VMS

digital

VMS Delta/XDelta Utility Manual

Order Number AA-LA86A-TE

VMS Delta/XDelta Utility Manual

Order Number: AA-LA86A-TE

April 1988

This manual describes the VMS Delta/XDelta Utility. This utility is used primarily to debug programs that run in privileged processor mode or at an elevated interrupt priority level.

Revision/Update Information: This document supersedes the *VAX/VMS Delta/XDelta Utility Reference Manual,* Version 4.0.

Software Version: VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital™

ZK4540

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

Contents

TABLES

Preface

Intended Audience

This document is written for programmers who debug system code for device drivers and other images that execute in privileged processor-access modes or at an elevated interrupt priority level (IPL).

Document Structure

This document consists of the following four sections:

- Description—Provides an overview and detailed usage information for the Delta/XDelta Utility (DELTA/XDELTA).
- Usage Summary—Outlines the following DELTA/XDELTA information:
  - –Invoking the utility
  - –Exiting from the utility
  - –Directing output
  - –Restrictions or privileges required
- Commands—Describes DELTA/XDELTA commands, including format, parameters, and examples.
- Appendix A—Describes a debugging session using DELTA.

Associated Documents

For additional information on topics covered in this document, refer to the *VMS Device Support Manual.*

Conventions

Convention	Meaning
RET	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

New and Changed Features

The Delta/XDelta Utility incorporates the following three changes for VMS Version 5.0:

- The ;L command—The ;L command has been added. It lists all loaded executive modules and their locations.
- Multiprocessing system—XDELTA runs on a multiprocessing system, but only one processor at a time can enter XDELTA.
- Requesting interrupts—All software interrupt requests for invoking XDELTA on all processors are at IPL 14_{10}.

DELTA/XDELTA Description

The Delta/XDelta Utility (DELTA/XDELTA) consists of two debuggers: DELTA and XDELTA. The debuggers are used to monitor the execution of user programs and the VMS operating system. DELTA and XDELTA use the same commands and the same expressions. They differ in how they operate. DELTA operates as an exception handler in a process context. XDELTA is invoked directly from the hardware SCB vector in a system context.

Because DELTA operates in a process context, use it to debug user mode programs or programs that execute at interrupt priority level (IPL) 0 in any processor mode. You cannot use DELTA to debug code that executes at an elevated IPL. Invoke it from within your process by specifying it as the debugger (as opposed to the symbolic debugger).

Because XDELTA is invoked directly from the hardware SCB vector, it can debug programs executing in any processor mode or at any IPL level. Because it is not process-specific, it is not invoked from a process. To use it, you must boot the processor with commands to include XDELTA in memory. XDELTA's existence terminates when you reboot the processor without XDELTA.

1 Entering and Exiting from DELTA/XDELTA

This section describes how to invoke and how to terminate DELTA and XDELTA debug sessions.

1.1 Invoking DELTA

To invoke DELTA, perform the following steps after assembling (or compiling) and linking your program:

1. Define DELTA as the default debugger instead of the symbolic debugger. Use the following command:

   ```
   $ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
   ```

2. Use the RUN command to execute your program. Use the following command:

   ```
   $ RUN/DEBUG program-name
   ```

When DELTA begins execution, it displays its name and current version number. DELTA displays the first executable instruction in the program with which it is linked. It displays the address of that instruction, a slash (/), and the instruction and its operands.

The name, current version number, and address are displayed as follows:

```
DELTA Version 5.0
address/instruction operands
```

DELTA is then ready for your commands.

1.2 Exiting from DELTA

To exit from DELTA, type EXIT and press RETURN. When you are in user mode, you exit DELTA, and your process remains. When you are in a privileged access mode, your process may be deleted as well.

1.3 Invoking XDELTA

To invoke XDELTA, perform the following steps:

1. Bootstrap the system using a console command or a command procedure that includes XDELTA.
2. An initial XDELTA breakpoint is taken to allow setting of additional breakpoints or examining and changing locations in memory. XDELTA displays the following breakpoint message:

   ```
   1 BRK at 8000EB63
    8000EB63/NOP
   ```

3. Proceed from the initial breakpoint, using the following command:

   ;P RETURN

The procedure for bootstrapping the system with XDELTA differs depending on your processor. Each procedure uses commands that include XDELTA in memory and cause the execution of a breakpoint in VMS initialization routines. Execution of the breakpoint instruction transfers program control to a fault handler located in XDELTA. The following sections describe the procedures for bootstrapping the processors, requesting an interrupt, and setting breakpoints in program code.

1.3.1 Bootstrapping XDELTA on a VAX 8800, 8700, 8550, or 8530

To boot a VAX 8800, 8700, 8550, or 8530, use the BOOT command in the following format:

```
>>>B dddn /R5:f
```

Substitute BCI, BDA, or UDA for *ddd*. Substitute the unit number of the drive holding the system disk for *n*. The /R5 qualifier enters a value for a flag that controls the way XDELTA is loaded. The flag is a 32-bit hexadecimal integer loaded into R5 as input to VMB.EXE, the primary bootstrap program. Refer to Table DELTA–1 for a description of the valid values for this flag.

Table DELTA–1 Boot Command Qualifier Values

Value	Description
0	Normal, nonstop bootstrap (default)
1	Stop in SYSBOOT (equivalent to @DxyGEN on the VAX–11/780)
2	Include XDELTA with the system, but do not take the initial breakpoint
6	Include XDELTA with the system, and take the initial breakpoint
7	Include XDELTA with the system, stop in SYSBOOT, and take the initial breakpoint at system initialization (equivalent to @DxyXDT on the VAX–11/750)

Note: When you deposit a boot command qualifier value in R5, make sure that any other values you would normally deposit are included. For example, if you were depositing the number of the system root directory from which you were booting as well as an XDELTA value, R5 would contain both values. If the system root directory value were 40000000, and the XDELTA value were 00000005, the final R5 value would be 40000005.

For example, if you have a BCI-controlled system disk with a unit number of two, the following command would load XDELTA and take the initial breakpoint:

```
>>>B BC12 /R5:6
```

This command boots the system with BCIBOO.COM, deposits two in R3, and deposits six in R5.

You can also boot with XDELTA by editing the appropriate *ddd*GEN.COM procedure so that the unit number of the drive is deposited in R3. Then you can enter the BOOT command in the following format:

```
>>>@dddGEN
```

Substitute BCI, BDA, or UDA for *ddd*. For example, suppose the system disk is on a BCI-controlled drive. Edit BCIGEN.COM so that the unit number of the drive is deposited in R3. At the console-mode prompt, enter the following command:

```
>>>@BCIGEN
```

1.3.2 Bootstrapping XDELTA on a VAX 8650 or a VAX 8600

There are two ways to boot a VAX 8650 or VAX 8600 with XDELTA, depending on whether the console RL02 includes a boot command file in the *dd*0XDT format, where *dd* is the device code of the system disk.

If DU0XDT is present, follow the standard boot procedure except in the following two steps:

- When you specify the bootstrap device, enter the following command:

  ```
  >>>DEPOSIT R3 u
  ```

  This command deposits the unit number of the drive holding the system disk, *u*, from which to boot.

- Then enter the following command to invoke DU0XDT:

  ```
  >>>@DU0XDT
  ```

  The command procedure boots the processor and prompts you from SYSBOOT. When the SYSBOOT> prompt appears, enter any SYSBOOT command.

To continue the bootstrapping operation, enter CONTINUE.

If the console media does not have the DU0XDT file, perform a normal bootstrap procedure using an available *ddu*GEN.COM, *ddu*BOO.COM, or DEFBOO.COM procedure, including the following steps:

1 Include the /NOSTART qualifer to the BOOT command to cause the processor to pause and prompt for console commands prior to starting the VMB initialization routines.

2 Select a value for the boot flag to control the loading of XDELTA from Table DELTA–1 in Section 1.3.1.

3 Examine the value of the boot flag in R5. If it is not the value you want, deposit the correct value.

For example, the following procedure is used to boot a VAX 8600 to include XDELTA, stop in SYSBOOT, take the initial breakpoint (flag value of 7), and continue the boot procedure:

```
>>> BOOT/NOSTART
>>> EXAMINE R5  40000000
>>> DEPOSIT R5 2  40000007
>>> CONTINUE
```

1.3.3 Bootstrapping XDELTA on a VAX 8350, 8300, 8250, or 8200

To bootstrap a VAX 8350, 8300, 8250, or 8200 processor with XDELTA, use the B command (the console BOOT command) as follows:

```
B[/R5:f] devname
```

The BOOT command qualifier, /R5:f, enters a value for a flag that controls how to load XDELTA. The flag is a 32-bit hexadecimal integer loaded into R5 as input to VMB.EXE, the primary bootstrap program. Refer to Table DELTA–1 for a description of the valid values for this flag. To use this qualifier, you must first modify the boot command procedure to remove (or comment out) the DEPOSIT R5 command.

The boot command procedure is specified by *devname* in the BOOT command. The *devname* format to use is *ddxu*, where *x* is the number of the VAXBI node to which the boot device unit is attached. If you do not specify *devname*, the default boot device is used.

If in R5 you specified the flag to load SYSBOOT, the SYSBOOT> prompt appears. Enter any SYSBOOT command.

For example, use the following commands to boot a VAX 8200 from the boot disk at VAXBI node 4, load XDELTA, stop in SYSBOOT, and take the initial breakpoint (that is, R5 contains 7):

```
>>> B/R5:7 DU40
SYSBOOT> CONTINUE
```

1.3.4 Bootstrapping XDELTA on a VAX–11/780 or a VAX–11/785

In addition to the normal system bootstrap command files, the VMS console RX01 for a VAX–11/780 or VAX–11/785 contains the following command files that bootstrap the system with XDELTA:

- DUAXDT.CMD
- DMAXDT.CMD
- DBAXDT.CMD

To bootstrap the system with XDELTA, follow the procedures in the operations guide for your processor, with the following exceptions:

- In R3, deposit the unit number of the drive holding the system disk.
- Specify one of the preceding command files.

For example, if the unit number of the drive holding the system disk is zero, enter the following command:

```
>>>DEPOSIT R3 0
```

Then specify the command file that corresponds to the drive holding the system disk. For example, if the system disk is on an RA80 drive with a controller designation of A, enter the following command:

```
>>>@DUAXDT
```

The command procedure boots the processor and prompts you from SYSBOOT. When the SYSBOOT> prompt appears, enter any SYSBOOT command.

To continue the bootstrapping operation, enter CONTINUE.

1.3.5 Bootstrapping XDELTA on a VAX–11/750, VAXstation 2000, MicroVAX 2000, MicroVAX 3500, MicroVAX 3600, MicroVAX I, or MicroVAX II

To bootstrap VMS with XDELTA on a VAX–11/750, a VAXstation 2000, a MicroVAX2000, a MicroVAX 3500, a MicroVAX 3600, a MicroVAX I, or a MicroVAX II, enter the following command to specify the boot device:

```
>>>B/n devname
```

The *B* command is the console's BOOT command.

The *devname* parameter is the name of the device from which to bootstrap the system. Specify the device name using the format ddcu. (See the *Guide to Maintaining a VMS System* for a complete description of the format of device names.) You must specify identifiers for both the controller and the unit identifiers; there are no defaults.

The */n* qualifier loads the value *n* into R5. The contents of R5 are passed as input to VMB.EXE. The value of *n* must be one of the 32-bit hexadecimal numbers described in Table DELTA–1 in Section 1.3.1.

For example, the following commands bootstrap VMS on a VAX–11/750 from DUA0 with XDELTA included, stop at XDELTA's initial breakpoint, and stop in SYSBOOT to allow setting of system parameters:

```
>>>B/7 DUA0
```

The /7 qualifier includes XDELTA in the system and stops the booting process in SYSBOOT, which issues a prompt. It also stops at the breakpoint in the system initialization routine.

You can enter SYSBOOT commands when you see the SYSBOOT> prompt.

To continue the bootstrapping operation, enter CONTINUE.

See the installation guide for your processor for more information on the B command.

1.3.6 Bootstrapping the System Using the VMS Console TU58

In addition to the normal system bootstrap command files, the VMS console TU58 for a VAX-11/730 or a VAX-11/725 contains the following command files that bootstrap the system with XDELTA:

- DQAXDT
- DQ0XDT
- DL0XDT
- DUAXDT
- DU0XDT

To bootstrap a VAX-11/730 or VAX-11/725 with XDELTA, follow the procedures outlined in the installation guide for the processor, but specify one of the preceding command files.

For example, to bootstrap the VAX-11/730 from DQA1, enter the following commands:

```
>>> D/G/L 3 1
>>> @DQAXDT
```

The first command, D, deposits the unit number, one, in R3. The second command, @DQAXDT, invokes the DQAXDT command procedure.

If the boot device is DQA0, invoke the DQ0XDT command procedure, as follows. You do not have to specify the unit number.

```
>>> @DQ0XDT
```

Either of these procedures boots the processor and prompts you from SYSBOOT. When SYSBOOT prompts you, enter any SYSBOOT command.

To continue the bootstrapping process, enter CONTINUE.

To bootstrap a VAX-11/750 with the console TU58, refer to the software installation guide for the VAX-11/750 processor. The console TU58 contains the command files DUAXDT, DMAXDT, and DBAXDT, which contain the command procedures that boot the system from DU, DM, and DB devices, respectively.

1.4 Requesting an Interrupt

If you set the boot control flag in R5 to 7, as described in Section 1.3, XDELTA will stop at an initial breakpoint during the system bootstrap process. You can then set other breakpoints or examine locations in memory.

Your program can also call the routine INI$BRK, which in turn executes the first XDELTA breakpoint. Note that INI$BRK is defined as XDELTA's breakpoint 1. Never clear breakpoint 1 from any code being debugged in XDELTA. Refer to Section 1.5 for the breakpoint procedure.

Once loaded into memory, XDELTA can also be invoked at any time from the console by requesting a software interrupt. For example, you might need to use a software interrupt to enter XDELTA if your program is in an infinite loop or no INI$BRK call had been made.

To request a software interrupt for all processors, deposit the value E_{16} into IPR 14_{16}.

For a VAX 8800, 8700, 8650, 8600, 8550, 8530, VAX-11/785, or VAX-11/780, enter the following commands at the console terminal to request the interrupt:

```
$ CTRL/P
>>>HALT
>>>D/I 14 E
>>>C
```

On a VAX 8350, 8300, 8250, 8200, a VAX-11/750, a VAX-11/730, or a VAX-11/725, enter the following commands:

```
$ CTRL/P
>>> D/I 14 E
>>> C
```

For the VAXstation 2000, MicroVAX 2000, MicroVAX 3500, MicroVAX 3600, MicroVAX I or MicroVAX II, perform the following steps:

1. Press and release the HALT button on the CPU control panel. When you release the HALT button, make sure it is popped out, or the system remains halted. You can also press the BREAK key (if enabled) on the console terminal.
2. Enter the following commands:

```
>>> D/I 14 E
>>> C
```

1.5 Accessing the Initial Breakpoint

When debugging a program, you can set a breakpoint in the code so that XDELTA gains control of program execution.

To set a breakpoint, place a call to the system routine INI$BRK in the source code.

For example, the following command calls the INI$BRK system routine to reach the breakpoint:

```
JSB G^INI$BRK
```

On systems that are not booted with XDELTA, the BPT instruction in INI$BRK is replaced with a NOP instruction.

Note: INI$BRK is defined as XDELTA's breakpoint 1. Never clear breakpoint 1 from any code being debugged with XDELTA.

The INI$BRK routine contains two instructions: BPT and RSB. You can use the INI$BRK routine as a debugging tool, placing calls to this routine in any part of the source code you want to debug. After the break is taken, the return address (the address in the program to which control returns when you proceed from the breakpoint) is on the top of the stack.

1.6 Proceeding from Initial XDELTA Breakpoints

When XDELTA reaches one of its breakpoints, it displays the following message:

```
1 BRK AT nnnnnnnn

address/instruction operands
```

On multiprocessors, the XDELTA breakpoint will be taken on the processor upon which the XDELTA software interrupt was requested, which is generally the primary processor.

At this point, XDELTA is waiting for input. If you want to proceed with program execution, enter the ;P command. If you want to do step-by-step program execution, enter the S command. If you know where you have set breakpoints, examine them using the ;B command. You can also set additional breakpoints or modify existing ones.

If you entered the ;P command to proceed with program execution, and the system halts with a fatal bugcheck, the system prints the bugcheck information on the console terminal. Bugcheck information consists of the following:

- Type of bugcheck
- Contents of the registers
- A dump of one or more stacks
- A list of loaded executive images

The contents of the program counter (PC) and the stack indicate where the failure was detected. Then, if the system parameter BUGREBOOT was set to 0, XDELTA issues a prompt. You can examine the system's state further by entering XDELTA commands.

1.7 Exiting from XDELTA

XDELTA remains in memory with the operating system until you reboot without it.

2 DELTA and XDELTA Symbols and Expressions

This section describes how to form the symbolic expressions used as arguments to many DELTA and XDELTA commands.

2.1 Symbols Supplied by DELTA and XDELTA

DELTA and XDELTA define symbols that are useful in forming expressions and referring to registers. The symbols are described in Table DELTA–2.

Table DELTA–2 DELTA/XDELTA Symbols

Symbol	Description
.	The current address, the address of the current location. The value of this symbol is set by the Open Location and Display Contents (/), Open Location and Display Instruction (!), and the Open Location and Display Indirect (TAB) commands.
Q	The last value displayed. The value of Q is set by every command that causes DELTA or XDELTA to display the contents of memory or the value of an expression.
Xn	Base register *n*, where *n* can range from 0 to F (hexadecimal). These registers are used for storing values, most often the base addresses of data structures in memory. For XDELTA only, XE and XF contain the addresses of two command strings that XDELTA stores in memory. See the Execute Command String (;E) command for more information. For XDELTA only, registers X4 and X5 contain specific addresses. X4 contains the address of the location that contains the address of the PCB of the current process on the current processor. The address that X4 contains is that of the per-CPU database for the current processor. X5 contains SCH$GL_PCBVEC, the symbolic address of the start of the PCB vector, the list of PCB slots.
Rn	General register *n*, where *n* can range from 0 to F (hexadecimal). RF+4 is the processor status longword (PSL), RE is the stack pointer, and RF is the program counter (PC).
Pn	The internal processor register at processor address *n*, where *n* can range from 0 to 3F (hexadecimal). See the *VAX Hardware Handbook* for a description of these processor registers.
G	^X80000000, the prefix for system space addresses. G2E, for example, is equivalent to ^X8000002E.
H	^X7FFE0000, the prefix for addresses in the control region (P1 space). H2E, for example, is equivalent to ^X7FFE002E.

2.2 Forming Numeric Expressions

Expressions are combinations of numbers, symbols that have numeric values, and arithmetic operators. Both DELTA and XDELTA store and display all numbers in hexadecimal. They also interpret all numbers as hexadecimal.

Expressions are formed using regular (infix) notation, rather than Polish or reverse Polish notation. Both DELTA and XDELTA ignore operators that trail the expression. Following is a typical expression (in hexadecimal):

```
G4A32+24
```

DELTA and XDELTA evaluate expressions from left to right. No operator takes precedence over any other.

DELTA and XDELTA recognize five binary arithmetic operators, one of which also acts as a unary operator. They are listed in Table DELTA–3.

Table DELTA–3 Arithmetic Operators

Operator	Action
+ or SPACE	Addition
-	Subtraction when used as a binary operator, or negation when used as a unary operator
*	Multiplication
%	Division
@	Arithmetic shift

The following example shows the arguments required by the arithmetic-shift operator:

```
n@j
```

In this example, *n* is the number to be shifted, and *j* is the number of bits to shift it. If *j* is positive, *n* is shifted to the left; if *j* is negative, *n* is shifted to the right. Argument *j* must be less than 20_{16} and greater than -20_{16}. Bits shifted beyond the limit of the longword are lost; therefore, the result must fit into a longword.

3 Guidelines for Debugging Programs with DELTA or XDELTA

When you use DELTA or XDELTA, there are no prompts, few symbols, and one error message. You move through program code by referring directly to address locations. This section provides guidelines for the following actions:

- Referencing addresses
- Referencing registers, the PSL, and the stack
- Interpreting the error message

For an example of a DELTA debug session, refer to Appendix A.

3.1 Referencing Addresses

When using DELTA or XDELTA to debug programs, you move through the code by referring to addresses. To help you identify address locations within your program, use a list file and a map file. The list file (.LIS) lists each instruction and its offset value from the base address of the program section. The full map file (.MAP) lists the base addresses for each section of your program. To determine the base address of a device driver program, refer to the *VMS Device Support Manual.*

Once you have the base addresses of the program sections, locate the instruction in the list file where you want to start the debug work. Add the offset from the list program to the base address from the map file. Remember that all calculations of address locations are done in hexadecimal. You can use DELTA/XDELTA to do the calculations for you with the = command.

To make address referencing easier, you can use offsets to a base address. Then you do not have to calculate all address locations. First, place the base address into a base register. Then move to a location using the offset to the base address stored in the register.

The following example of a simple MACRO program (EXAMPLE.MAR) and address referencing procedure shows how to reference addresses during a DELTA debug session. You can also use the same commands in an XDELTA debug session.

```
0000     1   .title  example
0000     2
0000     3   .entry  start   ^M<r3,r4>
0002     4           clrl    r3
0004     5           movl    #5,r4
0007     6   10$:    addl    r4,r3
000A     7           sobgtr  r4,10$
000D     8           ret
000E     9
000E    10   .end    start
```

The following procedure generates information to assist you with address referencing:

1 Use the /LIST qualifier to assemble the program and generate the list file.

To generate the list file for the preceding example, use the following command:

```
$ MACRO/LIST EXAMPLE
```

2 Use the /MAP qualifier with the link command to generate the full map file (.MAP file). Make sure that the default /DEBUG or /TRACEBACK qualifier is active for your link command. If not, specify /DEBUG or /TRACEBACK along with the /MAP qualifier.

To generate the map file for the example program, use the following command:

```
$ LINK/MAP EXAMPLE
```

3 Refer to the Program Section Synopsis of the map file. Locate the section that you want to debug. Look up the base address.

For the example program, the map file is EXAMPLE.MAP. A portion of the Program Section Synopsis is shown below. The first section of the program has a base address of 200.

```
                +--------------------------+
                ! Program Section Synopsis !
                +--------------------------+

Psect Name      Module Name   Base     End          Length
----------      -----------   ----     ---          ------

. BLANK .                   00000200 0000020D 0000000E (         14.)
                EXAMPLE     00000200 0000020D 0000000E (         14.)
```

4 Refer to the list file for the location of the specific instruction where you want to start debugging.

For the example program, start with the second instruction (MOVL #5,R4) with an offset of 4.

5 Enable DELTA using the following commands:

```
$ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
$ RUN/DEBUG EXAMPLE
```

6 If you want to store the base address in a base register, use the ;X command to load the base register.

For the example program, use the following DELTA/XDELTA command to store the base address 200 in base register 0.

```
200,0;X [RETURN]
```

7 Now you can move to specific address locations.

For example, if you wanted to place a breakpoint at the second instruction (MOVL #5,R4), you would calculate the address as 200 (base address) plus 4 (offset), or 204. Following is the DELTA/XDELTA command to set the breakpoint at address 204:

```
204;B [RETURN]
```

If you stored the base address in the base register, you could use the address expression X0+4 (or "X0 4", where the + sign is implied). Following is the command to set the breakpoint:

```
X0+4;B [RETURN]
```

Reverse this technique to find an instruction displayed by DELTA/XDELTA in the .LIS file, as follows:

1 Note the address of the instruction you want to locate in the .LIS file.

For example, DELTA/XDELTA displays the following instruction at address 020A:

```
20A!SOBGTR  R4,00000207
```

The following steps allow you to find the instruction at location 207.

2 Refer to the .MAP file, and identify the PSECT and MODULE where the address of the instruction is located. Check the base address value and the end address value of each PSECT and MODULE. When the instruction address is between the base and end address values, record the PSECT and MODULE names.

In the example, the instruction address is located in the EXAMPLE module (.BLANK. psect). The address instruction, 207, is between the base address 200 and the end address 20D.

3 Subtract the base address from the instruction address. Remember that all calculations are in hexadecimal and that you can use the DELTA/XDELTA = command to do the calculations. The result is the offset.

For the example, subtract the base address 200 from the instruction address 207. The offset is 7.

4 Refer to the .LIS file. Look up the MODULE, and then find the correct PSECT. Look for the offset value you calculated in the previous step.

In the example, there is only one PSECT and MODULE. Look up the instruction at offset 7. The program is branching to the following instruction:

```
10$:    addl r4,r3
```

3.2 Referencing Registers

To view the contents of the 16 general registers (including the program counter and the stack pointer) and the processor status longword (PSL), use the same DELTA/XDELTA commands as you use to view the contents of any memory location (for example, the /, LINEFEED, and the ESC commands). The symbols to use to identify the locations of the registers and PSL are as follows:

- The general registers are referred to by the symbol R and a hexadecimal number from 0_{16} to F_{16} representing the number of the register. For example, general register 1_{10} is $R1_{16}$, and general register 10_{10} is RA_{16}. The stack pointer is located in general register 14_{10}, RE_{16}. The program counter is in general register 15_{10}, RF_{16}.
- Upon entry to DELTA or XDELTA, the PSL is stored in the longword directly following the longword representing general register F_{16}. Reference it by using the general register F_{16} symbol plus a longword (RF+4).

3.3 Interpreting the Error Message

When you make an error entering a command in DELTA or XDELTA, you get the EH? error message. This is the only error message generated by DELTA or XDELTA. It is displayed if you enter an invalid command or reference an address that cannot be displayed.

3.4 Using XDELTA on Multiprocessing Systems

On multiprocessing systems, only one processor can use XDELTA at a time. If a second processor attempts to enter XDELTA when another processor has already entered it, the second processor waits until the first processor has exited XDELTA. If the processor using XDELTA sets a breakpoint, other processors are aware of the breakpoint. Therefore, when the code with the XDELTA breakpoint is executed on another processor, that processor will enter XDELTA and stop at the specified breakpoint.

When a breakpoint is taken by a processor in a multiprocessing environment, the processor's physical identification number is displayed on the XDELTA breakpoint message line as a 2-digit hexadecimal number. The following is an example of a breakpoint message in a multiprocessing environment:

```
1 BRK AT 00000400 ON CPU 03

00000400/MOVL #5,R4
```

XDELTA uses its own system control block (SCB) to direct all interrupt handling to an error handling routine in XDELTA. Therefore, an error encountered by XDELTA does not affect any other processors that share the standard system SCB.

DELTA/XDELTA Usage Summary

The Delta/XDelta Utility (DELTA/XDELTA) consists of two debuggers: DELTA and XDELTA. Use DELTA to debug user mode programs or programs that execute at interrupt priority level (IPL) 0. Use XDELTA to debug programs that execute at an elevated IPL, such as VMS executive routines, device drivers, and other privileged code.

usage summary

DELTA

To invoke DELTA, define DELTA as the default debugger, and run the program, as follows:

```
$ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
$ RUN/DEBUG program-name
```

To exit from DELTA, enter the EXIT command.

No privileges are required to run DELTA to debug a program that runs in user mode. To debug a program that runs in other processor-access modes, the process in which you link the program must have the necessary privileges.

To use the ;M command (Set All Processes Writable), your process must have change-mode-to-kernel (CMKRNL) privilege. To use the ;L command, (List All Loaded Executive Modules) you must have change-mode-to-executive (CMEXEC) privilege.

You cannot redirect output from a DELTA debug session.

XDELTA

To invoke XDELTA, bootstrap the system using a console command or command procedure that includes XDELTA. At the initial XDELTA breakpoint, set additional breakpoints, or examine and change locations in memory. Then proceed from the initial breakpoint. Refer to Section 1.3 for detailed steps on invoking XDELTA for each processor family.

To terminate an XDELTA debug session, reboot without XDELTA.

Because XDELTA is not process specific, privileges are not required. However, you must use the console terminal. You should run XDELTA only on a standalone system because all breakpoints are handled at IPL 31_{10}.

You cannot redirect output from XDELTA. Because you use XDELTA at a console terminal, you have a record of the debugging session. If your console terminal is a hardcopy terminal, you will have a written record; otherwise, output is typically stored in a console-maintained log file. MicroVAX systems do not maintain log files.

DELTA/XDELTA COMMANDS

This section describes how to use each DELTA and XDELTA command to debug a program.

Only DELTA uses the EXIT and ;M commands and arguments that specify a process identification. XDELTA defines some base registers that DELTA does not (refer to Section 2). With those exceptions, DELTA and XDELTA use the same commands. Differences are noted in command descriptions.

Enter the LINEFEED, ESC, TAB, and RETURN commands by pressing the corresponding key.

Each command includes an example. The program used for the examples is the same program listed in Appendix A.

Table DELTA–4 DELTA/XDELTA Command Summary

Command	Description
[	Set Display Mode
/	Open Location and Display Contents in Prevailing Width Mode
!	Open Location and Display Contents in Instruction Mode
LINEFEED	Close Current Location, Open Next
ESC	Open Location and Display Previous Location
TAB	Open Location and Display Indirect Location
"	Open Location and Display Contents in ASCII Mode
RETURN	Close Current Location
;B	Breakpoint
;P	Proceed from Breakpoint
;G	Go
S	Step Instruction
O	Step Instruction Over Subroutine
'string'	Deposit ASCII String
;E	Execute Command String
;X	Load Base Register
=	Display Value of Expression
;M	Set All Processes Writable (while running DELTA)
;L	Lists Names and Locations of Loaded Executive Images
EXIT	Exit from DELTA debugging session.

[(Set Display Mode)

Sets the width mode of displays produced by DELTA/XDELTA commands.

FORMAT

[*mode*

ARGUMENTS

mode

Specifies the display mode, as follows:

Mode	Meaning
B	Byte mode. Subsequent open and display location commands display the contents of one byte of memory.
L	Longword mode. Subsequent open and display location commands display the contents of a longword of memory. This is the default mode.
W	Word mode. Subsequent open and display location commands display the contents of one word of memory.

DESCRIPTION

The Set Display Mode command changes the prevailing display width to byte, word, or longword. The default display width is longword. The display mode remains in effect until you enter another Set Display Mode command.

EXAMPLE

```
R0/   00000001 ❶
[B ❷
R0/   01 ❸
```

❶ Contents of general register 0 (R0) are displayed using the / command. The display is the default mode, longword.

❷ Display mode is changed to byte mode using the [B command.

❸ Contents of R0 are displayed in byte mode. The least significant byte is displayed.

/ (Open Location and Display Contents in Prevailing Width Mode)

Opens a location and displays its contents in the prevailing display mode.

FORMAT

[pid:][start-addr-exp][,end-addr-exp]/ current-contents [new-exp]

ARGUMENTS

pid

The internal process identification (PID) of a process you want to access. If you specify zero or do not specify a PID, the default process is the current process. This argument cannot be used with XDELTA.

Subsequent open location and display contents commands entered after using the pid argument display the contents of the location of the specified process until you specify another PID with this command.

You can obtain the internal PID of processes by running the System Dump Analyzer Utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS/INDEX to determine the internal PID. Refer to the *VMS System Dump Analyzer Utility Manual* for more information about using SDA commands.

start-addr-exp

The address of the location to be opened, or the start of a range of addresses to be opened. If not specified, the address displayed is that currently specified by the symbol Q (last quantity displayed). Use the following syntax to display a single address location:

```
start-addr-exp/
```

You can also specify a register for this parameter. For example, if you want to view the contents of general register 3_{16} (R3), enter the following DELTA/XDELTA command:

```
R3/
```

end-addr-exp

The address of the last location to be opened. Use the following syntax to display a range of address locations:

```
start-addr-exp,end-addr-exp/
```

You can also specify a range of registers. For example, if you want to view the contents of general registers 3_{16} through 5_{16}, enter the following DELTA/XDELTA command:

```
R3,R5/
```

If you specify an address expression for **end-addr-exp** that is less than **start-addr-exp**, DELTA/XDELTA displays the contents of **start-addr-exp** only.

current-contents

You do not specify this parameter. It is a hexadecimal value, displayed by DELTA/XDELTA, of the contents of the location (or range of locations) you specified with the pid argument and the address expression. It is displayed in the prevailing width display mode.

new-exp

An expression, the value of which is deposited into the location just displayed. If you specify **new-exp** after a range of locations, the new value is placed only in the last location (specified by **end-addr-exp**).

When you specify **new-exp**, terminate the command by pressing RETURN.

If you want to deposit a new value into a location in another process (that is, you specified a PID other than the current process), you must have already set the target process to be writable using the ;M command.

If the value you deposit is longer than the last location where it will be deposited, the new value overwrites subsequent locations. For example, the values at address locations 202 and 204 are as follows:

```
202/ 05D053D4
204/ C05405D0
```

If you deposited the value FFFFFFFFF at address 202, the overflow value would overwrite the value stored at address location 204, as follows:

```
202/ 05D053D4  FFFFFFFFF RETURN
204/ C054FFFF
```

DESCRIPTION

The Open Location and Display Contents command opens the location or range of locations at **start-addr-exp** and displays **current-contents**, the contents of that location, in hexadecimal format. You can place a new value in the location by specifying an expression. A new value overwrites the last value displayed.

To display a range of locations, give the start-addr-exp argument as the first address in the range, followed by a comma, followed by the last address in the range (the end-addr-exp argument). For example, if you want to display all locations from 402 to 4F0, the command is as follows:

```
402,4F0/
```

This command changes the current address (. symbol) to the contents of the opened location. A subsequent Close Location command (RETURN) does not change the current address. However, a subsequent Close Current Location and Open Next command (ESC or LINEFEED) executes as follows:

- Writes any **new-exp** specified
- Closes the location opened by the / command
- Adds the number of bytes (defined by the prevailing display width mode) to the address just opened with the / command
- Changes the current address to the new value
- Opens the new location and displays the contents

The display mode remains hexadecimal until the next Open Location and Display Contents in Instruction Mode (!) command or Open Location and Display Contents in ASCII Mode (") command.

In DELTA, not XDELTA, you can examine the address space of any existing process, if your process has CMKRNL privilege. Use **pid** to specify the internal PID of the process you want to examine. For example, if you wanted to view address location 402 in the process with a PID of 20E00364, the command is as follows:

```
20E00364:402/
```

EXAMPLE

```
R0,RF/00000001
R1/00000000
R2/00000226
R3/7FF2AD94
R4/000019B4
R5/00000000
R6/7FF2AA49
R7/8001E4DD
R8/7FFED052
R9/7FFED25A
RA/7FFEDDD4
RB/7FFE33DC
RC/7FF2ADCC
RD/7FF2AD70
RE/7FF2AD68
RF/000006A7
```

Contents of all the general registers R0 through RF are displayed.

! (Open Location and Display Contents in Instruction Mode)

Displays an instruction and its operands.

FORMAT

[pid:][start-addr-exp][,end-addr-exp]!

ARGUMENTS

pid

The internal process identification (PID) of a process you want to access. If you specify zero, or do not specify any PID, the default process is the current process. This argument cannot be used with XDELTA.

Subsequent open location and display contents commands issued after using the pid argument display the contents of the location of the specified process until you specify another PID with this command.

You can obtain the internal PID of processes by running the System Dump Analyzer Utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS/INDEX to determine the internal PID. Refer to the *VMS System Dump Analyzer Utility Manual* for more information about using SDA commands.

start-addr-exp

The address of the instruction, or the first address of the range of instructions, to display. If you do not specify this parameter, the address displayed is that currently specified by Q (last quantity displayed). When you want to view just one location, the syntax is as follows:

```
start-addr-exp!
```

end-addr-exp

The address of the last instruction in the range to display. When you want to view several instructions, the syntax is as follows:

```
start-addr-exp,end-addr-exp!
```

Each location within the range is displayed with the address, a slash (/), and the MACRO instruction.

DESCRIPTION

The Open Location and Display Contents in Instruction Mode command displays the contents of a location or range of locations as a MACRO instruction. DELTA/XDELTA does not make any distinction between reasonable and unreasonable instructions or instruction streams.

This command does not allow you to modify the contents of the location. The command sets a flag that causes subsequent Close Current Location and Display Next (LINEFEED) and Open Location and Display Indirect Location (TAB) commands to display MACRO instructions. You can clear the flag by using the Open Location and Display Contents (/) command, which displays the contents of the location as a hexadecimal number, or Open Location and

Display Contents in ASCII Mode (″), which displays the contents of the location in ASCII.

When an address appears as an instruction's operand, DELTA/XDELTA sets the Q symbol to that address. Then enter ! again to go to the address specified in the instruction operand. DELTA/XDELTA changes Q only for operands that use program-counter or branch-displacement addressing modes; Q is not altered for operands that use literal and register addressing modes. This feature is useful for following branches.

EXAMPLE

```
69B!BRB     0000067A ❶
!CLRQ       -(SP) ❷
```

❶ The instruction at address 69B is displayed using the ! command. DELTA/XDELTA displays a branch instruction and sets Q (last address location displayed) to the branch address 67A.

❷ The instruction at address 67A is displayed using the ! command. The value of Q is used as the address location.

LINEFEED (Close Current Location, Open Next)

Closes the currently open location and opens the next location, displaying its contents.

FORMAT

LINEFEED

DESCRIPTION

The Close Current Location Open Next command closes the currently open location, then opens the next and displays its contents. This command accepts no arguments, and thus can only be used to open the next location. It is useful for examining a series of locations one after another. First, set the location where you want to start (for example, with the / or (!) command). Then, press LINEFEED repeatedly to examine each successive location.

The LINEFEED command displays the contents of the next location in the prevailing display mode and display width. If the current display mode is hexadecimal (the / command was used) and the display width is word, the next location displayed is calculated by adding a word to the current location. Its contents are displayed in hexadecimal. If the current display mode is instruction, the next location displayed is the next instruction, and the contents are displayed as a MACRO instruction.

On keyboards without a separate LINEFEED key, press CTRL/J. The LINEFEED key on LK201 keyboards (VT220, VT240, VT340, and workstation keyboards) generates different characters and cannot be used for the LINEFEED command. You must use CTRL/J.

This command is useful for displaying a series of MACRO instructions, a series of register values, or a series of values on the stack.

The values in the symbol Q and the symbol . are changed automatically.

EXAMPLE

```
6B9!CLRQ     -(SP) ❶  LINEFEED ❷
000006BB/CLRQ     -(SP)   LINEFEED
000006BD/PUSHL     X1+002E   LINEFEED
000006C1/PUSHAL     X1+003A   LINEFEED
000006C5/CLRQ     -(SP)   LINEFEED
000006C7/PUSHL     #00
```

❶ Instruction at address 6B9 is displayed using the ! command.

❷ Five successive instructions are displayed by pressing LINEFEED five times. LINEFEED is not echoed on the terminal.

ESC (Open Location and Display Previous Location)

Opens the previous location and displays its contents.

FORMAT

ESC

DESCRIPTION

The Open Location and Display Previous Location command decrements the location counter (.) by the width (in bytes) of the prevailing display mode, opens that many bytes, and displays the contents on a new line. The address of the location is displayed on the new line in the prevailing mode, followed by a slash (/) and the contents of that address.

This command is ignored if the prevailing display mode is instruction mode (set by the ! command).

Use this command to move backwards through a series of locations. Set the address where you want to start (for example, with the / command). Then press ESC repeatedly to display each preceding location. ESC is echoed as a dollar sign ($) on the terminal.

On keyboards without a separate ESC key, press CTRL/3. The ESC key on LK201 keyboards (VT220, VT240, VT340, and workstation keyboards) generates different characters and cannot be used for the ESC command. You must use CTRL/3.

EXAMPLE

```
R1/00000000 ❶  ESC  ❷
R0/00000001
```

❶ The contents of general register 1 are displayed using the / command.

❷ The contents of general register 0, the location prior to general register 1, are displayed by pressing ESC.

TAB (Open Location and Display Indirect Location)

Opens the location addressed by the contents of the current location and displays its contents.

FORMAT

TAB

DESCRIPTION

The Open Location and Display Indirect Location command opens the location addressed by the contents of the current location and displays the contents of the addressed location on a new line. The display is in the prevailing display mode. This command is useful for examining data structures that have been placed in a queue, or the operands of instructions.

To execute this command, press TAB.

This command changes the current address (.) to the location displayed.

This command does not affect the display mode.

EXAMPLE

```
69B!BRB     0000067A ❶ TAB
0000067A/CLRQ     -(SP) ❷
```

❶ The instruction at 69B is displayed using the ! command. DELTA/XDELTA displays a branch instruction.

❷ The instruction at the address referred to by the branch instruction is displayed by pressing TAB. DELTA/XDELTA displays the instruction at address 67A.

" (Open Location and Display Contents in ASCII)

Displays the contents of a location as an ASCII string.

FORMAT

[pid:] start-addr-exp[,end-addr-exp]"

ARGUMENTS

pid

The internal process identification (PID) of a process you want to access. If you specify zero, or do not specify any PID, the default process is the current process. This argument cannot be used with XDELTA.

Subsequent open location and display contents commands issued after using the pid argument display the contents of the location of the specified process until you specify another PID with this command.

You can obtain the internal PID of processes by running the System Dump Analyzer Utility (SDA). Use the SDA command SHOW SUMMARY to determine the external PID. Then use the SDA command SHOW PROCESS /INDEX to determine the internal PID. Refer to the *VMS System Dump Analyzer Utility Manual* for more information about using SDA commands.

start-addr-exp

The address of the location, or the start of a range of locations, to be displayed. If you want to view one location, the syntax is as follows:

```
start-add-exp"
```

end-addr-exp

The last address within a range of locations to be viewed. If you want to view a series of locations, the syntax is as follows:

```
start-add-exp,end-addr-exp"
```

DESCRIPTION

The Open Location and Display Contents in ASCII command opens the location or range of locations at **start-addr-exp** and displays the contents in ASCII format. This command does not change the width of the display (byte, word, longword) from the prevailing mode. If the prevailing mode is word mode, two ASCII characters are displayed; if byte mode, one character is displayed.

The display mode remains ASCII until you enter the next Open Location and Display Contents command (/) or Open Location and Display Contents in Instruction Mode command (!). These commands change the display mode to hexadecimal or instruction, respectively.

You can modify the contents of the locations, starting at **start-addr-exp**, with the Deposit ASCII string (') command.

EXAMPLE

```
235FC2 [W/415A  ❶
235FC2"ZA  ❷      LINEFEED  ❸
235FC4/PP
```

❶ The current display mode is word (displays one word in hexadecimal).

❷ The " command changes the prevailing display mode to ASCII but does not affect the width of the display.

❸ The next Close Current Location, Open Next command (LINEFEED) determines the address of the location to open by adding the width, in bytes, to the value contained in the symbol . (the current address). Then it opens the number of bytes equal to the width of the prevailing display mode, which in this example is two bytes.

The ASCII representation of the contents of the location presents the bytes left to right, while the hexadecimal representation presents them right to left.

RETURN (Close Current Location)

Closes a location that has been opened by one of the open location and display contents commands.

FORMAT

RETURN

DESCRIPTION

If you have opened a location with one of the open location and display contents commands (/, LINEFEED, ESC, !, or "), press RETURN to close the location. Use this command to make sure that a specific location has not been left open with the possibility of being overwritten.

You also press RETURN to terminate the following DELTA/XDELTA commands:

- ;X
- ;E
- ;G
- ;P
- ;B
- ;M
- 'string'
- ;L
- EXIT (DELTA only)

You can also use RETURN as an ASCII character in a quoted string. Refer to the Deposit ASCII String command (').

;B (Breakpoint)

Shows, sets, and clears breakpoints.

FORMAT

[addr-exp][,n][,display-addr-exp][,cmd-string-addr] **;B**

ARGUMENTS

addr-exp

The address where you want the breakpoint.

n

The number to assign to the breakpoint. For DELTA, the range is from 1 to 8. For XDELTA, the range is from 2 to 8. If omitted, DELTA/XDELTA assigns the first unused number to the breakpoint; if all numbers are in use, DELTA/XDELTA displays the error message, "EH?".

display-addr-exp

The address of a location, the contents of which are to be displayed in hexadecimal in the prevailing width mode when the breakpoint is encountered. If omitted, DELTA/XDELTA displays only the instruction that begins at the specified address.

cmd-string-addr

The address of the string of DELTA/XDELTA commands to execute when this breakpoint is encountered. Refer to the Execute Command String (;E) command. If omitted, DELTA/XDELTA executes no commands automatically and waits for you to enter commands interactively.

DESCRIPTION

The breakpoint command shows, sets, and clears breakpoints. The action of this command depends on the arguments used with it. Each action is described below.

Displaying Breakpoints

To show all the breakpoints currently set, enter ;B. For each breakpoint, DELTA/XDELTA displays the following information:

- Number of the breakpoint
- Address of the breakpoint
- Address of a location the contents of which will be displayed when the breakpoint is encountered
- Address of the command string associated with this breakpoint (for complex breakpoints, refer to the section in this Description on Setting Complex Breakpoints)

Setting Simple Breakpoints

To set a breakpoint, enter an address expression followed by ;B. Then press RETURN, as follows:

```
addr-exp;B RETURN
```

DELTA/XDELTA sets a breakpoint at the specified location and assigns it the first available breakpoint number.

When DELTA/XDELTA reaches the breakpoint, it completes the following actions:

- Suspends instruction execution.
- Sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as MACRO instructions.
- The following message is displayed, listing the number of the breakpoint, the address of the breakpoint, and the instruction stored at the breakpoint location:

  ```
  n BRK at address
  address/decoded-instruction
  ```

For XDELTA running in a multiprocessing environment, the number of the processor where the breakpoint was taken is also displayed as a 2-digit hexadecimal number.

After the breakpoint message is displayed, you can enter other DELTA/XDELTA commands. You can reset the flag that controls the mode in which instructions are displayed by entering the Open Location and Display Contents (/) command.

Setting a Breakpoint and Assigning a Number to It

To set a breakpoint and assign it a number, enter the address where you want the breakpoint, a comma, a single digit for the breakpoint number, a semicolon (;), the letter B, and then press RETURN. For DELTA, the breakpoint number can be from 1 through 8. For XDELTA, breakpoint 1 is reserved for INI$BRK. Therefore, the XDELTA breakpoint range is from 2 through 8.

For example, if you wanted to set breakpoint 4 at address 408, the command is as follows:

```
408,4;B RETURN
```

DELTA/XDELTA sets a breakpoint at the specified location and assigns it the specified breakpoint number.

Clearing Breakpoints

To clear a breakpoint, enter zero (0), followed by a comma, the number of the breakpoint to remove, a semicolon (;), the letter B, and then press RETURN. DELTA/XDELTA clears the specified breakpoint. For example, if you wanted to clear breakpoint 4, the command is as follows:

```
0,4;B RETURN
```

When using XDELTA, do not clear breakpoint 1. If you do, any calls to INI$BRK in your program will not result in entry into XDELTA.

Setting Complex Breakpoints

A complex breakpoint completes one or more of the following actions:

- Always displays the next instruction to be executed
- Optionally displays the contents of another, specified location
- Optionally executes a string of DELTA/XDELTA commands stored in memory

To use the complex breakpoint, you must first create the string of DELTA commands you want executed. Then deposit those commands at a memory location with the Deposit ASCII String command (').

To set a complex breakpoint, use the following syntax:

```
addr-exp,n,display-addr-exp,cmd-string-addr;B
```

The addr-exp argument is an expression whose value is the location at which the breakpoint is to be set.

The n argument is the number to assign to this breakpoint. The number of the breakpoint can range from 1 to 8 for DELTA and from 2 to 8 for XDELTA.

The display-addr-exp argument is an expression, the value of which is the address of a location whose contents are to be displayed when this breakpoint is encountered. Omit this argument by specifying zero or two consecutive commas.

The cmd-string-addr argument is an expression, the value of which is the address of the string of DELTA/XDELTA commands to be executed when this breakpoint is encountered. DELTA/XDELTA displays the information requested before executing the string of commands associated with complex breakpoints. You must have previously deposited the string of commands using the ' command or have coded the string into an identifiable location in your program.

EXAMPLE

```
;B
1  00000690
2  00000699 ❶
0,2;B ❷
;B
1  00000690 ❸
;P ❹
1 BRK AT 00000690
00000690/CMPL  R0,#000009A8 ❺
```

❶ Two breakpoints have already been set and are displayed. Using ;B, DELTA/XDELTA displays each breakpoint number and the address location of each breakpoint.

❷ Breakpoint 2 is cleared.

❸ Current breakpoints are displayed. Because breakpoint 2 has been cleared, DELTA/XDELTA displays just breakpoint 1.

❹ Program execution is continued using the ;P command.

❺ At breakpoint 1, program execution halts. DELTA/XDELTA displays the breakpoint message (the breakpoint number 1 and the address 690) and the instruction.

;P (Proceed from Breakpoint)

Continue program execution following a breakpoint.

FORMAT ;P

DESCRIPTION

The Proceed from Breakpoint command continues program execution at the address contained in the PC of the program. Program execution continues until the next breakpoint or until program completion.

EXAMPLE

```
;B
2  00000699 ❶
;P ❷
2 BRK AT 00000699
00000699/BSBB     000006A2 ❸
```

❶ Current breakpoints are displayed using ;B (breakpoint 2 at address 699).

❷ Program execution is continued using the ;P command.

❸ Program execution halts at breakpoint 2. DELTA/XDELTA displays the breakpoint message (the breakpoint number and the address) and the instruction.

;G (GO)

Continues program execution.

FORMAT

address-expression **;G**

PARAMETERS

address-expression
The address at which to continue program execution.

DESCRIPTION

The GO command places the address you specified in **address-expression** into the PC and continues execution of the program at that address. It is useful when you want to skip over specific lines of code or return to a previous program location to repeat execution.

EXAMPLE

```
6A2;G
```

Program execution is started at address 6A2.

S (Step Instruction)

Executes one instruction and displays the next. If the executed instruction is a call to a subroutine, it steps into the subroutine and displays the next instruction to be executed in the subroutine.

FORMAT

S

DESCRIPTION

The Step Instruction command executes one instruction and displays the next instruction (in instruction mode) and its address. Use this command to do single-step instruction execution, including single-step of all instructions in subroutines. If you want to do single-step instruction execution excluding single-step of instructions in subroutines, use the O command.

The instruction displayed has not yet been executed. This command sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as MACRO instructions. The Open Location and Display Contents (/) command clears the flag, causing the display mode to revert to longword, hexadecimal mode.

If the instruction being executed is a BSBB, BSBW, JSB, CALLG, or CALLS instruction, Step moves to the subroutine called by these instructions and displays the first instruction within the subroutine.

In general, you would move to the instruction where you want to start single-step execution with the GO (;G) command. Then press S to execute the first instruction and display the next one.

EXAMPLE

```
00000690/CMPL   R0,#000009A8    S ❶
00000697/BEQL   0000069D        S ❷
00000699/BSBB   000006A2        S ❸
000006A2/PUSHL  R2 ❹
```

❶ Step program execution is started at address 690. The instruction at 690 is executed, and the next instruction is displayed. Step execution is continued using S.

❷ At address 697, there is a branch instruction to the instruction at address 69D. However, because the condition (BEQL) is not met, program execution continues at the next instruction. The next S command is executed.

❸ At address 699, there is a branch instruction to the instruction at address 6A2, a subroutine. The next S command is executed.

❹ Program execution moves to the subroutine.

O (Step Instruction Over Subroutine)

Executes one instruction, steps over a subroutine by executing it, and displays the instruction to which the subroutine returns control.

FORMAT

O

DESCRIPTION

The Step Instruction Over Subroutine command executes one instruction and displays the address of the next instruction. If the instruction executed is a call to a subroutine, the subroutine is executed, and the next instruction displayed is the instruction to which the subroutine returns control. Use this command to do single-step instruction execution excluding single-step of instructions within subroutines. If you want to do single-step execution of all instructions, including those in subroutines, use the S command.

This command sets a flag to change the display mode to instruction mode. Any subsequent Close Current Location, Open Next (LINEFEED) commands and Open and Display Indirect Location (TAB) commands will display locations as MACRO instructions. The Open Location and Display Contents (/) command clears the flag, causing the display mode to revert to longword, hexadecimal mode.

If the executed instruction is BSBB, BSBW, JSB, CALLG, or CALLS, DELTA/XDELTA executes the subroutine called by these instructions and displays the instruction to which the subroutine returns control.

If you set a breakpoint in the subroutine and enter the O command, program execution breaks at the subroutine breakpoint. When you enter a Proceed command (;P), and program execution returns to the instruction to which the subroutine returns control, a message is displayed, as follows:

```
STEPOVER BRK AT nnnnnnnn

instruction
```

The message informs you that program execution has returned from a subroutine.

If you are using XDELTA in a multiprocessing environment, the physical identification number of the processor where the break was taken is also displayed as a 2-digit hexadecimal number.

EXAMPLE

```
6D5;B ❶
;P ❷
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00              ;P ❸
        PID=  0006      LOGINTIME=  12:50:29.45
2 BRK AT 00000699
00000699/BSBB    000006A2        ;P ❹
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00              ;P ❺
        PID=  0007      LOGINTIME=  12:50:37.08
2 BRK AT 00000699
00000699/BSBB    000006A2        O ❻
1 BRK AT 000006D5
000006D5/CALLS    #0C,@#7FFEDE00              ;P ❼
        PID=  0008      LOGINTIME=   12:50:45.64
STEPOVER BRK AT 0000069B ❽
0000069B/BRB    X1+047A
```

❶ One breakpoint has been set at address 699 in the main routine. A simple breakpoint is set at 6D5 using ;B. This breakpoint is in a subroutine.

❷ Program execution continues using ;P.

❸ Program execution stops at breakpoint 1, which is in the subroutine. DELTA/XDELTA displays the breakpoint message and the instruction at the new breakpoint. Program execution continues using ;P.

❹ The subroutine completes and displays some output. Program execution continues until breakpoint 2. DELTA/XDELTA displays the breakpoint message and the breakpoint 2 instruction. Program execution continues with the ;P command.

❺ Program execution stops at breakpoint 1. Program execution continues with the ;P command. The subroutine completes execution and displays the output.

❻ Program execution stops at breakpoint 2. The subroutine is stepped over to the next instruction using the O command.

❼ Program execution stops at breakpoint 1 in the subroutine. Program execution continues using the ;P command.

❽ The subroutine completes execution and displays output. DELTA/XDELTA displays a STEPOVER break message that states the O command has been completed, returning control at address 69B.

' (Deposit ASCII String)

Deposits the ASCII string at the current address.

FORMAT

'string'

ARGUMENTS

string
The string of characters to be deposited.

DESCRIPTION

The Deposit ASCII String command deposits **string** at the current location (.) in ASCII format. The second apostrophe is required to terminate the string. All characters typed between the first and second apostrophes are entered as ASCII character text. Avoid embedding an apostrophe (') within the string you want to deposit.

When you want to use key commands (LINEFEED, RETURN, ESC, TAB), press the key. These commands are entered as text.

This command stores the characters in eight-bit bytes and increments the current address (.) by one for each character stored.

This command does not change the prevailing display mode.

EXAMPLE

```
7FFE1600/'R0/ LINEFEED LINEFEED'
```

The ASCII string "R0/LINEFEED LINEFEED" is stored at address 7FFE1600. This string, if subsequently executed with the ;E command, examines the contents of general register 0 (the command R0/), then examines two subsequent registers (using two LINEFEED commands).

;E (Execute Command String)

Executes a string of DELTA/XDELTA commands stored in memory.

FORMAT

address-expression;E

ARGUMENTS

address-expression
The address of the string of DELTA/XDELTA commands to execute.

DESCRIPTION

The Execute Command String command executes a string of DELTA/XDELTA commands. Load the ASCII text command string to a specific location in memory using the Deposit ASCII String command ('), or code the string in your program into an identifiable location.

If you want DELTA/XDELTA to proceed with program execution after it executes the string of commands, end the command string with the ;P command. If you want DELTA/XDELTA to wait for you to enter a command after it executes the string of commands, end the command string with a null byte (a byte containing 0).

XDELTA, but not DELTA, provides two command strings in memory. The addresses of these command strings are stored in base registers XE and XF. The string addressed by XE displays the physical page number (PFN) database for the PFN in X0. The string addressed by XF copies the PFN in R0 to base register X0, then displays the PFN database for that PFN.

You can use the command strings provided with XDELTA to obtain the following information:

- Specified physical page number (PFN)
- PFN state
- PFN type
- PFN reference count
- PFN backward link or working-set-list index
- PFN forward link or share count
- Page table entry (PTE) that points to the PFN
- PFN backing-store address
- Virtual block number in the process swap image, the block to which the page's entry in the SWPVBN array points

EXAMPLE

```
7FFE1600,0;X ❶
7FFE1600 ❷
X0;E ❸
R0/00000001 ❹
R1/00000000
R2/00000000
```

❶ The address (7FFE1600) where an ASCII string is stored is placed into base register 0 using ;X.

❷ DELTA/XDELTA displays the value in X0.

❸ The command string stored at address 7FFE1600, which is to examine the contents of R0, R1, and R2 (R0/LINEFEED LINEFEED), is executed with ;E.

❹ DELTA/XDELTA executes the commands and displays the contents of R0, R1, and R2.

;X (Load Base Register)

Places an address in a base register.

FORMAT

address-expression,n;**X**

ARGUMENTS

address-expression
The address to place in the base register.

n
The number of the base register.

DESCRIPTION

To place an address in a base register, enter an expression followed by a comma (,), a number from 0 to F_{16}, a semicolon (;), and the letter X. DELTA/XDELTA places the address in the base register. DELTA/XDELTA confirms that the base register is set by displaying the value deposited in the base register.

For example, the following command places the address 402 in base register 0. DELTA/XDELTA then displays the value in the base register to verify it.

```
402,0;X RETURN
00000402
```

When DELTA/XDELTA displays an address that is within 2000_{16} bytes of an address stored in a base register, DELTA/XDELTA displays the base register identifier (Xn), followed by an offset that gives the address location in relation to the address stored in the base register. If the address falls outside this range, DELTA/XDELTA displays it as a hexadecimal value.

For example, if base register 2 contains 800D046A, and the address DELTA/XDELTA would display is 800D052E (offset of C4), DELTA/XDELTA displays that address as X2+C4. DELTA/XDELTA computes relative addresses for both opened and displayed locations and for addresses that are instruction operands.

DELTA/XDELTA provides symbols that make it easy to refer to processor (Pn), general (Rn), and base registers (Xn). Refer to Section 2.1 for a description of these symbols.

EXAMPLE

```
00000664/CLRQ     -(SP)     200,1;X ❶
00000200 ❷

X1 490!CMPL       R0,#000009A8 ❸
X1 499!BSBB       X1+04A2 ❹
```

❶ The base address of the program (determined from the map file) is virtual address 200. The base address is stored in base register 1 with ;X.

❷ DELTA/XDELTA displays the value in base register 1 just loaded, 200.

❸ The instruction at offset 490 is displayed in instruction mode using the ! command. The address reference is X1+490 (the + sign is implied when not specified). DELTA/XDELTA displays the instruction at address X1+490.

❹ The instruction at offset 499 is displayed. This instruction is a branch instruction. DELTA/XDELTA displays the address of the branch in offset notation.

= (Display Value of Expression)

Evaluates an expression and displays its value.

FORMAT

expression =

ARGUMENT

expression
The expression to be evaluated.

DESCRIPTION

The Display Value of Expression command evaluates an expression and displays its value in hexadecimal. The expression can be any valid DELTA/XDELTA expression. See Section 2.1 for a description of DELTA /XDELTA expressions.

All calculations and displays are in hexadecimal in the prevailing length mode.

EXAMPLE

```
FF+1=00000100 ❶
A-1=00000009 ❷
```

❶ FF_{16} and 1_{16} are added together. DELTA/XDELTA displays the sum in hexadecimal.

❷ 1_{16} is subtracted from A_{16}. DELTA/XDELTA displays the result in hexadecimal.

;M (Set All Processes Writable)

Sets the address spaces of all processes to be writable or read-only by your DELTA process. This command can be used only with DELTA. Use of this command requires CMKRNL privilege.

FORMAT

n;M

ARGUMENT

n

Specifies your process privileges for reading and writing at other processes. If 0, your DELTA process can only read locations in other processes; if 1, your process can read or write any location in any process. If not specified, DELTA returns the current value of the M (modify) flag (0 or 1).

DESCRIPTION

The Set All Processes Writable command is useful for changing values in the running system.

NOTE: **This is an activity that must be used very carefully during timesharing. For this reason, your process must have change-mode-to-kernel (CMKRNL) privilege to use this command. It is safest to use this command only on a standalone system.**

;L (List Names and Locations of Loaded Executive Images)

List the names and virtual addresses of all loaded executive images.

FORMAT

;L

DESCRIPTION

Use the ;L command when you are debugging code that resides in system space. Although you use this command mostly with XDELTA, you can use it with DELTA if your process has change-mode-to-executive (CMEXEC) privilege, and you are running a program in executive mode.

This command lists the names and locations of the loaded modules of the executive. A loading mechanism maps a number of images of the executive into system space. The ;L command lists the currently loaded images with their starting and ending virtual addresses. If you enter ;L before all the executive image are loaded (for example, at an XDELTA initial breakpoint), only those images which have been loaded will be displayed.

EXAMPLE

```
;L
PRIMITIVE_IO.EXE              800EAA00  800EBC00
SYSTEM_SYNCHRONIZATION.EXE    800EBC00  800ED400
SYSTEM_PRIMITIVES.EXE         800ED400  800F1000
```

The starting and ending virtual addresses of three loaded executive images are listed.

EXIT (Exit From DELTA Debugging Session)

Terminates the DELTA debugging session. Use with DELTA only.

FORMAT

EXIT

DESCRIPTION

Use the EXIT command to terminate a DELTA debugging session. You cannot use EXIT in XDELTA.

You may have to enter EXIT twice, such as when your program terminates execution via the $EXIT system service or via RET (to DCL).

A Sample DELTA Debug Session

This appendix gives an example of using DELTA. The program, LOGINTIM, uses the system service SYS$GETJPI to obtain the login times of each process. Although this is an example of using DELTA, most of the commands in the example could be used in a XDELTA debug session.

To run this program without error, you need WORLD privilege.

The .LIS file is listed below. Only the offsets and source code are shown.

```
0000     1 ;++
0000     2 ; This sample program uses the wildcard feature of GETJPI to get the
0000     3 ; LOGINTIM for each active process.  It outputs the PID and LOGINTIM
0000     4 ; for each and exits when there are NOMOREPROCs.
0000     5 ;--
0000     6
0000     7 ;
0000     8 ; Data areas.
0000     9 ;
0000    10 DEVNAM: .ASCID /SYS$OUTPUT/          ;Output device specifier
000E
0012    11
0012    12 CHAN:   .LONG  0                     ;Assigned output channel
0016    13
0016    14 ITMLST:                              ;Item list for GETJPI call
0016    15         .WORD  8                     ; Byte length of output buffer
0018    16         .WORD  JPI$_LOGINTIM         ; Specify LOGINTIM item code
001A    17         .ADDRESS      TIME           ; Address of output buffer
001E    18         .LONG  0                     ; Not interested in return length
0022    19         .LONG  0                     ;Item list terminator
0026    20
0026    21 TIME:   .QUAD  0                     ;Buffer to hold LOGINTIM
002E    22
002E    23 OUTLEN: .LONG  0                     ;FAO buffer length
0032    24 OUTBUF: .LONG  1024                  ;FAO buffer descriptor
0036    25         .ADDRESS BUF
003A    26 BUF:    .BLKB  1024                  ;FAO buffer
043A    27
043A    28 CTRSTR: .ASCID *!/!_PID= !XW!_LOGINTIME= !%T* ;FAO control string
0448
0454
045E    29
045E    30 PIDADR: .LONG  -1                    ;Wildcard PID control longword
0462    31
0462    32 ;++
0462    33 ; Start of program.
0462    34 ;--
0462    35 S:       .WORD   0                   ;Entry mask
0464    36          $ASSIGN_S   DEVNAM,CHAN     ;Assign output channel
0475    37          MOVAB   TIME,R2             ;Load pointer to LOGINTIM
047A    38                                      ;     output buffer
047A    39 LOOP:    $GETJPI_S   ITMLST=ITMLST,-;Get LOGINTIM for a process
047A    40                  PIDADR=PIDADR
0490    41          CMPL    R0,#SS$_NOMOREPROC ;Are we done?
0497    42          BEQL    5$                 ;If EQL yes
0499    43          BSBB    GOT_IT             ;Process data for this process
049B    44          BRB     LOOP               ;Look for another process
```

Sample DELTA Debug Session

```
049D     45
049D     46 5$:      MOVZBL  #SS$_NORMAL,R0      ;Set successful completion code
04A1     47          RET                        ;Return, no more processes
04A2     48
04A2     49 GOT_IT:  $FAO_S  CTRSTR,-           ;Format the output data
04A2     50                  OUTLEN,-
04A2     51                  OUTBUF,-
04A2     52                  PIDADR,R2
04B9     53          $QIOW_S CHAN=CHAN,-        ;Output to SYS$OUTPUT
04B9     54                  FUNC=#IO$_WRITEVBLK,-
04B9     55                       P1=BUF,-
04B9     56                       P2=OUTLEN
04DC     57           RSB                       ;Done with this process data
04DD     58
04DD     59           .END S
```

The .MAP file is listed below. Only the Program Section Synopsis with the PSECT, MODULE, base address, end address, and length are listed.

```
                        +--------------------------+
                        ! Program Section Synopsis !
                        +--------------------------+

Psect Name      Module Name       Base     End          Length
----------      -----------       ----     ---          ------
. BLANK .                      00000200 000006E2 000004E3 (       1251.)
                .MAIN.         00000200 000006E2 000004E3 (       1251.)
```

The DELTA debug session is listed in the following example:

```
$ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA ❶
$ RUN/DEBUG LOGINTIM ❷
DELTA Version 5.0

00000664/CLRQ    -(SP)    200,1;X ❸
00000200 ❹

X1 490!CMPL    R0,#000009A8    .;B ❺
X1 499!BSBB    X1+04A2         .;B ❻
;P ❼

1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8  R0/00000001 ;P❽
2 BRK AT 00000699
X1+499/BSBB    X1+04A2       0❾
        PID=  0000      LOGINTIME=  00:00:00.00 ❿
```

```
X1+049B/BRB     X1+047A         ;P ⓫
1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8  R0/00000001 ;P ⓬
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       0 ⓭
        PID=  0001      LOGINTIME=  00:00:00.00
X1+049B/BRB     X1+047A         ;P
1 BRK AT 00000690
X1+0490/CMPL    R0,#000009A8 ⓮
;B ⓯
1  00000690
2  00000699 ⓰
0,1;B ⓱
;B ⓲
2  00000699 ⓳
;P ⓴
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       0
        PID=  0004      LOGINTIME=  12:50:20.40
X1+049B/BRB     X1+047A         ;P㉑
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       ;P
        PID=  0005      LOGINTIME=  12:50:25.61 ㉒
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       X1 4B9!CLRQ    -(SP)㉓
[LINEFEED] ㉔
X1+04BB/CLRQ    -(SP)   [LINEFEED]
X1+04BD/PUSHL    X1+002E  [LINEFEED]
X1+04C1/PUSHAL    X1+003A  [LINEFEED]
X1+04C5/CLRQ    -(SP)   [LINEFEED]
X1+04C7/PUSHL    #00   [LINEFEED]
X1+04C9/MOVZWL    #0030;-(SP)   [LINEFEED]
X1+04CE/MOVZWL    X1+0012,-(SP)   [LINEFEED]
X1+04D3/PUSHL     #00   [LINEFEED]
X1+04D5/CALLS    #0C,@#7FFEDE00                .;B ㉕
;B ㉖
1  000006D5
2  00000699
;P ㉗
1 BRK AT 000006D5
X1+04D5/CALLS    #0C,@#7FFEDE00               ;P ㉘
        PID=  0006      LOGINTIME=  12:50:29.45
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       ;P ㉙
1 BRK AT 000006D5
X1+04D5/CALLS    #0C,@#7FFEDE00               ;P ㉚
        PID=  0007      LOGINTIME=  12:50:37.08
2 BRK AT 00000699
X1+0499/BSBB    X1+04A2       0 ㉛
1 BRK AT 000006D5
```

Sample DELTA Debug Session

```
X1+04D5/CALLS    #0C,@#7FFEDE00                ;P ❷
        PID=  0008      LOGINTIME=  12:50:45.64
STEPOVER BRK AT 0000069B ❸
X1+049B/BRB    X1+047A                         ;B ❹
1  000006D5
2  00000699 ❺
0,2;B ❻
0,1;B ❼
;B ❽
;P ❾
        PID=  0009      LOGINTIME=  12:51:22.51
        PID=  000A      LOGINTIME=  12:51:30.26
        PID=  000B      LOGINTIME=  12:51:36.21
        PID=  000C      LOGINTIME=  12:51:58.86 ❿
EXIT 00000001 ⓫
80187E7E/POPR    #03                          EXIT ⓬
```

❶ DELTA is enabled as the debugger.

❷ The example program LOGINTIM is invoked with DELTA.

❸ DELTA displays a version number and the first executable instruction. The base address of the program (determined from the map file) is virtual address 200. The base address is placed in base register 1 with ;X. Now references to an address can use the address offset notation. For example, a reference to the first instruction is X1+464 (or base address 200 + offset 464). Also, DELTA displays some address locations as offsets to the base address.

❹ DELTA displays the value in base register 1 just loaded 200.

❺ The instruction at address 690 is displayed in instruction mode using !. Its address location is expressed as the base address plus an offset. In the listing file, the offset is 490. The base address in base register X1 is 200. The address reference, then, is X1+490. (NOTE: The + sign is implied when not specified.)

A simple breakpoint is set at that address using the ;B command. The address reference for ;B is the . symbol, representing the current address. X1+490;B would have done the same thing.

❻ The same commands (! command to view the instruction and ;B to set a breakpoint) are repeated for the instruction at offset 499. When DELTA displays the instruction (BSBB GOT_IT), it displays the destination of the branch (GOT_IT) as the address location. DELTA displays the value as an offset to base register 1.

❼ Program execution is begun using ;P.

❽ Program execution halts at the first breakpoint. DELTA displays the breakpoint message (1 BRK AT 00000690) with the breakpoint number 1 and the virtual address. The virtual address is 00000690, which is the base address (200) plus the offset 490. DELTA then displays the instruction in instruction mode (CMPL R0,#000009A8). The contents of general register 0 are displayed with the / command. DELTA displays the contents of R0, which is 1. Program execution continues using the ;P command.

❾ Program execution halts at breakpoint 2. DELTA displays the breakpoint message, then the instruction. Step-instruction execution, excluding instructions in subroutines, is initiated with O.

⑩ The subroutine GOT_IT is executed, and the output (PID and login time) is displayed.

⑪ The O command halts program execution at the instruction where the subroutine returns control (BRB LOOP). DELTA displays the instruction in instruction mode (BRB X1+047A), where X1+047A is the address of the first instruction in LOOP. Program execution continues with ;P.

⑫ Breakpoint 1 is encountered again, DELTA displays the breakpoint message and the instruction. The contents of R0 are examined (/ command) and program execution continues (;P).

⑬ Breakpoint 2 is encountered again, DELTA displays the breakpoint message and the instruction. The subroutine is stepped over again with the O command. The subroutine is executed, and the output is displayed. The instruction where the subroutine returns control is displayed. Program execution continues (;P command).

⑭ Breakpoint 1 is encountered; DELTA displays the breakpoint message and the instruction.

⑮ All breakpoints in the program are listed with the ;B command.

⑯ DELTA displays the breakpoints (by breakpoint number) and the address locations.

⑰ Breakpoint 1 is cleared using 0,[breakpoint #];B. (Never clear breakpoint 1 in XDELTA.)

⑱ All breakpoints are listed again with ;B command.

⑲ DELTA displays breakpoint 2 (breakpoint 1 cleared).

⑳ Program execution continues using the ;P command.

㉑ Breakpoint 2 is encountered. DELTA displays the breakpoint message, and the instruction. The subroutine is executed with the O command and the subroutine output is displayed. The next instruction where the subroutine returns control is displayed. Program execution continues with the ;P command.

㉒ Breakpoint 2 is encountered. DELTA displays the breakpoint message and the instruction. Program execution continues to the next breakpoint with the ;P command. The subroutine is executed, and the subroutine output is displayed.

㉓ Breakpoint 2 is encountered again. The instruction at offset 4B9 (in the subroutine) is displayed using !. This instruction is part of the set-up for the call to the system service QIOW.

㉔ Successive address locations are displayed by pressing LINEFEED nine times. These instructions are the remainder of the set up and the call to the system service QIOW.

㉕ A breakpoint at X1+04D5 (the current address) is set using the ;B command. This breakpoint is in the subroutine. The . symbol represents the current address.

㉖ The current breakpoints in the program are listed. The new breakpoint is assigned breakpoint 1.

㉗ Program execution continues with the ;P command.

㉘ Program execution stops at the new breakpoint 1, which is in the subroutine GOT_IT. DELTA displays the breakpoint message and the instruction at the new breakpoint. Program execution continues with the ;P command.

㉙ The subroutine completes and displays the output, and program execution continues until breakpoint 2. DELTA displays the breakpoint message and the breakpoint 2 instruction. Program execution continues with the ;P command.

㉚ Program execution stops at breakpoint 1 in the subroutine. Program execution continues with the ;P command. The subroutine is executed, and the output is displayed.

㉛ Program execution stops at breakpoint 2. The O command is entered to execute and step over the subroutine.

㉜ Program execution stops at breakpoint 1 in the subroutine. Program execution continues with the ;P command.

㉝ The subroutine completes execution and displays output. DELTA displays a STEPOVER break message to state that the O command has been completed, returning control at address 69B (an instruction in the main routine).

㉞ The instruction where the subroutine returns is displayed, and program execution is halted. The ;B command is entered to display all current breakpoints.

㉟ The two current breakpoints are listed.

㊱ The command 0,2;B clears breakpoint 2.

㊲ The command 0,1;B clears breakpoint 1.

㊳ The ;B command is entered to display all current breakpoints. Because all breakpoints have been cleared, DELTA does not display any.

㊴ Program execution continues with the ;P command. Because there are no longer any breakpoints, the program executes to the end.

㊵ All current process login times are displayed.

㊶ Final exit status is displayed.

㊷ The DELTA EXIT command is entered to terminate the debugging session and leave DELTA.

Index

A

B

C

D

E

G

H

I

L

M

N

O

P

Q

R

S

T

V

X

Reader's Comments

VMS Delta/XDelta Utility Manual
AA–LA86A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------