Advanced Operating System (AOS) Debugger and Disk File Editor

User's Manual

093-000195-02

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

Ordering No. 093-000195
© Data General Corporation, 1976, 1977, 1978
All Rights Reserved
Printed in the United States of America
Revision 02, June 1978
Licensed Material - Property of Data General Corporation

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel, licensees, and customers. The information contained herein is the property of DGC and shall not be reproduced in whole or in part without DGC prior written approval.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic, or listing errors.

Advanced Operating System
(AOS)
Debugger and Disk File Editor
User's Manual
093-000195

Revision History:

Original Release - April 1976
First Revision - April 1977
Second Revision - June 1978

This document has been extensively revised from revision 01; therefore, change indicators have not been used.

The following are trademarks of Data General Corporation, Westboro, Massachusetts:

U.S. Registered Trademarks			Trademarks
CONTOUR I	INFOS	NOVALITE	DASHER
DATAPREP	NOVA	SUPERNOVA	microNOVA
ECLIPSE	NOVADISC		

Preface

This manual describes the Advanced Operating System (AOS) Debugger and its companion utility, the AOS Disk File Editor (called DEDIT). Both are interactive programs which you execute from the Command Line Interpreter (CLI). The Disk File Editor commands are a subset of Debugger commands. In AOS, executable programs (.PR files) are stored on disk precisely as they will be brought into memory and executed. Corresponding addresses on disk and in main memory have the same contents. Commands that affect *memory* locations in the Debugger affect *disk* locations when you execute DEDIT. The changes you make with the Debugger are effective only for that run of the Debugger; when you exit from it, the changes are wiped out. DEDIT changes, however, and Debugger changes in a shared area, are permanent and remain in the disk file.

You can use DEDIT to edit any kind of file on disk (a data file, for example). Normally, you correct a *program* using the Debugger, test it, and then make the changes permanent; you can make permanent changes to the disk file using DEDIT. Eventually, you will want to make permanent changes to the source program using the text editor; you then reassemble or recompile the source program and rebind it.

Although you can debug programs written in higher-level languages, like FORTRAN, you must know what assembly-language statements your compiler produces. To correct higher-level language programs, you may find that the runtime error messages guide you to problem areas more quickly. Your compiler manual will help you interpret these messages.

If you are completely unfamiliar with the debugging process, read the chapter in *Learning to Use Your Advanced Operating System* (093-000196) which contains a sample debugging session on an assembly-language program.

The *Debugger and Disk File Editor User's Manual* is organized as follows:

Chapter 1	Introduces the Debugger utility. All material in Chapter 1, except for the command that calls the Debugger and the material on breakpoints and on starting user program execution, applies also to the Disk File Editor.
Chapter 2	Describes arithmetic, logical and Boolean expressions. You can use these in both DEBUG and DEDIT.
Chapter 3	Explains the DEBUG/DEDIT 2-part address mechanism which allows you to access any bit, byte or word location in your memory area (Debugger) or disk file (DEDIT).
Chapter 4	Describes Debugger commands. Those that do *not* apply to the Disk File Editor are noted.
Chapter 5	Explains DEDIT -- how to invoke DEDIT from the CLI.
Appendix A	Explains the DEBUG and DEDIT error messages.
Appendix B	Summarizes DEBUG and DEDIT commands for easy reference.
Appendix C	Explains how to check the ANSI status of your console, since it determines which key you press for the new-line and (CR/LF) functions.

Correcting Typing Errors

You can correct your typing errors in either of the following ways:

- Press the RUBOUT key to delete the previously-typed character and backspace the cursor. For example:

  P 302 (RUBOUT) (RUBOUT))

  deletes 2 and 0 and repositions the cursor:

  P 3_

- Press CTRL-U to delete the entire command line.

Reader, Please Note:

The Debugger and Disk File Editor differ from other AOS programs (like the CLI) in that they do not treat the new-line, carriage-return and line feed keys on your console the same.

In this manual, the key you press to type an ASCII 012 is called a new-line and is represented in formats as). The key you press to type an ASCII 015 is represented in this book as (CR/LF). You use a new-line to delimit most DEBUG/DEDIT commands; (CR/LF) is itself a command.

ANSI-standard terminals have a new-line key and a carriage-return key on their keyboards; non-ANSI terminals have a carriage-return key and a line-feed key. Appendix B describes how to tell whether your terminal is an ANSI or non-ANSI standard model and whether its device characteristics are properly matched; it describes which key you would press for a new-line character and which for a (CR/LF).

Other notation conventions we use in this manual are:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{ \begin{array}{l} \text{required}_1 \\ \text{required}_2 \end{array} \right\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.
↑	SHIFT-N. Press the SHIFT and N keys to produce ↑.

All numbers are octal unless we indicate otherwise by using a decimal point; 3., for example.

Finally, we usually show all examples of entries and system responses in THIS TYPEFACE. But, where we *must* clearly differentiate your entries from system responses in a dialog, we will use

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR THE SYSTEM RESPONSE

Contents

Chapter 1 - Introduction

Entering and Terminating the Debugger 1-1
Debugger Command Formats 1-1
Temporary Variables 1-2
Debugger Error Response 1-2

Chapter 2 - Expressions

Definition of an Expression 2-1
Evaluating an Expression 2-1
Definition of Expression Operators 2-1
Arithmetic Operators 2-1
Indirect Operator @ 2-2
Logical Operators 2-2
Boolean Arithmetic Operators 2-3
Hierarchy of Operator Evaluation 2-3
Using Assembler Instructions in Expressions 2-4
Characters 2-4

Chapter 3 - Debugger Addressing

DOT Symbol (Period) 3-2

Chapter 4 - Debugger Commands

Breakpoint-Related Commands 4-1
Set a Breakpoint 4-1
Examine Breakpoints 4-2
Delete Breakpoints 4-3
Start User Program Execution 4-3
Set Variable Command 4-3
Display Accumulator Command 4-3

Chapter 4 (continued)

Location-Related Commands 4-3
Display Contents of a Location 4-3
Modify Contents of a Location 4-4
Display the Next Data Item 4-4
Display the Previous Data Item 4-4
Display a Range of Data Items. 4-5
Suppress Symbols 4-5
Display an ASCII String 4-6
Mode-Related Commands. 4-6
Display Current Display Modes. 4-7
Display Last Item with Different Display Modes. 4-7
Floating-Point Accumulator-Related Commands 4-8
Set Floating-Point Accumulator 4-8
Display Contents of a Floating-Point Accumulator 4-8
Floating-Point Status 4-8
Display Linked Elements 4-8
Compute an Expression and Display its Result 4-9
Interpret an Error Code 4-9
Append a Symbol Table 4-9
LOG Commands 4-9
Debugging Shared Libraries 4-9
Terminate the Debugger 4-10

Chapter 5 - AOS Disk File Editor (DEDIT)

Appendix A - Error Messages

Appendix B - DEBUG/DEDIT Command Formats

Breakpoint Commands (Do not apply to DEDIT) B-1
All Other Commands B-1

Appendix C - Command Terminator Keys and ANSI and Non-ANSI Standard Consoles

Chapter 1
Introduction

Debugging is the process of detecting, locating, and removing errors from your program. When you are debugging a program, you can control its execution and halt it at specified addresses.

With the debugging commands, you can:

- Monitor and alter memory locations, accumulators, and the carry bit.
- Set, delete, and examine breakpoints.
- Restart program execution at desired points within the program.
- Set the Debugger output format.
- Display portions of the program while you're debugging it.

After you determine the addresses and set the desired breakpoints, you start execution of the program. When it reaches a breakpoint, the program halts before executing the instruction at the breakpoint location. At this point you issue debugging commands. Then you can restart program execution either at this address, or at some other address within the program.

Entering and Terminating the Debugger

To debug a program, enter the following CLI command:

DEBUG pathname *[arguments-for-process]*)

pathname is either a filename in the working directory, or a pathname indicating the location of the file outside of the working directory. Pathnames are fully explained in the *AOS Programmer's Manual.*

For example:

DEBUG MYPROGRAM)

loads MYPROGRAM and enters the Debugger.

You can also enter the Debugger by coding the system call ?DEBUG in your program at the point where you want to transfer control to the Debugger. Like other system calls, ?DEBUG has an error and normal return, and when you key the P (proceed) Debugger command, the Debugger will return control to your program at the normal return.

After you successfully enter the Debugger, it displays the following prompt:

+

You then key in debugging commands immediately following the plus sign.

When you have completed debugging your program, return to the CLI by keying the command.

BYE)

Debugger Command Formats

You input a Debugger command in one of two different formats:

Type 1. command-code *[□ arg [;arg]...]*)
Type 2. *[arg[;arg]...]* keystroke-command

A Type 1 command consists of a command-code, followed optionally by a blank and a list of from one to *n* arguments. When you supply two or more arguments, you must separate them with semicolons (;). You terminate the command line by keying in a new-line character ( )) as described in the Preface.

For example:

P)

The P command-code directs the Debugger to proceed with program execution. In this example the program begins program execution at the address contained in the location counter.

A Type 2 command consists of an optional argument list (where optional arguments are in lowercase italic letters), followed by a single-character keystroke-command. An argument consists of one or more ASCII characters. When you supply two or more arguments, you must separate them with semicolons (;). The single keystroke-command determines the action. The **boldface** lowercase words in the format indicate that you must supply both the appropriate arguments and the keystroke-command terminator. In this manual, user input IS IN THIS TYPEFACE; Debugger output *IS IN THIS TYPEFACE.*

For example:

3105: *000015* +

The Debugger displays the contents of the location referenced by octal address 3105; the colon (:) keystroke-command specifies the display action. The contents of the location are displayed after you type the colon.

You can also use the colon alone (without an argument). In this case, the Debugger uses the contents of the last displayed location as the address of the next location to be displayed.

Example:

3105: *000050* + (display contents of location 3105)
: *000007* (display contents of location 50)

Other keystroke-commands include the equal sign (=), (CR/LF), and ↑ (SHIFT-N).

Temporary Variables

When you use the Debugger, you can define temporary variables using the SET command.

The Debugger itself maintains a special set of temporary variables. You can change these variables, just as you would your own variables, using the SET command. The values of any temporary variables are maintained only until you return to the CLI.

The following special symbols are the temporary variables maintained by the Debugger. You can use them in an expression as you would any symbol. (Note: You cannot use these special Debugger variables in the Disk File Editor, but you can still use the SET command to define your own temporary variables when you are in DEDIT.)

Symbol	Represents
#P	Current program counter (points to a breakpoint when the program halts)
#C	Carry bit
#0	Accumulator 0 (AC0)
#1	Accumulator 1 (AC1)
#2	Accumulator 2 (AC2)
#3	Accumulator 3 (AC3)
#R	Result of the last-executed "=" command

If you have the floating-point option:

Symbol	Represents
#FS1	High-order part of floating-point status
#FS2	Low-order part of floating-point status

Examples:

#1 = *001400*

Displays the contents of accumulator 1.

#1%EQ%1400 = *000001*

In this example the contents of accumulator 1 are 001400. The comparison is true.

Debugger Error Response

The Debugger checks all command lines for syntax errors. If it detects an error, it displays an appropriate error message; you can then enter the correct command. For a list of Debugger error messages and their meanings, see Appendix A.

End of Chapter

Chapter 2
Expressions

Definition of an Expression

An expression is a combination of variables, constants, special Debugger symbols, user-program symbols, and operators. Spaces can be used in expressions for legibility and are optional items.

Evaluating an Expression

You can have the Debugger evaluate an expression and either display the result, or use the result as an address and display the contents of that location. In both cases, the format is:

expression { : or = } *RETURNED VALUE*

Using the equal sign will give you the numerical value of the expression, while the colon will give you the contents of the location symbolized by the expression.

Example:

Given the following locations and their contents

Address	Contents
0015	000035
0016	002002
0017	010236
0020	000007

15+3= *20*
15+3: *000007+*

The octal digits 15 and 3 are the expression constants, the plus sign (+) is the arithmetic operator, and the equal sign and colon are the keystroke-commands. Notice that you do not use the new-line key (⤶) after the keystroke command.

NOTE: The Debugger accepts constants as octal values. You can, however, enter decimal constants by keying one or more digits, immediately followed by a decimal point.

Definition of Expression Operators

The Debugger recognizes four types of expression operators:

- Arithmetic
- Indirect operator (@)
- Logical
- Boolean arithmetic

Arithmetic Operators

Operator	Operation
+	Addition and unary plus
-	Subtraction and unary minus
*	Multiplication
/	Division
( )	Parentheses, when used algebraically.

NOTE: If you use the division operator (/), the type of divide performed depends on the sign submode (see "Changing Display Modes" in Chapter 4). If the sign submode is US, an unsigned divide is performed. If the submode is SI, a signed divide is performed.

Examples:

3+2+1= *6*

Octal constants and octal result.

10-3= *5*

Octal constants and octal result.

8.+2.= *12*

Decimal constants and *octal* result.

11/3= *3*

Octal constants and octal result.

7/2= *3*

Octal constants and octal result. The Debugger truncates the result.

2*3+4= *12*

Octal constants and octal result.

2*(3+4)= *16*

Octal constants and octal result.

Assume the following is part of an assembly-language program:

```
        .ENT ABC
        .NREL
        .
        .
ABC:    LDA 0,B
        LDA 1,C
        ADD 1,0
        MOV 0,2
        .
        .
B:      3
C:      4
        .
        .
```

You can use the symbol ABC in an expression since the symbol is referenced in the .ENT statement. Assume the instruction LDA 0,B has been loaded in the user process area at location 1400. Then

ABC= *1400*

displays the address of ABC, and

ABC+3= *1403*

displays the address of the MOV 0,2 instruction.

Indirect Operator @

When you use the commercial AT sign (@) within an expression, the Debugger evaluates the expression using the contents of a location. The AT sign must immediately precede the location's address, and implies "use the contents of the location instead of the address".

Examples:

Given the following addresses and their contents:

Address	**Contents**
001400	001500
001401	000007
001500	001600
001600	000050

@1400+1= *1501*

Add 1 to the contents of location 1400 (1500+1=1501).

@(1400+1)= *7*

Note that the Debugger does not evaluate this in the same way as the first example, but evaluates the expression in the following manner: first it evaluates the parenthetical expression (1400+1) and uses this as the address, then it displays the contents of location 1401.

@1400: *001600 +*

uses contents of location 001400 (1500) as the address and displays contents of location 001500.

@@1400: *000050 +*

This illustrates a double level of indirection. The Debugger uses the contents of location 1400 (1500) as the first address; uses the contents of that location (1600) as the second address; and displays the contents of that location.

Logical Operators

Logical operators compare values. When you use a logical operator in an expression, the comparison returns a value of 1 if true, and 0 if false. The Debugger compares expression values following algebraic rules.

The following logical operators compare two 16-bit signed integers (in twos-complement form).

Operator	Operation
%EQ%	Equal to
%LT%	Less than
%LE%	Less than or equal to
%GE%	Greater than or equal to
%GT%	Greater than
%NE%	Not equal

Examples:

5%EQ%5= *1*

5%GT%6= *0*

6%NE%5= *1*

0%GT%-1= *1*

Note that you enter a negative number by preceding the value with a minus sign (-).

@1401%EQ%7= *1*

In this example the contents of location 1401 are 000007. The Debugger evaluates the expression using the location's contents.

The following logical operators compare two 16-bit unsigned integers.

Operator	Operation
%EQ%	Equal to
%ULT%	Less than
%ULE%	Less than or equal to
%UGE%	Greater than or equal to
%UGT%	Greater than
%NE%	Not equal

Note that the EQ and NE operators are the same for both signed and unsigned logical compare operations.

Examples:

5%EQ%6= *0*

6%NE%7= *1*

5%UGE%5= *1*

3%UGT%6= *0*

8.%ULE%11= *1*

Decimal 8 less than octal 11

-1%UGT%0= *1*

This and the example below are true because the comparison is of 16-bit integers, with no sign bit.

-3%ULE%5= *1*

Boolean Arithmetic Operators

The following operators perform Boolean arithmetic operations.

Operator	Operation
%AND%	Logical and
%OR%	Inclusive or
%XOR%	Exclusive or

Examples:

6%AND%3= *2*

14%OR%3= *17*

13%OR%6= *17*

12%XOR%6= *14*

(5%LT%6)%OR%(3%GE%7)= *1*

The expression is evaluated as follows:

```
(5%LT%6)%OR%(3%GE%7)= 1
    1      %OR%    0
           1             = 1
```

(7%GT%10)%AND%(6%ULE%4)= *0*

Hierarchy of Operator Evaluation

The Debugger evaluates expression operators in the sequence shown in the following table. You can enclose portions in parentheses (as in algebraic notation) to modify the order of evaluation.

Operator	Operation
@	Indirect
+,-	Unary plus and minus
*,/	Multiplication and division
+,-	Addition and subtraction
%EQ%,%LT%,%LE%, %GE%,%GT%,%NE%, %ULT%,%ULE%, %UGE%,%UGT%	Arithmetic comparisons (all of equal priority)
%AND%, %OR%, %XOR%	Boolean

Examples:

5+3*2= *13*

5+3*-2= *177777*

6+3%AND%5+5= *10*

5%LT%6%AND%79.%ULE%4%OR%10%NE%7= *1*

This expression is evaluated as follows:

```
5%LT%6%AND%79.%ULE%4%OR%10%NE%7= 1
   1   %AND%    0   %OR%    1
            0       %OR%    1
                  1              = 1
```

Any intermediate values the Debugger generates while computing an expression's value will be truncated to the least significant 16 bits.

Using Assembler Instructions in Expressions

You can use Macroassembler instructions in Debugger expressions. To do so, you must enclose the instruction within angle brackets, < >. The Debugger then assembles the instruction and evaluates it as a 16-bit integer, which it uses in further computations in place of the instruction. The symbols # and @, when used inside angle brackets, retain the meaning assigned to them by the Macroassembler.

Examples:

<LDA 3 16>+5= *34023*

<JSR @ 17>= *6017*

Characters

You can type any ASCII characters which are not self-delimiting commands (comma, colon, etc.). To insert a *single* ASCII character, precede it with the quote character ("). To insert a *pair* of ASCII characters in a word, use the format a*400+b where a and b are the characters to be inserted. You can use characters in this form in any expression.

The following example displays location 1000 and replaces its contents with ASCII "A":

1000: *000000* + "A)

In the example below, the Debugger again displays location 1000 and replaces its contents with ASCII "A". This time the (CR/LF) delimiter displays the next location's contents.

1000: *000000* + "A(CR/LF)
1001:000401 +

In the following example, the Debugger inserts an ASCII "A" in the left byte of location 1000 and a "B" in the right byte.

+ 1000: *000000* + "A*400+"B)

The CR/LF delimiter could be used instead of new-line). In this case, the Debugger would also display the next location's contents.

End of Chapter

Chapter 3
Debugger Addressing

The Debugger uses a two-part address to reference locations in the user process area. The first part (base-address) is a number or symbol that references a 16-bit word; the second part (offset) is the number of words, bytes, or bits added to the base-address. By using this two-part address, you can reference any bit, byte, or word location in the user process area.

"Address" in this manual indicates you can use any of the following addressing formats (where base-address and offset refer to any valid Debugger expression):

base-address &address-mode-character offset

base-address &address-mode-character

&address-mode-character offset

offset (Uses preset default address mode.)

The base-address is the address of a word in the user process area. The ampersand (&) indicates that the following character is an address-mode character, and offset is the number of words, bytes, or bits added to the base-address (as specified by the address-mode character).

The address-mode character specifies the type of offset you want to enter (words, bits, or bytes). You can use the address-mode character with both base-address and offset, or with the base-address alone, or with the offset alone. When you enter the Debugger, the default address mode references a 16-bit word. You can change this default mode with the MODE command described in Chapter 4. When you use the address-mode character in an address, it temporarily overrides the previously-set default address mode, but is active only for the expression you are entering.

The address-mode characters are:

Character	Action
W	Offset references a 16-bit word.
Y	Offset references an 8-bit byte.
I	Offset references a single bit.

Examples:

Given the following locations in the user process area:

Address	Contents
001400	001500
001401	000007
001402	005001
001403	001600
001404	000050

1400 &W 1: *000007* +

adds 1 to base-address 1400 and displays its contents.

1402 &W -1: *000007* +

adds -1 to base-address 1402 and displays its contents.

Note that in both of the above examples, the offset added to the base-address is in words (&W).

1401 &Y 7: *050* +

adds 7 bytes (&Y) to base-address 1401 and displays the referenced byte (second byte in 1404).

1400 &I 6: *1* +

adds 6 bits to base-address 1400 and displays the referenced bit (bit 6 in location 1400).

1400 &W: *001500 +*

displays the contents of location 1400. When you omit the offset from the address, the offset is assumed to be 0.

1400 &Y: *003 +*

displays byte 0 (left byte) of location 1400.

&Y 3003: *007 +*

displays right byte of word 1401 (byte 3003 in your address space). When you omit the base-address, it is assumed to be 0.

&Y 1401*2+1: *007 +*

also displays right byte of word 1401.

&I 30006: *1 +*

displays bit 6 of word 1400 (bit 30006 in your address space).

1400: *001500 +*

In this example the address-mode character is omitted, so the Debugger used the previously-set address mode (in this case, word addressing).

DOT Symbol (Period)

The DOT symbol (.) is the address of the last location referenced by a Display or Modify command (see Chapter 4), and you can use it in any valid Debugger expression. If you use it as part of an address that contains an address-mode character, or in an arithmetic expression, then the Debugger uses only the base-address portion of the address of the last-referenced location.

Examples:

1400: *001500 +*
.: *001500 +*

.+2: *005001 +*

.&Y 5: *050 +*

1400: *001500 +*
@.+3= *1503*

End of Chapter

Chapter 4
Debugger Commands

Breakpoint-Related Commands

When an executing program encounters a breakpoint, it immediately passes control to the Debugger before executing the instruction at the breakpoint location. You can then key in debugging commands to examine and modify memory locations, accumulators, the carry bit, and the program counter. Then you can restart the program at the breakpoint, or at any other location within the program.

Set a Breakpoint

You can set as many breakpoints as you need, and you can specify if a breakpoint is conditional. For a conditional breakpoint, you supply an expression as part of your breakpoint-setting command. When the breakpoint is reached, the Debugger evaluates the expression and either skips or stops at the breakpoint according to the result of the evaluation. Figure 4-1 is a diagram of the logic for breakpoints set with Debugger.

To set a breakpoint, key in a breakpoint command in the following format:

B *[address][;breakpoint-condition][;breakpoint-count]*)

where *breakpoint-condition* and *breakpoint-count* can be any valid Debugger expression, and *address* must be the address of a 16-bit word.

The Debugger will:

1. Place the breakpoint at the location specified by *address.* If *address* is omitted, the Debugger will place the breakpoint at the last-referenced word.

2. Analyze and save the *breakpoint-condition* as a 16-bit unsigned integer. This tells the Debugger the circumstances under which the breakpoint will be taken. If you leave out *breakpoint-condition,* the breakpoint is unconditional.

3. Analyze and save the *breakpoint-count.* This is the number of times that the Debugger will bypass the breakpoint (when the *breakpoint-condition* is true) before taking it. If there is no *breakpoint-count,* the Debugger automatically sets the count to 1.

4. When your executing program encounters the breakpoint, the Debugger evaluates the *breakpoint-condition* in terms of the existing program state (accumulators, program counter, contents of location, etc.). If the evaluation of the *breakpoint-condition* produces a zero as the result, the Debugger will skip the breakpoint and proceed with the program. Only if the result is nonzero (true) will the Debugger decrement the *breakpoint-count.* When the *breakpoint-count* reaches zero, the Debugger will take the breakpoint, stop execution of the program, and reset the *breakpoint-count* to 1.

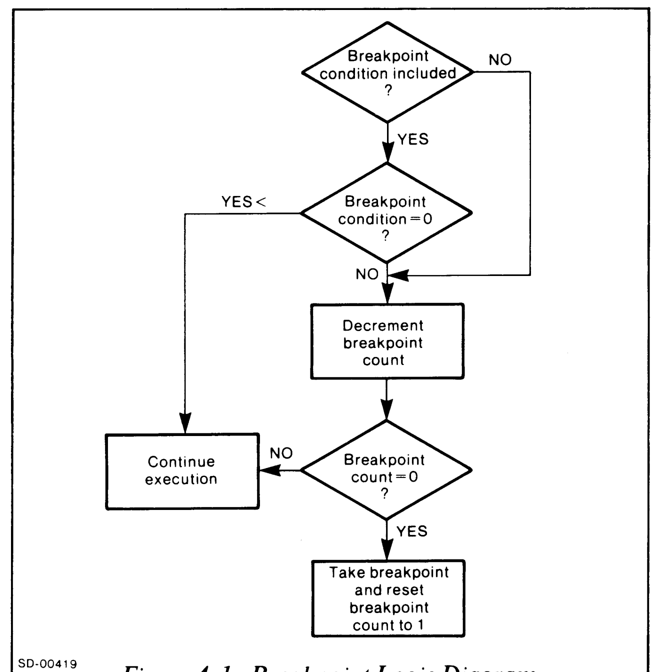

Figure 4-1. Breakpoint Logic Diagram

NOTE: You must specify breakpoints only at executable instructions. Do not set breakpoints at the following locations: data words, instructions modified during program execution, or the second word of a two-word instruction.

Examples:

B 447)

Set unconditional breakpoint at location 447.

B 447;#0%EQ%36)

Set conditional breakpoint at location 447. (See Chapter 2 for a description of these logical operators.) Note that breakpoint count is omitted, so the Debugger automatically sets the count to 1. When the breakpoint is reached, the Debugger evaluates the *breakpoint-condition* as follows:

- Compare the contents of accumulator 0 against the constant value 36.
- If the result is nonzero (true), take the breakpoint.
- If the evaluation result is zero (false), skip the breakpoint and do not alter the breakpoint count.

B 447;@1503%GT%0%OR%#3%NE%6)

Same type of conditional breakpoint as above. The Debugger evaluates the breakpoint-condition as follows: if the contents of location 1503 are greater than 0, or if accumulator 3 does not contain 6, take the breakpoint.

B 447;;6)

Set unconditional breakpoint. Note that if you omit a breakpoint-condition and include a breakpoint count you must include an additional semicolon (;). In this example, the count is set to 6. Thus, the program will pass through the breakpoint 6 times before the Debugger takes the breakpoint; but the Debugger will decrement the count each time the program passes through it, until it equals zero. When the count equals zero, the Debugger takes the breakpoint, then resets the count to 1. In this way, you can set the breakpoint count to skip breakpoints as required.

B 447;#0%NE%0;6)

Set conditional breakpoint; the breakpoint-count is set to six. Each time the Debugger evaluates the breakpoint-condition and the result is nonzero (true), it decrements the count by one. When the count equals 0, the Debugger takes the breakpoint.

447: *001500* +
B)

The first command displays the contents of location 447. The following B command sets an unconditional breakpoint at location 447.

NOTE: Breakpoint commands do not apply to the Disk File Editor.

Examine Breakpoints

To help you examine breakpoints, the Debugger assigns each one a number in the order in which you set them. The Debugger will display this number when the breakpoint is taken, or when you request the Debugger to display the existing breakpoints (see below). The Debugger returns the breakpoint identification number as:

!number

Because !number is a symbol for the breakpoint's location, you can use it interchangeably with the location address in any valid Debugger expression.

Example:

Assume you are setting the first breakpoint:

1400: *034016* + (Display contents of location 1400.)

B)	(Set unconditional breakpoint at location 1400. Since this is the first breakpoint, the Debugger assigns the value !0 to this breakpoint.)
!0= *1400*	(Display the value of the breakpoint identification number.)
!0: *034016* +	(Display the breakpoint location's contents.)

To display a complete list of your program breakpoint locations, key in the following command:

?B)

Example:

?B)

!0 7554	(where *7554* is the breakpoint address)
!1 7553 6	(where *6* is the current breakpoint count value)

Delete Breakpoints

There are two different commands which you can use to delete breakpoints. To delete one or more specific breakpoints, key in the following command:

DB address *[;address] ...*)

where *address* is the previously-set breakpoint address.

Examples:

DB 7553)

DB 7554;7553)

DB !0;7553)

Note that you can use the breakpoint identification number when deleting breakpoints. To delete all previously-set breakpoints, key in the following command:

NOBRK)

Start User Program Execution

To start or proceed with execution of your program, key in a Start User Program Execution command in this format:

P *[breakpoint-count]*)

The Debugger starts your program execution from the location contained in the program counter.

If you are at a breakpoint, you can change the breakpoint-count by keying in *breakpoint-count.* It can be any valid Debugger expression.

NOTE: "P" commands do not apply to the Disk File Editor.

Examples:

P)

Starts or continues program execution with the instruction whose address is in the program counter. #P is a *temporary variable* maintained by the Debugger. It represents the contents of the current program counter. If you want to start program execution at a point *other* than the one in the current program counter, change the temporary variable representing the current program counter (#P). Use the SET command described later in this chapter. Temporary variables are described under the SET command.

P 3)

This assumes that your program is halted at a breakpoint. The command resets the breakpoint count to 3.

Set Variable Command

To change the value of an existing temporary variable, or to create a new one, key in the following command:

SET variable-name; expression)

where expression is any valid Debugger expression, and variable-name is the name of the variable you wish to change or create. The Debugger will evaluate expression and set the result as the value of variable-name. If variable-name does not already exist, the Debugger will create it, then assign it the value of expression. Note that you also use this command to modify the accumulators (#0, #1, #2, #3) the carry bit (#C), and the program execution address (#P).

Examples:

SET #0;-1

changes the value of accumulator 0 to -1.

SET FOO;62

changes the value of FOO to 62. If FOO does not exist, the Debugger creates it and assigns it a value of 62.

Display Accumulator Command

To display the contents of accumulators 0 through 3, key in the following command:

?A)

Example:

?A)
 # 0=066666 # 1=000001 # 2=000007 # 3=000011

Location-Related Commands

Display Contents of a Location

To display the contents of a location, key in the following command:

[address]:

where *address* is any valid Debugger expression.

The Debugger displays the contents of the location referenced by *address* in the current display modes

(display modes are explained in the next command). If the address is omitted, the Debugger uses the data contained in the last displayed word as the address, and displays the contents of that location.

Examples:

3105: *000050* +

displays contents of location 3105.

: *000007* +

uses the contents of the last word as the address and displays its contents.

Additional examples of displaying a location's contents are in Chapter 2.

Modify Contents of a Location

To modify the contents of a location, key one of the following commands:

[address;] expression)
[address;] expression (CR/LF)
[address;] expression (SHIFT-N)

where *address* and *expression* can be any valid Debugger expression.

If you terminate the command with new-line), the Debugger modifies the contents of the location and displays the prompt (+) again. If you terminate the command with (CR/LF), the Debugger modifies the location's contents and displays the contents of the *next* location; if you use ↑, the Debugger modifies the location's contents, and displays the contents of the previous location.

The Debugger evaluates expression and stores its value in the location specified by *address.* If you omit the address, it stores the expression's value in the last-displayed location. In most cases you will want to examine a location before you modify it.

- If *address* or the last-displayed location is a word address, expression is stored as a word (16 bits).
- If it is a byte address, the data is stored as byte (8 bits).
- If it is a bit, the data is stored as a single bit.

If the expression evaluates to a value larger than that which will fit in the addressed location (word, byte, or bit), the Debugger gives you an error message.

Examples:

1400: *000011* + LDA 3 16)

displays contents of location 1400; then stores the instruction LDA 3 16 in location 1400.

1503 &Y 1;106 (CR/LF)
1504:401

replaces the contents of the right byte in location 1503 with 106; (CR/LF) causes the Debugger to display the contents of location 1504 *(401).*

Display the Next Data Item

After the Debugger displays the contents of a location, you can key in a (CR/LF) (ASCII 015) to display the next data item. (The command increments the offset part of the last-referenced item by 1 (word, byte, or bit), and uses that address.)

Example:

Given the following addresses and their contents:

Address	**Contents**
1400	005602
1401	000003
1402	000050
1403	000200

then,

MODE W)	(changes address mode to word addressing)
1400: *005602* + (CR/LF)	(displays locations 1400, then 1401)
1401:000003 + (CR/LF)	(displays location 1402)
1402:000050 +	
MODE Y)	(changes address mode to byte addressing)
1400 &Y: *013* + (CR/LF)	(displays left byte, then right byte in word 1400)
1400 & Y 1:202 + (CR/LF)	(displays left byte of location 1401).
1401 & Y:000 +	

We used &Y in the command 1400 &Y: because 1400 must be identified as a word, not a byte, location.

Display the Previous Data Item

When the Debugger displays the contents of a location, you can type (SHIFT-N) (↑) to display the previous data item. (This command decrements the offset part of the last-referenced item by 1 (word, byte, or bit), and uses that address.)

Example:

Given the same addresses and contents as in the previous example, then,

MODE W)	(changes address mode to word addressing)
1402: *000050* +↑	(displays location 1402, then 1401)
1401:000003 +↑	(displays location 1400)
1400:005602 +	

Display a Range of Data Items

To display a range of data items (words, bytes, or bits) in your address space, key in the following command:

DISP address-1;address-2 *[;increment][;condition]*)

The Debugger searches through all data items beginning at address-1 and ending with address-2. It successively adds the *increment* to each address, beginning with address-1, to select the locations it analyzes. The size of each location (word, byte, or bit) is specified by the address-mode character in address-1, or by the default mode if you omit the address-mode character. If you omit *increment,* then the Debugger uses an increment of 1.

For each location it selects, the Debugger evaluates *condition* (where *condition* is also any valid Debugger expression). If the result is true (nonzero), then it displays the item; otherwise the item is not displayed. If *condition* does not appear in the command, the Debugger unconditionally displays all items. Thus, you can use *condition* in this command to search for specific items in your address space.

Examples:

DISP 1400;1500)

displays the contents of all word locations beginning at address 1400 through 1500.

DISP 1400;1500;3)

displays the contents of every third word beginning at address 1400 through 1500 (i.e., 1400, 1403, 1406, etc.).

DISP 1400 &Y1;1405 &W)

displays every byte beginning with the right byte in location 1400 and ending with the left byte in location 1405.

DISP 1400;1415;;@.%NE%0)

displays the contents of every location whose contents are not equal to 0. Note that if you omit the increment and include a condition, you must insert an additional semicolon in place of the increment. The DOT symbol references the location currently being analyzed by the DISP command.

DISP 1400 &Y1;1430&W;2;@.%EQ% #3)

displays the contents of every other byte, starting with the right byte of word 1400 and ending with the left byte in word 1430, if the contents of the byte are equal to the contents of accumulator 3.

Suppress Symbols

To suppress symbols defined within a range of addresses, use the command:

NOSYM expression1;expression2)

where expression is any valid Debugger expression.

The Debugger will disregard any symbol defined between expression1 and expression2. This will help you avoid confusion caused by a large number of symbols for the locations you are using.

Example:

Given the following addresses, symbols, and contents:

Address	Symbol	Contents
1400	QAT	005602
1401	QAT+1	000003
1402	QAT+2	000050
1403	FUM	000200
1404	FUM+1	000073
1405	FUM+2	000652
1406	ZRK	003020

Without NOSYM, the dialog involved in displaying the data would be:

1400: *005602* + (CR/LF)
QAT+1:000003 + (CR/LF)
QAT+2:000050 + (CR/LF)
FUM:000200 + (CR/LF)
FUM+1:000073 + (CR/LF)
FUM+2:000652 + (CR/LF)
ZRK:003020 +

The NOSYM command, however, will make the Debugger ignore the symbols defined within the specified range:

NOSYM QAT;ZRK)
1400: *005602* + (CR/LF)
QAT+1:000003 + (CR/LF)
QAT+2:000050 + (CR/LF)
QAT+3:000200 + (CR/LF)
QAT+4:000073 + (CR/LF)
QAT+5:000652 + (CR/LF)
ZRK:003020 +

Display an ASCII String

To display an ASCII string, key in the following command:

DSTR byte-address *[;length]*)

Byte-address is the address of the first byte of the string to be displayed, and can be any valid Debugger expression. If byte-address does not contain an address-mode-character, the Debugger uses byte-address as a byte offset. This allows you to reference the contents of a location to get a byte address for display.

The byte string must be terminated by a null. If the byte string is greater than 131 characters, the Debugger will display only 131 characters. If you want to display fewer than 131 characters, specify *length* for the maximum number of characters you want to display.

Examples:

Given the following addresses and their contents:

Address	Contents	
507	001220	(Byte address of word 510)
510	040502	(AB)
511	041504	(CD)
512	042506	(EF)
513	000000	(NULL,NULL)

then,

DSTR 1220)
ABCDEF

DSTR @507;3)
ABC

DSTR 510 &Y 1)
BCDEF

Mode-Related Commands

The Debugger always displays the contents of locations according to four types of display submodes: *format, radix, shift* and *sign.* (This is in addition to the address mode, which allows you to access the contents of words, bytes, or bits.) The *format* submode defines the data representation (numeric, ASCII, symbolic, etc.); the *radix* submode defines the base of the number system used; the *shift* submode positions the data within the word; and the *sign* submode specifies signed or unsigned numbers. Once you set a display or address submode, it will be maintained until you change it.

FORMAT SUBMODES

Format Submode Character	Displays data as:
F	A 16-bit numeric constant.
H	An 8-bit numeric constant.
A	A pair of ASCII characters.
S	A symbol plus offset.
N	An instruction.
P	Single-precision floating-point data.
Q	Double-precision floating-point data.

When you enter the Debugger, the default format submode is F (16-bit numeric constant).

RADIX SUBMODES

Radix Submode Character	Action:
B	Display data in binary format.
O	Display numeric constants in octal.
D	Display numeric constants in decimal with a trailing period. The presence or absence of a trailing period lets you see at a glance whether numeric constants are octal or decimal.
X	Display numeric constants in hexadecimal (base 16).

When you enter the Debugger, the default radix submode is O (octal).

You can specify shifted data with the character:

SHIFT SUBMODE

Character	Action:
T *expression*	Display data shifted the number of bits specified in *expression*. *Expression* can be -15 to +15 bits. If *expression* > 0, the data is shifted left; if < 0, the data is shifted right.

SIGN SUBMODES

Sign Submode Character	Action:
US	Display constants without regard to sign (i.e., as 16-bit integers).
SI	Interpret the sign bit when displaying constants. The Debugger will display a negative number with a minus sign preceding it.

When you enter the Debugger, the default sign submode is US (unsigned).

ADDRESS MODES

Address Submode Character	Action:
W	Address offset references a 16-bit word. When you display data in this mode, you will display a 16-bit word.
Y	Address offset references an 8-bit byte. When you display data in this mode, you will display an 8-bit byte.
I	Address offset references a single bit. When you display data in this mode, you will display a single bit.

When you enter the Debugger, the default address mode is W (word addressing).

To change the default format, radix, sign and address mode characters, key in the following command:

MODE mode-character *[;mode-character] ...*)

(Of course, you can select only one mode character for each of the four submode, and one address mode, types, for a maximum of five mode-character arguments per MODE command.)

Examples:

MODE D:SI)

changes the radix and sign submodes.

MODE SI;Y)

changes the sign submode and address mode.

MODE Y;T-3;US;X;H)

changes all submodes and the address mode.

You can temporarily change the address mode for any address by typing an ampersand (&) and the new address mode character after the address. For example, assume that W is the current address mode:

73: *020111* + 73&Y: *040* + 73&I: *0* +

The first command displays the contents of location 73 in the default (word) mode; the second command displays the contents of 73's left byte; the third command displays the contents of 73's bit 0. Because an offset address was omitted in the Y and I commands, the Debugger assumed an offset of 0, and dealt only with byte 0 (left byte) and bit 0 of location 73.

The default address mode remains W after these commands.

Display Current Display Modes

To display the current display modes, key in the following command:

?M)

Example:

?M)
W F O US

Display Last Item with Different Display Modes

To redisplay the last displayed item in different display modes, enter one or more mode-changing characters in the format:

mode-character *[;mode-character] ...*(ESC)

where ESC is the ESC key on your console.

Example:

1400:041520 + A(ESC) *CP* (displays *041520* as two ASCII characters)

Floating-Point Accumulator-Related Commands

Set Floating-Point Accumulator

To set a floating-point accumulator, key in the following command:

SFP expression-1;expression-2)

where expression-1 is the number of the floating point accumulator:

0 = floating-point accumulator 0
1 = floating-point accumulator 1
2 = floating-point accumulator 2
3 = floating-point accumulator 3

expression-2 is the value to be stored in the accumulator.

Examples:

SFP 0;29.6)
SFP 0;041035 114631 114631 114631)

Both of these commands will store the same value in floating-point accumulator 0. If you enter a series of integers in the accumulator, the Debugger stores them in four parts of the floating-point accumulator in the order you entered them. If you enter fewer than four integers, the remaining portions are set to zero. Thus, if you are dealing with single precision numbers, enter only two values. The Debugger will set the unused parts of the floating-point accumulator to zero.

Display Contents of a Floating-Point Accumulator

You can display the contents of one or all of the floating-point accumulators by issuing the following command:

?F *[number]*)

where *number* is the number of the accumulator whose contents are to be displayed. If *number* is omitted, the contents of all floating-point accumulators are displayed.

NOTE: This command does not apply to DEDIT.

You can examine the floating-point accumulators only when the floating-point unit is in use; i.e., the current task must have a floating-point save area defined.

Floating-Point Status

Words 1 and 2 of the floating-point status are accessible by using the temporary symbols # FS1 and # FS2.

Display Linked Elements

If you have elements on a linked list, you can display any or all of them by typing the following command:

LLIST address-of-1st-element; *[link-offset]*
;[display-start-address];[display-stop-address]
;[display-condition];[terminator]
;[maximum-chain-length])

where:	means:
address-of-1st-element	the address of the first element in the linked list.
link-offset	offset into the element containing the link. If you omit this variable, it defaults to 0.
display-start-address	a Debugger address indicating the location within the element where the Debugger is to start data display (NOTE: this variable must be an *offset,* not an octal address). The Debugger adds this offset to the address of the current list element. The address mode determines whether a word, byte or bit is displayed. If you omit this variable, the default is the link offset.
display-stop-address	a Debugger address indicating the location within the element where the Debugger is to *end* data display. The address mode of this address will not change the data size displayed. If you omit this variable, the default is the start-address; data is displayed in single units automatically.
display-condition	You can include a logical or arithmetic expression to govern the display operation. It will be evaluated like any other Debugger expression. *display-condition* is analyzed after the Debugger finds the element, but before it displays data. If the result of the expression is 0 (false) the Debugger does *not* display the element. It continues to search through the chain. You can use the DOT symbol (.) to

display-condition	represent the address of the element. For example, you could examine status bits by setting a condition such as: (.+1&I0)%EQ% 1 This test bit 0 of word 1 of each element in the linked list before displaying it.
terminator	the terminator in the last link word. If you omit this variable, the Debugger will treat a 0 or -1 as the terminator.
maximum-chain-length	If you do not include a value here, the default is 32 (decimal).

NOTE: If you omit a variable, you must still include the semicolon that precedes it.

Compute an Expression and Display its Result

To compute an expression and display its result, key in the following command:

expression *[;mode-character] ...* =

where expression is any valid Debugger expression.

The Debugger evaluates the expression and displays its value immediately following the equal sign. It also stores the result in temporary variable # R.

Examples for computing an expression and displaying its value are contained in Chapter 3.

You can evaluate an expression and display its result in whatever mode you wish. To do so, key in the mode characters. If mode characters are omitted, result is displayed as a 16-bit octal constant.

Interpret an Error Code

The Debugger will display the text of an error message if you key in the following command:

MES error-code)

where error-code is any valid Debugger expression. The Debugger will display the text corresponding to this code on your console.

Example:

Assume accumulator 0 contains the value 25 in error code:

MES #0)
FILE DOES NOT EXIST

Append a Symbol Table

The Debugger automatically uses your program's symbol table when you are debugging. You can, however, add additional symbols by entering the following command:

STAB)
SYMBOL FILE NAME? symbol-table-pathname

The Debugger then adds and uses the symbols in symbol-table-pathname during the debugging session.

Example:

STAB)
SYMBOL FILE NAME? XTRSYMS.ST)

This command includes the symbols in the file XTRSYMS.ST as part of the debugging symbols.

LOG Commands

You can have all Debugger dialog saved in a file for later examination by typing:

LOG)

to which the Debugger will reply:

FILENAME?

Type the name of the file you want the dialog to be sent to from this point on. To close the file so that you can save it for later use, or enqueue it for printing, type the command:

CLOSE)

Debugging Shared Libraries

To debug a shared library, key in the following command *before* you attempt to set a breakpoint.

SHARE)
FILENAME? library-name)

where library-name is an ASCII string naming the library to be debugged. You must give this command or you will not be able to set breakpoints in the library.

You may set breakpoints in any part of the shared library which your program uses, whether or not the code is actually in your process area. After the code has been loaded, you may examine and change locations within it.

SHARE)
FILENAME? FOO.SL)

Do not debug the original copy of a shared library. In general, you should make copies of any shared program before you debug it. If your program (or the system) fails while you are debugging a shared library, breakpoints may be left inserted throughout it.

Terminate the Debugger

You can terminate the Debugger and return to the CLI by typing the command:

BYE)

End of Chapter

Chapter 5
AOS Disk File Editor (DEDIT)

The Disk File Editor utility (DEDIT) allows you to examine or modify locations in AOS disk files. You can edit any kind of file on disk using DEDIT. For example, after you have debugged a program, you might want to change its program file on disk with DEDIT to avoid reassembling and rebinding the source program.

DEDIT uses a subset of Debugger commands. The disk version of a program is not executing when you run DEDIT; thus Debugger commands which control breakpoints, program execution, special Debugger variables, and accumulators are meaningless to DEDIT. The only Debugger commands you cannot use in the Disk File Editor are those that set, display, and delete breakpoints, start user program execution and set and display special Debugger variables. You can use all the rest.

To execute the Disk File Editor (DEDIT), type the following CLI command:

XEQ DEDIT *[/switches]* ...pathname)

pathname — The pathname of the file you want to edit.

Optional Switches:

/I=pathname — DEDIT (Debugger) commands will come from the pathname specified. This lets you build a file of Debugger commands and apply it with a single DEDIT command. The file should contain commands in the normal format (each command terminated by new-line (or CR/LF) in some cases), including the last command.

/S=pathname — Include the symbol table file identified by pathname (similar to the STAB command).

/L=pathname — Save all DEDIT commands in a log file identified by pathname (similar to the LOG command).

When DEDIT is running and has successfully loaded the file you specified for editing, it displays the prompt:

+ (the same prompt as for the Debugger)

You type in commands immediately following the prompt, using the Debugger commands and expressions described in Chapters 2, 3, and 4 of this manual (unless you have used the /I switch).

When you have finished editing with DEDIT, type

BYE)

to return to the CLI.

CAUTION: If you submit Disk File Editor commands in an input file to DEDIT (instead of issuing them one by one from a console):

- Do not insert any characters or spaces between a colon command (:) and the value to be inserted (the patch value). For example:

  1234: STA O,X will not work
  1234:STA O,X will work

- You must end the DEDIT command file with a BYE command.

End of Chapter

Appendix A
Error Messages

UNKNOWN COMMAND

This happens when you enter a command such as:
1234 instead of 1234:
The Debugger tries to interpret 1234 as an instruction instead of an address.

ILLEGAL ADDRESS

You have asked the Debugger to look at a location not in your address base.

INSTRUCTION CANNOT HAVE BREAKPOINT

You are trying to put a breakpoint on an SCL instruction or an XOP1. This is illegal.

UNKNOWN SYMBOL

You have used a symbol not defined in the symbol file.

ILLEGAL INSTRUCTION

You have tried to use as an instruction a symbol that is not defined as an instruction.

NULL EXPRESSION

You didn't enter anything where an expression of some kind is required.

EXPRESSION TOO LONG OR TOO COMPLEX

The expression is too long to fit in the Debugger's work area.

UNKNOWN OPERATOR

You have used an operator other than +,-,/, etc.

EXPRESSION SYNTAX OR UNKNOWN SYMBOL ERROR

You have typed in something which the Debugger does not know how to interpret, for example: %$AF!

DATA TOO LARGE FOR ADDRESSED LOCATION

You are trying to enter a larger number (or longer string) than will fit in the addressed location.

ILLEGAL DISPLAY OR ADDRESS MODE

You used an undefined display or address mode character, or you tried to shift and the expression for the number of bits could not be computed.

NO MORE ROOM FOR SYMBOL TABLES

The Debugger can have no more than eight symbol tables in use, including the DEBUG.ST, which is always present.

COMMAND ACCEPTS NO ARGUMENTS

You have supplied arguments for a command which does not use any.

ILLEGAL COMMAND FOR UTILITY

This happens when, for example, you try to insert a breakpoint while using DEDIT.

LOG FILE ALREADY OPEN

You gave the LOG command while a previous LOG command was still in operation.

NO MORE ROOM FOR NO-SYMBOL PAIRS

The Debugger will not accept more than eight NOSYM commands.

ILLEGAL SYMBOL FILE

The file given as an argument to a STAB command is not a symbol file.

ILLEGAL BREAKPOINT ADDRESS

You have referenced a nonexistent breakpoint.

NO MORE SHARED LIBRARIES

You can debug no more than eight shared libraries at a time. If you issue a SHARE command when you have eight already, you will get this error.

TOO MANY SHARED SYMBOLS

You have tried to use more than one symbol from a shared library in an expression. This is illegal.

SHARED SYMBOL NOT CURRENTLY MAPPED

You have tried to load at a location in a shared library not in the address base.

FLOATING POINT ERROR

You are trying to set the floating-point accumulator, and either the accumulator number is invalid, or one of the numbers specified to go in one of the 16-bit parts of an accumulator is too large.

LIBRARY NOT BEING DEBUGGED

You are trying to set a breakpoint in a library not being shared.

TOO MANY CHARACTERS IN TEMPORARY VARIABLE

A temporary variable can have at most ten characters.

ROUTINE IN PROCESS OF LOAD

You are debugging a multitask program which calls routines from a shared library, and the routine you want to examine is being loaded at this time.

FLOATING POINT UNIT NOT IN USE

You are trying to use the floating-point unit without having designated a floating-point share area for the specific task.

End of Appendix

Appendix B
DEBUG/DEDIT Command Formats

This appendix lists all breakpoint commands as a group, then lists all other Debugger commands alphabetically.

Breakpoint Commands (Do Not Apply to DEDIT)

SET A BREAKPOINT
B *[address][;breakpoint-condition][;breakpoint-count]*)

DISPLAY EXISTING BREAKPOINTS
?B)

DELETE ONE OR MORE BREAKPOINTS
DB address *[;address] ...*)

DELETE ALL BREAKPOINTS
NOBRK)

All Other Commands

APPEND A SYMBOL TABLE
STAB)
FILENAME? symbol-table-name)

CHANGE DISPLAY AND ADDRESS MODE
MODE mode-character *[;mode-character] ...*)

CLOSE DIALOG LOG FILE
CLOSE)

COMPUTE EXPRESSION AND DISPLAY RESULT
expression *[;mode-character] ...*)

DEBUG A SHARED LIBRARY
SHARE)
FILENAME? library-name)

DISPLAY AN ASCII STRING
DSTR byte-address *[;length]*)

DISPLAY CONTENTS OF ACCUMULATORS 0-3
(does not apply to DEDIT)
?A)

DISPLAY CONTENTS OF A FLOATING-POINT ACCUMULATOR (does not apply to DEDIT)
?F *[number]*)

DISPLAY CONTENTS OF A LOCATION
[address]:

DISPLAY CURRENT DISPLAY MODES
?M)

DISPLAY LAST ITEM WITH DIFFERENT DISPLAY MODES
mode-character *[mode-character]* ...(ESC)

DISPLAY LINKED ELEMENTS
LLIST address-of-1st-element; *[link-offset]*
;[display-start-address];[display-stop-address]
;[display-condition];[terminator]
;[maximum-chain-length]

DISPLAY NEXT DATA ITEM
(CR/LF)

DISPLAY PREVIOUS DATA ITEM
(SHIFT N)

DISPLAY A RANGE OF DATA ITEMS
DISP address1;address2 *[;increment][;condition]*)

INTERPRET AN ERROR CODE
MES error-code)

MODIFY CONTENTS OF A LOCATION
[address;] expression) or (CR/LF)

Licensed Material - Property of Data General Corporation

SAVE DIALOG IN A FILE
LOG)
FILENAME? log-filename)

SET FLOATING-POINT ACCUMULATOR (does not apply to DEDIT)
SFP expression1;expression2)

SET THE VALUE OF A TEMPORARY VARIABLE
SET variablename;expression)

START USER PROGRAM EXECUTION (does not apply to DEDIT) P *[breakpoint-count]*)

SUPPRESS NEW SYMBOLS
NOSYM expression1;expression2

TERMINATE DEBUGGING/EDITING
BYE)

End of Appendix

Appendix C
Command Terminator Keys and ANSI and Non-ANSI Standard Consoles

It is important to know whether your consoles are ANSI or non-ANSI standard terminals, because this determines which key you should press to enter a new-line character (octal 012) and a (CR/LF) character (octal 015) when using the Debugger or Disk File Editor.

One way to tell is by checking the keyboard to see if it has a new-line and a carriage-return key. This usually means it is an ANSI standard terminal. If the keyboard has a carriage-return key and a line-feed key, it is usually a non-ANSI keyboard.

Here is an example of the three keys:

+MODE W)
+1400: *005602 + (CR/LF)*
1401& W:000003 +

This user types MODE W, a mode command, and terminates it with a new-line character. The user then types 1400 and a colon, which displays the contents of location 1400. The user then enters (CR/LF). (CR/LF) instructs DEBUG/DEDIT to display the next location and its contents.

Your terminal must be properly matched to its device characteristics. You can test this by typing the CLI command

CHARACTERISTICS)

from the console you want to test. The CLI will display the mnemonics that represent the characteristics for this console. If your terminal is ANSI standard, the mnemonic NAS (Not ANSI Standard) should appear on the line showing the characteristics that *do not* apply (/OFF). If your terminal is non-ANSI standard, NAS should appear on the line showing the characteristics that *do* apply (/ON).

In the example below, the console is a Model 4010A (hard-copy) terminal, which is non-ANSI standard.

) CHARACTERISTICS)
/HARDCOPY/LPP=66/CPL=80
/ON/ST/NAS
/OFF/SFF/EPI/SPO/RAF/RAT/OTT
/EOL/UCS/LT/ULC/PM/NRM/TO

If the NAS mnemonic appears on the wrong line for your terminal, you can correct it

A. for this run of your CLI process, by typing the command

   CHARACTERISTICS/{ON | OFF}/NAS)

B. or permanently, by having your system manager regenerate the operating system to change the device characteristics for this console.

When your console is properly matched to its characteristics, on an ANSI terminal you press the new-line key to input ASCII 012 (which we call the new-line character,)) and you press the carriage-return key to input ASCII 015 (which we call (CR/LF)). On a *non-ANSI* terminal, you press the carriage-return key, which AOS translates to new line, ASCII 012. You press the line-feed key which AOS translates to (CR/LF), ASCII 015.

You can use Table C-1 to determine whether your terminal is ANSI standard or not, if you know the model number, and whether the NAS mnemonic should be represented as OFF or ON in response to the CHARACTERISTICS command.

Table C-1. New-Line and CR/LF Keys on ANSI and non-ANSI Keyboards

Console Model Number	ANSI/ non-ANSI	NAS	If you press:	Result:	Called in AOS Manuals
4010A (TTY) 6040 (TTY) 40101 (CRT) 6012 (CRT)	Non-ANSI	ON	RETURN key	Translated to 012	new-line)
			LINE FEED key	Translated to 015	(CR/LF)
6052 and 6053 (CRT) other	ANSI	OFF	NEW-LINE key	012	new-line)
			RETURN key	015	(CR/LF)

End of Appendix

Index

Within this index, the letter "f" following a page entry means "and the following page".

+ (prompt) 1-1
= (keystroke command) 4-9
! (examine breakpoint) 4-2
: (keystroke command) 4-3
↑ or SHIFT-N (keystroke command) 4-4

A command 4-3
address modes 4-7
address-mode-characters 3-1,4-7
 I (offset reference for single bit) 4-7
 W (offset reference for 16-bit word) 4-7
 Y (offset reference for 8-bit byte) 4-7
angle brackets 2-4
append a symbol table 4-9
%AND% 2-3
ANSI and non-ANSI terminal differences C-1
arithmetic operators 2-1
 + (addition and unary plus) 2-1
 - (subtraction and unary minus) 2-1
 * (multiplication) 2-1
 / (division) 2-1
 ( ) (parenthesis for algebraic expressions) 2-1
ASCII characters
 inserting 2-4
 values for console keys C-1f
assembler instructions 2-4

B command 4-1
base address 3-1
Boolean operators 2-3
 %AND% 2-3
 %OR% 2-3
 %XOR% 2-3
breakpoint commands 4-1 to 4-3
 B (set breakpoint) 4-1
 DB (delete a breakpoint) 4-3
 ! (examine breakpoint) 4-2
 NOBRK (delete all breakpoints) 4-3
breakpoint condition 4-1
breakpoint-count 4-1
BYE command 1-1, 4-10
byte addressing 4-7

change modes 4-7
CHARACTERISTICS command C-1
characters, inserting 2-4
CLOSE command 4-9
close log file 4-9
command-code 1-1
command summary B-1f
command types 1-1
comparing values 2-2
conditional breakpoints 4-1, 4-2
consoles C-1f
CR/LF (keystroke command) 4-4

DB command 4-3
DEBUG command 1-1
Debugger command formats 1-1
DEDIT command 5-1
delete breakpoints 4-3
dialog, save Debugger 4-9
disk file editor (DEDIT) 5-1f
DISP command 4-5
display
 accumulator 4-3
 an ASCII string 4-6
 floating-point accumulator 4-8
 format 4-6
 linked elements 4-8
 location contents 4-3
 modes command 4-7
 next item 4-4
 previous data item 4-4
 radix 4-6
 range of data items 4-5
 result of an expression 4-9
 shift 4-7
 sign 4-7
 submodes 4-6
dot symbol (.) 3-2
DSTR command 4-6

entering the Debugger 1-1
%EQ% 2-2
error messages 4-9, A-1
examine a breakpoint 4-2
expressions,
  definition of 2-1
  evaluation of 2-1, 2-3
expression operators,
  arithmetic 2-1
  Boolean 2-3
  definition of 2-1
  indirect (@) 2-2
  logical 2-2

?F command 4-8
file editor, disk 5-1
floating-point status symbols 4-8
  #FS1, #FS2 4-8
format submode characters
  (F, H, A, S, N, P, Q) 4-6

%GE% 2-2
%GT% 2-2

I (address mode) 4-7
indirect operator (@) 2-2
instructions, Macroassembler 2-4
interpret an error code 4-9, A-1

keystroke commands 1-1, 1-2
  : 4-3
  CR/LF 4-4
  = 4-9
  ↑ (SHIFT-N) 4-4

linked elements, display 4-8
LLIST command 4-8
LOG command 4-9
log file
  close 4-9
  open 4-9
logical operators (for signed integers) 2-2
logical operators (for unsigned integers) 2-3
%LE% 2-2
%LT% 2-2

?M command 4-7
Macroassembler instructions 2-4
MES command 4-9
mode 4-6f
  address (I, Y, W) 4-7
  changing 4-7
  command (MODE) 4-7
  default 4-7
  format (A, F, H, N, P, Q, S) 4-6
  radix (B, D, O, X) 4-6
  sign (SI, US) 4-7

NAS mnemonic C-1f
%NE% 2-2
new-line iv
NOBRK command 4-3
NOSYM command 4-5
notation conventions iv

offset (address) 3-1
open log file 4-9
operator
  arithmetic 2-1
  Boolean 2-3
  evaluation order 2-3
  indirect (@) 2-2
  logical
    signed 2-2
    unsigned 2-3
%OR% 2-3

P command (start program) 4-3
prompt(+) 1-1

radix submode characters
  (B, O, D, X) 4-6
redisplay with different display modes 4-7

save Debugger dialog 4-9
set breakpoint 4-1
SET command 1-2, 4-3
set floating-point accumulator 4-8
SFP command 4-8
SHARE command 4-9
shared libraries 4-9
shift submode character (T) 4-7
sign submode characters (US, SI) 4-7
signed expressions 2-2
special symbols
  (#P, #C, #0, #1, #2, #3, ER, #FS1, #FS2) 1-2
STAB command 4-9
start program (P command) 4-3
suppressing symbols 4-5

terminating the Debugger 1-1, 4-10
temporary variables 1-2

%UGE% 2-3
%UGT% 2-3
%ULE% 2-3
%ULT% 2-3
unconditional breakpoints 4-1, 4-2

W (address mode) 4-7

%XOR% 2-3

Y (address mode) 4-7

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title ______________________________ No. ______________

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- ☐ EDP Manager
- ☐ Senior System Analyst
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ Other ______________

What programming language(s) do you use? ______________

How Do You Use This Manual?

(List in order: 1 = Primary use)

- ____ Introduction to the product
- ____ Reference
- ____ Tutorial Text
- ____ Operating Guide
- ____ ______________

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name ______________ Title ______________ Company ______________

Address ______________ Date ______________

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

SD-00742A

STAPLE