COMPAQ

Alpha Microprocessors SROM Mini-Debugger

User's Guide

Order Number: EC–QHUXD–TE

Revision/Update Information: This is a revised document. It supersedes the *Alpha Microprocessors SROM Mini-Debugger User's Guide* (EC–QHUXC–TE).

Compaq Computer Corporation

April 1999

Contents

A Support

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 motherboards:

- AlphaPC 264DP Motherboard
- AlphaPC 164SX Motherboard
- AlphaPC 164LX Motherboard

Audience

This document is for anyone who develops hardware 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.
- Appendix A contains information about customer support and associated documentation.

Conventions

The following conventions are used in this document:

Convention	Meaning
A percent sign (%)	Indicates a Tru64 UNIX operating system command prompt.
SROM>	Indicates an Alpha Microprocessors SROM Mini-Debugger prompt.
Boldface type	Indicates a command or an example of user input.
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 Alpha Microprocessors Motherboard Software Design Tool Kit (Alpha SDK) includes the Alpha Microprocessors SROM Mini-Debugger binary files suitable for programming an SROM. For information about how to program an SROM, refer to your ROM programmer manual.

After the SROM is programmed, it can be placed into the SROM socket on the motherboard. In addition, the mini-debugger is also available in the standard SROM provided with the Alpha microprocessor 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 Alpha motherboard user's guide for more information about how to access the mini-debugger and setting the required jumper.

2.2 Hardware Required

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

- An Alpha microprocessor motherboard or a system based on the Alpha microprocessor architecture with a connection from the microprocessor SROM interface to the host system's serial port (for example, an RS-232)
- A host system (a terminal or workstation)
- An SROM containing the SROM Mini-Debugger image

2.3 Hardware Debug Features

The mini-debugger image is loaded into the CPU's instruction cache at reset through the CPU's SROM interface. The mini-debugger provides commands to:

- Examine and deposit data in memory.
- Test memory.
- Examine and deposit internal CPU registers.
- Load an image into the motherboard's memory and transfer execution to it.

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 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. This section describes how to connect the SROM serial port of a motherboard to the following hardware:

- A terminal
- A PC running communication software
- A system running Windows NT
- An Alpha system running Tru64 UNIX

2.4.1 Connecting the Motherboard to a Terminal

To connect a motherboard to a terminal, connect the SROM serial port of the motherboard to the terminal communication line.

Set terminal settings as shown in the following table:

Terminal Setting	Value
Terminal emulation	VT100
Transmit/receive speed	9600 baud
Data bits	8
Parity	None
Stop bits	1

2.4.2 Connecting the Motherboard to a Personal Computer

You can also use communication (terminal emulation) software running on a PC to communicate with the motherboard. To connect the motherboard to a PC, connect the terminal communication line to the SROM serial port of the motherboard as described for the terminal.

2.4.3 Connecting to the Motherboard from a System Running Windows NT Version 4.0

A system running the Windows NT version 4.0 operating system supports serial communication with the motherboard. Use the Start button on the taskbar to configure a COM port for connection to the motherboard and follow these steps:

1. Choose the `Programs` menu.
2. Choose the `Accessories` menu.

3. Choose the `HyperTerminal` menu and choose the `HyperTerminal` icon. The Connection Description window appears.
4. Enter a name for the new connection such as `Direct Connection Com1`, choose an icon for the connection, and click OK.
5. Set the following terminal characteristics:

Terminal Setting	Value
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Click OK to save these settings. The motherboard connection appears in the HyperTerminal window.

2.4.4 Connecting to the Motherboard from a System Running Tru64 UNIX

An Alpha system running the Tru64 UNIX operating system supports serial communication through two ports that can be connected to the motherboard:

- /dev/tty00
- /dev/tty01

All examples and command descriptions that follow assume that the motherboard SROM port is connected to port `/dev/tty00`.

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

1. Log in as superuser.
2. 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 Tru64 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 an arbitrary name that you assign to that port.

   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
   ```

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

```
#  /sbin/init q
```

2.4.4.1 Connecting to a Serial Port Under Tru64 UNIX

After you modify the `/etc/remote` and `/etc/inittab` files, you can connect to the serial port under the Tru64 UNIX operating system using the Tru64 UNIX `tip` command. If the connection is successful, the mini-debugger prompt displays after you press a key. For example:

```
% tip port_name0
                              ! key is pressed.
SROM>
```

Type **~.** to exit the Tru64 UNIX `tip` command.

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 ASCII character. 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>
```

Once an ASCII character is typed, the mini-debugger automatically detects the baud rate of the terminal connected to the SROM serial port. Baud rates up to 19.2K are supported.

2.5.1 Default Conditions

If you are using the mini-debugger built into the standard Alpha motherboard SROM, then the proper initialization conditions are automatically set. When you see the mini-debugger prompt, the system has been initialized.

The example shown in Section 3.4, the xm command, demonstrates how to load and execute the Debug Monitor.

2.6 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. For 21164-based boards, machine checks are always enabled. 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
Abox    00000000.00000002
Icsr    00000000.00ff0000
PalB    00000000.00000000
ExAd    00000000.00200cb4
DcSt    00000000.0000000b
Hirr    00000000.00000000
Hier    00000000.00001890
BCtl    000007f8.00000000
BiSt    00000000.000010c1
BiAd    00000001.e0000018
Syn     00000000.00000000
FiAd    00000000.00200950

SROM>
```

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 interchangeably.
- Only the first two characters of a command line are significant; the rest are ignored.
- Numbers are input and output in hexadecimal format.
- Commands can be aborted at any time by pressing Ctrl/C (except for the xm command).
- For commands that prompt for input, pressing Return on an empty line defaults to a value of 0.
- In qw data mode, addresses are aligned to a quadword boundary (the three least significant bits of the addresses are zero). In all other modes, addresses are aligned to a longword boundary (the two least significant bits of the addresses are zero).
- In qw data mode, all reads and stores are performed using ldq/p and stq/p instructions, respectively. In all other modes, reads and stores use ldl/p and stl/p instructions, respectively.
- All stores are followed with two MB instructions. This keeps writes ordered (from the system's point of view) and prevents merging in the write queue. For more information on the MB instruction, see the CPU's hardware reference manual.
- For commands that use an address range, the ending address is not included in the range. The last read or store is performed in the immediately preceding quadword or longword, depending on the state of the qw data flag.

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
	Flag Commands
!d	Enable/disable negation of data to be written.
ba	Enable/disable use of base address.
di	Enable/disable display to screen.
fr	Enable/disable follow-with-read flag.
fw	Enable/disable follow-with-write flag.
lo	Enable/disable command repetition.
qw	Enable/disable quadword mode operations.
wa	Enable/disable write address mode.
	Memory Commands
bm	Read block of memory addresses.
cm	Compare two sections of memory.
cp	Copy a block of memory.
fm	Fill memory range with data pattern.
mt	Perform simple memory test.
	Deposit/Examine Commands
dc	Deposit internal CPU register.
dm	Deposit data to memory location.
ec	Examine internal CPU registers.
em	Examine data in a memory location.
	Print Commands
fl	Print current state of flags.
pr	Print contents of general-purpose CPU registers.
	Load/Execute Commands
rt	Exit mini-debugger.
st	Start executing at specified address.
xb	Begin execution of the last image loaded.
xm	Load external image to memory.
	Miscellaneous Command
sb	Set base address.

3.4 User Commands

This section contains a complete description and examples of the SROM Mini-Debugger commands. The commands are listed in alphabetical order. The Control Flags section in each command description lists the flags that affect the behavior of the command if the flag is enabled.

3.4.1 !d — Set Negate Data Flag

The negate data (**!d**) command enables or disables the use of the one's complement of the data to be written by toggling the negate data flag.

Control Flags

Not applicable.

Description

With the negate data flag enabled, writes will use the complement (negation) of the data specified by the user or automatically generated by the command in use.

The default state is off.

Example

In the following example, with the negate data flag enabled, the data supplied by the user (0) is complemented and written to address 500000.

```
SROM> !d
Neg Data ON

SROM> dm
A> 500000
D> 0

SROM> em
A> 500000
00000000.00500000: ffffffff
```

With the fill memory command, the data written with each write is also complemented, resulting in every other write having the original data.

```
SROM> fm
A> 600000
A> 600020
D> ffff0000

SROM> bm
A> 600000
A> 600020
00000000.00600000: 0000ffff
00000000.00600004: ffff0000
00000000.00600008: 0000ffff
00000000.0060000c: ffff0000
00000000.00600010: 0000ffff
00000000.00600014: ffff0000
00000000.00600018: 0000ffff
00000000.0060001c: ffff0000
```

If the write address flag is also enabled, then data written is the complement of the destination address.

```
SROM> wa
Wrt Addr ON

SROM> fm
A> 500000
A> 500010

SROM> bm
A> 500000
A> 500010
00000000.00500000: ffaffffff   ! This is the complement of x500000.
00000000.00500004: ffaffffb
00000000.00500008: ffaffff7
00000000.0050000c: ffaffff3
```

3.4.2 ba — Set Base Address Flag

The set base address (**ba**) flag command enables or disables the base address flag, which must be enabled to use the set base (**sb**) command. The **ba** command toggles the base address flag.

Control Flags

Not applicable.

Description

When the base address flag is enabled, the address entered with the **sb** command is added to addresses entered in any subsequent examine or deposit command.

The default state is off.

Example

To access addresses in the range 3.FFF80000 to 3.FFFFFFFF, it may be more convenient to enter its base address and work with offsets, rather than typing the absolute address every time.

```
SROM> sb
A> 3fff80000
00000003.fff80000
BaseAddr ON

SROM> bm
A> 0
A> 10
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> ba
00000003.fff80000
BaseAddr OFF

SROM> em
A> 400000
00000000.00400000: 00400000

SROM> ba
0000003.fff80000
BaseAddr ON

SROM> bm
A> 30
A> 40
00000003.fff80030: 00000001
00000003.fff80034: 0000be8a
00000003.fff80038: 47ff041f
00000003.fff8003c: 47ff041f
```

3.4.3 bm — Read Data from Block of Memory

The block memory (**bm**) command displays the data read from a specified range of addresses.

Control Flags

ba (base address flag)
di (display flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)

Description

The block memory command reads data from a block of memory locations and prints it out to the screen if the display flag is enabled. The range is specified by first entering the starting address, followed by the ending address.

Example

The following example shows a block memory command display:

```
SROM> wa              ! Data for fill memory command.
Wrt Addr ON

SROM> qw              ! Use quadword writes and 64-bit data.
QW ON

SROM> fm              ! Fill range with its own address.
A> 400000
A> 400010

SROM> bm              ! Display range.
A> 400000
A> 400010
00000000.00400000: 00000000.00400000
00000000.00400008: 00000000.00400008
```

3.4.4 cm — Compare Memory Sections

The compare (**cm**) command compares two sections of memory and displays any differences.

Control Flags

ba (base address flag)
lo (loop flag)
qw (quadword data flag)

Description

The compare command checks the equality of two blocks of memory and prints any differences. The first two addresses specify the starting and ending addresses of the first block to be checked and the third input provides the starting address of the second block.

Example

In the following example, two blocks are filled with the same data and then two locations are changed. The compare command shows those two locations as having different data, even though the display flag is disabled.

```
SROM> fm                ! Fill the first block with a known pattern.
A> 500000
A> 500020
D> 12345678

SROM> fm                ! Fill the second block with a known pattern.
A> 600000
A> 600020
D> 12345678

SROM> dm                ! Change first location.
A> 600004
D> bad

SROM> dm                ! Change second location.
A> 60000c
D> deed

SROM> di                ! Does not affect the compare command.
Disp OFF

SROM> cm                ! Compare both sections.
A> 500000
A> 500020
A> 600000
00000000.00500004: 12345678
00000000.00600004: 00000bad
00000000.0050000c: 12345678
00000000.0060000c: 0000deed
```

3.4.5 cp — Copy Data Between Memory Sections

The copy (**cp**) command reads data from a range of addresses and writes it to another address.

Control Flags

ba (base address flag)
lo (loop flag)
qw (quadword data flag)

Description

The copy command moves sections of data from one place in memory to another. The first two addresses specify the starting and ending addresses of the block to be moved. The third input provides the destination address for the copy.

Example

In the following example, a block starting at 400000 is filled with its own addresses and then copied to 500000:

```
SROM> wa                ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fm                ! Fill this block with its own addresses.
A> 400000
A> 400020

SROM> cp                ! Copy the block to 500000.
A> 400000
A> 400020
A> 500000

SROM> bm                ! This shows the data was moved to 500000.
A> 500000
A> 500010
00000000.00500000: 00400000
00000000.00500004: 00400004
00000000.00500008: 00400008
00000000.0050000c: 0040000c
```

3.4.6 dc — Deposit CPU Register

The deposit CPU register (**dc**) command changes the contents of internal CPU registers.

Control Flags

Not applicable.

Description

The deposit CPU register command changes the contents of internal CPU registers. These registers are CPU dependent, so command input is different for each Alpha CPU. Table 3–2 show the names assigned by the mini-debugger to the CPU registers; these are the only valid names that may be entered at the IPR prompt.

Table 3–2 CPU Register Names for the dc Command

Mini-Debugger Name	CPU Register Name	Description
	21264	
Cc	CC	Cycle counter
Va	VA	Virtual address
Vafo	VA_FORM	Virtual address format
ExCa	EXC_ADDR	Exception address
Ivaf	IVA_FORM	Instruction VA format
Ps	PS	Processor status
Ier	IER	Interrupt enable
Ierc	IER_CM	Interrupt enable and current mode
Sirr	SIRR	Software interrupt request
Isum	ISUM	Interrupt summary
Excs	EXC_SUM	Exception summary
PalB	PAL_BASE	PAL base address
Ictl	I_CTL	Ibox control
Pctr	PCTR_CTL	Performance counter control
Ista	I_STAT	Ibox status
Asn	ASN	Address space number
Aste	ASTE	AST enable register
Astr	ASTR	AST request register
Ppce	PPCE	Process performance counting enable
Fpe	FPE	Floating-point enable
Mmst	MM_STAT	Memory management status
Dcst	DC_STAT	Dcache status

Table 3–2 CPU Register Names for the dc Command (Continued)

Mini-Debugger Name	CPU Register Name	Description
	21164	
BCtl*	BC_CONTROL	Bcache control
BCfg*	BC_CONFIG	Bcache configuration
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
DcMd	DC_MODE	Dcache mode
Ipl	IPLR	Interrupt priority level

* Write-only registers whose values are obtained from a copy placed in PALtemp registers 1 and 2 when they were written with the dc command. Therefore, changing these internal CPU registers overwrites the saved contents of the general-purpose CPU registers 1 and 2 (also stored in PALtemp 1 and 2), affecting the output of the pr command.

Example

In the following example, output on a DP264 CPU is displayed:

```
SROM> dc                ! Enable machine checks and CRD interrupts.
IPR> ictl
D> 3506386
*ICTL   00000000.03506386

SROM> ec
Cc      00000000.0000b78d
Va      00000800.7ffffffc
Vafo    00000002.001ffff8
ExCa    00000000.00203ee8
Ivaf    00000000.00000000
Ps      00000001.00000000
Ier     00000001.00000000
Ierc    00000001.00000000
Sirr    00000000.00000000
Isum    00000000.00000000
Excs    00000000.00001fc0
PalB    00000000.00000000
Ictl    00000000.03506386
Pctr    00000000.10000040
Ista    00000000.00000000
Asn     00000000.00000004
Aste    00000000.00000004
Astr    00000000.00000004
Ppce    00000000.00000004
Fpe     00000000.00000004
Mmst    00000000.000002c1
Dcst    00000000.00000000
```

3.4.7 di — Display Data

The display (**di**) command enables or disables the display of data to the screen by toggling the display flag.

Control Flags

Not applicable.

Description

When the display flag is enabled, the examine commands print the data obtained. When it is disabled, the read operations still take place but the data is not displayed — not even errors. The **cm** and **mt** commands, which produce read operations of their own, ignore the state of this flag and always display the data if there is a mismatch.

The default state is on.

Example

In the following example, the **em** command performs a read operation and displays the data read in because the display flag is enabled. However, when the flag is disabled, the read operations performed by the **bm** command are not echoed to the screen.

```
SROM> em
A> 500000
00000000.00500000: ffffffff

SROM> di
Disp OFF

SROM> bm
A> 500000
A> 500010
```

The next example shows that the **cm** command is not affected by this flag's state and prints out data whenever a mismatch occurs.

```
SROM> di
Disp OFF

SROM> cm
A> 500000
A> 500010
A> 600000
00000000.00500000: ffffffff
00000000.00600000: 0000ffff
00000000.00500004: 000f0000
00000000.00600004: ffff0000
00000000.00500008: 000f0000
00000000.00600008: 0000ffff
00000000.0050000c: 000f0000
00000000.0060000c: ffff0000
```

3.4.8 dm — Deposit Memory

The deposit memory (**dm**) command writes a data pattern to one memory location.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The deposit memory command writes a data pattern to the specified memory location. If the quadword data flag is enabled, then 64 bits of data are written; otherwise, only 32 bits of data are used. The data pattern is provided by the user but can be affected by the state of the write address flag and the negate data flag.

Example

In this example, a continuous write to address 500000 is performed with the loop flag enabled. Note that because the quadword data flag is disabled, only the low 32 bits of the data pattern entered are written.

```
SROM> di              ! Enable display mode.
Disp ON

SROM> qw              ! Writes are performed 32 bits at a time.
QW OFF

SROM> lo              ! Repeat operation until a key is pressed.
Loop ON

SROM> dm              ! Only the low 32-bits are used in these writes.
A> 500000
D> 123456789abcdef
                      ! (key pressed)

SROM> em              ! Examine contents of address 500000.
A> 500000
00000000.00500000: 89abcdef
00000000.00500000: 89abcdef
00000000.00500000: 89abcdef
                      ! (key pressed)
```

3.4.9 ec — Examine CPU Registers

The examine CPU registers (**ec**) command prints the contents of internal CPU registers.

Control Flags

Not applicable.

Description

The examine CPU registers command displays the contents of internal CPU registers. These registers are CPU dependent, so command output is different for each Alpha CPU. Table 3–3 show the names assigned by the mini-debugger to the CPU registers.

Table 3–3 CPU Register Names for the ec Command

Mini-Debugger Name	CPU Register Name	Description
	21264	
Cc	CC	Cycle counter
Va	VA	Virtual address
Vafo	VA_FORM	Virtual address format
ExCa	EXC_ADDR	Exception address
Ivaf	IVA_FORM	Instruction VA format
Ps	PS	Processor status
Ier	IER	Interrupt enable
Ierc	IER_CM	Interrupt enable and current mode
Sirr	SIRR	Software interrupt request
Isum	ISUM	Interrupt summary
Excs	EXC_SUM	Exception summary
PalB	PAL_BASE	PAL base address
Ictl	I_CTL	Ibox control
Pctr	PCTR_CTL	Performance counter control
Ista	I_STAT	Ibox status
Asn	ASN	Address space number
Aste	ASTE	AST enable register
Astr	ASTR	AST request register
Ppce	PPCE	Process performance counting enable
Fpe	FPE	Floating-point enable
Mmst	MM_STAT	Memory management status
Dcst	DC_STAT	Dcache status

Table 3–3 CPU Register Names for the ec Command (Continued)

Mini-Debugger Name	CPU Register Name	Description
	21164	
BCtl*	BC_CONTROL	Bcache control
BCfg*	BC_CONFIG	Bcache configuration
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
ExAd	EXC_ADDR	Exception address
Ipl	IPLR	Interrupt priority level
Int	INTID	Interrupt ID
Isr	ISR	Interrupt summary
IcPE	ICPERR_STAT	Icache parity error status
DcMd	DC_MODE	Dcache mode
DcPE	DC_PERR_STAT	Dcache parity error status

* Write-only registers whose values are obtained from a copy placed in PALtemp registers 1 and 2 when they were written with the **dc** command. Therefore, changing these internal CPU registers overwrites the saved contents of the general-purpose CPU registers 1 and 2 (also stored in PALtemp 1 and 2), affecting the output of the **pr** command.

Example

The following example shows the output of an **ec** command on a DP264 CPU:

```
SROM> ec
Cc      00000000.0000b78d
Va      00000800.7ffffffc
Vafo    00000002.001ffff8
ExCa    00000000.00203ee8
Ivaf    00000000.00000000
Ps      00000001.00000000
Ier     00000001.00000000
Ierc    00000001.00000000
Sirr    00000000.00000000
Isum    00000000.00000000
Excs    00000000.00001fc0
PalB    00000000.00000000
Ictl    00000000.03506386
Pctr    00000000.10000040
Ista    00000000.00000000
Asn     00000000.00000004
Aste    00000000.00000004
Astr    00000000.00000004
Ppce    00000000.00000004
Fpe     00000000.00000004
Mmst    00000000.000002c1
Dcst    00000000.00000000
```

3.4.10 em — Examine Memory

The examine memory (**em**) command reads data from one memory location.

Control Flags

!d (negate data flag)
ba (base address flag)
di (display flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)

Description

The examine memory command reads data from the specified memory location and displays it on the screen if the display (**di**) flag is enabled. Depending on the state of the quadword data flag, 32 or 64 bits will be displayed.

Example

The following example shows output from an **em** command:

```
SROM> di                ! Enable display mode.
Disp ON

SROM> qw                ! Reads are performed 64 bits at a time.
QW ON

SROM> lo                ! Repeat operation until a key is pressed.
Loop ON

SROM> em                ! Repeat read until a key is pressed.
A> 3fff80000
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
                        ! (key pressed)
```

3.4.11 fl — Display Flags

The flags (**fl**) command displays the current state of all flags.

Control Flags

Not applicable.

Description

The behavior of many of the mini-debugger commands can be affected by the state of one or more of eight available flags. The flags command displays on the screen the current state of these flags.

Example

In the following example, the default state of all flags, except the display flag, is off (disabled). Their states can be changed by issuing the appropriate command.

```
SROM> fl
FollowWr OFF
FollowRd OFF
BaseAddr OFF
Neg Data OFF
Wrt Addr OFF
Loop OFF
Disp ON
QW OFF

SROM> wa
Wrt Addr ON

SROM> !d
Neg Data ON

SROM> fl
FollowWr OFF
FollowRd OFF
BaseAddr OFF
Neg Data ON
Wrt Addr ON
Loop OFF
Disp ON
QW OFF
```

3.4.12 fm — Fill Memory

The fill memory (**fm**) command writes data to the specified range of addresses.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The fill memory command writes to a range or block of memory locations. The range is specified by first entering the starting address, followed by the ending address. The data to be written to the entire block can be entered by the user after the ending address, or it can be automatically generated by the command, depending on the state of the write address and negate data flags.

Example

In this example, the starting address (400003) will be truncated to the nearest quadword (400000) because the quadword data flag is on. The ending address (40004F) is truncated to 400048 for the same reason. Therefore, the last address written with the specified data is 400040. Similar truncations can be seen with the **bm** command.

```
SROM> qw
QW ON

SROM> fm
A> 400003
A> 40004F
D> 0123456789ABCDEF

SROM> bm
A> 3ffff7
A> 40005f
00000000.003ffff0: fff0ffff.fff0ff7d
00000000.003ffff8: fff0ffff.fff0ffff
00000000.00400000: 01234567.89abcdef
00000000.00400008: 01234567.89abcdef
00000000.00400010: 01234567.89abcdef
00000000.00400018: 01234567.89abcdef
00000000.00400020: 01234567.89abcdef
00000000.00400028: 01234567.89abcdef
00000000.00400030: 01234567.89abcdef
00000000.00400040: 01234567.89abcdef
00000000.00400048: 00000000.00000000
00000000.00400050: 00000000.00000000
```

The write address flag allows for unique data to be written to each memory location. This can be useful in debugging memory problems.

```
SROM> qw
QW OFF

SROM> wa
Wrt Addr ON

SROM> fm
A> 400000
A> 400020

SROM> bm
A> 400000
A> 400024
00000000.00400000:  00400000
00000000.00400004:  00400004
00000000.00400008:  00400008
00000000.0040000c:  0040000c
00000000.00400010:  00400010
00000000.00400014:  00400014
00000000.00400018:  00400018
00000000.0040001c:  0040001c
00000000.00400020:  89abcdef
```

Sometimes, it is useful to have alternating patterns written to adjacent memory locations. The negate data flag (**!d**) can be helpful then.

```
SROM> !d
Neg Data ON

SROM> fm
A> 400000
A> 400020
D> 0

SROM> bm
A> 400000
A> 400020
00000000.00400000: ffffffff
00000000.00400004: 00000000
00000000.00400008: ffffffff
00000000.0040000c: 00000000
00000000.00400010: ffffffff
00000000.00400014: 00000000
00000000.00400018: ffffffff
00000000.0040001c: 00000000
```

3.4.13 fr — Set Follow-with-Read Flag

The follow-with-read (**fr**) command toggles the follow-with-read flag, enabling or disabling the execution of a read operation after the last operation executed by a command.

Control Flags

fw (follow-with-write flag)

Description

The follow-with-read command is an advanced command whose use is required in certain situations where a second operation is needed to achieve a desired result, such as continuously creating a cache victim or having an I/O read operation interspersed between memory writes. Enabling the follow-with-read flag causes a prompt for an additional address from which the read operation is performed, unless the follow-with-write flag is also enabled. In this case, the additional address is used for the write operation and the read operation is performed from the first address.

The read operation takes place at the end of the command sequence, but after all read and write operations required by the current command have executed (including any write operations due to the follow-with-write flag being enabled). The data read may or may not be displayed, depending on the current command.

The default state is off.

Example

In the following example, a memory hierarchy composed of a 2MB direct-mapped, write-back, read-allocate cache and system memory is assumed. This may be the case in a 21064-based system where the Dcache is off and an external 2MB cache is on.

To continuously write the same address and guarantee that the data makes it to the system memory and is not simply cached, you must create a victim after each write operation. To create a victim, map the read address to the same cache index as the write operation.

For example, writing to address 0x600000 and reading from 0x800000 ejects the cached write of address 0x600000. From the system memory's point of view, a read of 0x800000 will be followed with a write of 0x600000, followed by another read of 0x800000 and so on until the loop is broken by pressing a key.

```
SROM> lo              ! Repeat command until a key is pressed.
Loop ON

SROM> fr              ! Follow deposit command with a read.
FollowRd ON

SROM> di              ! Don't display the data from the read.
Disp OFF

SROM> dm
A> 600000             ! Address of write.
A> 800000             ! Address of read; a multiple of 2MB.
D> 0f0f0f0f           ! Data for the write.
                      ! (key pressed)
```

To test the data path and cache timing between the CPU and the external cache, provide the same address for the read as the write. For example, each write operation in the following **fm** command is followed with a read operation from the external cache. Note that the read address is also incremented by the same amount as the write address.

```
SROM> lo                ! Don't repeat fill memory command.
Loop OFF

SROM> di                ! Print the data from the reads.
Disp ON

SROM> wa                ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fr                ! Follow deposit command with a read.
FollowRd ON

SROM> fm
A> 600000               ! Starting address of fill memory command.
A> 600020               ! Ending address of fill memory command.
A> 600000               ! Address to begin reading from.
00000000.00600000: 00600000
00000000.00600004: 00600004
00000000.00600008: 00600008
00000000.0060000c: 0060000c
00000000.00600010: 00600010
00000000.00600014: 00600014
00000000.00600018: 00600018
00000000.0060001c: 0060001c
```

During the execution of the **mt** command, it may be desirable for a read operation to I/O space to follow each write and read operation in the test. This could detect potential problems in the memory controller where memory and I/O accesses are handled. This command does not print the data from the read operation to I/O space.

```
SROM> di                ! Ignored by the memory test command.
Disp ON

SROM> wa                ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fr                ! Follow deposit command with a read.
FollowRd ON

SROM> mt
A> 500000               ! Start memory test at this address.
A> 600000               ! End memory test at this address.
A> 3fff80000            ! Follow with reads beginning at this address.
```

If the follow-with-write flag is also enabled, the additional address requested is used for this write operation and the read operation is performed from the first address. In the following example, two blocks of memory are filled with their own addresses and read operations from the first block are displayed after each write operation to the first block.

```
SROM> di                ! Ignored by the memory test command.
Disp ON

SROM> wa                ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fw                ! Follow with a write (before the read).
FollowWr ON

SROM> fr                ! Follow deposit command with a read.
FollowRd ON

SROM> fm
A> 500000               ! Begin to write at this address
A> 500020               ! and end with this one.
A> 600000               ! Write to this address range after each
                        ! write from first block.
00000000.00500000: 00500000        ! Reads from first block.
00000000.00500004: 00500004
00000000.00500008: 00500008
00000000.0050000c: 0050000c
00000000.00500010: 00500010
00000000.00500014: 00500014
00000000.00500018: 00500018
00000000.0050001c: 0050001c
```

3.4.14 fw — Set Follow-with-Write Flag

The follow-with-write (**fw**) command toggles the follow-with-write flag, enabling or disabling the execution of a write operation after the last operation executed by a command.

Control Flags

Not applicable.

Description

The follow-with-write command is an advanced command whose use is required in certain situations where a second operation is needed to achieve a desired result, such as continuously creating a cache victim or having an I/O write operation interspersed between memory read operations. Enabling the follow-with-write flag causes a prompt for an additional address to which the write will be performed.

The write operation takes place at the end of the command sequence, but before any read operations due to the follow-with-read flag being enabled. The data used for the write operation varies. If the write operation follows a read operation, then the data used comes from the read operation. If the write operation follows a write operation, then it uses the data supplied by the user or the data generated by the command (see the **wa** and **!d** commands).

The default state is off.

Example

In the following example, the **fm** command writes the block beginning at 0x400000 with the specified data pattern, because the follow-with-write (**fw**) flag is enabled; it also writes the same data to the block starting 0x500000. For the **bm** command, the write operation gets its data from the previously executed read operations, in effect, making this sequence a copy command.

```
SROM> fw        ! Follow the last operation with a write.
FollowWr ON

SROM> fm
A> 400000       ! First block gets written with data pattern.
A> 400010
A> 500000       ! Second block also gets written with the same data pattern.
D> abcdef

SROM> bm
A> 500000       ! Read from this block.
A> 500010
A> 600000       ! Write to this block.
00000000.00500000: 00abcdef
00000000.00500004: 00abcdef
00000000.00500008: 00abcdef
00000000.0050000c: 00abcdef
```

```
SROM> fw
FollowWr OFF

SROM> bm        ! This shows that a copy function has taken place.
A> 600000
A> 600010
00000000.00600000: 00abcdef
00000000.00600004: 00abcdef
00000000.00600008: 00abcdef
00000000.0060000c: 00abcdef
```

The following example is similar to the previous one, except that the write address (**wa**) flag has been set. With **wa** enabled, the address is for the read data; the data is for the write data.

```
SROM> fw        ! Follow the last operation with a write.
FollowWr ON

SROM> fm
A> 400000       ! First block gets written with data pattern.
A> 400010
A> 500000       ! Second block also gets written with the same data pattern.
D> abcdef

SROM> wa    ! Write address flag is enabled.
Wrt Addr ON

SROM> bm
A> 500000       ! Read from this block.
A> 500010
A> 600000       ! Write to this block.
00000000.00500000: 00000000.00600000
00000000.00500008: 00000000.00600008

SROM> wa
Wrt Addr OFF
```

3.4.15 lo — Set Loop Flag

The loop (**lo**) command enables or disables looping or repeating of a command by toggling the loop flag.

Control Flags

Not applicable.

Description

When the loop flag is enabled, examine and deposit commands are repeated until you stop the looping by pressing a key. This feature is useful for hardware timing of read and write operations to the external cache or memory system, or anytime a command needs to be repeated continuously.

The default state is off.

Example

In the following example that shows the **dm** command, the data pattern is written to address 500000 continuously until a key is pressed. The **em** command reads and displays the data from address 500000 continuously until a keystroke is detected.

```
SROM> lo
Loop ON

SROM> dm
A> 500000
D> 12345678
                  ! (key pressed)

SROM> em
A> 500000
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
                  ! (key pressed)
```

The **mt** command continuously checks the range between 400000 and 500000.

```
SROM> wa
Wrt Addr ON

SROM> mt
A> 400000
A> 500000
!(key pressed)
```

3.4.16 mt — Memory Test

The memory test (**mt**) command performs a simple write-read-compare test.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The memory test command first writes to the memory range specified with the user-specified data or command-generated data. It then begins reading back the data and comparing it against what was written. If a mismatch is encountered, both the read and the expected data are displayed on the screen.

Example

In the following example, the data pattern 12345678 is written to the block starting at 400000 and ending at 600000. The range is then read back and compared with the expected data 12345678. In this example, no mismatches are found.

```
SROM> mt
A> 400000
A> 600000
D> 12345678
```

In this next example, a bit stuck high is detected by the memory test.

```
SROM> wa              ! Use address of write as the data.
Wrt Addr ON

SROM> mt
A> 400000
A> 500000
00000000.00400000:10400000 ! The data read back has an extra bit set.
Expect: 00400000
00000000.00400004:10400004
Expect: 00400004
00000000.00400008:10400008
Expect: 00400008
00000000.0040000c:1040000c
Expect: 0040000c
00000000.00400010:10400010
Expect: 00400010
00000000.00400014:10400014
Expect: 00400014
                          ! A key was pressed.
```

3.4.17 pr — Print Register

The print register (**pr**) command displays the contents of the general-purpose CPU registers.

Control Flags

Not applicable.

Description

The contents of the general-purpose CPU registers are saved to PALtemp registers when the mini-debugger is first entered. The **pr** command allows for their viewing. Note that because the **dc** command uses two of these PALtemp registers, the results in R1 and R2 will be changed after a **dc** command.

Note: This command may not be available in all standard SROMs supplied with the Alpha motherboards due to a lack of space. If this is the case, recompile the mini-debugger source files supplied in the Alpha SDK with the FULL_MDBG compile switch defined to enable this command.

Example

```
SROM> pr
R00:    00000000.00000000
R01:    00000000.00000006
R02:    000007f8.00000000
R03:    00000000.00000030
R04:    00000000.00000000
R05:    00000000.00000000
R06:    00000000.00300000
R07:    00000000.00001428
R08:    00000000.02000000
R09:    00000000.00000004
R10:    00000000.00026890
R11:    00000000.000262a0
R12:    00000000.00026560
R13:    0000004e.2001c665
R14:    00000000.00026590
R15:    00000000.000265a0
R16:    00000000.00026580
R17:    00000000.00008000
R18:    00000080.00000080
R19:    00000000.01ffdf30
R20:    ffffffff.fffdffff
R21:    00000000.00300000
R22:    0048484c.4c4e4e4e
R23:    00000000.0032edec
R24:    00000000.00000000
R25:    00000000.01ffdd25
R26:    00000000.00011fa8
R27:    00000000.00200080
R28:    00000000.0007a900
R29:    00000000.0007ce50
R30:    00000000.01ffdf30
```

3.4.18 qw — Set Quadword Data Flag

The quadword (**qw**) command enables or disables the quadword data flag.

Control Flags

Not applicable.

Description

The quadword command changes the state of the quadword data flag. When the quadword data flag is enabled, all operations are performed on 64-bit data. When off, only 32 bits of data are used.

The default state is off.

Example

In this example, the first **bm** command loads 32 bits of data using the **ldl/p** instruction from addresses 3.FFF80000, 3.FFF80004, 3.FFF80008, and 3.FFF8000C. After quadword mode is enabled, the same command performs only two 64-bit loads of data using the **ldq/p** instruction from addresses 3.FFF80000 and 3.FFF80008.

```
SROM> bm
A> 3fff80000
A> 3fff80010
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> qw
QW ON

SROM> bm
A> 3fff80000
A> 3fff80010
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80008: 00006579.00000038
```

3.4.19 rt — Return to Calling Program (Exit Mini-Debugger)

The return (**rt**) command exits the mini-debugger.

Control Flags

Not applicable.

Description

The return command exits the mini-debugger, returning to the calling program if there is one. For the standalone mini-debugger, it simply exits.

Returning from a mini-debugger built into the SROMs provided with the motherboards allows you to continue with the booting process, as if the mini-debugger had not been invoked.

The **rt** command is also useful during debugging of SROM code you may write. You can place calls to the mini-debugger throughout the SROM code to act as breakpoints. You can then examine the system and CPU state at those points and return to the SROM code to continue its execution.

Example

An example of the **rt** command follows:

```
SROM> rt
```

3.4.20 sb — Set Base Address

The set base (**sb**) address command defines the base address to be added to user-specified addresses when the base address flag is enabled.

Control Flags

Not applicable.

Description

Using a base address is convenient when examining or depositing of the same address frame is needed (particularly when the high-order bits of the address must be set). The base address entered with the set base command is added to the addresses entered in any subsequent examine or deposit commands, saving the user some typing. This command always enables the base flag.

Example

To access a flash ROM part that responds to addresses 3.FFF80000 through 3.FFFFFFFF, it may be more convenient to enter its base address and work with offsets, rather than typing the absolute address every time.

```
SROM> sb
A> 3fff80000
00000003.fff80000
BaseAddr ON

SROM> bm
A> 0
A> 10
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> ba
00000003.fff80000
BaseAddr OFF

SROM> em
A> 400000
00000000.00400000: 00400000

SROM> ba
00000003.fff80000
BaseAddr ON

SROM> bm
A> 30
A> 40
```

```
00000003.fff80030: 00000001
00000003.fff80034: 0000be8a
00000003.fff80038: 47ff041f
00000003.fff8003c: 47ff041f
```

3.4.21 st — Start Image Execution

The start image (**st**) command begins execution at a specified address.

Control Flags

Not applicable.

Description

The start image command transfers control to the code residing 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 and eliminates your ability to return to the mini-debugger.

Example

The **st** command begins execution of the uploaded image.

```
SROM> xm
A> 400000
D> 10

SROM> st              ! Address to start execution at.
A> 400000
```

3.4.22 wa — Set Write Address Flag

The write address (**wa**) command enables or disables the write address flag.

Control Flags

!d (negate data flag)

Description

When the write address flag is enabled, write operations use the destination address as their data. This allows for unique values to be written to individual locations (a useful feature when debugging the memory subsystem). If the negate data flag is enabled, the one's complement of the address being written is used instead. Commands that request data to be written will not do so if this flag is enabled.

The default state is off; data used by deposit commands is provided by the user.

Example

In the following example, after enabling the write address flag, the region between 400000 and 400020 is filled with its own address in 32-bit increments. With **qw** mode on, deposits are performed in quadword increments, and the data written changes in 64-bit increments.

```
SROM>  wa
Wrt Addr ON

SROM> fm
A> 400000
A> 400020

SROM> bm
A> 400000
A> 400020
00000000.00400000: 00400000
00000000.00400004: 00400004
00000000.00400008: 00400008
00000000.0040000c: 0040000c
00000000.00400010: 00400010
00000000.00400014: 00400014
00000000.00400018: 00400018
00000000.0040001c: 0040001c

SROM> qw
QW ON

SROM> fm
A> 800000
A> 800030
```

```
SROM> bm
A> 800000
A> 800030
00000000.00800000: 00000000.00800000
00000000.00800008: 00000000.00800008
00000000.00800010: 00000000.00800010
00000000.00800018: 00000000.00800018
00000000.00800020: 00000000.00800020
```

With the negate data flag enabled, the complement of the address is used instead.

```
SROM> qw
QW OFF

SROM> !d
Neg Data ON

SROM> fm
A> 500000
A> 500020

SROM> bm
A> 500000
A> 500020
00000000.00500000: ffafffff
00000000.00500004: ffaffffb
00000000.00500008: ffaffff7
00000000.0050000c: ffaffff3
00000000.00500010: ffafffef
00000000.00500014: ffafffeb
00000000.00500018: ffafffe7
00000000.0050001c: ffafffe3
```

3.4.23 xb — External Boot

The external boot (**xb**) command begins execution of the uploaded image.

Control Flags

Not applicable.

Description

The external boot command is used after an image has been uploaded with the **xm** command. It adds the necessary Icache flush code before it transfers control to the code residing at the address specified in the **xm** command. If no **xm** command has been executed, then this command has no effect.

Example

The following example begins execution of the uploaded image:

```
SROM> xm
A> 400000
D> 10

SROM> xb              !  Execution of code at 400000 begins.
```

3.4.24 xm — External Image to Memory

The external image to memory (**xm**) command loads an image into memory.

Control Flags

Not applicable.

Description

The external image to memory command loads an external image into memory. The first input is the destination address, followed by the number of bytes to load. The number of bytes should be a multiple of 8 and the starting address should be quadword aligned. Normally, this command is used by an external utility called uload.exe (for Tru64 UNIX) or xload.exe (for Windows NT).

Examples

This section contains three examples:

- Uploading an image with the **xm** command
- Downloading the Debug Monitor with the Tru64 UNIX uload utility
- Downloading the Debug Monitor with the Windows NT xload utility

Upload an Image with xm

You can upload an image by typing the starting address and the number of bytes to upload. In this example, 16 bytes are copied from the SROM port to address 400000.

```
SROM> xm
A> 400000
D> 10
```

Download the Debug Monitor with Tru64 UNIX uload

The following example shows how you can use the uload (Tru64 UNIX) utility, which in turn uses the **xm** command to download the Debug Monitor. You can follow this example if the flash has been erased accidentally.

```
! AlphaPC 264DP Log.
!
! If our flash has been erased and we want to reload
! and run the Debug Monitor, this session will load the
! code through the Debug Monitor and run it.


  tip tty00 -115200            !Connect to serial port tty00
  connected (Enter)             !and set baud rate.
  21264...V00002501
  SROM> ~.                       !Disconnect from tip session.
[EOT]
```

```
uload dp264dbm.cmp 300000 /dev/tty00 115200
                            !Load Debug Monitor at address
                            !300000, using serial port tty00, at
                            !baud rate 115200.

          **** ULOAD V2.0 [DEC 17 1997] ****
          **** SROM Binary Load Utility ****

Retrieving image to upload: dp264dbm.cmp
Size of file 0x31300 (201472) bytes
Connecting to seral port: /dev/tty/00
Syncing with remote terminal...
U
Error:                       !Error condition exists only
                             !until correct baud rate is found.

SROM>
Sending XM command.          !Uload sends XM command.
Destination = 0x300000, Size = 0x31300 (201472) bytes

  Bytes   Time
    0     00:00              !Uload finishes and exits.

SROM>
tip tty00 -115200          !Tip back to mini-debugger.
connected (Enter)


SROM> st                  !Starts Debug Monitor image execution at
A> 300000                !address 300000 and overwrites Icache.
```

Download the Debug Monitor with Windows NT xload

The following example shows how you can reload and run the Debug Monitor through the SROM Mini-Debugger Binary Load utility. You can follow this example if the flash has been erased accidentally.

```
! AlphaPC 264DP Log.
!
! If our flash has been erased and we want to reload
! and run the Debug Monitor, this session will load the
! code through the Debug Monitor and run it. Perform the following steps:
!
!
! 1. If you wish, you can open a hyperterminal to verify the
!    connection. To do this, open the hyperterminal on a Windows NT
!    system, with the baud rate  set to 19200, and flow control
!    set to none. You are now ready to start your SROM mini-debugger
!    session.
!
```

```
! 2. Power on your system, which displays the following:

     DP264...V00005201.01.000000041ac60030.02.000007d1.03.0373bef8c1.05.04..
     0608000000.14.17
                                  !Stops here until you press enter.

     (Enter)

     21264...V00002501
     SROM>

! 3. Stop the hyperterminal session by closing the window on the Windows
!    NT system. Open a command prompt window and go to the
!    directory where the file dp264dbm.cmp is located.

     C:\>cd ebsdk\ebfw\boot\dp264
     C:\ebsdk\ebfw\boot\DP264>

! 4. Before trying to XLOAD, ensure that xload.exe is in your path.

     C:\ebsdk\ebfw\boot\DP264>xload dp264dbm.cmp 300000 com1 fast
     *** Flamingo/Sandpiper/Pelican/Morgan SROM XLOAD Utility V2.1 ***

     %XLOAD-I-CMDLIN SROM> XM...
     %XLOAD-I-PAGSNT, X loaded page 315

     C:\ebsdk\ebfw\boot\DP264>

! 5. After XLOAD completes, reconnect the hyperterminal connection and
!    press the ENTER key to establish communication with the
!    SROM mini-debugger. Then type the following command:

     SROM>xb

!    This runs the Debug Monitor on your AlphaPC 264DP, through which
!    you can flash a new Debug Monitor or AlphaBIOS.

! 6. After the image is flashed into the system, replace your SROM
!    mini-debugger with the production SROM.
```

A
Support

A.1 Customer Support

The Alpha OEM website provides the following information for customer support.

URL	Description
http://www.digital.com/alphaoem	Contains the following links: • **Developers' Area:** Development tools, code examples, driver developers' information, and technical white papers • **Motherboard Products:** Motherboard details and performance information • **Microprocessor Products:** Microprocessor details and performance information • **News:** Press releases • **Technical Information:** Motherboard firmware and drivers, hardware compatibility lists, and product documentation library • **Customer Support:** Feedback form

A.2 Alpha Documentation

The following table lists some of the available Alpha documentation. You can download Alpha documentation from the Alpha OEM World Wide Web Internet site:

http://www.digital.com/alphaoem

Click on **Technical Information**.
Then click on **Documentation Library**.

Title	Order Number
Alpha Architecture Reference Manual[1]	EY–W938E–DP
Alpha Architecture Handbook	EC–QD2KC–TE
Alpha 21164 Microprocessor Hardware Reference Manual	EC–QP99C–TE
Alpha 21164 Microprocessor Data Sheet	EC–QP98C–TE

Title	Order Number
21164PC Alpha Microprocessor Hardware Reference Manual	EC–R2W0A–TE
AlphaPC 264DP Product Brief	EC–RBD0A–TE
AlphaPC 264DP User's Manual	EC–RB0BA–TE
AlphaPC 264DP Technical Reference Manual	EC–RB0DA–TE
AlphaPC 164SX Motherboard Product Brief	EC–R57CA–TE
AlphaPC 164SX Motherboard Windows NT User's Manual	EC–R57DB–TE
AlphaPC 164SX Motherboard DIGITAL UNIX User's Manual	EC–R8P7B–TE
AlphaPC 164SX Motherboard Technical Reference Manual	EC–R57EB–TE
AlphaPC 164LX Motherboard Product Brief	EC–R2RZA–TE
AlphaPC 164LX Motherboard Windows NT User's Manual	EC–R2ZQF–TE
AlphaPC 164LX Motherboard Tru64 UNIX User's Manual	EC–R2ZPC–TE
AlphaPC 164LX Motherboard Technical Reference Manual	EC–R46WC–TE
Alpha Motherboards Software Developer's Kit Product Brief	EC–QXQKD–TE
Alpha Motherboards Software Developer's Kit Read Me First	EC–QERSJ–TE
Alpha Microprocessors Motherboard Debug Monitor User's Guide	EC–QHUVG–TE
Alpha Microprocessors Motherboard Software Design Tools User's Guide	EC–QHUWE–TE

[1] Not available on website. To purchase the *Alpha Architecture Reference Manual*, contact your local sales office or call Butterworth-Heinemann (DIGITAL Press) at 1–800–366–2665.

Index

B

C

D

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

W

X