User's Manual

SYMBOLIC DEBUGGER

093-000044-03

Stand-alone DEBUG III	089-000073
RDOS DEBUG III (without mapping)	088-000021
RDOS DEBUG III (with mapping)	088-000053
RDOS IDEB (without mapping)	088-000041
RDOS IDEB (with mapping)	088-000069
RTOS IDEB	099-000060

Ordering No. 093-000044

Printed in the United States of America
Rev. 03, August 1973

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

This revision of the Symbolic Debugger User's Manual, 093-000044-03, supersedes 093-000044-02 and constitutes a major revision. The revision describes the following versions of the debugger: SOS DEBUG III, RDOS DEBUG III, and RDOS and RTOS IDEB. The Disk Operating System (DOS) version of the debugger has been eliminated.

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 from the teletype after a breakpoint occurs. The machine state can be monitored during execution using simple commands for the debugger from the teletypewriter.

The following versions of the symbolic debugger are available:

RDOS DEBUG III	Supplied in relocatable binary form as part of the RDOS library.
RDOS IDEB	Supplied in relocatable binary form as part of the RDOS library.
SOS DEBUG III	Supplied as a relocatable binary tape.
RTOS IDEB	Supplied in relocatable binary form as part of the RTOS library.

There are slight language differences between the versions and these are compared in the chart on page A-4. Users having an RDOS system receive two versions of the symbolic debugger. DEBUG III enables all interrupts during debugging; IDEB disables all interrupts. (If the user has an RDOS system with a Memory Management and Protection Unit, IDEB disables all interrupts except those that are essential for mapping purposes.)

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.

VERSIONS OF THE SYMBOLIC DEBUGGER

The versions of the symbolic debugger are:

SOS DEBUG III	used with the Stand-alone Operating System.
RDOS DEBUG III	used with the Real Time Disk Operating System. This version disables interrupts during debugging.
* RDOS IDEB	used with the Real Time Disk Operating System. This version disables interrupts during debugging.
RTOS IDEB	used with the Real Time Operating System.

Command language differences among the versions of the debugger are slight. In the descriptions of debugging commands, reference is made to the different versions of the debugger only when the command applies only to selected versions or is used in a slightly different manner in the different versions of the debugger. In addition, Appendix A lists the commands that may be used in each version and provides a chart comparing the language differences among the versions of the debugger.

Loading and operating procedures for the versions of the debugger are given in Appendix B.

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.

* Interrupts essential for mapping are not disabled by IDEB if the user version of RDOS has the mapping option; all other interrupts will be disabled.

SYMBOLIC DEBUGGING COMMANDS (Continued)

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.
↑	Pressing the SHIFT and N keys causes the symbol ↑ to be printed.
$	Pressing the ESC key causes the symbol $ to be printed. (The programmer can press SHIFT and 4 instead of ESC.
?	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.

-1$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 already 8 breakpoints in the program.

$I? - Attempt to open the interrupt register while in RDOS Debug III (the register is implemented in IDEB but not in Debug III under RDOS.)

INTERRUPTING DEBUGGING

All versions of the symbolic debugger inhibit interrupts except RDOS Debug III. RDOS Debug III operates under the single task monitor with interrupts always enabled. Therefore, the only way to interrupt debugging when using RDOS Debug III is to press CTRL and A. CTRL A aborts current program activity and returns to the next higher level (CLI).

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)
/ or !	Close the open location and open the location specified by the contents of that location.

Some examples of opening and closing locations are:

```
START/006011 6017↑
+762    000000↑
+761    177400↑
+760    177400)
```

Open symbolic location START and modify contents to 6017; open previous location and do not modify; close the location and open the previous location; close the location.

```
START/006011 6017↑
+762    000000↑
+761    177400↑
+760    177400
```

On the console, the example would appear as shown.

```
START/006017↓
START+1 001400↓
START+2 006073)
```

Open location START; open succeeding location; open succeeding location; open succeeding location and close it without opening the next.

```
START/006017
START+1    001400
START+2    006073
```

On the console, the example would appear as shown.

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
1000!.RDL 0
```
On the teletypewriter, the example will appear as shown.

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

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

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).

Example:

```
0$A 000000 1
```

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

```
$A
```

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

Example:

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

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
```

Example:

```
$W 000000 15000      - open word register and store 15000
$W 015000

$M 000000 000011     - 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)

Example:

```
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 issueing the command:

```
$Y
```

Example:

```
$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 version 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

The search bit of the register is significant in both SOS Debug III and in IDEB. The punch bit is significant only in SOS Debug III. RDOS Debug does not use the register.

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

```
$H
```

Example:

```
$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 interrupt register is used by SOS Debug III and IDEB; in RDOS Debug III interrupts are always enabled.

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 task control block register is used only by RDOS Debug III.

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

```
$T
```

Example:

```
$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 console input done register is not used by RDOS Debug III.

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

```
$T
```

Coding such as the following 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 console output done bit is used only in SOS Debug and in IDEB. The Carry bit is significant in all versions.

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 III, the starting location register is set by the user to contain a 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 is 0.

In RDOS Debug III, the starting location register contains the address of the task scheduler allowing control to be transferred to the scheduler to run the highest priority task. The register is not used under IDEB.

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 III)

$L 000000 4000     - set starting address for $R execution to 4000 (SOS Debug III)
```

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 numberic order, e.g., 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.

Example:

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

Breakpoints should not be set at the following types of locations:

1. Data words.
2. Instructions modified during execution.
3. Locations where interrupts cannot be delayed for relatively long times.

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 out is:

```
$B
```

SETTING, EXAMINING AND DELETING BREAKPOINTS (Continued)

Examining Breakpoint Locations (Continued)

Breakpoints are printed out in descending numerical order.

Example:

```
$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.

Example:

```
7$D
```

Deleting All Breakpoints

All breakpoints in a user 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 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 register is:

```
n$Q
```

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

BREAKPOINT COUNTERS (Continued)

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

Example:

```
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
```

Restart is at the user-set contents of $L in SOS Debug III, at USTSA (task scheduler address) in RDOS Debug III, and at USTSA (program starting address) in IDEB. 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 has been set.

```
$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.

```
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
```

PROGRAM RESTART COMMANDS (Continued)

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.

```
$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.

```
$R

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

Restart with Debugger Return under SOS

If no breakpoints are set, use of the $R or adr$R commands will cause the program to execute without return being made to the symbolic debugger at termination of execution. In SOS Debug III, two alternative commands are provided that allow the user to return via C(AC3).

The command:

```
$G
```

is identical to $R, except that AC3 contains the address of the debugger at restart time. A return is made to the debugger if an instruction points to C(AC3)

The command:

```
adr$G
```

is identical to adr$R except that like $G, AC3 contains the address of the debugger at restart time.

These commands are implemented only for stand alone Debug III.

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.

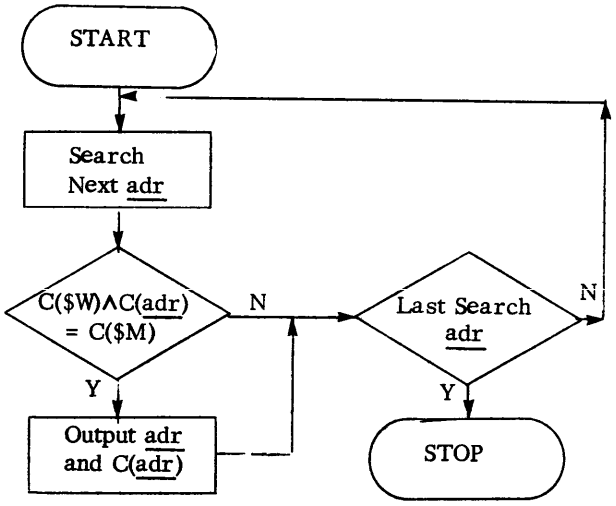
To perform a search, the programmer must:

1. Open the word register and place in it the desired configuration to be used in the search.
2. Open the mask register and set to ones those bit positions that will contain the configuration being searched for.
3. Give 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 comparision of 2. results in a match, the address and its contents are output.

The algorithm can be written as follows:



By default, the contents of both the word and mask registers are 0. To output all locations in memory or in a given range, the word and mask registers should be 0.

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.

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.

Examples:

```
$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 III and in IDEB, a search can be terminated by striking any key. In RDOS Debug III, however, the only means of terminating a search is to press CTRL A. Since CTRL A aborts further program activity, the user should avoid possible interminable searches whenever possible. IDEB and SOS Debug III 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 RDOS or SOS in the RLDR command line. The switch /U is attached to the file name of a program. When using the debugger in stand-alone mode but without the Stand-alone Operating System, local symbols are loaded by a response of

4

to the loader prompt *.

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 DISABLING LOCAL SYMBOLS

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

```
$K
```

Example:

```
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.

Example:

```
$K
START/U             - global symbol START is disabled.
0$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.

Example:

```
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.

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. (This format is not used in the SOS debugger.)
- 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

Examples of the output formats 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 format characters just described.

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
```

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

Under SOS Debug III, 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.
```

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 RDOS debuggers, a means is provided to save the current status of a debugged program if desired. The $V command allows the programmer to return from the debugger (either Debug III or IDEB) to the CLI level with the current state of the debugged program in the BREAK file, which includes the current status of all registers and patches made during debugging.

The programmer at the CLI level may take any action then appropriate. If he wishes to save the BREAK file, he can issue a CLI SAVE command at this time, which will save the file under the name given.

The format of the $V command is:

```
$V
```

Example:

```
DEB ALPHA
  :          }   debugging commands
  :          }
$V                return to CLI
BREAK
R
R

SAVE ALPHA        programmer saves BREAK file under the name, ALPHA
R
```

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 SYMBOLIC DEBUGGER

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).
$C $H $I $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry/Teletype Output register. Open the Search/Punch register. Open the Interrupt register. Open the Location register. 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).
$E adr$E $F n$F adr_1<$adr_2$$P	Punch	Punch an end block. Punch an end block, giving a starting address adr for execution. Punch ten inches of blank tape. Punch n inches of blank tape. Punch core from location adr_1 to location adr_2.
$G adr$G $P n$P n$Q $R adr$R	Execute (and Break Proceed Counter)	Restart program execution at address in location counter; store debugger address in AC3. Restart program execution at adr; store debugger address in AC3. Restart execution from a breakpoint with break proceed counter set to +1. Restart execution from a breakpoint with break proceed counter set to n. Open break proceed counter n (n = 0-7). Restart execution at address in location counter. Restart execution at adr.
$K n$K sym$K name%	Symbol Enable and Disable	Remove all local and global symbols from input and output. Remove all local symbols from input/output but retain (or enable) globals. Remove symbol sym from from output permanently. Enable all local and global symbols in program name (or restore to output all symbols removed by $K or n$K commands).
$S adr$S adr_1<$S adr_1<$adr_2$$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 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.

APPENDIX A

COMMAND SUMMARY - RDOS SYMBOLIC DEBUGGER, DEBUG III

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 acumulators. Open accumulator n (n=0-3).
$C $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry Register. Open the Location Register. Open the Mask Register. Open the Number Register. Open the Task Control Block 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 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 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 last typed datum in .SYSTM command format. Print subsequent data in .SYSTM command format.

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

APPENDIX A

COMMAND SUMMARY - RDOS/RTOS SYMBOLIC DEBUGGER, IDEB

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).
$C $H $I $M $N $T $W $Y	Monitor Special Registers	Open the Carry/Teletype Output register. Open the Search/Punch register. Open the Interrupt register. 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_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 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 preceding datum in .SYSTM command format. Print subsequent data in .SYSTM command format.

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

APPENDIX A

COMPARISON OF DEBUGGER COMMANDS

Command	SOS Debug	RDOS Debug	RDOS/RTOS 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
$E	yes	no	no
adr$E	yes	no	no
$F	yes	no	no
n$F	yes	no	no
$G	yes	no	no
adr$G	yes	no	no
$H	yes (Search and Punch)	no	yes (Search only)
$I	yes	no	yes
$K	yes	yes	yes
n$K	yes	yes	yes
sym$K	yes	yes	yes
$L	yes (address set by user)	yes (address of task scheduler)	no
$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	no	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
?	no	yes	yes
$?	no	yes	yes
name%	yes	yes	yes

APPENDIX B

OPERATING PROCEDURES

LOADING RDOS DEBUG III AND IDEB

RDOS Debug III is contained in the library tape, SYS.LB and is loaded with the user program when the user gives the global /D switch for the RLDR command, e.g.,

```
RLDR/D ALPHA )
```

This command loads ALPHA followed by DEBUG and appends the symbol table to the save file ALPHA.SV immediately after the last DEBUG location which was loaded.

To load IDEB in place of Debug III, IDEB must appear in the command line, e.g.,

```
RLDR/D ALPHA IDEB )
```

In this case, loading proceeds in the same way except that IDEB is loaded in place of Debug III.

In the examples shown above, the debugger is in each case loaded after the user program. However, the user may control when the debugger is loaded:

```
RLDR/D A B SYS.LB C D )
RLDR/D A B IDEB C D )
```

In the first example, Debug III is loaded following B and before C is loaded; in the second example IDEB is loaded in that same position.

Should both SYS.LB and IDEB appear in the same RLDR/D command line, the first debugger appearing in the line will be the one loaded, e.g.,

```
RLDR/D A B SYS.LB IDEB C )  - Debug III is loaded.
```

If the user wishes to load local symbols for one or more of his relocatable binary programs, he must append the local switch /U to those file names in the command line.

```
RLDR/D A B/U )
```

In the example, only local symbols in B will be loaded.

Once the appropriate debugger has been loaded, it is invoked by giving the command DEB followed by the name of the saved file to be debugged. (The first relocatable binary file name appearing in the RLDR command line is, by default, the name assigned the saved file.)

```
RLDR/D A B C )
DEB A )
```

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

LOADING SOS DEBUG III (PAPER TAPE)

SOS Debug III may be loaded in stand-alone from paper tape using the extended relocatable binary loader 091-000038. When the extended relocatable binary loader is in core, the following loader responses and procedures will load Debug III; user responses to the loader are underlined.

SAFE =) Carriage return gives standard save of 200 locations.

Mount Debug III in paper tape reader

*2 Load Debug III 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 (contents of location 406) and presses START. The debugger responds with a carriage return/line feed and the user may issue debugging commands.

LOADING SOS DEBUG III (MAGNETIC TAPE/CASSETTE)

If the user has a SOS magnetic tape/cassette system, he can use the extended relocatable loader 089-000120 to load the debugger. When the extended relocatable loader is in core, it issues the prompt, RLDR. The user responds with a command line giving the files to be loaded. The debugger must be contained in one of the files.

The local switch /U must be appended to the names of all files whose local user symbols are to be loaded. A command line might appear as:

```
(RLDR) MT1:0/S  MT0:1  $PTR/U/3 )
```

where MT0:1 is the debugger and the user programs are on paper tape.

Once the core image file has been created on MT1:0, the user can load MT1:0 using the core image loader as described in Chapter 5 of the SOS User's Manual.

If none of the previously loaded programs contains a starting address, the core image loader will halt once MT1:0 is transferred to core and will issue the message:

OK

LOADING SOS DEBUG III (MAGNETIC TAPE/CASSETTE) (Continued)

The user can then start the debugger at the contents of 406 or at the DEB address indicated in the loader map. (For normal program running, rather than the debugger, the user would start at location 377, which is a JMP @405 instruction where 405 = the starting address of the program.)

LOADING RTOS IDEB UNDER RDOS

RTOS IDEB is supplied as a file of the RTOS library tape 099-000060. 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)]
```

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

Procedures for executing the loaded program (either under RDOS or SOS) 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 RTOS IDEB UNDER SOS (PAPER TAPE)

RTOS IDEB may be loaded using the extended relocatable binary loader 091-000038. When the loader is in core, 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 RTOS IDEB UNDER SOS (PAPER TAPE) (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 and is 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 the 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.

LOADING RTOS IDEB UNDER SOS (MAGNETIC TAPE/CASSETTE)

For a magnetic tape/cassette system, extended relocatable loader 089-000120 can be used to load the debugger and system. When the loader is in core, it 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) MT0:9/S  MT1:0 ... MT0:4 )
```

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

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

```
.EXTN DEBUG
```

If IDEB has been extracted as a separate .RB file for loading, it is not necessary to declare it 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, 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.

RDOS/RTOS SYMBOLIC DEBUGGER, IDEB, COMMANDS

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).
$C $H $I $M $N $T $W $Y	Monitor Special Registers	Open the Carry/Teletype Output register. Open the Search/Punch register. Open the Interrupt register. 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_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 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 preceding datum in .SYSTM command format. Print subsequent data in .SYSTM command format.

RDOS SYMBOLIC DEBUGGER COMMANDS

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)
$C $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry Register. Open the Location Register. Open the Mask Register. Open the Number Register. Open the Task Control Block 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 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 all 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 Teletype-writer	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 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 last typed datum in . SYSTM command format. Print subsequent data in . SYSTM command format.

SOS SYMBOLIC DEBUGGER COMMANDS

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).
$C $H $I $L $M $N $T $W $Y	Monitor Special Registers	Open the Carry/Teletype Output register. Open the Search/Punch register. Open the Interrrupt register. Open the Location register. 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).
$E adr$E $F n$F adr_1 < adr_2 $P	Punch	Punch an end block. Punch an end block, giving a starting address adr for execution. Punch ten inches of blank tape. Punch n inches of blank tape. Punch core from location adr_1 to location adr_2.
$G adr$G $P n$P n$Q $R adr$R	Execute (and Break Proceed Counter)	Restart program execution at address in location counter; store debugger address in AC3. Restart program execution at adr; store debugger address in AC3. Restart execution from a breakpoint with break proceed counter set to +1. Restart execution from a break point with break proceed counter set to n. Open break proceed counter n (n = 0-7). Restart execution at address in location counter. 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 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 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.