Data General

Software Documentation

RDOS/DOS Debugging Utilities

RDOS/DOS
Debugging Utilities

069-400020-01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 069-400020

Printed in the United States of America
Revision 01, September 1984

NOTICE

RDOS/DOS
Debugging Utilities
069-400020

Revision History:

Original Release - July 1983

First Revision - September 1984

CONTENT UNCHANGED

The content in this revision is unchanged from 069-400020-00. This revision changes only printing and binding details.

Preface

The five parts of this manual each describe a separate debugging utility:

Part I explains the disk editor utility (DSKED), which provides powerful tools for editing a disk.

Part II describes the Symbolic Editor (SEDIT), another disk file editing utility that allows you to examine, analyze, and modify the contents of disk files. SEDIT works on random or contiguous files produced within the Data General Real-Time Disk Operating System (RDOS) or the Disk Operating System (DOS).

Part III covers the Symbolic Debugger, which allows you to detect, locate, and remove mistakes from a program. This debugger operates on mapped and unmapped RDOS systems, or the Real-Time Operating System (RTOS).

The RDOS symbolic debuggers closely resemble the RDOS symbolic editor, although SEDIT cannot set breakpoints or run a program.

Part IV explains the ENPAT utility, which performs syntax checking on all data input.

Part V describes PATCH, a utility that enters data into a patchfile to be used by the PATCH.SV utility.

Reading Path

You should be familiar with the *RDOS/DOS Command Line Interpreter* (DGC No. 069-400015) and the appropriate reference manual for your operating system—*RDOS System Reference* (DGC No. 093-400027), *RTOS System Reference* (DGC No. 093-000056), and *DOS Reference Manual* (DGC No. 093-000201-03)—when you begin working with this manual.

Manual Organization

We have organized this manual as follows:

Chapter 1 introduces the DSKED utility. This chapter contains information on invoking the utility, and on keyboard calculations allowed in DSKED.

Chapter 2 presents all of the display and escape commands in DSKED. It describes escape commands in detail. The format for each command is discussed and examples are given throughout the chapter.

Chapter 3 introduces the SEDIT utility. This chapter contains information on invoking the utility and on two of the most common error messages.

Chapter 4 describes the operation of SEDIT. The command format and the individual location and display commands are described in detail. Also included in this chapter is an explanation of the editing of save files, overlay files, text files, and the symbols used in each file.

Chapter 5 explains the search and display commands. Also explained are the increment registers used with each command.

Chapter 6 presents the enabling, disabling, and managing of symbols.

Chapter 7 introduces the Symbolic Debugger utility. This chapter contains information on invoking and loading the debugger. Its command format and two common error messages are also presented.

Chapter 8 describes the memory locations, special registers, accumulators, and how each are used in the Symbolic Debugger.

Chapter 9 explains program restarts and breakpoints in the Symbolic Debugger.

Chapter 10 describes the Search command.

Chapter 11 explains how to enable and disable global and local symbols.

Chapter 12 describes the different output format commands.

Chapter 13 explains how to save a debugged program.

Chapter 14 introduces and describes the ENPAT utility.

Chapter 15 introduces and describes the PATCH utility.

Appendix A lists a command summary for each utility.

Appendix B lists and describes the error messages for each utility.

Related Manuals

The following manuals are part of a series of books that document RDOS, DOS, and RTOS.

Introduction to RDOS (DGC No. 069-400011) describes the fundamentals of using RDOS and summarizes the features, utilities, and capabilities of the operating system.

Guide to RDOS Documentation (DGC No. 069-400012) describes all of the books that comprise the revised documentation set for RDOS and lists the previous books that each replaces.

How to Load and Generate RDOS (DGC No. 069-400013) explains how to load, generate, and maintain RDOS. Instructions are provided for preparing hardware, program loading, initializing disks, installing the bootstrap root and starter system, tailoring, and system backup, among others.

How to Generate Your DOS System (DGC No. 093-000222-01) explains how to generate and maintain DOS. Instructions are provided for preparing hardware, initializing disks, installing the bootstrap root and starter system, tailoring, system backup, and optimization, among others.

RDOS/DOS Command Line Interpreter (DGC No. 069-400015) discusses the user interface with the operating system. It covers the Command Line Interpreter (CLI) features and command mechanisms, as well as instructions on how to use CLI commands. It also presents features and operation of the Batch monitor, a CLI utility.

RDOS/DOS Text Editor (DGC No. 069-400016) documents how to load, use, and operate the single-user Text Editor (Edit) or Multi-user Text Editor (Medit) to create and edit text files.

RDOS/DOS Superedit Text Editor (DGC No. 069-400017) introduces the commands and concepts of the Superedit Text Editor (Speed), which offers many powerful features for editing text.

RTOS Reference Manual (DGC No. 093-000056) presents specific information on RTOS tasks, system calls, structures, and user device driver implementation for the system programmer.

RDOS System Reference (DGC No. 093-400027) describes RDOS system features, system calls, and user device driver implementation for assembly language and high-level language programming.

DOS Reference Manual (DGC No. 093-000201-03) discusses all features of DOS, including system control, memory management, and system calls, for system programmers.

RDOS/DOS User's Handbook (DGC No. 093-000105-04) summarizes the commands, calls, and error messages of RDOS and DOS, the Command Line Interpreter (CLI), the text editors (Edit, Medit, and Speed), the Batch monitor, and utility programs.

RODS/DOS Assembly Language and Program Utilities (DGC No. 069-400019) details the Extended Assembler (ASM), Macroassembler (MAC), Extended Relocatable Loader (RLDR and OVLDR), and Library File Editor (LFE) utilities that aid in programming.

RDOS/DOS Sort/Merge and Vertical Format Unit Utilities (DGC No. 069-400021) offers functional information on Sort/Merge (RDOSSORT), which helps you manipulate records in data files, and the Vertical Format Unit (VFU), which enables you to define data channel line printer page formatting.

RDOS/DOS Backup Utilities (DGC No. 069-400022) presents the features and operation of the utilities that perform disk and tape backup. These are BURST/TBURST, DBURST/MBURST/RBURST, DDUMP/DLOAD, FDUMP/FLOAD, and OWNER.

Typesetting Conventions

We use the following conventions for command formats in this manual:

COMMAND *required* [*optional*] . . .

Where	Means
COMMAND	Enter the command or its accepted abbreviation as shown. Uppercase letters indicate the command mnemonic.
required	Enter an argument such as a filename. Lowercase italic letters indicate this argument. Required arguments also appear as: $required_1$ | $required_2$ In this case, you may choose between the arguments. Do not type the vertical bar; it merely separates the choices.
[*optional*]	Brackets mean that you have the option of entering the argument indicated in lowercase italic letters. Command switches also appear in this format. Do not include the brackets in your code; they only set off the choices.
. . .	Repeat the preceding entry or entries. The explanation indicates exactly what to repeat.
. . .	Where this symbol appears, the process has continued without incident and you may now take the next action described.

In examples of dialogue, we use:

THIS TYPEFACE TO SHOW YOUR ENTRY

and

THIS TYPEFACE TO SHOW SYSTEM RESPONSES

Additionally, we use certain symbols in special ways:

Symbol	Means
⟨CR⟩	Carriage Return. Press the CR key on your keyboard. **NOTE:** If you have a D100, D200, or G300 terminal, you should press the New Line key (NEW LINE) on your keyboard instead.
␣	Include a space at this point. We use this to clarify command lines in some cases. Normally, you can see where you should put spaces.
CTRL-	Depress and hold the Control key (CTRL) while you press the character that follows CTRL-.
⟨NL⟩	New Line. Press the NEW LINE key on your keyboard.
R	RDOS/DOS Command Line Interpreter prompt.

All numbers are decimal unless we indicate otherwise; for example, to indicate octal 35, we use 35_8.

The keys defined as DEL and RUBOUT perform the same function. Depending on the console you are using, you will find one of these keys on your keyboard. In this manual, we use DEL to represent that function.

The up arrow symbol (↑) is also executed by different keys, depending on your console. You execute it by pressing either SHIFT-N or SHIFT-6. In this manual, we reference SHIFT-6 to execute the up arrow symbol.

We welcome your suggestions for the improvement of this and other Data General publications. To communicate with us, use the postpaid comment form at the end of this manual.

Table of Contents

Tables

Figures

Index

Part I

Disk Editor Utility

Chapter 1

Disk Editor

DSKED is a disk editing utility that enables you to examine, modify, and rearrange disk words. It can search for a specific word, fill different parts of the disk with a word, and display any part of the disk. The utility offers seven different output formats and a choice of number bases ranging from 2 to 16.

DSKED is designed for the following disk types:

- 4047A and 4047B Disk Cartridge Subsystem,
- 4048A Disk Pack Subsystem,
- 4057A Disk Pack Subsystem,
- 4231A Disk Pack Subsystem,
- 4234A Series Disk Cartridge Subsystem,
- 4237/4238 Disk Cartridge Subsystem,
- Models 6001 through 6008 Fixed-Head Disk System,
- 6030 Series Diskette Subsystem

Invoking DSKED

DSKED is supplied as DSKED.SV and runs as a standalone utility. It is loaded by the Binary Loader, TBOOT, MCABOOT, CBOOT, or BOOT, depending on its medium and the equipment running it. After loading, DSKED requests the disk type and the physical disk unit; valid responses put the program in command mode.

The first messages from DSKED are:

DISKEDIT

DISK DRIVE MODEL NUMBER?

The valid response is the disk drive model number and a Carriage Return (⟨CR⟩). Drive model numbers include:

NOVA and ECLIPSE

4047, 4048, 4057

4231, 4234, 4237/4238

60000, 60001 through 6008

6030, 6045, 6061, 6063

6064, 6067, 6070, 6097, 6099

6103, 6122, 6160, 6161

6225, 6227, 6234

MicroNOVA Disks

6060, 6061, 6095, 6096

6101, 6105

6220, 6222

UNKNOWN DRIVE TYPE

After a valid response, DSKED asks:

DISK UNIT?

The response must be the RDOS-supported device mnemonic for the disk, for example, DK0, DP7, or DP5F. An incorrect entry evokes the error message:

ILLEGAL DISK UNIT DECLARATION

The following message is output for fixed-head disks only:

DISK CONTAINS XXXX (n) BLOCKS

This is a convenience message. DSKED outputs the number of disk blocks in decimal—XXXX—and, within parentheses, in octal.

DSKED indicates that it is ready for editing commands by outputting a period (.).

Keyboard Calculations

The value of any field in a command can be calculated with addition (+), subtraction (−), or exclusive ORing (!), and all results are shown in octal. These calculations can be

specified in both the disk address and displacement portions of DSKED's escape and local command formats.

Parentheses can be used within individual calculations and are evaluated from left to right.

Within a calculation, a period can have two functions. When used as a character, it indicates the current location value; after a number, it indicates decimal notation for that number. For example,

```
14+.=
```

results in 21_8 when the current location is 5.

A period followed by an equal sign (.=) outputs the current location value:

```
14.+.=
```

and results in 23_8. The first period indicates a decimal number.

The following command:

```
(10.+2)+(17-7):4+5$C
```

calculates a disk address of 24_8 and a displacement of 11_8.

Calculation arguments can have the following forms:

- *Double-precision octal or decimal number* (decimal numbers must be followed by a period [.]). Represents any number.
- *Single-precision number/single-precision number*. Represents a double-precision disk address as two words.
- *"Charactercharacter."* Specifies one or two characters packed left to right in a word, with null fill. For instance, "" equals zero, *"character"* equals *character* * 400; and *"charactercharacter"* equals *character * 400 + character*.
- *Number; filename*. Returns the hash number of a filename. *Number* is the frame size; *filename* is an RDOS filename.
- *Head,sector,cylinder*. Indicates an address on moving-head disks only. The entries are separated by commas.

Chapter 2

DSKED Commands

This chapter describes the format of the commands used with the DSKED utility. There are two types of commands: display commands that display disk words in a specific format, and escape commands that affect the functions of the disk. Included in this chapter is a section on output format commands, which are classified as escape commands.

Command Format

DSKED commands consist of expressions, followed by a valid command code. There are two command types: display and escape. Display commands affect specific locations, while escape commands perform general disk functions. There is only one difference in the execution of the two types of commands. When you execute an escape command, you must press the ESC key ($) before you enter the command code.

The format of the escape command is as follows:

[*disk address*]:[*displacement*] *$ command code*

Display command lines do not require the use of the $ character. The format of display commands is as follows:

[*disk address*]:[*displacement*] *command code*

where:

disk address is the block number being accessed. When the diskaddress is missing, DSKED assumes the last block number accessed.

displacement is the displacement in the block at disk address. When the displacement is missing, the program assumes the last displacement used.

$ (produced by pressing the ESC key) precedes escape command codes. Escape commands perform general disk functions, such as dumps and searches. Display commands do not use an ESC.

command code is a single character that represents a command. A command code can also terminate a command.

Display Commands

Display commands display a disk word in a specific format. They are entered in the command code portion of the format and must not be preceded by an escape. A summary of display commands follows.

Command	Description
/	Opens current word and displays contents in current format.
'	Displays the last word or expression in ASCII format. (apostrophe)
←	Displays the last expression in half-word format as two octal numbers.
=	Displays the last expression in word format in the current number base.
*	Displays the last expression as an octal word.
⟨CR⟩	Stores the result of an expression in the currently opened word. If no word is currently opened, this is a no-op.
↓	Stores the result of an expression in the currently opened word and displays the next word.
↑	Stores the result of an expression in the currently opened word and displays the previous location.
$	Enters escape command mode; the character following the $ is a command.

Table 2.1 Display commands summary

Escape Commands

Escape commands affect general functions on the disk. These commands must be preceded by an Escape ($) in command lines; each is a terminating character. Table 2.2 describes each escape command.

Command	Description
M	Opens the mask register for display and modification.
W	Opens the word register for display and modification.
N	Opens the number register for display and modification. This register defines the output format of numbers.
H	Opens the output register for display and modification. The output register directs the output of search and dump commands to the teletypewriter if zero, or the line printer if nonzero.
J	Opens the increment register for display and modification.
S	Searches the disk for a specified value.
D	Dumps specified disk locations.
F	Fills specified disk locations with a word from the word register.
R	Forces a block to be read into core. Any blocks already in core are not written out, even though they may have been modified.
O	Forces block to be output to disk.
A	Prints block number currently in core.
B	Changes the name (address) of the block in core. The $O command can then write the block to the address specified in $B.
C	Calculates expression and prints result. If used alone, this command prints current location.
U	Restarts DSKED at the query: *DISKEDIT DISK DRIVE MODEL NUMBER?* and resets registers to startup values.
Z	Writes any modified block in core and halts program. The CONTINUE switch at the computer console restarts DSKED at the query: *DISKEDIT DISK DRIVE MODEL NUMBER*

Table 2.2 Escape commands summary

The remaining escape commands are output format commands. They enable you to specify and change the form of your output. Once a format is specified, it will apply to all future printouts until it is changed. Like all other escape commands, output format commands must be preceded by an Escape in each command line.

The output format commands are described in Table 2.3.

Command	Description
=	Outputs numbers in word format.
'	Outputs numbers in ASCII format. (apostrophe)
←	Outputs numbers in half-word octal format.
*	Outputs numbers in word format. The asterisk sets the number register to 8 for octal output.
:	Outputs the disk block number(s) in double-precision octal format.
\	Outputs the disk block number(s) as two octal words.
,	Outputs the disk block number(s) in head, sector, cylinder format (fixed-head disks only). (comma)

Table 2.3 Output format commands summary

Table 2.4 offers a few examples of the commands with explanations of their operations.

Command	Action
.$*	Outputs in octal word format.
.0:0/*040501 'AA*	Displays a disk word and examines it in ASCII.
.$'	Displays words in ASCII format.
.0/*AA* = *040501* ← *101 101*	Displays disk word in ASCII, then in half-word format.
.$ ←	Displays word in half-word format.
.0/ *101 101* ⟨CR⟩	Displays word in half-word format.
.$=	Displays in word format.
.0/*040501* ⟨CR⟩	Displays word in current format.
.$:	Outputs in double-precision format.
.15:45 + 2$C	Calculates disk octal address and displays result.
000015:047 .$,	Outputs disk address in head, sector, cylinder format.
.5:45$C 0,5,0:45	Displays disk address.

Table 2.4 Escape command examples

Escape Command Descriptions

In this section, each escape command is discussed in greater detail.

Word ($W) and Mask ($M) Registers

DSKED uses both the mask and word registers to search the disk. For searches, set the word register to represent the value sought on the disk. The mask register is set to mask specific bits of words (or to mask no bits).

As the search proceeds, the value of each location is ANDed with the mask register and then compared with the contents of the word register. If they match, the location and its value are printed out. When DSKED starts, the word and mask registers each contain zero, which will print all disk locations in a search command.

The word register is opened with the command

$W

The mask register is opened with the command

$M

An example of each command follows.

$W *000000* 1500 ⟨CR⟩

This command opens the word register and stores 1500.

$M *000000* 177700 ⟨CR⟩

This command opens the mask register and masks bits 0 through 9.

Number Register ($N)

The number register determines the base in which numbers will be printed when displayed in the word format. This register also defines the output format of numbers, and is coded as follows:

Code	Description
100000	Treats the number as signed.
4000	Suppresses leading zeroes.
base	Defines the base of the number on output; it must be an integer from 2 to 16.

By default, the number register is set to zero, which prints out the register contents in octal. When you set the contents of the number register to a base number, that number becomes the output base number. Only bases 2 through 16 are valid. The number register is opened for examination and modification with the command

.$N

Some examples of the command include:

.$N *000010* 10. ⟨CR⟩

Opens number register and sets number base to 10 (note the period: 10.)

.$N *00010* 16. ⟨CR⟩

Opens number register and sets number base to 16.

.$N *0010* 10 ⟨CR⟩

Opens number register and sets number base to 8. (No period follows base specification).

.%N *000010* 40000+10 ⟨CR⟩

Opens number register, suppresses leading zeros, and sets base at 8.

.$N 40010 ⟨CR⟩

Opens and examines the number register.

Increment Register ($J)

DSKED uses the increment register to search the disk. You can specify a value in this register to adjust the increment between search locations. The value in this register can also be used in fill and dump commands. If the value is zero or negative, then one becomes the increment.

The increment register is opened for examination and possible modification with the command

$J

An example of the command is

.$J *000000* 12 ⟨CR⟩

This command opens the increment register and stores 12.

Search Command ($S)

This command searches all or part of a disk for a specified value and prints the disk locations that contain the value.

The following is the procedure for performing a search:

1. Set the mask register to the mask desired for the search.
2. Set the word register to the value sought.
3. Set the increment register to the desired increment value.
4. Specify the search command. The search command ($S) searches the entire disk unless the search command line specifies otherwise.

The search command combines the following information to find a match:

mask-word AND disk word = word for match condition

where:

mask is the contents of the mask register.

word is the contents of the word register.

disk word is a word from the disk.

AND represents logical AND.

When DSKED starts, both the word register and the mask register contain zero, and the search increment is one:

When a match is made, the location is printed. After a match, DSKED adds the value in the increment register to the match location and continues searching. Comparing and incrementing continues until all of the specified disk area has been searched.

The search command can restrict a search to specific portions of the disk and is executed with the command

$S

The next three commands search different parts of the disk. The command

disk address:displacement$S

searches the disk from the beginning to the specified address and displacement.

The command

disk address:displacement⟨$S

searches from the specified address and displacement to the end of the disk.

The command

disk address:displacement⟨*disk address:displacement*$S

searches between the two specified addresses and displacements.

The following example shows the procedure for finding all words containing one on the disk.

```
.$W 000000 1⟨CR⟩
.$M 000000 -1⟨CR⟩
.$J 000000 ⟨CR⟩
.$S ⟨CR⟩
000000:102 \ 000001
001020:052 \ 000001
```

Sets the value of the word register to 1. Sets the mask register to −1, causing a default; the register compares all bits of the words. Sets the search increment register to 1 — the default value — and the $S command initiates a search. For each word that contains a 1, the system reports the full address and value, as shown at the end of the example.

Dump Command ($D)

This command dumps all or part of a disk to an output device. Unless you specify otherwise in the command line, the entire disk is dumped. The incremental value in the increment register can dump selected incremental locations. All dumped outputs include the disk address and all the data within the location. The command

$D

dumps the whole disk.

The command

disk address:displacement$D

dumps from the beginning of the disk to the specified address.

The command

disk address:displacement⟨$D

dumps all locations from the specified address and displacement to the end of the disk.

The command

disk address:displacement⟨*disk address:displacement*$D

dumps all locations between the two specified addresses and displacements.

An example of this command is

```
.1045:0⟨20$D ⟨CR⟩
001045:000/005126 000023 000000 004010
001342 000000 000000 003524
001045:010/000000 003013 002041 000000
000000 006551 000000 003311
001045:020/005322
```

This command dumps words 0 through 20 of block 1045.

Fill Command ($F)

The fill command writes a word from the word register into a specific location. Select the fill word and place it in the word register. The $F command writes the word to the specified locations. The incremental value in the increment register can fill incremental locations with this word. After specifying the fill command, you are asked:

REALLY DO IT?

The system waits for a valid response, either Y (yes) or N (no). Y writes the specified word into the specified location; N aborts the command.

The fill command

$F

fills the entire disk with the value in the word register.

The command

disk address:displacement$F

fills the disk from the beginning to the specified address with the value in the word register.

The command

disk address:displacement⟨$F

fills all locations from the specified address to the end of the disk with the value in the word register.

The command

disk address:displacement⟨*disk address:displacement*$F

fills all locations from the first address and displacement through the ending disk address and displacement with the value in the word register.

An example of the Fill command is

```
.$W 000000 -1
.$J 000000
.$F
REALLY DO IT?      Y
```

This command sequence sets the word register, and opens the increment register for display. The fill command then writes a −1 to all blocks of the disk.

Force Block to Be Read into Core Command ($R)

This command forces a user-specified disk block to be read into core. It is useful when the original (unedited) version of the previously requested disk block is required. The edited copy of the block is eliminated when the original version is read into core.

The changes made in a block are not written onto the disk until a different block is accessed, or until a $O or $Z command is entered.

After the original version of the block is in core, editing can continue.

The format of the $R command is

disk address:$R

For example, to force the current block to be read in again, type:

```
.$R
.
```

Force Block to Be Output to Disk ($O)

This command forces the block currently in core to be output to disk. The $O command is often used to clear core for a new block or to change the location of a block on the disk. After specifying the $O command, you are asked:

REALLY DO IT?

Valid responses are Y (yes) or N (no). A Y response outputs the block; an N response aborts the command.

The command to force the current block in core back to where it came from is:

$O

The command to force the current block in core to a new location is:

disk address:$O

Print the Block Number Currently in Core ($A)

This command prints the block number currently in core, using the selected block number format.

The format of this command is

$A

For example:

```
.5:01/000000 (CR)
.$A
000005
.102:53/126412 (CR)
.$A
.000102
```

This command sequence enters the block into core and displays the number of that block. It then accesses another block and displays its number.

Set the Current Core Block to a Different Address ($B)

Ordinarily, the block in core is sent back to its original disk address. The $B command specifies a different address for the core block; the $O command then writes the block to the address specified in $B.

The format of the $B command is

disk address:$B

For example:

```
.5:01/002300 (CR)
.$A
000005
.10:$B
.$A
000010
.01/002300
000010:0021/123456
```

The sequence above types the block into core, prints the block number, sets the block to a different destination, and prints the new destination.

Print Out Results of Expression Evaluation ($C)

This command calculates an expression and prints the results.

The command

$C

prints the current address and displacement.

The command

disk address:displacement$C

calculates the specified expression and prints the results.

Three examples follow.

```
.10:53/100245
.$C
000010:053
```

This sequence displays the contents of the disk location and evaluates the expression, using all defaults.

```
.5+5:50+30$C
000012:100
```

This command evaluates the expression and displays the result.

```
.$C
000010:053
```

The command causes the program to display the current address and displacement.

Restart the DSKED Program ($U)

This command restarts the program at the query

DISKEDIT DISK DRIVE MODEL NUMBER?

and resets all registers to their initial values.

The command format is

$U

Write Modified Core Block and Halt ($Z)

This command writes any current modified core block back to disk and halts the DSKED program. The CONTINUE console switch restarts the program at the question:

DISK TYPE

The format of this command is

$Z

Part II

Symbolic Editor

Chapter 3

Symbolic Editor

The Data General Symbolic Editor (SEDIT) allows you to edit disk file locations symbolically in any random or contiguous file created on an RDOS or DOS system. SEDIT is compatible with the RDOS debuggers and shares many commands with them. (To emphasize this similarity, we use the terms "location" and "address" interchangeably in this section.)

With SEDIT, you can display and modify locations in octal, or in instruction, system call, byte, or ASCII format. You can display whole blocks of words, and use special registers (described in Chapter 5) to find and modify locations. SEDIT works best on text files and on programs that have been loaded with a program symbol table. You *can* patch operating system files with SEDIT in octal, but you may find the ENPAT and PATCH utilities easier to use for this.

SEDIT is unique among Data General disk editors in that it searches the disk file for symbolic values. For example, take the instruction JSR @ 50 in a program. Using symbols, SEDIT might display this as

JSR @ .SAV

which tells you about the symbolic address for the JSR. SEDIT can also access the overlay file of a program from its save file.

Invoking SEDIT

You invoke the SEDIT utility with the command:

SEDIT *filename*

where *filename* is the name of any random or contiguous RDOS or DOS file.

SEDIT searches for the filename specified. If it cannot find the filename and you omitted an extension such as .SV, it will search for the filename with the .SV extension. Because SEDIT allows you to edit any overlay for any node from the save file, it is generally easier to edit an overlay file while editing the save file than to edit the overlay file directly.

When the utility finds the file, it opens it and displays a revision number and a period prompt. You can then enter SEDIT commands. If the file is read-protected or write-protected, SEDIT displays the error message:

PROBLEM READING FILE

and returns to the CLI when you try to read or modify the file.

When you have finished editing, type ESC Z (echoed as $Z) to terminate SEDIT and return to the CLI. For example:

```
R
SEDIT MYPROG ⟨CR⟩
SEDIT REV x.xx
.START/020451

$Z
DONE.
R
```

There are two global switches. The global switch /N tells the system not to search for a symbol table. Use this switch along with global switch /Z to edit a text file, or if the program lacks a symbol table. SEDIT may display the error message

NOT ENOUGH MEMORY FOR SYMBOL TABLE

This occurs when the program lacks a symbol table and you use /Z without /N.

The /Z switch indicates that this file starts at location zero. You *must* use this switch to edit a text file or stand-alone program file; do *not* use it for a conventional save file. Programs you can execute under RDOS begin at location 16, SEDIT assumes this if you omit /Z.

SEDIT Error Messages

SEDIT has two common error responses: U and ?. (Other SEDIT error messages are explained in Appendix B.)

The U message means that the characters you have typed are undefined. This can result from a simple mistake, or it

may mean that your program does not include a symbol table, or that you have not enabled this symbol.

The ? message means that your command syntax was wrong. Global commands must start on a new line, immediately after the prompt. This error can also occur if you type more than six octal numbers, or any value which exceeds $2^{16}-1$, such as 277777 or 88000, or if you use a space in an expression (for example, START □ + 10).

After it detects an error, SEDIT closes the current location (if open), and outputs a Carriage Return and its prompt. These errors do no harm.

Chapter 4

Operating SEDIT

SEDIT Command Format

All SEDIT commands have the format:

[*address*] COMMAND [*reply*] [*new-contents*]

The *address* can be a number, symbol, or period. Each SEDIT command is a single character. Commands that display an address are entered alone. These are called *local* commands. You enter other commands by pressing the ESC key, then the command letter; these are *global* commands. You can enter multiple local commands on one line, but you must start each global command on a new line. The *reply* is SEDIT's response—often the contents of address. You enter the *new-contents* (in any format described below) only if you want to modify the current contents. A Carriage Return tells SEDIT to accept the preceding *new-contents* (if any); it also closes the current address and displays a new prompt. SEDIT will output a Carriage Return if you make a syntactical mistake or enter an undefined symbol.

Here are three local commands in one line:

START/*020440*; *LDA 0 ER+10* LDA 0, ER+7 ⟨CR⟩

START is an address; the slash (/) is a local command that opens and displays the address symbolized by START; and *020440* is SEDIT's response, which indicates the contents of the address. The semicolon (;) command tells SEDIT to display the contents in instruction format (instead of octal); *LDA 0 ER+10* is SEDIT's translation of *020440* into instruction format. The last entry made by the user specifies the new contents for address START; the ⟨CR⟩ installs the new contents and closes the address.

An example of a global command is:

.$H *000000* 1 ⟨CR⟩

This opens the output register (described in Chapter 5) and stores 1 in it; this sends all output of search commands to the line printer. Global commands remain in effect until you change them.

Location Commands

Location commands open and close locations. When you open a location, it becomes the *current location*; this does not change until you access another location. The location commands are described in Table 4.1.

Command	Description
addr/	Opens address *addr* and displays contents.
addr!	Opens address *addr* and displays nothing.
↓ (NEW LINE)	Closes current address (if open) and opens and displays next address. (Use NEW LINE if your terminal lacks a LINE FEED key.)
↑ (SHIFT-N)	Closes current address (if open) and opens and displays preceding address.
⟨CR⟩	Closes current location (if any); returns prompt. The closed location remains the current location.
/	Closes current location (if open) and opens location specified by *contents* of current location. This is useful for pointers (for example, .LOAD/*LOAD/LDA 0 TEMP*).
.=	Displays current location in current radix.
.:	Displays current location symbolically.

Table 4.1 Location commands

When a location is open, SEDIT tries to insert whatever you type as the new contents for that location. When you do not want to modify the contents of an open location, be sure to close it with one of the commands above before typing new data. If you do type characters you do not want SEDIT to try and insert in this location, press DEL; this closes the current location without changing anything.

Display Commands

The display commands show the contents of locations in different formats and acts on the *last entry* that appears on the console. This can be an entry that you have typed in, or that SEDIT has displayed; or it can be an address or the

contents of an address. The display commands are described in Table 4.2.

Command	Description
=	Displays last entry in current radix (the default is octal). A .= command displays the current numeric location.
;	Displays last entry in machine instruction format.
'	Displays last entry in ASCII format; for example HI or ⟨n⟩⟨n⟩ if nonprinting characters.
\	Displays last entry in .SYSTM command format.
←(shift 0)	Displays last entry in half-word format.
&	Displays last entry in byte-pointer format.
:	Displays last entry in symbolic format.
*	Displays last entry in symbolic format with bit 0 zeroed.
.=	Displays current numeric location. (As with other utilities, you can enter a period to indicate the current location.)
.:	Displays current symbolic location. If no symbol is defined within 2000 locations, : displays location numerically.
n.=	Displays decimal number *n* in current radix (default is octal). For other conversions, see the section entitled Number Register in Chapter 5.
*n*xH=	Displays hexadecimal number *nx* in current radix. *n* must be a digit from 0 through 9; *x* can be any hex number(s) or hex letter(s).

Table 4.2 Display commands

Figure 4.1 shows examples of the local commands shown above.

SEDIT MYPROG ⟨CR⟩	Invokes SEDIT.
SEDIT REV x.xx	
.START/*020440* ⟨CR⟩	Opens address START and displays its contents (octal by default); closes location with ⟨CR⟩.
. .=*000452* ⟨CR⟩	Current address remains 452,
. .:*START* ⟨CR⟩	which is START symbolically.
.START! ⟨CR⟩	Opens address START without displaying contents; closes location. Address remains START.
.	The New-Line or Line Feed key closes current location (if open); then opens and displays next location.
START+1 126400 ↓	Closes START+1; opens START+2.
START+2 006017 ↑	Up arrow closes current location (if open); then opens and displays previous location.
START+1 126400 ⟨CR⟩	The New-Line key closes.
. .: START+1 ↓	Current address remains START+1. New-Line or Line Feed opens next. START+2 contains 6017.
START+2 006017	Uses contents of START+2 (6017)
./ 146032 ⟨CR⟩	as address; opens and displays address 6017.
START+2/ *006017*	Opens and displays START + 2 in octal,
; JSR @ OVLY1+16	displays in machine instruction format;
'⟨14⟩ ⟨17⟩	then in ASCII format (nonprinting values);
\.WRB 17	in .SYSTEM command format;
←1417	in half-word format;
& *30071*	in byte-pointer format;
:*6017*	in symbolic format (not a symbol);
**6017*	in symbolic format, OBO (here there is no difference between : and *);
.:*START+2*	and then displays current location symbolically.
.9999.=*023417*	Converts decimal 9999 (period indicates decimal) to octal.
0FH=*000017*	Converts hex numbers F and
2FFAH=*027772*	2FFA to octal.

Figure 4.1 Local command examples

You can convert most display commands into global commands by typing ESC and then the command. SEDIT then displays *contents of addresses* in the new format until you change it again. You can continue to use other display commands as usual. For example:

```
.START/020440 ⟨CR⟩
.$;
.START/LDA0ER+10↓
START+1SUB11↓
START+2JSR@OV1+16
START+3ISZ0 \.OVOP0 ⟨CR⟩
.$=
.START+3/012000
```

This command sequence displays the contents of the address in octal mode, and then changes the display mode. The

system displays in another format, and the mode is changed again.

You can enter new contents for an address in any mode, including the special data entry modes described next. For example:

```
.START+35/ 125120; MOVZL 1 1 MOVZR 1 1
.START+35/ 125220; MOVZR 1 1
```

The contents of START+35 are changed from MOVZL 1 1 to MOVZR 1 1, and the change is then verified. The display mode remains octal throughout.

SEDIT also allows you to examine radix 50 symbols. All user symbols are stored in radix 50 in the program symbol table; this allows any symbol to fit in two 16-bit words. You can use the left brackets to display the symbol name. The format is

firstword[*secondword*[

For example, assume you are stepping backward through a symbol table:

```
T.OVL+271 000000↑
T.OVL+270 131401↑
T.OVL+267 113014 131401 [113024[ROOT
```

This works because the RDOS/DOS Relocatable Loader (RLDR) builds the symbol table downward, not upward as it does the rest of the save file. (For more information on RLDR, see *RDOS/DOS Assembly Language and Program Utilities*, DGC No. 069-400019.) You need not actually type the numbers before the brackets; simply type the bracket after the appropriate number appears:

```
T.OVL+271 000000↑
T.OVL+270 131401[↑
T.OVL+267 113014[ROOT
```

To do this, remember to stay in synch; every second bracket must follow the first word of the symbol. Symbol entries in the table are three words long.

Editing a Save File

Assume you have determined with a debugger that you want to change an instruction in a program called WRITE.SV. You cannot or do not want to change the source program and reassemble and reload this program; you simply want to run it. In this case, you use the following procedure:

```
SEDIT WRITE ⟨CR⟩
SEDIT REV x.xx
.WRITE%
```

Invoke SEDIT and enable local symbols.

```
.START+10/020445; LDA 0 START+35 \.CCON 45
START+11 0006017; JSR @ 17
START+12 12021027; LDA 0 272
```

Display the contents of START+10 in octal, instruction, and system call format.

After examining succeeding locations, you decide to change the display mode to instruction format because you are looking for a machine instruction.

```
⟨CR⟩
.$;
.START + 32/ JMPSTART + 45↑
START + 33 LDA 2 START+71↑
START + 34 LDA 1 START+74↑
START + 35 AND 1 2 ANDS 1 2 ⟨CR⟩
```

Find the faulty instruction and change it to ANDS 1 2. Verify the change.

```
START + 35/ ANDS 1 2 ⟨CR⟩
.$Z
DONE
R
```

This verifies the change and leaves SEDIT. You can now run the program.

Editing Overlays ($O)

Using SEDIT, you can edit any overlay as if it were part of the save file. SEDIT opens the overlay file and reads the overlay you specify into its node; after editing, it writes any changes you make back to the overlay file. You can read any overlay into its node at any time.

If you have identified an overlay with the .ENTO pseudo-op, you can read the overlay into its node by typing

overlayname$O

If .ENTO was omitted from the overlay, you must specify the node number and overlay number

node]overlay$O

where *node* is the node number (0 for the node defined by the first pair of brackets in the RLDR command line, 1 for the node defined by the second pair of brackets, and so on), and *overlay* is the overlay number (0 for the first binary within the square brackets, 1 for the second binary within the square brackets, and so on). We used byte entry to enter the node and overlay numbers; you *can* use a word with the node number in the left byte and overlay number in the right byte—for example, 0 for node zero, overlay zero, 1 for node zero, overlay 1, and so on.

For the rest of this section, we will use a save and overlay file created by the command line

RLDR/D ROOT/U [OVLYO/U,OVLY1/U] ⟨CR⟩

This creates save file ROOT.SV, with one node, and overlay file ROOT.OL, which holds overlays OVLY0 and OVLY1. Both overlays contain an .ENTO OVLY0 and an .ENTO OVLY1 statement, respectively. The RLDR load map shows that the overlay node begins at location 517 and ends at 1117.

You can enable or disable symbols at any time, as described further in Chapter 6. When you enable symbols in one module, you disable symbols in the current module.

Figure 4.2 illustrates the loading and examination of overlays.

When you load another overlay after you have enabled symbols in an overlay, SEDIT displays symbols exactly as they were in the *original* overlay. For example, in the second overlay in Figure 4.2, PRNTB might have really contained the instruction LDA 0, FILEA instead of LDA 0 B. This can be very confusing, so you can clarify overlay symbols by following this sequence when you want to edit successive overlays in the same node:

1. Enable local symbols if you need them for the next step.
2. Get to node start and enable symbols in the desired overlay (*ovlyname*%).
3. Read in the desired overlay (*ovlyname*$O or [*node*]*ovly*$O).
4. Edit the overlay.
5. Re-enable symbols in the root program (for example, ROOT%); go to step 2.

SEDIT ROOT ⟨CR⟩	
SEDIT REV x.xx	
.ROOT%	Enables local symbols.
.515/*027117'.0*	Approaches the node start (this is the end of the overlay file text string, ROOT.OL.)
OFILE +4046000'L⟨0⟩↓	Symbol OFILE points to the overlay file name.
PRNTB 000000. = 000517↓	PRNTB is vacant, as is the rest of the node. There
PRNTB + 1 000000↓	is a value for it in the first overlay.
PRNTB + 2 000000↓	
PRNTB + 3 000000 ⟨CR⟩	
.OVLY0$0	Loads OVLY0 into its node (without .ENTO, you would use 0]0$0).
.515/ *027117'.0*↓	Approaches node as before.
OFILE + 4 046000'L⟨0⟩↓	Same value at 516.
PRNTB 020407	The value for PRNTB is an instruction. The overlay
; LDA 0 PRNTB + 7↓	is in the node.
PRNTB + 1 006017	PRNTB + 1 holds an instruction, and PRNTB + 2 holds a system call.
;JSR @ NSW + 16↓	
PRNTB + 2 010000.PCHA /0↓	
PRNTB + *3 002430*	PRNTB + 3 holds another instruction.
;JMP @ PRNTB + 6 ⟨CR⟩	
.OVLY0%	Enables symbols in first overlay. SEDIT displays LDA 0 B instead of LDA 0 PRNTB + 7.
.PRNTB/ *020407; LDA 0 B*↓	
PRNTB + 1 006017	Enabling overlay symbols does
;JSR *@ NSW + 16*↓	not affect the display of PRNTB + 1 or PRNTB + 2.
PRNTB + 2 01000 \.PCHA 0 ⟨*CR*⟩	
.OVLY1$0	Loads OVLY1 into node, replacing overlay 0.
.ROOT%	Enables ROOT's symbols; symbols are not enabled in this overlay.
.515/ *020407'.O*↓	Approaches node again.
OFILE + 4 046000'L ⟨O⟩↓	Same value at 516.
PRNTB 020407	Same value (instruction) in PRNTB.
;LDA 0 PRNTB + 7↓	
PRNTB + *1 006017*	Same value in PRNTB + 1 (instruction). Checks the overlay to determine whether it is new or old.
;JSR *@ NSW + 16*↓	
PRNTB + 2015401 \ .RDL 1	Identifies the overlay as 010000,.PCHA 0. The second overlay is in the node.

Figure 4.2 Loading and examining overlays

Including Symbols in Save and Overlay Files

If you want to edit a save file symbolically instead of exclusively with octal locations, you must include user symbols. SEDIT does not recognize any symbols at all unless the program includes a program symbol table. You can

instruct RLDR to include both a symbol table and debugger with the RLDR/D switch; or you can have it include a symbol table only by writing an .EXTN .SYM. statement into one of the program modules.

If the program includes a symbol table, SEDIT automatically recognizes *global* symbols (those specified in an .ENT psuedo-op in the program). To use *local* symbols (those not identified by .ENT) you must insert the global /U switch in the assembler command line and the *local* /U switch in the RLDR command line. After including local symbols, you then tell SEDIT to *see* them with the SEDIT command *name%*.

For example, assume you want to include all symbols from two source files named MYPROG and MYPROG1 in save file MYPROG. You would follow these steps:

```
{MAC/U MYPROG; MAC/U MYPROG1 }
{ASM/U MYPROG; ASM/U MYPROG1 } ⟨CR⟩
 RLDR/D MYPROG/U MYPROG1/U $LPT/L ⟨CR⟩
.
.
.R
```

This produces save file MYPROG.SV and sends the load map to the line printer. It is very helpful to have a copy of the map. If either MYPROG or MYPROG1 included an .EXTN .SYM., you could omit the RLDR global /D switch.

To enable local symbols in overlays, append global /U to each overlay name, as in the following:

```
RLDR/D MYPROG/U MYPROG1/U
[OVLYO/U, OVLY1/U]
```

Then invoke SEDIT and use the *name%* command as follows:

```
SEDIT MYPROG ⟨CR⟩
SEDIT REV x.xx
.MYPROG%
.
.
.
```

Chapter 6 describes enabling and disabling user symbols.

Text Files

When you invoke SEDIT to edit a text file, use the global /Z and /N switches. Generally, you will display locations in ASCII mode ($') and use the ASCII entry operator ("*xx*) and byte entry operators (*n*]*n*).

Assume you have created a text file with the CLI by typing:

```
XFER/A $TTI FILEA/R ⟨CR⟩
MESSAGE HELLO; DISK ⟨CR⟩
CTRL-Z
```

You need the /R switch in RDOS to organize the file randomly. The text editors automatically create random files.

Now you invoke SEDIT by typing:

```
SEDIT/Z/N FILEA ⟨CR⟩

SEDIT REV x.xx
```

The /Z switch tells SEDIT that the file begins at location zero; if you forget it, SEDIT displays the first 15 (octal) location contents as 0, and you cannot edit the file accurately. The /N switch instructs SEDIT not to look for a symbol table; if you forget it, SEDIT might display the error message:

```
NOT ENOUGH MEMORY FOR SYMBOL TABLE
```

and return to the CLI.

In the following example, we will check the contents of the locations and add a CLI command to the file.

```
.0/046505'ME ⟨CR⟩
.$'
```

This checks the first two locations, which contain ME, and changes the display mode to ASCII.

```
.
+1SS↓
+2AG↓
+3E↓
+4HE↓
+5LL↓
+60;↓
+7DI↓
+10SK↓
```

Locations 1 through 10—the end of the text message—are displayed.

```
+11⟨15⟩⟨0⟩↓
```

CR and null are the last nonzero words.

```
+12⟨0⟩⟨0⟩ ⟨CR⟩

.11/⟨15⟩⟨0⟩"; L ⟨CR⟩

.11/;L↓

+12⟨0⟩⟨0⟩"IS↓

+13⟨0⟩⟨0⟩"T/↓
```

The first few characters of a list command are added to the file. The " is used to insert ; L in location 11. Location 11 is checked and the next location opened. The characters I

and S are placed in 12 and again, the next location is opened. T/ is placed in location 13, and location 14 is opened.

```
+14⟨0⟩⟨0⟩"N⟨1?

.14/⟨0⟩⟨0⟩116]15 ⟨CR⟩

.14/N⟨15⟩

+15⟨0⟩⟨0⟩ ⟨CR⟩

.$Z

DONE.

R
```

The " mark cannot be used to enter more than two characters, so the byte entry operator (*n*]*n*) is used instead. (The ASCII code for N is 116.) Location 15 is left unchanged, and the editing process is terminated.

We have now added the CLI command LIST/N to the file. Verify it:

```
TYPE FILEA ⟨CR⟩

MESSAGE HELLO;DISK;LIST/N

R
```

We can execute the commands by typing:

```
@FILEA@ ⟨CR⟩
```

When you use SEDIT to edit a text file, it automatically extends the byte length of the file to an even multiple of 512 bytes. If you open a location beyond the existing end-of-file (for example, location 1001 in FILEA above), SEDIT automatically extends the file to the next multiple of 512 bytes.

Chapter 5

Searches, Displays, and Registers

The material in this chapter relates primarily to editing *save* files, not text files.

Often you will want to search a file for a specific word or display entire blocks of words on the console or line printer. This chapter explains the search and display commands and the special SEDIT registers that can help you use these commands effectively.

Search Command ($S)

The search command ($S) enables you to search forward in a file for the contents of any address (*addr*). To search to a location, type *addr* $S, and SEDIT will search forward from address zero and stop when it reaches *addr*.

To search for the contents of an address, you must open the word register with the $W command, type the word you want SEDIT to find, and close the word register. Then you type $M to open the mask register, enter a mask word, and close it. If the mask register contains zero (default), the search will match all words in the file. To match a specific word instead of a general group, enter −1(177777) in the mask register.

You can also set an increment for the search with the search increment register and direct output to the line printer by placing 1 in the output register.

After you set these registers (which are described later in this chapter), you enter your search command, and SEDIT seeks matching contents in the file.

The formats of the search command are listed in Table 5.1.

Command	Description
addr$S	Searches from location 0 to *addr*.
$S	Searches entire file for value in word register.
⟨*addr*$S	Searches from location zero to *addr* for value in word register.
addr⟨$S	Searches from *addr* to last location for word value.
addr⟨*addr*$S	Searches from *addr* to *addr* for word value.

Table 5.1 Search command formats

If you omit *addr*, SEDIT assumes zero as the lowest address and 77777 as the highest address. When SEDIT finds a word matching the contents of the word register, it displays *addr* and its contents in the current display mode. If you have enabled symbols, it displays locations symbolically, as in Figure 5.1.

```
SEDIT ROOT ⟨CR⟩
SEDIT x.xx
ROOT%                          Enables local symbols.
.ER$S                          Displays file to location ER.
+0 000000                      SEDIT always searches for
                               locations from location 0; it
                               may therefore take some
OVLY1 000000                   time to reach location ER.
.
.
.
RTURN+2 000401
ER 006017                      Location ER is found.
..=000503.:ER+1 ⟨CR⟩           Current position is 503;
                               symbolically, ER+1.
.$W 000000 JSR@ 17 ⟨CR⟩        To find locations containing
                               JSR @ 17, first place JSR
                               @ 17 in word register.
.$W 006017;JSR @ +17 ⟨CR⟩      Verify new value in word
                               register.
.$M 000000-1 ⟨CR⟩              To match a word specifi-
                               cally, place -1 in mask
                               register.
.$S                            Searches the save file for
                               JSR @ 17. SEDIT displays
                               the locations symbolically
START+2 0006017                that contain JSR @ 17.
START+6 006017
.
.
DEBUG+01277 006017
.$;                            Changes display to instruc-
                               tion mode.
.$S                            Searches again.
START+2 JSR OV1+16             SEDIT displays contents in
                               instruction format.
START+6 JSR OV1+16
.
DEBUG+1277 JSR @ OV1+16
.$K                            All symbols are killed. (See
                               Chapter 6.)
.$S                            Searches again.
+454 JSR @ 17                  Displays locations in octal,
                               contents in instruction for-
                               mat.
+460 JSR @ 17
.
2515 JSR @ 17
.1$K                           Re-enables global sym-
                               bols.
```

Figure 5.1 Search command ($S) example

Display Command ($D)

The display command ($D) resembles the search command, except that it cannot search for the contents of a location, and it displays the contents of eight locations on one line. You can display on the line printer by setting the output register to 1, and you can set the search increment register to display locations at desired increments. The $D command always displays contents in octal; you cannot change the output of $D to other formats.

The formats for the display command are shown in Table 5.2.

Format	Description
addr$D	Displays contents from location 0 to *addr*.
$D	Displays contents of all locations.
addr⟨$D	Displays from location *addr* to last location.
addr⟨*addr*$D	Displays from *addr* to *addr*.

Table 5.2 Display command formats

Figure 5.2 shows display examples. The first example displays locations one through 20, and locations 400 through 450 in a save file. Like most save files, this one uses only location 17 (in first range); its User Status Table, TCB(s), and overlay directory (if any) start at 400 (second range). Figure 5.3 displays locations in a text file.

```
SEDIT ROOT ⟨CR⟩

SEDIT REV x.xx

0 20$D

+0 000000 000000 000000 000000 000000 000000 000000 000000
OVLY1+7 000000 000000 000000 000000 000000 000000 000000 001155
OVLY1+17 000000
.400⟨500$D
"USTAD 000000 000060 006634 006114 006635 001120 001216 006635
USTAD+10 000000 177777 177777 000410 000424 000424 177777 00452
USTAD+20 000445 000000 177777 000000 001124 000000 000000 000000
USTAD+30 000000 000000 000000 177777 000000 000000 000000 000000
USTAD+40 000000 000000 000000 000000 000000 000001 177400 001001
USTAD+50 000000 000517 020440 126400 006017 012000 000424 020426
START+6 006017 010000 000420 020423 126400 006017 020000 000413
LOV0+5 002420 020416 126400 006017 020000 000405 002413 006017
RTURN+1 004400"
.
$Z
DONE
R
```

Figure 5.2 Display command examples

```
SEDIT/N/Z TESTFILE ⟨CR⟩
SEDIT REV x.xx
.0 10$D
+0 046505 051523 040507 042440 052110 044523 020111 051440
+10 052105
```

Figure 5.3 Displaying locations

SEDIT Registers

SEDIT registers are locations into which you can insert values. These values control searches, displays, and I/O and numeric output. There are five registers: output, number, search increment, mask, and word. You can use all the registers for searches and the first three for displays.

The default value in each register is zero (except for the search increment register). Any value you insert reverts to the default when you leave SEDIT and return to the CLI.

Output Register ($H)

By default, SEDIT sends search and display output to the console. You can direct this output to the line printer by entering the value 1 in the output register. Use the $H command to open and examine this register, you can then change it if you want. For example:

```
$H 000000 1 ⟨CR⟩
```

directs output of search and display commands to the line printer. When output of search commands goes to the printer, SEDIT inserts the value (in current output mode) of the sought word before the contents of each address. Display command output is the same on both console and printer.

Number Register ($N)

By default, SEDIT displays numbers in octal, unsigned, with leading zeroes. You can tell it to output numbers with a leading sign, suppress leading zeroes, or select radix 8, 10, or 16 by placing the proper value in the number register. Use the $N command to open and examine this register; you can then insert a new value if you choose to do so, as shown in Table 5.3.

New Value	Set Bit(s)	Effect
100000	B0	Signs numbers.
040000	B1	Suppresses leading zeroes.
000010	B12	Displays in octal.
(or 000000)	None	Same.
000012	B12,B14	Displays in decimal (10. has same effect).
000020	B11	Displays in hexadecimal (16. has same effect).

Table 5.3 Number register values

You can enter combinations of these values in the number register. When you enter any value but those above, SEDIT uses the default radix (octal).

```
.START/020440 (CR)
.$N00000010. (CR)
.START/08480. (CR)
.4440=02336. (CR)
.$N00001014000O (CR)
.START/+20440 (CR)
.$N-400001 6. (CR)
.START/2120H (CR)
15=000DH
```

The number is shown first in octal form–the default display–and then in decimal form. The $N command then converts an octal number. It inserts a sign, suppresses leading zeroes and displays the number in octal form. It then converts the display to hexadecimal form, which is indicated by the use of a final H.

Search Increment Register ($J)

By default, SEDIT increments locations by one in search and display commands. You can specify a different increment by opening the search increment register and inserting the increment you want. The $J command opens this register.

The default value in the search increment register is 000001; a negative or zero value is illegal and produces the default value.

The following example illustrates the use of the search increment register.

```
.$N00000040000 (CR)
.$M0-1 (CR)
.$W0STA3,50 (CR)
.$S
DEBUG+52754050
DEBUG+57254050
DEBUG+107554050
```

The .$N command suppresses leading zeroes, sets the mask register to −1, and inserts STA 3, 50 into the word register. It then searches all locations for STA 3, 50.

```
.$J14 (CR)
.$S
DEBUG+57254050
.$W540500 (CR)
.$S

+00

OVLY1+30

OVLY1+70

OVLY1+130
```

The $J command specifies an increment of 4 in the search increment register. The results of a search indicate there is only one STA 3, 50 at an increment of 4 from zero. The command then places zero in the word register and initiates a second search, yielding the information that several locations contain zero.

```
.$J410 (CR)
.(400$D
0.....
OVLY1+77.....
OVLY1+177.....
OVLY1+277.....
USTAD.....
.
.$J101 (CR)
```

The .$J command places an increment of 10 in the search increment register and issues a display command. Next, it specifies a search/display increment of 1.

Mask and Word Registers ($M and $W)

SEDIT uses both the mask and word registers for searches. If you want to search for a specific instruction, you must set both of these registers. As you have seen earlier in this chapter, you insert the word you want matched in the word register. After you issue the search command, SEDIT searches each location and ANDs the value in the mask register with the contents of the location. If this value matches the word register, SEDIT announces a match. The default value for both registers is 000000.

Because SEDIT ANDs the mask with the contents and the default mask is 000000, there will be no match unless you set the mask register. To match a specific word, insert −1 in this register. To match only a portion of a word, set the bits containing that portion to 1 in the mask register and set the other bits to 0.

The command

$M

opens the mask register and displays the contents.

Place the value you want SEDIT to match in the word register. You can enter this in any format; for example, 0 0, 101300, or .WRL 0. The command

$W

opens the word register and displays its contents. The following example shows application of the word and mask registers in search commands.

```
SEDIT ROOT (CR)
SEDIT REV x.xx
.$M000000-1 (CR)
.$W000000.RDL0 (CR)
.$S
5742015400
.15400.RDL0
```

The .$M sets the mask register to 1777777, or 16 ones. The .$W command inserts a read line call in the mask register, and the .$S command institutes a search of all the locations, turning up one match, in location 15400, which contains .RDL 0.

The mask register helps you search for all occurrences of one *type* of instruction. For example, assume you want to find all MOVZL instructions in a program. The octal instruction is 101120, but this applies only to MOVZL 0,0; with any other accumulators, there is no match. Bits 1 through 4 specify accumulators in MOV instructions, so you mask them out by inserting a word that contains 0 in bits 1, 2, 3, and 4, and 1 in the other bits. In the following example, this word is 103777.

```
SEDIT SYS ⟨CR⟩
SEDIT REV x.xx
.$W000000MOVZL ⟨CR⟩
.$M000000103777 ⟨CR⟩
.⟨4000$S
2535155120
2536175120
.155120; MOVZL23 ⟨CR⟩
.175120; MOVZL33
```

The $W command inserts MOVZL in the word register, and the $M command inserts the MOVZL mask in the mask register. The $S command searches locations zero through 4000 for MOVZLs, and finds two matches. The match words are then checked.

Chapter 6

SEDIT Symbols

As described in Chapter 4, global symbols are those declared in .ENT statements in a program. To recognize global symbols, SEDIT requires only that a save file include a program symbol table. The RLDR/D command includes both a program symbol table and a debugger; a .EXTN .SYM. statement in any program module will include *only* the symbol table. SEDIT always recognizes global symbols unless you have disabled them.

SEDIT also recognizes *local* symbols–those not declared with .ENT–if you assembled the program source file(s) using the global /U switch, and if you used the local /U switch in the RLDR command line. To enable local overlay symbols, append local /U to the overlay name in the brackets, as in the following:

```
RLDR/D MYPROG/U MYPROG1/U [OVLY0/U, ↑ ⟨CR⟩
OVLY1/U] ⟨CR⟩
```

When you proceed through a file, examining locations, SEDIT displays each location as it relates to the nearest previous symbol defined. However, if the nearest previous symbol is more than 2000_8 locations away, SEDIT displays the octal, not the symbolic, value of the location.

Enabling Symbols

If the program has a symbol table, SEDIT will understand all global symbols when you invoke it. To enable local symbols in any program module or overlay, type:

name%

where *name* is the module name, as assigned by the .TITL pseudo-op. The name is not necessarily the filename. If .TITL was omitted from the module, the default name is .MAIN. The *name%* command enables all local symbols in this module. It also disables local symbols in only one module at a time.

To edit overlays symbolically, see the section entitled Editing Overlays in Chapter 4.

To enable global symbols after you have killed all symbols ($K, below), type:

n$K

where *n* is a single octal digit (1 is convenient).

In any module, to enable all symbols you have not individually disabled, type:

name%

Disabling Symbols

Too many local symbols can cloud your overview while you are editing. Also, an individual symbol can interfere with editing if its value happens to fall at the wrong point. Occasionally, even global symbols hamper editing. You can disable all symbols, or individual symbols, with the command

$K

For example:

```
SEDIT MYPROG ⟨CR⟩        Invoke SEDIT.

SEDIT REV x.xx

.START/0240440           START was .ENTered.

.$K                      Disable all symbols.

.START/U                 START is now undefined.
```

To disable any single symbol, type:

name$K

where *name* is the symbol name. Note that SEDIT still recognizes the symbol when you type it in, but does not display it when you proceed forward or backward through locations in the file. Instead, SEDIT displays the location of the disabled symbol as it relates to the preceding active symbol (if the previous symbol is less than 2000_8 locations away—see above).

When you disable output of a single symbol, you cannot re-enable it during this SEDIT session. To re-enable the disabled symbol, leave SEDIT and return to SEDIT.

For example, assume program ROOT starts at symbolic location START, and the next symbol is LOV0. Here, it does not matter whether START and LOV0 are global or local symbols.

```
SEDIT ROOT ⟨CR⟩
SEDIT REV x.xx
.ROOT%
.START/020440↓
START+1126400↓
START+2006017↓
.
.
START+10000420↓
LOV0020423 ⟨CR⟩
```

The command enables all symbols it opens and displays location START and then all succeeding locations up to LOV0.

```
.LOV0$K
.START+10/000420↓
START+11020423
.LOV0/020423
```

SEDIT disables LOV0 for the sessions, and checks location START+10, which it finds to be unchanged. SEDIT then displays LOV0 as START+11. The utility still recognizes LOV0 on input, but does not display it.

```
.$Z
DONE.
R
SEDIT ROOT ⟨CR⟩
SEDIT REV x.xx
.ROOT%
.START+10/000420↓
LOV0020423
```

SEDIT is left and re-entered. The command enables all local symbols, LOV0 among them.

Managing Symbols

To enable global symbols while editing, you need do nothing if you have used the global /D switch in the RLDR command, or a .EXTN .SYM. statement in a source module.

To enable local symbols, type *name%*. (You must have used the assembler global /U and RLDR LOCAL /U switches.)

To disable all symbols, type:

$K

To disable the display of any symbol, type *symbolname*$K.

To re-enable global symbols, type 1$K (or any digit instead of 1). The sequence $K, 1$K disables all symbols and re-enables globals.

To re-enable all but explicitly disabled symbols, type:

name%.

By default, when you include the global /D switch in the RLDR command; RLDR places the debugger directly above your programs and places the symbol table (which either SEDIT or the debugger needs) directly above the debugger. If you use a .EXTN .SYM in a program module and omit RLDR global /D, the symbol table will be directly above the program. The load map (if you specified one with the RLDR local /L switch) will give you the location of each global symbol; the starting address of the debugger (if present); and the starting and ending address of the symbol table. If the program lacks a symbol table, you *can* use the addresses in the load map to patch, but this is much more awkward than using symbols.

Figure 6.1 shows various attempts to disable a distracting symbol.

```
SEDIT MYPROG ⟨CR⟩
SEDIT REV x.xx
.$N 000000 40000 ⟨CR⟩          Suppresses leading zeroes.
START/20440;LDA 0 ER+10        Displays location START.
.$;                            Displays in instruction mode.
.MYPROG%                       Enables all symbols.
.START/LDA 0 OFILE             Local symbol OFILE replaces global ER+10.
START+1 SUB 1 1                Examines succeeding locations.
.
.
.
LOOP+4/INC 0 0                 Examines LOOP+4, then the next location, and then the next...
LOOP+5 COM 0 0 SZR
RTCNS JMP RTCNS+10             Encounters symbol RTCNS, which is confusing.
RTCNS+1 JMP RTCNS+15 ⟨CR⟩
.=5005 ⟨CR⟩                    Checks current location in octal, and symbolically.
. .:RTCNS+1 ⟨CR⟩
.$K                            Disables all symbols and enables global symbols; this disables RTCNS if it is a local symbol.
.1$K
. .:RTCNS+1 ⟨CR⟩               Checks current location symbolically; RTCNS is still enabled. It is therefore global.
.RTCNS$K                       Disables RTCNS specifically.
. .:START+27 ⟨CR⟩              RTCNS is not displayed.
.MYPROG%                       Re-enables local symbols.
.LOOP + 6/ JMP LOOP+16         Continues Editing.
.
.
$Z
DONE
R
```

Figure 6.1 Disabling and enabling symbols

Part III

Symbolic Debugger

Chapter 7

Introduction to the Symbolic Debugger

Debugging is the process of detecting, locating, and removing mistakes from a program. When a programmer wishes to debug a program, the Symbolic Debugger must be loaded with the program. The programmer can then control program execution, causing the program to halt in the debugger at one or more points so that the programmer can examine the contents of memory locations and special registers (such as the accumulators and Carry) and correct the contents if necessary. Because the debugger is symbolic, the contents can be examined in source language format, although octal format or a number of special formats can also be used.

The DGC Symbolic Debugger allows you to set up to eight active breakpoints within a program. Program execution will halt before the instruction at the breakpoint location is executed, and you can then issue debugging commands. You can restart the program at the breakpoint instruction or at any other location.

The numbers following the titles indicate which debuggers are supplied to a single user. For example, MDEBUG and MIDEB are supplied to all users who have a NOVA Mapped RDOS system.

The versions are grouped into two categories. Within a given group of four versions, the Symbolic Debugger command languages are identical. Thus, the command languages within the versions of the generic type are identical, but there are slight differences in the commands and their interpretation in the two generic types.

RTOS users are supplied with the appropriate version of IDEB.

Symbolic Debugger Versions

The Symbolic Debugger that you receive is tailored to your individual computer and system configuration. Therefore, your version will depend upon whether you have a NOVA or ECLIPSE computer; whether you have an RDOS or RTOS operating system; and whether the system is mapped or unmapped. Two versions of the debugger are supplied with RDOS operating systems. The first interrupts during debugging and has the generic name DEBUG. The second disables interrupts during debugging and has the generic name IDEB.

The Symbolic Debugger is supplied as either a relocatable binary tape or a relocatable program on a library tape. Table 7.1 lists the different versions of the symbolic debugger and their tapes.

Loading the Symbolic Debugger

This section explains how the Symbolic Debugger is loaded under the different operating systems. Refer to the specific segment that deals with the type of system on which you are working.

Loading RDOS DEBUG

RDOS users receive one of the following library tapes, each of which contains a version of RDOS DEBUG.

Version	Title	Tape
DEBUG		
NOVA Unmapped RDOS DEBUG	DEBUG1	DEB.RB in USYS.LB
NOVA Mapped RDOS DEBUG	MDEBUG2	MDEB.RB in MSYS.LB
ECLIPSE Unmapped RDOS DEBUG	BEDBUG3	BDEB.RB in BSYS.LB
ECLIPSE Mapped RDOS DEBUG	AEDBUG4	ADEB.RB in ASYS.LB
IDEB		
NOVA Unmapped RDOS IDEB	IDEB1	IDEB.RB
NOVA Mapped RDOS IDEB	MIDEB2	MIDEB.RB
ECLIPSE Unmapped RDOS IDEB	BIDEB3	BIDEB.RB
ECLIPSE Mapped RDOS IDEB	AIDEB4	AIDEB.RB

Table 7.1 Symbolic Debugger versions

Library Tape	Relocatable Binary	Title of Version	Associated Computer and System
USYS.LB	DEB.RB	DEBUG	NOVA Unmapped RDOS
MSYS.LB	MDEB.RB	MDEBUG	NOVA Mapped RDOS
BSYS.LB	BDEB.RB	BDEBUG	ECLIPSE Unmapped RDOS
ASYS.LB	ADEB.RB	ADEBUG	ECLIPSE Mapped RDOS

Table 7.2 Library tapes containing RDOS DEBUG

The global switch /D of the RLDR command indicates that a debugger is to be loaded with the user programs. The name of the system tape itself need not appear in the command line, except when you wish to control the area of core into which the debugger is to be loaded.

For example, when you enter the following:

```
RLDR/D A B C ⟨CR⟩
```

the system loads the RDOS DEBUG version from the library.

The command line:

```
RLDR/D A B BSYS.LB C ⟨CR⟩
```

informs the system to load the RDOS DEBUG version from the library before loading user program C.

Loading IDEB

RDOS users receive one of the following relocatable binary tapes, each of which contains a version of IDEB:

Relocatable Tape	Title of Version	Associated Computer and System
IDEB.RB	IDEB	NOVA Unmapped RDOS
MIDEB.RB	MIDEB	NOVA Mapped RDOS
BIDEB.RB	BIDEB	ECLIPSE Unmapped RDOS
AIDEB.RB	AIDEB	ECLIPSE Mapped RDOS

Table 7.3 Versions of relocatable binary tapes

To load an IDEB version of the debugger, the name of the debugger must appear in the command line. If you also name the system library tape containing RDOS DEBUG in the command line, the IDEB version must appear first in order to be loaded. As with RDOS DEBUG, the global /D switch must be included in the command line.

The following are examples of the different ways in which you can load specific versions of the debugger. The command

```
RLDR/D A B C MIDEB ⟨CR⟩
```

loads an IDEB version following the user programs.

```
RLDR.D A BIDEB B C ⟨CR⟩
```

loads an IDEB version following program A and before user programs B and C.

```
RLDR/D A B USYS.LB C IDEB ⟨CR⟩
```

loads RDOS DEBUG because IDEB does not precede the library tape name.

```
RLDR/D A B AIDEB ASYS.LB C ⟨CR⟩
```

loads IDEB because a relocatable tape is named before the library tape ASYS.LB.

Loading the Symbolic Debugger for RTOS

RTOS users receive a version of IDEB, which is supplied as a file of the RTOS library and loaded under RDOS.

As indicated in the *RTOS System Reference*, you load the RTOS libraries by using the RLDR command to produce a save file of user binaries, drivers, RTOSGEN, and the libraries, as follows:

RLDR/C/D *user-binaries* [*user-drivers*] *RTOS-libraries* ⟨CR⟩

The global switch /D loads IDEB from the library. The load map can be obtained by listing either to the console ($TTO) or the line printer ($LPT).

The procedure for executing the loaded program is described in the *RTOS System Reference*. When no starting address is given after a .END in a user binary, you can start the debugger at the contents of 406, or at the DEB address indicated in the loader map.

Invoking the Symbolic Debugger

When the Symbolic Debugger has been loaded, it can be invoked from the CLI level by the DEB command, followed by the name of the save file. For example:

```
RLDR/D A B/U C ⟨CR⟩
```

loads programs A, B, and C along with RDOS DEBUG.

The command

```
DEB A ⟨CR⟩
```

enters the debugger, allowing you to debug the program A.SV.

When the DEB command is given, the debugger responds with a Carriage Return/Line Feed. You can now proceed to issue the debugging commands described in this manual.

Command Format

A symbolic debugging command has the format

[*argument*] [$]*command-code*

where:

command-code is a single teletypewriter character.

$ must precede all alphabetic command codes and certain symbolic command codes.

argument may be one of the following:

sym a user symbol

addr an address having any legal address format: octal or decimal integer, user symbol, or an expression of the form:

x + x + x...

where each x is a user symbol, or octal or decimal integer, separated from the following x by either a plus (+) or minus (−) sign. Decimal integers must be followed by a decimal point to distinguish them from octal integers.

n a decimal of octal integer.

name a user symbol that names a program.

addr⟨ a range of addresses from *addr* to 77777.

addr⟨addr a range of addresses from *addr* to *addr*.

Commands are described in the following chapters and provide facilities to:

- Set, delete, and examine breakpoints.
- Restart execution at selected points.
- Monitor memory, accumulators, and special registers.
- Enable and disable symbols available to the debugger.
- Place the debugged program in a file that can be saved.
- Perform core searches.
- Set the format of debugger output.
- Punch or print portions of the user program.

Debugger Error Responses

The two most common error responses are *U* and *?*.

The *U* message indicates that one or more of the characters that have been entered in a debugger command are undefined. This error response immediately follows the command line.

The *?* message means that the command syntax is incorrect. This message also follows immediately after the command line.

Chapter 8

Symbolic Debugger Memory Locations and Special Registers

Memory locations in the user program can be opened, examined, and modified using one of the following commands:

addr/ Opens the address and prints the contents.

addr! Opens the address and does not print the contents.

(*addr* is any expression that defines a location.)

When the memory location has been opened using either the backslash (\ ) or the exclamation point (!), you can modify the contents of the location and close it or close the location without any modifications.

You can modify the contents of a location by typing the new contents on the same line with the command that opens the location. The open location can then be closed in one of the four ways listed in Table 8.1.

Symbol	Action
⟨CR⟩	Carriage Return key. Closes the open location.
↓	New Line key. Closes the open location and opens the succeeding location.
↑	SHIFT-6 keys. Closes the open location and opens the preceding location.
\	Backslash. Closes the open location and opens the location specified by the contents of that location.

Table 8.1 Closing open locations

All versions of the Symbolic Debugger use these commands to open and close locations. A few examples of the commands follow.

START/006011 6017↑

Opens the symbolic location START and modifies contents to 6017.

+762 000000↑

Opens the previous location, but does not modify.

+761 177400↑

Closes the location and opens the previous one.

+760 177400 ⟨CR⟩

Closes the location.

START/006017↓

Opens the symbolic location START as well as the next location.

START+1 001400↓

Closes the open location and opens the next one.

START+2 006073 ⟨CR⟩

Opens the next location and closes it without opening the following location.

1000/.RDL 0 .RDL 1 ⟨CR⟩

Opens location 1000, changes the contents to .RDL 1, and closes the location.

1000! .RDL 0 ⟨CR⟩

Opens location 1000 without printing the contents. Changes the contents back to .RDL 0.

Monitoring Special Registers

Special registers are locations that contain program status information. Like memory locations, special registers can be opened, examined, modified, and closed. A special register is normally closed by pressing the Carriage Return key, although the backslash or exclamation point can be used where appropriate. All of the registers are used in both

versions of the debugger unless noted otherwise in the following sections.

Accumulators

The command that opens an accumulator for examination or modification is used in all versions of the debugger. The command format is

n$A ⟨CR⟩

where *n* is the number of the accumulator. The variables that you can use depend on which computer you are operating. They range from 0 through 3 for the NOVA computer and 0 through 7 for the ECLIPSE computer.

To examine all accumulators, enter the following command:

$A ⟨CR⟩

The utility performs a Carriage Return and prints the number of each accumulator, followed by its contents.

Floating-Point Accumulators

The four floating-point accumulators can be examined by using the command

$F

The debugger performs a Carriage Return/Line Feed and prints the number of each floating-point accumulator, followed by its contents.

Search Increment Register

The search increment register is used in performing memory searches. By default, locations in the range selected for the search will be searched forward in memory with an increment of one location. By setting the search increment register, you can adjust the increment to any number of locations, or call for a backward search of memory locations. To open the search increment register for examination and possible modification, use the command

$J

Two examples of the command are

```
$J 000001 -10 ⟨CR⟩
440⟨$S
00440 000766
00430 101300
00420 025400
.
.
.
00000 006017
```

The $J command opens the search increment register and sets a backward search for every 10 locations. The $S command then searches from location 440 down in memory.

```
$J 100010 2 ⟨CR⟩
102⟨114$S
00102 177777
00104 177777
00106 000000
.
.
.
00114 00232
```

The $J command opens the search increment register and sets a forward search at every second location. The $S command then searches from location 102 to location 114.

Mask and Word Registers

Mask and word registers are used to perform memory searches. You can use the word register to search for particular bit contents.

To set the word register to represent the contents for which you want to search memory, use the command

$W

As the search proceeds, the program compares the data at each location with the contents of the word register. If they match, it prints the data and its location. By default, the word register contains zero, which allows all of the memory locations to be printed.

You can set the mask register when you wish to mask the contents of certain bit positions in each memory register during the search. The command that allows you to do this is

$M

When you do not want masking, set the mask register to −1 (all ones). If you want to mask certain bit positions, set those positions to zero and set the unmasked positions to one. The system ANDs the contents of the mask register with the contents of the memory location and then compares the result with the contents of the word register. By default, the mask register is set to zero (all bit positions masked). Therefore, if a search requires comparison of memory contents with the word register — when the word register is nonzero — the user must set the contents of the mask register.

The following example shows how each command operates.

```
$W 000000 15000 ⟨CR⟩
$W 015000
$M 000000 177700 ⟨CR⟩
```

The $W command opens the word register and stores 15000. The second $W command displays the current word register setting. The $M command then opens the mask register and masks bits 0 through 9.

Number Register

The number register determines whether the contents of the registers will be printed out in octal or decimal form. The command used to set the number register is

$N

By default, the number register is set to zero and causes the contents to be printed out in octal. When the number register is set to nonzero, the contents of memory and special registers will be printed out in decimal. An example of the command is

```
START/006017 ⟨CR⟩
$N 000000 1 ⟨CR⟩
START/+3087 ⟨CR⟩
$N +1.0
```

The slash (/) causes the contents of START to be displayed in octal form. The $N command then changes the display format to decimal, and the contents of both START and the number register itself are then displayed in decimal form.

Symbol Table Pointer Register

You open the symbol table pointer register with the command

$Y

This register contains a pointer to the beginning of the User Status Table. In RDOS background loading, the table begins at location 402, which is labeled USTSS. In RDOS foreground loading, USTSS can be a different location. An example of the command is

```
$Y 000402 ⟨CR⟩
402/TMIN+63↓
+403 TMIN+52↓
+404 TMIN+64↓
+405 TMIN↓
+405 DEBUG↓
+407 TMIN+64↓
+410 +0↓
+411 177777
```

In this command sequence, the $Y command opens and examines the contents of the symbol table pointer register. The remainder of the sequence examines a part of the contents of the User Status Table, starting at location USTSS. (This example uses RDOS DEBUG with a minimum task scheduler, a single task program.)

Search/Punch Register

Bit 15 of the search/punch register specifies the output device to be used when printing memory search results. When bit 15 is set to zero (the default setting), the search results are printed at the console. To print search results at the line printer, bit 15 must be set to one. The register can be opened with the command

$H

Bit 0 of the search/punch register specifies the output device to be used when punching portions of the user program. When bit 0 is set to zero (the default setting), it punches output to the teletypewriter punch. When set to one, bit 0 punches output to the high-speed punch.

For example, the command

```
$H 000000 1 ⟨CR⟩
```

opens the search/punch register and sets bit 15 to one, causing the search output to be printed to the line printer.

Interrupt Register

The interrupt register determines whether interrupts are to be enabled. This register is used only in the IDEB version of the Symbolic Debugger. It can beopened by entering the command

$I

By default, the register is set to zero, which enables interrupts. To disable interrupts, the register is set to −1 (all ones). The command

```
$I 000000 −1 ⟨CR⟩
```

opens the interrupt register and disables interrupts.

NOTE: In IDEB under RDOS mapping, the interrupts necessary for mapping are never disabled.

Task Control Block Register

The task control block register contains the address of the task control block (TCB) of the currently executing task. It is opened with the command

$T

and contains status information for each task needed by the task scheduler in multitasking. This command is used only for RDOS DEBUG.

Console Input Done Register

The console input done register specifies the status of the console input by setting bit 0 of the register. It is used only in IDEB and can be opened with the command

$T

Setting bit 0 to a value of zero indicates that the console input is done. Setting the bit to one specifies that the input has not been completed. In the following example, the coding causes a transfer to the debugger while console input is still outstanding.

```
A:      SKPDN TTI
        JMP.-1
        DIAC 0 TTI
$T 100000
```

The first three lines of the example set the breakpoint at that instruction. The command $T sets bit 0 of the register to one.

Carry and Console Output Done Register

The carry and console output done register determines the status of the Carry flag in bit 0. When bit 0 is set to zero, the Carry flag is zero. When bit 0 is set to one, the Carry flag is one.

Bit 15 of the register indicates the status of output to the console. By default, this bit is set to zero, which signifies that the output is not done. Setting the bit to one indicates the output is done. The console output done bit is used only in IDEB. The register can be opened with the command

$C

In the following example, the coding causes a transfer to the debugger while the console output is still outstanding and Carry is set.

```
B: ADCO 0 0
DOAS 0 TTO
SKPDN TTO
JMP . -1
$C 100001
```

In the first four lines of this example, the breakpoint is set at that instruction. The command $C sets bits 0 and 15 to one.

Starting Location Register

When debugging under RDOS, the starting location register contains the contents of USTSA — the address of the task scheduler — thus allowing control to be transferred to the scheduler to run the task with the highest priority. The starting location register can be opened with the command

$L

An example of this command is

```
$L 000764
```

which determines the address of the task scheduler.

Extended Save Address Register

The extended save address register contains the contents of User Status Table at location 421, USTSV, which points to the Extended Save State routine. You open the register with the command

$E

Chapter 9

Symbolic Debugger Breakpoints and Program Restarts

Breakpoints are key elements in debugging. They permit you to execute a small portion of a program and then check its status.

One or more breakpoints can be set in a program using a debugger command. You can indicate when a breakpoint should interrupt program execution and cause a transfer to the debugger. In effect, when the breakpoint is encountered, the program instruction at which the breakpoint was set is transferred to the debugger, and a JMP instruction to the debugger is substituted in the user program.

Eight locations of page zero relocatable code are reserved for the eight debugger breakpoints. Any attempt to place other information in these locations and then execute them wipes out the user program.

Breakpoints are assigned values in reverse numerical order. For example, if you set four breakpoints in a program, the first breakpoint is numbered 7, the second 6, the third 5, and the fourth 4.

Setting a Breakpoint

The debugger command

addr$B

sets a breakpoint at the program address that is specified. Breakpoints should not be set at the following four locations:

1. Data Words.
2. Instructions modified during execution.
3. Locations where interrupts cannot be delayed for relatively long times.
4. The second word of two-word instructions.

In the following example, three breakpoints are set:

```
START$B
START+33$B
START+42$B
```

Examining Breakpoint Locations

To determine where breakpoints are currently set in a program, use the command

$B

This command prints breakpoint numbers and the locations at which they are set. Breakpoints are printed in descending numerical order. The following example examines the breakpoints set in the previous section.

```
$B
7B START
6B START+33
5B START+42
```

Deleting Breakpoints

The format for the command that deletes a single breakpoint is

n$D

where *n* is the number of the breakpoint to be deleted. The remaining breakpoint numbers stay the same. For example, if breakpoints 7, 6, and 5 are set, the command

```
6$D
```

deletes breakpoint 6; the numbers assigned to the other breakpoints remain unchanged.

All breakpoints in a program can be deleted by issuing the command

$D

Breakpoint Counters

Associated with each breakpoint is a break proceed counter that indicates when, during execution of the program, encountering that breakpoint will cause a switch to the debugger.

When a breakpoint is set, the break proceed counter for that breakpoint is set by default to one, which is the number of times the instruction at the breakpoint will be executed before the debugger is reentered. This means that when you restart the execution at that particular breakpoint, the instruction will be executed, but the debugger will be reentered the next time that breakpoint is encountered. The command

n$Q

opens the break proceed counter of breakpoint *n* for examination and possible modification. The variable *n* represents the number of the previously set breakpoint. You can modify the contents of the break proceed counter by typing the new contents immediately following the printout of the current contents. The following command executes the instruction at breakpoint 7 twice before reentering the debugger.

```
7$Q 000001 2
```

Program Restart Commands

You can restart a debugged program at any address within the program by issuing the command

addr$R

The program executes, looping through any breakpoints the number of times that the break proceed counters indicate; this continues until a breakpoint is encountered, provided one is set. An example of the command is

```
START+2$R
6B START+10
0 001654 1 000000 2 000000 3 006162
```

Restarting the Program at a Breakpoint

When a breakpoint causes transfer to the debugger, you can issue debugging commands and restart the program execution at the breakpoint by issuing the command

$P

The program executes, looping through the breakpoint the number of times that the break proceed counter indicates. When the debugger is reentered, the breakpoint and the contents of the registers are printed. An example of the command follows.

```
$R
7B START
0 006207 1 006162 2 000004 3 006162
$P
6B START+10
0 001654 1 000000 2 000004 3 000000
```

Overriding the Break Proceed Counter

To override the contents of the break proceed counter when restarting execution at a breakpoint, use the command

n$P

where *n* is the number of times the breakpoint instruction is to be executed. If, in the previous example, you used the command 5$P in place of $P, the results would be as follows:

```
$R
7B START
0 006207 1 006162 2 000004 3 006162
5$P
6B START+10
0 001654 1 000000 2 000004 3 000000
```

Chapter 10

Symbolic Debugger Search Commands

All or part of memory can be searched for a given bit configuration, and the matching addresses and their contents are output at the console or on the line printer. To perform a search, follow these steps:

1. Open the word register and place in it the configuration to be used in the search.
2. Open the mask register and set to one those bit positions that will contain the configuration for which you are searching.
3. Open the search increment register and set the search increment.
4. Give a search command, indicating whether all or part of memory is to be searched for the desired configuration.

The search algorithm is flowcharted in Figure 10.1 and operates as follows:

1. The contents of the address are ANDed with the contents of the mask register.
2. The result of step 1 is compared with the contents of the word register.
3. If the comparison in step 2 results in a match, the address and its contents are output.
4. The next designated location is searched.

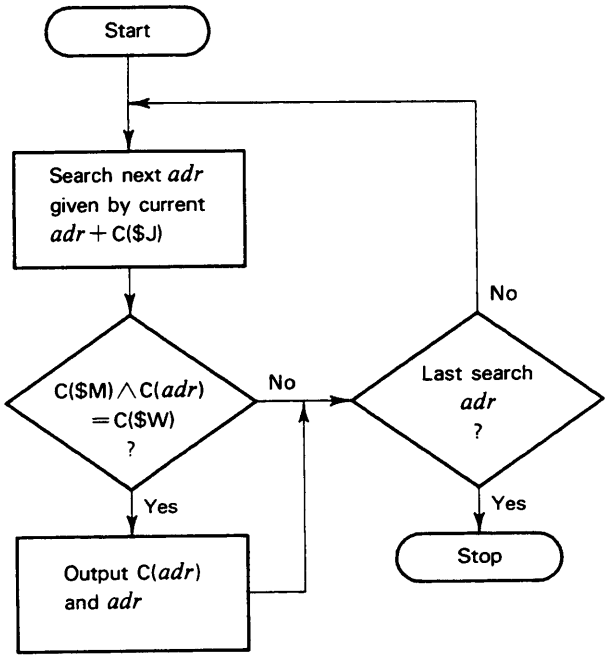

Figure 10.1 Search algorithm ID-00383

By default, the contents of both the word and mask registers are zero and the search increment is one. If default settings are used, all locations in the range given by the search command will be output. To inhibit all masking, set the mask register to −1.

The range of memory to be searched is determined by the individual search command. There are four different formats for the command.

The command

$S

causes all of memory to be searched.

The command

addr$S

causes a search from the start of memory through *addr*, the address specified in the command line.

The command

addr⟨$S

causes a search from *addr* through the end of memory.

The command

addr⟨*addr*$S

causes a search from the first address through the second address.

An example of a search command is:

```
$W 000000 6017 ⟨CR⟩
$M 000000 -1 ⟨CR⟩
START⟨START+40$S
START 0006017 ⟨CR⟩
START+10 006017 ⟨CR⟩
START+14 006017 ⟨CR⟩
START+20 006017 ⟨CR⟩
START+23 006017 ⟨CR⟩
START+27 006017 ⟨CR⟩
START+33 006017
```

The $W command puts the .SYSTM configuration in the word register. The $M command then sets the mask register to all ones, and the $S command activates a search over a range of 41 locations.

In IDEB, a search can be terminated by striking any key. In RDOS DEBUG, there is no way to terminate a search; therefore, you should avoid possible interminable searches whenever possible. IDEB also allows you to direct search output to the line printer with the output register.

Chapter 11

Symbolic Debugger Symbol Recognition

When the debugger is invoked, the symbol tables of all programs loaded with the debugger are available to the debugger. The symbol tables contain all global symbols and can contain local symbols as well.

By default, no user symbols are loaded. However, you can load user symbols contained in one or more of your programs by appending the local switch /U to the name of the program or programs in the RLDR command line. The following two examples of relocatable loader (RLDR) command lines illustrate the loading of local symbols.

```
RLDR/D A/U B/U C/U
```

This command line loads local symbols for all user programs.

```
RLDR/D A/U B C MIDEB
```

This command line loads local symbols for program A only.

During debugging, you may wish to disable debugger recognition of some or all symbols temporarily. You can re-enable recognition of these symbols at a later time.

Disabling Global and Local Symbols

To disable debugger recognition of all symbols, issue the command:

```
$K
```

The following is an example of the $K command.

```
START/006017
$K
START/U
```

The debugger recognizes START until the $K command is issued. Then the debugger indicates that START is an unknown symbol.

Enabling Global and Disabling Local Symbols

To enable or re-enable all global symbols while disabling (or continuing to disable) all local symbols, enter the following command:

n$K

where *n* can be any single digit.

An example of the command is:

```
$K
START/U
```

This section disables the global symbol START.

```
0$K
START/006017
```

This section re-enables the global symbol START.

At the same time that all global symbols are enabled by the *n*$K command, all local symbols are disabled. Since local symbols are often not loaded, the command is commonly used to re-enable all previously disabled ($K) global symbols.

Enabling All Symbols

You can re-enable all global and local symbols, removed with either *n*$K or $K, by issuing a command with the format:

name%

where *name* is the name of the loaded program.

To enable local symbols, the local /U switch must be given in the RLDR command line, using the *name%* command.

Removing a Symbol From Output

A given symbol can be removed from debugger output with the command:

sym$K

where *sym* is the name of the symbol to be removed.

This command affects output only. The symbol will still be recognized on input to the debugger. Once removed, the symbol cannot be restored to output during debugging.

When a symbol is removed with this command, the debugger replaces the symbol on output with an offset from the nearest previous symbol, provided that the substitute symbol is not more than 2000 locations removed. If there is no previous symbol within 2000 locations, or if no symbols remain, the absolute value of the location is substituted.

Chapter 12

Symbolic Debugger Output Formats

Output can be formatted in any one of several ways with the debugger. To change from one type of output to another, use the command:

$\$x$

where x is one of the characters in Table 12.1. This table lists and describes the commands with which you can output the contents of a location.

Command	Meaning
=	Formats output in octal or decimal numeric datum.
;	Formats output in NOVA instruction.
:	Formats output in symbolic format in which all enabled symbols are printed.
'	Formats output in two ASCII characters. Nonprinting ASCII characters are printed in octal code equivalents enclosed in angle brackets.
&	Formats output as byte pointer. The first 14 bits of the word are an address that contains or is to receive a byte, and the last bit specifies which byte -- left or right -- the word is to receive (0 is left and 1 is right).
(←)	Formats output in two numeric half words.
?	Formats output as .SYSTM command mnemonics.

Table 12.1 Symbolic Debugger output format commands

Shown below are a few examples of the output format command.

```
$'
START/⟨14⟩⟨7⟩ ⟨CR⟩
START+1 ⟨3⟩⟨0⟩ ⟨CR⟩
START+2 ⟨14⟩; ⟨CR⟩
START+3 ⟨252⟩P
$?
START/.WRB 17 ⟨CR⟩
START+1 .MEM 0 ⟨CR⟩
START+2 .WRB 73 ⟨CR⟩
START+3 125000 20 ⟨CR⟩
$←
START/14 17 ⟨CR⟩
START+1 3 0 ⟨CR⟩
START+2 14 73 ⟨CR⟩
START+3 252 120
```

To examine a given word in another format, open the location and enter one of the seven output format commands, or the following command:

* *symbolic format* with bit 0 set to zero

For example, if the current output format mode is .SYSTM mnemonic mode, the location can be opened and "translated" into the other output modes as shown:

```
START+15/.RDL 0 :DSZ+0 3 '⟨33⟩⟨0⟩ :15400
START+15/.RDL 0 =015400__33 0 &006600 0 *015400
```

The location as well as the contents can be transliterated as follows:

```
START+10=000773
```

In this way, any expression consisting of octal and decimal integers, all symbols known to the debugger, and the plus and minus operators may be converted to their equivalent instruction or to byte pointer formats. Thus, the = command, for example, can be used for expression evaluation.

```
100+25.=000131 '⟨0⟩Y:+131
100+25.;JMP+131__0 131 &000054 1
100+25.?.CREA 31
```

Chapter 13

Saving a Debugged Program

The Symbolic Debugger provides a means of saving the current status of a debugged program. The command

$V

allows you to exit the debugger with the current state of the debugged program, including the current status of all registers and patches made during debugging.

In RDOS DEBUG and IDEB, the $V command returns you to the CLI level with the current state of the debugged program in the BREAK file. If you wish to save the BREAK file, issue a CLI SAVE command at this time.

The following is an example of how the command works:

```
DEB ALPHA
  .
  .              (Debugging commands.)
  .
  .
$V               (Return to CLI.)
BREAK
R
SAVE ALPHA       (Save BREAK file under the name ALPHA.)
R
```

Before issuing a $V command, you should delete all breakpoints ($D). Otherwise, an attempt to debug the saved break file will result in a halt. Note that the $V command closes all open files. To execute the break file after issuing a $V, provide a routine that reopens all files that were open at the time the command was issued.

Part IV

ENPAT

Chapter 14

ENPAT

A patch is a one-word change made in a save or overlay file. ENPAT systematically encodes and enters data into a patch file for use by PATCH.SV, the patch utility. ENPAT checks syntax on all data input, but does not evaluate expressions or install patches. Patches are most often used to update operating system files; Data General supplies current patches with its system software.

The ENPAT command creates the patch file *filename*, or opens it for appending if it already exists. It then asks questions about each patch, accepts valid answers, and places them in *filename*. To install the patches in a save or overlay file see PATCH, Part V.

Command Format

The command format for the ENPAT utility is

ENPAT *filename*

When the filename already exists, it will be opened with an append, so that any data already contained in the patchfile will not be lost. If the filename does not already exist, it will be created, and the message

CREATING NEW PATCHFILE

will be output to the console.

No global or local switches are available with this command.

Operating ENPAT

Once ENPAT is in operation, it asks the following series of questions.

SAVE FILE (0) OR OVERLAY FILE (1)?

Answer 0 (or press ⟨CR⟩) when the patch is for a save file; answer 1 when the patch is for an overlay file. An invalid answer causes the question to be repeated.

PATCH LOCATION?

Type the location to be patched. You must enter one of the legal expressions that are defined in the next section of this chapter. Any illegal input results in an error message describing the problem. This message is displayed on the console and is followed by a repeat of the question.

CURRENT CONTENTS?

Type the original contents of the patch location. Any legal expression is accepted as input. When the expression is illegal, an error message will be output to the console and the question is repeated.

NEW CONTENTS?

Enter the new contents of the patch location. Only legal expressions are accepted. When the input is illegal, a message appears and the question is repeated.

CONDITIONAL?

Patches can be installed conditionally. Five types of conditional patching are allowed:

1. When a patch is to be installed under all circumstances, enter a Carriage Return.
2. When the patch is to be installed only if a symbol is defined in the loadmap (and is not a −1), enter the symbol name.
3. When the patch is to be installed only if a symbol is not defined in the loadmap (−1 is considered not defined), enter a hyphen (-) followed by the symbol name.
4. When the patch is to be installed only if the answer to a question is affirmative, enter the question to be asked, preceded by an open quote (''). A one is considered to be an affirmative answer; a zero or a Carriage Return are considered negative answers. This question is output to the console by the PATCH utility before the patch is installed.
5. A patch is a 1-word change to a save or overlay file. However, many changes or corrections involve multi-word changes. If the multi-word patch is to be installed conditionally, it is inconvenient to enter the same conditional many times. ENPAT accepts a SHIFT-6 ( ) as a conditional to refer to the last conditional resolved. Therefore, if a 3-word patch is to be conditionally installed in answer to a question, the conditional for the first word of the patch would be the question itself, and the conditionals for the second and third words need

only be the character (↑). If the answer to the question is affirmative, all three words of the patch are installed; if the answer is negative, none of the three words of the patch are installed.

NOTE: The conditional (↑) can be used only if the patch line in which it occurs follows a line which is conditional.

EXIT (0 = NO, 1 = YES)

When you have no further patches to input, type 1 to return to the next higher level (usually the CLI). If you wish to continue, type 0 or press the Carriage Return. ENPAT then issues a blank line followed by the first question.

The above questions generate one line of the patch file, which is the information needed to install one word of patch. If at any time while answering these questions you wish to void the current patch file line being created, type CTRL-C to restart the current patch file line at question one.

Expressions

Legal expressions are of the form E1 or E1*E2, where E1 and E2 are expressions, numbers (octal only), or symbols, and * is an operator. Legal operators are:

+	Addition
−	Subtraction
!	Inclusive OR
#	Exclusive OR
⟩	Right shift, zero fill
⟨	Left shift, zero fill
&	Logical AND
@	Indirection

Expression Evaluation

The ENPAT utility does not evaluate expressions. Its function is to check syntax and enter patches into the patch file. Expression evaluation is performed by the Patch utility. The following information is included only to aid in correctly entering data in the patch file.

All expressions are evaluated from left to right, except when parentheses are encountered. In this case, the expression within the parentheses is evaluated first. Expressions can contain up to 132 (decimal) characters and multiple sets of parentheses; however, nested parentheses are not allowed and will be flagged by ENPAT as an illegal expression. The following complicated expression is legal and is evaluated in the following manner:

5+(SYMBOL@0-6000)@1⟩1#2000

where the address of SYMBOL is 1000, the contents of 1000 is 6050, the contents of 55 is 100, and the contents of 100 is 4011.

SYMBOL@0	=	6050
6050-6000	=	50
5+50	=	55
55@1	=	4011
4011⟩1	=	2004
2004#2000	=	4

Negative Numbers

Negative numbers can be entered as $-n$ where n is an octal number of six digits or less. For example, the number negative one can be entered as −1 or 177777.

Symbols

When a symbol name is used, it is evaluated as its address, which is given in the load map. The indirection operator can be used to transfer the contents of the address to as many as 65,536 levels. For example, the address of symbol X as found in the load map is 1000. The contents of 1000 is 100. The contents of 100 is 10. Therefore, X resolves to 1000, X@0 resolves to 100, and X@1 resolves to 10.

Entering Patches for a User Save File

When you enter patch locations for a user save file, you must compensate for the fact that the user save file starts at location 16. All memory references must be offset by −16. For example, to reference ZMAX in a user save file, which is found at location 401 when the program resides in memory, you must enter the location as 401 − 16.

Entering Patches for a User Overlay File

When entering patches for a user overlay file, you must supply a nonsymbolic expression that evaluates to the absolute octal address of the location to be patched.

Comments

Comments can be inserted in any patch file using one of the text editors. These comments will then be output to the console by the Patch utility. Comments take one of the following three forms:

1. A ? in the first position of a line causes a Form Feed to be sent to the console. If the console is a CRT, this will clear the screen. If the console is a teletypewriter, this wastes a lot of paper. Do not worry if you have a teletypewriter and you do not always want to see the comments, because the Patch utility allows for the optional suppression of all types of comments to the console.

2. A ; in the first position of a line causes the remainder of the line to be echoed to the console.

3. A : in the first position of a line causes the message

STRIKE ANY KEY TO CONTINUE

to be output to the console. No further patch file evaluation is performed until you press a key. This can be used to prevent the dialogue contained in the patch file from running off the screen before it can be read.

ZMAX and NMAX

The current ZMAX and NMAX values for a save file can be referenced symbolically as ZMAX and NMAX. These values are also stored in or above NMAX, and it is important to update locations 401 and 404 accordingly. This does not change the values of the symbolic ZMAX or NMAX.

Part V

PATCH

Chapter 15

Introduction to PATCH

The PATCH utility installs the patches that you inserted in a patch file by means of the ENPAT utility. To apply Data General-supplied patches to an operating system, you must have instructed SYSGEN to save a load map file. When no map file exists for a program, you can patch it by appending the global switch (/N) to PATCH. If you omit arguments, PATCH will ask for them.

Command Format

The command format for the PATCH utility is:

PATCH *savefile*/S*patchfile*/P*loadmap-file*/L

When PATCH runs, it creates a patch dialogue file named *savefile*.PD, deleting any file of the same name first. This file records patch date, time, and locations for the last patch of *savefile*. Patches applied during the last run of PATCH are marked with an asterisk in the dialogue file.

Switches

Both global and local switches are used with this utility. The legal switches are described in Table 15.1.

Switch	Description
Global:	
/I	Inhibits the output of comments from the Patch file to the console ($TTO or $TTO1).
/N	No load map available. This is described in more detail in the following section.
Local:	
/S	Save filename (with or without the extension). This allows you to open the save file, optional overlay file, and the dialogue file.
/L	Load map filename. When an extension exists, it must be included in the file name. If the load map is in a subdirectory, the subdirectory must be initialized.
/P	Patch filename.

Table 15.1 Global and Local Patch switches

Patching Without a Load Map

A load map file will not be used when the global switch /N has been specified. If it encounters any symbols while evaluating the Patch file, the utility displays the symbol name in question in quotes and asks for the address of the symbol. You must respond with an octal number of less than six digits, terminated by a Carriage Return. Negative numbers are in the same form, but they are preceded by a minus sign (−).

Patch Dialogue File

While it modifies the save or the overlay file with PATCH.PF, the Patch utility builds a file that has the save filename and a .PD extension. This file contains a line with the date and time of the patching, a list of all locations patched, and a string indicating whether the patch was installed. An asterisk preceding this line indicates that the patch has been made on the current run, rather than a previous run.

Appendix A

Command Summaries

Command	Description
/	Opens the current word and displays the contents in the current format.
'	Displays the last word or expression in ASCII format.
—	Displays the last expression in half-word format as two octal numbers.
=	Displays the last expression in the word format in the current number base.
*	Displays the last expression as an octal word.
⟨CR⟩	Stores the result of the expression in the currently opened word. If no word is currently opened, this is a no-op.
⟨NL⟩	Stores the result of the expression in the currently opened word and displays the next word.
↑	Stores the result of the expression in the currently opened word and displays the previous location.
$	Enters the escape command mode. The character following this display command is an escape command.

Table A.1 DSKED display commands

Command	Description
A	Prints the block number currently in core.
B	Changes the name (address) of the block core.
C	Calculates the expression and prints the result. If used alone, the command prints the current location.
D	Dumps the specified disk locations.
F	Fills the specified disk locations with a word from the word register.
H	Opens the output register for display and modification. Directs the output of Search and Dump commands.
J	Opens the increment register for display and modification.
M	Opens the mask register for display and modification.
N	Opens the number register for display and modification. This register defines the output format of numbers.
O	Forces the block to be output to disk.
R	Forces the block to be read into core.

Table A.2 DSKED escape commands

Command	Description
S	Searches the disk for a specified value.
U	Restarts DSKED at the query: *DISKEDIT DISK DRIVE MODEL NUMBER?* and resets registers to startup values.
W	Opens the word register for display and modification.
Z	Writes any modified block in core and halts the program.
=	Outputs numbers in word format.
'	Outputs numbers in ASCII format.
—	Outputs numbers in half-word octal format.
*	Outputs numbers in word format.
:	Outputs disk block number(s) in double-precision octal format.
\	Outputs disk block number(s) as two octal words.
,	Outputs disk block number(s) in head, sector, and cylinder format (fixed-head disks only).

Table A.2 DSKED escape commands (continued)

Command	Description
/	Use contents as *addr*, open, and display
addr/	Opens *addr*, displays contents.
addr!	Opens *addr*, displays nothing. Opens next *addr*, displays contents.
↑ (SHIFT-6) or SHIFT-N	Opens previous *addr*, displays contents.
⟨CR⟩	Closes current location (if open), inserts new value (if any), gives prompt.
.=	Displays current location in current radix.
.:	Displays current location symbolically.
=	Displays last entry in current radix.
$=	Changes display mode in numeric, current radix (default is 8)
;	Displays last entry as an instruction.
$;	Changes display mode to instruction.
'	Displays last entry in ASCII.
$'	Changes display mode to ASCII.
\	Displays last entry in .SYSTM format.
$\	Changes display mode to .SYSTM format.
:	Displays last entry in symbolic format.
$:	Changes display mode to symbolic format.
*	Displays last entry as symbol, OBO.
n.=	Converts decimal *n* to current radix.
*nx*H	Converts hex *nx* to current radix (*n* = 0 – 9).
*word*1 [*word*2]	Converts 2 radix 50 words to symbol.
"*xx*=	Displays ASCII *xx* in current radix.
byte]*byte*=	Displays *byte*, *byte* in current radix.
byte]*byte*	Inserts *byte*, *byte* word in open addr.
+	Add one number to another or to an address.
–	Subtracts one number from another or an address.
name%%	Enables output of all local and global symbols in module *name*.

Table A.3 SEDIT commands

Command	Description
$D	Displays all locations on file.
addr$D	Displays locations from 0 through *addr*.
addr⟨$D	Displays locations from *addr* to end.
addr⟨*addr*$D	Displays locations *addr* through *addr*.
$H	Opens output register, displays contents. Zero sends $D and $S output to console, 1 to line printer ($LPT).
$J	Opens search increment register (used in $D and $S), displays contents.
$K	Disables output of all symbols.
n$K	Enables all global (.ENT) symbols.
name$K	Disables output of symbol *name*.
$M	Opens mask register (used with $W, in S commands), displays contents. Inserts –1 to match a specific word.
$N	Opens number register, displays contents. Inserts 100000 to sign output, 40000 to drop leading zeroes, 12 or 10. for decimal output; 0,8. or 10 for octal output; and 20 or 16. for hex output.
ovname$O	Loads overlay named by .ENTO into node.
$O	Loads overlay 0 into save file node 0.
node]*ov*$O	Loads overlay *ov* into *node*. Node number goes in left byte, overlay number in right byte.
$S	Searches all locations for value in $W (also uses value in $M, $J, $H for all searches).
addr$S	Searches locations from 0 to *addr*.
addr⟨$S	Searches locations from *addr* to end.
addr⟨*addr*$S	Searches locations *addr* through *addr*.
$W	Opens word register, displays contents. S commands search for the value in the word register.
$Z	Stops SEDIT and returns to CLI.
CTRL-A	Stops current SEDIT command, returns prompt.
CTRL-Q	Resumes console display, suspended by CTRL-S.
CTRL-S	Suspends console display, waits for CTRL-Q.

Table A.3 SEDIT commands (continued)

Command	Type	Meaning
addr!	Monitor Core*	Opens core location *addr*.
addr/		Opens core location *addr* and prints contents.
⟨CR⟩		Closes open location.
		Closes open location and opens subsequent location.
		Closes open location and opens the previous location.
$A	Monitor Accumulators	Prints contents of all accumulators.
n$A		Opens accumulator *nn* = 0 – 3 in NOVA computer; 0 – 7 in ECLIPSE computer.
$C	Monitor Special Registers	Opens the carry register.
$E		Opens the extended save register (USTSV).
$F		Prints contents of floating-point registers.
$H **		Opens the search punch register.
$I **		Opens the interrupt register.
$J		Opens the search increment register.
$L		Opens the location register (USTSA).
$M		Opens the mask register.
$N		Opens the number register.
$T		Opens the task control block register (USTCT).
$W		Opens the word register.
$Y		Opens the symbol table pointer register.

Table A.4 IDEB and DEBUG commands

Command	Type	Meaning
$B	Breakpoint	Prints location of all user program breakpoints.
addr$B		Inserts breakpoint at location *addr*.
$D		Deletes all breakpoints.
n$D		Deletes breakpoint *n*. (*n* = 0 – 7).
$V	Break file	Puts debugged program in break file for possible save of file.
$P	Execute (and Break Proceed Counter)	Restarts execution from a breakpoint with break proceed counter set to + 1.
n$P		Restarts execution from a breakpoint with break proceed counter set to *n*.
n$Q		Opens break proceed counter *n* (*n* = 0 – 7).
$R		Restarts execution at address stored in USTSA.
addr$R		Restarts execution at *addr*.
$K	Symbol Enable and Disable	Removes all local and global symbols from input and output.
n$K		Removes all local symbols from input/output but retains (or enables) globals.
sym$K		Removes symbol *sym* from output permanently.
name		Enables all local and global symbols in program *name* (or restores to output all symbols removed by $K or *n*K commands).
$S	Core Search*	Searches all memory.
addr $S		Searches memory from location 0 to *addr*.
$addr_1$⟨$S		Searches memory from location $addr_1$ to limit of memory.
$addr_1$⟨$addr_2$ $S		Searches memory from location $addr_1$ to $addr_2$ inclusive.

Table A.4 IDEB and DEBUG commands (continued)

Command	Type	Meaning
=	Format of Data Output to the Teletype-writer	Prints last data numeric format.
:		Prints last typed data in symbolic format.
;		Prints last typed data in instruction format.
←		Prints last typed data in instruction format.
'		Prints last typed data in ASCII format.
&		Prints last typed data in byte pointer format.
*		Prints last typed data in symbolic format with bit 0 = 0.
?		Prints last typed data in .SYSTM command format.
$:		Prints subsequent data in symbolic format.
$;		Print subsequent data in instruction format.
$←		Prints subsequent data in half-word format.
$'		Prints subsequent data in ASCII format.
$&		Prints subsequent data in byte pointer format.
$=		Prints subsequent data in numeric format.
$?		Prints subsequent data in .SYSTM command format.

* Under the mapping option, core examined must be within your address space.
** IDEB only.

Table A.4 IDEB and DEBUG commands (continued)

Appendix B

Error Messages

DSKED Error Messages

Many of the following error messages apply only to specific disk drive models. Some are the result of formatting errors on the disk and hardware malfunctions. Refer to the Peripherals manual (DGC No. 014-000632-01), for details on hardware induced errors.

ABORTING OPERATION

Execution of the current DSKED command is being terminated.

ADDRESS ERROR

The address read from the beginning of a track or sector does not match the track address specified in the controller's address register.

ATTEMPTED WRITE TO WRITE-DISABLED DRIVE

You have attempted to write to a write-disabled drive.

BAD SECTOR

The sector cannot be read because of physical damage, contamination, or an error in the sector header (format error).

BLOCK COUNT TOO BIG

You have attempted to transfer too many blocks at one time.

CHECKSUM ERROR

The checksum computed by the controller does not match the checksum read from disk block.

CONTROLLER OVERWRITE IN MEMORY BEYOND END OF BUFFER

CYLINDER ADDRESS ERROR

A cylinder address error is detected.

DATA CHANNEL DATA ERROR

An error in data transmission occurred somewhere along the internal data paths of a 6063/6064 disk subsystem.

DATA LATE ERROR

You have lost data during a read from disk because the CPU data channel transfer was not enabled in time to prevent over-write of data in the device buffer. Or, you have lost data during disk write because it was not in the device buffer when that word position was under the read/write head.

DEVICE TIMEOUT: xxxx

The device failed to perform a specified operation within a program-determined time period.

DISK ERROR

More than one error status bit is set.

DISK STATUS - DIA–nnnnn DIB–nnnnn DIC–nnnnn
ALT MODE 1 - DIA–nnnnn
ALT MODE 2 - DIA–nnnnn DIB–nnnnn

Applies only to disks: 6063/6064 and 6060/6061/6067/6122.

DRIVE UNSAFE

This is a hardware fault that can be cleared by cycling power to the disk drive. If this fails to clear the error, the drive may need repair.

DRIVE UNSAFE OR ADDRESS ERROR

Refer to drive unsafe and address error descriptions.

DRIVE UNSAFE OR WRITE TO WRITE-DISABLED DRIVE

Refer to Drive Unsafe or Write Locked error descriptions.

DUAL PROCESSOR OPERATION NOT SUPPORTED

This program may not operate properly when the other processor in a dual processor system is running a program.

ECC ERROR

A correctable disk data error is detected during a read operation.

ECC ERROR (UNCORRECTABLE)

An uncorrectable disk data error is detected during a read operation.

END OF FILE: FILE xxxxxx

You have reached the end of the file.

HARDWARE TIME OUT

The device failed to perform a requested operation within an internally determined time period.

ILLEGAL COMMAND (HARDWARE)

You have attempted to initiate a new command before the completion of the currently executing one; or you have issued a command out of proper sequence.

ILLEGAL DISK ADDRESS

You have specified an illegal disk address.

ILLEGAL DISK UNIT DECLARATION

You have specified a unit number that exceeds the maximum unit number for this controller or is not a numeric character.

INCOMPLETE TRANSFER

The device register(s) do not contain the expected data. An error flag is not set with this error.

INCORRECT DISK TYPE

You have entered a type of disk that is not serviced by the DSKED utility.

INCORRECT MEM ADDR &/OR SECTOR CNT FROM CONTROLLER

The memory address or the sector count, or both, read from the controller, do not match those specified by the program.

INVALID BAD BLOCK TABLE

One or more errors have occurred in Block 4, the RDOS bad block remapping table.

INVALID DISK NAME

Disk name specified is illegal.

INVALID DISK ID BLOCK

One or more errors or inconsistencies have occurred in Block 3, the RDOS disk ID block.

LINE TOO LONG: FILE xxxxxx

You have specified a file that contains a line too long for processing.

NO BLOCK IN CORE

This advisory message indicates that there are no disk blocks currently buffered in memory.

NO SUCH TRACK

You have specified a track address that exceeds the maximum track number for the disk device in use.

NOT A TOP LOADER

You have specified a disk drive that is not type 6045 or 6070. This error message is issued if the letter F is appended to a disk unit specifier for a device model that does not have a removable/nonremovable platter combination.

PARITY ERROR

A parity error occurred during data transfer between the controller and adapter in a disk subsystem.

POSITIONER FAULT

The head positioner in a disk drive malfunctioned. This error can be cleared by cycling power. Persistence of this error indicates the need for repair.

SEEK ERROR

The last seek command specified a track number greater than the maximum for the device in use. Another possibility is that the selected device failed to carry out the last seek command issued.

SERVO CLOCK FAULT

The servo clock of a disk drive malfunctioned. This error can be cleared by cycling power. Failure to clear this error indicates a need for repair.

SURFACE OR SECTOR ADDRESS ERROR

A surface or sector address error was detected during a sector header check. Reformatting may be required if retries fail to clear this condition.

TIME OUT

A device failed to execute a command within the program-determined time period.

TRACK BOUNDARY

You have reached the end of the track or cylinder.

UNDER-VOLTAGE FAULT

The power supplied to the disk drive is below normal voltage and it may be failing. The disk operation under these conditions can be unpredictable.

UNKNOWN DRIVE TYPE

You have specified a disk drive unknown to the system.

UNKNOWN ERROR CODE

You have created an error unknown to the system.

VERIFY ERROR

A data error occurred during a verify operation.

WRITE FAULT

A malfunction in the write circuits or a read/write occurred during a write operation.

WRITE LOCKED AREA

You have attempted a write operation on a write protected track (NOVA disk only).

SEDIT Error Messages

? or __?

Your SEDIT command syntax was wrong, or you pressed RUBOUT. SEDIT outputs a Line Feed and awaits another command.

ERROR IN ACCESSING PRINTER

When you directed output to the line printer (by setting $H to nonzero), SEDIT could not open the printer or could not write to it. Perhaps another program is using it.

FILE IS SEQUENTIAL: filename

SEDIT cannot edit a sequential file.

FILE NOT FOUND: filename

SEDIT searched for the file specified, then tried searching for the file with the .SV extension, but could not find the file.

HAD PROBLEM OPENING FILE: filename

SEDIT could not find, or could not open, the file specified.

ILLEGAL OVERLAY ADDRESS

This is an internal SEDIT error. The node address is somehow incorrect.

ILLEGAL OVERLAY NODE NUMBER

Either you have specified a node or overlay number higher than the file contains, or there is no overlay directory.

INSUFFICIENT MEMORY FOR OVLY TABLE

On a $O command, SEDIT tried to read the overlay directory into memory, but there was not enough memory available.

NO INPUT FILE NAME GIVEN IN COMMAND LINE

You must specify a filename for SEDIT to edit.

NO OVERLAY LOADED

No overlay has been loaded ($O) into the node you specified. This message appears only after *some* overlay has been loaded. For example, if your program has two overlay nodes, you might have loaded an overlay into one node and then tried to access the other node.

NOT ENOUGH MEMORY FOR SYMBOL TABLE

The symbol table of the program will not fit into memory. Alternatively, the file is not a save file and you forgot the global /N switch, or inappropriately used the /Z switch.

PROBLEMS OPENING OVERLAY FILE

SEDIT could not read the overlay directory or could not open the overlay file. SEDIT expects the overlay file to have the same name as the save file, with the .OL extension.

SOME PROBLEMS READING SOURCE FILE: filename

An error occurred as SEDIT tried to open the file or read/write to it.

U

You have entered an undefined command or symbol. SEDIT outputs a Carriage Return and awaits another command. To enable *local* user symbols, use the *global* /U switch in the assembler command line and use the *local* /U switch in the RLDR command line; then type *modulename%* from SEDIT.

ENPAT/PATCH Error Messages

All of the following error messages are fatal, and cause a return to the next highest level (usually the CLI).

CONTENTS OF PATCH LOCATION DOES NOT MATCH NEW OR CURRENT CONTENTS SPECIFIED

FIRST CHARACTER IN THE PATCH FILE LINE IS NOT A ";", ":", "?", "S", OR "0"

The first character of a patch file line must be one of these control characters, used to identify the contents of the line.

SAVE FILE ALREADY SPECIFIED

The local /S switch appears on the command line more than once.

LOAD MAP ALREADY SPECIFIED

The local /L switch appears on the command line more than once.

PATCH FILE ALREADY SPECIFIED

The local /P switch appears on the command line more than once.

PATCH LOCATION OUT OF RANGE

The patch location expression, found in the patch file line being evaluated, resolved to a negative address.

INDIRECTION ERROR - ADDRESS OUT OF RANGE

A negative address was encountered while resolving an indirection chain.

UNABLE TO EVALUATE THE PATCH LOCATION FIELD

The expression representing the patch location contains an illegal sequence of characters. Any expression accepted as legal input by the ENPAT utility can be evaluated by the PATCH utility. Therefore, PATCH file data should only be entered using ENPAT.

UNABLE TO EVALUATE THE CURRENT CONTENTS FIELD

The program encountered a negative address while resolving an indirection chain.

UNABLE TO EVALUATE THE NEW CONTENTS FIELD

The program encountered a negative address while resolving an indirection chain.

INSUFFICIENT MEMORY TO HOLD CORE RESIDENT LOAD MAP

When the global /N switch is not specified, the load map input is rebuilt and stored in memory above NMAX, but below HMA. Four words are required per symbol, including the symbol address.

CAN'T MAKE SENSE OUT OF THIS LOAD MAP

The symbol name may have exceeded six characters, or the program found an end-of-file error code while reading the load map before it found the symbol NMAX.

Index

C

D

E

F

G

H

I

J, K

L

M

N

O

P

R

S

T

U

V,W,Z

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA® 4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other _____
(Specify) _____

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

CUT ALONG DOTTED LINE

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD

TAPE

FOLD

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General Corporation, Westboro, MA 01580

069-400020-01