code to tell the computer what to do and an address to tell the computer what data to perform the operation on.

MAP

A diagram representing memory locations and outlining which locations are used by which programs.

MASK

A combination of bits that is used to clear or accept selected portions of any word, character or register while retaining or ignoring other parts. Also, to clear these selected locations with a mask.

MASK STORAGE

Devices such as a diskette and disk pack that stores large amounts of data readily accessible to the central processing unit.

MATRIX
A rectangular array of elements. A two-dimensional table can be considered a matrix.

MEMORY
The main storage in a computer from which instructions must be fetched and executed. Consists of a sequence of consecutively-numbered cells storing one word each.

MEMORY IMAGE FILE
An executable binary code program file created by the system command SAVE from the current contents of memory.

MEMORY MAP
A diagram or table showing specific memory requirements of one or more programs.

MEMORY REFERENCE INSTRUCTION
A computer instruction that accesses the computer memory during its execution, as opposed to a register instruction which only accesses registers in the CPU and I/O instructions which are commands to peripheral devices.

MNEMONIC
A symbolic representation of an operation code or address (for example, X for unknown variable, JMP for jump instruction).

MODE
A state or method of system or program operation.

MODULE
A routine that handles a particular function.

MONITOR
The collection of routines which schedules resources, I/O, system programs, and user programs, and obeys keyboard commands.

MONITOR SYSTEM
Editors, assemblers, compilers, loaders, interpreters, data management programs and other utility programs all automated for the user by a monitor.

MOST SIGNIFICANT DIGIT
The leftmost digit.

NESTING
The inclusion of one program language construction inside another.

NON-DIRECTORY DEVICE
A device such as a terminal or a line printer that cannot store or retrieve files.

NO-OP
Contraction of No Operation, an instruction that specifically causes the computer to delay for one instruction time, and then to get the next instruction.

OBJECT CODE
The result after assembling or compiling source code.

OCTAL
The number system with a base, or radix, of eight.

ODT
Octal Debugging Technique, an interactive program for finding and correcting bugs in programs in which the user communicates in octal notation.

OFF-LINE
Pertaining to equipment, devices or events which are not under direct control of the computer.

ONE'S COMPLEMENT
A number formed by settng each bit in an input number to the other bit: 1's become 0's and 0's become 1's.

OP-CODE
The part of a machine language instruction that identifies the operation that the CPU will be required to perform.

OPERAND
The data to be used when an instruction is executed.

OPERATING SYSTEM
See MONITOR SYSTEM

OPERATOR
That symbol or code which indicates an action or operation to be performed (e.g., + or TAD).

ORIGIN
The absolute address of the beginning of a section of code.

OUTPUT
The process of transferring data from memory to a mass storage device or to a listing device such as a line printer.

OVERFLOW
A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the program or system is capable of handling.

PACK
To conserve storage requirements by combining data.

PAGE
A 128 (decimal) word section of memory, beginning at an address which is a multiple of 200 (octal). There are 40 (octal) pages in a field, numbered 0-37 (octal).

PAL
The name of the OS/78 system's assembly language.

PARITY BIT
A bit that indicates whether the total number of binary one digits in a piece of data is even or odd.

PARITY CHECK
A check that tests whether the sum of all the bits in an entity is odd or even.

PASS
One complete reading of a set of input data. An assembler usually requires two passes over a source program in order to translate it into binary code.

PATCH
To modify a routine in an expedient way, usually by modifying the memory image rather than reassembling it.

PERIPHERAL
In a data processing system, any device distinct from the CPU, which provides the computer system with outside communication.

POINTER
A location containing the address of another word in memory.

PRINTOUT
A loose term referring to almost anything printed by a computer peripheral device; any computer-generated hard copy.

PROGRAM
A unit of instruction and routines necessary to solve a problem.

PROGRAM COUNTER
A register in the CPU that holds the address of the next instruction to be executed.

PROGRAMMABLE
Can be controlled by instructions in a program.

PROMPTING CHARACTER
A character that is displayed on the console terminal and cues the user to perform some action.

PSEUDO-OP
Contraction of "Pseudo Operation", an assembly language directive which does not directly translate into machine code but gives directions to the assembler on how to assemble the code that follows.

RADIX
The base of a number system, the number of digit symbols required by a number system. The decimal number system is radix 10.

RANDOM ACCESS
Pertaining to a storage device where data or blocks of data can be read without regard to their physical order (for example, diskette).

READ
To transfer information from a peripheral device into memory or a register in the CPU.

RECORD
A collection of related items of data treated as a unit, such as a line of source code, or a person's name, address, and telephone number.

REDUNDANCY
In any data, that portion of the total characters or bits that can be eliminated without any loss of information.

REGISTER
A device usually made of semiconductor components that is capable of storing a specified amount of data, frequently one word.

RELATIVE ADDRESS
The number that specifies the difference between the actual address and the base address.

RELOCATE
To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in the new location.

RESTART
To resume execution of a program.

RETURN
To pass control back to a calling program at a point following the call when a subroutine has completed its execution.

1. The set of instructions at the end of a subroutine that permits control to return to the proper point in the main program.

2. The point in the main program to which control is returned.

3. The name of a key on the terminal.

RING-BUFFER
A storage area for data accessed on a first-in, first-out basis whose area is reused circularly.

ROUTINE
A set of instructions arranged in proper sequence to cause the computer to perform a desired task.

RUN
1. A single continuous execution of a program.

2. To perform that execution.

RUN TIME
The time during which a program is executed.

SCRATCH PAD MEMORY
Any memory or registers used for temporary storage of partial results.

SECTOR
The smallest unit of physical storage on a file-structured device.

SEGMENT
To divide information into segments or to store portions of information program on a mass storage device to be brought into memory as needed.

SERIAL ACCESS
Pertaining to tne sequential or consecutive transmission of data to or from memory or a peripheral device.

SERIAL TRANSMISSION
A method of information transfer in which the bits composing the character are sent sequentially on a single path.

SERVICE ROUTINE
A program used for general support of the user; I/O routines, diagnostics, and other utility routines.

SHIFT
A movement of bits to the left or right, usually performed in the accumulator.

SIMULATE
To represent the function of a device, system or program with another device, system or program.

SOFTWARE
The executable instructions used in a computer system.

SOURCE LANGUAGE
Any programming language used by the programmer to write a program before it is translated into machine code.

SOURCE PROGRAM
The computer program written in the source language.

SPOOLING
The technique by which output to slow devices is placed into temporary files on mass storage devices to await transmission. This allows more efficient use of the system since programs using low speed devices can run to completion quickly and make room for others.

STATEMENT
An expression or instruction in a source language.

STORAGE
A general term for any device capable of retaining information.

STORE
To enter data into a storage device, especially memory.

STRING
A sequence of characters.

SUBROUTINE
A section of code, usually performing one task, that may be called from various points of a main program.

SUBSCRIPT
A value used to specify a particular item in an array.

SYMBIONT
A program (or task) that can run simultaneously and cooperatively with the OS/78 system. This is a feature of DEC station 78 series systems (see Appendix H).

SYMBOLIC ADDRESS
Alphanumeric characters used to represent a storage location in the context of a particular program. It must be translated to an absolute address by the assembler.

SYMBOL TABLE
A memory storage area or a listing that contains all derined symbols and the binary value associated with each one. Mnemonic operators, labels, and user defined symbols are all placed in the symbol table. (Mnemonic operators stay in the table permanently.)

SYNCHRONOUS
Pertaining to circuits or events where all changes occur simultaneously or in definite timed intervals.

SYSTEM
A combination of software and hardware which performs specific processing operations.

SYSTEM DEVICE
A peripheral mass storage device on which the system software resides. Its handler (SYS) is resident in the last page of field 0.

SYSTEM HEAD
The system area reserved on a file-structured device for the OS/78 Keyboard Monitor. It resides on the system device and contains, in addition to the Monitor, programs such as ODT and Command Decoder.

SYSTEM SOFTWARE
DEC-supplied programs which come in the basic software packages. These include editors, assemblers, compilers, loaders, etc.

TABLE
A collection of data sorted for ease of reference, generally a two-dimensional array.

TEMPORARY STORAGE
Storage locations or registers reserved for intermediate results.

TERMINAL
A peripheral device in a system through which data can enter or leave the computer, especially in a display and keyboard.

TEXT
A message or program expressed in characters.

TRACK
The set of all sectors of a file-structured device that are accessible at each head position.

TRUNCATION
The reduction of precision by ignoring one or more of the least significant digits without rounding off.

TWO'S COMPLEMENT
A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1's to 0's and all 0's to 1's, then adding 1.

UTILITIES
Programs to perform general useful functions.

VARIABLE
A piece of data whose value changes during the execution, assembly, or compilation of a program.

WORD
A 12-bit unit of data which may be stored in one addressable location in memory or on a peripheral device.

Ø FORTRAN carriage control
  character, 7-1Ø1
1 FORTRAN carriage control
  character, 7-1Ø1
/2 QUEUE command option, H-2
/8 LOAD command option, 3-43
/9 LOAD command option, 3-43
/=n DIRECT command option,
  3-25

A,
  Editor append command, 4-6
  field descriptor, 7-95
  ODT command, 9-6
ABS BASIC function, 6-48
ABSLDR.SV, 3-11, 3-34, 3-44
Absolute binary files, 3-42
Absolute value function, 6-48
Addition,
  BASIC operator (+), 6-17
  FORTRAN operator (+), 7-39
  PAL8 operator (+), 5-18
Addressing indirect and page zero,
  5-29
AINT FORTRAN function, 7-1Ø6
ALOG FORTRAN function, 7-1Ø6
ALOG1Ø FORTRAN function, 7-1Ø6
Alphanumeric literals, 7-31
AMAXØ FORTRAN function, 7-1Ø6
AMAX1 FORTRAN function, 7-1Ø6,
  6-1Ø6
AMINØ FORTRAN function, 7-1Ø6
AMIN1 FORTRAN function, 7-1Ø6
AMOD FORTRAN function, 7-1Ø6
Ampersand (&),
  BASIC operator, 6-18, 6-33
  PAL8 operator, 5-18
AND,
  BASIC function, 6-58
  PAL8 AND symbol (&), 5-18
.AND. FORTRAN operator, 7-42
Angle brackets (<>),
  BASIC relations, 6-21
  in PAL8 comments, 5-23
  left (<),
    Editor command, 4-12
    monitor, 2-13
  PAL8 conditionals, 5-23
  right (>) Editor command, 4-12
Arctangent function, 6-45
Arguments system retention of,
  2-17
Arithmetic expressions,
    BASIC, 6-17
    FORTRAN, 7-38
Arithmetic statements,
    FORTRAN, 7-45
    BASIC strings, 6-19
Arrays,
  BASIC, 6-24
  FORTRAN, 7-33, 7-47
ARROW SET command option, 3-62
ASC function, 6-52
ASCII,
  6-bit, 5-37
  character set, A-1
  conversion,
    BASIC function, 5-52
    PAL8, 5-22
  entering PAL8 text strings, 5-37
  file format, 2-26
ASF function, 7-7Ø
ASIN FORTRAN function, 7-1Ø7
Assembling a PAL8 program, 5-5
  CREF command, 3-19
  PAL command, 3-53
Assembly language programs,
  Command Decoder use, D-1
  math subroutines, B-1
Assembler termination, 5-39
ASSIGN,
  command, 2-11, 3-2
  statement, 7-58
ASSIGNed GOTO, 7-58
Assigning logical device names,
  3-2
Asterisk (*),
  BASIC, 6-33
  double (**),
    BASIC operator, 6-17
    FORTRAN operator, 7-39
  FORTRAN operator, 7-39
  prompting symbol, 2-21
  PAL8 location counter, 5-14
  PAL8 special character, 5-21
  wild character, 2-18
At (@) symbol indirect commands,
  2-2Ø
ATAN FORTRAN function, 7-1Ø7
ATAN2 FORTRAN function, 7-1Ø7
ATN function, 6-45
Autoindex registers, 5-28

B,
  Editor buffer command, 4-7
  ODT command, 9-5
/B,
  BASIC option, 6-79
  COMPARE option, 3-6
  COMPILE command option, 6-79

/B (cont.)
DIRECT command option, 3-25
Editor option, 4-21
PAL8 option, 5-9
.BA file name extension, 2-12
BACKSPACE statement, 7-86
BASIC, 6-1,
arithmetic expressions, 6-17
auto line numbering, 6-8
calling an old program, 6-4
changing program text, 6-7
character set, 6-2
command, 3-3
command summary, 6-80
commands, 6-3
commercial arithmetic, 6-19
compilation, 3-34
constants, 6-13
control statements, 6-36
creating a new program, 6-4
deleting text, 6-7
deleting program lines, 6-12
direct record I/O, 6-69
elements of the language, 6-13
ending a session, 6-11
erasing a program, 6-11
error messages, 6-86
execution, 3-34, 3-10
expression operators, 6-17
expressions, 6-17
file control, 6-62
file statements, 6-62
format control, 6-30
function summary, 6-83
functions, 6-43
interaction with BATCH, 6-80
interrupting program execution, 6-12
issuing monitor commands, 6-55
listing a program, 6-5
listing control, 6-12
logical functions, 6-58
merging text, 6-9
numeric,
constants, 6-13
field list use, 6-36
functions, 6-44
variables, 6-15
operating procedures, 6-3
overlay files, 6-80
overview, 6-1
printing format strings, 6-35
program termination, 6-41
program segmentation, 6-74
reading files, 6-65
relational expressions, 6-18
renaming a program, 6-10
resequencing a program, 6-11
running a program, 6-5
saving programs, 6-79

BASIC (cont.)
special print functions, 6-34
statements, 6-23
formatting, 6-22
summary, 6-81
storing a program, 6-10
string,
concatenation, 6-18
constants, 6-14
functions, 6-49
numbers, 6-20
relations, 6-22
variables, 6-15
subroutines, 6-42
subscripted variables, 6-16
user-defined functions, 6-78
using record I/O files, 6-72
variables, 6-14
writing files, 6-65
writing programs, 6-1
BASIC.SV, 3-3
BAT device handler, 3-67
Batch,
command processing, 8-3
commands, 8-2
error messages, 8-6
input file structure, 8-4
interaction with BASIC, 6-80
introduction, 8-1
SUBMIT command, 3-72
operating restrictions, 8-8
options, 8-2
BATCH.SV, 3-74
BCOMP.SV, 3-11,, 3-34
.BI file name extension, 2-12
BITMAP.SV, 3-48
Blank lines in FORTRAN programs, 7-25
BLOAD.SV, 3-34
BLOCK DATA statement, 7-54
.BN file name extension, 2-12
Boolean operators, PAL8, 5-18
BOOT,
command, 3-4
button, 2-6
BOOT.SV, 3-4
Bootstrapping OS/78, 2-6
Brackets ([]), 2-16
PAL8 special characters, 5-22
Breakpoints under ODT, 9-2
BTCHAx BATCH files, 3-74
Buffer size of Editor, 4-4
BYE command, 6-11

C,
Editor change command, 4-9
FORTRAN comment indicator, 7-23
ODT command, 9-7

/C,
  COMPARE option, 3-6
  COPY command option, 3-14
  DELETE command option, 3-24
  DIRECT command option, 3-25
  FORTRAN Loader option, 7-10
  FRTS option, 7-16
  LIST command option, 3-41
  LOAD command option, 3-44
  PAL8 option, 5-9
  RENAME command option, 3-56
  TYPE command option, 3-76
C:n/ QUEUE command option, H-2
CALL statement, 7-74
CANCEL command, 3-5, H-1
CAP$ function, 6-54
Caret - see circumflex
CCL.SV, 3-9, 3-3, 3-4
CDF instruction, 5-30, 5-43
CGET subroutine, 7-107
CHAIN,
  statement, 6-74
  USR function C-10
Chaining PAL8 programs, C-10
Character conversion function, 6-52
Character set, A-1
  BASIC, 6-2, 6-2
  FORTRAN, 7-20
  PAL8, 5-10
Characters,
  output by codes (BASIC), 6-60
  PAL8 formatting, 5-12
  wild, 2-18
CHKEOF FORTRAN subroutine, 7-108
CHR$ function, 6-53
CIF instruction, 5-30, 5-43
Circumflex (^) ODT command, 9-4
CLOCK FORTRAN subroutine, 7-108
CLOSE function (USR), C-8
CLOSE# statement, 6-64, 6-71
.CM file name extension, 2-12
COL,
  function, 6-60
  SET command option, 3-62
Comma (,),
  BASIC, 6-34
  FORTRAN use, 7-101
  PAL8 special character, 5-21
Command Decoder,
  calling from USR, C-1
  conventions, D-1
  calling, D-3
  error messages, D-2
  example of use, D-6
  option table, D-5
  special mode call, D-8
  tables, D-4
Commands, OS/78, 2-11, 3-1
  argument retention 2-17
Commands, OS/78 (cont.)
  ASSIGN, 3-2
  BASIC, 3-3
  BOOT, 3-4
  CANCEL, 3-5
  COMPARE, 3-6
  COMPILE, 3-10, 7-4
  COPY, 3-12
  CREATE, 3-18
  CREF, 3-19
  DATE, 3-21
  DEASSIGN, 3-22
  DELETE, 3-23
  DIRECT, 3-25
  DUPLICATE, 3-28
  EDIT, 3-32
  EXECUTE, 3-34
  format, 2-13
  FORMAT, 3-35
  full word switches, H-1
  GET, 3-37
  HELP, 2-22, 3-38
  incorrect, 2-14
  LIST, 3-40
  LOAD, 3-42
  MAP, 3-46
  MEMORY, 3-50
  ODT, 3-52, 9-3
  PAL, 3-53
  QUEUE, 3-54
  R, 3-55
  RENAME, 3-56
  REQUEST, 3-57
  SAVE, 3-59
  SET, 3-62
  SQUISH, 3-70
  START, 3-71
  SUBMIT, 3-72, 8-2
  summary, 2-26
  TERMINATE, 3-75
  TYPE, 3-76
  UA/UB/UC, 3-78
  ZERO, 3-79
Comments,
  BASIC, 6-26
  FORTRAN, 7-23
  PAL8, 5-12
COMMON statement, 7-50
  interaction with EQUIVALENCE, 7-54
COMPARE command, 3-6
  error messages, 3-9
  option summary, 3-6
COMPILE command, 3-10
  option summary, 6-79, 7-5
Computed GOTO, 7-58
Concatenation, 6-18
Conditional,
  assembly operators, 5-36
  delimiters, 5-23

Conditional (cont.)
transfer,
BASIC, 6-37
FORTRAN, 7-60
Constants,
BASIC, 6-13
FORTRAN, 7-27
exponential, 7-29
integer, 7-28
Hollerith, 7-31
logical, 7-30
octal, 7-30
CONTINUE statement, 7-66
Control character,
representation, 2-7
uparrow (^) echo control, 3-63
Control statements,
BASIC, 6-36
FORTRAN, 7-56
COPY command, 3-12
option summary, 3-16
error messages, 3-17
Core Control Block, 3-55, 3-59, 3-71, 9-2
Correcting errors, 2-15
COS function, 6-45
COS FORTRAN subroutine, 7-108
COSH FORTRAN subroutine, 7-108
CPUT FORTRAN subroutine, 7-109
CREATE command, 3-18
Creating a PAL8 program, 5-4
CREF command, 3-19
error messages, 3-20
option summary, 3-19
CREFLS.TM file, 3-19
CREF.SV, 3-19, 3-19
Cross-reference listing, 3-19, 5-7
CTRL/C, 2-7, 2-15
BASIC use, 6-12
BATCH use, 8-5
COPY use, 3-15
EDIT use, 4-3
CTRL/G, 5-24
CTRL/L, 4-3
CTRL/O, 2-7
BASIC, 6-6, 6-12
control character, 4-3
ODT use, 9-11
CTRL/Q, 2-7
BASIC use, 6-12
CTRL/S, 2-7
BASIC use, 6-12
CTRL/U, 2-7
BASIC use, 6-12
FORTRAN use, 7-16
CTRL/Z,
FORTRAN use, 7-16
with device handlers, F-2, F-4
CUR$ function, 6-59
Current location counter, 5-21
changing with PAGE, 5-33
PAL8 symbol, 5-14
PAL8, 5-21
Current page literals, 5-22
Cursor function, 6-59

D,
Editor delete command, 4-9
ODT command, 9-7
/D,
DUPLICATE command option, 3-30
Editor option, 4-21
Dash options (general-purpose), 2-16
DAT$ function, 6-57
Data field register, 5-30
DATA statement, 6-28, 7-54
DATE,
command, 3-21
FORTRAN subroutine, 7-109
Date,
file transfer by, 3-14
function, 6-57
DEASSIGN command, 3-22
Debugging,
function (BASIC), 6-57
in octal, 3-52, 9-1
DECIMAL pseudo-operator, 5-36
Decimal to octal conversion, 6-55
Decimal point (.), 7-28
DECODE function (USR), C-10
DECstation System Hardware, 1-1
configuration summary, J-1
DEF statement, 6-56
Default,
file name extensions, E-1
file specifications, 2-21
DEFINE FILE statement, 7-86
DEFINE# statement, 6-70
DELETE,
BASIC command, 6-7
key, 2-7, 6-12, 7-16
OS/78 command, 3-23
error messages, 3-24
option summary, 3-24
Demonstration programs, 2-10
Device Control Word Table, C-15
Device handlers, 3-67
calling, F-1
description and use, F-1
FETCH function (USR), C-4
file-structured devices, 2-23, F-4
function control word, F-2
line printer, F-3
loading from PAL8 programs, C-4

Device handlers (cont.)
multiple input file restrictions, F-5
obtaining characteristics via USR, C-13
removal from memory, C-14
restrictions of use, F-3
terminals, F-5
Device names,
permanent, 2-10
reassigning, 2-10
DEVICE pseudo-operator, 5-37
Devices - see terminals also
file-structured, 2-23, F-4
FORTRAN I/O, 7-7
storage capacity, 2-23
using RX01/RX02 and RL01K, 2-24
using VXA0, 2-25
.DI file name extension, 2-12
DIM FORTRAN function, 7-109
Diagnostic messages,
FORTRAN, 7-5
PAL8, 5-39
DIMENSION statement,
BASIC, 6-24
FORTRAN, 7-47
Direct,
assignment statements, 5-16
record I/O, 6-69
DIRECT command, 3-25
option summary, 3-25
error messages, 3-27
DIRECT.SV, 3-27
Directory (file), 2-25
Disk pack - see RL01K,
Diskette - see RX01/RX02 also
bootstrap procedure, 2-5
making backup copies, 2-8
Dismissing USR, C-13
Division,
BASIC operator (/), 6-17
FORTRAN operator (/), 7-39
PAL8 operator (%), 5-18
DO statement, 7-62, 7-64
Document,
conventions, Preface
reference, Preface
dollar sign ($),
BASIC, 6-34
Editor symbol, 4-10
ESCape key echo, 3-64
FORTRAN field descriptor, 7-98
PAL8 special symbol, 5-24
Double,
asterisk (**), 6-17
quote ("), 5-22
Double-density diskette
formatting, 3-30
DSK device handler, F-4
Dummy arguments in subprograms, 7-69
DUPLICATE command, 3-28
error messages, 3-31
option summary, 3-30

E,
Editor command, 4-7
FORTRAN field descriptor, 7-92
ODT command, 9-7
/E,
CREF command option, 3-19
DIRECT command option, 3-25
FRTS option, 7-16
PAL8 option, 5-9
SUBMIT command option, 3-72, 8-2
ECHO option (SET command), 3-62
EDIT command, 3-32
BASIC, 6-7
EDIT.SV, 3-18, 3-33
Editor - (see CREATE and EDIT commands also), 4-1
alteration commands, 4-9
append (A) command, 4-6
buffer (B) command, 4-7
buffer size, 4-4
calling procedure, 4-1
change (C) command, 4-9
changing existing file, 4-2
command mode, 4-2, 4-5
command summary, 4-23
command usage, 4-13 thru 4-18
control commands, 4-3
creating a new file, 4-1
delete (D) command, 4-9
Editor,exit (E) command, 4-7
error messages, 4-21,4-22
example session, 4-18
get (G) command, 4-6
insert (I) command, 3-6
interbuffer search (J) command, 4-10
kill (K) buffer command, 4-8
list (L) command, 4-6
list buffer (V) command, 4-9
listing commands, 4-6
next (N) command, 4-8
operating modes, 4-2
option summary, 4-21
output commands, 4-7
output current buffer (E) command, 4-7
output (P) command, 4-7
page (P) command, 4-4
quit (Q) command, 4-8
read (R) command, 4-5, 4-6
search character (S) command, 4-10

Editor (cont.)
search next (F) command, 4-10
special characters, 4-11
text mode, 4-2
text searches, 4-13
yank (Y) command, 4-10
EJECT pseudo-operator, 5-34
$END command (BATCH), 8-3
END statement, 6-41, 7-68
END# statement, 6-68
End-of-file,
BASIC, 6-68
FORTRAN ENDFILE statement, 7-87
PAL8, 5-37
ENTER function (USR), C-7
Exponentiation, 6-17
Equals sign (=), 2-16
BASIC, 6-21, 6-23
Editor command, 4-12
FORTRAN use, 7-45
PAL8 direct assignment, 5-16
PAL8 special character, 5-21
EQUIVALENCE statement, 7-52
interaction with COMMON, 7-54
.EQV. FORTRAN operator, 7-42
Error messages,
BASIC, 6-86
BATCH, 8-6
Command Decoder, D-2
COMPARE command, 3-9
COPY command, 3-17
CREF command, 3-20
DELETE command, 3-24
DIRECT command, 3-27
DUPLICATE command, 3-31
Editor, 4-21,4-22
FORMAT command, 3-36
FORTRAN Loader, 7-11
FRTS, 7-17
HELP command, 3-39
LIST command, 3-41
MAP command, 3-49
ODT, 9-8
OS/78 summary, G-1
PAL8, 5-39
SET command, 3-68
system halts, G-1
TYPE command, 3-77
ERROR function (USR), C-11
Errors, correcting keyboard, 2-15, 6-12
ESC SET command option, 3-62
ESCape key, 2-7
BATCH operation, 3-73
dollar sign ($) echo control, 3-64
Editor intrabuffer search command, 4-10
FORTRAN use, 7-14
terminator, 2-13
ESCape sequences in BASIC, 6-76
Exclamation point (!),
BASIC, 6-26
PAL8 operator, 5-18
EXECUTE command, 3-34
Executing PAL8 programs, 5-7
EXP function, 6-46
EXP FORTRAN function, 7-110
Exponential,
function, 6-46
operator (**), 7-39
real constant, 7-29
Expressions,
BASIC, 6-17
FORTRAN, 7-38
PAL8, 5-18
EXPUNGE pseudo-operator, 5-35
Extensions, file name 2-11, E-1
EXTERNAL statement, 7-48

F,
Editor search next command, 4-10
field descriptor, 7-91
/F,
COPY command option, 3-14
DIRECT command option, 3-25
PAL8 option, 5-9
F4.SV, 3-11, 3-34
/ff LOAD command option, 3-42
/ffnnnn, LOAD command option, 3-42
.FALSE. logical constant, 7-30
FETCH function (USR), C-4
FIELD pseudo-operator, 5-29
Fields,
BASIC numeric, 6-33
memory data and instruction, 5-30
FILE statement, 6-63
FILE# statement, 6-70
FILENAME pseudo-operator, 5-37
Files, 2-25
Files,
absolute binary, 3-42
ASCII format, 2-26
BASIC, 6-62
batch input, 3-72
changing ASCII, 4-2
closing, 6-71, C-8
creating ASCII, 4-1
deleting tentative, C-15
directory, 2-25
listing, 3-25
eliminating empty, 3-70
fixed length, 6-70
input table, D-4
memory-image,
creation, 3-59
loading and executing, 3-55, 3-71

Files (cont.)
  names, 2-10
    extensions, 2-12, E-1
  opening for record I/O, 6-70
  output spooling under BATCH, 3-73
  output table, D-4
  permanent, C-6
  postdeletion, 3-13
  predeletion, 3-13
  protection during transfer, 3-14
  resetting BASIC, 6-67
  restrictions on multiple input, F-5
  specifications,
    using defaults, 2-21
    wildcard input, 2-19
    wildcard output, 2-19
  tentative, C-7
  testing for end, 6-68
  testing for open, 6-67
  transfer by date, 3-14
  transfer command COPY, 3-12
  types, 2-25
  USR search, C-6
  variable length string, 6-69
FILEV# statement, 6-69
FIX$ function, 6-55
FLOAT FORTRAN function, 7-110
FNa function, 6-56
FOR statement, 6-39
FORLIB.RL, 3-44, 7-19
FORM FEED, 5-12, 5-24
FORMAT command, 3-35
  error messages, 3-36
  option summary, 3-35
Format,
  ASCII files, 2-26
  BASIC statements, 6-22
  characters,
    PAL8, 5-12
  control,
    BASIC, 6-30
  statements,
    FORTRAN, 7-88
  OS/78 commands, 2-13
Formatted I/O (FORTRAN), 7-77
Formatting,
  double density diskettes, 3-30
  RL01K disk packs, 3-35
  single-density diskettes, 3-30
FORTRAN,
  arithmetic,
    assignment statements, 7-45
    conditionals, 7-60
  arrays, 7-33, 7-53
  assignment statements, 7-45
  auxiliary I/O statements, 7-85
  BLOCK DATA subprograms, 7-54
FORTRAN (cont.)
  carriage control, 7-100
  compilation, 3-34, 3-10
  constants, 7-27
  continuation lines, 7-24
  control statements, 7-56
  DATA statements, 7-54
  data types, 7-27, 7-27, 7-33
  direct access I/O, 7-80
  error message summary, 7-5
  execution, 3-34
  exponential constants, 7-29
  expressions, 7-38
    operators, 7-39
  field descriptors, 7-89
  FORMAT statements, 7-88
  format,
    control and I/O lists, 7-103
    specification separators, 7-101
  FRTS, 7-15
    error messages, 7-17
  grouping and group repeat specifications, 7-100
  Hollerith constants, 7-31
  I/O,
    devices, 7-7
    formatting, 7-80
    lists, 7-77
    statements, 7-76
  identification field, 7-25
  integer constants, 7-28
  introduction, 7-1
  label field, 7-23
  language, 7-19
    summary, 7-112
  library, 7-19
    functions, 7-76, 7-105
    subroutines, 7-105
  line format, 7-25
  literals, 7-97
  Loader, 7-7
    error messages, 7-11
    options, 7-10
  logical,
    assignment, 7-46
    conditionals, 7-62
    constants, 7-30
    expressions, 7-42
    operators, 7-42
    unit numbers, 7-77
  octal constants, 7-30
  operating procedures, 7-4
  program loading, 3-42
  real constants, 7-28
  relational expressions, 7-41
  relocatable binary files, 3-44
  runtime system (FRTS), 7-1, 7-12
    I/O assignment, 7-12
  scale factors, 7-99

FORTRAN (cont.)
  sequential I/O, 7-80
  special characters, 7-21
  specification statements, 7-46
  statements, 7-21
  storage declaration, 7-50
  subprograms, 7-69, 7-72
    declaration, 7-48
    user-written, 7-70
  subroutines, 7-73
  subscripts, 7-36
  symbols, 7-26
  variables, 7-31
FOTP.SV, 3-15, 3-23, 3-40, 3-56
FRTS, 7-12
FRTS.SV, 3-34
-FT dash option, 2-17
.FT file name extension, 2-12
FUNCTION statement, 7-72
Functions,
  BASIC, 6-43
  FORTRAN, 7-105
  user-defined, 6-78

G,
  Editor get command, 4-6
  field descriptor, 7-93
  ODT command, 9-5
/G,
  BASIC option, 6-79
  COMPILE command option, 6-79, 7-5
  FORTRAN Loader option, 7-10
  LOAD command option, 3-43, 3-45
  PAL8 option, 5-9
General-purpose dash options, 2-16
GET command, 3-37
GET# statement, 6-71
Global subroutine calls, 5-22
GOSUB statement, 6-42
GOTO statement, 6-36, 7-57

H field descriptor, 7-96
/H,
  PAL8 option, 5-9, 5-9
  SUBMIT command option, 8-2
Handler - see device handler,
HANDLER option (SET command), 3-62
Hardware configuration summary, J-1
HEIGHT option (SET command), 3-62
HELP command, 2-22, 3-38
  error messages, 3-39
HELP.SV, 3-38
.HN file name extension, 2-12
Hollerith constants, 7-31

I,
  Editor insert command, 4-6
  field descriptor, 7-90
  PAL8 indirect addressing pseudo-operator, 5-29
/I LOAD command option, 3-43
I/O statements,
  FORTRAN, 7-76
I/O,
  BASIC, 6-26
  direct record (BASIC), 6-69
IDIM FORTRAN function, 7-110
IF GOTO statements, 6-37
IF statement, 7-60
IFDEF pseudo-operator, 5-36
IFIX FORTRAN function, 7-110
IFNDEF pseudo-operator, 5-36
IFNZRO pseudo-operator, 5-36
IFZERO pseudo-operator, 5-36
Indirect,
  addressing, 5-29
  commands (@ symbol), 2-20
INIT option (SET command), 3-62
Input file table, D-4
INPUT# statemen6-65, 6-65
Input/output options, 2-15
INQUIRE function (USR), C-13
Instruction field,
  buffer, 5-30
  register, 5-30
Instructions,
  memory reference, 5-24
  PAL8, 5-11
INT function,
  BASIC, 6-47
  FORTRAN, 7-110
Integer function, 6-47
INTEGER, 7-27
IOR function, 6-58
IOT microinstructions, 5-28
ISIGN FORTRAN function, 7-110
Iteration,
  BASIC, 6-39
  FORTRAN, 7-62

J Editor command, 4-10
/J PAL8 option, 5-9
$JOB BATCH command, 3-72, 8-3
Job status word, 3-60

K Editor command, 4-8
/K PAL8 option, 5-9
KEY$ function, 6-59, 6-77
Keyboard errors, 2-15
KT8A Memory Management Control, 5-31

L,
  Editor list command, 4-6
  field descriptor, 7-94
  ODT command, 9-6
-L dash option, 2-17
/L,
  COMPILE command option, 7-5
  PAL9 option, 5-9
  QUEUE command option, H-2
  SET HANDLER command option, 3-67
LA34/38, 2-6
LA36, 2-6
LA120, 2-6
Labels, 5-11
.LD file name extension, 2-12
LEN function, 6-50
LET statement, 6-23
LINE FEED, 2-7
  Editor command, 4-12
  PAL8 special character, 5-24
  ODT command, 9-3
Link generation and storage, 5-38
LIST command, 3-40
  BASIC, 6-5
LISTNH command, 6-6
Literals,
  FORTRAN, 7-97
  PAL8 current page, 5-22
  PAL8 storage, 5-39
LOAD command, 3-42
LOAD.SV, 3-11, 3-34
Loading PAL8 programs, 5-6
Loading,
  diskettes, 2-1
  RL01K disk packs, 2-3
Location counter - see current
  location counter
Locking USR in memory, C-12
LOG function function, 6-47
LOGICAL statement, 7-27
Logical,
  devices,
    assigning names, 3-2
    deassigning names, 3-22
    permanent names, 2-10
  expressions, 7-42
  unit numbers, 7-77
LOOKUP,
  function (USR), C-6
Looping,
  BASIC, 6-39
  FORTRAN statements, 7-64
  nested (BASIC), 6-40
Lower-case characters,
  conversion, 6-54
  line printer use, 3-64
  PAL8 restrictions, 5-11
LPT device handler, 3-67, F-3
LQP device handler, 3-67, F-3
-LS dash option, 2-17
.LS file name extension, 2-12

M command (ODT), 9-7, 9-8
/M,
  CREF command option, 3-19
  DIRECT command option, 3-25
  DUPLICATE command option, 3-30
  SUBMIT command option, 3-72
Machine instruction set, 5-24
MAP command, 3-46
  option summary, 3-48
  error messages, 3-49
Mathematical subroutines, B-1
MAX0 function, 7-111
MAX1 function, 7-111
MEMORY command,, 3-50
Memory,
  fields, 3-50
  map, 3-46, 5-8
  page references, 5-38
  reference instructions, 5-24
  reserving with ZBLOCK, 5-33
Memory-image files,
  creation, 3-59
  execution, 3-71
  loading, 3-37,
  loading and executing, 3-55
Memory Management Control, 5-31
Microinstructions, 5-25
  IOT, 5-28
  operate, 5-26
MIN0 function, 7-111
MIN1 function, 7-111
MIN12 function, 7-111
Minus (-),
  BASIC, 6-17, 6-34
  FORTRAN, 7-28, 7-39
  PAL8, 5-18
MOD function,, 7-111
Monitor,
  issuing commands from BASIC,
    6-55
  period (.) prompting symbol,
    2-6
-MP dash option, 2-17
.MP file name extension, 2-12
$MSG command (BATCH), 8-3
Multiplication,
  FORTRAN, 7-39
  PAL8, 5-18

N Editor next command, 4-8
/N,
  COMPILE command option, 7-5
  COPY command option, 3-13
  DELETE command option, 3-24
  DUPLICATE command option, 3-30
  PAL8 optioin, 5-9
  QUEUE command option, H-2

/n,
  FORMAT command option, 3-35
  MAP command option, 3-48
NAME command, 6-10
Natural logarithm, 6-47
-NB dash option, 2-17
Nested loops, 6-40
NEW command, 6-4
NEXT statement, 6-39
nnnn+ and nnnn- ODT commands, 9-4
Non-system device, 2-23
.NOT. FORTRAN operator, 7-42
Number sign (#), 6-33
Numbers,
  BASIC string, 6-20
  PAL8, 5-13
Numeric,
  constants, 6-13
  fields, 6-33, 6-36
  functions, 6-44
  variables, 6-15
Numeric-to-string conversion, 6-54

/O,
  DELETE command option, 3-24
  DIRECT command option, 3-25
  LIST command option, 3-41
  PAL8 option, 5-9, 5-30
  RENAME command option, 3-56
  TYPE command option, 3-76
OCS$ function, 6-55
OCT function, 6-54
OCTAL,
  pseudo-operator, 5-36
  statemnent, 7-27
Octal constants, 7-30
  3-52, 9-1
Octal Debugging Technique (ODT),
Octal-to-decimal conversion, 6-54
ODT command, 3-52
  command summary, 9-9
  commands, 9-3
  error messages, 9-8
  illegal characters, 9-5
  introduction, 9-1
  programming notes, 9-8
  setting breakpoints, 9-6
  setting search limits, 9-7
  special characters, 9-3
  word searches, 9-8
OLD command, 6-4
ON GOTO statement, 6-38
ON GOSUB statement, 6-43
OPEN# statement, 6-67
Operands,
  as PAL8 symbols, 5-18
  PAL8, 5-12
Operate microinstructions, 5-26
Operators,
  PAL8 arithmetic and logical,
    5-18
  PAL8 conditional assembly, 5-36
OR,
  BASIC function, 6-58
  PAL8, 5-18
.OR. FORTRAN operator, 7-42
OS/78,
  bootstrapping, 2-6
  command argument retention, 2-17
  Command Decoder, D-1
  commands, 2-11
    summary, 2-26
  devices,
    handlers, 3-6, F-3
    permanent names,
  error message summary, G-1
  file names and extensions, 2-10
  file-structured devices, 2-23
  full word command options, I-1
  hardware configurations, J-1
  introduction to, 1-1
  logical device names, 2-10
  making software backup, 2-8
  monitor commands from BASIC,
    6-55
  startup procedure, 2-5
  system demonstration, 2-10
  User Service Routine, C-1
Output file table, D-4
Overlay files (BASIC), 6-80

P,
  Editor,
    output command, 4-7
    page command, 4-4
  FORTRAN scale factor symbol,
    7-99
/P,
  CREF command option, 3-19
  DUPLICATE command option, 3-30
  FORMAT command option, 3-35
  LOAD command option, 3-44
-PA dash option, 2-17
.PA file name extension, 2-12
Page zero addressing, 5-29
PAGE,
  pseudo-operator, 5-33
  SET command option, 3-62
PAL command, 3-53
PAL8, 3-34, 3-53
  assembly, 3-10, 5-5
  chaining programs, C-10
  character set, 5-10
    restrictions, 5-11
  command string examples, 5-2
  comments, 5-12

PAL8 (cont.)
conditional,
assembly operators, 5-36
expressions, 5-23
cross-reference listing, 5-7
current location counter, 5-14
current page literals, 5-22
diagnostic messages, 5-39
direct assignment statements, 5-16
error messages, 5-10
expressions, 5-18
file extensions, 5-1
FORM FEED character, 5-12
formatting characters, 5-12
instructions, 5-11
introduction to, 5-1
label, 5-11
link generation and storage, 5-38
listing control, 5-34
literals, 5-39
loading and saving a program, 5-6
math subroutines, B-1
memory map, 5-8
numbers, 5-13
off-page references, 5-38
operands, 5-12
option summary, 5-9
page zero addressing pseudo-operator, 5-29
permanent symbols, 5-14
program,
execution, 3-37
loading, 3-42
pseudo-operators, 5-29
radix control, 5-36
special characters, 5-21
statements, 5-11, 5-13
symbol table, 5-16
alteration, 5-35
symbolic instructions, 5-17
symbolic operands, 5-18
symbols, 5-13
TAB character, 5-12
text strings, 5-37
user-defined symbols, 5-14
using Command Decoder, D-1
PAL8.SV, 3-19, 3-34, 3-53
Parentheses (()), 2-16
FORTRAN, 7-44
PAL8, 5-22
Patching a memory-image (SAVE) file, 3-43
PAUSE,
pseudo-operator, 5-37
SET command option, 3-62
statement, 7-67
PDP-8 instruction set, 5-24
Percent (%), 5-18
Period (.),
BASIC, 6-34
Editor current line symbol, 4-11
monitor prompting symbol, 2-6
PAL8 special character, 5-21
Permanent file lookup, C-6
Permanent symbols, 5-14
PIP.SV, 3-70
Plus sign (+),
BASIC operator, 6-17
FORTRAN, 7-28
carriage control character, 7-101
operator, 7-39
PAL8 operator, 5-18
PMT$ function, 6-60
PNT function, 6-31, 6-60, 6-76
POS function, 6-51
Postdeletion of files, 3-13
Predeletion of files, 3-13
Print head position function, 6-60
PRINT statement, 6-29
PRINT# statement, 6-65
PRINT USING statement, 6-32
Program chaining, C-10
Prompt function, 6-60
Pseudo-operators,
DECIMAL, 5-36
DEVICE, 5-37
EXPUNGE, 5-35
EJECT, 5-34
FIELD, 5-29
FILENAME, 5-37
IFDEF, 5-36
IFNDEF, 5-36
IFNZRO, 5-36
IFZERO, 5-36
OCTAL, 5-36
PAGE, 5-33
PAL8, 5-29
PAUSE, 5-37
RELOC, 5-34
TEXT, 5-36
XLIST, 5-34
ZBLOCK, 5-33
PUT# statement, 6-71

Q,
Editor quit command, 4-8
/Q,
COMPILE command option, 7-5
COPY command option, 3-15
DELETE command option, 3-24
LIST command option, 3-41
SUBMIT command option, 3-72, 8-2
TYPE command option, 3-76

Question mark (?) wild character, 2-18
QUEUE command, 3-54, H-2
Quote ("),
  double - PAL8 special character, 5-22

R,
  Editor read command, 4-5, 4-6
  monitor command, 3-55
/R,
  DIRECT command option, 3-25
  DUPLICATE command option, 3-30
  MAP command option, 3-48
.RA file name extension, 2-12
Radix control, 5-36
RALF.SV, 3-11
Random number function, 6-48
READ statement, 6-28, 7-80
READONLY option (SET command), 3-62
Real constants, 7-28
REAL statement, 7-27
Reassigning device names, 2-11
Record,
  field definition, 6-70
  size statement, 6-69
Reference Documents, Preface
Relational,
  expressions,
    BASIC, 6-18
    FORTRAN, 7-41
  operators,
    BASIC strings, 6-22
RELOC pseudo-operator, 5-34
Relocatable binary files, 3-44
REM statement, 6-26
RENAME command, 3-56
RENAME command option summary, 3-56
REQUEST command, 3-57, H-1, H-2
RESEQ program, 6-11
RESET function (USR), C-14
Resetting BASIC files, 6-67
RESTORE statement, 6-28, 6-67
RETURN key, 2-7
  BASIC, 6-12
  ODT command, 9-3
  PAL8 special character, 5-24
RETURN statement, 6-42, 7-75
REWIND statement, 7-87
.RL file name extension, 2-12
RL01K disk pack,
  bootstrap procedure, 2-5
  formatting, 3-35
  making backup copies, 2-8
  storage capacity, 2-23
  use, 2-24
RLFMT.SV, 3-36
RLxx device handler, 3-67, F-4
RND function, 6-48
Rounding in BASIC, 6-21
RUN command, 6-5
RX01/RX02,
  duplicating diskettes, 3-28
  storage capacity, 2-23
  use, 2-24
RXAx device handler, 3-68, F-4
RXCOPY.SV, 3-30

S Editor search character command, 4-10
/S,
  BASIC option, 6-79
  COMPARE option, 3-6
  COMPILE command option, 6-79
  DUPLICATE command option, 3-30
  FORTRAN Loader option, 7-10
  LOAD command option, 3-42, 3-44
  MAP command option, 3-48
  PAL8 option, 5-9
SAVE command, 3-59
  BASIC, 6-10
Saving,
  PAL8 programs, 5-6
  BASIC programs, 6-79
SCOPE option (SET command), 3-62
SCRATCH command, 6-11
SEG$ function, 6-52
Semicolon (;) ODT command, 9-4
Semiconditional transfer, 6-38
SEQUENCE command, 6-8
SET command, 3-62
  error messages, 3-68
  options, 3-62
Setting terminal characteristics, 2-5
SGN function 6-48,
SIGN FORTRAN function, 7-111
SIN,
  BASIC, 6-45
  FORTRAN, 7-111
Single-density diskette formatting, 3-30
SINH FORTRAN function, 7-112
Six-bit ASCII, 5-37
Slash (/), 2-16
  BASIC operator, 6-17
  BATCH command, 8-3
  Editor symbol, 4-12
  FORTRAN operator, 7-39
  FORTRAN use, 7-101
  ODT command, 9-3
  PAL8 comments, 5-12
SLUx device handler, 3-68, F-5
Source file comparison, 3-6

Space character,
  FORTRAN carriage control, 7-101
  PAL8 operator, 5-18
  PAL8 special character, 5-24
Spooler (symbiont) commands, H-2
Spooling, 3-73, 8-1, H-1
SPOOLR.SV, 3-54, H-1
SQR function, 6-46
SQRT FORTRAN function, 7-112
Square brackets ([]), 2-16,
  Preface
Square root function, 6-46
SQUISH command, 3-70
SRCCOM.SV, 3-9
START button, 2-6
START command, 3-71
Statement,
  format,
    BASIC, 6-22
    FORTRAN, 7-24
  terminators,
    OS/78 monitor, 2-7
    PAL8, 5-13
Statements,
  FORTRAN, 7-21
  PAL8, 5-11, 5-16
STEP statement, 6-39
STOP statement,
  BASIC, 6-41
  FORTRAN, 7-68
Storage capacity of devices, 2-23
STR$ function, 6-54
String,
  arithmetic, 6-19
  concatenation, 6-18
  constants, 6-14
  integer function, 6-55
  length function, 6-50
  numbers, 6-20
  to numeric conversion, 6-53
  relations, 6-22
  variables, 6-15
SUBMIT command, 3-72,
  option summary, 8-2
Subprogram declaration, 7-48
SUBROUTINE statement, 7-73
Subroutines,
  BASIC, 6-42
  FORTRAN, 7-73
  mathematical, B-1
  PAL8 global calls, 5-22
  semiconditional BASIC, 6-43
Subscripts,
  BASIC, 6-16
  FORTRAN, 7-36
Substring function, 6-51, 6-52
Subtraction,
  BASIC, 6-17
  FORTRAN, 7-39
  PAL8, 5-18
.SV extension, 2-12, 3-59
Symbiont operation,
  CANCEL command, 3-5
  commands, H-1
  CUSP coding conventions, H-2
  REQUEST command, 3-57
  spooler program, 3-54
  writing your own, H-3
Symbol table,
  PAL8, 5-16, 5-35, 5-41
Symbolic editor, 4-1
Symbols,
  FORTRAN, 7-26
  PAL8, 5-13, 5-41
    use as instructions, 5-17
    use as operands, 5-18
    permanent, 5-14
    user-defined, 5-14
SYS device handler, F-4
System Hardware, 1-1
System information,
  additional documentation,
    Preface
  HELP command, 2-22
System date, 3-21
System device, 2-23
System table reset via USR, C-14

T field descriptor, 7-97
-T TTY dash option, 2-17
/T,
  COMPARE option, 3-6
  COPY command option, 3-14
  MAP command option, 3-48
  PAL8 option, 5-9
  RENAME command option, 3-56
  SUBMIT command option, 3-72, 8-2
TAB,
  BASIC function, 6-31
  character,
    PAL8 operator, 5-18
    PAL8 use, 5-12
  FORTRAN, 7-24
    function, 7-110
TAN BASIC function, 6-46
TANH FORTRAN function, 7-112
Tentative file lookup, C-7
Terminals,
  BASIC ESCape sequences, 6-76
  changing attributes, 2-5, 3-62
  control in BASIC, 5-59
  conventions, 2-7
  LA36, 2-6
  operating parameters, 2-6
  VT52, 2-6
  VT100, 2-5
TERMINATE command, 3-75
TEXT pseudo-operator, 5-37

TIME FORTRAN function, 7-112
.TM file name extension, 2-12
TRC function, 6-57
.TRUE. logical constant, 7-30
Truncation BASIC string
  arithmetic, 6-21
TTY device handler, F-5
TYPE command, 3-76
  option summary, 3-76
  error messages, 3-77

/U,
  COPY command option, 3-12
  CREF command option, 3-19
  DIRECT command option, 3-25
  SUBMIT command option, 3-72, 8-2
UA/UB/UC commands, 3-78
Unconditional transfer,
  BASIC, 6-36
  FORTRAN GOTO, 7-57
Underline (_) ODT command, 9-4
Unformatted FORTRAN I/O, 7-80
Unloading RL01K disk packs, 2-5
Uparrow (^),
  BASIC operator, 6-17
  echo control, 3-63
  ODT command, 9-4
  PAL8 operator, 5-18
Updating BASIC records, 6-71
Upper-case conversion, 6-54
User Service Routine (USR), C-1
User-defined functions in BASIC,
  6-78
USR, C-1
  CHAIN function, C-10
  CLOSE function, C-8
  DECODE function, C-10
  ENTER function, C-7
  ERROR function, C-11
  FETCH function, C-4
  INQUIRE function, C-13
  LOOKUP function, C-6
  RESET function, C-14
  restrictions, C-3
  standard call, C-1
  summary of functions, C-1
  USRIN function, C-12
  USROUT function, C-13
USRIN function (USR), C-12
USROUT function (USR), C-13

V,
  Editor list buffer command, 4-9
/V,
  COPY command option, 3-12
  DELETE command option, 3-24
  DIRECT command option, 3-25
  LIST command option, 3-41
  RENAME command option, 3-56
  TYPE command option, 3-76
VAL function, 6-53
Value assignment, BASIC, 6-23
Variables,
  BASIC, 6-14
  FORTRAN, 7-32
  numeric (BASIC), 6-15
  string (BASIC), 6-15
  subscripted (BASIC), 6-16
VLUx device handler, 3-68, F-5
VT52 terminal, 2-6
VT100 terminal, 2-5
VXA0,device handler,
  characteristics, F-4
  installing with SET, 3-68
  storage capacity, 2-23
  use, 2-25

W ODT command, 9-8
/W PAL8 option, 5-9
WIDTH SET command option, 3-62
Wildcards, 2-18
WRITE statement, 7-83

X FORTRAN field descriptor, 7-97
/X,
  COMPARE command option, 3-6
  CREF command option, 3-19
XLIST pseudo-operator, 5-34
.XOR. FORTRAN operator, 7-42

Y Editor yank command, 4-10

Z PAL8 page zero addressing
  pseudo-operator, 5-29
ZBLOCK pseudo-operator, 5-33
ZERO command, 3-15, 3-79

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 —