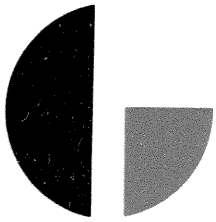


DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100

PROGRAM

Debug III User's Manual

TAPES

Binary: 089-000030

ABSTRACT

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

093-000044-00

CONTENTS

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 Debugger 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

DEBUG III uses a pointer to the user symbol table, placed by the loader in location 44. Disks are optional. The version can be used with any NOVA or SUPERNOVA configuration but 8K is recommended to insure that the debugger, user program, and symbol table can all be loaded. This version is described in full in the manual.

2. Relocatable - Disk Operating System

DEBUG III uses a symbol pointer contained in the user status area. Certain features of version 1 are unnecessary. These include all disk commands, all punch commands, and the register commands $T and $I. Any NOVA or SUPERNOVA configuration having the Disk Operating System can be used.

3. Absolute - No Disk Operating System

DEBUG III can be used without symbolic debugging features with any NOVA or SUPERNOVA. Use of disks is optional. Absolute addresses replace symbolic input and output.

Instructions for loading the various DEBUG III versions are given in the APPENDIX.

OPERATION

DEBUG III is loaded into memory with a user program or programs. DEBUG III can be loaded before or after the user program. In the relocatable, non-DOS DEBUG III described in this manual, DEBUG III starts at location 400 if loaded before the user program and can be restarted from that point if control is lost during the testing process. Loading instructions for this and other versions of DEBUG III are given in the APPENDIX.

To use symbolic addressing when debugging, the user program must contain at least one user symbol. 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.

COMMAND FORMAT

Commands to DEBUG III have one of the following formats:

c

$ c

arg$ c

where: command c is a single teletype code
argument arg may be null, a digit, an address, or an expression.
$ is the mode change character. (Press ESC Key or Shift 4 on the teletypewriter for this symbol.)

Expressions have the form:

x±x±x±...

where: each x is some symbol, octal number, or decimal number
+ and - are addition and subtraction.

Commands begin at the first character position on the teletype line.

SYMBOLS AND CONVENTIONS

SYMBOL	MEANING	EXAMPLE
+	addition	NEXT+SUM+1
-	subtraction	SUM2-SUM1
(Space)	separates instruction fields	LDA 0 0 0
,	separates instruction fields	LDA 0, 0, 0
#	adds 10 (used with a source-destination accumulator instruction)	ADCZ# 0 1 SZC= 106032 command response
@	adds 100000 to data and adds 2000 to an instruction	@= 100000 LDA 0 @0 0= 022000 command response
"	delimits beginning and end of a one character string; delimits beginning and, optionally, the end of a two-character string.	"a" "ab" "ab
.	can be used as first character of a user symbol	.PR2
.	terminates a decimal number	-5.
.	can be used to represent current location. In the example, AA is the current location symbol and the commands AA/ and ./ produce identical results. The JMP instruction causes a transfer to the current location plus 2	AA/ NEG 0 0 ./ NEG 0 0 JMP .+2

SYMBOLS AND CONVENTIONS (cont'd.)

SYMBOL	MEANING	EXAMPLE
↵	conventional representation for pressing carriage RETURN key. This closes a memory register after examination and possible modification	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 be typed.) $ signals a mode change in a command.	$S $L
↑	symbol 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 the RUBOUT key.

COMMAND SUMMARY

The Symbolic Debugger provides extensive facilities for examining and modifying program status. The summary below indicates available commands.

COMMAND		PAGE WHERE DESCRIBED
=	Print last typed quantity in numeric form.	18
:	Print last typed quantity in symbols.	18
;	Print last typed quantity in instruction form.	18
←	Print last typed quantity in half-word form.	18
'	Print last typed quantity in ASCII form.	18
$=	Print subsequent quantities in numeric form.	18
$:	Print subsequent quantities in user symbol form.	18
$;	Print subsequent quantities in instruction form.	18
$←	Print subsequent quantities in half-word form.	18
$'	Print subsequent quantities in ASCII form.	18
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
name %	Enable symbols in program name.	2, 17
$K	Kill all user symbols.	17
n$K	Kill local user symbols. n is single digit.	17
sym$K	Kill user symbol sym.	17
$S	Search all memory	10
adr$S	Search memory from location 0 to location adr.	10
adr<$S	Search memory from location adr to 077777 inclusive.	10
adr1<adr2$S	Search memory from location adr1 to adr2 inclusive.	10
$F	Punch 10 decimal inches blank tape.	9
n$F	Punch n (octal or decimal) inches blank tape.	9
$E	Punch end block on tape.	9
adr$E	Punch end block with transfer point adr.	9
adr1<adr2$P	Punch binary tape from location adr1 to adr2 inclusive.	9

COMMAND SUMMARY (cont'd.)		PAGE WHERE DESCRIBED
$T	Open TTI done register.	15
$C	Open Carry and TTO done register.	15
$I	Open interrupt register.	15
n$A	Open accumulator register n. (n = 0-3).	14
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
$P	Proceed from current breakpoint with break proceed counter set at +1.	12
n$P	Proceed from current breakpoint with break proceed counter set at n	12
adr$O	Activate disk block with address adr.	8
n)	Open disk register n. of currently active disk block.	8
n(	Open and print disk register n of currently active disk block.	8
$A	Print all accumulator register contents.	14
$M	Open mask register.	14
$W	Open word register.	14
$L	Open location register.	14
$N	Open numbers register.	14
$H	Open high/low punch register.	9
$R	Restart program at address in location register.	16
sym $R	Restart program at address sym.	16
$G	Restart program at address in location register; set C(AC3) to address of Debugger.	16 16
sym$G	Restart program at address sym; set C(AC3) to address of Debugger.	16 16

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↵
ADD+1 LDA 3 -1 3 ↵+

Register AA is opened, modified, and closed. The next location is opened and its contents printed (/ remains in effect during chained openings unless altered.)

AA/ADD 0 0 SZC SUB 0, 0↑
A STA 0 TR

Register AA is opened, modified and closed. The previous location, A, is opened and its contents printed.

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

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.

OPENING, MODIFYING AND CLOSING DISK REGISTERS

Where a disk is used, locations in a disk block can be examined and modified in a manner similar to memory locations. A disk may have up to 2000_8 blocks (0 to 1777_8). A block contains 400_8 locations. Activating a given block deactivates a previously active block.

FORMAT	MEANING
sym $O	Activate a disk block with name sym, where sym is numeric (0 to 1777_8) or symbolic.
adr (	Open active disk register adr and print contents. adr may be symbolic, octal (0-377), or decimal.
adr)	Open active disk register adr.
↵	Close active disk block register.
↵↓	Close active disk block register and open succeeding register.
↵↑	Close active disk block register and open previous register.

NOTES: The ↵, ↵↓ and ↵↑ commands are used in the same way as when closing memory registers.

These commands are not needed when using DEBUG III with the Disk Operating System.

EXAMPLES

```
1$O                          ←activate disk block 1
RAD(LDA 0 USE                ←open and print register RAD
RAD+1 ADD 0 3 SZR
RAD LDA 0 USE

0$O                          ←activate disk block 0;
                               deactivate disk block 1
```

PUNCH COMMANDS

FORMAT	MEANING
$H	Open and print contents of high/low register. Zeroes in the register mean the teletype punch; otherwise, the high-speed punch is meant.
$F	Punch 10 decimal inches blank tape.
n$F	Punch n (octal or decimal) inches blank tape.
$E	Punch an end block on the tape and halt. (Used only with teletype punch.)
adr$E	Punch an end block on tape with transfer to location adr when the tape is read in by the binary or relocatable loader.
adr1<adr2$P	Punch in binary from address adr1 to adr2.
NOTES:	Any $P command that does not contain the < symbol will be interpreted as a break proceed command. (see page 12). The DEBUG III version used with the Disk Operating System does not have punch commands.

EXAMPLES:

```
$H      177777      ←high-speed punch in effect.
40. $F              ←punch 40 decimal inches blank tape.
LTT<BRR$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.
```

When using the teletype punch, ($H contains 0), the user must stop and start the punch to prevent debugging commands from being punched as shown:

```
$H      000000      ←teletype punch in effect
$F                  ←punch 10 decimal inches blank tape. User then presses
                     ON button on teletype and presses CONTINUE on
                     operator panel.
```

When punch stops, user presses OFF on teletype punch, presses CONTINUE on operator panel.

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

. X$E               ←punch end block and set start for . X when 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.
```

SEARCH COMMANDS

FORMAT	MEANING
$S	Search from location 0 to location 77777.
adr$S	Search from location 0 to adr.
adr<$S	Search from location adr to 77777.
adr1<adr2$S	Search from location adr1 to adr2.

NOTES: Current contents of the word and mask registers determine contents searched for. See page 14. Pressing any teletype key will cause a search to abort.

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 = 000377, then:

```
 $S                         <-might produce a response:
FIE   LDA 1 FIELD
FIE2  LDA 1 FIELD
FOO   ADDZR# 0 1 SZC        <-match on right 8 bits
FIE3  LDA 1 FIELD
```

ETTING 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.

Registers 10 to 17 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 12 to resume execution after a break.

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

1. Data words or words used as indirect pointers.
2. Instructions modified during execution.
3. Instructions that enable and disable interrupts.
4. Locations where interrupts cannot be delayed for relatively long time.
5. Locations which test interrupt status.

EXAMPLES:

```
$B          ←command to print out existing breakpoints.
7B TT       ←response
6B TT2
5B TT3
TT4$B       ←command to set a new breakpoint.
$B
7B TT
6B TT2
5B TT3
4B TT4
6$D         ←command to delete breakpoint 6.
$B
7B TT
5B TT3
4B TT4
```

ENTERING AND LEAVING THE DEBUGGER VIA BREAKPOINTS

A user can set a breakpoint at a given instruction in his program, as described on page 9. Breaks are not visible to the user unless the STOP and EXAMINE switches on the operator's panel are set. 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 current status of the accumulators.

When the user has completed debugging and wishes to restart execution, he issues a $P or n$P command. 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$P	Set break proceed counter to n, where n is the number to times the instruction will execute before a transfer to the debugger occurs; proceed with execution.
n$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            ADCZ# 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

./ATOM+6 JMP ATOM2

ATDIG+4$R

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

5$Q 000001                              ←break proceed counter 5 is at 1.
6$Q 000007                              ←break proceed counter 6 is at 7.

$P                                      ←execution resumed

5B ATOM2

0 000001 1 000000 2 001461 3 001522
        C(AC0) set to 1 by single loop through breakpoint 5.

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

5B ATOM2
0 000101 1 000000 2 001461 3 001522
        C(AC0) set to 101 by 100 (+1) loops through breakpoint 5.
```

EXAMINING AND SETTING SPECIAL REGISTERS

Registers that are used for special purposes and are not accessed in sequential order are:

Accumulators 0 to 3.

Word register which can be loaded with information for searching.

Mask register which can be used to mask all or part of the word to be searched for.

Numbers register which determines whether numbers in the special registers will be interpreted as decimal or octal.

Location register which contains a starting location set by the user.

FORMAT	MEANING
$A	Print contents of the four accumulators.
n$A	Open accumulator n (n = 0 to 3).
$W	Open word register and print contents.
$M	Open mask register and print contents.
$N	Open numbers register and print contents.
$L	Open location register and print contents.
NOTE:	Other special registers are the interrupt register, the Carry and TTO-done register, and the TTI-done register, described on page 15.

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, and then closed.

```
$N 000000 1                 ←user puts 1 in numbers register.
$A 0 +80.    1  +32.   2  +9.    3  +15.

$N 000000 0                 ←numbers register modified to zero.
$W 000000 NEG               ←NEG loaded into word register.
$M 000000-1                 ←mask register loaded to permit search
                              on NEG with any instruction field format.

$S                          ←a search will cause printout of all NEG
                              instructions.
$L 000071                   ←contents of location register checked
$R                            before resuming execution at that location.
```

There are three other special registers:

Interrupt register 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.

Teletype Input register 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.

Carry and Teletype Output register contains the current state of the carry flag and 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
$I	Open and print contents of interrupt register.
$T	Open and print contents of teletype input register.
$C	Open and print contents of carry and teletype output register.
NOTE:	The DEBUG III version used with the Disk Operating System does not have $T and $I commands.

EXAMPLES:	USER CODE	REGISTER STATUS WHEN DEBUG ENTERED
A:	SKPDN TTI JMP .-1 JMP DEBUG	$T 000101
AMOD:	SKPDN TTI JMP .-1 DIAC 0 TTI JMP DEBUG	$T 100000
B:	ADC 0 0 DOAS 0 TTO JMP DEBUG	$C 000000
BMOD:	ADC 0 0 DOAS 0 TTO SKPDN TTO JMP .-1 JMP DEBUG	$C 000001

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)
sym $R	Restart program at address given by sym.
$G	Restart program at C($L); set C(AC3) to address of debugger.
sym $G	Restart program at address given by sym; set C(AC3) to address of debugger.

EXAMPLES:

```
$L 000261
$R                  ←contents of location counter checked
                      and user program restarted at that point.

7B BQ               ←after a break, user restarts his program
                      at a location different from the breakpoint.
TOP$R               ←restart program at TOP.


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

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 output but retain global symbols. Absolute values are used instead of local symbols. n is any single digit.
sym $K	Remove the user symbol named sym permanently from output. The user symbol having a value closest to sym is used instead.
name%	Restore to output all user symbols previously removed from the program named name by n$K commands.
NOTE:	Symbols are removed from output by the n$K, and sym $K commands but may still be used on input.

EXAMPLES: Suppose that a program given 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.

CONVERSION MODE COMMANDS

There are five different formats in which information may be printed out, and a symbol is associated with each format. Formats of most utility to the user are the instruction, user symbol, and numeric formats.

FORMAT	MEANING
=	Print last quantity in numeric format.
:	Print last quantity in user symbols. Where user symbols exist, symbols are used.
;	Print last quantity in instruction format.
←	Print last quantity in half-word format.
'	Print last quantity ASCII characters. (The symbol is an apostrophe or an accent acute.)
$=	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.
NOTE:	The default format for instructions is instruction format; the default format for accumulator and other special register contents is numeric.

EXAMPLES:

```
INIT/JMP ABC   :BUFF+3   =00077/   ;JMP ABC   ← 1 370   '<1><370>
```

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.

```
ATI/JSR @ .SAVE          ←print instruction at location ATI (default instruction
                           format)
$:                       ←change to user symbol format
./CALL ↵                 ←print current location
ATI+ CHAR ↵                and next two locations in user
ATI+2 24050 ↵              symbol format
$=                       ←change to numeric format
./17551      000252 ↵    ←print current location and next three locations in
17552        017550 ↵      numeric format
17553        016635 ↵
17554        006331 ↵
```

ERROR RESPONSES

The debugger uses the following two error responses:

SYMBOL	MEANING	EXAMPLES
U	Unidentified 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.)

APPENDIX

LOADING AND USE OF SYMBOLIC AND ABSOLUTE VERSIONS OF DEBUG III

1. Relocatable Debugger - No Disk Operating System

a. Loading

Load the Bootstrap loader as described in document 093-000002-01

Load the Binary loader as described in document 093-000003-00

Load the Relocatable Loader as described in document 093-000039-00. 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.

*4 S ←command to load all symbols. Loader sets pointer in location 44 and signals "S".

*2 ←User program should be in high speed tape reader ready for loading when this command is given. The user program must have at least one symbol.

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

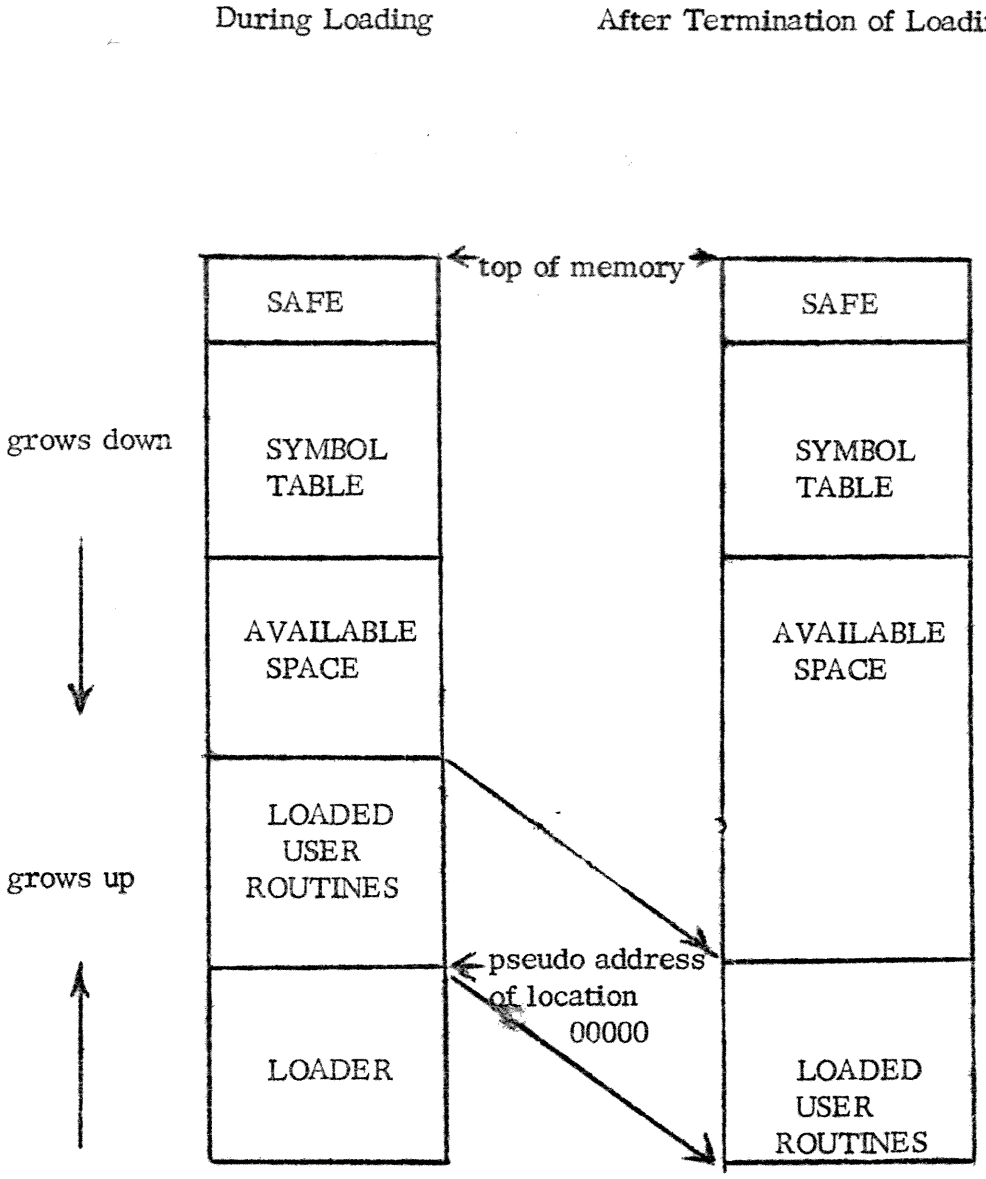
```
NMAX   005640
ZMAX   000255
DEBUG  000400
```

*8 ←Terminate loading

User sets the data switches on console panel to 400_8 and presses START.

User is now in the debugger and can issue a name % command.

Storage is allocated as shown below.



b. Use

All commands described in this manual apply to the Relocatable Debugger without Disk Operating System. 8K core configurations are recommended; it is usually not possible to load the symbol table when using a 4K configuration.

2. Absolute Debugger - 4K and 8K core configurations

a. Loading

1. Key in the Bootstrap loader origin

07757	=	4K configuration
17757	=	8K configuration

2. Enter core configuration in data switches

07770	=	4K
17770	=	8K

3. Mount Binary loader tape in input device and set BIT 0 data switch for input device

1	=	high speed tape reader
0	=	teletype reader

4. Press RESET. Press START.

5. Mount DEBUG III tape in input device and set data switches as follows: BIT 0 data switch indicates input:

0	=	teletype reader
1	=	high speed tape reader

Remaining data switches indicate core configuration:

07777	=	4K configuration
17777	=	8K configuration

6. Press START.

7. Mount user program tape in input device and press START.

b. Use

Input to and output from absolute versions of DEBUG III include all instruction mnemonics and symbols (@, #, etc.) used in assembly language. No pointer is set by the binary loader to a symbol table, however, and symbols cannot be interpreted.

Examination of a series of memory locations might result in the following printout:

```
1712/
1712        STA     2 2107
1713        LDA     0 1654
1714        MOV     0 0 SZR
1715        JSR     1677
1716        LDA     0 1651
```

The command: name%

is meaningless in absolute debugger mode. Otherwise, all commands described in this manual are available for debugging if absolute locations are substituted for user symbols.

The commands:

: and $:

which normally cause a change to symbolic printout will be interpreted by the debugger to mean:

= and $ =

or a change to numeric format.

3. Relocatable Debugger - Disk Operating System

Information on loading the relocatable debugger used with the operating system will be available at a later date. Since the operating system handles many of the functions described in this manual, the following commands will be unnecessary:

sym $O adr (adr)	disk commands
$F n $F $E adr $E adr 1 <adr 2$P	punch commands
$I $T	interrupt and TTI register commands