User's Manual

SYMBOLIC DEBUGGER

093–000044–04

Ordering No. 093-000044

Printed in the United States of America
Rev. 04, February 1975

NOTICE

Original Release	August 1970
First Revision	October 1971
Second Revision	December 1971
Third Revision	August 1973
Fourth Revision	February 1975

This revision of the Symbolic Debugger User's Manual, 093-000044-04, supersedes 093-000044-03 and constitutes a major revision to the manual.

INTRODUCTION

The Symbolic Debugger interfaces with user routines, allowing the user to monitor and correct his program during execution. The Symbolic Debugger provides up to eight active breakpoints within a user routine. Accumulators, Carry and memory can be examined and modified during execution using simple debugger commands issued from the console.

The Symbolic Debugger is tailored to the user's configuration. Thus, the symbolic Debugger version that a user receives will depend upon whether he has an ECLIPSE™* or NOVA®* computer and whether his operating system is SOS (Stand-alone Operating System, unmapped RDOS (Real Time Disk Operating System), mapped RDOS, or RTOS (Real Time Operating System).

RDOS users receive two versions of the Symbolic Debugger. The first enables interrupts during debugging and has the generic name DEBUG III (or simply DEBUG) and the second disables interrupts and has the generic name IDEB. The user may load either one of the debuggers with his program.

The different Symbolic Debugger versions are described in Chapter 1. There are very slight language differences among the versions. A language comparison is given in Appendix A and each command description in the manual indicates for which versions of the Symbolic Debugger the command is valid.

*NOVA is a registered trademark and ECLIPSE is a trademark of Data General Corporation, Southboro, Massachusetts.

TABLE OF CONTENTS

CHAPTER 1

GENERAL DESCRIPTION

USE OF 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 is loaded together with the program. The programmer may 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 can correct the contents if necessary. Since the debugger is symbolic, the contents may be examined in source language format, although octal format or a number of special formats may also be used.

The DGC Symbolic Debugger allows the programmer to set up to eight breakpoints within his program. When the program executes, execution will halt before the instruction at the breakpoint location is executed and the programmer can then issue debugging commands. The programmer can then restart execution at the breakpoint instruction or at another location if he wishes.

SYMBOLIC DEBUGGER VERSIONS

The Symbolic Debugger that the user receives is tailored to his computer and system configuration. Thus his version will depend upon whether he has a NOVA or an ECLIPSE computer, whether he has an RDOS or a SOS operating system, and whether his operating system is mapped or unmapped. Users having the RDOS operating system are supplied with two debugger versions. 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 as a relocatable program on a library tape. Following is a list of the different symbolic debugger versions and their tapes:

Version	Title	Tape	Generic Type
NOVA Unmapped RDOS DEBUG	DEBUG[1]	DEB.RB in USYS.LB	DEBUG
NOVA Mapped RDOS DEBUG	MDEBUG[2]	MDEB.RB in MSYS.LB	
ECLIPSE Unmapped RDOS DEBUG	BDEBUG[3]	BDEB.RB in BSYS.LB	
ECLIPSE Mapped RDOS DEBUG	ADEBUG[4]	ADEB.RB in ASYS.LB	
NOVA Unmapped RDOS IDEB	IDEB[1]	IDEB.RB	IDEB
NOVA Mapped RDOS IDEB	MIDEB[2]	MIDEB.RB	
ECLIPSE Unmapped RDOS IDEB	BIDEB[3]	BIDEB.RB	
ECLIPSE Mapped RDOS IDEB	AIDEB[4]	AIDEB.RB	
NOVA Unmapped SOS DEBUG	SADEB	SADEB.RB	SOS DEBUG
NOVA Mapped SOS DEBUG	SAMDEB	SAMDEB.RB	
ECLIPSE Unmapped SOS DEBUG	SABDEB	SABDEB.RB	
ECLIPSE Mapped SOS DEBUG	SAADEB	SAADEB.RB	

Numbers following the titles indicate which debuggers are supplied to a single user, (e.g., MDEBUG and MIDEB are supplied to all users having a NOVA Mapped RDOS system.)

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

RTOS users are supplied with the appropriate version of IDEB.

SYMBOLIC DEBUGGING COMMANDS

A symbolic debugging command has the general format:

[argument] [$] command-code

where: command-code is a single teletypewriter character.

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

argument may be one of the following:

sym	user symbol.
adr	an address having any legal address format: octal or decimal integer, user symbol, or an expression of the form: $x \pm x \pm x \ldots$ where each x is a user symbol, or octal or decimal integer, separated from the following x by either + (plus) or - (minus). Decimal integers must be followed by a decimal point to distinguish them from octal integers.
n	a decimal or octal integer.
name	a user symbol that names a program.
adr <	a range of addresses from adr to 77777.
adr < adr_n	a range of addresses from adr to adr_n.

Commands are described in the chapters following 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.

CONVENTIONS AND SYMBOLS IN COMMAND LINES

) Pressing the RETURN key is represented by the symbol) . There is no printout on the teletypewriter printer when RETURN is pressed.

↓ Pressing the LINE FEED key is represented by the symbol ↓. There is no printout on the teletypewriter printer when LINE FEED is pressed.

CONVENTIONS AND SYMBOLS IN COMMAND LINES

↑ Pressing the SHIFT and N keys causes the symbol ↑ to be printed.

$ Pressing the ESC key causes the symbol $ to be printed.

? Pressing RUBOUT causes the symbol ? to be printed. (See section immediately following.)

CORRECTING TYPING ERRORS

Pressing the RUBOUT key causes the debugger to ignore the current command line and prints a ? . The programmer may then type a new command line. (Any character causing an illegal command line can be used; the RUBOUT convention, however, is convenient.)

DEBUGGER ERROR RESPONSES

An attempt to use an undefined symbol in a command to the debugger will result in an error response of

U

typed immediately following the command, for example:

STAR/U ← User references symbolic location STAR. Symbol STAR is not found in the program. Debugger responds with U and awaits a new command.

All other command errors result in an error response of

?

Typed immediately following the command as shown in the examples following.

ADD@? ← Improper termination of command.

-I$R? ← Illegal address preceding $R.

START+6$B? ← This is an attempt to set a breakpoint at symbolic location START+6. The error response is printed if there are alreal y 8 breakpoints in the program.

$I? ← Attempt to open the interrupt register while in RDOS DEBUG (The register is implemented in IDEB but not in DEBUG under RDOS.)

****

CHAPTER 2

MONITORING MEMORY AND SPECIAL REGISTERS

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

adr/	Open adr and print contents.
adr!	Open adr. Do not print contents.

where: adr may be any acceptable expression defining a location.

When the memory location has been opened, using either the / or ! command, the user may modify the contents of the location and close it or he may close the location without modification.

The user modifies the contents of a location by typeing the new contents on the same line with the command that opens the location.

The user can close the open location in one of four ways:

)	Close the open location (RETURN key)
↓	Close the open location and open the succeeding location (LINE FEED key)
↑	Close the open location and open the proceding location (SHIFT and N keys)
/	Close the open location and open the location specified by the contents of that location.

Commands that open and close locations are used in all debugger versions. Some examples of the commands are:

```
START/006011 6017↑   ← Open symbolic location START and modify contents to 6017; open
+762    000000 ↑        previous location and do not modify; close the location and open the
+761    177400 ↑        previous location; close the location.
+760    177400 )

START/006011 6017↑   ← On the console, the example would appear as shown.
+762    000000 ↑
+761    177400 ↑
+760    177400
```

```
START/006017↓        ← Open location START; open succeeding location; open succeeding
START+1 001400 ↓        location; open succeeding location and close it without opening the next.
START+2 006073 )

START/006017         ← On the console, the example would appear as shown.
START+1   001400
START+2   006073
```

MONITORING MEMORY (Continued)

```
1000/.RDL 0 .RDL 1 )   ← Open location 1000 and change the contents to .RDL 1.
                         Close the location.

1000! .RDL 0 )         ← Open location 1000 without printing the contents. Change the contents
                         contents back to .RDL 0.

1000/.RDL 0 .RDL 1     ← On the teletypewriter, the example will appear as shown.

1000!.RDL 0
```

```
 open        open    open
 START       6017    106415
   ↓          ↓       ↓
START/006017 /106415 /0400000 ↓
106416  000000 )

START/006017 /106415 /040000  ← On the console, the example would appear as shown.
106416  000000
```

MONITORING SPECIAL REGISTERS

Special registers are locations containing program status information. They can be opened, examined, modified, and closed in a way similar to memory locations. A special register is normally closed by pressing the RETURN key, although the / or ! convention can be used where appropriate.

Accumulators

The command that opens an accumulator for examination and possible modification is:

```
n$A
```

where: n is the number of the accumulator (0-3 in the NOVA; 0-7 in the ECLIPSE computer).

This command is used in all debugger versions. An example of the command is:

```
0$A  000000 1
```

All accumulators may be examined, but not modified, by the command:

```
$A
```

The debugger will perform a carriage return/line feed and print the number of each accumulator, followed by its contents.

This command is used in all debugger versions. An example of the command is:

```
$A
0 000000  1 000565  2 001337  3 043131
```

MONITORING SPECIAL REGISTERS (Continued)

Floating Point Accumulators

The four floating point accumulators may be examined, but not modified, by using the command:

```
$F
```

The debugger will perform a carriage return/line feed and print the number of each floating point accumulator, followed by its contents.

This command is used in RDOS DEBUG and IDEB. The register is not valid in SOS DEBUG where the $F command causes punching of paper tape leader or trailer. An example of the command is:

```
$F
0  006452  1  000417  2  000000  3  106000
```

Search Increment Register

The search increment register is used in performing memory searches. (See Chapter 4 for the search commands.) By default, locations in the range selected for the search by the user will be searched forward in memory with an increment of one location. By setting the search increment register, the user may adjust the increment to any number of locations between locations to be examined or can adjust the increment for a backward search of memory locations.

The search increment register is opened for examination and possible modification by the command:

```
$J
```

This command is used in all debugger versions. Examples of the command are:

```
$J  000001  -10          ← Open register and set for backward search of every 10(8) locations.
440<$S                   ← Search from location 440 down in memory
 00440 000766
 00430 101300
 00420 025400
   :
   :
 00000 006017
```

```
$J  100010  2            ← Open register and set to search forward every second location.
102<114$S                ← Search from location 102 to location 114.
 00102 177777
 00104 177777
 00106 000000
 00110 000005
 00112 000000
 00114 000232
```

Mask and Word Registers

The mask and word registers are used in performing memory searches. (See Chapter 6 for search commands.) Certain types of searches are for particular bit contents and these require use of the word register.

The word register is set by the user to represent the contents for which memory is being searched. As the search proceeds, the contents of each location are compared with the contents of the word register. If they match, the location and the contents are printed out. By default, the word register contains 0, which allows the printout of all memory locations.

The mask register is set by the user if he wishes to mask the contents of certain bit positions in each memory register during the search. If no masking is desired, the mask register is set to -1 (all ones). If certain bit positions are to be masked, those positions are set to zero and the unmasked positions are set to one.

The contents of the mask register are ANDed with the contents of the memory location and then compared with the contents of the word register. By default, the mask register is set to 0 (all bit positions masked). Thus, if a search required comparison of memory contents with the word register, which occurs if the word register is non-zero, the appropriate contents of the mask register must be set by the user.

The word register is opened by issuing the command:

```
$W
```

The mask register is opened by issuing the command:

```
$M
```

These commands are used in all debugger versions. An example of the commands is:

```
$W 000000 15000     ← Open word register and store 15000
$W 015000

$M 000000 177700    ← Open mask register and mask bits 0-9
```

Numbers Register

The numbers register determines whether the contents of registers will be printed out in octal or in decimal. By default, the numbers register is set to 0, which causes register contents to be printed out in octal. If the user sets the contents of the numbers register to non-zero, contents of memory and special registers will be printed out in decimal, i.e., as signed decimal numbers with a decimal point.

The numbers register is opened for examination and modification by the command:

```
$N
```

Numbers Register (Continued)

This command is used in all debugger versions. An example of the command is:

```
START/006017      ← Contents of START in octal
$N 000000 1       ← Change numbers register to non-zero
START/+3087.      ← Contents of START in decimal
$N  +1.0          ← Contents of numbers register in decimal
```

Symbol Table Pointer Register

The symbol table pointer register contains a pointer to the beginning of the User Status Table. In SOS loading and in RDOS background loading, this is location 402, labeled USTSS. In RDOS foreground loading, USTSS may be a different location. The symbol table pointer register is opened by issuing the command:

```
$Y
```

This command is used in all debugger versions. An example of the command is:

```
$Y 000402          ← Contents of symbol table pointer register

402/TMIN+63        ← Examination of part of contents of the UST starting at USTSS.
+403  TMIN+52         (RDOS DEBUG with minimum task scheduler.)
+404  TMIN+64
+405  TMIN
+406  DEBUG
+407  TMIN+64
+410  +0
+411  177777
```

Search/Punch Register

Bit 15 of the search/punch register specifies the output device to be used in printing memory search results:

Bit 15	
0	Print search results at the console.
1	print search results on the line printer.

The default setting of bit 15 is 0.

Bit 0 of the register specifies the output device to be used when punching portions of the user program as follows:

Bit 0	
0	Punch output to the teletypewriter punch.
1	Punch output to the high speed punch.

The default setting of bit 0 is 0.

Search/Punch Register (Continued)

The search/punch register is opened for examination and possible modification by issuing the following command:

```
$H
```

The register is not used in RDOS DEBUG. In IDEB only the search bit of the register is significant. In SOS DEBUG both the search and the punch bits are significant. An example of the command is:

```
$H  000000  1
```

← Open search/punch register and set bit 15 to print search output to the line printer.

Interrupt Register

The interrupt register determines whether or not interrupts are to be enabled. The register is used in all IDEB and SOS DEBUG versions.

By default, the register is set to 0, which enables interrupts. (Note that in IDEB under RDOS mapping, the interrupts necessary for mapping are never disabled though other interrupts are.) To disable interrupts, the register is set to -1 (all ones).

The interrupt register is opened for examination and possible modification by the command:

```
$I
```

Example:

```
$I 000000  -1
```

← Open interrupt register and disable interrupts.

Task Control Block Register

The task control block register contains the address of the task control block (TCB) of the currently executing task. The TCB contains status information for each task, needed by the task scheduler in multitasking.

The register is opened for examination and possible modification by the command:

```
$T
```

The task control block register is used only for RDOS DEBUG. The command is not valid for IDEB or SOS DEBUG, where $T invokes the console input done register (see next page).

```
$T 000740
```

← Open TCB register and examine contents.

Console Input (TTI) Done Register

The console input done register specifies the status of console input in bit 0 as follows:

Bit 0	
0	Console input done
1	Console input not done

The register is opened for examination and possible modification by the command:

```
$T
```

The register is used in IDEB and SOS DEBUG versions. The command is not valid in RDOS DEBUG where $T invokes the task control block register (see previous page). Coding such as the following example would cause a transfer to the debugger while console input was still outstanding:

```
A: SKPDN TTI
      JMP .-1
      DIAC 0 TTI
                        ← Breakpoint set at this instruction

$T 100000               ← Bit 0 of the register is set to 1
```

Carry and Console Output Register

The Carry and console output done register specifies the status of the Carry flag in bit 0 as follows:

Bit 0	
0	Carry flag is 0.
1	Carry flag is 1.

Bit 15 of the register indicates the status of output to the console as follows:

Bit 15	
0	Console output is not done.
1	Console output is done.

The default setting of bit 15 is 0.

The Carry bit of the register is used in all debugger versions. The console output done bit is significant only in IDEB and in SOS DEBUG.

The register is opened for examination and possible modification by the command:

```
$C
```

Coding such as the following would cause a transfer to the debugger while console output was still outstanding and Carry was set.

Carry and Console Output Register (Continued)

```
B: ADCO 0 0
DOAS 0 TTO
SKPDN TTO
JMP .-1
   --              ← Breakpoint set at this instruction.


$C 100001          ← Bits 0 and 15 are set to one.
```

Starting Location Register

In SOS DEBUG, the starting location register is set by the user to contain the starting address to be used for execution when the command $R is issued (see Chapter 3). By default, the contents of the starting location register under SOS DEBUG is the contents of USTSA.

When debugging under RDOS, the starting location register will contain the contents of USTSA, address of the task scheduler, allowing control to be transferred to the scheduler to run the highest priority task.

The starting location register may be opened for examination and possible modification by the command:

```
$L
```

Example:

```
$L 000764          ← Determine address of task scheduler (RDOS DEBUG or IDEB)

$L 000000 4000     ← Set starting address for $R execution to 4000 (SOS DEBUG)
```

Extended Save Address Register

The extended save address register contains the contents of User Status Table location 421, USTSV. USTSV points to the extended save state routine.

The extended save address register is opened for examination and possible modification by the following command:

```
$E
```

The register is used in RDOS DEBUG and in IDEB. The command is not valid in SOS DEBUG where the command causes punching of an end block (see Chapter 7). An example of the command is:

```
$E 002135
```

****

CHAPTER 3

BREAKPOINTS AND PROGRAM RESTARTS

Breakpoints are key elements in debugging. Breakpoints permit the user to execute a small portion of his program and then check program status.

The user sets one or more breakpoints in his program using a debugger command. He can then indicate through a command when a breakpoint should cause program execution to cease and a transfer be made 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 (10_8) 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 will wipe out the user program.

Breakpoints are assigned numeric values in reverse numeric order; for example, if the user sets four breakpoints in his program, the breakpoints will be numbered:

7
6
5
4

starting with breakpoint 7.

SETTING, EXAMINING AND DELETING BREAKPOINTS

Setting a Breakpoint

The format of the command that sets a breakpoint is:

```
adr$B
```

where: adr is the program address at which the breakpoint is set.

This command is used in all debugger versions. An example of the command is:

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

Breakpoints should not be set at the following 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.

Examining Breakpoint Locations

The user may wish to determine where breakpoints are currently set in his program. The command that will cause breakpoint numbers and the locations at which they are set to be printed is:

```
$B
```

SETTING, EXAMINING AND DELETING BREAKPOINTS (Continued)

Examining Breakpoint Locations (Continued)

Breakpoints are printed out in descending numerical order.

This command is used in all debugger versions. An example of the command is:

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

Deleting a Single Breakpoint

The format for the command to delete a single breakpoint is:

```
n$D
```

where: n is the number of a previously set breakpoint.

The command causes a specific breakpoint to be deleted; the remaining breakpoint numbers remain the same, e.g., if breakpoints set are 7, 6, and 5, the command:

```
6$D
```

deletes breakpoint 6 while numbers assigned to the remaining breakpoints remain unchanged.

This command is used in all debugger versions. An example of the command is:

```
7$D
```

Deleting All Breakpoints

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

```
$D
```

This command is used in all debugger versions.

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 1, which is the number of times the instruction at the breakpoint will be executed before the debugger is reentered. This means that when the user restarts execution at that particular breakpoint, the instruction at the breakpoint will be executed, but the debugger will be reentered the next time that breakpoint is encountered.

The user may open the break proceed counter for examination and possible modification. The command to open the break proceed counter is:

```
n$Q
```

where: n is the number of the previously set breakpoint.

BREAKPOINT COUNTERS (Continued)

The contents of the break proceed counter are modified as are any register's contents by typing the desired contents immediately following the printout of the current contents.

This command is used in all debugger versions. An example of the command is:

```
7$Q 000001 2
```

Change to two executions of the instruction at breakpoint 7 before reentering the debugger.

PROGRAM RESTART COMMANDS

Restarting the Program

The debugged program may be restarted by issuing the command:

```
$R
```

This command is used in all debugger versions and causes restart at the contents of $L. However, the contents of $L differs in the debugger versions. In SOS DEBUG $L is user-set to the start of execution. In RDOS DEBUG $L contains the contents of USTSA, the task scheduler address, and in IDEB $L contains the contents of USTSA, which is the program starting address. The program will execute, looping through any breakpoints the number of times indicated by the break proceed counters, until a breakpoint is encountered, provided that a breakpoint is set. An example is.

```
$R
7B START
0 006207  1 006162  2 000000  3 006162
```

Restarting the Program at a Given Location

The user may specify the location at which program execution is to resume by issuing the command:

```
adr$R
```

where: adr is an address within the program.

This command is used in all debugger versions. An example of the command is:

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

Restarting the Program at a Breakpoint

When a breakpoint causes transfer to the debugger, the user issues debugging commands and can then restart the program execution at the breakpoint by issuing the command:

```
$P
```

Restarting the Program at a Breakpoint (Continued)

The program will execute, looping through the breakpoint the number of times indicated by the break proceed counter. When the debugger is reentered, the breakpoint and the contents of the registers will be printed.

This command is used in all debugger versions. An example of the command is:

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

The user has the option to override the contents of the break proceed counter when restarting execution at a breakpoint. The format of the command is:

```
n$P
```

where: n is the number of times the breakpoint instruction is to be executed.

This command is used in all debugger versions. An example of the command is:

```
$R
7B START
0  006207  1  006162  2  000011  3  006162
5$P
6B START+10
0  001654  1  000000  2  000011  3  000000
```

****

CHAPTER 4

SEARCH COMMANDS

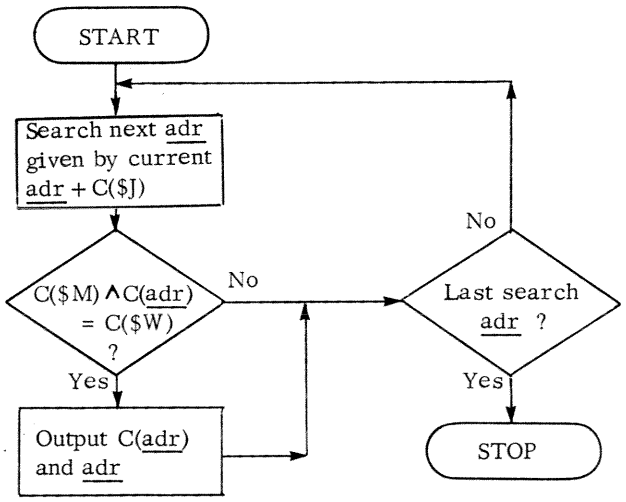
All or part of memory may be searched for a given bit configuration, and the matching addresses and their contents will be output at the console or on the line printer. To perform a search, the programmer takes the following action:

1. Opens the word register and places in it the desired configuration to be used in the search.
2. Opens the mask register and sets to ones those bit positions that will contain the configuration being searched for.
3. Opens the search increment register and sets the increment between locations to be examined.
4. Gives a search command, indicating whether all or part of memory is to be searched for the desired configuration.

The search algorithm is as follows:

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

The algorithm can be written as follows:



By default, the contents of both the word and mask registers are 0 and the search increment is +1. 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 search commands determine the range of memory to be searched. The command:

```
$S
```

causes all of memory to be searched.

SEARCH COMMANDS (Continued)

The command:

```
adr$S
```

causes a search from the start of memory through adr.

The command:

```
adr<$S
```

causes a search from adr through the end of memory.

The command:

```
adr1<adr2$S
```

causes a search from adr_1 through adr_2.

These search commands are used in all debuggers. Some examples of the commands are:

```
$W 000000 6017          ← Put .SYSTM configuration in $W
$M 000000 -1            ← Set $M to all ones.
START<START+40$S        ← Search range of 41 locations.
START 006017
START+10  006017
START+14  006017
START+20  006017
START+23  006017
START+27  006017
START+33  006017
```

```
$;                      ← Change to instruction format (see Chapter 6).

$W 000000 LDA 2         ← Put LDA 2 in $W (search on LDA 2 0 ).

$M 177777                 Check $M contents.
$S                      ← Search all memory.
4024  LDA 2 0
10362  LDA 2 0
22026  LDA 2 0
22041  LDA 2 0
36403  LDA 2 0
42335  LDA 2 0
```

In SOS DEBUG and in IDEB, a search can be terminated by striking any key. In RDOS DEBUG, there is no way to terminate a search; therefore, the user should avoid possible interminable searches whenever possible. IDEB and SOS DEBUG also allow search output directed to the line printer using the $H register.

****

CHAPTER 5

ENABLING AND DISABLING 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 may contain local symbols.

Local symbols are loaded for a given program only if requested by the user. This can be done during loading under either RDOS or SOS. In the RLDR command line, the local switch /U must follow the file name of the program for which user symbols are to be loaded. See Appendix B for loading procedures and an example of the /U switch.

During debugging, the user may wish to temporarily disable debugger recognition of some or all symbols and later re-enable recognition of the symbols.

DISABLING ALL GLOBAL AND LOCAL SYMBOLS

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

```
$K
```

This command is used in all debugger versions. An example of the command is:

```
START/006017        ← Debugger recognizes START
$K
START/U             ← 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, the user gives a command having the format:

```
n$K
```

where: n may be any single digit.

This command is used in all debugger versions. An example of the command is:

```
$K
START/U             ← Global symbol START is disabled.
0$K
START/006017        ← Global symbol START is re-enabled.
```

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

All global and local symbols, removed by either n$K or $K will be re-enabled by issuing a command of 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 and this command must be used.

This command is used in all debugger versions. An example of the command is:

```
RLDR/D ALPHA/U
DEB ALPHA
ALPHA%
```

REMOVING A SYMBOL FROM OUTPUT

A given symbol may be removed from debugger output by issuing a command of the format:

```
sym $K
```

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

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

When sym is removed by this command, the debugger will replace the sym on output by an offset from the nearest previous symbol, provided that the substitute symbol is not more than 2000_8 locations removed. If there is no previous symbol within 2000_8 locations or if no symbols remain, the absolute value of the location is substituted.

This command is used in all debugger versions.

****

CHAPTER 6

CHANGING OUTPUT FORMAT

Output may be formatted in any one of several ways, using the debugger. The contents of a location may be output as:

- An octal or decimal numeric datum.
- Two numeric half words.
- Two ASCII characters. Non-printing ASCII characters are printed in octal code equivalents enclosed in angle brackets.
- Symbolic format in which all enabled symbols are printed.
- Byte pointer format. The first 14 bits of the word are an address that contains or is to receive a byte and the last bit specifies that it is either the left or right byte (0-left; 1-right).
- .SYSTM command mnemonic format.
- A NOVA instruction.

To change from one type of output to another type of output format for all future printouts, the general format of the command is:

```
$x
```

where: x is one of the following characters:

= numeric format
; instruction format
: symbol format
' ASCII format
& byte pointer format
- half word format
? .SYSTM mnemonic format

This command is used in all debugger versions. Examples of the command are:

```
$=
START/006017
START+1   001400
START+2   006073
START+3   125120

$:
START/6017
START+1   DEBUG+41
START+2   6073
START+3   125120

$;
START/JSR @+17
START+1   JMP +0 3
START+2   JSR @+73
START+3   MOVZL 1 1
```

CHANGING OUTPUT FORMAT (Continued)

Examples of the ouput formats (continued)

```
$'
START/<14 ><17 >
START+1  <3 ><0 >
START+2  <14 >;
START+3  <252 >P

$?
START/ .WRB 17
START+1  .MEM 0
START+2  .WRB 73
START+3  125000  20

$←
START/14  17
START+1  3 0
START+2  14 73
START+3  252 120
```

To examine a given word in another format, the location is opened and the user types the command:

```
x
```

where: x is one of the seven output formats previously described or the following:

* symbolic format with bit 0 set to 0

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 may, of course, be transliterated, e.g.,

```
START+10=000773
```

In this way, any expression consisting of octal and decimal integers, symbols known to the debugger and the + and - operators may be converted to its equivalent instruction, byte pointer format, etc. Thus, the = command, for example, may be used for expression evaluation.

```
100+25.=000131 '<0 >Y:+131
100+25.;JMP +131 ← 0  131 $000054 1
100+25.?.CREA 31
```

****

CHAPTER 7

PUNCH COMMANDS

In SOS DEBUG only, the user has the option of using the debugger to punch all or part of his program. Either the high speed punch or the teletype punch may be selected for punch output by setting bit 0 of the $H register (see page 8).

The format of the command to punch all or part of a program in binary is:

```
adr1<adr2$P
```

where: adr$_1$ is the first address to be punched and adr$_2$ is the last address to be punched.

The command must include starting and terminating punch addresses and the symbol < to distinguish the command from the break proceed commands ($P and n$P).

Leader and trailer may also be punched. The commands to punch blank tape are:

```
$F
n$F
```

where: n gives the number of inches of blank tape to be punched and may be given in either octal or decimal.

If the $F command is not preceded by an argument giving the number of inches to be punched, by default 10_8 inches will be punched.

The user may also punch an end block terminating the punched program, using the commands:

```
$E
adr$E
```

where: adr provides the starting address for execution of the punched program. If adr is not given, the user must provide a starting address via the data switches at execution time.

The example following illustrates punching to the high speed punch

```
$H 100000          ← High speed punch in effect
$F                 ← Punch 10 octal inches leader.
LEM<LEM+100$P      ← Punch 101 locations starting at LEM.
LEM$E              ← Punch end block with address LEM.
10.$F              ← Punch 10 decimal inches trailer.
```

PUNCH COMMANDS (Continued)

When punched output is to the teletypewriter punch, the computer will halt to allow the user to stop and start the punch to prevent debugging commands from being punched as shown in the example following.

$H 000000	← Teletypewriter punch in effect.
6.$F	Punch 6_{10} inches leader. Computer HALTS with Carry light on. User presses ON button on teletypewriter and presses CONTINUE on operator panel.
	When punch stops, the computer HALTS with Carry light off. User presses OFF on teletypewriter punch and presses CONTINUE on operator panel.
.X<.X3$P	← Punch from .X to .X3. User presses ON on TTY and CONTINUE. When punching stops, user presses OFF on the TTY and CONTINUE.
.X$E	← Punch end block with starting address of .X. User presses ON on the TTY and CONTINUE. When punching stops, user presses OFF on the TTY and CONTINUE.
$F	← Punch 8_{10} inches trailer. User presses ON on the TTY and CONTINUE. When punching stops, user presses OFF on the TTY and CONTINUE.

****

CHAPTER 8

SAVING A DEBUGGED PROGRAM

In the Symbolic Debugger a means is provided to save the current status of a debugged program if desired. The $V command allows the programmer 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 causes a return to the CLI level with the current state of the debugged program in the BREAK file. The programmer at the CLI level can then take any appropriate action. If he wishes to save the BREAK file, he can issue a CLI SAVE command at this time.

In SOS DEBUG, issuing a $V command causes a jump to C(USTBR)-1. If the programmer is using the Core Image Loader Writer, this executes the Core Image Writer, which issues a # prompt, after which the programmer specifies the device and file to which the debugged program is to be written and then indicates the upper core address to be written (in accordance with Core Image Writer operation as specified in the SOS User's Manual, 093-000062). If the programmer is not using the CILW, the location contains a -1 by default and a return is made to the debugger; the programmer may provide a jump to his own location outside the debugger by changing the default contents of USTBR.

The format of the $V command is:

```
$V
```

Examples:

```
DEB ALPHA
  :          } ← Debugging commands
  :
$V
BREAK          ← Return to CLI (RDOS DEBUG and IDEB)
R

SAVE ALPHA     ← Programmer saves BREAK file under the name, ALPHA
R
```

```
DEB  CT1:5     ← Debug CT1:5
  :          } ← Debugging commands
  :
$V             ← $V break issued
#1:5           ← Old save file is specified for the new program
NMAX:  16200   ← NMAX is specified
OK             ← File has been rewritten to CT1:5
               ← CONTINUE key causes return to debugger
$R             ← Program is started at the beginning address.
```

Before issuing a $V command, the user should delete all breakpoints ($D). Otherwise, an attempt to debug the saved break file will result in a halt.

Note that the $V command causes all open files to be closed. To execute the break file after issuing a $V, the user must provide a routine to reopen all files that were open at the time the $V was issued.

****

APPENDIX A

COMMAND SUMMARY - SOS DEBUG

Command	Type of Command	Meaning
adr!	Monitor Core*	Open core location adr.
adr/		Open core location adr and print contents.
)		Close open location.
↓		Close open location and open the subsequent location.
↑		Close open location and open the previous location.
$A	Monitor Accumulators	Print contents of all accumulators.
n$A		Open accumulator n (n=0-3 in NOVA Computer; 0-7 in ECLIPSE Computer).
$C	Monitor Special Registers	Open the Carry/Teletypewriter Output register.
$H		Open the Search/Punch register.
$I		Open the Interrupt register.
$J		Open the Search Increment register.
$L		Open the Location register.
$M		Open the Mask register.
$N		Open the Number register.
$T		Open the Teletypewriter Input register.
$W		Open the Word Register.
$Y		Open the Symbol Table Pointer Register
$B	Breakpoint	Print location of all user program breakpoints.
adr$B		Insert breakpoint at location adr.
$D		Delete all breakpoints.
n$D		Delete breakpoint n (n=0-7).
$V	Save Debugged Program	Save status of debugged program and return to CILW for possible save of file.
$E	Punch	Punch an end block.
adr$E		Punch an end block, giving a starting address adr for execution.
$F		Punch ten inches of blank tape.
n$F		Punch n inches of blank tape.
adr_1<$adr_2$$P		Punch core from adr_1 through adr_2.
$P	Execute (also, Break Proceed Counter)	Restart execution from breakpoint with break proceed counter set to +1.
n$P		Restart execution from breakpoint with break proceed counter set to n.
n$Q		Open break proceed counter n (n=0-7).
$R		Restart execution at address in location counter.
adr$R		Restart execution at adr.
$K	Symbol Enable and Disable	Remove all local and global symbols from input/output.
n$K		Remove all local symbols from input/output but retain (or enable) globals.
sym$K		Remove symbol sym from output permanently.
name%		Enable all local and global symbols in program name (or restore to output all symbols removed by $K or n$K commands).
$S	Core Search*	Search all memory.
adr$S		Search memory from location 0 to adr.
adr_1<$S		Search memory from adr_1 to limit of memory.
adr_1<$adr_2$$S		Search memory from location adr_1 to adr_2 inclusive.
=	Format of Data Output to the Teletypewriter	Print last typed datum in numeric format.
:		Print last typed datum in symbolic format.
;		Print last typed datum in instruction format.
←		Print last typed datum in half-word format.
!		Print last typed datum in ASCII format.
&		Print last typed datum in byte pointer format.
?		Print last typed datum in .SYSTM format.
*		Print last typed datum in symbolic format with bit 0=0.
$=		Print subsequent data in numeric format.
$:		Print subsequent data in symbolic format.
$;		Print subsequent data in instruction format.
$←		Print subsequent data in half-word format.
$!		Print subsequent data in ASCII format.
$&		Print subsequent data in byte pointer format.
$?		Print subsequent data in .SYSTM format.

* Note: Under the mapping option, core examined must be within the user's address space.

APPENDIX A

COMMAND SUMMARY - RDOS DEBUG

Command	Type of Command	Meaning
adr! adr/) ↓ ↑	Monitor Core*	Open core location adr. Open core location adr and print contents. Close open location. Close open location and open subsequent location. Close open location and open the previous location.
$A n$A	Monitor Accumulators	Print contents of all accumulators. Open accumulator n (n=0-3 in NOVA Computer; 0-7 in ECLIPSE Computer).
$C $E $F $J $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry Register. Open the Extended Save Register (USTSV). Print contents of Floating Point Registers. Open the Search Increment Register. Open the Location Register. (USTSA). Open the Mask Register. Open the Number Register. Open the Task Control Block Register (USTCT). Open the Word Register. Open the Symbol Table Pointer Register.
$B adr$B $D n$D	Breakpoint	Print location of all user program breakpoints. Insert breakpoint at location adr. Delete all breakpoints. Delete breakpoint n. (n = 0-7).
$V	Break file	Put debugged program in break file for possible save of file.
$P n$P n$Q $R adr$R	Execute (and Break Proceed counter)	Restart execution from a breakpoint with break proceed counter set to +1. Restart execution from a breakpoint with break proceed counter set to n. Open break proceed counter n (n = 0-7). Restart execution at address stored in USTSA. Restart execution at adr.
$K n$K sym$K name%	Symbol Enable and Disable	Remove all local and global symbols from input and output. Remove all local symbols from input/output but retain (or enable) globals. Remove symbol sym from output permanently. Enable all local and global symbols in program name (or restore to output all symbols removed by $K or n$K commands).
$S adr$S adr_1 < $S adr_1 < adr_2 $S	Core Search*	Search all memory. Search memory from location 0 to adr. Search memory from location adr_1 to limit of memory. Search memory from location adr_1 to adr_2 inclusive.
= : ; ← ' & * ? $: $; $← $' $& $= $?	Format of Data Output to the Teletypewriter	Print last typed datum in numeric format. Print last typed datum in symbolic format. Print last typed datum in instruction format. Print last typed datum in half-word format. Print last typed datum in ASCII format. Print last typed datum in byte pointer format. Print last typed datum in symbolic format with bit 0=0. Print last typed datum in .SYSTM command format. Print subsequent data in symbolic format. Print subsequent data in instruction format. Print subsequent data in half-word format. Print subsequent data in ASCII format. Print subsequent data in byte pointer format. Print subsequent data in numeric format. Print subsequent data in .SYSTM command format.

*NOTE: Under the mapping option, core examined must be within the user's address space.

Command	Type of Command	Meaning
adr! adr/) ↓ ↑	Monitor Core*	Open core location adr. Open core location adr and print contents. Close open location. Close open location and open the subsequent location. Close open location and open the previous location.
$A n$A	Monitor Accumulators	Print contents of all accumulators. Open accumulator n (n = 0-3) in NOVA Computers; 0-7 in ECLIPSE Computers)
$C $E $F $H $I $J $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry/Teletype Output register. Open the Extended Save register (USTSV). Print contents of Floating Point registers. Open the Search/Punch register. Open the Interrupt register. Open the Search Interrupt register. Open the Location register (USTSA). Open the Mask register. Open the Number register. Open the Teletype Input register. Open the Word register. Open the Symbol Table Pointer register.
$B adr$B $D n$D	Breakpoint	Print location of all user program breakpoints. Insert breakpoint at location adr. Delete all breakpoints. Delete breakpoint n. (n = 0-7).
$V	Break file	Put debugged program in break file for possible save of file.
$P n$P n$Q $R adr$R	Execute (and Break Proceed Counter	Restart execution from a breakpoint with break proceed counter set to +1. Restart execution from a breakpoint with break proceed counter set to n. Open break proceed counter n (n = 0-7). Restart execution at address in USTSA. Restart execution at adr.
$K n$K sym$K name%	Symbol Enable and Disable	Remove all local and global symbols from input and output. Remove all local symbols from input/output but retain (or enable) globals. Remove sym from output permanently. Enable all local and global symbols in program name (or restore to output all symbols removed by $K or n$K commands).
$S adr$S adr < $S adr_1 < $adr_2$$S	Core Search*	Search all memory. Search memory from location 0 to adr. Search memory from location adr_1 to limit of memory. Search memory from location adr_1 to adr_2 inclusive.
= : ; ← ' & * ? $= $: $; $← $' $& $?	Format of Data Output to the Teletypewriter	Print last typed datum in numeric format. Print last typed datum in symbolic format. Print last typed datum in instruction format. Print last typed datum in half-word format. Print last typed datum in ASCII format. Print last typed datum in byte pointer format. Print last typed datum in symbolic format with bit 0 = 0. Print preceding datum in .SYSTM command format. Print subsequent data in numeric format. Print subsequent data in symbolic format. Print subsequent data in instruction format. Print subsequent data in half-word format. Print subsequent data in ASCII format. Print subsequent data in byte pointer format. Print subsequent data in .SYSTM command format.

*NOTE: Under the mapping option, core examined must be within the user's address space.

COMPARISON OF DEBUGGER COMMANDS

Command	SOS DEBUG	RDOS DEBUG	IDEB
$A	yes	yes	yes
n$A	yes	yes	yes
$B	yes	yes	yes
adr$B	yes	yes	yes
$C	yes (TTO and Carry)	yes (Carry only)	yes (TTO and Carry)
$D	yes	yes	yes
n$D	yes	yes	yes
$E	yes (used in punching)	yes (Extended Save register)	yes (Extended Save register)
adr$E	yes	no	no
$F	yes (used in punching)	yes (print floating point registers)	yes (print floating point registers)
$H	yes (Search and Punch)	no	yes (Search only)
$I	yes	no	yes
$J	yes	yes	yes
$K	yes	yes	yes
n$K	yes	yes	yes
sym $K	yes	yes	yes
$L	yes (address set by user)	yes (task scheduler-USTSA)	yes (task scheduler-USTSA)
$M	yes	yes	yes
$N	yes	yes	yes
$P	yes	yes	yes
n$P	yes	yes	yes
adr_1 < adr_2 $P	yes	no	no
n$Q	yes	yes	yes
$R	yes (restart at C (L))	yes (restart at USTSA)	yes (restart at USTSA)
adr$R	yes	yes	yes
$S	yes	yes	yes
adr$S	yes	yes	yes
adr_1 < $S	yes	yes	yes
adr_1 < adr_2 $S	yes	yes	yes
$T	yes (TTI register)	yes (TCB register)	yes(TTI register)
$V	yes	yes	yes
$W	yes	yes	yes
$Y	yes	yes	yes
adr/	yes	yes	yes
adr!	yes	yes	yes
)	yes	yes	yes
↓	yes	yes	yes
↑	yes	yes	yes
=	yes	yes	yes
:	yes	yes	yes
;	yes	yes	yes
←	yes	yes	yes
'	yes	yes	yes
&	yes	yes	yes
*	yes	yes	yes
?	yes	yes	yes
$:	yes	yes	yes
$;	yes	yes	yes
$←	yes	yes	yes
$'	yes	yes	yes
$&	yes	yes	yes
$=	yes	yes	yes
$?	yes	yes	yes
name%	yes	yes	yes

APPENDIX B

OPERATING PROCEDURES

LOADING THE SYMBOLIC DEBUGGER UNDER RDOS

Loading RDOS DEBUG

RDOS users receive one of the following library tapes, each of which contains the indicated version of RDOS DEBUG:

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

The global /D switch 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 the user wishes to control the area of core into which the debugger is to be loaded.

Examples:

```
RLDR/D A B C)                 ←Load RDOS DEBUG version from library.

RLDR/D A B BSYS. LB  C)       ←Load RDOS DEBUG version from library before loading
                                user program C.
```

Loading IDEB

RDOS users receive one of the following relocatable binary tapes, containing 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

To load an IDEB version of the debugger, the name of the debugger must appear in the command line. If the user also names the system library tape containing RDOS DEBUG specifically 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:

Examples:

```
RLDR/D  A  B  C  MIDEB)          ←Load an IDEB version following user programs.

RLDR/D  A  BIDEB  B  C)          ←Load an IDEB version following A but before user programs B and C.

RLDR/D  A  B  USYS. LB  C  IDEB)←Although IDEB appears in the command line, RDOS DEBUG
                                  will be loaded.
RLDR/D  A  B  AIDEB ASYS. LB C)  ←Although ASYS. LB appears in the command line, the IDEB
                                  version will be loaded.
```

Loading Local (User) Symbols for Use in Debugging

By default, no user symbols are loaded. However, the user can load user symbols contained in one or more of his programs by appending the local switch /U to the name of the program or programs in the RLDR command line.

Examples:

```
RLDR/D A/U B/U C/U )        ← Load local symbols for all user programs.

RLDR/D A/U B C MIDEB)       ← Load local symbols only for program A.
```

Invoking the Symbolic Debugger

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

```
RLDR/D A B/U C )            ← Load A, B, and C and RDOS DEBUG.

DEB A )                     ← Enter the debugger to debug A.SV.
```

When the DEB command is given, the debugger will respond with a carriage return/line feed, and the user may now proceed to issue the debugging commands described in this manual.

LOADING THE SYMBOLIC DEBUGGER UNDER SOS

SOS users receive one of the following relocatable binary tapes of SOS DEBUG:

Relocatable Binary	Title of Version	Associated Computer and System
SADEB.RB	SADEB	NOVA Unmapped SOS
SAMDEB.RB	SAMDEB	NOVA Mapped SOS
SABDEB	SABDEB	ECLIPSE Unmapped SOS
SAADEB.RB	SAADEB	ECLIPSE Mapped SOS

Loading SOS DEBUG under Magnetic Tape/Cassette Systems

To load SOS DEBUG under a SOS magnetic tape/cassette system, SOS DEBUG must first be transferred from paper tape to a file on magnetic tape or cassette. The relocatable loader must be loaded into core. The relocatable loader then issues the prompt:

```
RLDR
```

and the user responds by typing a command line of the format:

output-core-image SOS-debugger user-programs

For example:

```
RLDR MT1:0/S MT0:2 $PTR/U/3        ← Magnetic tape system.
```

In the example, MT0:2 is the SOS debugger and the relocatable binary user programs are on paper tape. The local switch /U is used to cause user symbols to be loaded for each of the programs. Once the core image file, MT1:0, has been created, the user loads MT1:0 using the Core Image Loader:

```
#1:0                       ← User requests loading of MT1:0 in response to CILW prompt #.
```

Loading SOS DEBUG under Magnetic Tape/Cassette Systems (Continued)

The user can then invoke the debugger under the CLI with the DEB command:

```
DEB MT1:0
```

The debugger gives a carriage return/line feed, and the user can then issue debugging commands as described in this manual.

Loading SOS DEBUG under Paper Tape Systems

Users with SOS paper tape systems receive SOS DEBUG as a relocatable binary tape. SOS DEBUG is loaded by the relocatable loader, which must first be loaded into core. Once the relocatable loader is in core, it will give the prompt SAFE=. The user/loader dialogue to load SOS DEBUG is given below. User responses are underlined.

SAFE=)	Carriage return gives standard save of 200 locations.
	Mount DEBUG in paper tape reader.
*2	Load DEBUG from PTR.
*4	Load all symbols.
	Mount user program in paper tape reader.
*2	Load user program from PTR.
.	.
.	.
.	.
	Mount last user program in paper tape reader.
*2	Load last user program from PTR.
*6	Print a loader map.
*8	Terminate load.

When loading is terminated, the user sets the data switches to the address of the debugger, contained in location 406 and presses START. The debugger responds with a carriage return/line feed, and the user may issue debugging commands, as described in this manual.

LOADING THE SYMBOLIC DEBUGGER FOR RTOS

RTOS users receive a version of IDEB, which is supplied as a file of the RTOS library. RTOS libraries are loaded under either RDOS or SOS.

Loading IDEB under RDOS

As indicated in Appendix B of the RTOS User's Manual, 093-000056, the RTOS libraries are loaded using the RLDR command to produce a save file of user binaries, drivers, RTOSGEN and the libraries:

```
RLDR/C/D user-binaries {user-drivers} RTOSGEN RTOS-libraries [{$TTO/L} {$LPT/L}] )
```

Loading IDEB under RDOS (Continued)

The /D global switch loads IDEB from the library. The load map may be obtained by listing either to the $TTO or $LPT.

Procedures for executing the loaded program are given in Appendix B of the RTOS User's Manual. If there is no starting address given after a .END in a user binary, the user can start the debugger at the contents of 406 or at the DEB address indicated in the loader map.

Loading IDEB under the SOS Magnetic Tape/Cassette System

The relocatable loader is loaded into core and issues the prompt RLDR. The user responds with a command line giving the files to be loaded and the save file to be created (indicated by the /S switch). The RTOS libraries, in which IDEB is contained, are loaded last. For example:

```
RLDR  CT0:9/S  CT1:0 ... CT0:4
```

When local symbols are required, the local switch /U should be appended to the user binary file in the load command.

Since IDEB is part of the library, one of the user binaries must declare the debugger as external, i.e.,

```
.EXTN  DEBUG
```

If the debugger has been extracted for loading as a separate .RB tape, it need not be declared as external.

The save file (indicated in the load line by the /S switch) must be created. Once the save file is created, the user loads it using the core image loader as described in Chapter 5 of the SOS User's Manual, 093-000062. When the OK prompt is given by the CILW, the user places 376 in the data switches, presses RESET and START. This causes RTOS to initialize the system. If no starting address is given after a .END, the user can start execution in the debugger by starting at the contents of 406 or at the DEB address indicated in the loader map.

Loading IDEB under the SOS Paper Tape System

The relocatable loader is loaded into core and issues the prompt SAFE= . The following loader responses and procedures will load RTOS from the $PTR. User responses are underlined.

SAFE=)	Carriage return gives standard save of 200 words.
	Mount RTOSGEN.
*2	Load RTOSGEN.
	Mount user drivers, if any.
*2	Load user drivers.
*4	Load all symbols.
	Mount user binaries.
*2	Load first user binary.
. . .	. . .
	Mount RTOS libraries.
*2	Load first library.
. . .	. . .
*6	Print a loader map.
*8	Terminate load.

Loading IDEB under the SOS Paper Tape System (Continued)

The order of loading is not critical, except that the RTOS libraries must be loaded last. If the debugger is in the RTOS library, a user relocatable binary must declare the debugger as an external, i.e.,

```
.EXTN DEBUG
```

so that the IDEB library program may be entered. If IDEB has been extracted as a relocatable binary to be loaded as a separate .RB file, it does not need to be declared as an external.

Placing 376 in the data switches, pressing RESET and START after loading will cause RTOS to initialize the system. If no starting address has been given after a .END, the user can start execution in the debugger by starting at the contents of 406 or at the DEB address indicated in the loader map.

INDEX OF COMMANDS

Data General

PROGRAMMING DOCUMENTATION
REMARKS FORM

Document Title	Document No.	Tape No.

SPECIFIC COMMENTS: List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

GENERAL COMMENTS: Also, suggestions for improvement of the Publication.

FROM:

Name	Title	Date

Company Name

Address (No. & Street)	City	State	Zip Code

Form No. 10-24-004

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Programming Documentation

FOLD UP SECOND FOLD UP

STAPLE