DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100

PROGRAM

Debug III User' s Manual

TAPES

Relocatable Binary: 089-000073

DOS Dump: 088-000002 (SYS.LB)

ABSTRACT

Debug III is a routine used for symbolic debugging of user programs.

093-000044-02

TABLE OF CONTENTS

INTRODUCTION 1

COMMAND SUMMARY 2

SYMBOLS AND CONVENTIONS 5

USER TYPING ERRORS 6

OPENING, MODIFYING AND CLOSING MEMORY REGISTERS 7

REMOVING AND RESTORING USER SYMBOLS 9

STARTING AND RESTARTING A PROGRAM 10

SETTING AND DELETING BREAKPOINTS 11

ENTERING AND LEAVING THE DEBUGGER VIA BREAKPOINTS 12

SEARCH COMMANDS 14

EXAMINING AND SETTING SPECIAL REGISTERS 16

EXAMINING AND SETTING SPECIAL REGISTERS $T AND $C 18

PUNCH COMMANDS 19

CONVERSION MODE COMMANDS 21

ERROR RESPONSES 23

OPERATION 24
- Symbol Table 24
- Loading Stand-alone Debug III 25
- Using Stand-alone Debug III 26
- Loading Debug III under DOS 27

INTRODUCTION

The NOVA symbolic debugger, Debug III, is a program that interfaces with user routines as an aid in debugging. Debug III provides for up to 8 active breakpoints within the user's routines. The accumulators, carry, and memory can be examined and modified from the teletype after a breakpoint has occurred. The machine state can be monitored during execution of a routine using simple commands to the debugger from the teletype. The debugger interfaces with any NOVA routine, including those using the NOVA interrupt structure. The Stand-alone version of Debug III can also be used to punch ranges of memory in binary format acceptable as input to the Binary Loader.

The following versions of Debug III are available:

1. Relocatable - No Disk Operating System (Stand-alone).

2. Relocatable - Disk Operating System (DOS).

The following differences exist between the two debuggers:

1. Punch commands and certain register commands are meaningless in DOS operation and are therefore illegal. The commands involved are:

$F n$F $E adr$E adr$_1$ < adr$_2$$P $H	punch commands
$I	Interrupt register command
$T	TTI register command
$L	Location register command
$Y	Symbol table pointer register command

2. Under DOS, all I/O is handled by the system. Therefore, the DOS debugger does not recognize I/O mnemonics (DIA, NIOS, DOAS, etc.). However, programmed system commands for I/O are recognized (.OPEN, .RDL, .CLOSE, .WRS, etc.).

3. The $R command has a slightly different meaning in Stand-alone and DOS Debug III. The difference is noted in the writeup of that command. The $G and adr $G commands are only implemented in Stand-alone.

COMMAND SUMMARY

The Symbolic Debugger provides extensive facilities for examining and modifying program status. The single-character command code may be preceded by an argument. The argument is normally separated from the command code by the symbol $ or the escape code (ESC) which echoes $. In the command summary following, the argument may be null or one of the following:

sym	-	user symbol
adr	-	an address having any legal address format, such as an octal or decimal number, user symbol, or expression. An expression has the format: x±x±x ... where each x is a symbol, octal, or decimal number separated from the following x by either + or - .
n	-	an integer.
$adr_1 < adr_2$	-	a range of addresses from adr_1 to adr_2 inclusive.
name	-	a user symbol that is the name of a user program.

Command	Meaning	Page Where Described
adr/	Open register adr and type contents.	7
adr !	Open register adr.	7
↙	Close open register.	7
↓	Close open register and open next.	7
↑	Close open register and open previous.	7
$K	Remove all local and global symbols from input and output.	9
n$K	Remove all local symbols from input and output but retain (or enable) globals.	9
sym $K	Remove sym from output permanently.	9
name %	Enable all local and global symbols or restore to output all symbols removed by $K and n$K commands within program name.	9

COMMAND SUMMARY (Continued)

Command	Meaning	Page Where Described
$P	Proceed from current breakpoint with break proceed counter set at +1.	12
n$P	Proceed from current breakpoint with break proceed counter set at n.	12
n$Q	Open break proceed counter n (n = 0-7).	12
$B	Print locations of all user program breakpoints.	11
adr$B	Insert breakpoint at location adr.	11
$D	Delete all breakpoints.	11
n$D	Delete breakpoint n. (n = 0-7)	11
$M	Open mask register.	14
$W	Open word register.	14
$S	Search all memory.	14
adr$S	Search memory locations 0 to adr.	14
adr < $S	Search memory locations adr_1 to 077777 inclusive.	14
adr_1 < $adr_2$$S	Search memory locations adr_1 to adr_2 inclusive.	14
$R	Restart program at address in location register (Stand-alone). Restart program at starting address in UST (USTSA) (DOS).	10 10
adr$R	Restart program at address adr.	10
$G	Restart program at address in location register and set C(AC3) to address of debugger. *	10
adr$G	Restart program at address adr; set C(AC3) to address of debugger. *	10
$A	Print all accumulator register contents.	16
n$A	Open accumulator register n. (n = 0-7)	16
$I	Open interrupt register. *	16
$L	Open location register. *	16
$N	Open numbers register.	16
$Y	Open symbol table pointer register. *	16
$T	Open TTI done register. *	18
$C	Open Carry and TTO done register.	18

*Only implemented in Stand-alone Debug III.

Command	Meaning	Page Where Described
$F	Punch 10 decimal inches blank tape. *	19
n$F	Punch n (octal or decimal) inches blank tape. *	19
$E	Punch end block on tape. *	19
adr$E	Punch end block with transfer point adr. *	19
adr_1<adr_2P	Punch binary tape from location adr_1 to adr_2 inclusive. *	19
$H	Open search/punch output register. *	19
=	Print last typed quantity in numeric form.	21
:	Print last typed quantity in symbolic form.	21
;	Print last typed quantity in instruction form.	21
←	Print last typed quantity in half-word form.	21
'	Print last typed quantity in ASCII form.	21
&	Print last typed quantity in byte-pointer form.	21
$=	Print subsequent quantities in numeric form.	21
$:	Print subsequent quantities in user symbol form.	21
$;	Print subsequent quantities in instruction form.	21
$←	Print subsequent quantities in half-word form.	21
$'	Print subsequent quantities in ASCII form.	21
$&	Print subsequent quantities in byte-pointer form.	21

* Only implemented in Stand-alone Debug III.

SYMBOLS AND CONVENTIONS

SYMBOL	MEANING	EXAMPLE
+	Addition.	NEXT+SUM+1
-	Subtraction.	SUM2-SUM1
(Space)	Separate instruction fields.	LDA 0 0 0
,	Separate instruction fields.	LDA 0,0,0,
#	Add 10 to data (used with source-destination accumulator instruction.)	ADC# 0 1 SZC= 106032 command response
@	Add 100000 to data, and add 2000 to an instruction.	@= 100000 LDA 0 @0 0= 022000 command response
"	Delimit beginning and end of a one character string; delimit beginning and, optionally, the end of a two-character string.	"A" "AB" "AB
.	Can be used as any character within a user symbol.	.PR2
.	Denote a decimal number.	-5.
.	Can be used to represent the current location. In the example, AA is the current location symbol and the commands AA/ and ./ produce identical results.	AA/NEG 0 0 ./ NEG 0 0

SYMBOLS AND CONVENTIONS (Continued)

SYMBOL	MEANING	EXAMPLE
↵	Conventional representation for pressing carriage RETURN key. This closes a memory register after examination and possible modfication.	AA/NEG 0 0 NEG 1 1↵
↓	Conventional representation for pressing LINE FEED key. This closes a memory register after examination and possible modification and opens the next consecutive register.	AA/NEG 1 1↓ AA+1/LDA 0 FD21↓ AA+2/STA 1 FD10
$	Conventional representation for the ESC key (although the $ symbol may also be typed by pressing the Shift and 4 keys). $ delimits the argument from the command. (ABCB is ambiguous, ABC$B is not.)	$S $L
↑	Symbol is used to close a memory register after examination and possible modification and open the previous register.	AA+2/STA 1 FD10↑ AA+1/LDA 0 FD21

USER TYPING ERRORS

The user can kill an incorrect command or typing error by pressing RUBOUT key.

OPENING, MODIFYING, AND CLOSING MEMORY REGISTERS

FORMAT	MEANING
adr!	Open register adr.
adr/	Open register adr and print contents.
↵	Close most recently opened register.
↓	Close most recently opened register and open the succeeding register.
↑	Close most recently opened register and open the previous register.
NOTES:	When a register is opened and a command is given that does not reference the register, the register is automatically closed. If $ is typed with a register open, the error message ? will be printed. / or ! remain in effect until changed when opening and closing registers repetitively using ↑ and ↓.

EXAMPLES:

```
ABC!                                  ←  Open register ABC.

ABC/ADD 0 1                           ←  Register ABC is opened and contents printed.

AA/ADD 0 0 SZC   SUB 0, 0↵            ←  Register AA is opened, modified and closed.

AA/ADD 0 0 SZC   SUB 0 0 ↓               Register AA is opened, modified and closed, and
AA+1 LDA 3 -1 3                          the next location is opened and its contents
                                         printed. (/ remains in effect during chained
                                         openings.)

AA/ADD 0 0 SZC   SUB 0, 0↑            ←  Register AA is opened, modified and closed, and
A STA 0 TR                               the previous location,A, is opened and its contents
                                         printed.

AA/ADD 0 0 SZC↓
AA+1 LDA 3 -1 3↓                      ←  Series of openings and closing of sequential registers.
AA+2 JMP BB↑
AA+1 LDA 3 -1 3↵
```

EXAMPLES (Continued)

AA/JMP BB/ADD 0 0	The second / closes register AA unmodified and opens a register via the 15-bit C(AA).
AA/JMP BB = 000423	Register AA is still open.
AA/JMP BB .= 003000	Register is closed by . = . The value of address AA is 3000.

REMOVING AND RESTORING USER SYMBOLS

The symbol tables of the assembled programs loaded with the debugger contain the user symbols known to the debugger. These are the local symbols - those known only in a single assembled program - and the global symbols known throughout the loaded programs.

FORMAT	MEANING
$K	Remove all symbols (local and global) from input and output. Absolute values are used instead.
n$K	Remove all local symbols from input and output but retain or enable global symbols. Absolute values are used instead of local symbols. n is any single digit.
sym$K	Remove the user symbol named sym permanently from output. The user symbol having a value closest to sym is used instead, provided there is a user symbol within 2000_8.
name%	Restore to output all user symbols previously removed from the program named name by n$K and $K commands, i.e., enable all local symbols except any removed by a sym $K command.
NOTE:	Symbols are removed from output by the sym $K command but may still be used on input. If no local symbols have been loaded, Ø$K will enable all globals.

EXAMPLES: Suppose that a program given in the title XX by the .TITL pseudo-op contains symbols C72, .FD40 and T2. Then:

```
505/LDA 1 C72
C72$K
505/LDA 1 .FD40+7
.FD40$K
./LDA 1 T2+23
1$K
./LDA 1 +50
XX%
505/LDA 1 T2+23
```

← In this example, each time a symbol is removed, the debugger substitutes the closest symbol with appropriate offset. When all local symbols are removed by a 1$K command, an absolute value is substituted. The command XX% restores all symbols not permanently removed from output.

STARTING AND RESTARTING A PROGRAM

Four commands are available for starting and restarting a user program at a location other than a breakpoint.

Two of the commands simply give a starting location. The other two commands provide that AC3 will contain the address of the debugger at restart time, so that a return is made to the debugger if an instruction points to C(AC3).

FORMAT	MEANING
$R	Restart program at address given in location register, C($L). (Stand-alone Debug III). Restart program at program's starting address in User Status Table (USTSA, location 405). (Debug III under DOS).
adr$R	Restart program at address given by adr.
$G	Restart program at C($L); set C(AC3) to address of debugger (Stand-alone Debug III).
adr$G	Restart program at address given by adr; set C(AC3) to address of debugger. (Stand-alone Debug III).

EXAMPLES:

```
$L 000261          ← contents of location counter checked and user program
$R                   restarted at that point if using Stand-alone Debug III.

7B BQ              ← after a break, user restarts his program at a location TOP.
TOP$R

USE4$G             ← user restarts program at a different location and sets
                     the debugger location in AC3.
```

SETTING AND DELETING BREAKPOINTS

The user can set up to eight breakpoints in his program. When a breakpoint is encountered during execution, the breakpoint causes a transfer to the debugger before the instruction at which it is set is executed. In effect, the setting of the breakpoint causes the program instruction to be transferred to the debugger and a JMP instruction to the debugger to be substituted in the user program.

Eight (10_8) locations of 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. Breakpoint numbers are assigned in reverse numeric order: 7 6 5 ... 0.

FORMAT	MEANING
$B	Print locations of all breakpoints.
adr$B	Set a breakpoint at location adr.
$D	Delete all breakpoints.
n$D	Delete breakpoint n where n = 7 6 ... 0.

NOTES: See page 11 to resume execution after a break.

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.

EXAMPLES:

```
$B              ← command to print out existing breakpoints.
7B TT           ← response
6B TT2
TT4$B           ← command to set a new breakpoint
$B
7B TT
6B TT2
5B TT4
6$D             ← command to delete breakpoint 6.
$B
7B TT
5B TT4
```

ENTERING AND LEAVING THE DEBUGGER VIA BREAKPOINTS

A user can set a breakpoint at a given instruction in his program, as described in "Setting and Deleting Breakpoints". Breaks are not visible to the user unless the STOP and EXAMINE switches on the operator's panel are used. During program execution a transfer is made to the debugger when the breakpoint is encountered. The instruction at which the breakpoint is set is not executed. The debugger prints the breakpoint number, the instruction address, and the current status of the accumulators.

When the user has completed debugging and wishes to restart execution, he issues a $P or n$P. Execution resumes with the breakpoint instruction. The user, in resuming execution, can set the number of times the instruction at which the break occurred will be executed before the debugger is to be reentered.

FORMAT	MEANING
$P	Set break proceed counter to +1 and proceed with execution from current break. Command $P is equivalent to 1$P.
n$P	Set break proceed counter to n, where n is the number of times the instruction will execute before a transfer to the debugger occurs; proceed with execution.
n$Q	Open break proceed counter n, where n is 0-7, and print contents.
EXAMPLE:	Suppose a user program contains three breakpoints at symbolic locations ATOM1, ATDIG, and ATOM2. A partial listing might be:

```
00011-006201-ATOM1:  CALL              ;AC0 WILL CONTAIN THE
00012-000052-        CHAR              ;INPUT CHARACTER.
00013-024000-        LDA 1, C72
00014-106032         ADCZ# 0, 1, SZC
00015-024001-        LDA 1, M60
00016-107046         ADDO 0, 1, SEZ    ;IS IT A DIGIT?
00017-000417         JMP ATOM2         ;NO

00020-045407 ATDIG:  STA 1, ATEM, 3    ;SAVE THE DIGIT.
00021-024002-        LDA 1, C12
00022-045402         STA 1, NUMB+1, 3
00023-021403         LDA 0, NUMB+2, 3 ; FORM A NUMBER
00024-025404         LDA 1, NUMB+3, 3 ;FROM THE STRING OF DIGITS.
00025-006201-        CALL
00026-000621         DMPY              ;MULTIPLY PREVIOUS
00027-000001         NUMB              ;NUMBER BY 10.
```

```
                        .
                        .
                        .
00035-000754            JMP ATOM1
00036-024003-ATOM2:     LDA 1, C133         ;IS THE CHARACTER IN
00037-106032            ADC# 0, 1 SZC       ;AC0 A LETTER?
00040-024004-           LDA 1, M100
```

Presume the user is in the debugger. He prints out his breakpoints and his current location:

```
$B
7B ATOM1
6B ATDIG
5B ATOM2

./JMP ATOM2
.= 417
.: ATOM1+6
```

} See Conversion Mode Commands.

```
ATDIG+4$R

5B ATOM2                                    ← Debugger prints status information.
0 000000 1 000000 2 001461 3 001522

5$Q 000001 3                                ← Break proceed counter 5 is changed to 3.
6$Q 000007                                    Break proceed counter 6 is at 7.

$P                                          ← Execution resumed. Since 5$Q was
                                              changed to 3, this is equivalent to
                                              3$P.

5B ATOM2
0 000003 1 000000 2 001461 3 001522         ← C(AC0) set to 3 by looping through
                                              breakpoint 5.

100$P                                       ← Execute, looping through breakpoint
                                              100 times.

5B ATOM2
0 000103 1 000000 2 001461 3 001522         ← C(AC0) set to 103 by 100 (+3) loops
                                              through breakpoint 5.
```

SEARCH COMMANDS

Search commands allow all or part of memory to be searched for the contents of the word register $W, as modified by the contents of the mask register $M.

The search algorithm is implemented as follows:

1. The contents of each successive location is read from memory.
2. The debugger performs a logical AND with this word and the contents of the mask register.
3. The result of the logical AND is compared with the word register.
    a. If the words are equal, the true contents of the memory location are printed,based on the contents of the search/punch register $H. (See page 19, "Punch Commands.")
    b. If the words are unequal, nothing is printed.

NOTE: If $H = ØØØØØ1, search output will be to the line printer. If print mode is instruction or numeric, search output will be in both modes; all other output modes will result in the single associated output. See page 19, "Punch Commands" and page 21, "Conversion Mode Commands."

FORMAT	MEANING
$W	Open word register and print contents.
$M	Open mask register and print contents. A value of -1 (177777) indicates no masking of word register.
$S	Search from location Ø to 77777 inclusive.
adr$S	Search from location Ø to adr inclusive.
adr < $S	Search from location adr to 77777 inclusive.
$adr_1 < adr_2$$S	Search from location adr_1 to adr_2 inclusive.

NOTES: In Stand-alone mode, pressing any teletype key will cause the search to be aborted.

To output all locations, $W and $M should = Ø.

EXAMPLES: If the current contents of the word register = LDA 1, FIELD and current contents of the mask register = -1, then:

```
$S                  ← might produce a response:
FIE  LDA 1 FIELD
FIE2 LDA 1 FIELD
FIE3 LDA 1 FIELD

FIE3-DIM$S          ← might produce a response:

FIE  LDA 1 FIELD
FIE2 LDA 1 FIELD

FIE2-2 < $S         ← might produce a response:

FIE2 LDA 1 FIELD
FIE3 LDA 1 FIELD

FIE2 < FIE3+SYN$S   ← might produce a response:

FIE2 LDA 1 FIELD
FIE3 LDA 1 FIELD
```

If the current contents of the word register = FIELD and the contents of the mask register = ØØØ377, then:

```
$S                      ← might produce a response:
FIE  LDA 1 FIELD
FIE2 LDA 1 FIELD
FOO  ADDZR# Ø 1 SZC     ←Match on 8 bits if FIELD = 232
FIE3 LDA 1 FIELD          (Since ADDZR# Ø 1 SZC = 103232).

$W ØØØØØØ NEG       ← NEG loaded into word register.
$M ØØØØØØ -1        ← Mask register loaded to permit
                      search on NEG Ø Ø field format.
$S                  ← A search will cause printout of all
                      NEG Ø Ø instructions.
```

EXAMINING AND SETTING SPECIAL REGISTERS

Some of the registers that are used for special purposes and can be altered by the user are:

Accumulators 0 to 3.

Symbol Table Pointer register which contains a pointer USTSS; USTSS is a User Status Table address (location 402) that contains a pointer to the beginning of the symbol table. The pointer to the end of the symbol table, USTES, is in location 403 immediately following USTSS. The register is only meaningful for Stand-alone Debug III.

Numbers register which determines whether numbers in the special registers will be interpreted as decimal (non-zero) or octal (zero).

Location register which contains a starting location set by the user, to be used when a $R command is issued, provided no argument address precedes $R. The register is only meaningful for Stand-alone Debug III.

Interrupt register which contains the status of Interrupt Enable. The register is set to -1 if interrupts are enabled when the debugger is entered. Otherwise, the register is all zeroes. The interrupt register is only meaningful for Stand-alone Debug III.

FORMAT	MEANING
$A	Print contents of the four accumulators.
n$A	Open accumulator n (n = 0-3).
$Y	Open symbol table pointer register and print contents.
$N	Open numbers register and print contents.
$L	Open location register and print contents.
$I	Open and print contents of interrupt register.

EXAMPLES: $N 000000 ← numbers register contains all zeroes (octal).

$A 0 000100 1 000040 2 000011 3 000017

Number of the register is printed followed by the contents, given in octal.

2$A 000011 000015/ ← AC2 is opened, contents altered to 000015, then closed

EXAMINING AND SETTING SPECIAL REGISTERS (Continued)

EXAMPLES: (Continued)

```
$N 000000 1          ← user puts 1 in numbers register.

$A 0 +64.  1 +32.  2 +13.  3 +15.
```

Contents of accumulators again printed, this time in decimal.

```
$N 000001 0          ← numbers register modified to zero.

$L 000071            ← contents of location register checked
$R                     before resuming execution at that location.

$Y 000402            ← symbol table pointer register normally
                       points to location 402. Contents of the
                       register can be altered, however.
```

EXAMINING AND SETTING SPECIAL REGISTERS $T AND $C

The teletype input register $T contains the status of teletype input. Bit 0 is set to 1 if teletype input is not done. The register contains the character if teletype input is done. The teletype input register is only meaningful for Stand-alone Debug III.

The carry and teletype output register $C contains the current state of the carry flag and the status of teletype output. Bit 0 is set to 1 if the carry flag is 1, bit 15 is set to 1 if teletype output is done.

FORMAT	MEANING
$T	Open and print contents of teletype input register.
$C	Open and print contents of carry and teletype output register.

EXAMPLES:

User Coding	$T and $C Register Status When Debug Entered		
A: SKPDN TTI JMP -1 JMP DEBUG	$T	000101	(A was typed)
AMOD: SKPDN TTI JMP .-1 DIAC 0 TTI JMP DEBUG	$T	100000	
B: ADCZ 0,0 DOAS 0 TTO JMP DEBUG	$C	000000	
BMOD: ADCO 0 0 DOAS 0 TTO SKPDN TTO JMP .-1 JMP DEBUG	$C	100001	

PUNCH COMMANDS

FORMAT	MEANING
$H	Open and print contents of search/punch register. Bit Ø = Ø - Punch output is to teletype punch. Bit Ø = 1 - Punch output is to high-speed punch. Bit 15=Ø - Search ($S) output is to teletype printer. Bit 15=1 - Search ($S) output is to line printer.
$F	Punch ten inches of blank tape.
n$F	Punch n inches (octal or decimal) of blank tape. A decimal point immediately following n indicates decimal output.
$E	Punch an end block on the tape.
adr$E	Punch an end block on tape with transfer to location adr when the tape is read in the binary loader.
adr_1<adr_2P	Punch in binary from address adr_1 to adr_2 inclusive.
NOTES:	Any $P command that does not contain the symbol < will be interpreted. as a break proceed command (See page 11). The Debug III version used with the Disk Operating System does not have punch commands.

EXAMPLES:

$H 1ØØØØØ	← high speed punch in effect; search output to teletype.
40. $F	← punch 40 decimal inches of blank tape.
LTT<BRR$P	← binary punch from location LTT to BRR.
LTT$E	← punch end block and set binary loader to start at LTT.
50$F	← punch 40 decimal inches blank tape.
$H ØØØØØ1	← search output to line printer.
$H 1ØØØØ1	← search output to line printer; punch output to high-speed punch.

When using the teletype punch, ($H Bit Ø = Ø), the user must stop and start the punch to prevent debugging commands from being punched as shown:

$H ØØØØØØ	← teletype punch in effect.
$F	← punch 10 decimal inches blank tape. Computer HALTs with carry light ON. User then presses ON button on teletype and presses CONTINUE

EXAMPLES: (Continued)

on the operator panel.

When punch stops, the computer HALTs with carry light OFF. The user presses OFF on the teletype punch, presses CONTINUE on operator panel.

.X<X3$P ← punch from .X to X3. User presses ON on the TTY and CONTINUE.

When punch stops, user presses OFF on the TTY and CONTINUE.

.X$E ← punch end block and set start for .X when the tape is read in. User presses ON on the TTY and CONTINUE.

When punch stops, user presses OFF on the TTY and CONTINUE.

8.$F ← punch 8 decimal inches of blank tape. User presses ON on the TTY and CONTINUE.

When punch stops, user presses OFF on the TTY and CONTINUE.

CONVERSION MODE COMMANDS

There are six different formats in which information may be printed out, and a symbol associated with each format.

FORMAT	MEANING
=	Print last quantity in numeric format.
:	Print last quantity in user symbol format.
;	Print last quantity in instruction format.
←	Print last quantity in half-word format.
'	Print last quantity in ASCII characters. (The symbol is an apostrophe or an acute accent.)
&	Print last quantity in byte-pointer format.
$=	Print information following in numeric format.
$:	Print information following in user symbol format.
$;	Print information following in instruction format.
$←	Print information following in half-word format.
$'	Print information following in ASCII format.
$&	Print information following in byte-pointer format.
NOTE:	The default format for memory words is instruction format; the default format for accumulator and other special register contents is octal.

EXAMPLES:

INIT/JMP ABC :BUFF+3 =000777 ;JMP ABC ← 1 370 '<1> <370> & 000377 1

The instruction at location INIT is printed out in default instruction format. Then in each subsequent conversion, the quantity printed last is converted to the requested format.

EXAMPLES: (Continued)

Note that non-printing ASCII characters are printed in octal, enclosed by angle brackets (< >).

```
ATI/JSR @.SAVE          ← print instruction at location ATI (Default instruction
                          format)
$:                      ← change to user symbol format.
./CALL  ↓               ← print current location and next two locations
ATI+CHAR ↓                in symbol format.
ATI+2 24050 ↙
$=                      ← change to numeric format.
./ 24050 ↓              ← print current location and next three locations
17552       017550 ↓      in numeric format.
17553       016635 ↓
17554       006331 ↙
```

ERROR RESPONSES

The debugger uses the following two error responses:

SYMBOL	MEANING	EXAMPLES
U	Undefined symbol	1400+SST/U (where SST is not found) ./LDA 1 FDF LDA 1 FFU (attempt to substitute unidentified symbol FF for user symbol FDF.)
?	Do not understand; command attempt aborted.	ADD@? -1$R ? (in each case the command is improperly terminated, contains a given symbol in an illegal position, etc.) AB$B? (An attempt to set a breakpoint at location AB when there are already 8 breakpoints set.)

OPERATION

Debug III is loaded into memory with a user program or programs. Debug III can be loaded before or after the user program.

Symbol Table

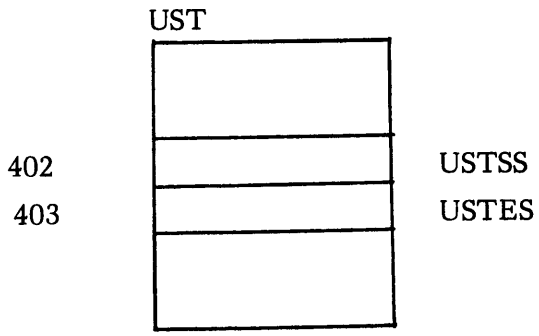
To use symbolic addressing for local symbols when debugging, the program title must be specified to the debugger. User symbols are made known to the debugger by issuing the command:

name%

where: name is the title given the user program via the .TITL pseudo-op.

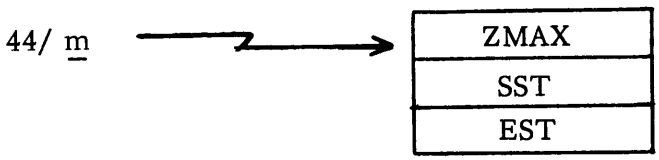
The command can be issued as soon as the debugger and user program are loaded.

By default, for both Stand-alone and DOS debuggers, the starting and terminating address of the symbol table are stored in the User Status Table in locations 402 and 403 respectively.



The debugger is initialized to have the USTSS address, 402, in the symbol table pointer register ($Y).

Users having versions of Stand-alone Debug III described in manual #093-000044-00 should note that this is a change from the pointer set in location 44 by the Relocatable Loader.



m is a pointer to ZMAX. m+1 contains the address of the start of the symbol table.

Symbol Table (Continued)

The Extended Relocatable Loader will set the proper pointer for this current version of Debug III. When using an older version of the relocatable loader, use the following procedure:

After loading Debug III:

1. Open location 44 to determine the value of m.

2. Use the $Y command to open the symbol table pointer register. It will contain 000402. Change the value of the register to contain m +1 .

3. Enter name% to make local symbols known to Debug III.

Loading Stand-alone Debug III

1. Load the Bootstrap loader as described in document 093-000002.

2. Load the Binary Loader as described in document 093-000003.

3. Load the Relocatable Loader or the Extended Relocatable Loader as described in document 093-000039.

Following are the loader signals and standard user responses to load Debug III and the user program. User responses are underscored.

SAFE = ↵ ← carriage return gives a standard save of 200 locations.

*2 ← Debug III should be in high speed tape reader for loading when this command is given.

*4S ← command to load all symbols.

*2 ← user program should be in high speed tape reader ready for loading when this command is given.

. ← additional programs can be loaded from the reader using *2 signal.
.
.

*6 ← command to print a loader map is usually given.

```
NMAX    005640
ZMAX    000255
EST     004027
SST     004077
```

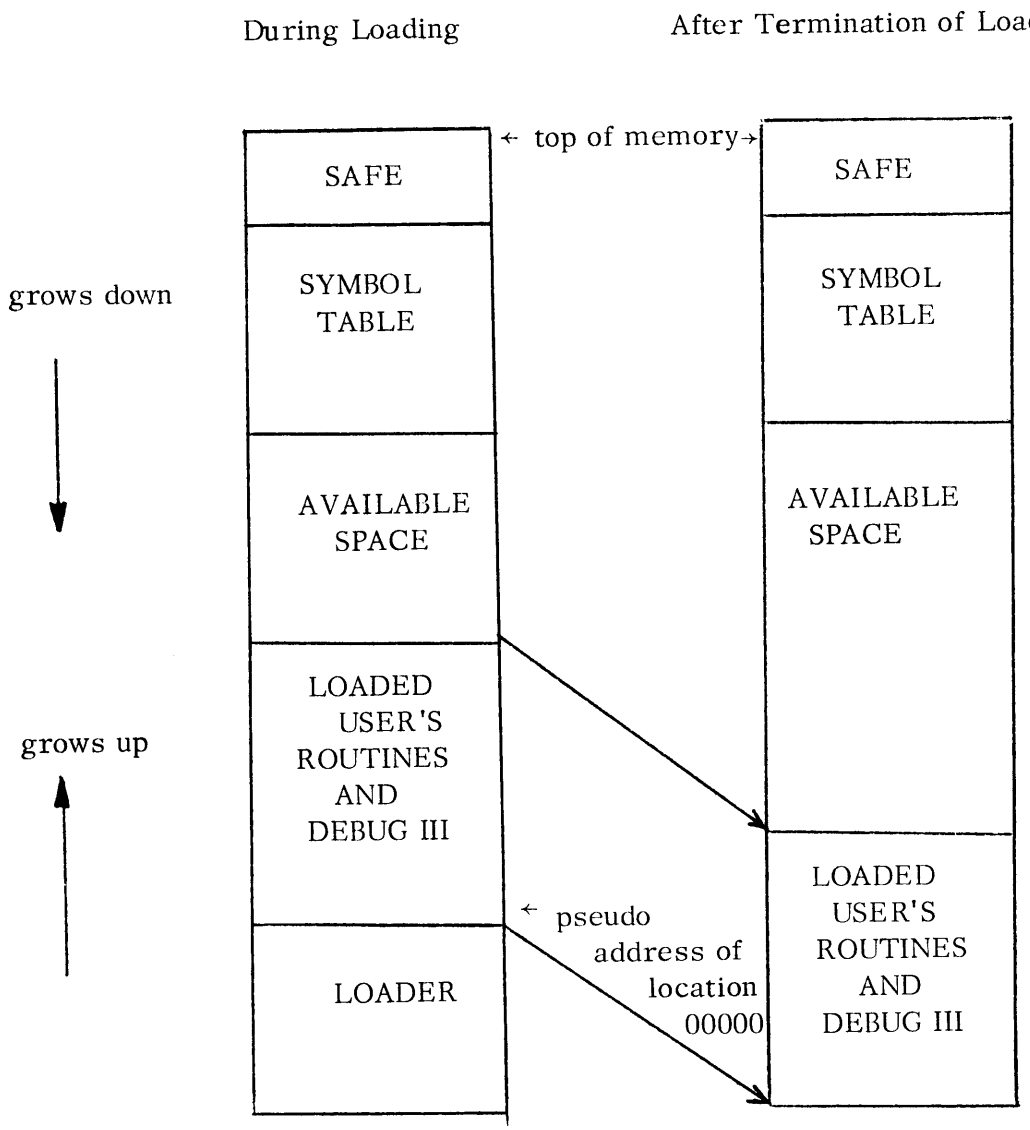
*8 ← terminate loading.

4. User sets the data switches on console panel to DEBUG and presses START.

5. User is now in the debugger and can issue any Debug commands.

Stand-alone Debug III can be used with any NOVA configuration. However, in order to load the user program, symbol table, and Debug III, a core configuration of at least 8K is recommended.

Storage is allocated by the Relocatable Loader as shown below:



Loading Debug III under DOS

Debug III is supplied as a relocatable binary file within the library SYS. LB. The library is, in turn, supplied as one file of dumped tape 088-000002. Debug III is loaded when the global switch /D is given in the RLDR command, e.g.,

RLDR /D PROG1 PROG2 PROG3 ↵

Debug III is, by default, loaded after the other programs. To adjust the loading of Debug III, the name of the library file is put into the command line in the desired place:

RLDR /D SYS. LB PROG1 PROG2 PROG3 ↵

In this case Debug III will be loaded before all other programs. The starting address of Debug III is stored in the User Status Table in location 406.

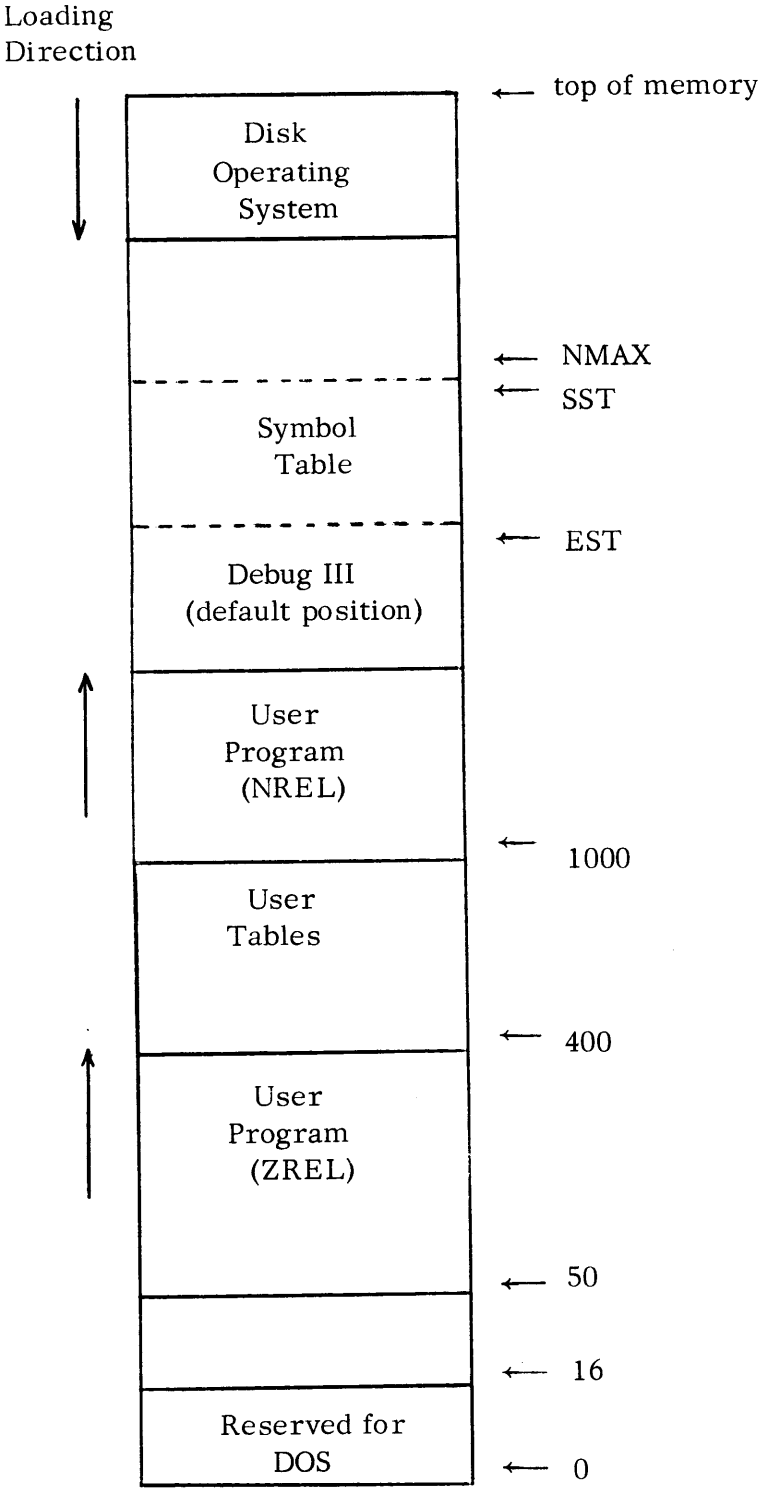
During loading the loader reserves 10_8 locations at the beginning of ZREL storage for the eight Debug III breakpoints.

To start in the debugger, the user issues the command:

DEB filename ↵

where: filename is the name of the save file produced as output by the loader.

Storage is allocated by the Relocatable Loader under DOS as shown below:



SST - Start of Symbol Table.

EST - End of Symbol Table. First address below the symbol table during loading.

NMAX - The first available address for further loading.

The Relocatable Loader permits loading beginning at location 16. ZREL code begins at location 50. NREL code begins at location 1000.

The Disk Operating System resides in upper memory only. The first 16 locations, 0-15, are reserved for use by DOS.