RSX–11M/M–PLUS and Micro/RSX

Debugging Reference Manual

Order No. AA-EZ50A-TC

digital software

RSX-11M/M-PLUS and Micro/RSX

Debugging Reference Manual

Order No. AA-EZ50A-TC

RSX-11M Version 4.2
RSX-11M-PLUS Version 3.0
Micro/RSX Version 3.0

digital equipment corporation · maynard, massachusetts

First Printing, July 1985

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 (c) 1985 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	DIBOL	PDT
DEC/CMS	EduSystem	RSTS
DEC/MMS	IAS	RSX
DECnet	MASSBUS	UNIBUS
DECsystem-10	MicroPDP-11	VAX
DECSYSTEM-20	Micro/RSTS	VMS
DECUS	Micro/RSX	VT
DECwriter	PDP	digital

ZK2564

HOW TO ORDER ADDITIONAL DOCUMENTATION

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

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

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6215 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

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

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

DIRECT MAIL ORDERS (CANADA)

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

DIRECT MAIL ORDERS (INTERNATIONAL)

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

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

CONTENTS

		Page
PREFACE		vii
CHAPTER 1	INTRODUCTION TO ODT	
1.1	OVERVIEW OF ODT	1-1
1.2	LINKING ODT WITH A USER PROGRAM	1-2
1.2.1	Linking ODT from MCR	1-2
1.2.2	Linking ODT from DCL	1-3
1.2.3	Linking to Enable Instruction and Data Space Features (RSX-11M-PLUS and Micro/RSX)	1-3
1.2.3.1	Enabling Instruction and Data Space	1-3
1.2.3.2	Linking ODTID.OBJ Explicitly	1-4
1.2.3.3	Enabling Supervisor-Mode Library Debugging	1-4
1.3	INVOKING ODT	1-4
1.4	RETURNING CONTROL TO THE HOST SYSTEM	1-4
1.5	INTERRUPTING A DEBUGGING SESSION	1-5
1.5.1	Resuming a Debugging Session (RSX-11M-PLUS and Micro/RSX)	1-5
CHAPTER 2	ODT CHARACTERS AND SYMBOLS	
2.1	VARIABLES USED IN COMMAND DESCRIPTIONS	2-1
2.2	ADDRESS EXPRESSION FORMATS	2-2
2.2.1	Absolute and Relative Addressing	2-2
2.2.2	Forming Expressions	2-3
2.3	OPERATOR AND COMMAND SUMMARY	2-4
CHAPTER 3	CONTROLLING PROGRAM EXECUTION WITH ODT	
3.1	SETTING AND REMOVING BREAKPOINTS	3-1
3.1.1	Setting Breakpoints	3-1
3.1.2	Removing Breakpoints	3-2
3.2	BEGINNING PROGRAM EXECUTION	3-2
3.3	CONTINUING TASK EXECUTION	3-3
3.4	USING THE BREAKPOINT PROCEED COUNT	3-4
3.5	STEPPING THROUGH THE PROGRAM	3-4
CHAPTER 4	DISPLAYING AND ALTERING THE CONTENTS OF LOCATIONS WITH ODT	
4.1	ALTERING THE CONTENTS OF A LOCATION	4-1
4.2	CLOSING A LOCATION	4-2
4.3	OPENING WORD AND BYTE LOCATIONS	4-2
4.3.1	Opening Word and Byte Locations Specifying an Address	4-2
4.3.2	Reopening the Location Last Opened	4-2
4.3.3	Moving Between Word and Byte Modes	4-3
4.4	OPENING THE NEXT SEQUENTIAL LOCATION	4-3
4.5	OPENING THE PRECEDING LOCATION	4-4
4.6	OPENING ABSOLUTE LOCATIONS	4-4
4.7	OPENING PC-RELATIVE LOCATIONS	4-4
4.8	OPENING RELATIVE BRANCH OFFSET LOCATIONS	4-5
4.9	RETURNING FROM A CALCULATED LOCATION	4-5
4.10	USING DIFFERENT OUTPUT MODES	4-6

4.10.1 ASCII Mode 4-6
4.10.2 Radix-50 Mode 4-7

CHAPTER 5 USING REGISTERS IN ODT

5.1 GENERAL REGISTERS 5-1
5.1.1 Examining and Setting General Registers 5-1
5.1.2 Contents of General Registers 5-2
5.2 ODT INTERNAL REGISTERS 5-2
5.2.1 Relocation Registers 5-5
5.2.1.1 Setting Relocation Registers 5-6
5.2.1.2 Clearing Relocation Registers 5-6
5.2.2 The Reentry Vector Register 5-6

CHAPTER 6 MEMORY OPERATIONS IN ODT

6.1 REGISTERS USED IN MEMORY OPERATIONS 6-1
6.1.1 Search Limit Registers 6-1
6.1.2 Search Mask Register 6-2
6.1.3 Search Argument Register 6-2
6.1.4 Device Control LUN Registers 6-2
6.2 SEARCHING MEMORY 6-2
6.2.1 Searching for a Word or Byte 6-3
6.2.2 Searching for Inequality of a Word or Byte . . . 6-3
6.2.3 Searching for a Reference 6-3
6.3 FILLING MEMORY 6-4
6.4 LISTING MEMORY 6-4
6.4.1 Command Format 6-4
6.4.2 Listing Format 6-5

CHAPTER 7 PERFORMING CALCULATIONS IN ODT

7.1 CALCULATING RELOCATABLE ADDRESSES 7-1
7.2 CALCULATING OFFSETS 7-1
7.3 EVALUATING EXPRESSIONS 7-2
7.3.1 Equal Sign Operator 7-2
7.3.2 Current Location Indicator 7-3
7.3.3 Constant Register Indicator 7-3
7.3.4 Quantity Register Indicator 7-3
7.3.5 Radix-50 Evaluation 7-4

CHAPTER 8 THE EXECUTIVE DEBUGGING TOOL (XDT)

8.1 THE ADVANTAGE OF XDT 8-1
8.2 HOW TO INCLUDE XDT IN YOUR RSX-11M OR RSX-11M-PLUS SYSTEM 8-1
8.3 LOADABLE XDT ON MICRO/RSX AND PRE-GENERATED RSX-11M-PLUS SYSTEMS 8-2
8.4 PROCESSOR STATES 8-2
8.4.1 The Stack Depth Indicator and Interrupt Processing 8-3
8.5 ENTERING XDT 8-4
8.5.1 XDT and Synchronous System Traps (SSTs) 8-4
8.5.1.1 Processor Traps and System Crashes 8-5
8.5.2 Entering XDT from a Virgin System Boot 8-6
8.5.3 Entering XDT Using the BRK Command 8-7
8.5.4 Entering XDT Using the BPT Instruction 8-7
8.5.4.1 Inserting a BPT Instruction Using the OPEN Command 8-7
8.5.4.2 Inserting a BPT Instruction Using the ZAP Utility 8-8

8.5.5 Entering XDT When the System Is Hung (RSX-11M and RSX-11M-PLUS) 8-9

CHAPTER 9 DEBUGGING WITH XDT

9.1 DEBUGGING WITH XDT 9-1
9.1.1 Using XDT to Debug the Executive 9-1
9.1.2 Using XDT to Debug Privileged Tasks 9-1
9.1.3 Using XDT to Debug a Driver 9-2
9.1.4 Using XDT to Examine a Memory Location 9-3
9.1.5 Turning Off the Processor Clock 9-4
9.1.6 T-bit Error 9-4
9.2 INTERPRETING BUG CHECKS 9-4
9.3 XDT COMMANDS AND OPERATORS 9-7

CHAPTER 10 ADDITIONAL DEBUGGING AIDS

10.1 ACCESSING OTHER DEBUGGING AIDS 10-1
10.1.1 MCR Command Line 10-1
10.1.2 DCL Command Line 10-1
10.2 THE TRACE DEBUGGING PROGRAM 10-2
10.2.1 The Trace Listing 10-2
10.2.2 Bias Values and Ranges 10-3
10.2.2.1 Specifying a Bias Value 10-3
10.2.2.2 Specifying Ranges to be Traced 10-3

APPENDIX A ERROR DETECTION

A.1 INPUT ERRORS A-1
A.2 TASK IMAGE ERROR CODES A-2

APPENDIX B PROCESSOR STATUS WORD

EXAMPLES

6-1 ODT Listing Format 6-5
10-1 Sample Trace Output 10-2

FIGURES

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

TABLES

2-1 Variables Used in ODT Command Descriptions 2-1
2-2 Forms of Address Expressions 2-4
2-3 ODT Operators and Commands 2-4
5-1 ODT Single Registers 5-3
5-2 ODT Register Sets 5-4
7-1 Numeric Equivalents of Radix-50 Characters 7-4
8-1 XDT Trap Entry Codes 8-5
9-1 Common Facility-Independent Error Code Definitions 9-6
9-2 Standard Bugcheck Format Facility Code Definitions 9-7
9-3 XDT Operators and Commands 9-8

PREFACE

MANUAL OBJECTIVES

This manual describes the On-Line Debugging Tool (ODT), used to debug user task images, and the Executive Debugging Tool (XDT), used to debug privileged tasks on RSX11-M, RSX-11M-PLUS, and Micro/RSX systems. It provides reference information on all ODT and 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, RSX-11M-PLUS, or Micro/RSX operating systems. Readers should understand the user interface of the operating system that they are using. RSX-11M and RSX-11M-PLUS users should be familiar with the contents of the RSX-11M/M-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 MANUAL

This manual is divided into ten chapters and two appendixes, as follows:

Chapter 1, Introduction to ODT, gives an overview of ODT. It explains how to link the debugger into a user task image and how to begin and end a debugging session.

Chapter 2, ODT Characters and Symbols, explains the special symbols used in ODT and includes a reference table of all ODT commands, in alphabetical order. New ODT users should read Chapters 3 through 7 for explanations of the commands before studying the table of commands in detail. Experienced ODT users can use Chapter 2 for quick reference.

Chapter 3, Controlling Program Execution, describes the command used to begin program execution, to stop execution at breakpoints, and to continue execution after breakpoints. It also explains how to execute a program one or more instructions at a time.

Chapter 4, Displaying and Altering the Contents of Locations, explains how to open and close task locations, how to change the contents of locations, and how to display the contents of locations in different modes.

Chapter 5, Using Registers, describes all of the registers used by ODT. It includes reference tables as well as explanations of how registers are set and cleared. Experienced ODT users may wish to consult the tables in this chapter, as well as those in Chapter 2, for quick reference regarding specific registers.

Chapter 6, Memory Operations, describes ODT's memory search, fill, and list capabilities.

Chapter 7, Performing Calculations, describes how to use ODT to perform arithmetic calculations.

Chapter 8, The Executive Debugging Tool, gives information about XDT and describes how to enter XDT.

Chapter 9, Debugging with XDT, describes XDT commands and operators and explains the differences between ODT and XDT.

Chapter 10, Additional Debugging Aids, explains how to link debuggers other than ODT into a user task image. It describes the Trace program, a debugging aid that can be used in conjunction with ODT.

Appendix A, Error Detection, describes how ODT and XDT respond to errors in user input or program logic. It lists all ODT and XDT error message codes in alphabetical order.

Appendix B, Processor Status Word, shows the format of the PS and summarizes the functions of its bits.

ASSOCIATED MANUALS

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

CONVENTIONS USED IN THIS MANUAL

Throughout this book symbols and other notation conventions are used to represent keyboard characters, to convey textual information, and to aid the reader's understanding of material. These symbols and conventions are explained below.

Convention	Meaning
(XXX)	A symbol with a 1- to 3-character abbreviation, such as (LF) or (RET), indicates that you press a key on the terminal.
CTRL/a	This phrase indicates that you press the key labeled CTRL while simultaneously pressing another key, such as C or Y. In examples, this control key sequence is shown as ^A, because that is how the system displays it on your terminal.

red ink	User input appears in red ink in the examples throughout this book. System responses appear in black ink.
shading	Gray shading denotes information specific to RSX-11M-PLUS and Micro/RSX.
	Pink shading denotes information specific to RSX-11M.

SUMMARY OF TECHNICAL CHANGES

This manual combines the IAS/RSX-11 ODT Reference Manual and information about XDT. Information specific to IAS has been deleted.

The RSX-11M, 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. XDT on RSX-11M-PLUS systems and loadable XDT use bug checks to report this type of error. This manual includes information on interpreting bug checks.

This manual also corrects small technical errors in the text and examples of the previous version. It represents a significant reorganization of material, intended to make information more accessible to readers.

CHAPTER 1

INTRODUCTION TO ODT

This chapter gives an overview of the On-Line Debugging Tool (ODT). ODT is a utility for debugging task images. You can use ODT to:

- Control program execution
- Display the contents of locations or registers
- Alter the contents of locations or registers
- Search and fill memory
- Perform calculations

ODT commands consist of one character; some commands take a numeric or alphabetic character as an argument. All ODT commands, and the symbols that are used in them, are listed in Chapter 2. Chapters 3 through 7 describe how to use the commands.

This chapter describes how to link the debugger into a user task image, initiate a debugging session, and end a debugging session.

1.1 OVERVIEW OF ODT

ODT is special code that you link into your task image to help you debug your program. When you run a task into which ODT has been linked, the debugger receives control of the task automatically upon task initiation. Through ODT, you can then execute your task gradually, setting breakpoints at selected locations or stepping through the program one instruction at a time. Chapter 3 describes ODT commands for controlling program execution.

You can examine any location in your program -- instruction or data, word or byte -- by "opening" the location with ODT. While the location is open, you can immediately change the contents. You can move forward or backward to examine and modify other locations. Thus, you can test any number of modifications without rebuilding your task. Chapter 4 describes ODT commands for examining and altering locations and for moving from one location to another.

ODT operates through the use of a number of registers, all of which you can set and reset. Some of these registers are used to store information about your program while ODT has control. Eight registers can be set to the locations of breakpoints. Eight can be set to relocation biases -- the absolute base addresses of relocated object modules. You can use other registers to store values that you may want to use repeatedly during your debugging session. Chapter 5

describes the ODT registers. You can use ODT to search for bit patterns in memory, to fill blocks of memory with a value, or to list blocks of memory on an output device. Chapter 6 describes these operations.

During a debugging session, you can perform a variety of calculations: determining offsets, evaluating arithmetic expressions, and constructing Radix-50 words. These calculations are described in Chapter 7.

1.2 LINKING ODT WITH A USER PROGRAM

ODT is provided on your system as an object module, LB:[1,1]ODT.OBJ. The version of ODT supporting the instruction and data space features of RSX-11M-PLUS and Micro/RSX is provided in the object module LB:[1,1]ODTID.OBJ. To use ODT, you must link the appropriate object module with the object module(s) of your program. When the resulting task image is run, ODT is invoked and initiated automatically.

If the task image is overlaid, ODT is linked into the root segment so that the debugger will always be available.

The following sections describe how to link ODT into a task image in different environments. Section 1.2.1 describes how to link ODT if your command line interpreter (CLI) is MCR. Section 1.2.2 describes how to link ODT if your command line interpreter is DCL. Section 1.2.3 describes how to enable the instruction and data space and supervisor-mode features of ODT used under some RSX-11M-PLUS and Micro/RSX systems.

The information in subsequent sections on initiating and using ODT applies equally to RSX-11M, RSX-11M-PLUS, and Micro/RSX environments, except as noted.

1.2.1 Linking ODT from MCR

To link ODT with your program when your CLI is the Monitor Console Routine (MCR), first invoke the Task Builder by typing TKB in response to the MCR prompt. The Task Builder replies with its own prompt, TKB>. In response to this prompt, enter a Task Builder command specifying the name of the file(s) to be linked. Include the /DA switch, which indicates that a debugger (in this case ODT, the default) should be linked into the image. The following example shows the resulting command line:

```
TKB>MYTASK/DA=MYFILE1,MYFILE2
```

The Task Builder accesses the file ODT.OBJ in UFD [1,1] on the library device and links it with the files MYFILE1.OBJ and MYFILE2.OBJ into the task MYTASK.

Using ODT requires that you consult an up-to-date map of your task. Therefore, you should in most cases request a new map when you build the task, as in the following command line:

```
TKB>MYTASK/DA,MYTASK/CR/-SP=MYFILE1,MYFILE2
```

For more information on using the Task Builder consult the RSX-11M/M-PLUS and Micro/RSX Task Builder Manual.

1.2.2 Linking ODT from DCL

To link ODT with your program(s) when your command line interpreter is the DIGITAL Command Language (DCL), use the /DEBUG qualifier with the LINK command. ODT requires that you consult an up-to-date map of your task. To obtain a current map of the image file produced, include the /MAP qualifier. The following example shows the resulting command line:

```
> LINK/MAP/DEBUG/TASK:MYTASK MYFILE1,MYFILE2
```

Object modules MYFILE1.OBJ and MYFILE2.OBJ are linked with object module ODT.OBJ in UFD [1,1] on the library device. The resulting task image is named MYTASK.TSK.

For further information on using DCL, consult the RSX-11M Command Language Manual, the RSX-11M-PLUS Command Language Manual or the Micro/RSX User's Guide, as appropriate to your system.

1.2.3 Linking to Enable Instruction and Data Space Features (RSX-11M-PLUS and Micro/RSX)

To use the separate instruction and data space capabilities found on some RSX-11M-PLUS and Micro/RSX systems, you must link your program with the object module LB:[1,1]ODTID.OBJ instead of ODT.OBJ. Section 1.2.3.1 describes the MCR and DCL command lines that link this object module for tasks that have been built using separate instruction and data space. Section 1.2.3.2 describes the command line that links this object module explicitly. You use this command line if, for example, you want to use data space windows, but did not build the task using separate instruction and data space. Section 1.2.3.3 describes how to enable debugging for supervisor-mode libraries.

1.2.3.1 **Enabling Instruction and Data Space** - Separate instruction and data space is a feature of RSX-11M-PLUS and Micro/RSX systems only. ODT has the following instruction and data space commands: D, I, U, and Z. To enable these commands, you must link LB:[1,1]ODTID.OBJ with your program instead of ODT.OBJ. (See Table 2-3 for a description of these commands.)

If your command line interpreter is MCR and your task was built using separate instruction and data space, you enable these commands by adding the /ID switch, as well as the /DA switch, to the TKB command line. The following example shows the resulting command line:

```
TKB>MYTASK/DA/ID=MYTASK
```

You can add other switches to the command line as desired. Consult the RSX-11M/M-PLUS and Micro/RSX Task Builder Manual for information on TKB command lines.

If your command line interpreter is DCL and your task was built using separate instruction and data space, you enable these commands by using the qualifier /CODE:DATA_SPACE, as well as the /DEBUG qualifier, with the LINK command. The following example shows the resulting command line:

```
> LINK/DEBUG/CODE:DATA_SPACE MYTASK
```

You can add other qualifiers to the LINK command. See the RSX-11M Command Language Manual, RSX-11M-PLUS Command Language Manual or the Micro/RSX User's Guide for more information.

1.2.3.2 **Linking ODTID.OBJ Explicitly** - If your task was not built using separate instruction and data space, but you want to use data space windows, you must link ODTID.OBJ explicitly, specifying the debugger object module in the MCR or DCL command line. The following example shows the resulting MCR command line:

```
TKB>MYTASK=MYTASK,LB:[1,1]ODTID/DA
```

This example shows the resulting DCL command line:

```
> LINK /DEBUG:LB:[1,1]ODTID MYTASK
```

1.2.3.3 **Enabling Supervisor-Mode Library Debugging** - On RSX-11M-PLUS and Micro/RSX systems with separate instruction and data space, you can use ODT to debug supervisor-mode libraries. There are two ways to enable the Z command, which sets the current mode of ODT to supervisor mode. If your task was built using separate instruction and data space, link it as described in Section 1.2.3.1. If your task was not built using separate instruction and data space, link it specifying ODTID.OBJ explicitly, as described in Section 1.2.3.2.

To set breakpoints or write into the supervisor-mode libraries, you must install the library with READ/WRITE access or build the task as privilege: 0.

1.3 INVOKING ODT

Regardless of what operating system or command line interpreter you use, enter the RUN command, specifying the name of the task image file. ODT is invoked automatically when you run a task image into which ODT has been linked, as described in the previous sections.

ODT responds with a message indicating that it has been invoked and identifying the task image that it controls. On the next line, ODT displays its prompt, an underscore(_), indicating that it is ready to accept commands.

The following example shows how ODT is invoked when HIYA.TSK is run:

```
>RUN HIYA
ODT:TT15
_
```

In response to the ODT prompt, you can enter any ODT command. ODT commands are immediate-action commands; that is, ODT responds to the commands as soon as they are typed, without waiting for a line terminator. Therefore, commands cannot be corrected once they have been typed. You can, however, erase an incorrectly typed command argument by typing an illegal character or command (such as a non-octal number like 8 or 9) or by pressing CTRL/U or the DELETE key. In response, ODT discards your input line, displays a question mark (?), and prompts for another command.

Error detection is described in greater detail in Appendix A.

1.4 RETURNING CONTROL TO THE HOST SYSTEM

To return control from ODT to the host operating system, type X in response to the ODT prompt. This command causes execution of the system Task Exit directive, which terminates task execution.

1.5 INTERRUPTING A DEBUGGING SESSION

When you run a task linked with ODT, you can return to the command line interpreter prompt at any time by typing CTRL/C. Your task is still active. To stop execution of the task, enter the ABORT command in response to the MCR or DCL prompt. You cannot resume the aborted debugging session; you can only run your program again.

On RSX-11M-PLUS and Micro/RSX systems, you can interrupt task execution without aborting your task and then continue debugging. After typing CTRL/C use the commands described in the following section.

1.5.1 Resuming a Debugging Session (RSX-11M-PLUS and Micro/RSX)

RSX-11M-PLUS and Micro/RSX allow you to interrupt and then resume execution from the point at which the program was interrupted. To use this feature, do not enter the ABORT command. Instead, type the DEBUG command in response to the command line interpreter prompt. This command overrides the task's current status. Among other things, ODT unsets any WAIT-FOR-EVENT, STOP, or SUSPEND state that had been set. The DEBUG command also causes a T-bit (trace bit) exception, as described in Appendix A. ODT generates a TE error message, showing the current value of the program counter as the location where the error occurred. This message is followed by the ODT prompt (_).

The DEBUG command has the following format:

```
DEBUG [taskname]
```

The task name argument is the specification of the task to be interrupted, as used when the task was originally invoked. If you do not specify a task name, the default is a task initiated through the RUN command.

The following example shows how the DEBUG command is used:

```
>RUN HIYA
ODT:TT15
_G
^C
>DEBUG
TE:004020
_
```

The DEBUG command is especially useful if your program is caught in a loop, or if you need to stop execution before the next breakpoint.

CHAPTER 2

ODT CHARACTERS AND SYMBOLS

This chapter describes all the ODT operators and commands, and explains the meanings of ODT-specific symbols used in this manual. (Symbols and conventions common to the documentation set are listed in the Preface.)

2.1 VARIABLES USED IN COMMAND DESCRIPTIONS

Table 2-3 and the command descriptions in Chapters 3 through 7 use lowercase alphabetic variables to represent numeric and alphabetic arguments specified in commands. These symbols are explained in Table 2-1.

Table 2-1
Variables Used in ODT Command Descriptions

Variable	Meaning
a	An address expression representing the address of a task image location. The various forms in which an address expression can be specified are explained in Section 2.2.
k	An octal value up to 6 digits long with a maximum value of 177777(octal), or an expression representing such a value. An expression may include arithmetic operators or indicators, as described in Section 2.2.2. If more than 6 digits are specified, ODT truncates to the low-order 16 bits. If the octal value is preceded by a minus sign, ODT takes the two's complement of the value.
m	An octal value six digits long, used to represent a search mask.
n	An octal integer between 0 and 7.
x	An alphabetic character. A list of legal alphabetic characters is given in Table 2-3 where the variable x is used.

2.2 ADDRESS EXPRESSION FORMATS

An address expression, represented throughout this manual by the lowercase letter a, is an expression interpreted by ODT as a 16-bit (6-digit octal) value. You use an address expression to refer to a location in your task.

You can specify an address expression in either absolute or relative (relocatable) form, as described in Section 2.2.1. You can include in the address expression various operators and symbols, as described in Section 2.2.2.

2.2.1 Absolute and Relative Addressing

Each location has an absolute address assigned to it when the task is built. You can refer to the location using this 6-digit octal value. However, when the task is built again, with modules added or changed, this value may not refer to the same location. Therefore, it is often more convenient to refer to locations using relative (relocatable) addressing, which is less likely to be affected by subsequent task builds.

When you use relative addressing, you refer to a location not by its absolute value but by its position relative to a movable point. Usually, this movable point is the base (starting) address of the module to which the location belongs, because the distance between the base address and the addresses of locations within the module is easily determined from a task map or listing and is not likely to change without your knowledge. The movable point can, however, be any point that is convenient for debugging.

To use this form of addressing, you must first establish a simple means of referring to movable points through the use of ODT's relocation registers $0R through $7R. Each time you run a task built with ODT, consult a task map to determine the absolute addresses of convenient movable points. The map's memory allocation synopsis contains the base addresses of all the modules in the task. Follow the procedure described in Section 5.2.1.1 to set ODT's eight relocation registers to absolute addresses.

Once a relocation register is set, you can use the number of that register, 0 through 7, in forming relative addresses.

A relative address has the following form:

```
n,k
```

n

The number of a relocation register, 0 through 7, representing $0R through $7R.

, (comma)

A required separator between the two parts of the relative address.

k

The relative location, that is, the distance of the desired location from the value contained in register $nR. Usually, this is the location's position within the module whose base address is the value of the register.

Thus, relative address 0,100 refers to location 100 within the module whose base address is stored in ODT's relocation register $0R. Relative address 5,300 refers to location 300 within the module whose base address is stored in relocation register $5R.

Bias value refers to the value stored in a relocation register. It is a quantity equal to the distance (bias) between a relative location and its absolute address. Offset refers to the second part of a relative address. It is the distance of a relative location from the closest value (less than that location) stored in a relocation register. These terms are used throughout this manual.

2.2.2 Forming Expressions

An expression is a string of numbers, symbols, and operators that ODT interprets as a number. For example, 3+6 is an expression; ODT would interpret it as the octal value 11.

You can use an expression to represent an absolute address, a register containing a bias value, or an offset, as described in Section 2.2.1.

An expression used in an ODT session can contain any of the following elements:

- Octal numbers. ODT will not accept input containing an 8 or 9. It treats these as illegal characters and displays a question mark and a new prompt.
- The arithmetic operators a plus sign (+) or a space, indicating that values should be added, or a minus sign (-), indicating that the value that follows it should be subtracted from the value that precedes it.
- The unary operator minus sign (-), indicating that the value that follows it is negative and should be interpreted in two's complement form.
- ODT register indicators Q or C, representing $Q and $C registers, as described in Sections 7.3.4 and 7.3.3, respectively. When Q or C is used to represent a register containing a bias value, it must have a value in the range 0 through 7. When Q or C is used to represent an offset, it may contain any 16-bit value.
- The name of one of ODT's registers, used in the operations described in Chapters 5 and 6.
- The current location indicator (.), described in Section 7.3.2.

In evaluating expressions, ODT proceeds from left to right. It does not assign precedence to any operator or recognize parentheses to establish precedence. Therefore, you must be careful to form expressions so that they will be interpreted correctly. You can use the equal sign operator (=), described in Section 7.3.1, to determine the value of expressions before using them in ODT operations.

Table 2-2 shows how ODT interprets the various forms of address expressions. This table assumes a value of 003400 for relocation register 3 ($3R) and a value of 3 for the constant register ($C).

Table 2-2
Forms of Address Expressions

Address Expression Input	ODT Octal Interpretation
5	000005
-17	177761
3,150	003550
C	000003
C,10	003410
C,C+C	003406
3,C	003403
$3	Task general register 3

2.3 OPERATOR AND COMMAND SUMMARY

ODT commands are a combination of symbols and letters. Some commands have multiple forms.

Table 2-3 summarizes the ODT commands and operators, which are explained in detail in Chapters 3 through 7. The lowercase letters used in the command descriptions are explained in Table 2-1.

Table 2-3
ODT Operators and Commands

Format	Meaning
+ or space	Arithmetic operator used in expressions. Add the preceding argument to the following argument to form the current argument.
-	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.
*	Radix-50 separator used in constructing Radix-50 words (see Section 7.3.5).

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
.	Current location indicator. Causes the address of the last explicitly opened location to be used as the current address for ODT operations.
;	Argument separator. Separates multiple arguments, allowing an address expression or ODT register value to be identified.
(RET) or k (RET)	Command that closes the currently open location and prompts for the next command. If (RET) is preceded by k, the value k replaces the contents of the currently open location before it is closed.
(LF) or k (LF)	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 (LF) 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, opens the immediately preceding location 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 PC-relative offset and calculates the address of the next location to be opened; then closes the currently open location, and opens and displays the contents of the new location 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 thus evaluated. If @ is preceded by k, the value k replaces the contents of the currently open location before it is closed.

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
> 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, then closes the currently open location, and opens and displays the contents of the relative branch location 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 word location most recently opened by a /, LF, or ^. 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 through R7.
$x or $xn	Expression that represents the address of one of ODT's internal registers, where x is one of the following alphabetic characters, and n is one octal digit. Registers exist within ODT in the following order: S Processor Status register (hardware PS) W Directive Status Word (DSW) register for the user's task A Search argument register M Search mask register L Low memory limit register H High memory limit register C Constant register Q Quantity register F Format register X Reentry vector register

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
	nB Breakpoint address registers
	nG Breakpoint proceed count registers
	nI Breakpoint instruction registers
	nR Relocation registers
	nV SST vector registers
	nE SST (synchronous system trap) stack contents registers
	nD Device control LUN (logical unit number) registers
" or a"	Word mode ASCII operator. Interprets and displays the contents of the currently open (or the last previously opened) location as two ASCII characters, and stores this word in the quantity register ($Q). If " is preceded by a, the value a is taken as the address of the location to be interpreted and displayed.
' or a'	Byte mode ASCII operator. Interprets and displays the contents of the currently open (or the last previously opened) location as one ASCII character, and stores this byte in the quantity register ($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 last 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 a is taken as the address of a word location to be opened and displayed.

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
\ or a\	Byte mode octal operator. Displays the contents of the last byte location opened, and stores this octal byte in the quantity register ($Q). If \ is preceded by a, ODT takes the value a as the address of a byte location to be opened and displayed.
k=	Command that interprets and displays expression value k as six octal digits and stores this word in the quantity register ($Q).
8, 9, DELETE, or CTRL/U	Illegal expressions that cancel the current command. ODT then awaits a new command. The decimal values 8 and 9 are not legal characters and, thus, when entered, cause ODT to ignore the current command. The DELETE and CTRL/U functions are not operational in RSX-11M unless the terminal driver supports transparent read/write (a system generation option).
B	Command that removes all breakpoints from the user task.
nB	Command that removes the nth breakpoint from the user task.
a;nB	Command that sets breakpoint n in the user task at address a. If n is omitted, ODT assumes the lowest-numbered sequential breakpoint available.
C	Constant register indicator. Represents the contents of register $C (constant register).
D	Command that accesses data space. After this command is issued, ODT interprets all references to locations as referring to the D-space of the task (RSX-11M-PLUS and Micro/RSX only).
E or kE or m;E or m;kE	Command that searches memory between the address limits specified by the low memory limit register ($L) and the high memory limit register ($H). ODT examines these locations for references to the effective address specified in the search argument register ($A), as masked by the value specified in the search mask register ($M). (The mask should normally be set to 177777 for the E command.) Such references may be equal to, PC-relative to, or a branch

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
	displacement to the location specified in $A. If E is preceded by k, the value k replaces the current contents of $A before ODT initiates the search. If E is preceded by m, the current contents of $M are replaced with the value m before ODT initiates the search.
F or kF	Command that fills memory locations within the address limits specified by the low memory limit register ($L) and the high memory limit register ($H) with the contents of the search argument register ($A). If F is preceded by k, the value k replaces the current contents of $A before ODT initiates the fill operation.
G or aG	Command that begins task execution, following these steps: sets BPT instructions in or restores BPT instructions to all breakpoint locations in the task image; restores the Processor Status Word and user program registers; and starts execution at the address specified by the program counter (user register $7). If G is preceded by a, the value a replaces the current contents of $7 before proceeding as described above.
I	Command that accesses instruction space. After this command is issued, ODT interprets all references to locations as referring to the I-space of the task. (RSX-11M-PLUS and Micro/RSX only)
K	Command that, using the relocation register whose contents are equal to or closest to (but less than) the address of the currently open location, computes the physical distance (in bytes) between the address of the currently open location and the value contained in that relocation register. ODT displays this offset and stores the value in the quantity register ($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. ODT displays this offset and stores the value in the quantity register ($Q).

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
a;nK	Command that computes the physical distance (in bytes) between address a and the value contained in relocation register n. ODT displays this offset and stores the value in the quantity register ($Q).
L or kL or a;L or a;kL or n;a;kL	Command that lists all word or byte locations in the task between the address limits specified by the low memory limit register ($L) and the high memory limit register. 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 $L before initiating the list operation. If the value n is either zero or not specified, the display goes to your terminal (TI:). If a nonzero value is specified for n, the display goes to the console (CO:).
N or kN or m;N or m;kN	Command that searches memory between the address limits specified by the low memory limit register ($L) and the high memory limit register ($H) for words with bit patterns that do not match those of the search argument specified in the search argument register ($A). Only bit positions set to 1 in the mask are compared. This search is identical in form and function to the word (W) search described below, except that ODT performs a test for inequality.
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. If the value k precedes O, this command 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 user program to execute from the current breakpoint location and stops when the next breakpoint location is encountered or the end of the program is reached. If the value k is specified, ODT 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.

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
Q	Quantity register indicator. Represents the contents of register $Q (quantity register).
R	Command that sets all relocation registers to the highest address value, 177777(octal), so that they cannot be used in forming addresses.
nR	Command that sets relocation register n to the highest address value, 177777(octal), so that it cannot be used in forming addresses.
a;nR	Command that sets relocation register n to address value a. If n is omitted, ODT assumes relocation register 0.
S or nS	Command that executes one instruction and displays the address of the next instruction to be executed. If n is specified, ODT executes n instructions and displays the address of the next instruction to be executed.
U	Command that sets the current mode of ODT to user mode (RSX-11M-PLUS and Micro/RSX only).
V	Command that enables ODT's handling of all SST vectors, and writes the addresses of ODT's trap entry points into the table used by the SVDB$ Executive directive. (See Table 5-2 for a discussion of the SST vector registers and the $nV/ command.)
W or kW or m;W or m;kW	Command that searches memory between the address limits specified by the low memory limit register ($L) and the high memory limit register ($H) for words with bit patterns that match those of the search argument specified in the search argument register ($A). ODT compares each memory word and the search argument for equality under the mask specified in the search mask register ($M). Only bit positions set to 1 are compared. When a match occurs, ODT displays the address of the matching location and its contents. If W is preceded by k, the value k replaces the current contents of $A before initiating the search. If W is preceded by m (identified by the semicolon that follows it), the value m replaces the current contents of $M before ODT initiates the search.

(Continued on next page)

Table 2-3 (Cont.)
ODT Operators and Commands

Format	Meaning
X	Command that causes ODT to exit and returns control to the Executive of the host operating system.
Z	Command that sets the current mode of ODT to supervisor mode. (RSX-11M-PLUS and Micro/RSX only)

CHAPTER 3

CONTROLLING PROGRAM EXECUTION WITH ODT

When you run a task image into which ODT has been linked, ODT takes control before the first instruction of the task is executed. Information about the task is stored in ODT's internal registers, as described in Section 5.1.2.

At this point, you can execute your task immediately or issue ODT commands that affect locations or registers.

3.1 SETTING AND REMOVING BREAKPOINTS

A common method of using ODT is to set breakpoints at important points in the task and then execute the task. When a breakpoint is reached, execution is suspended. You can examine locations or registers to see how your task is executing. You can then change elements of your task and see how the changes affect execution.

3.1.1 Setting Breakpoints

To set a breakpoint at a location, issue a B (Breakpoint) command in the following format:

```
a;nB
```

a

An address expression (in any of the forms described in Section 2.2) representing the location at which the breakpoint is to be set. This location must always be the first word of an instruction.

n

The number of the breakpoint address register (from 0 to 7) to be used to store the address of the specified location. If you omit n, breakpoint address registers are assigned sequentially, beginning with register 0.

You can also set a breakpoint by opening a breakpoint address register as a word location and changing its contents. The address of a breakpoint address register is its register name, $nB. Opening and changing the contents of word locations is described in Chapter 4. Registers are described in Chapter 5.

In RSX-11M-PLUS and Micro/RSX systems where separate instruction and data space are used, breakpoints always refer to instruction space, regardless of which space you are referring to when you set the breakpoints. When a debugging session begins, you are automatically accessing instruction space. You access data space by entering the D command; you return to instruction space by entering the I command.

In RSX-11M-PLUS and Micro/RSX systems, each breakpoint address register is associated with a mode indicator that shows whether the breakpoint occurs in user or supervisor mode; this mode indicator depends on the mode in effect at the time the breakpoint is set. You set supervisor mode by entering the Z command; you return to user mode by entering the U command.

3.1.2 Removing Breakpoints

You can clear breakpoint address registers (and thus remove breakpoints) using the nB command, where n represents the number of the register. If you omit n, all breakpoint address registers are cleared. You can also clear a breakpoint and reset it by specifying a new address expression for a breakpoint address register, using the a;nB command.

The following example shows how breakpoints are set, cleared, and reset:

```
_B
_1020;B
_2030;B
_3040;B
_4050;B
_2032;1B
_3B
_
```

At the end of this example, breakpoint address register 0 is set to location 1020, breakpoint address register 1 is set to 2032, and breakpoint address register 2 is set to 3040. Breakpoint address register 3 is clear.

Note that ODT immediately generates a carriage return and line feed, and a new prompt when you type the letter B.

You can also clear a breakpoint register by opening it as a word location whose address is $nB and changing its contents, as described in Chapter 4.

3.2 BEGINNING PROGRAM EXECUTION

To begin executing your task, type the G (Go) command. At the G command, the following takes place:

- The task's starting address is returned to the program counter from the ODT general register in which it was stored.
- The task's stack and other general registers are restored.

- The contents of each location at which a breakpoint was set are swapped with the contents of the corresponding breakpoint instruction register. (These registers, described in Section 5.2, are initialized by ODT to BPT instructions.)

- The task begins executing.

The task continues to execute until it reaches one of the following:

- A breakpoint

- An error of type BE, EM, FP, IO, or TR (described in Appendix A)

- The end of the program

Once the task is executing, you cannot stop it except by aborting and then restarting it. (RSX-11M-PLUS and Micro/RSX systems include commands to reenter an interrupted program, as described in Sections 1.5.1.)

When the task reaches a breakpoint, ODT executes the BPT instruction that was swapped into the breakpoint location. At the BPT instruction the following takes place:

- Task execution is suspended.

- The contents of the user task general registers are stored in ODT internal registers.

- The original contents are restored to all breakpoint locations from the breakpoint instruction registers where they have been stored.

- ODT issues a message indicating that a breakpoint has been reached. This message has the format nB:a, where n is the breakpoint address register number and a is the location of the breakpoint that was stored in that register.

- ODT issues its prompt.

While task execution is suspended, you may issue any ODT command.

3.3 CONTINUING TASK EXECUTION

You can continue task execution by typing either the P (Proceed) command, the G command, or the aG command. The task continues executing until it reaches a breakpoint, one of the errors specified in Section 3.2, or the end of the program.

Use the P command to continue execution after a breakpoint. When you type P, the contents of the user general registers are restored, the BPT instructions are swapped into all breakpoint locations, and task execution resumes at the instruction following the last logical instruction executed. If execution stopped because of a breakpoint, it will resume at the breakpoint location. If execution stopped because of an error, it will resume at the location following the error location, not at the error location itself.

You can resume execution using the G command. However, because the G command does not transparently restore the breakpoint instruction, you should not use it to resume execution after a breakpoint.

To resume execution at a specific location, use the aG command. The argument a is an address expression representing the task location. The address specified must correspond to a word location boundary, that is, an even location. Registers are affected as described in Section 3.2. Execution begins at the specified location.

Note that you can use only G or aG to begin execution of a task. If you type P when no G command has been executed, ODT responds with a question mark and a new prompt.

3.4 USING THE BREAKPOINT PROCEED COUNT

If you set a breakpoint inside an execution loop, you may want to suspend execution only when the loop has been executed a certain number of times. You can specify how many times a loop should be executed by including a breakpoint proceed count with the P command, in the form kP. The loop is executed k-1 times; execution is suspended when the breakpoint is reached for the kth time.

The kP command is associated only with the breakpoint that has most recently occurred. The count k is stored as an octal value in a breakpoint proceed count register ($nG), where n is a number corresponding to the number of the appropriate breakpoint address register.

You can examine the breakpoint proceed count registers, or set them directly, at any time following the procedures for examining and setting word locations described in Chapter 4. These registers are all initialized by ODT with the value 1. If you change the value of a register, the new breakpoint proceed count will be used when the breakpoint is next encountered as a result of the P command.

3.5 STEPPING THROUGH THE PROGRAM

Another method of executing a task in stages is to use the S (Step) command. With this command, you can execute user task instructions one at a time or several at a time.

The command has the format nS, where n is the number of instructions that ODT should execute before suspending execution. The default value of n is 1.

When n instructions have been executed, ODT suspends task execution and prints a message of the form 8B:a, where a is the location of the next instruction to be executed. (The format of a is relative by default, as explained in Chapter 4.) ODT then prompts for another command.

The S command is implemented through the T-bit in the Processor Status Word (PSW) (see Appendix B). The T-bit is set when you issue the command. When the nth instruction is executed, control is returned to the task.

The following example shows ODT's response to the program execution commands described in this chapter:

```
_1,1052;B
_1,2052;1B
_G
0B:1,001052
_P
1B:1,002052
_S
8B:1,002056
_S
8B:1,002062
```

CHAPTER 4

DISPLAYING AND ALTERING THE CONTENTS OF LOCATIONS WITH ODT

During an ODT session, you can alter the contents -- either instructions or data -- of locations in your task. To alter the contents of a location, you must first open the location.

You open a location by displaying its contents, using any of the commands described in Sections 4.3 through 4.9. The contents displayed are automatically placed into the quantity register ($Q).

ODT displays a location by showing the address, a mode operator (either word mode or byte mode, depending on the size of the location opened), and the contents of the location. The format in which the location is displayed is controlled by the contents of the format register ($F), as described in Table 5-1. By default, ODT displays addresses in relative form whenever it has both the number of the relocation register containing the bias value closest to (but less than) the address and the relative location of the address from that value. When this information is not available, ODT prints the address in absolute form. (Relative and absolute forms are described in Section 2.2.1.)

ODT does not generate a carriage return or line feed after displaying the contents of a location. Until the location is closed, the cursor remains on the same line, wrapping as necessary.

4.1 ALTERING THE CONTENTS OF A LOCATION

You alter the contents of a location by typing the new contents immediately after the displayed contents. The new contents can be an absolute octal value (of up to six digits) or an expression equivalent to a 6-digit octal value, as described in Section 2.2.2. If you enter an octal value, you may omit leading zeros.

In the following examples, the value 1234 is substituted for the value 123456 in the location represented by the address expression 2,0. The value 177426 (the two's complement of the expression 16-370) replaces the value 000000 in the location represented by the address expression 4,10.

```
_2,0/123456 1234

_4,10/000000 16-370
```

After you have altered the contents of a location, you can verify the new contents by displaying them in a variety of modes, using the commands described in Section 4.10. These commands do not close the location. You can also display, and thus verify, the new contents by closing the location and then reopening it.

Note that you must close the currently open location before you can alter the contents of a new location.

4.2 CLOSING A LOCATION

To close one location without automatically opening another location, enter the RETURN command. This command has no effect on ODT when no location is open.

Entering RETURN generates a carriage return/line feed combination. ODT then prompts for another command, as follows:

```
_1,200/450123   RET
_
```

To close one location and automatically open another location, you can use any of the following commands, which are described in Sections 4.4 through 4.9:

```
LF     ^     _     @     <     >
```

4.3 OPENING WORD AND BYTE LOCATIONS

ODT interprets the slash character (/) as a word mode octal operator and the backslash character (\) as a byte mode octal operator. Using these operators in ODT commands provides the most direct way to open word and byte locations.

You can also open word and byte locations and display their contents in ASCII, or open and display words in Radix-50. These modes are described in Section 4.10.

4.3.1 Opening Word and Byte Locations Specifying an Address

To open a word location beginning at an address, in response to the ODT prompt, type an address expression corresponding to that address, followed by a slash (a/). The address must be even numbered. ODT opens the word location beginning at the specified address and displays the contents of that location as a 6-digit octal number.

To open a byte location, type an address expression corresponding to an odd- or even-numbered address, followed by a backslash (a\). ODT opens the byte location beginning at the specified address and displays the contents of that location as a 3-digit octal number.

The following examples show the effects of the a/ and a\ commands:

```
_1000/012675   RET
_1001\025   RET
```

4.3.2 Reopening the Location Last Opened

You can use the word mode and byte mode octal operators without address arguments to reopen the location last opened. The slash (/) command opens the word location last opened and displays the word at that location. The backslash (\) command opens the byte last opened and displays the contents of that byte. (If the last location opened was a word, the byte opened and displayed is the low-ordered byte of that word.)

When no location is open, you can also use the circumflex (^) command to open the last-opened location, as described in Section 4.5.

4.3.3 Moving Between Word and Byte Modes

The word mode and byte mode octal operators establish word mode and byte mode, respectively.

Once you have opened a location using the word mode octal operator (/), all locations subsequently opened will be octal words until the mode is changed. Once you have opened a byte location using the byte mode octal operator (\), all locations subsequently opened will be octal bytes until the mode is changed.

You can change from word to byte mode by opening a location with the a\ command or by specifying an odd-numbered address as the value a in the a/ command. Subsequent locations will be displayed as bytes until a word location is explicitly opened using an even-numbered address as the value a in the a/ command (or the a" or a% commands, described in Section 4.10).

The following example shows a change from word mode to byte mode and back again using an odd-numbered address in the a/ command. (The LINE FEED command, which opens the next sequential location in whatever mode is currently in use, is described in Section 4.4.)

```
_1001/123 321  RET
_/321  LF

001002 \021  LF

001003 \010  LF

001004 \201  RET
_1006/102054
```

If a word location is open, you can examine its low-order byte by typing the byte mode octal operator (\) immediately after the displayed contents of the location. The location remains open and you remain in word mode. The following example shows this use of the byte mode octal operator:

```
_1006/102054 \054  LF

001010/012345
```

You can also examine words or bytes of an open location in ASCII or Radix-50 modes, as described in Section 4.10.

4.4 OPENING THE NEXT SEQUENTIAL LOCATION

To open and examine successive locations, use the LINE FEED command. (On VT200-series terminals, a line feed is generated by pressing CTRL/J.) The LINE FEED command closes the currently open location and opens the next sequential location. If the currently open location is a word, the next sequential location will be opened as a word. If the currently open location is a byte, the next sequential location will be opened as a byte.

If you specify a value before entering the LINE FEED command, that value replaces the contents of the open location, as described in Section 4.1.

4.5 OPENING THE PRECEDING LOCATION

To back up in your task and open the location preceding the currently open location, use the circumflex (^) command. This command closes the currently open location. If the currently open location is a word location, the ^ command opens the word location immediately preceding it. If the currently open location is a byte, the ^ command opens the preceding byte.

If no location is currently open, the ^ command opens and displays the contents of the last-opened location. The contents may be a word or a byte, depending on the mode currently in effect.

If you specify a value before entering the ^ command, that value replaces the contents of the open location, as described in Section 4.1.

The following example shows the use of the LINE FEED and ^ commands. Location 232, relative to the bias contained in relocation register 0, is opened as a word and its contents are altered. In response to the LINE FEED and ^ commands, ODT proceeds to the next word location and then backs up to location 232 to display the new contents.

```
_0,232/005036 005046 (LF)

0,000234 /012746 ^
0,000232 /005046
```

4.6 OPENING ABSOLUTE LOCATIONS

To proceed from an open location to the location whose address is contained in that open location, use the at sign (@) command. This command closes the currently open location and uses the contents of that location as the absolute address of the next location to be opened. You can specify new contents for the original location by entering a value before the @ command, as described in Section 4.1.

You can use the @ command only if the currently open location is a word.

Opening an absolute location does not necessarily mean that the location is displayed as an absolute address. As shown in the following example, where relocation register 2 is set to contain the bias value 370 (as described in Section 5.2.1), ODT by default still displays the location as a relative address:

```
_370;2R
_2,600/012345 12746@
2,012356 /027117
```

Location 12356, relative to bias value 370, is equivalent to the absolute address specified, 12746.

4.7 OPENING PC-RELATIVE LOCATIONS

To open a location relative to the program counter, use the underscore (_) command. This command adds the contents of the currently open location to the value of the program counter, which is the address of the currently opened location plus 2. ODT then closes the currently

open location and opens the location whose address is the result of its calculation. If you enter a value before the _ command, this value replaces the contents of the open location and becomes the value used in the calculation.

You can use the _ command only if the currently open location is a word.

If the currently open location contains an odd number (or if it contains an even number but is already a byte location), so that the calculated address does not fall on a word boundary, the _ command opens a byte at the location calculated.

The following examples show how the _ command is used:

```
_1000/000040 _
001042 /052407

_0,232/012345 _
0,012601 /041

_0,232/012345 123456_
0,123712 /020301
```

4.8 OPENING RELATIVE BRANCH OFFSET LOCATIONS

Use the right-angle bracket (>) command to open a location at a branch offset relative to the currently open location. The offset is calculated as follows:

1. The low-order byte of the contents of the currently open location is interpreted as a signed value. A negative value results in a negative branch offset.

2. This value is multiplied by 2.

3. The resulting offset is added to the PC value, which is the address of the currently open location plus 2.

The > command closes the currently open location and opens the location whose address is the value calculated. Its effects are shown in the following examples:

```
_1,66/005046>
1,000204 /000601

_1032/000407 301>
000636 /000010
```

If you specify a value before entering the > command, the low-order byte of that word is used in the offset calculation. The value replaces the contents of the open location, as described in Section 4.1.

4.9 RETURNING FROM A CALCULATED LOCATION

If you have used any of the three address calculation commands described in the last three sections (@, _, or >) and wish to return to the location from which you began to calculate addresses, use the left-angle bracket (<) command. This command closes the currently open location and reopens the previous word.

The following example shows the use of the < command:

```
_1036/021346 ^
001034/172543 101036_
102074 /000002 <
001034 /101036
```

If the currently open location was not opened by a @, _, or > command, the < command simply closes and reopens the current location.

4.10 USING DIFFERENT OUTPUT MODES

The examples in the previous sections have shown ODT output in word mode octal and byte mode octal. However, you can also use ODT to display the contents of locations in word or byte mode ASCII and word mode Radix-50.

These modes follow the same rules as word mode octal and byte mode octal:

- You can use the LINE FEED command to open succeeding locations in the same mode in which the currently open location was opened.
- You can enter any mode operator to display the contents of the currently open location in another mode without changing the mode in effect or closing the location.

The interaction of mode operators was shown in Section 4.3.3, where a location opened in word mode octal was examined in byte mode. The LINE FEED command that followed opened the next sequential location in word mode octal.

4.10.1 ASCII Mode

ODT interprets the quotation mark character (") as a word mode ASCII operator and the apostrophe (') as a byte mode ASCII operator. You open a location in word mode ASCII with the a" command and in byte mode ASCII with the a' command.

If you open a location in any mode and then type a word mode ASCII operator, the contents of the open location are displayed as two ASCII characters, but the location is not closed.

If you open a location in any mode and then type a byte mode ASCII operator, the contents of the low-order byte of the open location are displayed as one ASCII character. The location is not closed.

The following examples show these uses of the ASCII operators:

```
_0,440"AB

_2,100'H

_0,232/034567 'w "w9   LF

0,000234/000123   RET
_'S
```

If you enter the word mode ASCII operator to examine the contents of a location, and the location is aligned on a byte boundary (an odd-numbered address), ODT does not return an ASCII character. Instead, it displays the contents of the location in the mode currently in effect, as follows:

```
0,000235\025 "025
```

4.10.2 Radix-50 Mode

ODT interprets the percent sign (%) as a word mode Radix-50 operator. (There is no byte mode Radix-50 operator, because Radix-50 is a method of fitting three characters into a word and cannot be used in smaller units.)

You can use the Radix-50 operator to open locations. The a% command opens the location specified in the address expression a and displays its contents as three Radix-50 characters. The % command reopens the last-opened word and displays its contents as three Radix-50 characters.

If a word location is open, you can enter the % operator to examine the Radix-50 contents of that location without closing the location.

The following examples show these uses of the word mode Radix-50 operator:

```
_4,232%IG1

_4,232/034567   RET
_%IG1

_4,000232/034567 %IG1
```

Like the word mode ASCII operator, the Radix-50 operator cannot be used to interpret values that begin on byte boundaries. If you enter the Radix-50 operator when the currently open location has an odd address, ODT simply displays the byte value in the current mode.

Remember that you must enter new contents for a location as an octal value or an expression, not as Radix-50 characters. To determine the octal equivalent of Radix-50 characters, use the Radix-50 evaluator (*), described in Section 7.3.5.

CHAPTER 5

USING REGISTERS IN ODT

ODT has a number of 16-bit registers. Some of these registers are used for temporary storage of values. Some contain values used repeatedly throughout the execution of your task under ODT. All the registers are word locations that you can examine and alter.

Each ODT register has a unique name beginning with a dollar sign ($). The $ and the character or characters that follow it make up an address expression that identifies the register.

This chapter explains how ODT uses its registers. Tables 5-1 and 5-2 summarize the registers and are useful for quick reference.

5.1 GENERAL REGISTERS

ODT has eight general registers, numbered $0 though $7, which store the contents of the user program's general registers when ODT has control. These registers are automatically set when ODT is first invoked and when a breakpoint occurs. They can also be set by the user.

5.1.1 Examining and Setting General Registers

To examine a general register, enter the register name as the address expression in the a/, a", or a% command. For example, you can enter any of the following:

```
$7/
$3%
$1"
```

ODT opens a register like any other word location. You can then alter the contents of the register or use any of the following commands, as described in Chapter 4:

RET LF / ^ _ @ " % > ' \

ODT treats the general registers as sequential word locations.

5.1.2 Contents of General Registers

When you issue the RUN command and ODT initially gains control, information about the user task is stored in the general registers as follows:

Register	Contents
$0	Task's entry-point address
$1	First three characters of task's run-time name (Radix-50)
$2	Last three characters of task's run-time name (Radix-50)
$3-$4	Version number of user task if the program included the .IDENT directive; otherwise, the version number of ODT

When a breakpoint occurs, ODT's general registers store the contents of the task's general registers.

5.2 ODT INTERNAL REGISTERS

The ODT internal registers store values for use during a debugging session. For example, they store the locations of breakpoints and the memory limits to be used in search operations. Each register is a 16-bit location that you can open by specifying the register name as the address expression with any ODT command that opens a word location. You can enter any of the following:

```
$3R/
$A"
$C%
```

It is rarely useful to examine an internal register in ASCII or Radix-50 mode.

You can alter the contents of these registers as you would the contents of any word. However, this is not recommended in some cases, as noted in Tables 5-1 and 5-2.

Ten of the ODT internal registers are single registers; that is, there is only one register for each function. You refer to one of these registers as $x, where x is an alphabetic character. Table 5-1 lists these registers in alphabetical order. In the task, they appear in the order listed in Table 2-3, that is:

```
$S $W $A $M $L $H $C $Q $F $X
```

You can access these registers as sequential word locations in this order, as in the following example:

```
_$S/000000 (LF)
$W /000001 (LF)
$A /000000 (LF)
$M /177777
```

Table 5-1
ODT Single Registers

Register	Function
$A	Search argument register. You set this register to a word search argument by opening with the / operator, or to a byte search argument by opening with the \ operator. It can also be set using the memory commands described in Chapter 6.
$C	Constant register. The 16-bit value in this register can be used as an address expression or a value through the constant register indicator C, described in Sections 2.2.2 and 7.3.3.
$F	Format register. When this register is set to 0, ODT displays all user task addresses in relative form, if an appropriate bias value is available in one of the relocation registers. When this register is set to any other value, ODT displays user task addresses in absolute form. See Section 2.2.1 for a description of absolute and relative forms of addresses.
$H	High memory limit register. The location contained in this register is the upper location limit for ODT search, list, and fill memory operations. Initialized to 0.
$L	Low memory limit register. The location contained in this register is the lower location limit for ODT search, list, and fill memory operations. Initialized to 0.
$M	Search mask register. You set this register to a word search mask by opening with the / operator, or to a byte search mask by opening with the \ operator. It can also be set by arguments specified with the memory commands described in Chapter 6. Initialized to -1, 177777(octal).
$Q	Quantity register. ODT sets this register to the last value displayed, as described in Section 7.3.4. $Q is also used for the results of expression calculations using the = operator.
$S	Processor status register. This stores the Processor Status Word (see Appendix B) resulting from the last instruction executed prior to a breakpoint. Users do not normally change the contents of this register directly.
$W	Directive Status Word register. This contains the Directive Status Word (DSW) of the task, indicating the success or specific cause of rejection of the most recently executed directive. The contents of this register are maintained across breakpoints. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for details on the DSW.
$X	Reentry vector register. A positive value in this register causes ODT to retain the register values for successive entries of ODT, as described in Section 5.2.2.

The other ODT internal registers are grouped into sets of eight or three sequential word locations. The integer n is part of the register name, in the form $nx; you must always include n, even if its value is 0.

Table 5-2 lists the register sets alphabetically. In a task, they appear as sequential word locations in the order listed in Table 2-3, that is:

$nB $nG $nI $nR $nV $nE $nD

Table 5-2
ODT Register Sets

Register	Range of n	Function
$nB	0 - 7	Breakpoint address register n. This contains user-specified address of location (breakpoint) in the user task whose contents are to be swapped with the contents of $nI when a G or P command is executed. A ninth register, $8B, is used by ODT for single-step execution.
$nD	0 - 2	Device control LUN (logical unit number) register n. As described in Section 6.1.4, register $0D contains the LUN of the user terminal, and register $1D contains the LUN of the console device. Register $2D contains the QIO event flag number, normally a default value of 000034 (octal).
$nE	0 - 2	SST stack contents register n. The top three items on the user program stack are placed into these registers when a synchronous system trap occurs. Stack contents depend on the type of trap taken, as explained in the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual.
$nG	0 - 7	Breakpoint proceed count register n, where n corresponds to breakpoint address register n. This contains the number of times the breakpoint location should be encountered before the breakpoint is recognized. Each register is initially set to 1 and can be set through the kP command (see Section 3.4), or by opening $nG and altering its contents. A ninth register, $8G, is used by ODT for single-step execution.
$nI	0 - 7	Breakpoint instruction register n. This register is initialized to contain a BPT instruction, op code 000003, which is swapped with the contents of register $nB when the G or P command is executed. The functions of the BPT instruction are described in Section 3.2. A ninth register, $8I, is used by ODT for single-step execution.

(Continued on next page)

Table 5-2 (Cont.)
ODT Register Sets

Register	Range of n	Function
$nR	0 - 7	Relocation register n. This contains the relocation bias of a relocatable object module, enabling ODT to display user task addresses in relative form, if $F is set to 0 (see Table 5-1). ODT initializes each register to 177777(octal).
$nV	0 - 7	SST vector register n. This contains the entry-point address of the ODT routine for handling a synchronous system trap. If both ODT and the user program have SST vectors enabled for the trap, ODT automatically receives the trap, except for vector 6 ($6V), which must be explicitly enabled through the V command (see Table 2-3). ODT handling of a trap can be disabled by clearing the register; the user program vector then receives the trap. Registers correspond to traps as follows:

Register	SST Vector
$0V	Odd address reference in word instruction (also, on some processors, illegal instruction executed)
$1V	Memory protection violation
$2V	T-bit trap or BPT instruction executed
$3V	IOT instruction executed
$4V	Reserved or illegal instruction executed
$5V	Non-RSX-11 EMT instruction executed
$6V	TRAP instruction executed
$7V	PDP-11/40 floating-point exception error

The following sections describe the functions of ODT internal registers $nR, and $X in greater detail. The ODT internal registers $C, and $Q are described in Chapter 7. Registers used in memory operations ($L, $H, $M, $A, and $nD) are described in Chapter 6.

5.2.1 Relocation Registers

ODT's eight relocation registers allow you to refer to locations by relative addresses instead of absolute addresses. Since relative addresses are easy to determine from source file listings, using them makes debugging faster and simpler.

When ODT is initialized, each relocation register is set to 177777(octal). This is the highest possible memory address and therefore cannot be used in constructing address expressions. To make a relocation register useful, you place in it the base address of a relocatable module or another convenient point, as explained in Section 2.2.1. This address functions as a relocation bias that is added to the relative address in an address expression to form the absolute address of a location.

You obtain the base (starting) address of a module by consulting the memory allocation synopsis in your task map. This part of the map gives the octal starting address of each program section and each module that makes up a program section. It also shows the extent of the module, in octal and decimal.

The following figure shows a memory allocation synopsis for a brief task:

```
SECTION                        TITLE   IDENT   FILE

. BLK.:(RW,I,LCL,REL,CON)  001264 001012 00522.
                           001264 000574 00380. HIYA          HIYA.OBJ;1
$$RESL:(RO,I,LCL,REL,CON)  010152 000112 00074.
$$$ODT:(RW,I,GBL,REL,OVR)  002276 005654 02988.
                           002276 005654 02988. ODTRSX M06    ODT.OBJ;1
```

5.2.1.1 **Setting Relocation Registers** - You can set relocation registers either by opening them as word locations and altering them, or by using special ODT commands that affect relocation registers.

To open a relocation register as an octal word, use the register name $nR as the address expression a in the a/ command (or any of the other commands described in Chapter 4 that open words). You can enter a new value for the register after examining the existing contents.

The ODT command a;nR sets register $nR to the location specified as address expression a. If you omit n, register $0R is assumed.

5.2.1.2 **Clearing Relocation Registers** - To remove a relocation register from consideration in calculating addresses, enter the nR command, where n is the number of the relocation register. This command sets the register to 177777 (octal), so that it is no longer useful in constructing address expressions. If you omit n, all relocation registers are set to 177777(octal).

5.2.2 The Reentry Vector Register

If you have fixed a task in memory (see the FIX command in the RSX-11M/M-PLUS MCR Operations Manual), you can use the reentry vector register, $X, to maintain register values set during your debugging session and to keep track of your access to the task.

The reentry vector register contains the value -1 when your task is built. When you execute the task for the first time, the register value is incremented to 0. The 0 value causes ODT to omit the task name from the invocation message line (described in Section 1.3) the next time you enter the task. This omission indicates that the task is fixed in memory.

If you intend to reenter the task for further debugging, you should set $X to 1 or another positive nonzero value. As long as the value of $X is positive and nonzero, the fixed task is reentered at the value stored in $7 (the program counter), and the values stored in ODT's registers are maintained. You can continue to debug the task using the breakpoints, constants, and other values established in an earlier debugging session. If $X is not positive, all registers are initialized when you reenter the task.

You can use the reentry vector register as a counter to record how many times you have entered a fixed task. To do this, set the register to 1 the first time you enter your task and increment it each time you enter the task again.

CHAPTER 6

MEMORY OPERATIONS IN ODT

ODT allows you to perform three kinds of operations on blocks of memory in your task:

- Search memory for bit patterns or references to locations
- Fill memory with a value
- List blocks of memory on an output device

Section 6.1 describes how to establish the registers used in memory operations. The subsequent sections of this chapter describe how to use ODT commands to perform these operations.

6.1 REGISTERS USED IN MEMORY OPERATIONS

ODT memory operation commands function between limits in memory that you must specify. Search and fill commands require an argument to be searched for or deposited. Search operations also require a search mask.

ODT maintains registers to contain all these values. You can set these registers as word or byte locations (as described in Chapters 4 and 5) before issuing memory operation commands. You can also specify a search argument and a search mask as the k and m arguments in the commands themselves. If you do not specify an argument in one of these commands, ODT uses the current contents of the appropriate register. If you do specify an argument, that argument replaces the contents of the register.

6.1.1 Search Limit Registers

There are two search limit registers: $H, containing the high memory limit for a search, fill, or list operation; and $L, containing the low memory limit. You deposit a memory location in one of these registers by opening it as a word location and changing its value to the address of the location. You can specify the location in either absolute or relative form, as follows:

```
_$L/000000 1000   RET
_/ 001000 2,4060  LF

_$H/000000 3,100  RET
_
```

If the value in $L is greater than the value in $H, ODT does not perform the memory operation requested using these registers. Instead, ODT displays its prompt.

6.1.2 Search Mask Register

ODT initializes the search mask register $M to 177777(octal), so that all bits are set to 1. You set the value of the register by opening it as a word location and changing its value. Only bit positions set to 1 in the search mask are compared in the search operation. The value compared is that set for the corresponding bit position in the search argument register $A.

You can also set register $M by specifying a value m, followed by a semicolon, in any of the search commands described in Section 6.2.

6.1.3 Search Argument Register

The search argument register $A contains the value searched for in a memory search operation or filled with in a memory fill operation. You can set this value by opening register $A as a word or byte location and changing its contents, or by specifying the argument k in one of the search commands described in Sections 6.2 and 6.3.

As noted in Section 6.1.2, only bit positions set to 1 in the search mask are compared in any memory search operation.

6.1.4 Device Control LUN Registers

The device control LUN registers $0D and $1D contain the logical unit numbers of the user terminal (TI:) and the console device (CL:), respectively. You specify one of these registers as the value n in the n;a;kL command (see Section 6.4.1) to indicate what device should be used for a listing.

The Task Builder assigns default values for these registers: 000007 (octal) for $0D and 000010 (octal) for $1D. To reset these registers, you can link your task using the TKB option UNITS=keyword as described in the RSX-11M/M-PLUS and Micro/RSX Task Builder Manual.

You may find it more convenient to assign a new value for CL: before beginning your debugging session. Use the MCR command ASN or the DCL command ASSIGN in one of the following formats:

```
ASN devicename=CL:    (MCR command)
```

or

```
ASSIGN CL:  devicename    (DCL command)
```

For more information on these commands, see the appropriate command line interpreter manual for your system.

6.2 SEARCHING MEMORY

There are three memory search commands: W, N, and E. Each of these commands has several forms, depending on the number of registers that already contain values that you want to use in the search operation. The following sections describe these command forms.

6.2.1 Searching for a Word or Byte

The W command searches for occurrences of the search argument (comparing bit positions specified in the search mask) within the range set by the contents of the search limit registers.

The full form of the command is m;kW, where m specifies the search mask and k specifies the search argument. However, you can omit either or both of these arguments if the corresponding registers contain the values that you want to use. If you omit m, you should also omit the semicolon argument separator.

ODT performs an exclusive OR (XOR) operation on the contents of each location and the search argument; it then ANDs the result of this comparison with the search mask. A result of zero indicates a match. When a match occurs, ODT prints the address and contents of the location and repeats the search operation until the high memory limit is reached.

6.2.2 Searching for Inequality of a Word or Byte

The N command is the opposite of the W command. It examines the search range for words or bytes that do not exactly match the search argument in the positions determined by the search mask.

The full form of the command is m;kN, where m specifies a search mask and k specifies a search argument. As with the W command, you can omit either or both of these arguments.

The search algorithm proceeds like that for the W command, except that ODT only displays a location's address and contents when the AND operation has resulted in a nonzero value.

6.2.3 Searching for a Reference

The E command searches for memory locations containing instructions whose execution results in a reference to the task address specified as the search argument. Because the search argument represents an address, it can only be a word, not a byte.

The full form of the command is m;kE, where m represents the search mask and k the search argument. You can omit either or both of these arguments if you want to use the values already contained in registers $M and $A. For effective use of the E command, the search mask should be set to 177777(octal), so that all bit positions are compared.

ODT compares each location within the search limits and displays the address and contents of locations that contain any of the following:

- The search argument as an absolute address
- A relative address offset reference to the absolute address specified as the search argument
- A relative address branch reference to the absolute address specified as the search argument

6.3 FILLING MEMORY

The F command fills the block of memory defined by the high and low memory limit registers with the value in the search argument register. You can set this register using the command $A/ (see Section 6.1.3), or specify the argument k with the F command, in the form kF.

If the last location opened was a word, the memory range is filled with words. If the last location was a byte, the memory range is filled with bytes. The low-order byte in register $A is used.

In the following example, word locations 1000 through 1776 are set to 0, and byte locations 2000 through 2777 are filled with ASCII spaces (40 octal):

```
_1000;1R
_2000;2R
_3000;3R
_$L/000000 1,0   (RET)
_$H/000000 2,-2  (RET)
_0F
_$L/001000 2,0   (RET)
_$H/001776 3,-1  (RET)
_$A\000 40   (RET)
_F
```

6.4 LISTING MEMORY

The L command lists on an output device the block of memory defined by the high and low memory limit registers. The following sections describe how you request a listing and what the listing looks like.

6.4.1 Command Format

The L command has the following format:

```
n;a;kL
```

n

The device control LUN register number for the listing operation. A value of 0 indicates the user terminal (TI:). Any other value is interpreted as 1 and indicates the console listing device (CL:). The default is 0.

a

The low memory limit for the listing operation. If you omit a, the value of register $L is used. If you specify a, that value is placed in $L.

k

The high memory limit for the listing operation. If you omit k, the value of register $H is used. If you specify k, that value is placed in $H.

You must include the semicolon argument separator (;) between a and k if you specify the argument a. You must include two semicolons if you specify the argument n.

6.4.2 Listing Format

A memory listing is formatted in groups of eight units. Each line begins with a location, in relative form if possible (see Section 2.2.1), followed by eight words or eight bytes in the current output mode. A memory listing is displayed in whatever mode was used to open the last opened location. Thus, you can list blocks of memory in word mode octal, byte mode octal, word mode ASCII, byte mode ASCII, or word mode Radix-50, as described in Section 4.10.

The following example shows the output displayed on the output device in response to various listing commands. Note that the question mark displayed in response to the ' command is not in this case ODT's error indicator. It is merely the ASCII character stored in the next byte.

Example 6-1 ODT Listing Format

```
_1344;1400L
001344 /047503 046125 020104 020111 040510 042526 054440 052517
001364 /020122 040516 042515 050040 042514 051501 037505
_1344" CO L
001344 "CO UL D  I  HA VE  Y OU
001364 "R  NA ME  P LE AS E?
_1344\ 103 L
1344 \103 117 125 114 104 040 111 040
1354 \110 101 126 105 040 131 117 125
1364 \122 040 116 101 115 105 040 120
1374 \114 105 101 123 105
_1344' C L
001344 'C O U L D   I
001354 'H A V E   Y O U
001364 'R   N A M E   P
001374 'L E A S E
_' ?
_1300;R
_$H/001400 0,101
_1344' C L
0,000044 'C O U L D   I
0,000054 'H A V E   Y O U
0,000064 'R   N A M E   P
0,000074 'L E A S E ?
```

CHAPTER 7

PERFORMING CALCULATIONS IN ODT

ODT performs a variety of arithmetic calculations useful in determining offsets, Radix-50 equivalents, and other values. This chapter describes commands that perform these calculations. Section 7.1 explains how to calculate relocatable addresses. Section 7.2 explains how to calculate offsets. Section 7.3 describes how to evaluate expressions.

7.1 CALCULATING RELOCATABLE ADDRESSES

If you know the absolute (relocated) address of a location and want to determine its relative address, or what relocation register contains the closest base address, use one of the forms of the a;nK command.

If you specify both a, the absolute address, and n, a relocation register, in the a;nK command, ODT calculates and displays the relative address, as follows:

```
_4000;2K =2,001460
```

Note that the equal sign is part of ODT's response, not part of the command that you enter.

If you omit n, ODT uses the relocation register whose contents are closest to (but less than) the absolute address specified.

If you omit a, ODT assumes the address of the last location opened. You should omit the semicolon argument separator if you omit a.

To determine the absolute address of an open location or of the last-opened location, enter a dot (current location indicator) followed by an equal sign (expression evaluation operator), as described in Section 7.3.2.

7.2 CALCULATING OFFSETS

The O (Offset) command calculates and displays the PC-relative offset and the branch displacement from one location to another.

There are two forms of this command. The aO command calculates the offset from the currently open location to the location represented by address expression a. This form of the command can be used only when a location is open; you type it on the same line as the displayed contents of the open location.

The a;kO command calculates the offset from the location represented by address expression a to the location represented by address expression k. (In this case, k can have any of the address expression forms described in Section 2.2.) This command can be entered either on the same line as an open location or on a separate line, in response to the ODT prompt.

The O command (in either form) calculates either positive or negative offsets. Negative offsets are displayed in two's complement form.

ODT displays the PC-relative offset and the branch displacement as 6-digit octal numbers. The PC-relative offset is preceded by an underscore and followed by a space. The branch displacement is preceded by a right-angle bracket (>), as shown in the following example:

```
_1034/103421 10460 _000010 >000004
```

A location that is open when you use the aO or a;kO commands remains open after the offset and branch displacement are displayed. You can perform another calculation, change the contents of the location, or enter any ODT command that affects an open location.

Offsets can be calculated in either I- or D-space (RSX-11M-PLUS and Micro/RSX systems only).

7.3 EVALUATING EXPRESSIONS

You can evaluate expressions during your debugging session using the techniques described in the following sections. To evaluate an expression while a location is open, enter the evaluation command on the same line as the displayed contents of the location. ODT places the results of its evaluation into the $Q register. To replace the contents of the open location, you enter Q or the value of the expression. You can also evaluate expressions when no location is open by typing the evaluation command in response to the ODT prompt.

7.3.1 Equal Sign Operator

To evaluate an expression, enter the expression followed by the equal sign (=). The expression is converted to a 6-digit octal value, placed in the $Q register, and displayed. ODT truncates the octal value of 16 bits when necessary.

Negative values are calculated, stored, and displayed in two's complement form. You can specify a negative value either in two's complement form or with the minus sign.

You can perform addition and subtraction within an expression to be evaluated. To add values, include a plus sign (+) or a space between the values. To subtract values, include a minus sign (-). ODT does not recognize parentheses or assign precedence to any operator. Expressions are evaluated left to right.

An address expression, in relative or absolute form, can be all or part of an expression to be evaluated.

You can include one of these three indicators in the expression: the current register indicator, the constant register indicator, or the quantity location indicator. These indicators are described in the following sections.

If you enter the equal sign without an expression to be evaluated, ODT evaluates the null expression as zero and enters zeros in the $Q register.

The following examples show the evaluation of expressions using the equal sign. Relocation register $0R contains the value 370. The constant register contains the value 40.

```
_0,0=000370
_0,16=406
_0,C=000430
_0,16+16+2=0000426
_16-370=177426
_177777+16+16=000033
_-1+16+16=000033
_C 177777=000037
_232323=032323
```

7.3.2 Current Location Indicator

The dot indicator (.) represents the address of the currently open location. You use this symbol to include the address of the currently open location as part or all of an expression to be evaluated.

The following example shows how the current location indicator is used:

```
_320;1R
_1,10/000000 .+10=000340
```

7.3.3 Constant Register Indicator

The C indicator specifies the 16-bit value contained in the constant register, $C. You can set this register to any value and use the indicator in place of any a or k argument in an ODT command (as shown in Section 2.2). You change the value of C by opening the $C register as a word location and changing its contents.

7.3.4 Quantity Register Indicator

ODT stores the last value that it displayed in the quantity register, $Q. When you open a location, ODT stores that location's contents in the $Q register. If the location is a byte, the $Q register contains that byte in its low-order byte and zeros in its high-order byte.

You can refer to this 16-bit value by using the quantity register indicator Q. The quantity register indicator is especially useful for changing the contents of open locations and for setting registers, as shown in the following examples:

```
_1342/173214 Q+10  RET
_/173224  RET

_$3/013624 Q;5R  RET
_5,20/013644
```

7.3.5 Radix-50 Evaluation

To enter Radix-50 characters, you must know the numeric value of each Radix-50 word. A Radix-50 word, as explained in Section 4.10.2, contains three Radix-50 characters. To determine the value of the Radix-50 word, enter the numeric equivalents of the Radix-50 characters in that word, separated by asterisks, as an expression to be evaluated. Follow the expression with an equal sign, as shown in Section 7.3.1. ODT calculates a 6-digit octal value, places that value in the $Q register, and displays it immediately after the equal sign, as follows:

```
_33*24*12=125752
```

Note that you cannot evaluate Radix-50 characters in conjunction with any other evaluation operation (addition, subtraction, location calculation). You cannot use any other symbol (C, Q, .) in the expression to be evaluated.

If you specify the equivalents of only two Radix-50 characters, ODT fills the high byte of the word with zeros, as necessary.

The Radix-50 character set includes all alphabetic and numeric characters (A through Z, 0 through 9) plus three special characters: dollar sign ($), dot (.), and space ( ). Table 7-1 contains the numeric equivalents of all Radix-50 characters.

Table 7-1
Numeric Equivalents of Radix-50 Characters

Radix-50 Character	Numeric Equivalent	Radix-50 Character	Numeric Equivalent
Space	0	T	24
A	1	U	25
B	2	V	26
C	3	W	27
D	4	X	30
E	5	Y	31
F	6	Z	32
G	7	$	33
H	10	.	34
I	11	Unused	35
J	12	0	36
K	13	1	37
L	14	2	40
M	15	3	41
N	16	4	42
O	17	5	43
P	20	6	44
Q	21	7	45
R	22	8	46
S	23	9	47

The following example shows how the asterisk (*) is used in conjunction with the Radix-50 operator (see Section 4.10.2):

```
_1054/003151 %AAA 1*3*5=003275 3275 (RET)
%ACE
```

CHAPTER 8

THE EXECUTIVE DEBUGGING TOOL (XDT)

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 ODT. You should be an experienced ODT user before using XDT.

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 9, Debugging with XDT, for a list of ODT commands not used in XDT and a table of XDT operators and commands.

8.1 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 the Executive Debugging Tool 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.

8.2 HOW TO INCLUDE XDT IN YOUR RSX-11M OR RSX-11M-PLUS SYSTEM

System support for 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 in your system.

On RSX-11M systems, including XDT in the system diminishes the size of system pool by approximately 2.5K bytes.

On RSX-11M-PLUS systems with I- and D-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 I- and D-space support does not reduce the size of available system pool.

On RSX-11M-PLUS systems without I- and D-space support, including XDT in the system diminishes the size of system pool by approximately 2.5K bytes.

8.3 LOADABLE XDT ON MICRO/RSX AND PRE-GENERATED 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. The greatest advantage of this feature is that having XDT in the system does not diminish the amount of available system pool. It also allows you to load XDT only when you need to use it for debugging.

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

```
LOA /EXP=XDT /VEC [/PAR=parname] [/HIGH]
```

parname

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

/HIGH

If you specify the /HIGH qualifier, XDT is loaded at the top of the partition.

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

```
UNLOAD /EXP=XDT
```

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

8.4 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 data base 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 through PR6) or completely non-interruptible (PR7). The shared system data base 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 $FORK 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. It runs at PR7 and, therefore, is totally non-interruptible. Pending interrrupts 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/M-PLUS 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

8.4.1 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 serviced by 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.

8.5 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. There are two reasons you might want to do so:

- To locate a suspected software bug in a particular piece of system code
- 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.

8.5.1 XDT and Synchronous System Traps (SSTs)

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 8.5.1.1) result in a trap to XDT. A trap to XDT is indicated by a message at the console terminal in the following form:

```
xx: address
XDT>
```

xx:

One of the entry codes listed in Table 8-1.

address

The PC at the time of the trap.

Each entry code is associated with a particular trap vector, also listed in Table 8-1. See Appendix A for more information about error detection.

Table 8-1
XDT Trap Entry Codes

Entry Code	Vector	Reason
BC:	30	Bug Check -- internally detected software fault (Loadable XDT only)
BE:	14	Breakpoint Entry -- a BPT instruction
EM:	30	EMT Instruction
FP:	244	Floating Point Instruction (on 11/40 only; RSX-11M only)
IL:	10	Illegal Instruction
IO:	20	IOT Instruction
MP:	250	Memory Protection Violation
OD:	4	Odd Address or Nonexistent Memory
SO:	4	Stack Overflow
TE:	14	T-bit Trap
nB:	14	XDT Set Breakpoint (n is a register number 0 through 8)

8.5.1.1 **Processor Traps and System Crashes** - An SST generally indicates that there is a software fault which may cause corruption of an Executive database. For example, inserting an odd address into the link pointer of a Task Control Block 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 ($STKDP <= 0).
- The BPT instruction is fatal when executed in system state, unless XDT is present in the system. The BPT instruction is a means for entering XDT. When you deliberately set a breakpoint in XDT (see the XDT nB command in Table 8-4), XDT actually inserts a BPT instruction where you want the breakpoint and saves the instruction it replaced with the breakpoint.
- The TRAP instruction is legal only in system state ($STKDP = 0). The directive processors use the TRAP instruction to post error codes back to the directive dispatcher.

8.5.2 Entering XDT from a Virgin System Boot

A virgin system is an RSX-11M or RSX-11M-PLUS system that has completed execution of SYSVMR.CMD but has not yet been saved (see the MCR command SAVE in the RSX-11M/M-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:

```
BOO 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. INITL 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 on 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 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 retains initial control when the system is rebooted -- not XDT, 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.

8.5.3 Entering XDT Using the BRK Command

The BRK (Breakpoint to Executive Debugging Tool) command passes control of the system to XDT. Note that on RSX-11M and RSX-11M-PLUS systems this is an MCR command. The message XDT prints on the console includes a 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.

8.5.4 Entering XDT Using the BPT Instruction

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 cancelling the BPT instruction is as follows:

- Replace a system state instruction with the BPT instruction
- The system traps to XDT when it executes the BPT instruction
- Use XDT to restore the original instruction replaced by BPT
- Decrement the PC by subtracting 2 from the contents of register R7
- Set any desired breakpoints 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 restore any intruction. When you have debugged the driver, take out the BPT instruction and reassemble the source code.

8.5.4.1 **Inserting a BPT Instruction Using the OPEN Command** - You can use the command OPEN 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 and 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 re-installing the image wipes out any BPT instruction set with the OPEN command.

To examine and replace an instruction in the Executive, use the following command syntax:

```
OPEN  addr
```

addr

The address in the Executive that is to be opened.

To examine and replace an instruction in a memory-resident privileged task, use the following command syntax:

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

addr

The address in the task that is to be opened.

/TASK=taskname

The name of the memory-resident task.

To examine and replace an instruction in a driver, use the following command syntax:

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

addr

The address in the driver that is to be opened.

/DRV=ddnn:

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:

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

8.5.4.2 **Inserting a BPT Instruction Using the ZAP Utility** - Since 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 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.

8.5.5 Entering XDT When the System Is Hung (RSX-11M and 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 CPU, examine the PC, deposit an odd address in the PC, then continue. Since the system will attempt to execute an odd address, it will trap to XDT.

CHAPTER 9

DEBUGGING WITH XDT

This chapter describes how to use XDT. It includes information about interpreting bug checks, 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 ODT commands not used in XDT.

9.1 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. The system clock continues to run, however.

If the trap to XDT was unintentional, you can try to isolate the fault that caused the trap. The RSX-11M/M-PLUS Guide to Writing an I/O Driver contains some helpful information on isolating faults and tracing system faults 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 PC by 2.

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

9.1.1 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.

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 using the XDT command P, or you can cause the system to enter the crash dump routine by entering the XDT command X.

9.1.2 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.

9.1.3 Using XDT to Debug a Driver

I/O drivers in RSX-11 systems, including Micro/RSX systems, can operate in system or interrupt state. You can use XDT in either of these states to set breakpoints and 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 APR. You cannot set XDT breakpoints in this manner (see the XDT ;Bn command), 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 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:

- Replace the current mapping context (3163) with starting address of the driver (2104)
- Replace the MOV instruction (010405) with a BPT instruction (3)

- Replace the driver address (2104) with previous mapping context (3163)
- Enter the XDT P (Proceed) command

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 example below:

```
XDT>172352/ 003163 2104
XDT>120102/ 010405 3
XDT>172352/ 002104 3163
XDT>P
   .
   .
   .
BE:120104
XDT>
```

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

9.1.4 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 through 4 map the Executive while the system is in XDT. Kernel APR 7 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 MCR task if the BRK command caused the trap to XDT).

To examine a memory location, you must divide the physical address into two components: the relocation bias and the displacement. Next, you must manually map this section of physical memory by putting the relocation bias into Kernel APR 5. Then, you can examine the location by referencing the virtual address as 120000 + displacement.

NOTE

If you were to map with Kernel APR 6, then the virtual address would be 140000 + displacement.

Kernel APRs 0 through 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

9.1.5 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 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.

9.1.6 T-bit Error

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

```
TE:  address
XDT>
```

Generally, a T-bit trap occurs when the T-bit is set in the PSW by some other mechanism 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 APR 5 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 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 APR 5 to map to the driver, deposit a BPT instruction in the driver using XDT, and restore the original contents of kernel APR 5. Return to user mode using the XDT command P.

9.2 INTERPRETING BUG CHECKS

The RSX-11M, 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.

On RSX-11M systems, XDT is entered from the Executive via the IOT instruction. XDT prompts with:

```
IO:nnnnnn
XDT>
```

Follow these two steps to isolate the failure:

1. Find the location of the IOT instruction in the source code for the Executive.

2. Ascertain the type of corruption from the context of the IOT. IOT instructions included in the source code for this purpose are typically generated by the CRASH macro.

RSX-11M-PLUS uses a different mechanism for reporting this type of fault: the bug check. The bug check uses the EMT instruction to enter XDT. On RSX-11M-PLUS systems with resident XDT, XDT prompts with:

```
EM:nnnnnn
XDT>
```

The steps for isolating the failure are similar to those described for RSX-11M.

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:

```
BC:nnnnnn ffffff eeeeee s
```

nnnnnn

The address within the Executive where the bug check was executed.

ffffff

The octal facility code.

eeeeee

The octal error code.

s

Either the letter F or the letter N, which represent fatal and non-fatal faults, respectively. If the letter N appears, XDT allows you to use the proceed command.

Table 9-1 shows error codes that are independent of which facility detected the fault. The high bit for these error codes will always be zero. The definiton, symbolic name, and octal value of each code are shown.

Table 9-2 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 9-1
Common Facility-Independent Error Code Definitions

SST type errors - Major error code 1	
BE.ODD 000100	Odd address or other trap four
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
BE.ISR 000203	Interrupt service routine modified R0-R3
BE.FHW 000204	Fatal hardware error
BE.CSR 000205	Device CSR disappeared during powerfail
BE.IDC 000206	Internal database consistency error
BE.ACP 000207	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 non-existent flags

Table 9-2
Standard Bugcheck Format Facility Code Definitions

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
BF.MP 000302	Exec - Multiprocessing
Multiprocessor-specific type errors	
BE.NDS 100100	Init failure - D-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 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 procesing
BF.QIO 000312	Exec - QIO directive
BF.OPT 000313	Exec - Seek optimization
BF.ACC 000314	Exec - System resource accounting
BF.KAS 000315	Exec - Kernal AST support
BF.DIR 000316	Exec - Miscellaneous directives
BF.SAN 000317	Exec - Crash with sanity timer message

9.3 XDT COMMANDS AND OPERATORS

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

No $M - Mask register

No $X - Entry flag registers

No $V - SST vector registers

No $D - I/O 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

All XDT command I/O goes to or from the console terminal. Table 9-3 contains all of the XDT commands and operators.

Table 9-3
XDT Operators and Commands

Format	Meaning
+ or space	Arithmetic operator used in expressions. Add the preceding argument to the following argument to form the current argument.
-	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.
*	Radix-50 separator used in constructing Radix-50 words.
.	Current location indicator. Causes the address of the last explicitly opened location to be used as the current address for XDT operations.
;	Argument separator. Separates multiple arguments, allowing an address expression or XDT register value to be identified.
(RET) or k(RET)	Command that closes the currently open location and prompts for the next command. If (RET) is preceded by k, the value k replaces the contents of the currently open location before it is closed.
(LF) or k(LF)	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 (LF) is preceded by k, the value k replaces the contents of the currently open location before it is closed.

(Continued on next page)

Table 9-3 (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 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.

(Continued on next page)

Table 9-3 (Cont.)
XDT Operators and Commands

Format	Meaning
$n	Expression that represents the address of one of eight general registers, where n is an octal digit identifying R0 through 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 $nx	Expression that represents the address of one of XDT's internal registers, where x is one of the following alphabetic characters, and n is one octal digit. Registers exist within XDT in the following order: S Processor status register (hardware PS) A Search argument register L Low memory limit register H High memory limit register C Constant register Q Quantity register F Format register nB Breakpoint address registers nG Breakpoint proceed count registers nI Breakpoint instruction registers nR Relocation registers
" or a"	Word mode ASCII operator. Interprets and displays the contents of the currently open (or the last previously opened) location as two ASCII characters, and stores this word in the quantity register ($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 last 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.

(Continued on next page)

Table 9-3 (Cont.)
XDT Operators and Commands

Format	Meaning
% or a%	Word mode Radix-50 operator. Interprets and displays the contents of the currently open (or the last 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, DELETE, or CTRL/U	Illegal expressions that cancel the current command. ODT then awaits a new command. The decimal values 8 and 9 are not legal characters and thus, when entered, cause XDT to ignore the current command. The DELETE and CTRL/U functions are not operational in RSX-11M unless the terminal driver supports transparent read/write (a system generation option).
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.
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.

(Continued on next page)

Table 9-3 (Cont.)
XDT Operators and Commands

Format	Meaning
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 D-space (RSX-11M-PLUS and Micro/RSX only).
G or aG	Command that begins system execution at the current location in the program counter, following these steps: sets BPT instructions in or restores BPT instructions to all breakpoint locations; restores the Processor Status Word; and 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 ($7) contents before proceeding as described above.
I	Command that accesses instruction space. After this command is issued, XDT interprets all references to locations as referring to the I-space of the task (RSX-11M-PLUS and Micro/RSX only).
K	Command that, using the relocation register whose contents are equal to or closest to (but less than) the address of the currently open location, computes the physical distance (in bytes) between the address of the currently open location and the value contained in that relocation register. XDT displays this offset and stores the value in the quantity register ($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).

(Continued on next page)

Table 9-3 (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. 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 stops when the next breakpoint location is encountered or 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(octal), so that they cannot be used in forming addresses.
nR	Command that sets relocation register n to the highest address value, 177777(octal), so that 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.

(Continued on next page)

Table 9-3 (Cont.)
XDT Operators and Commands

Format	Meaning
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 10

ADDITIONAL DEBUGGING AIDS

The Task Builder on your system allows you to specify the debugger of your choice to help you in program development. You should build only one debugger into your task at a time. If you want to switch from one debugger to another, you should rebuild your task.

Section 10.1 shows how you specify other debuggers to the Task Builder for the three environments described in Chapter 1. Section 10.2 describes the Trace program, a debugging aid available on your system.

10.1 ACCESSING OTHER DEBUGGING AIDS

The following sections show how to specify a debugger other than ODT to be linked with your object module(s). The example in each section shows a command line for linking the Trace debugging aid, described in Section 8.2. You can specify the file name of any debugger in place of [1,1]TRACE.OBJ.

10.1.1 MCR Command Line

To link a debugger with your task using MCR, specify the name of the debugger object module as input to the Task Builder. Follow the debugger object module name with the /DA switch, as in the following example:

```
TKB>MYTASK=MYFILE,[1,1]TRACE/DA
```

The /DA switch identifies the file specified as a debugger. Since the Task Builder assumes that the file type of input files is .OBJ, you need not specify the file type of the debugger object module.

10.1.2 DCL Command Line

To link a debugger into your task using DCL, specify the name of the debugger object module as an argument to the /DEBUG qualifier with the LINK command, as in the following example:

```
> LINK/DEBUG:[1,1]TRACE/TASK:MYTASK MYFILE
```

Since DCL assumes that the file type of input files for the task builder is .OBJ, you need not specify the file type of the debugger object module.

10.2 THE TRACE DEBUGGING PROGRAM

The Trace program is a debugging aid that can be used instead of or along with ODT to provide information about the execution of a user task. Trace is most appropriate for use with relatively simple tasks or with sections of tasks. Trace cannot be used with XDT.

Trace is an object module that you specify to the Task Builder when you build your task, as described in Section 8.1. It is located in UFD [1,1] on the system disk, with the name TRACE.OBJ.

Trace is not an interactive program like ODT. When you run your task, Trace is executed once and prints its listing on pseudo device CL:. To run Trace again, you must run your task again.

10.2.1 The Trace Listing

A Trace listing contains two lines of information for each instruction executed in the user's task. The first line is made up of five octal words, representing the contents of the following registers:

1. Current relative program counter (PC)
2. Current PC
3. Next PC
4. Processor Status Word
5. Directive Status Word

The relative PC is determined by subtracting a user-specified bias value from the actual PC. Section 8.2.2 describes how you specify this bias value.

The second line of the Trace listing contains eight octal words representing the contents of the following:

1-6. R0 through R5
7. Stack pointer
8. The top word of the stack

Example 8-1 is a sample Trace listing for part of a user task.

Example 8-1 Sample Trace Output

```
001714  003174  003176  170020  000001
    002637  000120  000000  140200  000000  000000  001256  003074

001716  003176  003202  170024  000001
    002637  000120  000000  140200  000000  000000  001256  003074

001722  003202  003074  170024  000001
    002637  000120  000000  140200  000000  000000  001260  001260

001614  003074  003100  170020  000001
    002612  000120  000000  140200  000000  000000  001260  001260
```

10.2.2 Bias Values and Ranges

You can use the GBLPAT Task Builder option to specify:

- The bias value to be used in determining the relative PC
- The range(s) of task locations to be traced

10.2.2.1 **Specifying a Bias Value** - To specify a bias value for relative PC calculation, enter an option line in the following format in response to the Task Builder prompt:

```
GBLPAT=segname:.BIAS:value
```

segname

The name of the task's root segment.

value

The octal value to be subtracted from the actual PC to establish relative PC. (If a value is not specified, the initial stack pointer is used.)

10.2.2.2 **Specifying Ranges to be Traced** - To specify up to four ranges of locations for which execution should be traced, enter an option line in the following format in response to the Task Builder prompt:

```
GBLPAT=segname:.RANGE:low1:high1 [...  :lown:highn]
```

segname

The name of the task's root segment.

low1...lown

The low addresses, relative to the bias value, of ranges to be traced.

high1...highn

The high addresses, relative to the bias value, of ranges to be traced.

There can be up to four ranges. You must specify both the low and the high address of each range.

APPENDIX A

ERROR DETECTION

ODT and XDT respond to errors in user input and to certain hardware-detected errors that occur during task execution. This appendix describes these errors, ODT and XDT's response to them, and what action the user can take to correct them.

A.1 INPUT ERRORS

ODT and XDT use the question mark (?) to indicate that they have detected an error in user input. After displaying the question mark, the debugger generates a carriage return and line feed, and prompts for another command.

ODT and XDT respond with the question mark to any of the following input errors:

- Reference to an address without an operator
- Reference to an address outside the task's partition (ODT only)
- Reference to an address that is not mapped (XDT only)
- Reference to a nonexistent register -- for example, $20
- Reference to supervisor space by a nonprivileged user (ODT only)
- 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.

Neither ODT nor XDT tells you what error has caused it to display the question mark. However, an error sometimes causes them to return one of the error codes listed in Section A.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), ODT and XDT repeat their prompt but do not display a question mark.

A.2 TASK IMAGE ERROR CODES

As described in Table 5-2, eight SST vector registers are used to contain pointers to error-handling routines. Upon detecting an error condition, ODT and XDT activate the appropriate routine and display 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. ODT and XDT display 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 through $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; 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 a location which 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, execution of an illegal instruction.

Users Action: Check your code for the use of a word instruction when a byte instruction was intended (MOV instead of MOVB, for example) or a typographical error in the address specification.

TE

Explanation: T-bit exception. The T-bit was set by some other mechanism than a breakpoint or an S or P command. This 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 B

PROCESSOR STATUS WORD

The Processor Status Word (PS), stored at hardware location 777776, contains information on the current status of the processor. The information contained in this location includes:

- 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 Processor Status Word is shown in Figure B-1.

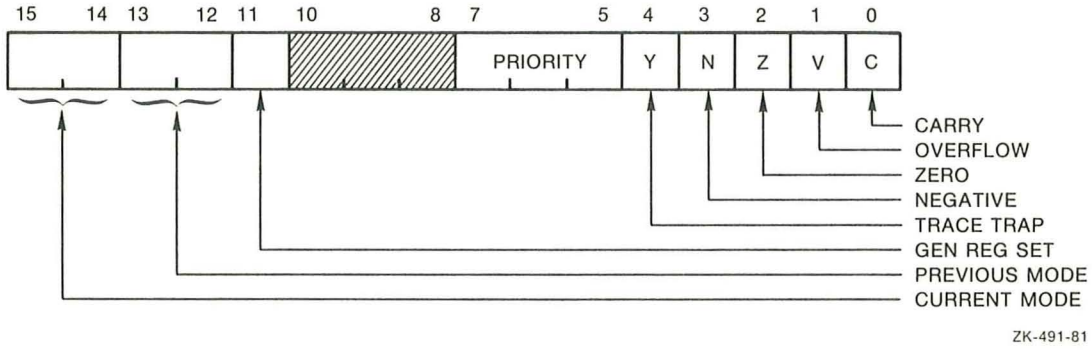

Figure B-1 Format of the Processor Status Word

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 through 5 show the current priority of the central processor. The central processor operates at any one of eight levels of priority (0 through 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 trap 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 Processor Status Word will be loaded. The trap (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, described in Section 3.5.

The condition codes N, Z, V, and C (bits 3 through 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

INDEX

A register, 2-6, 5-3, 6-2, 9-10
a symbol, 2-1
ABORT command, 1-5
Absolute
  address, 2-2
  location, 2-5, 4-4, 9-9
Address
  absolute, 2-2, 5-6
  relative, 2-2
    format, 2-2
  relocatable, 2-2, 5-6, 9-12
    calculating, 2-9, 7-1
Address expression
  See Expression
Apostrophe operator (')
  See Operator
Argument
  register, 2-6, 5-3
  separator, 2-5, 9-8
Arithmetic
  calculations, 7-1
Arithmetic operator
  See Operator
ASCII
  displaying, 4-6
  operator, 2-7
    byte mode, 4-6, 9-10
    word mode, 4-6, 9-10
Asterisk separator (*)
  See Separator
At sign command (@), 2-5, 4-4, 9-9

B command, 2-8, 3-1 to 3-2, 9-11
B register, 2-7, 5-4, 9-10
Backslash operator ()
  See Operator
Bias value, 2-3
  Trace program, 10-3
BPT trap instruction, 8-5
Branch
  location, 2-6, 9-9
  offset, 4-5, 7-1
    calculating, 4-5
Breakpoint, 3-1, 3-3, 8-7, 9-13
  address register, 2-7, 3-1, 5-4, 9-10
    clearing, 3-2
  inserting with OPEN, 8-7
  inserting with ZAP utility, 8-8
  instruction register, 2-7, 5-4, 9-10
  proceed count, 3-4
    register, 2-7, 5-4, 9-10
  removing, 2-8, 3-2, 9-11
  setting, 2-8, 3-1
    XDT, 9-2
BRK command, 8-7
Bug check, 9-4
Byte location
  displaying, 4-2 to 4-3
  opening, 4-2 to 4-3
Byte mode
  changing to word mode, 4-3
  operator
    ASCII, 2-7, 9-10
    octal, 2-8, 4-2, 9-11

C register, 2-6, 5-3, 9-10
  indicator, 2-8, 7-3, 9-12
Character, 2-4
Circumflex command (), 2-5, 4-2, 4-4, 9-9
Comma separator (,)
  See Separator
Command
  at sign (@), 2-5, 4-4, 9-9
  B, 2-8, 9-11
  circumflex (), 2-5, 4-2, 4-4, 9-9
  D, 2-8, 9-12
  E, 2-9, 6-2
  equal sign (=), 2-8, 7-2, 9-11
  F, 2-9, 6-4
  G, 2-9, 8-6, 9-12
  I, 2-9, 9-12
  K, 2-9, 9-12
  L, 2-10, 6-4, 9-13
  left-angle bracket (<), 2-6, 9-9
  Line feed, 2-5, 4-3, 9-8
  N, 2-10, 6-2
  O, 2-10, 7-1, 9-13
  P, 2-10, 8-7, 9-13
  R, 2-11, 9-13
  Return, 2-5, 4-2, 9-8
  right-angle bracket (>), 2-6, 9-9
  S, 2-11, 9-1, 9-14
  U, 2-11
  underscore (_), 2-5, 4-4, 9-9
  V, 2-11
  W, 2-11, 6-2
  X (ODT), 1-4, 2-12
  X (XDT), 9-1, 9-14
  Z, 2-12
Constant register
  See C register
CTRL/C
  ODT, 1-5
CTRL/J, 4-3
CTRL/O
  XDT, 9-13
CTRL/U
  ODT, 2-8
Current location indicator (.), 7-3, 9-8

D command, 2-8, 9-12

D register, 2-7, 5-4, 6-2
D-space
  See data space
Data space, 7-2, 9-3
  command, 2-8, 9-12
  enabling, 1-3
DCL command
  linking
    ODT, 1-3
    ODTID, 1-3
    supervisor libraries, 1-4
DEBUG command
  RSX-11M-PLUS and Micro/RSX, 1-5
Device control
  LUN register, 2-7, 5-4, 6-2
Directive Status Word
  See DSW
Dollar sign ($), 2-6, 5-1, 9-10
Dot (.) indicator
  See Register indicator
Driver
  debugging, 9-2
DSW
  register, 5-3
DSW register
  See W register

E command, 2-9, 6-2 to 6-3
E register, 2-7, 5-4
EMT trap instruction, 8-5, 9-5
Equal sign command (=), 2-8, 9-11
Equal sign operator (=)
  See Operator
Error
  detection, A-1
  error codes, 9-6, A-2
  facility codes, 9-6
  T-bit, 9-4
  task image, A-2
Executive Debugging Tool
  See XDT
Exit command
  ODT, 2-12
  XDT, 9-14
Expression, 2-3
  address, 2-2
  evaluating, 2-3, 7-2
  format, 2-3
  illegal, 2-8, 9-11
  Radix-50
    evaluating, 7-4
  register address, 2-6, 9-10

F command, 2-9, 6-4
F register, 2-6, 5-3, 9-10
Fill command
  See F command
Format
  memory listing, 6-5
  PSW, B-1
  Trace program listing, 10-2
Format register
  See F register

G command, 2-9, 3-2, 3-4, 8-6, 9-12
G register, 2-7, 5-4, 9-10
GBLPAT
  See TKB
General register, 5-1
  contents, 5-2
  examining, 5-1
  setting, 5-1
Go command
  See G command

H register, 2-6, 5-3, 6-1, 9-10
High limit register
  See H register

I command, 2-9, 9-12
I register, 2-7, 5-4, 9-10
I-space
  See instruction space
Indicator
  See Register indicator
Instruction space, 3-2, 7-2, 9-3
  command, 2-9, 9-12
  enabling, 1-3
Internal register, 5-2
  accessing, 5-2
Interrrupt processing, 8-3
IOT trap instruction, 8-5, 9-4

K command, 2-9, 7-1, 9-12
k symbol, 2-1

L command, 2-10, 6-4, 9-13
L register, 2-6, 5-3, 6-1, 9-10
Left-angle bracket command (<), 2-6, 4-5, 9-9
Limit register, 5-3
Line feed command, 2-5, 4-3, 9-8
LINK command, 1-3
  /DEBUG qualifier, 10-1
  specifying a debugger, 10-1
Linking
  ODT
    from DCL, 1-3
    from MCR, 1-2
    to enable instruction and data space features, 1-3
List command
  See L command
Location
  absolute, 2-5, 4-4, 9-9
  altering, 4-1
  branch, 4-5
  closing, 4-2
  displaying, 4-1
    format, 4-1
  indicator, 7-3
  opening, 4-1
    ASCII, 4-6
    branch offset, 4-5
    byte, 4-2
    next sequential, 4-3
    preceding, 4-4

Location
  opening (Cont.)
    Radix-50, 4-7
    word, 4-2
  PC-relative, 4-4
  reopening last opened, 4-2
  returning from, 4-5
Location indicator
  See Register indicator
Loop, 3-4
Low limit register
  See L register

M register, 2-6, 5-3, 6-2
m symbol, 2-1
Mask register
  See M register
MCR command
  linking
    ODT, 1-2
    ODTID, 1-3
    supervisor libraries, 1-4
Memory
  E command, 6-2
  examining memory location
    XDT, 9-3
  F command, 6-4
  fill command, 2-9
  H register, 2-6
  L command, 6-4
  L register, 2-6
  limit register, 5-3, 6-1
  list command, 2-10
  listing
    format, 6-5
  N command, 6-2
  search command, 2-9 to 2-11, 6-2
  W command, 6-2
Message
  invocation, 1-4
Minus sign operator (-)
  See Operator
Mode
  user, 9-2

N command, 2-10, 6-2 to 6-3
n symbol, 2-1

O command, 2-10, 7-1, 9-13
Octal operator, 2-3, 2-7 to 2-8
  byte mode, 9-11
  word mode, 9-11
ODT
  exiting, 1-4
  invoking, 1-4
  linking, 1-2
  overview, 1-1
ODTID module, 1-3
Offset, 2-3
  branch, 7-1
  calculating, 2-10, 7-1, 9-13
    instruction and data space, 7-2
Offset (Cont.)
  negative, 7-2
  PC-relative, 7-1, 9-13
  positive, 7-2
OPEN command, 8-7
Operating system
  return to, 2-12
Operator, 2-3 to 2-4
  apostrophe ('), 2-7, 4-6, 9-10
  ASCII
    byte mode, 4-6
    word mode, 4-6
  backslash (), 2-8, 4-2 to 4-3, 9-11
  byte mode
    ASCII, 2-7, 9-10
    octal, 2-8, 4-2, 9-11
  equal sign (=), 7-2
  minus sign (-), 2-3 to 2-4, 9-8
  per cent sign (%), 2-7, 4-7, 9-11
  plus sign (+), 2-3 to 2-4, 9-8
  quotation mark ("), 2-7, 4-6, 9-10
  Radix-50
    word mode, 4-7
  slash (/), 2-7, 4-2 to 4-3, 9-11
  space, 9-8
  word mode
    ASCII, 2-7, 9-10
    octal, 2-7, 4-2, 9-11
    Radix-50, 2-7, 9-11

P command, 2-10, 3-3 to 3-4, 8-7, 9-13
PC-relative
  location, 2-5, 4-4
  offset, 2-10, 7-1, 9-13
Per cent sign operator (%)
  See Operator
Plus sign operator (+)
  See Operator
Proceed command
  See P command
Proceed count, 3-4
  register, 5-4
Processor clock, 8-9
  turning off, 9-4
Processor states, 8-2
  priority, 8-3
Processor Status Word
  See PSW
Processor traps
  XDT, 8-5
Prompt
  ODT, 1-4
  XDT, 9-4
PSW, B-1
  format, B-1
  register, 2-6, 5-3

Q register, 2-6, 5-3, 7-2, 9-10
  indicator, 2-11, 7-3, 9-13

Quantity register
  See Q register
Question mark (?)
  user input error, A-1
Quotation mark operator (")
  See Operator

R command, 2-11, 9-13
R register, 2-7, 5-5, 9-10
  clearing, 2-11, 5-6, 9-13
  setting, 2-11, 5-6, 9-13
Radix-50
  character set, 7-4
  displaying, 4-7
  evaluating, 7-4
  numeric equivalents, 7-4
  opening, 4-7
  operator, 2-7
    word mode, 4-7, 9-11
  separator (*), 2-4, 7-4, 9-8
Range
  Trace program, 10-3
Reentry vector register
  See X register
Reference
  search, 6-3
Register, 2-6, 5-1
  A, 2-6, 5-3, 6-2, 9-10
  address expression, 5-1
  B, 2-7, 5-4, 9-10
    clearing, 3-2
  breakpoint
    address, 5-4, 9-10
    instruction, 5-4, 9-10
    proceed count, 5-4, 9-10
  C, 2-6, 5-3, 9-10
    indicator, 9-12
  D, 2-7, 5-4, 6-2
  E, 2-7, 5-4
  F, 2-6, 4-1, 5-3, 9-10
  G, 2-7, 5-4, 9-10
  general, 5-1, 9-10
    contents, 5-2
    examining, 5-1
    setting, 5-1
  H, 2-6, 5-3, 6-1, 9-10
  I, 2-7, 5-4, 9-10
  indicator, 2-8, 2-11
  internal, 5-2
    accessing, 5-2
  L, 2-6, 5-3, 6-1, 9-10
  M, 2-6, 5-3, 6-2
  memory operations, 6-1
  Q, 2-6, 5-3, 7-2, 9-10
    indicator, 9-13
  R, 2-7, 5-5, 9-10
    clearing, 2-11, 5-6, 9-13
    setting, 2-11, 5-6, 9-13
  S, 2-6, 5-3, 9-10
  search limit, 6-1
  V, 2-7, 5-5
  W, 2-6, 5-3
  X, 2-6, 5-3, 5-6
  XDT internal, 9-10
Register indicator, 2-3
  C register, 2-3, 7-3, 9-12
  current location (.), 2-3, 7-3, 9-8
  Q register, 2-3, 7-3, 9-13
Register set, 5-4
Relative
  address, 2-2
    format, 2-2
  branch location, 2-6, 4-5, 9-9
Relocatable
  address, 2-2, 5-6, 9-12
    calculating, 2-9, 7-1
Relocation register
  See R register
Return command, 2-5, 4-2, 9-8
Right-angle bracket command (>), 2-6, 4-5, 9-9

S command, 2-11, 3-4, 9-1, 9-14
S register, 2-6, 5-3, 9-10
Search
  argument register, 2-6, 5-3, 6-2
  byte, 6-3
  command, 6-2
    E, 6-3
    N, 6-3
    W, 6-3
  limit register, 6-1
  mask register, 2-6, 5-3, 6-2
  memory
    command, 2-9 to 2-10
  reference, 6-3
  word, 6-3
Semicolon separator (;)
  See Separator
Separator
  argument (,), 2-4, 9-8
  argument (;), 2-5
  Radix-50 (*), 2-4, 7-4, 9-8
Slash operator (/)
  See Operator
Space operator
  See Operator
SST
  stack content register, 2-7, 5-4
  vector
    handling, 2-11
    register, 2-7, 5-5
  XDT, 8-4
Stack Depth Indicator
  See $STKDP
Step command
  See S command
$STKDP, 8-3, 8-6, 9-1
Supervisor library, 1-4
  install
    READ/WRITE, 1-4
Supervisor mode, 1-4, 3-2
  command, 2-12
  debugging, 1-4

Supervisor mode (Cont.)
  setting, 2-12

Task
  fixed, 5-6, 9-2
  privileged, 9-1
    debugging with ZAP, 8-8
Task Builder
  See TKB
Task execution
  aborting, 1-5
  beginning, 2-9, 3-2, 9-12
  continuing, 2-10, 3-3
  resuming, 1-5, 3-4
Task map, 1-2 to 1-3, 5-6
TE trap, 1-5
TKB
  /DA switch, 10-1
  GBLPAT option, 10-3
  linking
    ODT, 1-2
    ODTID, 1-3
    supervisor libraries, 1-4
  specifying a debugger, 10-1
  UNITS option, 6-2
Trace program, 10-2
  listing, 10-2
    format, 10-2
Trap, 1-5, 2-11, 5-5, 8-4, A-2
  entry codes, 8-5
  handling, 2-11, 5-5
  instruction
    BPT, 8-5
    EMT, 8-5, 9-5
    IOT, 8-5, 9-4
    TRAP, 8-5
  SST vector register, 5-5
TRAP trap instruction, 8-5

U command, 2-11
Underscore command (_), 2-5, 4-4, 9-9
User mode, 3-2
  command, 2-11
  setting, 2-11

V command, 2-11
V register, 2-7, 5-5
Variable, 2-1
Vector
  reentry register, 5-3, 5-6

W command, 2-11, 6-2 to 6-3
W register, 2-6, 5-3
Word location
  displaying, 4-2 to 4-3
  opening, 4-2 to 4-3
  underscore command (_), 4-5
Word mode
  changing to byte mode, 4-3
  operator
    ASCII, 2-7, 9-10
    octal, 2-7, 4-2, 9-11
    Radix-50, 2-7, 9-11
X command (ODT), 1-4, 2-12
X command (XDT), 9-1, 9-14
X register, 2-6, 5-3, 5-6
x symbol, 2-1
XDT, 8-1
  BPT
    inserting with OPEN, 8-7
    restoring instruction, 8-8
    setting, 8-7
  bug check, 9-4
  commands, 9-8
  debugging
    driver, 9-2
    privileged task, 9-1
  entering, 8-4
    BPT instruction, 8-7
    BRK command, 8-7
    from hung system, 8-9
    virgin system boot, 8-6
  examining memory location, 9-3
  including in system, 8-1
  loadable, 8-2
    error code, 9-5
    facility code, 9-5
  operators, 9-8
  SST, 8-4
  trap entry codes, 8-5

Z command, 2-12
ZAP utility
  debugging privileged task, 8-8
  inserting breakpoint, 8-8
  modifying system image, 8-8

RSX-11M/M-PLUS and
Micro/RSX Debugging
Reference Manual
AA-EZ50A-TC

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 user/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

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

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line

Printed in U.S.A.