Alpha Microprocessors SROM Mini-Debugger

User's Guide

Order Number: EK–AMSMD–UG. A01

Revision/Update Information: This is a new manual.

Digital Equipment Corporation
Maynard, Massachusetts

April 1996

Contents

3 SROM Mini-Debugger Command Set

Index

Tables

Preface

Introduction

This document describes how to use the Alpha Microprocessors SROM Mini-Debugger (also referred to as the mini-debugger) to debug hardware with one of the following Alpha motherboards:

- The Alpha PCI 64-275
- The Alpha PCI 164-266/300

Audience

This document is intended for anyone who develops hardware or software to be used with an Alpha microprocessor.

Content Overview

The information in this document is organized as follows:

- Chapter 1 is a general overview of the mini-debugger.
- Chapter 2 describes how to set up and start the mini-debugger.
- Chapter 3 describes the mini-debugger command set.

Conventions

The following conventions are used in this document:

Convention	Meaning
A percent sign (%)	Indicates a Digital UNIX operating system command prompt.
SROM>	Indicates an Alpha Microprocessors SROM Mini-Debugger prompt.
Boldface type	Indicates an Alpha Microprocessors SROM Mini-Debugger command keyword.
Italic type	Indicates special emphasis or the title of a manual.
`Monospaced type`	Indicates an operating system command, a file name, or a directory path name.
Note	Provides general information.

1
Introduction

1.1 Overview

The Alpha Microprocessors SROM Mini-Debugger provides basic hardware debugging capability through the SROM serial port of the Alpha microprocessor. Using only an SROM containing the mini-debugger, a clock source, a CPU chip, and a few gates, you can exercise the device connected to the CPU to debug cache, memory, and I/O subsystems until the board is functional enough to support a more fully featured monitor.

1.2 General Features

The mini-debugger has the following features:

- Basic hardware debugging capability
- A monitor that can point to hardware addresses and exercise them
- The ability to examine and deposit memory
- A case-independent command language
- Support for variable baud rates and processor speeds

2
Getting Started

2.1 Overview

The mini-debugger is available in the standard SROM provided with the Alpha motherboards. It can be invoked *after* the standard SROM has completed CPU and system initialization and before it begins execution of the image loaded from ROM. Refer to the Design Guide for the Alpha PCI 64-275 or the Alpha PCI 164-266/300 for more information about how to access the mini-debugger. (It requires setting a jumper.)

2.2 Hardware Required

To run the mini-debugger, you need the following:

- An Alpha motherboard or a system based on the Alpha microprocessor architecture with a connection from the microprocessor SROM interface to the SROM serial port (for example, an RS232)
- A host system (a terminal or workstation)

Note

The recommended host system is an Alpha platform running the Windows NT or Digital UNIX operating system.

2.3 Hardware Debug Features

The mini-debugger resides in the instruction cache and is designed to be loaded at reset through the SROM interface. The mini-debugger provides commands to

- Examine and deposit data in memory.
- Examine and deposit internal CPU registers.
- Transfer execution to any address in the board's memory range.

The mini-debugger's primary purpose is to debug hardware so that the memory interface works, thus allowing a more complex debugger such as the Alpha Microprocessors Debug Monitor or the full SRM test software to be loaded to debug other parts of the system or software.

2.4 Setting Up the SROM Serial Port Connection

To use the mini-debugger, you must first establish a connection from your Alpha microprocessor system or motherboard to the serial port on your host system or terminal. This section describes how to connect the SROM serial port of an motherboard to the following hardware:

- Terminal
- DEC 2000 Model 300 AXP system running the Windows NT operating system
- DEC 3000 Model 500 AXP system running the OSF/1 operating system

2.4.1 Connecting the Board to a Terminal

To connect a board to a terminal, connect the SROM serial port of the board to the terminal communication line. Auto baud detection is enabled by typing an uppercase U. The recommended baud rate is 9600.

2.4.2 Connecting to the Motherboard from a DEC 2000 Model 300 AXP System

The DEC 2000 Model 300 AXP system running the Windows NT operating system supports an SROM serial port. To enable the SROM serial port for use with the motherboard, follow these steps:

1. Select the `Program Manager` icon.
2. Select the `Accessories` icon.
3. Select the `Terminal` icon.
4. Set the following terminal characteristics:

Terminal Setting	Value
Terminal mode	VT*nnn*—7 bit
Transmit/receive speed	9600 baud
Character format	No parity
Stop bits	1

2.4.3 Connecting to the Motherboard from a DEC 3000 Model 500 AXP System

The DEC 3000 Model 500 AXP system supports serial communications and Ethernet communications.

The DEC 3000 Model 500 AXP system running the Digital UNIX operating system supports serial communication through two ports:

- /dev/tty00—Synchronous/asynchronous communication port (25-pin RS232 connector)
- /dev/tty01—Alternate console printer (modular jack)

Either port can be connected to the SROM serial port. For consistency, all examples and command descriptions assume that serial port 1 is connected to port `/dev/tty00`.

To enable these ports for use with the motherboard, follow these steps:

1. Modify the following two files:

   /etc/remote
   /etc/inittab

   a. Add the following two lines to the /etc/remote file. These lines define a device to connect to when using the Digital UNIX tip command.

      ```
      port_name0:dv=/dev/tty00:br#9600:pa=none:
      port_name1:dv=/dev/tty01:br#9600:pa=none:
      ```

      The *port_name* refers to the name that you assigned to the port on the Digital UNIX host. For example, *port_name0* could be tty00.

   b. Modify the /etc/inittab file to disable logins on the two serial communication ports by setting the third field to off. For example, modify the tty00 and tty01 lines as follows:

      ```
      tty00:23:off:/usr/sbin/getty /dev/tty00 9600
      tty01:23:off:/usr/sbin/getty /dev/tty01 9600
      ```

2. Reboot the system, or issue the following command to ensure that the modified files take effect:

   ```
   % /sbin/init q
   ```

2.4.3.1 Connecting to a Serial Port Under Digital UNIX

After modifying the /etc/remote and /etc/inittab files, you can connect to the serial port under the Digital UNIX operating system using the Digital UNIX tip command.

For example, to connect to the host running the Digital UNIX operating system, enter this command:

```
% tip  tty00
```

2.5 Starting and Running the Mini-Debugger

After the SROM serial port connection has been made, you can initialize the mini-debugger by typing an uppercase U. This returns an SROM> prompt, which indicates that you are ready to begin debugging hardware and displays the mini-debugger version number.

For example:

```
U
V00000801
SROM>
```

The uppercase U automatically detects the baud rate of the terminal connected to the SROM serial port. Baud rates up to 19.2K are supported.

Note

Auto baud detection requires that the first character typed be an uppercase U.

2.5.1 Default Conditions

The mini-debugger built into the Alpha board SROM contains the proper initialization conditions.

The following sample sessions show you how to set up and run the standalone version of the mini-debugger.

The following conditions apply:

- Machine checks are disabled. (This does not apply to designs based on the Alpha 21164.)
- Data cache (Dcache) is off.
- The ABOX_CTL register is set to 0. (This does not apply to designs based on the Alpha 21164.)
- The ICCSR register is set to 000005F800000000 for boards based on the 21064 and to 0x824E000000 for boards based on the Alpha 21164.
- The quadword data mode flag and the high-memory mode flag are cleared.
- The pal_base is set to where the code is loaded.
- In boards based on the Alpha 21164, all three sets of internal cache are turned on and flushed. The block size has been set to 64 bytes.

Before you begin to debug hardware with the mini-debugger you need to set several other registers depending on the microprocessor. See the sample sessions for more information.

2.6 Sample Session on the Alpha PCI 64-275

To run the mini-debugger on the Alpha PCI 64-275, set up the BIU_CTL register with the backup cache (Bcache) disabled. For example:

```
BIU_CTL :    0000004E 4001E644    /* for AlphaPCI 64-275
```

After main memory is initialized, the Bcache can be enabled as shown in the Alpha PCI 64-275 sample log.

To initialize the memory controller for a 32-megabyte bank of memory in bank 0, the following values must be loaded into the DECchip 21071-CA memory control registers:

Register	Data
Global Timing Register	00000025
Refresh Timing Register	00000444
Bank 0 Timing Register A	00002684
Bank 0 Timing Register B	00000C01
Bank 0 Base Address Register	00000000
Bank 0 Configuration Register	000000EB

Finally, all of memory must be written to initialize data parity.

To initialize and enable the Bcache, follow these steps:

1. Initialize the cache tag RAMs.

   This is done by enabling the cache in the DECchip 21071-CA and configuring the DECchip 21071-CA to ignore the tag, and to allocate blocks in the cache on writes.

2. Perform a memory write to every address from address 0 up to the size of the cache.

   This will load good data, tag, and tag control parity into the Bcache.

3. Configure the DECchip 21071-CA to use the Bcache tags, and set the BIU_CTL register in the DECchip 21064 to enable the Bcache.

4. Enable the Dcache.

Before accessing the PCI bus on the Alpha PCI 64-275, several registers in the DECchip 21071-DA must be initialized.

- The PCI Master Latency Timer must be set up before any PCI accesses are attempted.
- The address extension registers (HAXR*x*) that are used for different types of PCI accesses must also be initialized.

The Alpha PCI 64-275 sample log file initializes HAXR2 before attempting a PCI configuration cycle.

2.6.1 Alpha PCI 64-275 Sample Log File

```
/* Sample Alpha PCI 64-275 initialization log file

SROM> dc   /* Deposit to BIU_CTL Register
IPR> biu
LoD> 4001e664   /* Bcache off -- set to 4001e644 for
HiD> 4e                         /*    the Alpha PCI 64-275
*biu
    /* The Following commands init the memory
    /* subsystem.
SROM> ba   /* Set base address to base of DECchip 21071-CA
AL> 80000000   /* CSR register
AH> 1
BaseAddr  ON: 00000001.80000000
SROM> dm   /* Init Global Timing Register
A> 200
D> 25
SROM> dm   /* Init Refresh Timing Register
A> 220
D> 444
SROM> dm   /* Init Bank 0 Timing Register A
A> c00
D> 2684
SROM> dm   /* Init Bank 0 timing Register B
A> e00
D> c01
SROM> dm   /* Set Bank 0 Base Address Register to 0
A> 800
D> 0
```

```
SROM> dm   /* Init Bank 0 Configuration Register
A> a00    /* 32MB of memory in bank 0
D> eb

SROM> ca   /* Disable base address
BaseAddr OFF: 00000001.80000000

SROM> fm   /* Write good data/parity to all of memory
A> 0
A> 2000000
D> 0

    /* At this point the memory subsystem has been
    /* initialized and configured for 32MB in
    /* bank 0 using 2MBx36 SIMMs


    /* The following commands init and
    /* enable the Bcache.

SROM> sa   /* Enable base address
BaseAddr  ON: 00000001.80000000

SROM> dm   /* Write General Control Register. Enable
A> 0    /* Bcache, ignore tag, allocate on writes
D> 1b4

SROM> dm   /* Clear Tag Enable Register
A> 60
D> 0

SROM> ca   /* Disable base address
BaseAddr OFF: 00000001.80000000

SROM> fm   /* Write to entire cache to init cache Tags
A> 0    /* and data.
A> 200000
D> 0

SROM> sa   /* Enable base address
BaseAddr  ON: 00000001.80000000

SROM> dm   /* Set Tag Enable Register to 2MB cache
A> 60
D> 3fe0

SROM> dm   /* Clear ignore tag bit
A> 0
D> b4

SROM> ca   /* Disable base address
BaseAddr OFF: 00000001.80000000
```

```
SROM> dc   /* Enable cache in BIU_CTL
IPR> biu
LoD> 4001e665
HiD> 4e
*biu

    /* At this point the cache has been
    /* initialized and is enabled.

SROM> dc   /* Enable Dcache and enable Machine Checks
IPR> abox
LoD> 42a
HiD> 0
*abx

    /* The next section inits the PCI interface of
    /* the DECchip 21071-DA chip

SROM> hm   /* Enable high memory
Hi Mem

SROM> dm   /*  Init PCI Master Latency Timer
AL> a00001e0
AH> 1
D> ff

SROM> dm   /* Init HAXR2
AL> a00001c0
AH> 1
D> 0

SROM> em   /* Perform a configuration space read of the
AL> e0080000   /* SIO
AH> 1
00000001.e0080000: 04848086

SROM>
```

2.7 Sample Session on the Alpha PCI 164-266/300

To run the mini-debugger as a standalone program on the Alpha PCI 164-266 /300, the following setup needs to be performed. The mini-debugger initialized the CPU when you typed uppercase U (see Section 2.5.1). You must also initialize the rest of the board.

To set up the mini-debugger for the Alpha PCI 164-266/300, follow these steps outlined below. The sample log in Section 2.7.1 illustrates this sequence:

1. Flush the secondary cache (L2) and turn on only one set. Flushing the secondary cache prevents parity errors. Turning on only one set forces reads to come from memory and not cache.

2. Turn off the Bcache to force reads to come from memory and not cache. The Bcache Configuration register must have the Read and Write speeds set to the ratio of the sysclock to CPU. Refer to the board's design guide to determine this ratio (see section on configuration jumpers).
3. Initialize the CIA control register.
4. Initialize the CIA acknowledge register to disable the Bcache Victim Acknowledge signal.
5. Initialize the memory control register. This register controls the refresh rate, and memory width.
6. Set the bank timing registers. The Alpha PCI 164-266/300 uses only one of these three registers.
7. Set MBA0 which controls the only memory bank in the Alpha PCI 164-266 /300. The value is dependent on the amount of memory and type of SIMMs used. Refer to the following table:

Memory Size	SIMM Type	MBA Value
32MB	1Mx36 (4MB)	10000011
64MB	2Mx36 (8MB)	10008011
128MB	4Mx36 (16MB)	10000073
256MB	8Mx36 (32MB)	10008073
512MB	16Mx36 (64MB)	100001F5

8. Wake up the memory by performing eight consecutive RAS cycles to each SIMM side.
9. Turn on Dcache, Bcache, and all three sets in the secondary cache.
10. Initialize memory and the caches by writing to memory.
11. At this point, memory initialization is complete. If I/O tests are desired, then you need to initialize that part of the system (see the following steps).
12. Reset ISA bus.
13. Configure SIO, enabling accesses to RTC, configuration RAM (configuration jumpers), and flash ROM space.

2.7.1 Alpha PCI 164-266/300 Sample Log File

The following sample log file initializes the Alpha PCI 164-266/300.

```
/*Alpha PCI 164-266/300 Log.

(Shift-U)

V00000801
SROM> hm
Hi Mem

/* Flush secondary cache, set block size to 64 bytes  */
/* and turn on set S0 only to facilitate memory */
/* memory accesses.  Note that the secondary cache can't */
/* be turned off completely.   */
SROM> dm
AL> fff000a8
AH> ff
D> 3002

/* Check previous write operation.  */
SROM> em
AL> fff000a8
AH> ff
000000ff.fff000a8: 00003002

/* Note that the Bcache has been disabled and */
/* error reporting has been turned off.  */
SROM> dc
IPR> bctl
LoD> 8050
HiD> 0
*BCTL   00000000.00008050

/* The Read and Write speeds in the Bcache Cfg */
/* register must be set to the sysclock-to-cpu  */
/* ratio.  This examples shows a ratio of 8. */
SROM> dc
IPR> bcfg
LoD> 1e22880
HiD> 0
*BCFG   00000000.01e22880

/* Check previous write operations.  */
SROM> ec
BCtl    00000000.00008050
BCfg    00000000.01e22880
.... /* The rest of the registers were omitted here */

/* Set the CIA_CTRL register */
SROM> dm
AL> 40000100
AH> 87
D> 2100c0f1
```

```
/* Check previous write operation.  */
SROM> em
AL> 40000100
AH> 87
00000087.40000100: 2100c0f1

/* Set the PCI timer register */
SROM> dm
AL> 400000C0
AH> 87
D> ff00

/* Set the CIA_CACK_EN register.  Note that the */
/* Bcache victim bit has been set to 0 since the */
/* Bcache has been disabled.   */
SROM> dm
AL> 40000600
AH> 87
D> 0

/* Check previous write operation.  */
SROM> em
AL> 40000600
AH> 87
00000087.40000600: 00000000

/* Initialize the memory control registers. */
SROM> ba
AL> 50000000
AH> 87
BaseAdr ON :    00000087.50000000

SROM> lm
Lo Mem

/* Set the refresh rate, Bcache size to 0 (disabled) */
/* and memory width to 256-bits.   */
SROM> dm
A> 0
D> 1fe01

SROM> em
A> 0
00000087.50000000: 0001fe01

/* Set Bank Timing Register #1.   */
SROM> dm
A> b40
D> 60208140

SROM> em
A> b40
00000087.50000b40: 60208140
```

```
/* Set MBA0 which controls the only memory bank on */
/* EB164.  This example applies to all 8 SIMMs fully */
/* populated with 2Mbx36 SIMMs (8MBytes SIMMs) which */
/* gives a total of 64MBs.  See table for other values. */
SROM> dm
A> 600
D> 10008011

SROM> em
A> 600
00000087.50000600: 10008011

SROM> ca
BaseAdr OFF:    00000087.50000000

/* Perform 8 consecutive RAS cycles to "wake up" memory. */
/* Because we can't turn off the secondary cache completely, we must */
/* make sure that the reads are not cached.  */
/* Do the following 2 reads 8 times */

/* Read from side 0 of SIMM */
SROM> em
A> 0
00000000.00000000: ffffffff

/* Read from size 1 of SIMM, ejecting previous read */
SROM> em
A> 2000000   /* Should be half your memory size */
00000000.02000000: ffffffff

/* Turn on the DCache */
SROM> dc
IPR> dcmd
LoD> 1
HiD> 0
*DCMD   00000000.00000001

/* Turn on the Bcache */
SROM> dc
IPR> bctl
LoD> 8051
HiD> 0
*BCTL   00000000.00008051

/* The BCFG value shown is for a 266MHz CPU, with 2MB, 10ns Bcache. */
SROM> dc
IPR> bcfg
LoD> 1e22772
HiD> 0
*BCFG   00000000.01e22772
```

```
SROM> ec
BCtl    00000000.00008051
BCfg    00000000.01e22772
.... /* The rest of the registers were omitted here */
DcMd    00000000.00000001
.... /* The rest of the registers were omitted here */

SROM> hm
Hi Mem

/* Tell memory controller the size of Bcache (2MB) */
SROM> dm
AL> 50000000
AH> 87
D> 1fe21

/* Enable CACK of Bcache Victims */
SROM> dm
AL> 40000600
AH> 87
D> 8

/* Enable all 3 sets in the secondary cache and leave at 64-byte blocks */
SROM> dm
AL> fff000a8
AH> ff
D> f000

SROM> em
AL> fff000a8
AH> ff
000000ff.fff000a8: 0000f000

/* Initialize memory and caches */
SROM> lm
Lo Mem

/* Fill memory to obtain good ECC */
SROM> fm
A> 0
A> 4000000       /* Total memory size in board */
D> 12345678

SROM> bm
A> 0
A> 20
00000000.00000000: 12345678
00000000.00000004: 12345678
00000000.00000008: 12345678
00000000.0000000c: 12345678
00000000.00000010: 12345678
00000000.00000014: 12345678
00000000.00000018: 12345678
00000000.0000001c: 12345678
```

```
SROM> bm
A> 200000
A> 200020
00000000.00200000: 12345678
00000000.00200004: 12345678
00000000.00200008: 12345678
00000000.0020000c: 12345678
00000000.00200010: 12345678
00000000.00200014: 12345678
00000000.00200018: 12345678
00000000.0020001c: 12345678

SROM> bm
A> 400000
A> 400020
00000000.00400000: 12345678
00000000.00400004: 12345678
00000000.00400008: 12345678
00000000.0040000c: 12345678
00000000.00400010: 12345678
00000000.00400014: 12345678
00000000.00400018: 12345678
00000000.0040001c: 12345678

/* Reset the ISA bus */
SROM> ba
AL> 0
AH> 87
BaseAdr ON :    00000087.00000000

/* Assert the reset signal */
SROM> dm
A> 809a0
D> 5800

/* Deassert the reset signal */
SROM> dm
A> 809a0
D> 5000

/* Set UBCSA register to allow access to RTC, keyboard, */
/* lbios and xbios */
SROM> dm
A> 809c0
D> c30000

/* Set UBCSB register to allow access to Confram (configuration */
/* jumpers) and to disable others */
SROM> dm
A> 809e0
D> ff000000
```

```
/* Read SIO identification registers */
SROM> em
A> 80000
00000087.00080000: 04848086

SROM> ca
BaseAdr OFF:    00000087.00000000

SROM> hm
Hi Mem

/* Write out to the leds post-card.  You should see an "F0" */
/* printed on the LEDs.         */
SROM> dm
AL> 80001000
AH> 85
D> f0

/* Read first byte of flash.  If it has an image with the */
/* standard ROM header, you will see the following. */
SROM> em
AL> fff80000
AH> 86
00000086.fff80000: 5a5ac3c3
```

3

SROM Mini-Debugger Command Set

3.1 Overview

This chapter describes the Alpha Microprocessors SROM Mini-Debugger command set.

3.2 Command and User Interface Features

The following list describes some of the features of the mini-debugger command language:

- Uppercase or lowercase characters can be used.
- The delete key provides primitive command-line editing.
- Numbers are input and output in hexadecimal format.
- For commands that prompt for input, the default input value is all zeros.
- Addresses are masked on an even longword boundary in **lw** mode and on an even quadword boundary in **qw** mode.
- Data and address inputs are taken one longword at a time.
- The looping commands initiate an infinite loop. To exit, press any key.

3.3 Command Summary

Table 3–1 summarizes the command set for the Alpha Microprocessors SROM Mini-Debugger. These commands are described in the following sections.

Table 3–1 Command Summary

Command	Description
dm	Deposits data to one memory location.
em	Examines one memory location.
qw	Sets quadword data mode (64-bit data).
lw	Sets longword data mode (32-bit data).
fm	Fills a block of memory with a data pattern.
bm	Displays a range (block) of memory locations.
hm	Sets high-memory mode (64-bit address).
lm	Sets low-memory mode (32-bit address).
ba	Gets a base address and sets the base address mode flag.
sa	Sets the base address mode flag.
ca	Clears the base address mode flag.
wm	Performs a looping write to one memory location.
rm	Performs a looping read from one memory location.
!m	Performs a looping write/read sequence at one memory location.
dc	Deposits data to one CPU register.
ec	Examines contents of CPU registers.
xm	Loads an external image to memory.
st	Starts execution at an address.
xb	Begins execution of the last image loaded with **xm**.
rt	Returns to the calling program.
pr	Prints register contents.

3.3.1 Deposit and Examine Data at One Memory Location (dm, em)

The deposit memory (**dm**) command deposits a data pattern at one memory location. The examine memory (**em**) command examines the data pattern at one memory location and displays the value on the terminal screen.

This sequence deposits data and reads it back.

```
SROM> dm
A> 100000
D> deadbeef

SROM> em
A> 100000
00000000.00100000: deadbeef
```

Note

A memory barrier (MB) instruction is executed after the store in a **dm** command to force the data to its destination.

3.3.2 Set Quadword or Longword Data Mode (qw, lw)

The quadword (**qw**) command sets the quadword data mode flag. All deposits and examines then reflect 64-bit data. When doing a deposit you are prompted first for the low longword of data, then for the high longword.

The longword (**lw**) command clears the quadword data mode flag (or sets longword data mode). All deposits and examines then reflect 32-bit data. This is the default.

Note

With the quadword data mode flag (**qw**) set, all reads and writes are done with ldq/p and stq/p instructions, respectively. With this flag cleared (**lw**), all reads and writes are done with ldl/p and stl/p instructions, respectively.

The following example shows how the **qw** command affects the output of the **dm** and **em** commands.

```
SROM> qw
QW Mode

SROM> dm
A> 100000
LoD> 12345678
HiD> 90abcdef

SROM> em
A> 100000
00000000.00100000: 90abcdef.12345678
```

The following example shows how the **lw** command affects the output of the **dm** and **em** commands.

```
SROM> lw
LW Mode

SROM> dm
A> 100000
D> deadbeef

SROM> em
A> 100000
00000000.00100000: deadbeef
```

3.3.3 Fill and Block Read Memory Range (fm, bm)

The fill memory (**fm**) command writes over the specified range of memory locations. The block read memory (**bm**) command displays the specified range of memory locations. Both the **fm** and the **bm** commands prompt for a starting and ending address. The block read or write is performed on the specified address range, up to, but not including, the end address. For example, with an ending address of 4000 in **lw** mode, the last longword location accessed is 3FFC. In **qw** mode, the last quadword location accessed is 3FF8.

In the following example of the **fm** command sequence, 100003 is the lower boundary of the fill. Because the quadword data mode flag (**qw**) has been set, the lower boundary is truncated to the nearest quadword (100000). The upper boundary of the fill (10004F) is truncated to 100048. Therefore, the last address written with the specified data pattern is 100040.

In the example **bm** command sequence, the lower boundary FFFF7 is truncated to FFFF0, and the upper boundary is truncated to 100058. Therefore, the last address read is 100050.

The following are examples of the **fm** and **bm** commands.

```
SROM> qw
QW Mode

SROM> fm
A> 100003
A> 10004f
LoD> deadbeef
HiD> 76543210

SROM> bm
A> ffff7
A> 10005f
00000000.000ffff0: 00000000.00000000
00000000.000ffff8: 00000000.00000000
00000000.00100000: 76543210.deadbeef
00000000.00100008: 76543210.deadbeef
00000000.00100010: 76543210.deadbeef
00000000.00100018: 76543210.deadbeef
00000000.00100020: 76543210.deadbeef
00000000.00100028: 76543210.deadbeef
00000000.00100030: 76543210.deadbeef
00000000.00100038: 76543210.deadbeef
00000000.00100040: 76543210.deadbeef
00000000.00100048: 00000000.00000000
00000000.00100050: 00000000.00000000

SROM>
```

Note

An MB instruction is done after the last store. Write-buffer merging occurs unless it is disabled.

3.3.4 Set High-Memory or Low-Memory Address Mode (hm, lm)

The high-memory (**hm**) command sets the high-memory address mode flag that allows the mini-debugger to accept 64-bit addresses. When **hm** is selected, any command that requires an address prompts for the low longword of address, then for the high longword of address.

The low-memory (**lm**) command clears the high-memory address mode flag so that the mini-debugger accepts 32-bit addresses. When you enter a command that requires an address, you are prompted only for the low longword of address; this is the default.

In the following example, a blank input defaults to zero:

```
SROM> hm
Hi Mem

SROM> bm
AL> ffff0
AH>
AL> 100050
AH>
00000000.000ffff0: 00000000.00000000
00000000.000ffff8: 00000000.00000000
00000000.00100000: 76543210.deadbeef
00000000.00100008: 76543210.deadbeef
00000000.00100010: 76543210.deadbeef
00000000.00100018: 76543210.deadbeef
00000000.00100020: 76543210.deadbeef
00000000.00100028: 76543210.deadbeef
00000000.00100030: 76543210.deadbeef
00000000.00100038: 76543210.deadbeef
00000000.00100040: 76543210.deadbeef
00000000.00100048: 00000000.00000000
```

3.3.5 Get, Set, and Clear a Base Address (ba, sa, ca)

The base address flag is useful for reading and writing the same address frame, particularly when the upper longword of memory address is required. The base address is added to the address specified for any command in both **hm** and **lm** modes. The base address (**ba**) command loads the base address and sets the base address flag. The set address (**sa**) command sets the base address flag and enables the previously loaded base address. The clear address (**ca**) command clears the base address mode flag leaving the base address untouched.

In the following example, the system has an I/O device mapped at 1.00000000 and a control and status register mapped at 1.F0000000. The test determines if a timeout bit gets set in the control status register (CSR) when reading from the unconnected I/O device. To get, set, and clear a base address, follow these steps:

1. Set the base address to the CSR address, 1.F0000000.
2. Clear the base address mode flag, so that physical address 0 is used as the reference for the next operation.
3. Set the high-memory address mode flag.
4. Read from the disconnected I/O device to cause a timeout.
5. Set the base address mode flag, so that the base address specified in step 1 is used as the reference for the next operation.
6. Read offset zero from the base address with the high-memory address mode flag set. The timeout bit, bit 29, is set.
7. Clear the high-memory address mode flag.
8. Read offset zero from the base address with the high-memory address mode flag cleared.

The following example demonstrates the previous steps:

```
SROM> ba
AL> f0000000
AH> 1
BaseAddr  ON: 00000001.f0000000

SROM> ca
BaseAddr OFF: 00000001.f0000000

SROM> hm
Hi Mem

SROM> em
AL>
AH> 1
00000001.00000000: 00000000

SROM> sa
BaseAddr  ON: 00000001.f0000000

SROM> em
AL>
AH>
00000001.f0000000: 20000000

SROM> lm
Lo Mem
```

```
SROM> em
A>
00000001.f0000000: 20000000

SROM>
```

3.3.6 Looping Write to a Memory Location (wm)

The write memory (**wm**) command performs a looping write to a single memory location. The loop can be terminated by pressing any key.

In the following example, **lm** and **lw** modes are assumed:

```
SROM> wm
A> 80000
D> 55555555
```

In this example, the mini-debugger writes 5's to location 80000 until you terminate the loop by pressing another key.

3.3.7 Looping Read from a Memory Location (rm)

The read memory (**rm**) command performs a looping read to a single memory location. The loop can be terminated by pressing any key.

In the following example, **lm** and **lw** modes are assumed:

```
SROM> rm
A> 80000
```

In this example, the mini-debugger reads location 80000 until you terminate the loop by pressing any key.

3.3.8 Looping Write/Read Sequence at a Memory Location (!m)

The write/read sequence (**!m**) command performs a write followed by a read at the specified address.

The following example performs a continuous write of all f's followed by a read memory at 100000, until you press any key:

```
SROM> !m
A> 100000
D> ffffffff
```

3.3.9 Deposit Data to a CPU Register (dc)

The deposit CPU register (**dc**) command deposits data to the specified CPU register. Table 3–2 lists the supported CPU registers and the command option names.

Table 3–2 Options for dc Command

Option	Register Name
biu	BIU_CTL
abox	ABOX_CTL
palb	PAL_base
bctl[1]	Bcache control
bcfg[1]	Bcache configuration
icsr	ICCSR or ICSR
dcmd[1]	Dcache mode
ipl[1]	IPL

[1]Implemented for the Alpha 21164 microprocessor only.

When you issue the **dc** command, you are prompted for the register, the low longword of data, and the high longword of data.

In the following example, the ABOX_CTL register is written with data that enables machine checks and enables the data cache (Dcache):

```
SROM> dc
IPR> abox
LoD> 402
HiD>
*abx
```

3.3.10 Examine CPU Register (ec)

The examine CPU register (**ec**) command displays a list of CPU registers, including the registers that can be written to (as specified in the **dc** command) and the registers dumped by a machine check.

Note

The BIU_CTL, and ABOX_CTL are write-only registers. The examine is performed on the corresponding *shadow* register. The shadow register reflects what you entered.

For example:

```
V00000801     /*SROM version number
SROM> ec
AboxCtl 00000000.00001428
Icsr    00000000.00ff0000    /* Read format */
PalBase 00000000.00000000
ExcAddr 00000000.00000a76
DcStat  00000000.00000003
DcAddr  00000007.ffffffff
BiuCtl  0000004e.2001c665
BiuStat 00000000.00003040
BiuAddr 00000000.00318940
FillSyn 00000000.00000000
FillAdr 00000000.00318944

SROM>
```

Note

The **ec** command output will vary depending on the CPU. The the Alpha 21164 microprocessor does not contain the BIU_CTL register. Consequently, the **ec** command does not display biu_ctl, biu_stat, fil_ addr, HiLw_syn, or LoLw_syn.

The following example was generated using the mini-debugger on an Alpha PCI 164-266/300:

```
SROM> ec
BCtl    00000000.00028051 /* Bcache control register (BC_CONTROL) */
BCfg    00000000.01e21772 /* Bcache configuration register (BC_CONFIG) */
Icsr    000000c2.4e000000 /* IBox control and status (ICSR) register.*/
PalB    00000000.00000000 /* Palcode Base (PAL_BASE) */
ExAd    00000000.0031dcb1 /* Exception Address reg. (EXC_ADDR)*/
ExMk    00000000.00000000 /* Exception Mask. (EXC_MASK) */
ExSm    00000000.00000000 /* Exception Summary (EXC_SUM). */
Ipl     00000000.0000001f /* Interrupt Priority Level (IPL) reg. */
Int     00000000.00000015 /* Interrupt ID (INTID)  */
Isr     00000000.00200000 /* Interrupt summary (ISR). */
IcPE    00000000.00000000 /* Icache Parity Error Status (ICPERR_STAT) */
DcMd    00000000.00000001 /* Dcache Mode (DC_MODE) */
DcPE    00000000.00000000 /* Dcache Parity Error Status (DC_PERR_STAT) */
MmSt    00000000.00016051 /* Memory management fault status (MM_STAT) */
VaFm    00000002.0000f000 /* Virtual Address (VA_FORM) */
Va      00000000.03c00000 /* Faulting Virtual Address (VA) */

SROM>
```

3.3.11 Load an External Image to Memory (xm)

The external memory (**xm**) command loads an external image into memory. To do this, you need a workstation, with a corresponding version of XLOAD.EXE (VMS) or uload (ULTRIX), connected to the motherboard or your Alpha microprocessor board.

3.3.12 Start Execution at an Address (st)

The start (**st**) command begins executing the downloaded image at the specified address. If the address does not contain executable code, then the machine hangs. You must recycle the power to start again.

Note

The Alpha Microprocessors SROM Mini-Debugger is stored in the instruction cache (Icache). An **st** command causes the Icache to be overwritten.

3.3.13 Begin Execution of Last Image Loaded (xb)

The **xb** command performs a necessary instruction cache flush before it transfers control to the image loaded using the **xm** command, and it begins executing the image.

3.3.14 Return to Calling Program (rt)

The **rt** command allows you to place calls to the mini-debugger code from the SROM code and then return to the SROM code after performing a desired operation in the mini-debugger. For example, you could print the contents of all general-purpose registers at the time the mini-debugger was called.
To call the mini-debugger from your program, use the JSR or BSR instructions. The return address is expected to be in R0, and R19 must contain the standard signature (0xDECB0000) to prevent the SROM from executing the CPU initialization code. PALtemp registers will also be modified and thus should not be used in the SROM code (to avoid overwriting data).

3.3.15 Print Register Contents (pr)

The **pr** command displays the contents of the general-purpose CPU registers.

Note

This command may not be available in all standard SROMs supplied with the Alpha motherboards due to a lack of space. You may want to recompile the mini-debugger source files supplied in the EBSDK with the compile switch "FULL_MDBG" defined to enable this command.

3.4 Onboard Machine Check Handler

The onboard machine check handler is useful in debugging certain memory faults. You must set bit 1 of the ABOX_CTL register to enable machine checks. When a machine check is encountered, as it might be in a read, the machine check handler prints the following message followed by the SROM> prompt:

```
MCHK
exc_addr 00000000.00004021
biu_stat 00000000.00002240
fil_addr 00000000.00004028
HiLw_syn 00000000.00000000
LoLw_syn 00000000.00000000
dc_stat  00000000.00000330
dc_addr  00000007.ffffffff

SROM>
```

Index