

RSX-11M-PLUS and Micro/RSX XDT Reference Manual

Order No. AA-JT78A-TC

RSX-11M-PLUS Version 4.0
Micro/RSX Version 4.0

First Printing, September 1987

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

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


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

Copyright ©1987 by Digital Equipment Corporation

All Rights Reserved.
Printed in U.S.A.

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

The following are trademarks of Digital Equipment Corporation:

DEC	EduSystem	UNIBUS
DEC/CMS	IAS	VAX
DEC/MMS	MASSBUS	VAXcluster
DECnet	MicroPDP-11	VMS
DECsystem-10	Micro/RSX	VT
DECSYSTEM-20	PDP	
DECUS	PDT	
DECwriter	RSTS	
DIBOL	RSX	

ZK4351

**HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS**

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

* Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminister, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by T_EX, the typesetting system developed by Donald E. Knuth at Stanford University. T_EX is a trademark of the American Mathematical Society.

Contents

Preface	v
---------	---

Summary of Technical Changes	xi
------------------------------	----

Chapter 1 The Executive Debugging Tool

1.1	The Advantage of XDT	1-1
1.2	How to Include XDT in Your RSX-11M-PLUS System	1-2
1.3	Loadable XDT on Micro/RSX and RSX-11M-PLUS Systems	1-2
1.4	Processor States	1-3
1.4.1	The Stack Depth Indicator and Interrupt Processing	1-4
1.5	Entering XDT	1-5
1.5.1	XDT and Synchronous System Traps	1-5
1.5.1.1	Processor Traps and System Crashes	1-8
1.5.2	Entering XDT from a Virgin System Boot	1-8
1.5.3	Using the BRK Command to Enter XDT	1-9
1.5.4	Using the BPT Instruction to Enter XDT	1-9
1.5.4.1	Using the OPEN Command to Insert a BPT Instruction	1-9
1.5.4.2	Using the ZAP Utility to Insert a BPT Instruction	1-11
1.5.5	Entering XDT When the System Is Hung (RSX-11M-PLUS)	1-11

Chapter 2 Debugging with XDT

2.1	Debugging with XDT	2-1
2.1.1	Using XDT to Debug the Executive	2-1
2.1.2	Using XDT to Debug Privileged Tasks	2-2
2.1.3	Using XDT to Debug a Driver	2-2
2.1.4	Using XDT to Examine a Memory Location	2-3
2.1.5	Turning Off the Processor Clock	2-4
2.1.6	T-Bit Error	2-5
2.2	Interpreting Bugchecks	2-5
2.3	XDT Commands and Operators	2-9

Chapter 3 Error Detection

3.1	Input Errors	3-1
3.2	Task Image Error Codes	3-2

Appendix A Processor Status Word

Appendix B Executive Symbols Supported by Loadable XDT

Index

Figures

A-1	Format of the Processor Status Word	A-1
-----	-------------------------------------	-----

Tables

1-1	XDT Trap Entry Codes	1-6
2-1	Common Facility-Independent Error Code Definitions	2-6
2-2	Standard Bugcheck Format Facility Code Definitions	2-8
2-3	Variables Used in XDT Command Descriptions	2-10
2-4	XDT Operators and Commands	2-10
B-1	Executive Symbols Supported by Loadable XDT	B-1

Preface

Manual Objectives

This manual describes the Executive Debugging Tool (XDT), which is used to debug privileged tasks on RSX-11M-PLUS and Micro/RSX systems. The manual provides reference information on all XDT commands, as well as information on how to use the commands to debug task images.

Intended Audience

This manual is intended for all systems and applications programmers who develop task images under the RSX-11M-PLUS or Micro/RSX operating systems. Readers should understand the user interface of the operating system that they are using. RSX-11M-PLUS users should be familiar with the contents of the *RSX-11M-PLUS Guide to Program Development* before reading this manual. Micro/RSX users should be familiar with the contents of the *Micro/RSX Guide to Advanced Programming* before reading this manual.

Structure of This Document

Chapter 1 gives information about XDT and describes how to enter XDT.

Chapter 2 describes XDT commands and operators and explains the differences between the On-Line Debugging Tool (ODT) and XDT.

Chapter 3 describes how XDT responds to errors in user input or program logic. It lists all XDT error message codes in alphabetical order.

Appendix A shows the format of the Processor Status Word (PSW) and summarizes the functions of its bits.

Appendix B lists the supported Executive symbols that loadable XDT will automatically search for in the Executive map.

Associated Documents

The *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver* contains information about debugging a user-written driver. The information directory of the host operating system describes other manuals that will be of interest to XDT users.

Conventions Used in This Document

The following conventions are used in this manual:

Convention	Meaning
>	A right angle bracket is the default prompt for the Monitor Console Routine (MCR), which is one of the command interfaces used on RSX-11M-PLUS systems. All systems include MCR.
\$	A dollar sign followed by a space is the default prompt of the DIGITAL Command Language (DCL), which is one of the command interfaces used on RSX-11M-PLUS and Micro/RSX systems. Many systems include DCL.
MCR>	This is the explicit prompt of the Monitor Console Routine (MCR).
DCL>	This is the explicit prompt of the DIGITAL Command Language (DCL).
xxx>	Three characters followed by a right angle bracket indicate the explicit prompt for a task, utility, or program on the system.
UPPERCASE	Uppercase letters in a command line indicate letters that must be entered as they are shown. For example, utility switches must always be entered as they are shown in format specifications.
command abbreviations	Where short forms of commands are allowed, the shortest form acceptable is represented by uppercase letters. The following example shows the minimum abbreviation allowed for the DCL command DIRECTORY: \$ DIR
lowercase	Any command in lowercase must be substituted for. Usually the lowercase word identifies the kind of substitution expected, such as a filespec, which indicates that you should fill in a file specification. For example: filename.filetype;version This command indicates the values that comprise a file specification; values are substituted for each of these variables as appropriate.

Convention	Meaning
/keyword, /qualifier, or /switch	A command element preceded by a slash (/) is an MCR keyword; a DCL qualifier; or a task, utility, or program switch. Keywords, qualifiers, and switches alter the action of the command they follow.
parameter	Required command fields are generally called parameters. The most common parameters are file specifications.
[option]	Square brackets indicate optional entries in a command line or a file specification. If the brackets include syntactical elements, such as periods (.) or slashes (/), those elements are required for the field. If the field appears in lowercase, you are to substitute a valid command element if you include the field. Note that when an option is entered, the brackets are not included in the command line.
[...]	Square brackets around a comma and an ellipsis mark indicate that you can use a series of optional elements separated by commas. For example, (argument[...]) means that you can specify a series of optional arguments by enclosing the arguments in parentheses and by separating them with commas.
:argument	Some parameters and qualifiers can be altered by the inclusion of arguments preceded by a colon. An argument can be either numerical (COPIES:3) or alphabetical (NAME:QIX). In DCL, the equal sign (=) can be substituted for the colon to introduce arguments. COPIES=3 and COPIES:3 are the same.
()	Parentheses are used to enclose more than one argument in a command line. For example: SET PROT = (S:RWED,O:RWED)
,	Commas are used as separators for command line parameters and to indicate positional entries on a command line. Positional entries are those elements that must be in a certain place in the command line. Although you might omit elements that come before the desired element, the commas that separate them must still be included.

Convention	Meaning
[g,m] [directory]	<p>The convention [g,m] signifies a User Identification Code (UIC). The g is a group number and the m is a member number. The UIC identifies a user and is used mainly for controlling access to files and privileged system functions.</p> <p>This may also signify a User File Directory (UFD), commonly called a directory. A directory is the location of files.</p> <p>Other notations for directories are: [ggg.mmm], [gggmmm], [ufd], [name], and [directory].</p> <p>The convention [directory] signifies a directory. Most directories have 1- to 9-character names, but some are in the same [g,m] form as the UIC.</p> <p>Where a UIC, UFD, or directory is required, only one set of brackets is shown (for example, [g,m]). Where the UIC, UFD, or directory is optional, two sets of brackets are shown (for example, [[g,m]]).</p>
filespec	<p>A full file specification includes device, directory, file name, file type, and version number, as shown in the following example:</p> <pre data-bbox="651 982 964 1010">DL2: [46,63] INDIRECT.TXT;3</pre> <p>Full file specifications are rarely needed. If you do not provide a version number, the highest numbered version is used. If you do not provide a directory, the default directory is used. Some system functions default to particular file types. Many commands accept a wildcard character (*) in place of the file name, file type, or version number. Some commands accept a filespec with a DECnet node name.</p> <p>A period in a file specification separates the file name and file type. When the file type is not specified, the period may be omitted from the file specification.</p> <p>A semicolon in a file specification separates the file type from the file version. If the version is not specified, the semicolon may be omitted from the file specification.</p> <p>A vertical ellipsis shows where elements of command input or statements in an example or figure have been omitted because they are irrelevant to the point being discussed.</p>
KEYNAME	<p>This typeface denotes one of the keys on the terminal keyboard, for example, the RETURN key.</p>
"print" and "type"	<p>The term "print" refers to any output sent to a terminal by the system. The term "type" refers to any user input from a terminal.</p>
black ink	<p>In examples, what the system prints or displays is printed in black.</p>

Convention	Meaning
red ink	In interactive examples, what the user types is printed in red. System responses appear in black.
xxx	A symbol with a 1- to 3-character abbreviation, such as <code>x</code> or <code>RET</code> , indicates that you press a key on the terminal. For example, <code>RET</code> indicates the RETURN key, <code>LF</code> indicates the LINE FEED key, and <code>DEL</code> indicates the DELETE key.
CTRL/g	The symbol <code>CTRL/g</code> means that you are to press the key marked CTRL while pressing another key. Thus, <code>CTRL/Z</code> indicates that you are to press the CTRL key and the Z key together in this fashion. <code>CTRL/Z</code> is echoed on some terminals as <code>^Z</code> . However, not all control characters echo.

Summary of Technical Changes

The following sections list features that are new to XDT or that have been modified for the RSX-11M-PLUS and Micro/RSX Version 4.0 operating systems. These new or modified features are documented in the *RSX-11M-PLUS and Micro/RSX XDT Reference Manual*.

Also, major changes to the organization of the manual are included at the end of this summary.

New or Modified Features

Loadable XDT has the following new or modified features:

- Loadable XDT is now supported on all RSX-11M-PLUS operating systems.
- Expanded entry display has been provided for loadable XDT.
- Support has been added for the automated searching of symbol addresses listed in the Executive map.

Loadable XDT

All RSX-11M-PLUS systems now support loadable XDT.

Expanded Entry Display

The expanded entry display gives you the contents of the Executive's registers, the kernel stack, and the addresses of the kernel APR5 and APR6 mapping registers at the time the system crashes.

Automated Searching of Symbol Addresses

The automated searching of symbol addresses listed in the Executive map requires only that you specify a supported symbol name instead of its address to display the contents at that address.

Changes to the Document

The information contained in the *RSX-11M-PLUS and Micro/RSX XDT Reference Manual* formerly resided in the *RSX-11M-PLUS and Micro/RSX Debugging Reference Manual*.

Chapter 1

The Executive Debugging Tool

The Executive Debugging Tool (XDT) is an interactive tool for debugging privileged code such as Executive modules, I/O drivers, interrupt service routines, and privileged tasks. The command interface is nearly identical to that of the On-Line Debugging Tool (ODT). You should be an experienced ODT user before using XDT. For more information on ODT, see the *RSX-11M-PLUS and Micro/RSX Debugging Reference Manual*.

Note

If you are not an experienced system programmer, you may find that experimenting with XDT produces undesirable results. As with ODT, where incorrect use could corrupt your program, incorrect use of XDT could corrupt your system. Use it carefully.

The major difference between XDT and ODT is that XDT is a tool for debugging privileged code—code that executes in system state or interrupt state—and ODT is a tool for debugging nonprivileged code—code that executes in user (or task) state. See Chapter 2 for a list of ODT commands not used in XDT and for a table of XDT operators and commands.

1.1 The Advantage of XDT

On RSX-11 systems without XDT support, any software fault occurring in system state or interrupt state results in a system crash. However, on RSX-11 systems that include XDT support, a software fault in system state or interrupt state causes the system to trap to XDT rather than crash.

When a software fault causes the system to trap to XDT, XDT has exclusive control of the system, and all other system activity is suspended. You can then use XDT commands and operators to examine registers, memory locations, and system data structures to locate the software fault that caused the trap—provided, of course, that the software fault is not one that corrupts either the XDT code itself or the trap vectors.

1.2 How to Include XDT in Your RSX-11M-PLUS System

System support for resident XDT is optional. You must explicitly specify during the system generation procedure that XDT support be included in your system image. See the system generation manual appropriate to your system for more information on including XDT support on your system.

On RSX-11M-PLUS systems with instruction and data space support, XDT supports both instruction space and data space referencing. Because XDT occupies physical address space without taking up Executive virtual data address space, including XDT in a system with instruction and data support does not reduce the size of available system pool.

On RSX-11M-PLUS systems without instruction and data space support, including XDT in the system diminishes the size of system pool by approximately 2.5 kilobytes (Kb).

1.3 Loadable XDT on Micro/RSX and RSX-11M-PLUS Systems

On Micro/RSX systems, XDT is a loadable system-level debugger. On RSX-11M-PLUS systems, XDT may be loadable or resident. When you load XDT, the LOAD task sets up XDT in a special partition in memory outside of the area generally allocated to Executive code. Using loadable XDT has the following advantages:

- Makes more of system pool available to the system
- Allows you to load XDT only when you need to use it for debugging
- Improves system performance when you use loadable XDT over resident XDT

To load XDT on a Micro/RSX or an RSX-11M-PLUS system, use the command format shown next.

Format

```
LOA /EXP=XDT[/PAR=parname][/HIGH][/FLAGS=n]
```

Parameters

/EXP=XDT

Causes XDT to be loaded as an extended Executive partition.

/PAR=parname

Specifies the partition in which XDT is to be loaded. If you omit the /PAR qualifier, GEN is the default partition name.

/HIGH

Causes XDT to be loaded at the top of the partition.

/FLAGS=n

Determines whether XDT displays startup messages when XDT is loaded into the system and whether XDT displays the expanded entry format when the system fatally traps to XDT. With startup messages enabled, the system returns with three messages indicating whether

XDT loaded successfully, the physical address of the pool node, and the starting address of XDT. With the expanded entry format enabled, XDT displays the following information:

- The Executive's registers at the time of the crash
- The contents of the kernel stack
- The addresses of the kernel APR5 and APR6 mapping registers

The following values for n determine the displays given:

n	Expanded Entry Display	Startup Messages
0	No	Yes
1	Yes	Yes
2	No	No
3	Yes	No

See Section 1.5.1 for an example of the startup messages and the expanded entry format.

To unload XDT on a Micro/RSX or an RSX-11M-PLUS system, use the command format shown next.

Format

```
UNLOAD /EXP=XDT
```

For more details on using the LOAD command, see the *Micro/RSX System Manager's Guide* or the *RSX-11M-PLUS and Micro/RSX System Management Guide*.

1.4 Processor States

XDT responds to software faults occurring in system state or interrupt state. RSX-11 systems operate in various software states, depending on the type of processing taking place at a given time. (Software states are different from hardware processor modes. See the appropriate *PDP-11 Processor Handbook* or the *PDP-11 Architecture Handbook* for a description of processor modes.) A summary of the system's software states follows:

- **User state:** The processing state in which the system executes nonprivileged user task code. In user state, the processor operates in user mode or supervisor mode and is fully interruptible (PR0). The system stack is empty whenever user state code is executing.
- **System state:** The state in which the system executes privileged code. It is the only state in which the shared system database may be safely modified. In system state, the processor operates in kernel mode and is completely interruptible (PR0).
- **Interrupt state:** The Executive uses interrupt state to perform device-critical processing after a peripheral device interrupt. In interrupt state, the processor operates in kernel mode and can be either partially interruptible (PR4 to PR6) or completely noninterruptible (PR7). The shared system database cannot be safely modified in interrupt state, since operation at interrupt state may have preempted a system state process that may already have been

modifying a system data structure. Driver processes may switch from interrupt state to system state by calling the \$FORK word to safely access shared system data.

When XDT executes, it has complete control of the system, so you can examine or modify any system data structure. XDT runs at PR7 and, therefore, is totally noninterruptible. Pending interrupts must wait until you exit from XDT and resume normal execution.

Refer to the descriptions of interrupt processing and the \$FORK process in the *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver* for more information.

For a detailed description of processor interrupt priorities, refer to your *PDP-11 Processor Handbook*.

The following priority scheme governs the order by which the system services the processing states (interrupt level 7 is the highest priority interrupt):

1. Interrupt State Processing:
 - PR7—Various device interrupts
 - PR6—Various device interrupts
 - PR5—Various device interrupts
 - PR4—Various device interrupts
2. System State Processing:
 - Processing of traps from user state
 - Processing of driver processes suspended by \$FORK
3. User State Processing:
 - Processing of all user tasks

1.4.1 The Stack Depth Indicator and Interrupt Processing

The Executive maintains a word, called the Stack Depth Indicator (\$STKDP), to indicate the number of interrupted system state processes that must be completed before servicing any suspended system state processes and returning to user state. A second word, \$FORK, heads a list of suspended driver processes awaiting execution in system state.

While executing nonprivileged task code in user state, \$STKDP contains a value of +1, and the processor uses the user stack. When \$STKDP contains a value other than +1, the Executive operates in system state or interrupt state, and the processor uses the kernel stack.

As part of the process of switching from user state to system or interrupt state (caused, for example, by a device interrupt or an Executive directive issued by a user state task), the Executive decrements \$STKDP to 0. Each subsequent interrupt causes the Executive to further decrement \$STKDP (-1, -2, and so on). The value (other than +1) contained in \$STKDP indicates the number of interrupted system state processes waiting to be returned to the Executive.

As each interrupt level process completes its interrupt state execution, the Executive increments \$STKDP until all the suspended interrupt state processes have been serviced and \$STKDP again contains a zero (0). The Executive then services any system state processes still waiting (in the \$FORK queue) before switching to user state and the user stack.

1.5 Entering XDT

Entry to XDT may or may not be intentional. For example, a coding change to a driver or a part of the Executive might introduce an error, such as an illegal instruction. The execution of the illegal instruction would cause the system to fault and immediately enter XDT. In this case, entry to XDT is obviously not intentional.

You may, however, deliberately cause the system to enter XDT. You might want to do so for the following two reasons:

1. To locate a suspected software bug in a particular piece of system code
2. To test and debug a new or recently modified section of code

The following sections discuss system traps (the mechanism by which the system enters XDT) and several ways to cause the system to trap to XDT.

1.5.1 XDT and Synchronous System Traps

A system trap is an event that transfers program control from the program through a trap vector to a trap handling routine. System traps usually occur due to the execution of either an illegal instruction or a specific trap-causing instruction. Traps provide software with a means of monitoring and reacting to those events. The Executive initiates corresponding system trap processing when particular events occur.

Synchronous system traps (SSTs) are events that occur at the same time and in direct relation to the incorrect execution of program instructions. SSTs that occur in system state are the means by which XDT gains control of the operating system instead of letting it immediately crash.

In a system with XDT support, all SSTs occurring at system state (except as described for the TRAP instruction; see Section 1.5.1.1) result in a trap to XDT. A trap to XDT is indicated by a message at the console terminal in the form shown next.

Format

xx: address [facility error F/N]

Parameters

xx:

Specifies one of the entry codes listed in Table 1-1.

address

Indicates the program counter (PC) at the time of the trap.

facility

Specifies one of the octal facility codes listed in Chapter 2, Table 2-2. This information is provided only with the BC entry code.

error

Specifies one of the octal error codes listed in Chapter 2, Table 2-2. This information is provided only with the BC entry code.

F/N

Specifies either the letter F or the letter N, which represents fatal and nonfatal faults, respectively. If the letter N appears, XDT allows you to use the Proceed command (P). This information is provided only with the BC entry code.

Each entry code is associated with a particular trap vector, which is also listed in Table 1-1. See Chapter 3 for more information about error detection.

For more information on the optional parameters provided with the BC entry code, see Chapter 2.

Table 1-1: XDT Trap Entry Codes

Entry Code	Vector	Fatal	Reason
BC:	30	Varies	Bugcheck—internally detected software fault. Fatalness indicated in message line (loadable XDT only)
BE:	14	No	Breakpoint entry—a BPT instruction
EM:	30	No	EMT instruction
IL:	10	Yes	Illegal instruction
IO:	20	No	IOT instruction
MP:	250	Yes	Memory-protection violation
OD:	4	Yes	Odd address or nonexistent memory
SO:	4	Yes	Stack overflow
TE:	14	No	T-bit trap
nB:	14	No	XDT set breakpoint (n is a register number 0-8)

To begin determining the cause of a system failure, you can specify loadable XDT to give an extended entry display of the contents of the Executive's registers, the kernel stack, and the addresses of the kernel APR5 and APR6 mapping registers at the time the system crashes. The information contained in this display may indicate the location of the code causing the trap. The following example illustrates how to set loadable XDT to give you the display and a sample display:

```
>LOA /EXP=XDT/FLAGS=1 [RET] ①
XDT -- Initialization complete and successful
-- Node address = 004220
-- XDT address = 01010600
```

```

OD:000001 ②
XDT>$0/007760 $1/001110 $2/000001 $3/000000 ③
XDT>$4/000001 $5/000000 $6/001004 $7/000340
XDT>
XDT>$6/001004 q ④
D 001004 /017446
D 001006 /030000
D 001010 /000021
D 001012 /125244
D 001014 /127134
D 001016 /000004
D 001020 /000002
D 001022 /120446
D 001024 /131614
D 001026 /170000
D 001030 /000000 172352/002504 172354/005015 ⑤
XDT>172372/003504 172374/010240
XDT> ⑥

```

- ① When you load XDT (assuming /FLAGS equals 0 or 1), the system returns with three messages indicating the following:
 - 1 Whether XDT loaded successfully
 - 2 The physical address of the pool node containing XDT transfer vectors (For more information on pool nodes, see the *RSX-11M-PLUS and Micro/RSX System Management Guide*.)
 - 3 The physical starting address of the XDT partition

Otherwise, the system returns the command line interpreter (CLI) prompt.

Specifying the /FLAGS=1 qualifier indicates that you also want the expanded entry format displayed.
- ② The XDT trap entry code (in this case, an odd address or nonexistent memory trap).
- ③ The general registers and the addresses contained in each register.
- ④ The kernel stack, including the addresses in the Executive's registers.
- ⑤ The addresses of the kernel APR5 and APR6 mapping registers.
- ⑥ The XDT prompt. From here you can use XDT commands to determine the cause of the crash.

1.5.1.1 Processor Traps and System Crashes

An SST generally indicates that there is a software fault that may cause corruption of an Executive database. For example, inserting an odd address into the link pointer of a Task Control Block (TCB) results in an odd address trap the next time the Executive reads through the TCB list.

The PDP-11, including the MicroPDP-11, has four trap instructions: EMT, IOT, BPT, and TRAP. These instructions have the following results when executed in system state:

- Both the EMT and IOT instructions are fatal when executed in system state ($\$STKDP \leq 0$).
- The BPT instruction is fatal when executed in system state, unless XDT is present in the system. The BPT instruction is a means for entering XDT. When you deliberately set a breakpoint in XDT (see the XDT nB command in Chapter 2, Table 2-4), XDT actually inserts a BPT instruction where you want the breakpoint and saves the instruction it replaced with the breakpoint.
- The TRAP instruction is legal only in system state ($\$STKDP = 0$). The directive processors use the TRAP instruction to post error codes back to the directive dispatcher.

1.5.2 Entering XDT from a Virgin System Boot

A virgin system is an RSX-11M-PLUS system that has completed execution of SYSVMR.CMD but has not yet been saved (see the Monitor Console Routine (MCR) command SAVE in the *RSX-11M-PLUS MCR Operations Manual*). If the virgin system includes XDT support, the normal system startup immediately transfers control to XDT, which displays a message on the system console terminal similar to the following:

```
B00 DL: [1,54]
XDT: 35
XDT>
```

The number following the colon (:) is the system base level number.

When the system traps to XDT in this situation, the system initialization code (INITL) has not yet executed. Therefore, some system data structures have not yet been defined or initialized. (On systems supporting memory management, memory management has not yet been enabled.)

After you have set any desired breakpoints, enter the XDT command G (Go) to return control to the Executive module INITL. The INITL module then continues with the system initialization. If a software fault occurs during system initialization (or if you have previously set a breakpoint in the INITL module), the system traps to XDT upon encountering the fault (or the breakpoint) instead of causing a fatal system crash. You can then use XDT to try to locate the error that caused the fault or to take the appropriate action for the specified breakpoint.

Note that a saved system does not trap to XDT when it is bootstrapped. Part of the action of INITL is to deallocate the memory it uses to system pool after it completes execution. In other words, INITL is not part of the system image after SAVE executes. Furthermore, because SAVE has control of the system when it copies the system image to disk, it—not XDT—retains initial control when the system is rebooted, unless a suitable breakpoint has been placed in the SAVE module.

Also note that if you save the virgin system with breakpoints set, the saved system traps to XDT each time it reaches one of those breakpoints. This may occur both while the system is being saved and during the system reboot.

1.5.3 Using the BRK Command to Enter XDT

The BRK (Breakpoint to Executive Debugging Tool) command passes control of the system to XDT. Note that on RSX-11M-PLUS systems this is an MCR command. The message XDT prints on the console includes a program counter (PC) inside the MCR task. This is a convenient way to invoke XDT. From this point you can map to any desired location (for example, within a driver) and set breakpoints. If XDT is not included in the system, the BRK command has no effect.

Typing the XDT command P (Proceed) normally restores the system to the state that existed just before the execution of the BRK command.

1.5.4 Using the BPT Instruction to Enter XDT

There are several ways to replace a system instruction with a BPT instruction to cause the system to trap to XDT. The simplest method is to use the BRK command. You can also use the OPEN command, for putting breakpoints into drivers or memory-resident privileged tasks, or the ZAP utility, for putting breakpoints in a privileged program before you run it. Or, you can include a BPT instruction in the macro source program before assembling it. (This last method is useful when debugging a driver.)

The general procedure for setting and canceling the BPT instruction is as follows:

1. Replace a system state instruction with the BPT instruction.
2. The system traps to XDT when it executes the BPT instruction.
3. Use XDT to restore the original instruction replaced by BPT.
4. Decrement the PC by subtracting 2 from the contents of register R7.
5. Set any desired breakpoints by using XDT commands and proceed with the XDT P (Proceed) or S (Single Step) command.

If you include a BPT instruction in the source code, you are not replacing an instruction. Therefore, there is no need to decrement the PC or to restore any instruction. When you have debugged the driver, take out the BPT instruction and reassemble the source code.

1.5.4.1 Using the OPEN Command to Insert a BPT Instruction

You can use the OPEN command to replace an instruction in the Executive, a device driver, or a memory-resident privileged task with the BPT instruction. Note that on RSX-11M-PLUS systems this is an MCR command. With this method, the BPT instruction affects only the image in memory, not the image on disk. Therefore, rerunning or reinstalling the image wipes out any BPT instruction set with the OPEN command.

To examine and replace an instruction in the Executive, use the command syntax shown next.

Format

```
OPEN addr[/KNLI]
```

Parameters

addr

Specifies the address in the Executive that is to be opened.

/KNLI

Accesses the contents of the address in the Executive. Specify this keyword if you need to place a breakpoint into a directive common.

To examine and replace an instruction in a memory-resident privileged task, use the command syntax shown next.

Format

```
OPEN addr/TASK=taskname
```

Parameters

addr

Specifies the address in the task that is to be opened.

/TASK=taskname

Specifies the name of the memory-resident task.

To examine and replace an instruction in a driver, use the command syntax shown next.

Format

```
OPEN addr/DRV=ddnn:
```

Parameters

addr

Specifies the address in the driver that is to be opened.

/DRV=ddnn:

Specifies the device mnemonic (ddnn:) for the driver to be opened.

Note

To examine and replace an instruction within a privileged task, the privileged task must be fixed in memory.

Once you have completed testing or debugging, use XDT to restore the original instruction replaced by BPT, as follows:

- Decrement the PC by subtracting 2 from the contents of register R7.
- Set any desired breakpoints by using XDT commands and proceed with the XDT P (Proceed) or S (Single Step) command.

1.5.4.2 Using the ZAP Utility to Insert a BPT Instruction

Because the OPEN command operates only on the running system, any changes made to the system with the OPEN command are lost when the system is rebooted. One way to permanently retain those changes is to save the system.

Another way to permanently retain changes to the system is by using the Task/File Patch Program (ZAP). ZAP lets you modify the system image on disk. If you manually set a BPT instruction in the Executive code by using ZAP, the system permanently retains that breakpoint on disk. The trap to XDT occurs when the image is running in memory. To remove the breakpoint, you must use ZAP to restore the original instruction on the disk image.

You can also use the ZAP utility to debug an overlaid privileged task. Suppose, for example, that you want to set a breakpoint in an overlay segment of a privileged task. Since this segment is not in memory, you cannot use the OPEN command to insert a BPT instruction; you would have to use ZAP to insert the BPT instruction in the privileged task's disk image file.

1.5.5 Entering XDT When the System Is Hung (RSX-11M-PLUS)

If the system is hung and commands are not being processed, you cannot force the system to enter XDT as previously described. However, any processor traps in system state force the system to enter XDT. One way to force the system to enter XDT is to use the switch registers or the console to change the clock interrupt service routine to an odd address. For example, if the system has a KW11-L clock with vector 100, simply deposit an odd address into location 100 with the switch register. At the next clock interrupt, the system traps to XDT.

If the system is stuck in a tight loop, halt the central processing unit (CPU), examine the PC, deposit an odd address in the PC, and then continue. Since the system will attempt to execute an odd address, it will trap to XDT.

Chapter 2

Debugging with XDT

This chapter describes how to use the Executive Debugging Tool (XDT). It includes information about interpreting bugchecks, which detect certain types of internal system corruption. A table of XDT operators and commands is included in this chapter, as well as a table of the On-Line Debugging Tool (ODT) commands not used in XDT.

2.1 Debugging with XDT

Once the system traps to XDT, you are on your own to do whatever testing or debugging is necessary. Virtually all other system activity has ceased. However, the system clock continues to run.

If the trap to XDT was unintentional, you can try to isolate the fault that caused the trap. The *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver* contains some helpful information on isolating faults and tracing system faults by using specific system data structures.

If you intentionally caused the system to trap to XDT for testing or debugging purposes, remember to restore the instruction you replaced with the BPT instruction and decrement the program counter (PC) by 2.

You can cause the system to enter the system crash dump routine by entering the XDT command X. (In ODT, this same command merely causes ODT to exit.)

2.1.1 Using XDT to Debug the Executive

Because the Executive executes only in system state, you can cause a trap to XDT by setting a breakpoint anywhere within the Executive code. Because the Executive is always mapped, you can set breakpoints within Executive code at any time.

Note

The directive commons portion of the Executive is unmapped. Therefore, setting breakpoints in this area yields no real benefits in debugging Executive code.

With the XDT command S, you can single step through the execution of individual instructions in the Executive code. When you have finished testing or debugging, you can resume the execution of the system by using the XDT command P, or you can cause the system to enter the crash dump routine by entering the XDT command X.

2.1.2 Using XDT to Debug Privileged Tasks

A privileged task must be executing in system state ($\$STKDP = 0$) in order to trap to XDT. If a privileged task encounters a fault while executing in user state, the task either aborts or traps to ODT (if that task was task-built to include ODT).

XDT and ODT processing are completely independent of each other. You can use XDT to debug the portions of a privileged program that execute in system state, regardless of the presence of ODT. You can use ODT to debug those portions of the same task that execute in user mode.

Whenever you attempt to set breakpoints in a task with XDT, you must make sure that the task is mapped. One way to do this is to assemble a BPT instruction into the task source code at the beginning of the system state code. When the system encounters the BPT instruction, it traps to XDT with the task mapped. At this point, you can use any of the XDT commands and operators.

You can also fix the task in memory and use the OPEN command to set a breakpoint. The advantage of this method is that you do not need to reassemble and rebuild the privileged task to insert the breakpoint. The disadvantage is that you must decrement the PC and replace the original instruction.

Note

Fixing the task and using the OPEN command does not work when the system state code is contained in an overlay.

2.1.3 Using XDT to Debug a Driver

I/O drivers in RSX-11 systems can operate in system or interrupt state. You can use XDT in either of these states to set breakpoints and to examine or modify driver data structures to perform debugging operations.

You must make sure that the driver is mapped whenever you attempt to set breakpoints in it with XDT. One way to do this is to assemble a BPT instruction into the driver source code at one of the normal entry points to the driver. When the system encounters the BPT instruction, it traps to XDT with the driver mapped. At this point, you can use any of the XDT commands and operators.

You can also use the OPEN command to replace an instruction in the driver with a BPT instruction. The advantage of this method is that you do not need to reassemble and rebuild the driver to remove the breakpoint. The disadvantage is that you must decrement the PC and replace the original instruction upon encountering the breakpoint.

A third method of inserting a breakpoint is to force the driver to be temporarily mapped through an Active Page Register (APR). You cannot set XDT breakpoints in this manner (see the XDT command ;Bn), but you can replace an instruction in the driver with a BPT instruction after the driver is mapped (just as you would if you used the OPEN command to replace the instruction).

In the following example, assume that you have already entered XDT by using the BRK command. By looking at a PAR listing, you know that the driver is at physical address 210400. From this point you can proceed as follows:

1. Replace the current mapping context (3163) with the starting address of the driver (2104).
2. Replace the MOV instruction (010405) with a BPT instruction (3).
3. Replace the driver address (2104) with previous mapping context (3163).
4. Enter the XDT command P (Proceed).

When the system encounters the BPT instruction set in the driver, it traps to XDT from that breakpoint (BE:120104). You can then begin debugging procedures with XDT commands and operators. The sequence appears in the following example:

```
XDT>172352/ 003163 2104 [RET]
XDT>120102/ 010405 3 [RET]
XDT>172352/ 002104 3163 [RET]
XDT>P
```

```
BE:120104
XDT>$/ 120104 120102 [RET]
XDT>
```

This method is useful if you discover at an inopportune time that you would like to set a breakpoint in the driver.

2.1.4 Using XDT to Examine a Memory Location

This is actually more difficult than it seems. Consider the following: The system has trapped to XDT and you want to examine a memory location in a partition that contains a device driver. Suppose you have a 20K Executive running on a mapped system. Kernel APRs 0 to 4 map the Executive while the system is in XDT. Kernel APR7 maps the I/O page. Kernel APRs 5 and 6 contain the APR bias of whatever is mapped at the time the system enters XDT (that is, APRs 5 and 6 map the Monitor Console Routine (MCR) task if the BRK command caused the trap to XDT).

In the following example, assume that the system has already trapped to XDT and that you want to examine locations in the kernel data space portion of memory. To examine a memory location, proceed as follows:

1. Divide the physical address (107432) into two components: the relocation bias (1074) and the displacement (32).
2. Manually map this section of physical memory by putting the relocation bias (1074) into data space kernel APR5.
3. Examine the location by referencing the virtual address as 120000 + displacement (32).
4. Replace the APR bias with original mapping context (3163) after examining the memory location. This step is not mandatory, but it does represent good debugging practice.

The sequence appears in the following example:

```
XDT>172372/ 3163 1074 [RET]
XDT>120032/ 10403 [RET]
XDT>172372/ 3163 [RET]
XDT>
```

Note

If you were to map with kernel APR6, then the virtual address would be 140000 + displacement.

Kernel APRs 0 to 7 map the following range of addresses (see the *PDP-11 Processor Handbook* for more information on the kernel APRs):

I-Space			D-Space		
Kernel	APR0	172340	Kernel	APR0	172360
	APR1	172342		APR1	172362
	APR2	172344		APR2	172364
	APR3	172346		APR3	172366
	APR4	172350		APR4	172370
	APR5	172352		APR5	172372
	APR6	172354		APR6	172374
	APR7	172356		APR7	172376

2.1.5 Turning Off the Processor Clock

It is sometimes necessary in a debugging session to single step through code in the Executive. To do this type of debugging with some parts of the Executive (for instance, with interrupt handling routines), it is necessary to have the system completely inactive. It may, therefore, be necessary to turn off the clock, which usually interrupts at a rate of 60 times a second.

For example, if the system has a KW11-L line clock with the control and status register (CSR) address 177546, you can turn off the clock by placing a zero (0) in the CSR—this action clears the interrupt enable bit in the clock CSR.

2.1.6 T-Bit Error

Using XDT to debug a user-written driver has special pitfalls. One problem that can arise is a T-bit error as follows:

```
TE: address  
XDT>
```

Generally, a trace (T) bit trap occurs when the T-bit is set in the Processor Status Word (PSW) by some mechanism other than a breakpoint or an XDT P or S command. The T-bit error results when control reaches a breakpoint that you have set, using XDT, in a loaded driver. The T-bit error, rather than the expected BE: trap, occurs unless kernel APR5 maps to the driver at the time XDT sets the breakpoint.

If you want to set a breakpoint in a loaded driver, you cannot set the breakpoint with XDT until the driver is mapped (that is, you cannot set a breakpoint in a driver if you entered XDT by using the MCR command BRK).

You can avoid this T-bit error by assembling the driver with an embedded BPT instruction or by using either the ZAP utility or the OPEN command to replace a driver instruction with the BPT instruction.

Another method is to use the BRK command to enter XDT. Then, use kernel APR5 to map to the driver, to deposit a BPT instruction in the driver using XDT, and to restore the original contents of kernel APR5. Return to user mode by using the XDT command P.

2.2 Interpreting Bugchecks

The RSX-11M-PLUS and Micro/RSX Executives all contain code that detects certain types of internal system corruption. If XDT is included in the system, the Executive attempts to enter XDT as soon as the system corruption is detected. By doing this, the system will more likely be in a state where the fault that caused the corruption can be isolated.

For reporting this type of fault, RSX-11M-PLUS uses the bugcheck, which uses the EMT instruction to enter XDT. On RSX-11M-PLUS systems with resident XDT, XDT prompts with the following:

```
EM: nnnnnn  
XDT>
```

Use the following two steps to isolate the failure:

1. Find the location of the EMT instruction in the source code for the Executive.
2. Ascertain the type of corruption from the context of the EMT. The EMT instructions included in the source code for this purpose are typically generated by the CRASH macro.

In systems with loadable XDT, two additional pieces of information are provided. These are the facility code and the error code. The facility code indicates which component of the system

detected the fault. The error code indicates what fault was detected. Loadable XDT prompts with the following:

BC:nnnnn ffffff eeeee s

Parameter	Meaning
nnnnnn	Specifies the address within the Executive where the bugcheck was executed.
ffffff	Specifies the octal facility code.
eeeeee	Specifies the octal error code.
s	Specifies either the letter F or the letter N, which represents fatal and nonfatal faults, respectively. If the letter N appears, XDT allows you to use the Proceed command.

Table 2-1 shows error codes that are independent of which facility detected the fault. The high bit for these error codes will always be zero. The definition, symbolic name, and octal value of each code are shown.

Table 2-2 shows facility codes and error codes for errors that can only be issued by a particular facility. The high bit for these error codes will always be zero. The definition, symbolic name, and octal value of each code are shown.

Table 2-1: Common Facility-Independent Error Code Definitions

Error	Error Code	Meaning
Synchronous system traps (SST) type errors—Major error code 1		
BE.ODD	000100	Odd address or other trap 4
BE.SGF	000102	Segment fault
BE.BPT	000104	Breakpoint or T-bit trap
BE.IOT	000106	IOT instruction
BE.ILI	000110	Illegal instruction
BE.EMT	000112	EMT instruction
BE.TRP	000114	TRAP instruction
BE.STK	000116	Stack overflow
Internal inconsistency errors—Major error code 2		
BE.NPA	000200	Task with no parent aborted (P/OS)
BE.SGN	000201	Feature not included in system
BE.2FR	000202	Double fork detected

Table 2-1 (Cont.): Common Facility-Independent Error Code Definitions

Error	Error Code	Meaning
BE.ISR	000203	Interrupt service routine modified R0-R3
BE.FHW	000204	Fatal hardware error
BE.CSR	000205	Device control and status register (CSR) disappeared during powerfail
BE.IDC	000206	Internal database consistency error
BE.ACP	000207	Ancillary Control Processor (ACP) task aborted
BE.HSP	000210	Header subpacket problem in Error Logging
BE.NCT	000211	No current task

System pool-related errors—Error code 3

BE.NPL	000300	No pool for operation
BE.DDA	000301	Double deallocation
BE.SIZ	000302	Size of block invalid
BE.BAK	000303	Deallocated block below pool
BE.POV	000304	Deallocation overlaps end of pool
BE.FSI	000305	Fragment with invalid size detected

Group global event flag errors—Error code 4

BE.GGF	000400	Task locked to nonexistent flags
--------	--------	----------------------------------

Table 2-2: Standard Bugcheck Format Facility Code Definitions

Error	Error Code	Meaning
I/O driver subsystem—Facility code 2		
BF.TTD	000200	Terminal driver
Executive components—Facility code 3		
BF.EXE	000300	Exec—General and miscellaneous
BF.XDT	000301	Exec—Executive Debugging Tool (XDT)
BF.MP	000302	Exec—Multiprocessing
Multiprocessor-specific-type errors		
BE.NDS	100100	Init failure—Data space not loaded
BE.NCK	100200	Clock not available
BE.URM	100300	Fork to offline UNIBUS run
BE.WTL	100400	Attempt to lock already owned lock
BE.UNO	100500	Attempt to unlock not by owner
BE.ILC	100600	Illegal lock count value
BE.LNS	100700	Lock not locked
BE.OCP	101000	At entry, another central processing unit (CPU) showed ownership
BE.MLK	101100	Attempt to exit multiple lock
BE.NIN	101200	No reason for interprocessor interrupt
BE.UNP	101300	Some UNIBUS run not connected
BF.POL	000303	Exec—Pool handling routines (CORAL)
BF.ERR	000304	Exec—Hardware error processing subsystem
BF.INT	000305	Exec—Internal consistency checking routine
BF.INI	000306	Exec—INITL—initialization module
BF.DVI	000307	Exec—DVINT common interrupt handler
BF.PAR	000310	Exec—Parity memory support
BF.XIT	000311	Exec—Task exit/abort processing

Table 2-2 (Cont.): Standard Bugcheck Format Facility Code Definitions

Error	Error Code	Meaning
BF.QIO	000312	Exec—QIO directive
BF.OPT	000313	Exec—Seek optimization
BF.ACC	000314	Exec—System resource accounting
BF.KAS	000315	Exec—Kernel asynchronous system trap (AST) support
BF.DIR	000316	Exec—Miscellaneous directives
BF.SAN	000317	Exec—Crash with sanity timer message

2.3 XDT Commands and Operators

XDT commands are generally compatible with ODT commands. However, XDT does not contain the following commands that are available in ODT:

- No \$M—Mask register
- No \$X—Entry flag registers
- No \$V—SST vector registers
- No \$D—I/O logical unit number (LUN) registers
- No \$E—SST data registers
- No \$W—\$DSW (Directive Status Word) word
- No E —Effective address search command
- No F —Fill memory command
- No N —Word search command
- No V —Restore SST vectors command
- No W —Memory word search command

The command descriptions in Table 2-4 use lowercase alphabetic variables to represent numeric and alphabetic arguments specified in commands. These variables are explained in Table 2-3.

Table 2-3: Variables Used in XDT Command Descriptions

Variable	Meaning
a	An octal address expression representing the address of a task image location.
#symbol	Identical to a where a represents the octal address of a symbol listed in the Executive map. XDT automatically searches an internal symbols table that corresponds to the Executive map, finds the address of the symbol, and then displays the contents of that location. For example, specifying # <code>\$ACTHD/</code> displays the contents of the location that the symbol <code>\$ACTHD</code> maps to in the Executive. For a list of the supported symbols, see Appendix B (loadable XDT only).
k	An octal value up to six digits long with a maximum value of <code>177777₈</code> , or an expression representing such a value. An expression may include arithmetic operators or indicators. If more than six digits are specified, XDT truncates to the low-order 16 bits. If the octal value is preceded by a minus sign, XDT takes the two's complement of the value.
n	An octal integer between 0 and 7.
x	An alphabetic character. A list of legal alphabetic characters is given in Table 2-4 where the variable x is used.

All XDT command I/Os go to or from the console terminal. Table 2-4 contains all of the XDT commands and operators.

Table 2-4: XDT Operators and Commands

Format	Meaning
+ (plus sign) or space	Arithmetic operator used in expressions. Add the preceding argument to the following argument to form the current argument.
- (minus sign)	Arithmetic operator used in expressions. Subtract the following argument from the preceding argument to form the current argument. Also used as a unary operator to indicate a negative value.
, (comma)	Argument separator. Separates the number of a relocation register from a relative location to specify a relocatable address.
* (asterisk)	Radix-50 separator used in constructing Radix-50 words.
. (period)	Current location indicator. Causes the address of the last explicitly opened location to be used as the current address for XDT operations.
; (semicolon)	Argument separator. Separates multiple arguments, allowing an address expression or XDT register value to be identified.
<code>[RET]</code> (RETURN command) or k <code>[RET]</code>	Command that closes the currently open location and prompts for the next command. If RETURN is preceded by k, the value k replaces the contents of the currently open location before it is closed.
<code>[LF]</code> (LINE FEED command) or k <code>[LF]</code>	Command that closes the currently open location, opens the next sequential location (a word or a byte, depending on the mode in effect), and displays its contents. If LINE FEED is preceded by k, the value k replaces the contents of the currently open location before it is closed.

Table 2-4 (Cont.): XDT Operators and Commands

Format	Meaning
^ or k^	Command that closes the currently open location, opens the immediately preceding location (a word or a byte, depending on the mode in effect), and displays its contents. If ^ is preceded by k, the value k replaces the contents of the currently open location before it is closed.
_ or _k	Command that interprets the contents of the currently open location as a program counter (PC) relative offset and calculates the address of the next location to be opened; closes the currently open location and opens and displays the contents of the new location (a word or a byte, depending on the mode in effect) thus evaluated. If _ is preceded by k, the value k replaces the contents of the currently open location before it is closed.
@ or k@	Command that interprets the contents of the currently open word location as an absolute address, closes the currently open location, and opens and displays the contents of the absolute location (a word or a byte, depending on the mode in effect) thus evaluated. If @ is preceded by k, the value k replaces the contents of the currently open location before it is closed.
> or k>	Command that interprets the low-order byte of the currently open word location as a relative branch offset and calculates the address of the next location to be opened; closes the currently open location and opens and displays the contents of the relative branch location (a word or a byte, depending on the mode in effect) thus evaluated. If > is preceded by k, the value k replaces the contents of the currently open location before it is closed.
< or k <	Command that closes the currently open location (opened by a _, @, or > command) and reopens the previous location (a word or a byte, depending on the mode in effect). If the currently open location was not opened by a _, @, or >, then < simply closes and reopens the current location. If < is preceded by k, the value k replaces the contents of the currently open location before it is closed.
\$n	Expression that represents the address of one of eight general registers, where n is an octal digit identifying R0-R7. The initial contents of these locations represent the general register content at the time XDT received control. By changing these locations, you can change the register contents for when control is restored to the Executive (using the S, P, or G command).
\$x or \$nx	Expression that represents the address of one of XDT's internal registers, where x is one of the following alphabetic characters, and n is one octal digit. Registers exist within XDT in the following order: <ul style="list-style-type: none"> S Processor Status register (hardware PS) A Search argument register L Low memory limit register H High memory limit register

Table 2-4 (Cont.): XDT Operators and Commands

Format	Meaning
	C Constant register
	Q Quantity register
	F Format register
	nB Breakpoint address registers
	nG Breakpoint proceed count registers
	nI Breakpoint instruction registers
	nR Relocation registers
" or a"	Word mode American Standard Code for Information Interchange (ASCII) operator. Interprets and displays the contents of the currently open (or the previously opened) location as two ASCII characters and stores this word in the quantity register (\$Q). If " is preceded by a, the value a is taken as the address of the location to be interpreted and displayed.
' or a'	Byte mode ASCII operator. Interprets and displays the contents of the currently open (or the previously opened) location as one ASCII character and stores this byte in the quantity register (\$Q). If ' is preceded by a, the value a is taken as the address of the location to be interpreted and displayed.
% or a%	Word mode Radix-50 operator. Interprets and displays the contents of the currently open (or the previously opened) location as three Radix-50 characters and stores this word in the quantity register (\$Q). If % is preceded by a, the value a is taken as the address of the location to be interpreted and displayed.
/ or a/	Word mode octal operator. Displays the contents of the last word location opened and stores this octal word in the quantity register (\$Q). If / is preceded by a, the value is taken as the address of a word location to be opened and displayed.
\ or a\	Byte mode octal operator. Displays the contents of the last byte location opened and stores this octal byte in the quantity register (\$Q). If \ is preceded by a, XDT takes the value a as the address of a byte location to be opened and displayed.
k=	Command that interprets and displays expression value k as six octal digits and stores this word in the quantity register (\$Q).
8 or 9, DEL, or CTRL/U	Invalid expressions that cancel the current command. ODT then awaits a new command. The decimal values 8 and 9 are not valid characters and thus, when entered, cause XDT to ignore the current command.
B	Command that removes all breakpoints. Breakpoint can be in drivers, privileged tasks, and other system-level code as well as in the Executive itself.

Table 2-4 (Cont.): XDT Operators and Commands

Format	Meaning
nB	Command that removes the nth breakpoint. Breakpoint can be in drivers, privileged tasks, and other system-level code, as well as in the Executive itself.
a;nB	Command that sets breakpoint n at address a. Breakpoint can be in drivers, privileged tasks, and other system-level code as well as in the Executive itself. If n is omitted, XDT assumes the lowest-numbered available sequential breakpoint.
C	Constant register indicator. Represents the contents of register \$C (constant register).
D	Command that accesses data space. After this command is issued, XDT interprets all references to locations as referring to data space.
G or aG	Command that begins system execution at the current location in the program counter, following these steps: <ul style="list-style-type: none">• Sets BPT instructions in or restores BPT instructions to all breakpoint locations• Restores the Processor Status Word (PSW)• Starts execution at the address specified by the program counter (register \$7) If G is preceded by a, the value a replaces the current program counter (\$7) contents before proceeding as described above.
I	Command that accesses instruction space. After this command is issued, XDT interprets all references to locations as referring to the instruction space of the task.
K	Command that, using the relocation register whose contents are equal to or closest to (but less than) the address of the currently open location, computes the physical distance (in bytes) between the address of the currently open location and the value contained in that relocation register. XDT displays this offset and stores the value in the quantity register (\$Q).
nK	Command that computes the physical distance (in bytes) between the address of the currently open or the last-opened location and the value contained in relocation register n. XDT displays this offset and stores the value in the quantity register (\$Q).
a;nK	Command that computes the physical distance (in bytes) between address a and the value contained in relocation register n. XDT displays this offset and stores the value in the quantity register (\$Q).

Table 2-4 (Cont.): XDT Operators and Commands

Format	Meaning
L or kL or a;L or a;kL	Command that lists all the word or byte locations between the address limits that are specified by the low memory limit register (\$L) and the high memory limit register (\$H). If L is preceded by k, the value k replaces the current contents of \$H before initiating the list operation. If L is preceded by a, the value a replaces the current contents of \$L before initiating the list operation. Note that XDT's primitive terminal interface code recognizes CTRL/S and CTRL/Q so that the output produced by this command may be easily controlled. In loadable XDT, typing CTRL/O cancels the list command and returns the XDT> prompt.
aO or a;kO	Command that calculates and displays the PC-relative offset and the 8-bit branch displacement from the currently open location to address a; or calculates and displays the PC-relative offset and the 8-bit branch displacement from the specified address a to the specified address k.
P or kP	Command that causes the system to proceed with execution from the current breakpoint location and to stop when the next breakpoint location is encountered or when the next trap occurs, if any. If k is specified, XDT proceeds with program execution from the current location and stops at the breakpoint only after encountering it the number of times specified by integer k.
Q	Quantity register indicator. Represents the contents of register \$Q (quantity register).
R	Command that sets all relocation registers to the highest address value, 177777 ₈ , so they cannot be used in forming addresses.
nR	Command that sets relocation register n to the highest address value, 177777 ₈ , so it cannot be used in forming addresses.
a;nR	Command that sets relocation register n to address value a. If n is omitted, XDT assumes relocation register 0.
S or nS	Command that executes one instruction and displays the address of the next instruction to be executed. If n is specified, XDT executes n instructions and displays the address of the next instruction to be executed.
X	Command that exits from the Executive to the system crash dump routine.

Chapter 3

Error Detection

The Executive Debugging Tool (XDT) responds to errors in user input and to certain hardware-detected errors that occur during task execution. This chapter describes these errors, XDT's response to them, and what action the user can take to correct them.

3.1 Input Errors

XDT uses the question mark (?) to indicate that it has detected an error in user input. After displaying the question mark, XDT generates a carriage return and a line feed, and then XDT prompts for another command.

XDT responds with the question mark to any of the following input errors:

- Reference to an address without an operator
- Reference to an address that is not mapped
- Reference to a nonexistent register—for example, \$20
- Input of an illegal character—for example, 8 or 9

If you have typed an incorrect input string—for example, contradictory arguments for the W command—you may find that the simplest course of action is to cancel the input string by typing an illegal character. You cannot, however, erase a string once you have entered the command—the character W, in this case.

XDT does not tell you what error has caused it to display the question mark. However, an error sometimes causes XDT to return one of the error codes listed in Section 3.2, plus information on the location at which the error occurred.

In some cases (for example, if you attempt a memory operation when \$L is greater than \$H), XDT repeats its prompt but does not display a question mark.

3.2 Task Image Error Codes

Eight synchronous system trap (SST) vector registers are used to contain pointers to error-handling routines. Upon detecting an error condition, XDT activates the appropriate routine and displays an error message. This message has the form *cc:k*, where *cc* is a 2-character error code and *k* is the location at which the error occurred. XDT displays the location as a relative address if there is a relocation register containing a base address less than the absolute address of the location.

The following examples are error messages from a debugging session:

MP:007414

OD:1,003507

The remainder of this chapter is an alphabetic list of error codes. Each error code is followed by an explanation and a description of what action the user should take in response to the error.

- BE** **Explanation:** Breakpoint instruction executed at unexpected location. The address of the breakpoint instruction does not match the contents of any register, \$0B to \$7B.
User Action: Examine your code to determine why the unexpected breakpoint occurred; then, continue with the P command.
- EM** **Explanation:** Invalid EMT instruction executed. Only EMT 377 and EMT 376 (for a privileged task) are allowed by the Executive for execution of Executive directives. Normally, vector address 30 is used for this trap sequence.
User Action: If you want to use an EMT trap handler that you have written, set SST vector register 5 (\$5V) to the appropriate vector address.
- FP** **Explanation:** Floating-point instruction error. One of the following has occurred: division by zero; illegal Floating Op Code; flotation overflow or underflow; or conversion failure.
User Action: Check your code for sequences that may have caused one of these conditions.
- IL** **Explanation:** Reserved or illegal instruction executed. The task tried to execute a nonexistent instruction or an EIS or FPP instruction in a system with no EIS or FPP hardware.
User Action: Check your code for typographical errors or the use of a nonexistent instruction.
- IO** **Explanation:** IOT instruction executed. Normally, vector address 20 is used for this trap sequence.
User Action: To change the handling of I/O traps, set SST vector register 3 (\$3V) to the appropriate vector address.
- MP** **Explanation:** Memory-protection violation or illegal memory reference. The task tried to access a location outside of the ranges mapped, or the task tried to access a location that it did not have the privilege to access.
User Action: Check your code for typographical or programming errors that could lead to this condition.

- OD** **Explanation:** Odd address reference on word instruction. The PC contained an odd address when trying to access a word in memory. Also, on some processors, indicates execution of an illegal instruction.
User Action: Check your code for the use of a word instruction when a byte instruction was intended (MOV instead of MOVB, for example), or check for a typographical error in the address specification.
- TE** **Explanation:** T-bit exception. The T-bit was set by some mechanism other than a breakpoint or an S or P command. This error can occur if bit 4 is set in a word that is interpreted as the PSW due to its position on the stack.
User Action: Check that the stack contains appropriate values.
- TR** **Explanation:** TRAP instruction executed. Normally, vector address 34 is used for this trap sequence.
User Action: To change the handling of TRAP instructions, set SST vector register 6 (\$6V) to the appropriate vector address.

Appendix A

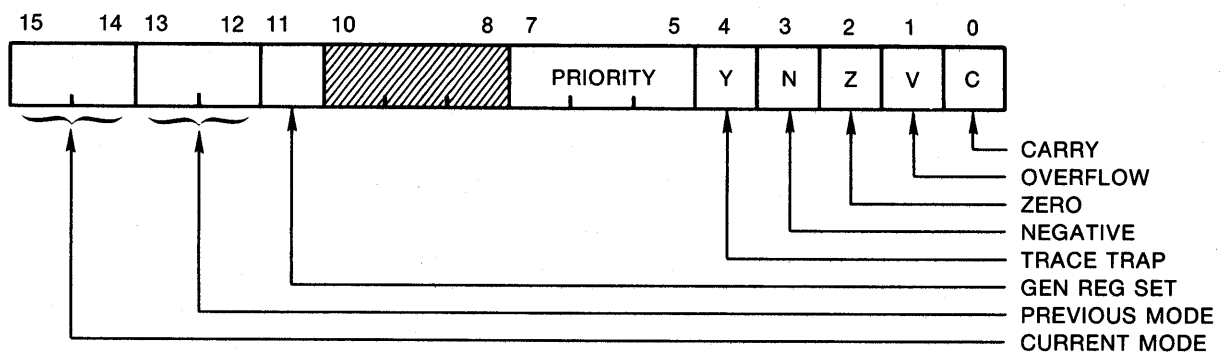
Processor Status Word

The Processor Status Word (PSW), stored at hardware location 17777776, contains information on the current status of the processor. The information contained in this location includes the following:

- The current and previous operational modes of the processor (mapped system only)
- The current processor priority
- An indicator that, when set, causes a trap upon completion of the current instruction
- Condition codes describing the results of the last instruction executed

The format of the PSW is shown in Figure A-1.

Figure A-1: Format of the Processor Status Word



ZK-491-81

Bits 15 and 14 indicate the current processor mode: user mode (11), supervisor mode (01), or kernel mode (00). Bits 13 and 12 indicate the previous mode; that is, the mode the machine was in (user, supervisor, or kernel) prior to the last interrupt or trap.

Bits 7 to 5 show the current priority of the central processor. The central processor operates at any one of eight levels of priority (0 to 7). When the central processor is operating at level 7 (the highest priority), an external device cannot interrupt it with a request for service. The central processor must be operating at a lower priority than the external device's request in order for the interrupt to take effect.

The trace (T) bit (bit 4) can be set or cleared under program control. When set, a processor trap will occur through location 14 upon completion of the current user instruction, and a new PSW will be loaded. The T-bit is especially useful in debugging programs because it provides an efficient means for stepping through the task one instruction at a time. ODT uses the T-bit to execute instructions when you are stepping through your program with the S command, as described in Chapter 2.

The condition codes N, Z, V, and C (bits 3 to 0, respectively) indicate the result of the last central processor operation. These bits are set as follows:

- N=1 If the result was negative
- Z=1 If the result was zero
- V=1 If the operation resulted in an arithmetic overflow
- C=1 If the operation resulted in a carry from the most significant bit

Appendix B

Executive Symbols Supported by Loadable XDT

Table B-1 lists the Executive symbols that XDT automatically searches for in an internal table. You use the symbol (#symbolname) the same way as you would use the "a" variable in an XDT command line. Specifying the symbol name eliminates having to look in the Executive map for the symbol's address. After locating the symbol, XDT displays the symbol's value. See Chapter 2 for more information on how to specify these symbols.

Table B-1: Executive Symbols Supported by Loadable XDT

Symbols from SYSXT—System entrance and exit routines						
\$CFORK	\$CKMAP	\$DBTRP	\$DIRSV	\$DIRXT	\$DSPKA	\$EXDOP
\$EXROP	\$EXRP1	\$FINBF	\$FINDI	\$FINXT	\$FORK	\$FORK1
\$FORK0	\$FORK2	\$GENBF	\$GGFRN	\$IFORK	\$INTSC	\$INTSE
\$INTSF	\$INTSI	\$INTSV	\$INTX1	\$INTXT	\$LSUPD	\$LSUP1
\$NONSI	\$QFORK	\$SGFIN	\$SWSTK	\$WPIN0	\$WPIN1	\$WPIN2
\$WPIN3	\$WPBR					
Symbols from LOWCR—Low core vector area						
\$BCERR	\$BCFAC	\$BCPC	\$BILNG	\$CPBIT	\$CPCRM	\$CSFSV
\$CSHSV	\$CURPR	\$CXDBL	\$FMAPP	\$HEADR	\$HFMSK	\$ICAVL
\$KXBAS	\$KXPTR	\$KXVC1	\$PROCN	\$RQSCH	\$SAHDB	\$SAHPT
\$SGFFR	\$SIRWF	\$STACK	\$STKDP	\$STRTM	\$SUPFL	\$TKTCB
\$UMPC	\$UMPS	\$UMRHD	\$UMRWT	\$UMR4	\$UMR5	\$XXLOW
\$XXHGH						

Table B-1 (Cont.): Executive Symbols Supported by Loadable XDT

Symbols from BFCTL—Buffer control routine						
\$BLXIO	\$GTBYT	\$GTWRD	\$PTBYT	\$PTWRD		
Symbols from CORAL—Core allocation and deallocation routines						
\$ALCLK	\$ALOCB	\$ALOC1	\$ALPKT	\$ALSEC	\$ALSC1	\$ALSPK
\$ALVBK	\$DCLKA	\$DEACB	\$DEAC1	\$DECLK	\$DEPKT	\$DESC1
\$DESEC	\$DESPK	\$PLTRQ				
Symbols from DRSUB—General executive subroutines						
\$CPALO	\$CPCON	\$CPDEA	\$CPSEN	\$DRCL2	\$DRCL3	\$DRQRQ
\$FINDR	\$IMASG	\$MPDC1	\$MPDC2	\$MPDC3	\$MPDC4	\$MPDCV
\$MPPRO						
Symbols from EXESB—General executive subroutines						
\$ACHCK	\$ACHKB	\$ACHKP	\$ACHKW	\$ACHRO	\$ACHUI	\$CEFI
\$CEFIG	\$CEFN	\$CEFNG	\$CKBFB	\$CKBFI	\$CKBFR	\$CKBFW
\$CVDVN	\$DRWSE	\$MPLUN	\$MPLND	\$MPLNE	\$SRGEF	\$TICLR
\$TKWSE						
Symbols from IOSUB—I/O subroutines						
\$DCWIO	\$DECAL	\$DECBF	\$DECIO	\$DECIP	\$DVMSG	\$DVMG1
\$GSPKT	\$GTPKT	\$INIBF	\$IOALT	\$IODON	\$IODSA	\$IOFIN
\$IOKIL	\$IOKL1	\$IOKL2	\$QPKRQ	\$QPKR1	\$QUEBF	\$REQUE
\$REQU1	\$SCDVT	\$SCDV1	\$TSTBF	\$ULDRQ		
Symbols from MDSUB—Mass storage device subroutines						
\$BLKCK	\$BLKC1	\$BLKC2	\$CKLBN	\$CRPAS	\$CVLBN	\$DEATR
\$DLNK	\$ECCOR	\$LCKPR	\$MPPKT	\$MPVBN	\$RQCNC	\$RQCND
\$RLCN	\$SHFND	\$SHFN1	\$SHSAV	\$VOLVD		

Table B-1 (Cont.): Executive Symbols Supported by Loadable XDT

Symbols from MEMAP—Memory mapping subroutines						
\$ASUMR	\$DEUMR	\$DQUMR	\$MPPHY	\$MPUBM	\$MPUB1	\$RELCD
\$RELOC	\$RELOM	\$RELOP	\$RELUI	\$STMAP	\$STMP1	\$SWACD
\$SWAC1	\$WTUMR					
Symbols from PLSUB—PLAS subroutines						
\$CKACC	\$CRATT	\$DELRG	\$DETRG	\$SRNAM	\$SRATT	\$SRWND
\$UNMAP						
Symbols from QUEUE—Queue manipulation routines						
\$CLINS	\$CLRMV	\$CLRSM	\$GTSPK	\$QCLIL	\$QCLNR	\$QCNTP
\$QCPKT	\$QINSB	\$QINSF	\$QINSP	\$QMCR	\$QMCR1	\$QRMVA
\$QRMVF	\$QRMVT	\$QSPIB	\$QSPIF	\$QSPIP	\$QSPRF	\$SCMDQ
\$SRCCQ	\$SRCQ1	\$SRUCB	\$SRUC1			
Symbols from REQSB—Request subroutine						
\$ABCTK	\$ABTSK	\$ACCRG	\$ACTRM	\$ACTTK	\$ALTRG	\$BILDS
\$CALTA	\$CLSRF	\$DASTT	\$DCAST	\$DEARG	\$DQAC	\$DRTHR
\$ERREC	\$ERTHR	\$EXRQF	\$EXRQN	\$EXRQP	\$EXRQS	\$EXRQU
\$FNDSP	\$ICHPK	\$LDREG	\$LOADT	\$MAPTK	\$NXTSK	\$QASTC
\$QASTT	\$QUEXT	\$REMOV	\$REMO1	\$RLCPS	\$RLPAR	\$RLPR1
\$SETCR	\$SETF	\$SETFG	\$SETM	\$SETMG	\$SETRT	\$SETRQ
\$SRAST	\$SRMUT	\$SRPRO	\$SRSTD	\$STPCT	\$STPTK	\$TCBCP
\$TSKRP	\$TSKRQ	\$TSKRT	\$TSPAR	\$TSTCP	\$UISET	
Symbols from SSTSR—Synchronous system trap (SST) routines						
\$EMSST	\$FPPRQ	\$FPPR7	\$FPPR8	\$ILINS	\$IOTRP	\$SGFLT
\$TRACE	\$TRP04					

Table B-1 (Cont.): Executive Symbols Supported by Loadable XDT

Symbols from SYSXT—System entrance and exit routines						
\$CFORK	\$CKMAP	\$DBTRP	\$DIRSV	\$DIRXT	\$EXDOP	\$EXROP
\$EXRP1	\$FINBF	\$FINDI	\$FINXT	\$FORK	\$FORK1	\$FORK0
\$FORK2	\$GENBF	\$GGFRN	\$IFORK	\$INTSC	\$INTSE	\$INTSF
\$INTSI	\$INTSV	\$INTX1	\$INTXT	\$LSUPD	\$LSUP1	\$NONSI
\$QFORK	\$SGFIN	\$SWSTK	\$WPIN0	\$WPIN1	\$WPIN2	\$WPIN3
\$WPBR	\$DSPKA					
Symbols from SCBDF\$—Statistics Control Block (SCB) offset definitions						
S.CTM	S.EMB	S.FRK	S.KS5	S.ITM	S.KRB	S.KTB
S.LHD	S.PKT	S.RCNT	S.ROFF	S.STS	S.ST2	S.ST3
S.URM						
Symbols from TCBDF\$—Variable Task Control Block (TCB) offset definitions						
T.ACN	T.CPU	T.CTX	T.IRM	T.ISIZ	T.OCBH	T.RDCT
T.RRM	T.SAST					
Symbols from UCBDF\$—Variable Unit Control Block (UCB) offset definitions						
U.CTX	U.FPRO	U.UAB	U.LOG			
Symbols from PCBDF\$—Attachment descriptor block offset definitions						
A.IOC	A.MPCT	A.PCB	A.PCBL	A.PRI	A.STAT	A.TCB
A.TCBL						
Symbols from CTBDF\$—Controller Table Block (CTB) offset definitions						
L.CLK	L.DCB	L.ICB	L.KRB	L.LNK	L.NAM	L.NUM
L.STS						
Symbols from DCBDF\$—Device Control Block (DCB) offset definitions						
D.DSP	D.LNK	D.MSK	D.NAM	D.PCB	D.VCAN	D.VCHK
D.VDEB	D.VKRB	D.VNXC	D.VTIN	D.VTOU	D.VINI	D.VOUT
D.VPWF	D.VUCB	D.UCB	D.UCBL	D.UNIT		

Table B-1 (Cont.): Executive Symbols Supported by Loadable XDT

Symbols from KRBDF\$—Controller Request Block (KRB) offset definitions						
K.CON	K.CRQ	K.CSR	K.FRK	K.HPU	K.IOC	K.OFF
K.OWN	K.PRI	K.PRM	K.STS	K.URM	K.VCT	
Symbols from PCBDF\$—Partition Control Block (PCB) offset definitions						
P.ATT	P.BLKS	P.CBDL	P.CSBA	P.DPCB	P.HDLN	P.HDR
P.IOC	P.LNK	P.MAIN	P.NAM	P.OWN	P.PRI	P.PRO
P.REL	P.RMCT	P.RRM	P.SIZE	P.STAT	P.ST2	P.SWSZ
P.SUB	P.TCB	P.WAIT				
Symbols from SCBDF\$—Mapping assignment block definitions						
M.BFVH	M.BFVL	M.LNK	M.UMRA	M.UMRN	M.UMVH	M.UMVL
Symbols from TCBDF\$—Fixed Task Control Block (TCB) offset definitions						
T.ACTL	T.ASTL	T.ATT	T.DPRI	T.EFLG	T.EFLM	T.GGF
T.HDLN	T.IID	T.IOC	T.LBN	T.LDV	T.LNK	T.MXSZ
T.NAM	T.OFF	T.PCB	T.PCBV	T.PRI	T.RCVL	T.RRFL
T.SRCT	T.STAT	T.ST2	T.ST3	T.ST4	T.TCBL	T.TKSZ
T.TIO	T.UCB					
Symbols from UCBDF\$—Fixed Unit Control Block (UCB) offset definitions						
U.MUP	U.LUIC	U.OWN	U.DCB	U.RED	U.CTL	U.STS
U.UNIT	U.ST2	U.CW1	U.CW2	U.CW3	U.CW4	U.SCB
U.ATT	U.BUF	U.CNT	U.UCBX	U.ACP	U.VCB	U.CBF
U.UMB	U.PRM					

Index

A

Absolute location, 2-11
Address
 relocatable, 2-13
American Standard Code for Information
 Interchange
 See ASCII
A register, 2-11
Argument separator, 2-10
ASCII
 operator
 byte mode, 2-12
 word mode, 2-12
At sign command (@), 2-11
a variable, 2-10

B

B command, 2-12
BPT trap instruction, 1-8
Branch location, 2-11
Breakpoint, 1-9, 2-14
 address register, 2-12
 inserting with OPEN command, 1-9
 inserting with ZAP utility, 1-11
 instruction register, 2-12
 proceed count register, 2-12
 removing, 2-12
 setting with XDT, 2-2
B register, 2-12
BRK command, 1-9
Bugcheck, 2-5
Byte mode
 operator
 ASCII, 2-12
 octal, 2-12

C

Circumflex command (^), 2-11
Command
 at sign (@), 2-11
 B, 2-12
 circumflex (^), 2-11
 D, 2-13
 equal sign (=), 2-12
 G, 1-8, 2-13
 I, 2-13
 K, 2-13
 L, 2-14
 left angle bracket (<), 2-11
 LINE FEED, 2-10
 O, 2-14
 P, 1-9, 2-14
 R, 2-14
 RETURN, 2-10
 right angle bracket (>), 2-11
 S, 2-2, 2-14
 underscore (_), 2-11
 variables
 a, 2-10
 k, 2-10
 n, 2-10
 #symbol, 2-10
 list, B-1
 x, 2-10
 X, 2-1, 2-14
C register, 2-12
 indicator, 2-13
CTRL/O
 XDT, 2-14
Current location indicator (.), 2-10

D

Data space, 2-4

Data space (cont'd.)

command, 2-13

D command, 2-13

Dollar sign (\$), 2-11

Driver

debugging, 2-2

E

EMT trap instruction, 1-8, 2-5

Equal sign command (=), 2-12

Error

codes, 2-6, 3-2

detection, 3-1

facility codes, 2-6

task image, 3-2

T-bit, 2-5

Exit command, 2-14

Expression

invalid, 2-12

register address, 2-11

F

F register, 2-12

G

G command, 1-8, 2-13

G register, 2-12

H

H register, 2-11

I

I command, 2-13

Instruction space, 2-4

command, 2-13

Interrupt processing, 1-4

IOT trap instruction, 1-8

I register, 2-12

K

K command, 2-13

k variable, 2-10

L

L command, 2-14

Left angle bracket command (<), 2-11

LINE FEED command, 2-10

Location

absolute, 2-11

L register, 2-11

M

Memory

examining memory location, 2-3

Mode

user, 2-2

N

n variable, 2-10

O

O command, 2-14

Octal operator

byte mode, 2-12

word mode, 2-12

Offset

calculating, 2-14

PC-relative, 2-14

OPEN command, 1-9

Operator

apostrophe ('), 2-12

backslash (\), 2-12

byte mode

ASCII, 2-12

octal, 2-12

minus sign (-), 2-10

percent sign (%), 2-12

plus sign (+), 2-10

quotation mark ("), 2-12

slash (/), 2-12

space, 2-10

word mode

ASCII, 2-12

octal, 2-12

Radix-50, 2-12

P

P command, 1-9, 2-14

PC-relative offset, 2-14

Processor clock, 1-11

turning off, 2-4

Processor states, 1-3

priority, 1-4

Processor Status Word

See PSW

Processor traps

XDT, 1-8

PSW, A-1

format, A-1

Q

- Q register, 2-12
 - indicator, 2-14
- Question mark (?)
 - user input error, 3-1

R

- Radix-50
 - operator
 - word mode, 2-12
 - separator (*), 2-10
- R command, 2-14
- Register
 - A, 2-11
 - B, 2-12
 - breakpoint
 - address, 2-12
 - instruction, 2-12
 - proceed count, 2-12
 - C, 2-12
 - indicator, 2-13
 - F, 2-12
 - G, 2-12
 - general, 2-11
 - H, 2-11
 - I, 2-12
 - L, 2-11
 - Q, 2-12
 - indicator, 2-14
 - R, 2-12
 - clearing, 2-14
 - setting, 2-14
 - S, 2-11
 - XDT internal, 2-11
- Register indicator
 - C register, 2-13
 - current location (.), 2-10
 - Q register, 2-14
- Relative branch location, 2-11
- Relocatable address, 2-13
- RETURN command, 2-10
- Right angle bracket command (>), 2-11
- R register, 2-12
 - clearing, 2-14
 - setting, 2-14

S

- S command, 2-2, 2-14
- Separator
 - argument (,), 2-10
 - argument (;), 2-10

- Separator (cont'd.)
 - Radix-50 (*), 2-10
- Space operator
 - See Operator
- S register, 2-11
- SST, 1-5
- Stack Depth Indicator
 - See \$STKDP
- \$STKDP, 1-4, 1-8, 2-2
- #symbol variable, 2-10
 - list, B-1
- Synchronous System Trap
 - See SST
- System Trap, 1-5

T

- Task
 - execution
 - beginning, 2-13
 - fixed, 2-2
 - privileged, 2-2
 - debugging with ZAP, 1-11
- Trap, 3-2
 - entry codes, 1-6
 - instructions
 - BPT, 1-8
 - EMT, 1-8, 2-5
 - IOT, 1-8
 - TRAP, 1-8
- TRAP trap instruction, 1-8

U

- Underscore command (_), 2-11

V

- Variable, 2-10
 - #symbol
 - list, B-1

W

- Word mode
 - operator
 - ASCII, 2-12
 - octal, 2-12
 - Radix-50, 2-12

X

- X command, 2-1, 2-14
- x variable, 2-10

Z

ZAP utility

- debugging privileged task, 1-11
- inserting breakpoint, 1-11
- modifying system image, 1-11

**READER'S
COMMENTS**

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

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.

What kind of user are you? Programmer Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

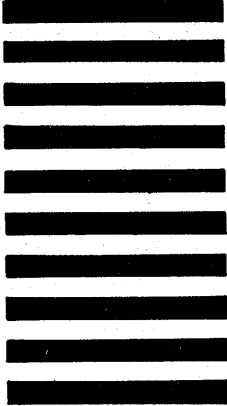
City _____ State _____ Zip Code _____
or Country

----- Do Not Tear - Fold Here and Tape -----

digitalTM



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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987



----- Do Not Tear - Fold Here -----