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

Order No. AA–JS09A–TC

Contents

Chapter 3 Controlling Program Execution with ODT

Chapter 4 Displaying and Altering the Contents of Locations with ODT

Chapter 5 Using Registers in ODT

Chapter 6 Memory Operations in ODT

Chapter 7 Performing Calculations in ODT

Chapter 8 Additional Debugging Aids

Appendix A Error Detection

Appendix B Processor Status Word

Index

Examples

Figures

Tables

Preface

Manual Objectives

This manual describes the On-Line Debugging Tool (ODT) used to debug user task images. It provides reference information on all ODT commands, as well as information on how to use the commands to debug task images.

Intended Audience

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

Structure of This Document

Chapter 1 gives 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 explains the special symbols used in ODT and includes a reference table with an alphabetical listing of ODT commands. New ODT users should read Chapters 3 to 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 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 with one or more instructions at a time.

Chapter 4 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 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 want to consult the tables in this chapter, as well as those in Chapter 2, for quick reference regarding specific registers.

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

Chapter 7 describes how to use ODT to perform arithmetic calculations.

Chapter 8 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 describes how ODT responds to errors in user input or program logic. It lists all ODT error message codes in alphabetical order.

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

Associated Documents

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

Conventions Used in This Document

The following conventions are used in this manual:

Convention	Meaning
xxx	A symbol with a 1- to 3-character abbreviation, such as DEL 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.

Summary of Technical Changes

The following changes are reflected in this version of the manual:

- Information specific to the RSX-11M operating system has been deleted.
- The information on the Executive Debugging Tool (XDT) is now contained in the *RSX-11M-PLUS and Micro/RSX XDT Reference Manual.*
- This manual also corrects technical errors in the text and examples of the previous version. It represents a significant reorganization of material that is 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 do the following:

- 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 to 7 describe how to use 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, by setting breakpoints (by using BPT instructions in your program) at selected locations or by 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. Chapter 7 describes these calculations.

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 operating systems 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 or modules 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 the Monitor Console Routine (MCR). Section 1.2.2 describes how to link ODT if your CLI is the DIGITAL Command Language (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. Section 1.2.4 describes how to change logical unit number (LUN) assignments specific to ODT.

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

1.2.1 Linking ODT from MCR

To link ODT with your program or programs when your CLI is MCR, first invoke the Task Builder (TKB) 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 TKB command and specify the name of the file or files to be linked. Include the /DA switch, which indicates that a debugger (in this case ODT, the default) should be linked into the image. 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 /CR/-SP switches. The following example shows the resulting command line:

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

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

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

1.2.2 Linking ODT from DCL

To link ODT with your program or programs when your CLI is 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 [RET]
```

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

For further information on using DCL, consult 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

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. 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 with ODT.OBJ. (See Table 2-3 for a description of these commands.)

If your CLI 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/-SP=MYTASK [RET]
```

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

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

```
$ LINK/DEBUG/CODE:DATA_SPACE/MAP MYTASK [RET]
```

You can add other qualifiers to the LINK command. See the *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, and specify the debugger object module in the MCR or DCL command line. The following example shows the resulting MCR command line:

```
TKB>
```

The following example shows the resulting DCL command line:

```
$
```

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 by 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 and use either the /RW switch or the :RW argument on the RESSUP or SUPLIB options, respectively. You can alternatively build the task as privilege: 0.

1.2.4 Assigning ODT LUNs

When you build a task, the TKB automatically assigns default values for registers $0D and $1D. These registers contain the LUNs of the user terminal (TI) and the console device (CL), respectively. However, you may want to assign new values for the registers. For example, if you are debugging an editor task, you need to assign TI to another terminal so the output from the task is directed to that terminal and does not interrupt your debugging session. Or, if you want to direct output to another printer, you can assign CL to that printer. To override these values, you can link your task by using the TKB options ASG and GBLPAT, as described in the *RSX-11M-PLUS and Micro/RSX Task Builder Manual.* The following example shows a series of task-build command options that direct ODT to use TT4 instead of TI:

```
TKB>MYTASK/DA,MYTASK/-SP=MYTASK [RET]
TKB>/ [RET]
Enter Options:
TKB>ASG=TT4:1 [RET]
TKB>GBLPAT=MYTASK:.ODTL1:1 [RET]
TKB>// [RET]
>
```

ODT allocates two extra LUNs to direct output to the TI and CL devices. The LUNs are pointed to by the global symbols .ODTL1 and .ODTL2. The global symbol .ODTL1 is the address of a word that contains the ODT LUN for input and output to TI. The global symbol .ODTL2 is the address of a word that contains the ODT LUN for output to CL. To redirect the output to either of these devices, you need to specify these symbols as parameters to the GBLPAT option. For more information on reserved global symbols, see the *RSX-11M-PLUS and Micro/RSX Task Builder Manual.*

You can specify any terminal device (or logical resolving to the same). Also, you can use any available LUN.

Alternatively, 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:

```
MCR>ASN devicename=CL:
```

or

```
DCL>ASSIGN CL: devicename
```

For more information on these commands, see the appropriate CLI manual for your system.

1.3 Invoking ODT

Regardless of what operating system or CLI you use, enter the RUN command and specify 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 that indicates it has been invoked and that identifies the task image it controls. On the next line, ODT displays its prompt, an underscore (_), which indicates it is ready to accept commands.

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

```
>RUN HIYA [RET]
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 nonoctal number like 8 or 9) or by pressing CTRL/U or by pressing 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 (CLI) prompt at any time by pressing 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.

Note

If your terminal has the CTRL/C abort characteristic set and if you want to be able to resume a debugging session after pressing CTRL/C, you need to turn CTRL/C abort off. Otherwise, pressing CTRL/C will have the same effect as entering the ABORT command.

You can interrupt task execution without aborting your task and then continue debugging. After pressing CTRL/C, use the commands described in the following section.

1.5.1 Resuming a Debugging Session

RSX-11M-PLUS and Micro/RSX operating systems allow you to interrupt and then resume task 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 CLI 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 [RET]
ODT:TT15
_G
[CTRL/C]
>DEBUG [RET]
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 On-Line Debugging Tool (ODT) operators and commands, and it 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

The command descriptions in Chapters 2 to 7 and Table 2–3 use lowercase alphabetic variables to represent numeric and alphabetic arguments specified in commands. These variables 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 six digits long with a maximum value of 177777_8, 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 six 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 by 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 by 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 to $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 to 7, in forming relative addresses.

Relative Address Format

n,k

Parameters

n

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

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 to 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 (.), as 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_8 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 to 7. The lowercase letters used in the command descriptions are explained in Table 2–1.

Table 2–3: ODT Operators and Commands

Format	Meaning
+ (plus sign) or space	Arithmetic operator used in expressions. Add the preceding argument to the following argument to form the current argument.
- (hyphen)	Arithmetic operator used in expressions. Subtract the following argument from the preceding argument to form the current argument. Also used as a unary operator to indicate a negative value.
, (comma)	Argument separator. Separates the number of a relocation register from a relative location to specify a relocatable address.
* (asterisk)	Radix-50 separator used in constructing Radix-50 words (see Section 7.3.5).
. (period)	Current location indicator. Causes the address of the last explicitly opened location to be used as the current address for ODT operations.
; (semicolon)	Argument separator. Separates multiple arguments, which allows an address expression or ODT register value to be identified.
RET (RETURN command) or k RET	Command that closes the currently open location and prompts for the next command. If RETURN is preceded by k, the value k replaces the contents of the currently open location before it is closed.

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
LF (LINE FEED command) 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 LINE FEED 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 Program Counter (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.
> 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 /, LINE FEED, or ^ command. If the currently open location was not opened by a _, @, or > , then < simply closes and reopens the current location. If < is preceded by k, the value k replaces the contents of the currently open location before it is closed.
$n	Expression that represents the address of one of eight general registers, where n is an octal digit identifying R0–R7.

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
$x or $xn	Expression that represents the address of one of ODT's internal registers, where x is one of the following alphabetic characters, and n is one octal digit. Registers exist within ODT in the following order: S Processor Status register (hardware PS) W Directive Status Word (DSW) register for the user's task A Search argument register M Search mask register L Low memory limit register H High memory limit register C Constant register Q Quantity register F Format register X Reentry vector register nB Breakpoint address registers nG Breakpoint proceed count registers nI Breakpoint instruction registers nR Relocation registers nV Synchronous System Trap (SST) vector registers nE SST stack contents registers nD Device control LUN (logical unit number) registers
" or a"	Word mode American Standard Code for Information Interchange (ASCII) operator. Interprets and displays the contents of the currently open (or the last previously opened) location as two ASCII characters, and stores this word in the quantity register ($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.

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
% 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.
\ 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, DEL, 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.
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 $C (constant register).
D	Command that accesses data space. After this command is issued, ODT interprets all references to locations as referring to the data space of the task.
E or kE or m;E or m;kE	Command that searches memory between the address limits specified by the low memory limit register ($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_8 for the E command.) Such references may be equal to, PC-relative to, or a branch 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.

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
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 by following these steps: sets BPT instructions in or restores BPT instructions to all breakpoint locations in the task image; restores the Processor Status Word (PSW) 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 $7 before proceeding as described above.
I	Command that accesses instruction space. After this command is issued, ODT interprets all references to locations as referring to the instruction space of the task.
K	Command that, using the relocation register whose contents are equal to or closest to (but less than) the address of the currently open location, computes the physical distance (in bytes) between the address of the currently open location and the value contained in that relocation register. ODT displays this offset and stores the value in the quantity register ($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 ($Q).
a;nK	Command that computes the physical distance (in bytes) between address a and the value contained in relocation register n. ODT displays this offset and stores the value in the quantity register ($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).

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
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 continues with program execution from the current location and stops at the breakpoint only after encountering it the number of times specified by integer k.
Q	Quantity register indicator. Represents the contents of register $Q (quantity register).
R	Command that sets all relocation registers to the highest address value, 177777_8, so that they cannot be used in forming addresses.
nR	Command that sets relocation register n to the highest address value, 177777_8, so that it cannot be used in forming addresses.
a;nR	Command that sets relocation register n to address value a. If n is omitted, ODT assumes relocation register 0.
S or nS	Command that executes one instruction and displays the address of the next instruction to be executed. If n is specified, ODT executes n instructions and displays the address of the next instruction to be executed.
U	Command that sets the current mode of ODT to user mode.
V	Command that enables ODT's handling of all SST vectors, and writes the addresses of ODT's trap entry points into the table used by the SVDB$ Executive directive. (See Table 5-2 for a discussion of the SST vector registers and the $nV/ command.)

Table 2–3 (Cont.): ODT Operators and Commands

Format	Meaning
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.
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.

Chapter 3
Controlling Program Execution with ODT

When you run a task image into which the On-Line Debugging Tool (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 5.2.

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.

Breakpoint Command Format

a;nB

Parameters

a

Specifies 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

Specifies 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 that use separate instruction and data space breakpoints, always refer to instruction space, regardless of which space you referred to when you set the breakpoints. When a debugging session begins, you automatically access instruction space. You access data space by entering the D command; you return to instruction space by entering the I command.

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) by 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. The a;nB command allows you to specify a new address expression.

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, a 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 Task 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 (PC) 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, IL, IO, MP, OD, TE, 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 Section 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 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 by 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

You can use the S (Step) command as another method for executing a task in stages. 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 (trace 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
```

3.6 Setting Breakpoints in Overlay Segments

When debugging overlaid tasks with ODT, you cannot set a breakpoint in an overlay segment that has not been loaded into memory. ODT sets the breakpoints in memory only. It does not change the disk image of the task being debugged. So, if you set a breakpoint in the range of memory addresses that the overlay will occupy before it has been loaded into memory, the breakpoint will be overwritten when the overlay is actually written. In the same way, breakpoints in an overlay segment are lost once the segment is overwritten by a different overlay segment.

You can, however, set a breakpoint in the overlay run-time routines so that you receive control after an overlay has been loaded, but before it is executed. If you set a breakpoint at global symbol $ALBP2 in your task, you will receive control each time an overlay is loaded into memory from disk and before the overlay is executed. Doing so enables you to put breakpoints in the overlay segment.

With this method, your task will breakpoint on every overlay load. Because you will probably only be interested in debugging one particular overlay segment, you must keep track of the sequence and number of overlay loads to know when the overlay segment that you are interested in has been loaded. Or, if there is a unique location in the overlay segment that you are interested in, you can merely examine the corresponding memory location in your task each time an overlay is loaded until the contents of the memory match the value in the overlay segment. Then you know that the segment has been loaded.

A breakpoint at a second location in the overlay run-time routines, $ALBP1, will give you control every time control is transferred to an overlay that is already resident in memory and that does not have to be loaded from disk. This breakpoint location is generally less useful than the first.

Because the global symbols $ALBP1 and $ALBP2 are defined in the system library routine, they are not included in a task's map file by default. To include these global symbols in your map file, build your task with the Task Builder (TKB) /MA switch (if your CLI is MCR) or with the LINK command /SYSTEM_LIBRARY_DISPLAY qualifier (if your CLI is DCL).

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. To display a location's contents, use any of the commands described in Sections 4.3 to 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_8 is substituted for the value 123456_8 in the location represented by the address expression 2,0. The value 177426_8 (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. Use the commands described in Section 4.11 to display the contents. 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 (by pressing the RETURN key). This command has no effect on ODT when no location is open.

Entering the RETURN command 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 to 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 American Standard Code for Information Interchange (ASCII), or you can open and display words in Radix-50. These modes are described in Section 4.11.

4.3.1 Opening Word and Byte Locations at a Specified 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-order 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 mode 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 by using an even-numbered address as the value a in the a/ command (or the a" or a% commands, as described in Section 4.11).

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 4.11.

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. On VT100-series terminals, a line feed is generated by pressing the LINE FEED key.) 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 127460
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 (PC), 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 Opening the Directive Status Word

Use the ODT internal register W to examine the Directive Status Word (DSW). Normally, task memory location 46 contains the DSW. However, when using ODT, location 46 may reflect the status from the last directive issued by ODT, and not the last directive issued by the task. ODT register W always contains the correct DSW for the task.

4.11 Using Different Output Modes

The examples in the previous sections showed 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.

4.11.1 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
```

4.11.2 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 (*), as described in Section 7.3.5.

Chapter 5
Using Registers in ODT

The On-Line Debugging Tool (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 to $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 American Standard Code for Information Interchange (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 you can set 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, as 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. It is 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. It is initialized to 0.
$M	Search mask register. You set this register to a word search mask by opening with the / operator, or you can set 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. It is initialized to -1, 177777_8.
$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 register stores the Processor Status Word (PSW) (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 register contains the Directive Status Word (DSW) of the task, which indicates the success or specific cause of rejection of the most recently executed directive. The contents of this register are maintained across breakpoints. Always use this register to examine the DSW. The contents of memory location 46 do not reflect the correct DSW for the task. See the *RSX-11M-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 register contains the 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_8.
$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 (SST) occurs. Stack contents depend on the type of trap taken, as explained in the *RSX-11M-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.
$nR	0–7	Relocation register n. This contains the relocation bias of a relocatable object module, which enables 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_8.
$nV	0–7	Synchronous System Trap (SST) vector register n. This contains the entry-point address of the ODT routine for handling a SST. 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:

Table 5–2 (Cont.): ODT Register Sets

Register	Range of n	Function
	Register	**SST Vector**
	$0V	Odd address reference in word instruction (also, on some processors, illegal instruction executed)
	$1V	Memory-protection violation
	$2V	T-bit (trace 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_8. 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 you can set them 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_8, so that it is no longer useful in constructing address expressions. If you omit n, all relocation registers are set to 177777_8.

5.2.2 The Reentry Vector Register

If you have fixed a task in memory (see the FIX command in the *RSX-11M-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

The On-Line Debugging Tool (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, which contains the high memory limit for a search, fill, or list operation; and $L, which contains 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 $177,777_8$, 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. To set this value, open register $A as a word or byte location and change its contents, or specify 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 device control LUN register $2D contains the event flag number, which is 34_8 by default. Unlike the values for registers $0D and $1D, the TKB option GBLPAT cannot override the value for register $2D. See Section 1.2.4 for more information on changing the values of the registers $0D and $1D.

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 $177,777_8$, 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. To set this register, use 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 to 1776 are set to 0, and byte locations 2000 to 2777 are filled with American Standard Code for Information Interchange (ASCII) spaces (40_8):

```
_1000;1R
_2000;2R
_3000;3R
_$L/000000 1,0 [RET]
_$H/000000 2,-2 [RET]
_OF
_$L/001000 2,0 [RET]
_$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 format shown next.

Listing Memory Command Format

n;a;kL

Parameters

n

Specifies 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

Specifies 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

Specifies 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 4.11.

The following example shows the output displayed on the output device in response to various listing commands. Note in this case that the question mark (?) displayed in response to the ' command is not 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

The On-Line Debugging Tool (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; and 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 period (.) (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 program counter (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 instruction or data space.

7.3 Evaluating Expressions

You can evaluate expressions during your debugging session by 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 current location 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 4.11.2, 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, or location calculation). You cannot use any other symbol (C, Q, or .) 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 to Z and 0 to 9) plus three special characters: dollar sign ($), period (.), 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

Table 7–1 (Cont.): Numeric Equivalents of Radix-50 Characters

Radix-50 Character	Numeric Equivalent	Radix-50 Character	Numeric Equivalent
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 4.11.2):

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

Chapter 8
Additional Debugging Aids

The Task Builder (TKB) 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 8.1 shows how you specify other debuggers to TKB for the three environments described in Section 1.2. Section 8.2 describes the Trace program, a debugging aid available on your system.

8.1 Accessing Other Debugging Aids

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

8.1.1 MCR Command Line

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

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

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

8.1.2 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 shown in the following example:

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

Because 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.

8.2 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 is an object module that you specify to TKB when you build your task, as described in Section 8.1. It is located in the DEBUG.OLB library in directory [1,1] on the system disk, with the name TRACE.OBJ. To extract TRACE.OBJ from the library DEBUG.OLB, first set your default directory and protection User Indentification Code (UIC) to [1,1] on the system disk. Then, if your command line interpreter (CLI) is MCR, issue the following Librarian Utility Program (LBR) command:

```
>LBR TRACE.OBJ=DEBUG.OLB/EX:TRACE RET
```

If your CLI is DCL, issue the following LIBRARY command:

```
$ LIBRARY/EXTRACT/OUTPUT:TRACE.OBJ DEBUG.OLB TRACE RET
```

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.

8.2.1 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, which represents the contents of the following registers:

- Current relative program counter (PC)
- Current PC
- Next PC
- Processor Status Word (PSW)
- Directive Status Word (DSW)

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:

$1–6_{10}$ R0 to R5

7_{10} Stack pointer

8_{10} 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
```

8.2.2 Bias Values and Ranges

You can use the GBLPAT TKB option to specify the following:

- The bias value to be used in determining the relative PC
- The range or ranges of task locations to be traced

8.2.2.1 Specifying a Bias Value

To specify a bias value for relative PC calculation, enter an option line in the format shown next in response to the TKB prompt.

Format

GBLPAT=segname:.BIAS:value

Parameters

segname

Specifies the name of the task's root segment.

value

Specifies 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.)

8.2.2.2 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 format shown next in response to the TKB prompt.

Format

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

Parameters

segname
Specifies the name of the task's root segment.

low1 . . . lown
Specifies the low addresses, relative to the bias value, of ranges to be traced.

high1 . . . highn
Specifies 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

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

A.1 Input Errors

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

ODT responds 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
- Reference to a nonexistent register—for example, $20
- Reference to supervisor space by a nonprivileged user
- 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.

ODT does not tell you what error has caused it to display the question mark. However, an error sometimes causes it 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 repeats its prompt but does not display a question mark.

A.2 Task Image Error Codes

As described in Table 5-2, eight Synchronous System Trap (SST) vector registers are used to contain pointers to error-handling routines. Upon detecting an error condition, ODT activates the appropriate routine and displays an error message. This message has the form cc:k, where cc is a 2-character error code and k is the location at which the error occurred. ODT displays the location as a relative address if there is a relocation register containing a base address less than the absolute address of the location.

The following examples are error messages from a debugging session:

```
MP:007414

OD:1,003507
```

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

BE **Explanation:** Breakpoint instruction executed at unexpected location. The address of the breakpoint instruction does not match the contents of any register, $0B to $7B.

User Action: Examine your code to determine why the unexpected breakpoint occurred; then continue with the P command.

EM **Explanation:** Invalid EMT instruction executed. Only EMT 377 and EMT 376 (for a privileged task) are allowed by the Executive for execution of Executive directives. Normally, vector address 30 is used for this trap sequence.

User Action: If you want to use an EMT trap handler that you have written, set SST vector register 5 ($5V) to the appropriate vector address.

FP **Explanation:** Floating-point instruction error. One of the following has occurred: division by zero; illegal Floating Op Code; flotation overflow or underflow; or conversion failure.

User Action: Check your code for sequences that may have caused one of these conditions.

IL **Explanation:** Reserved or illegal instruction executed. The task tried to execute a nonexistent instruction, an Extended Instruction Set (EIS), or Floating Point Processor (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 program counter (PC) contained an odd address when trying to access a word in memory. Also, on some processors, execution of an illegal instruction.

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

TE **Explanation:** T-bit (trace 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 Processor Status Word (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.

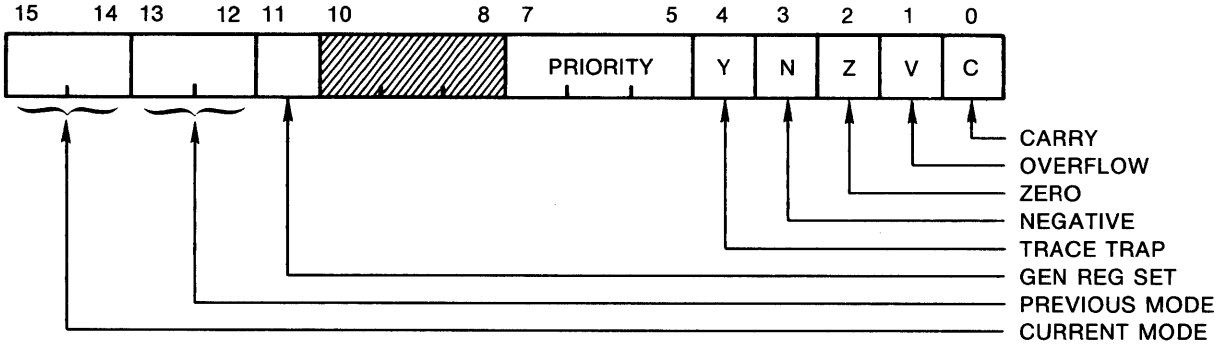
Appendix B
Processor Status Word

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

The T-bit (trace bit, bit 4) can be set or cleared under program control. When set, a processor trap will occur through location 14 upon completion of the current user instruction, and a new PSW will be loaded. The T-bit is especially useful in debugging programs, because it provides an efficient means for stepping through the task one instruction at a time. ODT uses the T-bit to execute instructions when you are stepping through your program with the S command, as described in Section 3.5.

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

N=1 If the result was negative

Z=1 If the result was zero

V=1 If the operation resulted in an arithmetic overflow

C=1 If the operation resulted in a carry from the most significant bit

Index

A

B

C

D

E

F

N

O

P

Q

R

S

T

U

V

W

X

Z

RSX-11M-PLUS and Micro/RSX
Debugging Reference
Manual
AA-JS09A-TC

READER'S COMMENTS

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

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

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

What kind of user are you? ____ Programmer ____ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

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

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

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

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