Alpha Microprocessors Motherboard Debug Monitor

User's Guide

Order Number: EK–AMEBD–UG. A01

Revision/Update Information: This is a new manual.

Digital Equipment Corporation
Maynard, Massachusetts

April 1996

Contents

3 Remote Debugging

4 Debug Monitor User Commands

Tables

Preface

Introduction

This document describes the software features of an Alpha microprocessor board. The board software is intended to provide software monitor and debug capabilities to customers who use an Alpha microprocessor board as a development platform for creating their own Alpha microprocessor-based systems.

Audience

This document is for anyone who develops software or hardware to be used with an Alpha microprocessor. The Alpha Microprocessors Board Debug Monitor supports the following products:

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

Content Overview

The information in this document is organized as follows:

- Chapter 1 is an introduction to the board debug monitor.
- Chapter 2 describes how to use this debug monitor.
- Chapter 3 describes how to use remote debugging.
- Chapter 4 lists all debug monitor commands.
- Appendix A contains information about technical support services and associated documentation.

Conventions

The following conventions are used in this document:

Convention	Definition
A percent sign (%)	Indicates a Digital UNIX operating system command prompt.
A pound sign (#)	Indicates a Digital UNIX superuser prompt and indicates that these commands are performed from the root directory level.
Square brackets ([ ])	Denote optional syntax.
Boldface type	Indicates debug monitor command text.
EB*xxx*>	Indicates a debug monitor command prompt.
Italic type	Emphasizes important information, indicates variables in command syntax and complete titles of documents.
`Monospaced type`	Indicates an operating system command, a file name, or a directory path name.

All numbers are decimal unless otherwise indicated. Where there is ambiguity, numbers other than decimal have a subscript indicating their base.

1
Introduction

1.1 Overview

The Alpha Microprocessors Board Debug Monitor can be used to load code into the system and perform other software debug functions, such as memory read/write and instruction breakpointing. Combined with a hardware interface, the debug monitor can be used to write and debug software, such as device drivers, for workstation and PC-type products as well as for embedded control products, such as laser printers, communication engines, and video products.

The debug monitor is provided with the board in a a 1MB flash ROM. You can develop your code on a host system and load the software into the board through a serial port, Ethernet port, user-supplied floppy drive, or the extra ROM socket.

1.2 General Features

The debug monitor offers the ability to

- Download files via serial and Ethernet ports, ROM socket, and user-supplied floppy drive.
- Examine and deposit the board system register, CPU internal processor registers (IPRs), and I/O mapped registers.
- Examine and modify DRAM and I/O mapped memory.
- Disassemble CPU instructions in memory.

- Transfer control to programs loaded into memory.
- Perform native debugging, including breakpoints and single stepping.
- Perform full source-level debugging using the Digital Ladebug debugger for Digital UNIX running on a remote host that communicates through an Ethernet connection.

1.3 System Requirements

The recommended host system for software development is an Alpha system running the Windows NT or Digital UNIX operating system. Alpha hardware is the platform upon which the initial set of portable development tools is provided. The native Digital UNIX and Windows NT software development tools are used in conjunction with the portable tools.

The Digital UNIX operating system also supports the bootstrap protocol (BOOTP) for downloading executable images to the board and ladebug for remote debugging. The examples in this manual that pertain to a host system are based on Alpha hardware running the Digital UNIX operating system.

2

Getting Started

2.1 Overview

This chapter describes how to set up your board and host system.

2.2 Hardware and Operating System Requirements

The minimum configuration you need to use your board is a power supply and a terminal. However, in order to take full advantage of the board, you need Alpha hardware running the Windows NT or Digital UNIX operating system.

2.3 Configuring Your System

This section describes how to connect your board to the following:

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

2.3.1 Connecting the Board to a Terminal

To connect the board to a terminal, connect the terminal communication line to serial port 1 of the board. Your terminal should be set to match the baud rate of the board. The default speed of the serial port is 9600 baud.

After the terminal and the board are connected and the board is powered on, the terminal screen should display the banner and prompt. For example:

```
DECchip 21064 Evaluation Board (EB64) Debug Monitor
 Version:   Tue May 04 16:55:54 EDT 1993
 Bootadr: 0x100000,  memSize:  0x2000000 (32MB)

EB64>
```

2.3.2 Connecting the Board to a Personal Computer

Communication (terminal emulation) software running on a PC can also be used to communicate with the board. To connect the board to a PC, connect the terminal communication line to serial port 1 of the board as described for the terminal.

2.3.3 Connecting to the Board from a System Running Windows NT

A system running the Windows NT operating system supports serial communication with the board. To configure a COM port, follow these steps:

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

Terminal Setting	Value
Data bits	8 bit
Transmit/receive speed	9600 baud
Character format	No parity
Stop bits	1

Save these settings in a file. For example, settings for the EB64 could be saved in a file called `eb64.trm`.

For consistency, all examples and command descriptions assume that the board serial port 1 is connected to COM1.

2.3.4 Connecting to the Board from a System Running Digital UNIX

Digital UNIX supports serial communications and Ethernet communications with the board.

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

- /dev/tty00
- /dev/tty01

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

To enable these ports for use with the board, 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 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 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
   ```

2.3.4.1 Connecting to a Serial Port Under Digital UNIX

After you modify the /etc/remote and /etc/inittab files, you can connect to the serial port under the Digital UNIX operation system using the Digital UNIX tip command. If the connection is successful, the board prompt displays, and you are ready to use the debug monitor **load** or **boot** commands to download your file. For example,

```
% tip port_name0
EB64>  load
Send File now ...
```

Type ~> to cause the Digital UNIX tip command to send the file to the board.

2.3.4.2 Setting Up the Host System as a BOOTP Server

The bootstrap protocol (BOOTP) needs to be defined so that the commands **netload** and **netboot** work correctly. To set up a Digital UNIX system as a BOOTP server, follow these steps:

1. Modify the /etc/inetd.conf file. This file enables both the BOOTP and the TFTP daemons. The TFTP daemon is required by the BOOTP daemon.

   a. Add the following line to specify the directories that can be accessed by the TFTP daemon:

      ```
      tftp dgram udp wait root /usr/sbin/tftpd tftpd /directory1 /directory2
      ```

      If no directory is specified, all files with public access can be accessed by the TFTP daemon.

   b. To start the BOOTP daemon, enter the following line:

      ```
      bootps  dgram udp wait root /usr/sbin/bootpd bootpd -d -d -d
      ```

2. If BOOTP is already running on your system, you want to stop it. To stop BOOTP, enter the following commands:

   ```
   # ps uax | grep bootpd
   # kill -KILL process_id_number
   # ps uax | grep inetd
   # kill -HUP process_id_number
   ```

3. To restart BOOTP, enter the following command:

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

   The changes made to the /etc/inetd.conf file will now take effect.

4. Modify the /etc/bootptab file to specify the Ethernet hardware address of the board and the IP address assigned to that node. See your network administrator to obtain an IP address. Refer to the literature supplied with your Ethernet card to obtain information about the hardware address. If the hardware address is accessible through software, you can use the **einit** command to display it. For example, the following lines modify this file for the EB64:

   ```
   remote_system_name0:ht=ethernet:ha=BA9876543210:ip=16.123.45.67:\
   :hd=/directory1:bf=filename:vm=auto:
   ```

BOOTP checks this file to see if it has changed each time it receives a request. If it has changed, the new file is read. The *directory* and *filename* are the defaults for the **netload** and **netboot** commands. If no argument is specified with either command, the file loaded is */directory1/filename*.

Verify the BOOTP Server

To verify that the BOOTP server has been set up properly, you can look at the daemon.log file. This file shows directories accessed for the **netload** or **netboot** commands.

```
# tail -f /var/adm/sylog.dated/dated_dir/daemon.log
```

The following example displays a boot request from an example daemon log file:

```
May  5 10:40:28 eval bootpd[328]: request from hardware address BA9876543210
May  5 10:40:28 eval bootpd[328]: found: eb64 (BA9876543210) at (16.123.45.67)
May  5 10:40:28 eval bootpd[328]: file /users/eval/boot/size.eb64 not found
May  5 10:40:28 eval bootpd[328]: vendor magic field is 0.0.0.0
May  5 10:40:28 eval bootpd[328]: sending RFC1048-style reply
```

You can refer to the Digital UNIX man pages for more information about bootp, bootpd, tftp, tftpd, inet, inetd, and init.

2.3.4.3 Setting up the Host System as a Ladebug Client

The debug monitor supports remote debugging for Digital UNIX host systems with Ladebug. The Ladebug software does not accept numeric internet addresses. You can give your board an internet name in the /etc/hosts file. In the /etc/hosts file, the format is the internet protocol (IP) address followed by the host system name. For example:

```
12.345.67.89    remote_system_name0
```

2.4 Updating the Debug Monitor Firmware

The firmware is stored in a flash ROM. You can update the debug monitor firmware with an update utility. The firmware update utility (`fwupdate.exe`) is located on the Debug Monitor Firmware diskette. The way in which you invoke the utility depends on the firmware console you are updating.

2.4.1 Updating Firmware in a Flash ROM

The update utility is used to add, update, or reload the firmware in the Flash ROM. You can access the utility using Windows NT, the debug monitor, or the Alpha SRM Console Version V4.4–1 or higher.

To update the flash ROM, the update enable/disable jumper must be in the enable position, which is the default. See your board user's guide for more information about jumper positions.

Use the following table to determine the update procedure for your board:

If your system is running the...	Then see this section...
Windows NT Firmware	Updating the Flash ROM from Windows NT Firmware
Debug monitor firmware	Updating the Flash ROM from the Debug Monitor
Alpha SRM Console firmware	Updating the Flash ROM from the Alpha SRM Console

2.4.2 Updating the Flash ROM from Windows NT Firmware

If you have set auto-boot, you will need to trap the procedure before the Windows NT operating system begins to boot.

Windows NT Firmware Conventions

The currently selected option in a Windows NT Firmware menu is highlighted. To select and choose different options in the menus, use the following keys:

Key	Description
Arrow	The arrow keys are used to select different options.
Enter	The Enter key is used to choose the highlighted option.
Escape	The escape key is used to close a menu or cancel an operation.

When you power up your board system, the firmware displays a blue screen on the monitor and initializes the firmware drivers. If autoboot is enabled, cancel

it by pressing the escape key before the timeout period expires. This allows you to interact with the firmware.

Starting the Firmware Update Utility

To invoke the firmware update utility to update the firmware in a flash ROM, follow this procedure:

1. Insert the Debug Monitor Firmware diskette into drive A and verify that the CD–ROM drive does not contain a compact disc.
2. Restart your board system.
3. From the Boot menu, choose **Supplementary menu...**.
4. From the Supplementary menu, choose **Install new firmware**.

   The firmware update utility will reinitialize some system components; this may appear as if your system is restarting.
5. Proceed to the Using Firmware Update section, and follow the procedures.

2.4.3 Updating the Flash ROM from the Debug Monitor

The firmware update utility can be invoked by the debug monitor directly from the diskette.

Starting the Firmware Update Utility

To start the firmware update utility using the debug monitor, follow this procedure:

1. Insert the firmware update diskette into drive A and verify that the CD–ROM drive does not contain a compact disc.
2. At the debug monitor prompt, enter the following:

   EB*xxx* **fwupdate**

   Where EB*xxx* is:

   For Alpha PCI 64-275: EB64

   For Alpha PCI 164-266/300: EB164

   The firmware update utility will reinitialize some system components. This may appear as if your system is restarting.

2.4.4 Updating the Flash ROM from the Alpha SRM Console

The firmware update utility can be invoked by the Alpha SRM Console firmware directly from a diskette.

Starting the Firmware Update Utility from a Diskette

The firmware update utility is provided on a diskette. To start the firmware update utility from the diskette, follow this procedure:

1. Insert the firmware update diskette into drive A and verify that the CD–ROM drive does not contain a compact disc.
2. At the Alpha SRM Console prompt, enter the following:

   ```
   >>> fwupdate
   ```

   The firmware update utility will reinitialize some system components; this may appear as if your system is restarting.
3. Proceed to the Using Firmware Update section, and follow the procedures.

2.4.5 Using Firmware Update

To use firmware update, follow this procedure:

1. Observe the initialization messages being displayed on the terminal attached to the COM1 serial port or on the graphics display unit from starting the firmware update utility. When the message `Initializing Flash Driver` is displayed, press the A key to enable the Advanced menu.
2. From the Advanced menu, choose **Update Debug Monitor**.
3. When the system prompts you to continue the update, choose **Yes**.
4. If the console selection does not match the firmware you flashed, you will be prompted to update the console selection. If you are prompted to update the console selection, use the arrow keys to choose **Yes**.
5. When the update has completed, restart your board system.

2.5 Switching to the Debug Monitor Firmware

The following procedures describe how to switch from the Windows NT Firmware and the Alpha SRM Console firmware to the debug monitor firmware.

Switching from Windows NT Firmware

Use the following procedure to switch to the debug monitor firmware from the Windows NT Firmware:

1. From the Boot menu, choose **Supplementary menu...**.
2. From the Supplementary menu, choose **Set up the system...**.
3. From the Setup menu, choose **Machine specific setup...**.
4. From the Machine specific setup menu, choose **Switch to Debug Monitor**.
5. Restart your board system.
6. Observe the debug monitor prompt on the terminal attached to the COM1 serial port and on the graphics display unit.

Switching from the Alpha SRM Console Firmware

To switch to the debug monitor firmware from the Alpha SRM Console Firmware, follow this procedure:

1. At the Alpha SRM Console prompt, enter the following command:

   ```
   >>> deposit -b toy:3f 0
   ```

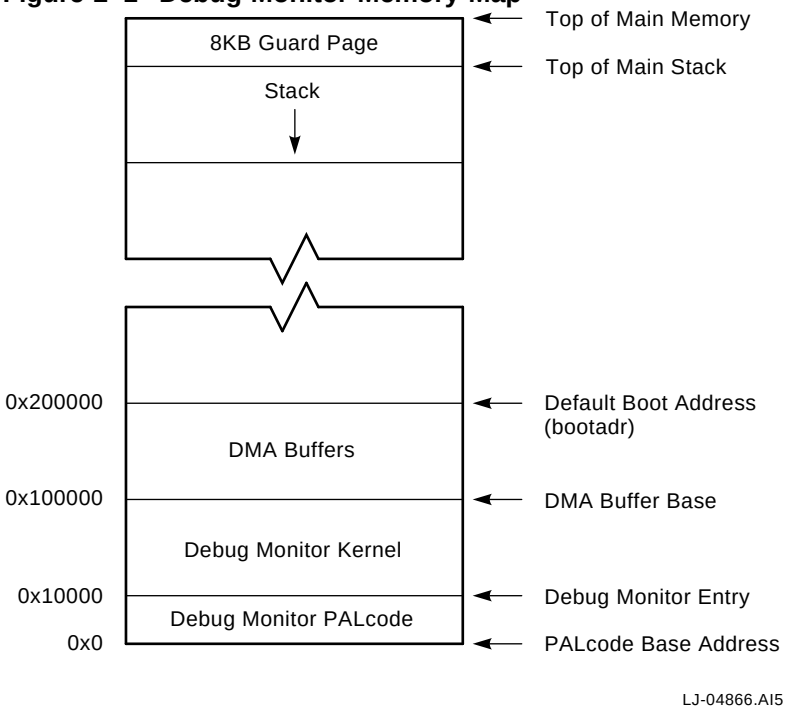
2. Restart your board system.
3. Observe the debug monitor prompt on the terminal attached to the COM1 serial port and on the graphics display unit.

2.6 Debug Monitor Memory Map

The debug monitor image is loaded from the system ROM into memory at physical address 0 by the SROM initialization code. At startup, the debug monitor determines the amount of memory present in the board based on parameters that are passed in from the SROM initialization code. One of these parameters determines the *Top of Main Memory*. Refer to your board user's guide for more information about the SROM initialization code and supported memory configurations.

Figure 2–1 shows the basic outline for the debug monitor memory map.

Figure 2–1 Debug Monitor Memory Map



The debug monitor image consists of PALcode at physical address zero and the debug monitor kernel at physical address 10000_{16}. After loading the image into memory, the SROM initialization code begins execution of the image in PALmode at the PALcode base address. The PALcode used in the debug monitor was designed to support Digital UNIX (formerly DEC OSF/1) and was later adapted to the debug monitor. Refer to the *Alpha AXP Architecture Reference Manual* and the *PALcode for Alpha Microprocessors System Design Guide* for more information about Digital UNIX PALcode.

2.6.1 Stack

PALcode starts execution of the debug monitor kernel at physical address 10000_{16}. Upon entry to the debug monitor kernel, the debug monitor establishes the initial stack pointer at the first 8-kilobyte boundary below the top of main memory. From there the stack grows downward.

2.6.2 DMA Buffers

Various devices used with the board require direct memory access (DMA). The device drivers provided in the debug monitor for these devices are designed to perform their DMA within a 1-megabyte range starting at 1 megabyte (physical address 100000_{16}). At startup, the debug monitor initializes the I/O subsystem with DMA windows that include this range. The device drivers included with the debug monitor that require DMA are the Ethernet and floppy drivers. Although the **ebuff** command can be used to change the base of the Ethernet buffers, the buffers must remain within this 1-megabyte window.

2.7 Downloading Files

The board supports loading files into memory from a serial port, the Ethernet, and a diskette. The user can either load the file into memory, or load and execute the file in a single step. The next table shows the commands for the specific I/O devices. See Chapter 4 for more details about these commands.

I/O Device	Use this Command to Load into Memory...	Use this Command to Load into Memory and Execute...
ROM socket	romload	romboot
Serial port	load	boot
Ethernet	netload	netboot
Diskette	flload	flboot

The default boot address (**bootadr**) is 200000_{16}. However, you can change the default boot address with the **bootadr** command. The new setting is then stored in the battery-backed RAM.

2.8 Execution Commands

After your program is loaded, you are ready to execute it. If the command loads and executes a program, you may want to re-execute the program during the board session. The board debug monitor has two commands to execute programs: **go** and **jtopal**. See Chapter 4 for more details about these commands.

2.9 Resetting the Debug Monitor

If the software hangs the board, then the hardware reset on the board can be used to reset to the debug monitor command line. For information about connecting the reset signals, see your board user's guide.

3
Remote Debugging

The debug monitor supports remote debugging for Digital UNIX host systems with Ladebug. The Ladebug software provides the full source-level debugging capabilities of most programs that run on the board, including the debug monitor.

This chapter describes some debugging hints for use with the debug monitor and the remote debugger. This chapter also describes the guidelines for writing programs that allow you to take full advantage of remote debugging.

3.1 What is a Debugger?

A debugger is a tool that helps you locate run-time programming errors or bugs. You use the debugger on executable programs created when a program has been compiled and linked successfully.

3.2 What is a Remote Debugger?

A remote debugger is a tool that helps you locate run-time programming errors or bugs in a program running on a remote system. The remote system can be a system that cannot support a full programming environment by itself. You use a remote debugger on executable programs compiled and linked for the remote system.

3.3 Remote Debug Server

The debug monitor's remote debug server (the part of the monitor that communicates with Ladebug) uses interrupts and an Ethernet device. Interrupts are used by the debug monitor to poll the Ethernet device for messages from Ladebug. Any program that changes the interrupt handler must tell the debug server when to poll the Ethernet.

3.4 Programming Guidelines for Remote Debugging

The following sections describe the programming guidelines for remote debugging.

3.4.1 The Run-Time Environment

When a program is started by the debug monitor's **go** command, it is started at the appropriate IPL to enable real-time clock interrupts (usually IPL 4). If the program does not install its own interrupt handler, then the debug monitor will handle all interrupts. If a program does install its own interrupt handler using the Write System Entry Address PAL call, then it must be prepared to handle all interrupts as described in the following sections. When a program completes normally, the debug monitor re-installs its own interrupt handler.

3.4.2 Types of Programs

For the purposes of this chapter, programs may be classified into three types. These are:

- Programs that do not use the Ethernet or do not include their own interrupt handler.
- Programs that include their own interrupt handler.
- Programs that make use of the Ethernet.

3.4.2.1 Programs that Do Not Use the Ethernet or Include their Own Interrupt Handler

There is only one restriction for programs that do not use the Ethernet and that use the debug monitor interrupt handler:

- Do not disable the real time clock interrupt and the Ethernet interrupts for long periods.

Long delays may cause Ladebug to think that there is a problem with the Ethernet link to the target. If network delays are insignificant, Ladebug will tolerate periods of up to 10 seconds with interrupts disabled, although it will normally warn the user of possible network problems if interrupts are disabled for more than a second. Ethernet interrupts are disabled at IPL 3 or more, and real-time clock interrupts are disabled at IPL 5 or more. Writing to the control registers of the Ethernet device or to the real-time clock can also disable the interrupts. It is possible to set breakpoints or to single step uninterruptable code. There is no restriction on the time that can be spent at the breakpoint.

3.4.2.2 Programs that Include their Own Interrupt Handler, but Do Not Use the Ethernet

Programs that define or install their own interrupt handler must ensure that the debug monitor polls the Ethernet device often enough to receive all the messages sent to it by Ladebug. An easy way to do this is to use the `ladbx_poll` function. When this function is called, the following occurs:

- All frames that have been received on the Ethernet device are read.
- All remote debug frames are processed and acted upon.
- Any Ethernet interrupt is cleared.

The `ladbx_poll` function is a void function that takes no arguments. It must be called often enough to allow the debug monitor to respond promptly to all received Ethernet frames. To ensure that this function gets called at the proper time, enable either Ethernet or timer interrupts (or both) and call it every time an interrupt occurs.

3.4.2.3 Programs that Use the Ethernet

Programs cannot share an Ethernet device with the debug monitor. The debug monitor can drive a selection of different types of Ethernet devices on ISA or PCI cards, and an individual Ethernet device can be selected with the debug monitor **edevice** command.

3.4.3 PALcode Environment

Most programs will be able to use the Digital UNIX compatible PALcode included with the debug monitor; however, for the programs that install their own PALcode, the following guidelines must be followed:

- For remote debug to work, the following Digital UNIX PALcode calls must be implemented according to the interface described in the DEC OSF/1 section of the *Alpha AXP Architecture Reference Manual*.

  IMB
  RDUSP
  RTI
  SWPIPL
  WRENT

- The interface to the system must conform to the standards described in the DEC OSF/1 section of the *Alpha AXP Architecture Reference Manual*.
- The debug server uses the DBGSTOP PAL call to implement breakpoints. The program must contain an identical implementation of the DBGSTOP PAL call.

This PAL call, rather than the BPT PAL call, is used because complex programs (such as operating systems) are likely to reset the EntIF system entry point during initialization.

- The program reset PALcode routine must preserve the address of the debug entry point through the installation of the new PALcode. For the board PALcode, this address is held in the PAL temporary register with symbolic name ptEntDbg. The user-defined PALcode must also either preserve the address of the interrupt entry point (ptEntInt) or set the IPL to a level that prevents all interrupts until the program sets up its own interrupt handler containing a call to `ladbx_poll`.

3.5 The Ladebug Command Line Options for Remote Debugging

Versions 1.3 or later of Ladebug support the following command line options to support remote debugging.

Command Line Option	Description
-rn *node_name*	Specifies IP node name of the target node; required for remote debug; no default.
-pid *process_id*	Specifies the process id of the process to be debugged. The Ladebug software debugs a running process rather than loading a new process.
-rfn *arbitrary string*	Specifies the file name (or other identifier) of the image to be loaded on a remote system. Defaults to the local object file name. Passed to the remote system uninterrupted. Will often have to be quoted to avoid shell command line interpretation on the local system. Can be used only with -rn; do not combine with -pid.
-rinsist	Connects to a running remote process using the connect insist protocol message instead of the connect protocol message. This option functions as a request to the server to connect to the client even if some other client is already connected. (The previously connected client is disconnected.) Use only with -rn and -pid.
-rp *debug protocol name*	Specifies the remote debug protocol to be used. The valid value and default, is `ladebug_preemptive`.
-rt *transport protocol name*	Specifies the transport protocol to be used for remote debug. The valid value and default is `UDP`.

Note

The debug monitor server can only be used to debug already loaded processes; therefore, the `pid` option must always be specified. Because the debug monitor is not a multiprocessing system, the process_id specified with this option is ignored.

For example:

```
%ladebug size.out -rn eb64 -pid 0
```

This example connects to the server on the node with IP node name eb64 and asks to debug the process with pid 0. The local object file is called size.out.

3.6 Building the Executable File

To build the executable file for remote debugging, follow these steps:

1. Compile your source files using the -g option. This preserves the symbolic information in the source files.
2. Link the source files with the -N and -T*x* options; where *x* is the load address for the executable on the board.
3. Use the CSTRIP utility to strip the coff header from the executable file. Keep the unstripped executable file.

3.7 Starting a Ladebug Session

The debug monitor **ladebug** command configures the board as a remote debugger target. Communication is performed through the Ethernet connection.

To debug a program running on an board using Ladebug running on a remote host, follow these steps:

1. Set up the host Digital UNIX machine as described in Chapter 2.
2. Start the board.
3. Load the program into memory on the board.
4. Set a breakpoint in the program.
5. Execute the program. The program will stop at the breakpoint and print the instruction line at that location.

6. Issue the **ladebug** command. This causes the board to wait for a connection from Ladebug.
7. From the host system, enter the command to start Ladebug and cause it to connect to the board.

The following example shows how to set up a sample session:

```
EB64> netload size
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 100000
Init Done.
Ethernet BA-98-76-54-32-01
Attempting BOOTP...success.
     my IP address: 16.123.45.67
 server IP address: 16.123.45.69
gateway IP address: 16.123.45.69
Loading from /users/eval/boot/size ...
####
EB64> stop 200000
EB64> go
Executing at 0x200000...

00200000:  23DEFFF0         lda    sp, -16(sp)
EB64> ladebug
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 100000
Init Done.
Client connected : client is FFFFFFFFA0107F10
```

The following command, entered from the host system, starts Ladebug and causes it to connect to the EB64:

```
% ladebug size.out -rn eb64 -pid 0
Welcome to the Ladebug Debugger Version 1.3.1
------------------
object file name: size.out
machine name: eb64
process id: 0
Reading symbolic information ...done
Connected to remote debugger
(ladebug)
```

The (ladebug), from the previous example, is the Ladebug prompt. You are now ready to debug a process that is running on the EB64. To end this session and return to the debug monitor command prompt, use the Ladebug quit command to disconnect from the server.

Refer to the Ladebug documentation for more information about how to run Ladebug.

4

Debug Monitor User Commands

4.1 Overview of Command Features

This chapter describes how to use the Alpha Microprocessors Board Debug Monitor commands.

The debug monitor supports advanced command line editing, including cursor key movements and an Emacs-like editing interface. In addition, a history buffer has been added to facilitate repetition of commands.

Keys	Description
. (period)	Repeats the last command entered.
↑ (up arrow) Ctrl/P[1]	Scrolls up (older entries) the history buffer.
↓ (down arrow) Ctrl/N	Scrolls down (newer entries) the history buffer.
← (left arrow) Ctrl/B	Moves cursor one character to the left.
→ (right arrow) Ctrl/F	Moves cursor one character to the right.
Backspace Delete Ctrl/H	Deletes the character preceding the cursor.
Ctrl/D	Deletes character at cursor position.
Ctrl/K	Kills text from cursor to end of line.
Ctrl/R	Refreshes the current line.
Ctrl/U	Erases the current line of command text.

[1] If you connected to the board through the UNIX `tip` command, you must press Ctrl/P twice to get the normal effect of Ctrl/P.

Keys	Description
End[2] Ctrl/E	Moves to the end of the line.
Esc/B	Moves cursor to the previous word.
Esc/Backspace Esc/DELETE	Deletes previous word.
Esc/D	Deletes the next word.
Esc/F	Moves cursor to the next word.
Home[2] Ctrl/A[3]	Moves to the beginning of the line.
Insert	Toggles between insert and overwrite mode.
Return Ctrl/J Ctrl/M	Enters current command.

[2]This key requires that the keyboard be connected directly to the board.

[3]If you connected to the board through the UNIX `tip` command, you must press Ctrl/P before you press this key.

4.2 Using the Commands

This section describes the debug monitor command categories.

- Download and execution commands

  The board software basic load command expects to receive Motorola S-records that are stored in the appropriate memory location. The Ethernet port provides improved download performance by using the Internet BOOTP protocol (a UDP-based protocol). This feature allows the board system to determine its Internet address, the address of a boot server, and the name of a file to boot. The debug monitor also supports loading files from a floppy drive or the secondary ROM socket.

  The execution commands can be used to transfer control to a program in memory. These commands begin execution of a program in memory at the specified address, or automatically with a download command.

- Examine and modify memory commands

  These commands are used to examine and change memory in various formats beginning at a specified address and ending at a specified address. Quadwords (64 bits), longwords (32 bits), halfwords (16 bits), and bytes (8 bits) are all supported by these commands.

- PCI commands

These commands are used to access PCI configuration space.

- Utility commands

  These commands are used to display and modify the date and time, display the version or revision of the board debug monitor, and get information about commands implemented in the current version.

- Debug commands

  These commands are used to debug software. Debug commands display internal CPU registers and provide debug capabilities, including breakpoints and single stepping.

- Miscellaneous commands

  These commands are used to read and write the system register, perform an interrupt acknowledge cycle, call a subroutine, and connect to serial communication ports.

- Ethernet commands

  These commands are used to set up and verify status of the Ethernet port.

- Diagnostic commands

  These commands are used to verify that the board is working properly.

4.3 Command Quick Reference

The Table 4–1 contains a summary of all debug monitor commands. The commands are grouped by category and function.

Table 4–1 Command Summary Table

Command	Parameters	Description
	Download and Execution Commands	
load	–	Downloads S-records through a serial port.
boot	–	Downloads S-records through a serial port and begins execution.
netload	file, address	Downloads the specified file through the Ethernet port at the current boot address or specified address.
netboot	file, address	Downloads the specified file through the Ethernet port and begins execution.
flcd	drive_pathname	Changes the current working directory to the specified drive or path.
flcopy	source_file, destination_file	Copies the specified file to another location.
fldir	drive_pathname	Displays a list of files in the current or specified directory.
flload	file, address	Downloads the specified diskette file.
flboot	file, address	Downloads the specified diskette file and begins execution.
flread	first_sector, last_ sector, destination_ address, iterations	Reads logical sectors from a diskette.
flwrite	first_sector, last_ sector, source_ address, iterations	Writes data by logical sectors to a diskette.
flsave	file, start_address, end_address	Saves the specified memory range to the specified file.
romload	type, address	Loads the specified image from ROM to the specified address.
romboot	type, address	Loads the specified image from ROM and begins execution.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Download and Execution Commands	
romlist	none	Lists the ROM image headers contained in ROM.
bootadr	address	Sets default boot address.
bootopt	type	Selects the operating system and firmware type to be used on the next power on.
go	start_address	Starts execution at the specified address.
jtopal	start_address	Starts execution at the specified address in PALmode.
init	none	Reinitializes the debug monitor.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Examine and Modify Memory Commands	
eml	address, iterations, silent	Displays longword data at the specified memory address.
emq	address, iterations, silent	Displays quadword data at the specified memory address.
dml	address, data, iterations	Deposits the specified longword data in the specified memory address.
dmq	address, data, iterations	Deposits the specified quadword data in the specified memory address.
pq	start_address, end_address, iterations, silent	Prints memory in quadword (64-bit) format.
pl	start_address, end_address, iterations, silent	Prints memory in longword (32-bit) format.
pw	start_address, end_address, iterations, silent	Prints memory in word (16-bit) format.
pb	start_address, end_address, iterations, silent	Prints memory in byte (8-bit) format.
cq	address	Edits memory quadwords (64-bit).
cl	address	Edits memory longwords (32-bit).
cw	address	Edits memory words (16-bit).
cb	address	Edits memory bytes (8-bit).
fill	start_address, end_address, fill_pattern	Fills the specified memory block with the specified 32-bit pattern.
copy	start_address, end_address, destination	Copies a memory range to the specified address.
compare	start_address, end_address, compare_address	Matches a memory range to a specified address.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Examine and Modify Memory Commands	
dis	start_address, end_address	Displays memory as CPU instructions.
sum	start_address, end_address	Prints a checksum of a memory range.
rl	register, iterations, silent	Reads a longword from a register port in I/O address space.
rw	register, iterations, silent	Reads a word from a register port in I/O address space.
rb	register, iterations, silent	Reads a byte from a register port in I/O address space.
wl	register, data, iterations	Writes a longword to a register port in I/O address space.
ww	register, data, iterations	Writes a word to a register port in I/O address space.
wb	register, data, iterations	Writes a byte to a register port in I/O address space.
mrl	address, iterations, silent	Reads a longword from memory in I/O address space.
mrw	address, iterations, silent	Reads a word from memory in I/O address space.
mrb	address, iterations, silent	Reads a byte from memory in I/O address space.
mwl	address, data, iterations	Writes a longword to memory I/O address space.
mww	address, data, iterations	Writes a word to memory I/O address space.
mwb	address, data, iterations	Writes a byte to memory I/O address space.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Examine and Modify Memory Commands	
sq	start_address, end_address, string, inverse	Searches the specified memory range by quadwords for the specified pattern.
sl	start_address, end_address, string, inverse	Searches the specified memory range by longwords for the specified pattern.
sw	start_address, end_address, string, inverse	Searches the specified memory range by words for the specified pattern.
sb	start_address, end_address, string, inverse	Searches the specified memory range by bytes for the specified pattern.
	PCI Commands	
pcishow	none	Displays the contents of each PCI slot and current PCI to system address space mapping.
prl	pci_address, id, bus	Reads a longword from the specified address in PCI configuration space.
prw	pci_address, id, bus	Reads a word from the specified address in PCI configuration space.
prb	pci_address, id, bus	Reads a byte from the specified address in PCI configuration space.
pwl	pci_address, id, data, bus	Writes a longword to a specified address in PCI configuration space.
pww	pci_address, id, data, bus	Writes a word to a specified address in PCI configuration space.
pwb	pci_address, id, data, bus	Writes a byte to a specified address in PCI configuration space.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
		Utility Commands
bcon	none	Enables the backup cache.
bcoff	none	Disables the backup cache.
date	yymmddhhmmss	Modifies or displays the date and time.
flash	source_address, destination_offset, bytes_to_write	Programs data into flash memory.
fwupdate	none	Loads and runs the firmware update utility.
help	command_name	Displays a list of commands or displays parameter fields and syntax if a command is specified.
apropos	keyword	Displays help text containing the specified keyword.
ident	start_address, end_address	Displays RCS ID strings found in the specified memory range.
sysshow	none	Displays SROM parameters.
version	none	Displays the debug monitor version information.
		Debug Commands
preg	address	Displays CPU general-purpose registers.
pfreg	address	Displays CPU floating-point registers.
creg	register_number, value	Modifies CPU general-purpose registers.
cfreg	register_number, value	Modifies CPU floating-point registers.
stop	address	Sets a breakpoint at the specified address.
bpstat	none	Displays the current breakpoint status.
step	none	Executes the next instruction after a breakpoint.
cont	none	Continues execution from a breakpoint.
delete	address	Removes breakpoint from the specified address.
ladebug	none	Starts a Ladebug server for a remote debug session.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Miscellaneous Commands	
rsys	none	Reads the EB64 system control register.
wsys	data	Writes the EB64 system control register.
rabox	none	Reads the CPU ABOX_CTL register.
wabox	data	Writes to the CPU ABOX_CTL register.
rbiu	none	Reads the CPU BIU_CTL register.
wbiu	data	Writes to the CPU BIU_CTL register.
riccsr	none	Reads the CPU ICCSR register.
wiccsr	data	Writes to the CPU ICCSR register.
rbcfg	none	Reads the backup cache configuration register.
wbcfg	bcfg_data, bctl_data	Writes the backup cache configuration register.
rbctl	none	Reads the backup cache control register.
wbctl	bctl_data, bcfg_data	Writes the backup cache control register.
iack	none	Performs an interrupt acknowledge cycle.
rmode	status	Sets the **dis** command register display mode.
setty	port	Specifies the port used for debug monitor interaction.
setbaud	port, baud_rate	Sets the communication port baud rate. The default is 9600.
tip	port	Connects to a specified serial communication port.

(continued on next page)

Table 4–1 (Cont.) Command Summary Table

Command	Parameters	Description
	Ethernet Commands	
edevice	device_number	Selects a registered Ethernet device.
eshow	none	Displays all registered Ethernet devices.
ereg	none	Displays the Ethernet controller registers.
estat	none	Displays Ethernet statistics.
einit	none	Initializes Ethernet controller and displays the Ethernet hardware address.
estop	none	Stops the Ethernet controller.
ebuff	address	Sets the base address for Ethernet DMA buffers.
edmp	status	Sets or clears display of packets received or transmitted.
eprom	status	Sets or clears flag for receiving all packets (promiscuous mode).
arpshow	none	Displays all known address resolution protocol (ARP) entries.
	Diagnostic Commands	
beep	duration, frequency	Causes speaker to beep for the specified duration and frequency.
memtest	iterations, start_address, end_address, increment	Tests memory range. Uses longword accesses to memory.

4.4 User Commands

The following section contains a complete description and examples of the debug monitor commands. The commands are listed in alphabetical order.

apropos

The **apropos** command displays help descriptions for the specified keyword.

Format

apropos keyword

Parameters

keyword
Specifies the string to match in the help command text.

Description

The **apropos** command is an additional form of **help**. This command searches the help file and displays all matches for the specified keyword.

Example

```
EB66> apropos load
load:
 Downloads S records through a serial port
 syntax: load
 arguments:

boot:
 Downloads S records through a serial port and begins execution
 syntax: boot
 arguments:

netload:
Downloads file via the Ethernet port to address.  Address defaults to
bootadr
 syntax: netload file address
 arguments: <opt str> <opt hex>

netboot:
 Downloads file through the Ethernet port and begins execution
 syntax: netboot file address
 arguments: <opt str> <opt hex>

         Hit any key to continue. Control-C to quit...
```

arpshow

The **arpshow** command is used to display all known address resolution protocol (ARP) entries.

Format

arpshow

Parameters

None.

Description

The **arpshow** command displays an IP routing table entry. If there are no ARP entries, nothing is shown for that device. The Ethernet device number displayed matches the number that is displayed when the **eshow** and **edevice** commands are entered.

Example

```
EB64> arpshow
Arp Table Contents (at 0x00074570):
        Ethernet Device 0
        IP Address: 16.123.45.67
        MAC Address: BA-98-76-54-32-10
```

beep

The **beep** command is used to test the speaker.

Format

beep duration frequency

Parameters

duration
Specifies the duration of the beep in milliseconds.

frequency
Specifies the frequency in Hertz.

Description

The **beep** command causes the speaker to beep for the specified duration and frequency.

Example

```
EB64> beep  10 1000
```

bcoff

The **bcoff** command disables the backup cache.

Format

bcoff

Parameters

None.

Description

The **bcoff** command is used to disable the external (backup) cache. Use of this command assumes that the cache has already been initialized (usually by the SROM). If the cache is initialized but already disabled, this command has no effect on the state of the cache.

Example

```
EB66+ bcoff
...CAR = 67B0D8E840031294

EB164> bcoff
Old BC_CTL = 0x00028051  &  BC_CFG = 0x01E22772
New BC_CTL = 0x00028050  &  BC_CFG = 0x01E25880
    CIA_CACK_EN = 0x0  &  CIA_MCR = 0x0001FE01
```

bcon

The **bcon** command enables the backup cache.

Format

bcon

Parameters

None.

Description

The **bcon** command is used to enable the external (backup) cache when it has been disabled using the **bcoff** command. If the cache has never been initialized (usually by the SROM), the **bcon** cannot be expected to be capable of enabling it. If the cache is already enabled, this command has no effect on the state of the cache.

Example

```
EB66+ bcon
...CAR = EFB0D8E940031295


EB164> bcon
Old BC_CTL = 0x00028050  &  BC_CFG = 0x01E25880
New BC_CTL = 0x00028051  &  BC_CFG = 0x01E22772
    CIA_CACK_EN = 0x8  &  CIA_MCR = 0x0001FE21
```

boot

The **boot** command downloads S-records through the active serial communication port and begins execution.

Format

boot *use communication software command to access file*

Parameters

Dependent on communication software.

Description

The **boot** command downloads an S-record file through a serial communication port. The S-records contained in the specified file determine the location in memory where the program is loaded. The S-record is received via communication software as a raw text file. The program is then automatically executed.

Example

In this example, the host system connects to the board through the Digital UNIX `tip` command. The ~> is a `tip` command option to copy a file from the Digital UNIX host system to the board (local to remote). The S-records contained in the `size.sr` file are loaded into memory and executed.

```
% tip kdebug
EB64> boot
 Send File now...
~>Local file name? /users/eval/demo2/size.sr
```

bootadr

The **bootadr** command allows you to display or modify the default boot address.

Format

bootadr [address]

Parameters

address
The starting address at which a program is loaded. Programs loaded with the **netboot** command automatically begin program execution at this address. The default address is 200000_{16}.

Description

The boot address is the address at which your programs load and begin execution. The **bootadr** command sets the default address for the load commands to begin execution or to download your program into memory. If the **bootadr** command is specified without an address, then the command displays the current default boot address. If you set the boot address value, the value is stored in battery-backed RAM.

Example

This example sets the starting address to 20000_{16}. The next file that is loaded begins execution from this address.

```
EB64> bootadr 20000
```

bootopt

The **bootopt** command selects the operating system and firmware type to be used on the next power on.

Format

bootopt [type]

Parameters

type
Specifies the operating system type. If the specified image is not found at power on, the first image is booted. If there are no ROM headers, the whole ROM will be loaded at address 0.

Description

The **bootopt** command selects the operating system and associated firmware type that will be used the next time you power on your board. If no type is specified, a list of predefined types is displayed along with the current selection. Use the **romlist** command to display the images contained in the ROM. You can specify the type as a number or a name.

Type_number	Type_name	Description
0	DBM	Alpha Board Debug Monitor
1	NT	Windows NT
2	VMS	OpenVMS
3	UNIX	Digital UNIX
7	LINUX	Linux, Milo

The **bootopt** command can also be used to select a ROM image based on its position in the ROM. Specifying the type as "#0" selects the whole ROM. Specifying the type as "#1" selects the first image; "#2" selects the second image, and so on. The **bootopt** command is not supported for the EB64, EB64+, or the EB66.

Example

```
AlphaPC64> bootopt

Predefined bootoptions are...
  "0" "Alpha Evaluation Board Debug Monitor" "DBM"
  "1" "The Windows NT Operating System" "NT"
  "2" "OpenVMS" "VMS"
  "3" "Digital UNIX, formerly DEC OSF/1" "UNIX"
  "7" "Linux" "Milo"

O/S type selected: "OpenVMS"
....Firmware type: "Alpha SRM Console"

AlphaPC64> bootopt 0

O/S type selected: "Alpha Evaluation Board Debug Monitor"
....Firmware type: "Alpha Evaluation Board Debug Monitor"

AlphaPC64> bootopt nt

O/S type selected: "The Windows NT Operating System"
....Firmware type: "Windows NT Firmware"

AlphaPC64> bootopt #1

Firmware image 1 selected.
....Firmware type: "Unknown"

AlphaPC64> bootopt unix

O/S type selected: "Digital UNIX, formerly DEC OSF/1"
....Firmware type: "Alpha SRM Console"

AlphaPC64> bootopt #0

Load and boot entire ROM at address zero.
....Firmware type: "Unknown"

AlphaPC64> bootopt

Predefined bootoptions are...
  "0" "Alpha Evaluation Board Debug Monitor" "DBM"
  "1" "The Windows NT Operating System" "NT"
  "2" "OpenVMS" "VMS"
  "3" "Digital UNIX, formerly DEC OSF/1" "UNIX"
  "7" "Linux" "Milo"

Load and boot entire ROM at address zero.
....Firmware type: "Unknown"
```

bpstat

The **bpstat** command displays the current breakpoint status.

Format

bpstat

Parameters

None.

Description

The **bpstat** command lists the breakpoints set with the **stop** command. The disassembled instructions for that location are also displayed.

Example

```
EB64> stop 200000
EB64> stop 200FC0
EB64> bpstat
{break} at 00200000:  23DEFFF0         lda    sp, -16(sp)
{break} at 00200FC0:  27BB0001        ldah    r29, 1(r27)
```

cb

The **cb** command allows you to edit memory bytes (8-bit).

Format

cb [address]

Parameters

address
Specifies the address of the memory byte you want to change.

Description

The **cb** command allows you to modify the contents of a specified memory address. If no address is specified, then the next byte is selected. The debug monitor displays the address followed by the current data and a colon (:). For example:

```
00200090:  1D :
```

To modify the contents of this memory location, type the new data after the colon and press the Return key. To end the editing of memory locations, type any non-alphanumeric character except a period (.). The non-alphanumeric character can be typed after the modified byte (on the same line). To leave the current location unchanged, press the Return key on an empty line.

Example

In this example, the bytes at 300000_{16} and 300003_{16} have been modified, leaving the ones at 300001_{16} and 300002_{16} unchanged.

```
EB164> pb 300000 300008
00300000: 1f 04 ff 47 1f 04 ff 47 45 00 60 c3 00 00 00 00 ...G...GE.‘.....
EB164> cb 300000
00300000:  1f: aa
00300001:  04:
00300002:  ff:
00300003:  47: dd
00300004:  1f: ;
EB164> pb 300000 300008
00300000: aa 04 ff dd 1f 04 ff 47 45 00 60 c3 00 00 00 00 .......GE.‘.....
```

cfreg

The **cfreg** command modifies the saved CPU floating-point register state.

Format

cfreg register_number value

Parameters

register_number
Identifies the register.

value
Specifies the new value of the register in hexadecimal numbers.

Description

The **cfreg** command modifies the saved CPU floating-point register state to contain the specified value.

The program register contents are stored in memory to the saved-state area when a breakpoint is encountered. Modifications to a register using the **cfreg** command are applied to that register when execution of the program is resumed using the **step** or the **cont** command.

Example

```
EB64> pfreg
Floating Point Registers
register file @: 0000C840
f00: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f04: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f08: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
PC:  000000000000000D   PS:  000000000000000D
EB64> cfreg 12 abababababab
EB64> cfreg 14 fefefefefe
EB64> pfreg
Floating Point Registers
register file @: 0000C840
f00: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f04: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f08: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12: 000000ABABABABAB 0000000000000000 000000FEFEFEFEFE 0000000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
PC:  000000000000000D   PS:  000000000000000D
```

cl

The **cl** command allows you to edit memory longwords (32-bit).

Format

cl [address]

Parameters

address
Specifies the address of the memory longword you want to change.

Description

The **cl** command allows you to modify the contents of a specified memory address. If no address is specified, then the next longword is selected. The debug monitor displays the address followed by the current data and a colon (:). For example:

```
00200090:  E7E0101D :
```

To modify the contents of this memory location, type the new data after the colon and press the Return key. To end the editing of memory locations, type any non-alphanumeric character except a period (.). The non-alphanumeric character can be typed after the modified byte (on the same line). To leave the current location unchanged, press the Return key on an empty line.

Example

For this example, the memory data at address 0 has been modified from 91E01122 to E7E01021.

```
EB64> cl 0
00000000:  91E01122: e7e01021'
EB64> pl 0 0
00000000: E7E01021 00000000 00000000 00000000 !...............
```

compare

The **compare** command matches a memory range to a specified address.

Format

compare start_address end_address compare_address

Parameters

start_address
Specifies the memory address at which to start the **compare**.

end_address
Specifies the last address that will be compared.

compare_address
Specifies the address to be compared to the memory range.

Description

The **compare** command compares each longword (32 bits) within a specified range in memory to another specified location. It then prints the data that differs.

Example

```
EB66+ copy 3fff80000 3fffd0000 400000
EB66+ fill 400200 400220
EB66+ fill 400400 400440 ffffffff
EB66+ compare 3fff80000 3fffd0000 400000
3FFF80200: 64 86 00 E7 64 00 80 FF   00400200: 00 00 00 00 00 00 00 00
3FFF80208: 7B 06 78 C3 44 A0 10 C0   00400208: 00 00 00 00 00 00 00 00
3FFF80210: F4 9B 10 E0 C3 80 00 80   00400210: 00 00 00 00 00 00 00 00
3FFF80218: 00 CC 00 64 83 00 84 74   00400218: 00 00 00 00 00 00 00 00
3FFF80400: E2 39 37 05 49 99 76 26   00400400: FF FF FF FF FF FF FF FF
3FFF80408: 4B 96 16 C4 4A 36 B7 C1   00400408: FF FF FF FF FF FF FF FF
3FFF80410: 4A 16 04 36 43 00 90 D6   00400410: FF FF FF FF FF FF FF FF
3FFF80418: 6E 0D 00 C0 E2 20 00 08   00400418: FF FF FF FF FF FF FF FF
3FFF80420: 75 40 00 D6 76 42 00 D6   00400420: FF FF FF FF FF FF FF FF
3FFF80428: 76 97 00 08 65 88 00 D6   00400428: FF FF FF FF FF FF FF FF
3FFF80430: 66 95 00 39 67 00 80 FF   00400430: FF FF FF FF FF FF FF FF
3FFF80438: 79 7B 44 00 39 67 99 36   00400438: FF FF FF FF FF FF FF FF
3FFFD0000: FF FF FF FF FF FF FF FF   00450000: 2D 00 00 00 00 00 00 00
```

cont

The **cont** command continues execution from a breakpoint.

Format

cont

Parameters

None.

Description

The **cont** command continues from a breakpoint. The program continues until another breakpoint or the end of the program is reached.

Example

```
EB64> stop 100000
EB64> go
Executing at 0x100000...
00100000:  C1000003          br    r8, 100010
EB64> step
00100010:  2F880007       ldq_u    r28, 7(r8)
EB64> step
00100014:  A49E0000         ldq    r4, 0(sp)
EB64> cont
This simple program prints the size of
various data types in bytes.
 char   =  1
 short  =  2
 int    =  4
 long   =  8
 float  =  4
 double =  8
```

copy

The **copy** command copies the specified memory range to the new specified address.

Format

copy start_address end_address destination

Parameters

start_address
Specifies the starting address for this copy.

end_address
Specifies the last address to be included in this copy.

destination
Specifies the new starting address for the memory range.

Description

The **copy** command copies the data from the specified block of memory to a new location in memory. The original location is unchanged.

Example

This example displays the original location and the destination before and after the **copy** command.

copy

```
EB64> pl 8000000
08000000: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000010: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000020: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000030: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000040: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000050: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000060: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000070: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................

EB64> pl 9000150
09000150: 00000000 00000000 00000000 00000000 ................
09000160: 00000000 00000000 00000000 00000000 ................
09000170: 00000000 00000000 00000000 00000000 ................
09000180: 00000000 00000000 00000000 00000000 ................
09000190: 00000000 00000000 00000000 00000000 ................
090001A0: 00000000 00000000 00000000 00000000 ................
090001B0: 00000000 00000000 00000000 00000000 ................
090001C0: 00000000 00000000 00000000 00000000 ................

EB64> copy 8000000 8000080 9000150
EB64> pl 9000150
09000150: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
09000160: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
09000170: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
09000180: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
09000190: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
090001A0: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
090001B0: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
090001C0: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................

EB64> pl 8000000
08000000: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000010: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000020: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000030: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000040: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000050: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000060: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000070: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
```

cq

The **cq** command allows you to edit memory quadwords (64-bit).

Format

cq [address]

Parameters

address
Specifies the address of the memory quadword you want to change.

Description

The **cq** command allows you to modify the contents of the specified memory address. If no address is specified, then the next quadword is selected. The debug monitor displays the address followed by the current data and a colon (:). For example:

```
00200090:  00000000E7E0101D :
```

To modify the contents of this memory location, type the new data after the colon and press the Return key. To end the editing of memory locations, type any non-alphanumeric character except a period (.). The non-alphanumeric character can be typed after the modified byte (on the same line). To leave the current location unchanged, press the Return key on an empty line.

Example

This example modifies only quadword 200020_{16}.

cq

```
EB64> cq 200020
00200020:  0000000004000000: 0000000011111111
00200028:  0000000000000000:
00200030:  3402010400120106:
00200038:  0402010004020100:
00200040:  FBFDFEFFFFFDFEFF: ;

EB64> pq 200000
00200000: FA7D7299CE7F3299 DA65FA99DA7D32D9 .2...r}..2}...e.
00200010: FFFFFFFBFBFFFFDB FFFFFFFFFFFFFFFF ................
00200020: 0000000011111111 0000000000000000 ................
00200030: 3402010400120106 0402010004020100 .......4........
00200040: FBFDFEFFFFFDFEFF FBFDFEFFFBFDFEFF ................
00200050: CFE7FF99CB6FF799 EEE7FBFBFFFFFFFF ..o.............
00200060: 0000000004020000 0000000000000000 ................
00200070: 1402010620100106 050A050004020100 ... ............
```

creg

The **creg** command modifies the saved CPU general-purpose register state.

Format

creg register_number value

Parameters

register_number
Identifies the register.

value
Specifies the new value of the register in hexadecimal numbers.

Description

The **creg** command modifies the saved CPU general-purpose register state to contain the specified value.

The program register contents are stored in memory to the saved-state area when a breakpoint is encountered. Modifications to a register using the **creg** command are applied to that register when execution of the program is resumed using the **step** or **cont** command.

Example

```
EB64> preg
General Purpose Registers
register file @: 0000C040
r00: 0000000000000020 0000000000000005 000000000000C000 000000000000000D
r04: 00000000000003F8 0000000000000000 0000000000000000 000000000000000D
r08: FFFFFC000005F470 0000000000027340 0444306453605341 0A110C485F6EA26E
r12: 208090EA6024C19C 882C08AA92065B2D 4100610AE100244F 9E2891ACA8A9D984
r16: 0000000000100000 000000000000000D 0000000000000006 0000000000000030
r20: 0000000E20026335 5619A46B2B1A5125 0000000000000000 000000000000000D
r24: 0000000000000003 0000000000000000 FFFFFC0000042C3C 0000000000100000
r28: FFFFFC02C0000000 FFFFFC000006C1E0 0000000000FFDF40 0000000000000003
PC:  000000000000000D   PS:  000000000000000D
EB64> creg 04 555
EB64> preg
General Purpose Registers
register file @: 0000C040
r00: 0000000000000020 0000000000000005 000000000000C000 000000000000000D
r04: 0000000000000555 0000000000000000 0000000000000000 000000000000000D
r08: FFFFFC000005F470 0000000000027340 0444306453605341 0A110C485F6EA26E
r12: 208090EA6024C19C 882C08AA92065B2D 4100610AE100244F 9E2891ACA8A9D984
r16: 0000000000100000 000000000000000D 0000000000000006 0000000000000030
r20: 0000000E20026335 5619A46B2B1A5125 0000000000000000 000000000000000D
r24: 0000000000000003 0000000000000000 FFFFFC0000042C3C 0000000000100000
r28: FFFFFC02C0000000 FFFFFC000006C1E0 0000000000FFDF40 0000000000000003
PC:  000000000000000D   PS:  000000000000000D
```

cw

The **cw** command allows you to edit memory words (16-bit).

Format

cw [address]

Parameters

address
Specifies the address of the memory word you want to change.

Description

The **cw** command allows you to modify the contents of the specified memory address. If no address is specified, then the next word is selected. The debug monitor displays the address followed by the current data and a colon (:). For example:

```
00200090:  101D :
```

To modify the contents of this memory location, type the new data after the colon and press the Return key. To end the editing of memory locations, type any non-alphanumeric character except a period (.). The non-alphanumeric character can be typed after the modified byte (on the same line). To leave the current location unchanged, press the Return key on an empty line.

Example

This example modifies words 200094_{16} through 200098_{16}.

cw

```
EB64> pw 200090
00200090: 3BB9 CA6D FFB9 CFE7 3FBF FFFF 33F9 CE67 .;m......?...3g.
002000A0: 0000 0400 0000 0000 0000 0000 0000 0000 ................
002000B0: 8166 309A 4166 3402 8960 0402 8D46 359A f..0fA.4‘...F..5
002000C0: FEFF FFFD FEFF FBFD FEFF FBFD FEFF FBFD ................
002000D0: 3399 DA65 BB99 CFF7 37BF FFFF 33D9 CE67 .3e......7...3g.
002000E0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
002000F0: 8142 2012 0166 3402 8140 0402 4504 049A B.. f..4@....E..
00200100: FEFF FFFD FEFF FBFD FEFF FBFD FEFF FBFD ................

EB64> cw 200090
00200090:  3BB9:
00200092:  CA6D:
00200094:  FFB9: ffff
00200096:  CFE7: 0000
00200098:  3FBF: 0101
0020009A:  FFFF: ;
EB64> pw 200090 20009A
00200090: 3BB9 CA6D FFFF 0000 0101 FFFF 33F9 CE67 .;m.........3g.
```

date

The **date** command displays or modifies the date and time.

Format

date [yymmddhhmmss]

Parameters

yymmddhhmmss
To modify the date, supply the year, month, day, hour, minute, and second.

Description

If the **date** command is specified alone, the day, month, year, and time is displayed. If you supply a parameter, the date is modified.

Example

This example displays the current date and time setting.

```
EB64> date
Jun  1 12:58:19 1992
```

These examples show how to modify the date and time setting.

```
EB64> date 930211000000
EB64> date
Feb 11 00:00:04 1993

EB64> date 930211135700
EB64> date
Feb 11 13:57:02 1993
```

delete

The **delete** command removes a breakpoint from the specified address.

Format

delete address

Parameters

address
Specifies the address from which to delete the breakpoint.

Description

The **delete** command removes a breakpoint from the specified address. You can use an asterisk (*) to remove all breakpoints.

Example

```
EB64> delete  00200050
```

dis

The **dis** command displays memory as CPU instructions.

Format

dis [start_address [end_address]]

Parameters

start_address
Specifies the address at which to start disassembling instructions. If the address is not specified, the address of the last **load** command, the last breakpoint, or the last **dis** command is used.

end_address
Specifies the address at which to end disassembling instructions. The default is the start_address plus 32 bytes (8 instructions).

Description

The **dis** command disassembles instructions starting with the specified address. You can specify an address range of instructions to be disassembled. If no parameters are specified, then the command starts with the current address and disassembles the next eight instructions. If a file is downloaded to memory, then the default starting address for the **dis** command is the first memory location in the downloaded file. If a breakpoint is encountered, then the default starting address is the breakpoint address.

The **rmode** command is used to select whether the hardware or software register names are displayed when instructions are disassembled. The hardware register names are shown by default. The **rmode** setting is stored in nonvolatile RAM.

dis

Example

```
EB64> dis 243a0
000243A0:  43020122        subl    r24, r2, r2
000243A4:  48441722         sll    r2, 0x20, r2
000243A8:  74420050          mt    r2, cc
000243AC:  64630082          mf    r3, pt2
000243B0:  209F07E1         lda    r4, 2017(zero)
000243B4:  48855724         sll    r4, 0x2A, r4
000243B8:  44640103         bic    r3, r4, r3
000243BC:  47203019         and    r25, 0x1, r25
EB64> dis
000243C0:  4B037698         srl    r24, 0x1B, r24
000243C4:  4703F118         bic    r24, 0x1F, r24
000243C8:  47190418         bis    r24, r25, r24
000243CC:  4B055738         sll    r24, 0x2A, r24
000243D0:  44780403         bis    r3, r24, r3
000243D4:  746300A2          mt    r3, A2
000243D8:  77FF0055          mt    zero, flushIc
000243DC:  77FF0000          mt    zero, 0
EB64>
```

dml

The **dml** command deposits the specified longword data in the specified memory location.

Format

dml address data [iterations]

Parameters

address
Specifies the memory address.

data
Specifies the longword data to be stored.

iterations
Specifies how many times the command is executed. The default is 1.

Description

The **dml** command deposits the specified longword data in the specified memory location. A memory barrier (MB) instruction is executed after the store to force the stored data out of the chip.

Example

```
EB64> dml d0000 FC04FF00
```

dmq

The **dmq** command deposits the specified quadword data in the specified memory location.

Format

dmq address data [iterations]

Parameters

address
Specifies the memory address.

data
Specifies the quadword data to be stored.

iterations
Specifies how many times the command is executed. The default is 1.

Description

The **dmq** command deposits the specified quadword data in the specified memory location. A memory barrier (MB) instruction is executed after the store to force the stored data out of the chip.

Example

```
EB64> dmq d0000 00000000FC04FF00
```

ebuff

The **ebuff** command sets the base address for the Ethernet transmit receive buffers.

Format

ebuff [address]

Parameters

address
Specifies the address for the transmit and receive buffers. The default is 100000_{16}.

Description

The **ebuff** command sets the address in physical memory where the transmit and receive buffers are located. If specified without an address, this command displays the current location of the buffers in memory.

Example

```
EB64> ebuff 180000
```

edevice

The **edevice** command is used to select the registered Ethernet device the debug monitor will use.

Format

edevice [device_number]

Parameters

device_number
Specifies the net device number of any registered Ethernet device. If no device number is provided, the current device number is displayed.

Description

The **edevice** command sets the debug monitor to use one of the registered Ethernet devices. Use the **eshow** command to display all of the registered Ethernet devices.

Example

```
EB64> eshow

All registered Ethernet devices:

 Net      Type
 Device

 0       AM79C960
 1       WD3003
 2       Digital Semiconductor 21040
 3*      Digital Semiconductor 21040

EB64> edevice 1
```

edmp

The **edmp** command displays packets received or transmitted to the terminal screen.

Format

edmp [status]

Parameters

status
Determines whether packets are displayed. Status can be 1 (on) or 0 (off).

Description

The **edmp** command sets or clears the display of packets received or transmitted to the screen. If this command is entered with no status, then the current status is displayed.

Example

```
EB64> edmp
packet dumps are OFF.

EB64> eprom 1
EB64> edmp 1
```

einit

The **einit** command initializes the Ethernet controller.

Format

einit

Parameters

None.

Description

The **einit** command initializes the Ethernet controller and displays the Ethernet hardware address.

Example

```
EB64> einit
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 80000
Init Done.
Ethernet BA-98-76-54-32-10
```

eml

The **eml** command examines and displays a longword of data in memory.

Format

eml address [iterations [silent]]

Parameters

address
Specifies the memory address.

iterations
Specifies how many times the command is executed. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this parameter to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **eml** command displays a longword of data from the specified memory location.

Example

```
EB64> eml d0000
FC04FF00
```

emq

The **emq** command examines and displays a quadword of data in memory.

Format

emq address [iterations [silent]]

Parameters

address
Specifies the memory address.

iterations
Specifies how many times the command is executed. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this parameter to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **emq** command displays a quadword of data from the specified memory location.

Example

```
EB64> emq d0000
00000000FC04FF00
```

eprom

The **eprom** command sets or clears a flag for receiving all packets (promiscuous mode).

Format

eprom [status]

Parameters

status
Determines whether packets are displayed. Status can be 1 (on) or 0 (off).

Description

The **eprom** command sets a flag for receiving packets. If status is set to 1 (on), then promiscuous mode is turned on and packets can be continuously received. If this command is entered with no status, then the current status is displayed. The default status is 0 (off).

Example

```
EB64> eprom
Promiscuous Mode is DISABLED.

EB64> eprom 1
```

ereg

The **ereg** command displays the Ethernet controller registers.

Format

ereg

Parameters

None.

Description

The **ereg** command displays the Ethernet controller registers. This command's output is dependent on the Ethernet device selected for the board. For example, the ISA-based AM79C960 controller must be in stop mode (write 0 to register port 372 and write 4 to the data port 370) to view most of its registers.

Example

```
EB64> ww 372 0
EB64> ww 370 4
EB64> ereg
Ethernet Controller Base Address 360,  CSR 0...126
0   0004   1   0000   2   0008   3   0000   4   1115   5   8000   6   1200
7   0000   8   0000   9   0000   10  0000   11  0000   12  0008   13  1A2B
14  D637   15  4080   16  0000   17  0008   18  0CC8   19  0008   20  1F88
21  0008   22  1308   23  0008   24  0018   25  0008   26  0030   27  0008
28  0028   29  0008   30  0038   31  0008   32  FFFF   33  FDFF   34  0040
35  0008   36  0018   37  0008   38  FFFF   39  FDFF   40  F9C0   41  8308
42  FFC4   43  0308   44  F9C0   45  8308   46  3CFD   47  FFFF   48  FFFF
49  FFFF   50  FFFF   51  FFFF   52  DFFF   53  7EFF   54  FFFF   55  FFFD
56  EFFF   57  FFFF   58  FFFF   59  EFFF   60  0038   61  0008   62  F000
63  8308   64  1F88   65  0008   66  FFC4   67  0308   68  8000   69  0235
70  0202   71  0000   72  FFFC   73  FFFF   74  FFFF   75  FFFF   76  FFFC
77  FFFF   78  FFFE   79  FFFF   80  E810   81  FFFF   82  0000   83  FFFF
84  0038   85  0008   86  F000   87  FFFF   88  3003   89  2000   90  FFFF
91  FFFF   92  FFFE   93  FFFF   94  0235   95  FFFF   96  1308   97  8308
98  F9C0   99  0235   100 FFFF   101 FFFF   102 FFFF   103 FFFF   104 0000
105 0202   106 FFFF   107 FFFF   108 8000   109 0235   110 FFFF   111 FFFF
112 0000   113 FFFF   114 00A2   115 FFFF   116 FFFF   117 FFFF   118 FFFF
119 FFFF   120 FFFF   121 FFFF   122 FFFF   123 FFFF   124 FC00   125 FFFF
126 0000

Ethernet Controller ISACSR0 ... 7
0 0005   1 0005   2 0003   3 0000   4 0000   5 0084   6 0008   7 0090
```

eshow

The **eshow** command displays all of the registered Ethernet devices.

Format

eshow

Parameters

None.

Description

The **eshow** command displays all of the installed device drivers and works for all the boards. To set the debug monitor to use one of these devices, see the **edevice** command. An asterisk following the netdevice number indicates the selected Ethernet device to be used by the debug monitor Ethernet commands.

Example

```
EB64> eshow

All registered Ethernet devices:

 Net      Type
 Device

 0       AM79C960
 1       WD3003
 2       Digital Semiconductor 21040
 3*      Digital Semiconductor 21040
```

estat

The **estat** command displays Ethernet statistics.

Format

estat

Parameters

None.

Description

The **estat** command displays Ethernet statistics kept by the Ethernet device driver.

Example

```
EB64> estat
                  secs:         7                mc bytes rcv:    130075
             bytes rcv:   1297171                  mc frms rcv:       625
             bytes snt:         0               frms snt dfrd:         0
              frms rcv:      3129            frms snt - cllsn:         0
              frms snt:         0       frms snt - mult cllsn:         0

  snd flrs - xs cllsn:         0              snd flrs - def:         0
        snd flrs - cc:         0              rcv flrs - fcs:         0
      snd flrs - shrt:         0             rcv flrs - ferr:         0
       snd flrs - opn:         0               rcv flrs flen:         0
      snd flrs - flen:         0                   data ovrn:         0
        cllsn chk flr:         0
```

estop

The **estop** command stops the Ethernet controller.

Format

estop

Parameters

None.

Description

The **estop** command allows you to stop sending or receiving packets from an Ethernet device selected with the **edevice** command.

Example

```
EB66> eshow
All registered Ethernet devices:

        Net     Type
        Device
        0*      Digital Semiconductor 21040
        1       AM79C960
EB66> edevice
Using network device 0
EB66> estop

Stopping network device 0 in PCI slot 20:
```

fill

The **fill** command fills a specified memory block with the specified 32-bit pattern.

Format

fill start_address end_address [fill_pattern]

Parameters

start_address
Specifies the start address for the fill pattern.

end_address
Specifies the end address for the fill pattern. The fill pattern includes the end_address.

fill_pattern
Specifies a longword hexadecimal number as the fill pattern for the specified address. The default is 0.

Description

The **fill** command fills a specified block of memory with a specified pattern. The data or fill pattern specified is placed in memory starting at the first address specified, and it fills through the last (or end) address specified.

Example

This example displays the original value in address range 08000000 through 08000080 and the value of the same address range after the **fill** command.

fill

```
EB64> pl 8000000
08000000: E7E01021 00000000 00000000 00000000 !...............
08000010: 00000000 00000000 00000000 00000000 ................
08000020: E7E01095 00000000 00000000 00000000 ................
08000030: 00000000 00000000 00000000 00000000 ................
08000040: 00000000 00000000 00000000 00000000 ................
08000050: 00000000 00000000 00000000 00000000 ................
08000060: 00000000 00000000 00000000 00000000 ................
08000070: 00000000 00000000 00000000 00000000 ................

EB64> fill 8000000 8000080 1f1f1f1f
EB64> pl 8000000 8000080
08000000: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000010: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000020: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000030: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000040: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000050: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000060: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000070: 1F1F1F1F 1F1F1F1F 1F1F1F1F 1F1F1F1F ................
08000080: 1F1F1F1F 00000000 00000000 00000000 ................
```

flash

The **flash** command programs data into flash memory.

Format

flash [source_address [destination_offset [bytes_to_write]]]

Parameters

source_address
Specifies the address in memory of the data to be programmed into the flash. The default is the default boot address (see **bootadr**).

destination_offset
Specifies the offset, in bytes, into the flash where the first byte of source data will be programmed. If not provided, you are prompted with a default destination_offset value. The destination_offset combined with the size the data to be written must fit within the remaining space in the flash.

bytes_to_write
Specifies how many bytes to write beginning at the source_address. This parameter causes the flash command to ignore any standard header that might be included in the source data. This value defaults to the value in the image size field of the standard header. If not specified and if there is no standard image at the beginning of the source data, this value is assumed to be the remaining space in the flash.

Description

The **flash** command is used to program the flash memory on the boards containing this type of memory. It reads data from memory at the specified source address and programs it into the flash at the specified offset. The amount of data written can be specified by the user or determined by the flash command.

Example

```
AlphaPC64> netload pc64dbm.rom
Attempting BOOTP...
Loading /users/eval/pc64/pc64dbm.rom at 300000
 My IP address:     16.123.45.67
 Server IP address: 16.123.45.69
####################File loaded
AlphaPC64>

AlphaPC64> flash

Image source address : 0x300000
Standard image header: Found.
  Header Size......... 56 bytes
  Image Checksum...... 0x6eeb (28395)
  Memory Image Size... 0x30B2C (199468 = 194 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 2
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 0x30B2C (199468 = 194 KB)
  Firmware ID (Opt.).. 0200009511221015  .."....
  ROM offset.......... 0x00000000
  Header Checksum..... 0x71fb

Enter destination offset or press RETURN for default [0]:

Flash offset         : 0x0
Image size w/ header : 199524  (Segment 0 to 3 inclusive).

        !!!!! Warning: About to overwrite flash memory !!!!!
             Press Y to proceed, any other key to abort.

Update cancelled by user.

AlphaPC64> flash

Image source address : 0x300000
Standard image header: Found.
  Header Size......... 56 bytes
  Image Checksum...... 0x6eeb (28395)
  Memory Image Size... 0x30B2C (199468 = 194 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 2
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 0x30B2C (199468 = 194 KB)
  Firmware ID (Opt.).. 0200009511221015  .."....
  ROM offset.......... 0x00000000
  Header Checksum..... 0x71fb

Enter destination offset or press RETURN for default [0]: 40000
```

```
Flash offset         : 0x40000
Image size w/ header : 199524  (Segment 4 to 7 inclusive).

        !!!!! Warning: About to overwrite flash memory !!!!!
             Press Y to proceed, any other key to abort.

 Writing Flash Block: 4W 5W 6W 7W
 Verifying Flash Block: 4V 5V 6V 7V

AlphaPC64> romlist
ROM image header found at offset: 0x040000
  Header Size......... 56 bytes
  Image Checksum...... 0x6eeb (28395)
  Memory Image Size... 0x30B2C (199468 = 194 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 2
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 0x30B2C (199468 = 194 KB)
  Firmware ID (Opt.).. 0200009511221015  .."....
  ROM offset.......... 0x00000000
  Header Checksum..... 0x71fb

! Change the Image Destination field from 300000 to 400000
! Note that because no changes were performed to the Header
! Checksum field after the change, a header checksum
! error will be reported with romlist.

AlphaPC64> dml 500000 400000
AlphaPC64> flash 500000 40018 4

Image source address : 0x500000

Flash offset         : 0x40018
Data image size      : 4 (Segment 4 to 4 inclusive).

        !!!!! Warning: About to overwrite flash memory !!!!!
             Press Y to proceed, any other key to abort.

 Writing Flash Block: 4W
 Verifying Flash Block: 4V

AlphaPC64> romlist
```

flash

```
ROM image header found at offset: 0x040000
  Header Size......... 56 bytes
  Image Checksum...... 0x6eeb (28395)
  Memory Image Size... 0x30B2C (199468 = 194 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000400000
  Header Version...... 2
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 0x30B2C (199468 = 194 KB)
  Firmware ID (Opt.).. 0200009511221015  .."....
  ROM offset.......... 0x00000000
  Header Checksum..... 0x71fb
ERROR: Bad ROM header checksum. 0x79fb
```

flboot

The **flboot** command downloads the specified file from the diskette and begins execution of that file.

Format

flboot file [address]

Parameters

file
Specifies the name of the file to access on the diskette.

address
Specifies the address at which to load the file. The default is the boot address.

Description

The **flboot** command downloads the specified file into the specified address or the boot address. The downloaded file automatically begins execution in PALmode as if a **jtopal** command had been entered.

Example

```
EB64> flboot  size2
High Density selected

size2    .            20 bytes  11/21/1991 13:42:20
loading...
cluster:   2 sector:  33 buffer:  200000
done...
Jumping to 0x200000...
```

flcd

The **flcd** command displays or changes the current working directory or drive.

Format

flcd [drive_pathname]

Parameters

drive_pathname
Specifies the new drive and working directory.

Description

The **flcd** command allows you to change the current working directory for the current drive. It can also be used to switch to a different default drive. If no parameters are specified, then the default drive and working directory are displayed.

Drives are specified by using the letters A through Z. The path is a list of subdirectories separated by a slash (/) for UNIX users or a backslash (\) for DOS users. The top level directory (known as the root directory) is represented by a slash (/) or backslash (\). A path can be an absolute or relative path. An absolute path begins with the root directory, whereas a relative path begins with the current working directory.

Subdirectory entries also contain two special entries that can be used to specify a path. One period (.) represents the current directory and two periods (..) represent the directory above the current level.

Example

```
AlphaPC64> flcd
 a:\

AlphaPC64> fldir
High Density selected
10/04/95  02:07p                203088 rom.cmp
10/04/95  02:08p                203140 rom.rom
10/06/95  10:05a            <DIR>       dir1
10/06/95  10:05a            <DIR>       dir3
                               1048576 bytes free
AlphaPC64> flcd dir1
a:\dir1\

AlphaPC64> fldir
High Density selected
10/06/95  10:05a            <DIR>       .
10/06/95  10:05a            <DIR>       ..
10/06/95  10:05a            <DIR>       dir2
                               1048576 bytes free

AlphaPC64> flcd /dir1/dir2
a:\dir1\dir2\

AlphaPC64> fldir
High Density selected
10/06/95  10:05a            <DIR>       .
10/06/95  10:05a            <DIR>       ..
                               1048576 bytes free

AlphaPC64> flcd ../../dir3
a:\dir1\dir2\..\..\dir3\

AlphaPC64> fldir
High Density selected
10/06/95  10:05a            <DIR>       .
10/06/95  10:05a            <DIR>       ..
04/28/95  05:50p                   71 diff.lst
                               1048576 bytes free


AlphaPC64> flcd b:
b:\

AlphaPC64> fldir
High Density selected
09/07/95  10:28a                 6688 srom
10/03/95  05:59p               202980 rom.rom
                               1247232 bytes free
```

flcopy

The **flcopy** command copies one file to another location.

Format

flcopy source_file destination_file

Parameters

source_file
Specifies the file to be copied. If no drive and path are specified, the default drive and path are used.

destination_file
Specifies the name of the copied file. If no drive and path are specified the default drive and path are used. Note that a destination filename must always be specified, even if copying to a subdirectory.

Description

The **flcopy** command allows you to copy a file to another destination. An optional drive and path specification may be specified for either the source or destination filename. If they are not specified, then the default drive and path are used.

Example

```
AlphaPC64> flcd \dir3
a:\dir3\

AlphaPC64> fldir
High Density selected
10/06/95  10:05a           <DIR>       .
10/06/95  10:05a           <DIR>       ..
04/28/95  05:50p                   71 diff.lst
                             1048064 bytes free

AlphaPC64> flcopy diff.lst ..\dir1\dir2\diff2.lst
High Density selected
Copying files...
Done...

AlphaPC64> fldir ..\dir1\dir2\
High Density selected
10/06/95  10:05a           <DIR>       .
10/06/95  10:05a           <DIR>       ..
10/06/95  10:48a                   71 diff2.lst
                             1047552 bytes free

AlphaPC64> flcopy diff.lst b:\diff2.lst
High Density selected
High Density selected
Copying files...
Done...

AlphaPC64> fldir b:\
High Density selected
09/07/95  10:28a                 6688 srom
10/03/95  05:59p               202980 rom.rom
10/06/95  10:53a                   71 diff2.lst
                                1246720 bytes free
```

fldir

The **fldir** command displays a list of files in the current or specified directory.

Format

fldir [drive_pathname]

Parameters

drive_pathname
Specifies the drive or subdirectory.

Description

The **fldir** command displays a directory of files in the current or specified directory.

Drives are specified by using the letters A through Z. The path is a list of subdirectories separated by a slash (/) for UNIX users or a backslash (\) for DOS users. The top level directory (known as the root directory) is represented by a slash (/) or backslash (\). A path can be an absolute or relative path. An absolute path begins with the root directory, whereas a relative path begins with the current working directory.

Subdirectory entries also contain two special entries that can be used to specify a path. One period (.) represents the current directory and two periods (..) represent the directory above the current level.

Example

```
AlphaPC64> flcd
a:\

AlphaPC64> fldir
High Density selected
10/04/95  02:07p                203088 rom.cmp
10/04/95  02:08p                203140 rom.rom
10/06/95  10:05a            <DIR>       dir1
10/06/95  10:05a            <DIR>       dir3
                               1048064 bytes free

AlphaPC64> fldir /dir1
High Density selected
10/06/95  10:05a            <DIR>       .
10/06/95  10:05a            <DIR>       ..
10/06/95  10:05a            <DIR>       dir2
                               1048064 bytes free

AlphaPC64> flcd dir1\dir2
a:\dir1\dir2\

AlphaPC64> fldir ..\..\dir3
High Density selected
10/06/95  10:05a            <DIR>       .
10/06/95  10:05a            <DIR>       ..
04/28/95  05:50p                    71 diff.lst
                               1048064 bytes free

AlphaPC64> fldir b:\
High Density selected
09/07/95  10:28a                  6688 srom
10/03/95  05:59p                202980 rom.rom
                               1247232 bytes free
```

flload

The **flload** command downloads the specified file from the diskette.

Format

flload file [address]

Parameters

file
Specifies the name of the file to access on the diskette.

address
Specifies the address at which to load the file. The default is the boot address.

Description

The **flload** command downloads the specified file into the specified address or the boot address. The program can then be executed with the **go** or **jtopal** commands.

Example

```
EB64> bootadr
00200000
EB64> flload  size2
High Density selected

size2    .            20 bytes  11/21/1991 13:42:20
loading...
cluster:   2 sector:  33 buffer:  200000
done...
```

flread

The **flread** command reads logical sectors from a diskette.

Format

flread [first_sector [last_sector [destination_address [iterations]]]]

Parameters

first_sector
Specifies the first logical sector of diskette to read. The default is sector 0 (the boot sector).

last_sector
Specifies the last logical sector of diskette to read. The default sector is the same as the first_sector value.

destination_address
Specifies the beginning address where data will be loaded. The default is the boot address.

iterations
Specifies the number of times to repeat the reading of the sector range. The default is 1.

Description

The **flread** command reads the data from the specified logical sectors of a diskette into memory. The iterations parameter can be used to repeat the task a specified number of times.

Example

```
AlphaPC64> flread
Reading sectors 0 to 0 to 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64>flread 1
Reading sectors 1 to 1 to 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64> flread 0 10
Reading sectors 0 to 10 to 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64> flread 0 10 300000
Reading sectors 0 to 10 to 0x300000 1 time(s).
High Density selected
Done...

AlphaPC64> flread 0 10 300000 5
Reading sectors 0 to 10 to 0x300000 5 time(s).
High Density selected
Done...
```

flsave

The **flsave** command writes a memory range to a file.

Format

flsave file start_address end_address

Parameters

file
Specifies the name of the file to be created with the data. If no drive or path is specified, the file is created in the default working directory.

start_address
Specified the starting memory address of data to be saved.

end_address
Specifies the ending memory address of data to be saved. The end_address is included in the saved data.

Description

The **flsave** command can be used to write a section of memory to the specified file starting at the specified start_address and ending at the specified end_address. The filename can specify a drive and a path.

Example

```
AlphaPC64> flsave test.txt 300000 300400
High Density selected
Saving range 0x300000 to 0x300400 to file  test.txt

AlphaPC64> flsave  b:\test.txt 300000 300400
High Density selected
Saving range 0x300000 to 0x300400 to file  b:\test.txt
```

flwrite

The **flwrite** command writes data to logical sectors on a diskette.

Caution

This is a destructive command. You must be careful of which sectors you write to or you may render the disk unusable.

Format

flwrite [first_sector [last_sector [source_address [iterations]]]]

Parameters

first_sector
Specifies the first logical sector of diskette to be written. The default is sector 0 (the boot sector).

last_sector
Specifies the last logical sector of diskette to be written. The default sector is the same as the first_sector value.

source_address
Specifies the beginning address where data to be written resides. The default is the boot address.

iterations
Specifies the number of times to repeat the writing of the sector range. The default is 1.

Description

The **flwrite** command writes data from memory to the specified logical sectors of a diskette.

The iterations parameter can be used to repeat the task a specified number of times.

Example

```
AlphaPC64> flwrite
Writing sectors 0 to 0 from 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64> flwrite 1
Writing sectors 1 to 1 from 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64> flwrite 0 10
Writing sectors 0 to 10 from 0x200000 1 time(s).
High Density selected
Done...

AlphaPC64> flwrite 0 10 300000
Writing sectors 0 to 10 from 0x300000 1 time(s).
High Density selected
Done...

AlphaPC64> flwrite 0 10 300000 5
Writing sectors 0 to 10 from 0x300000 5 time(s).
High Density selected
Done...
```

fwupdate

The **fwupdate** command loads and runs the Firmware Update Utility from diskette.

Format

fwupdate

Parameters

None.

Description

The **fwupdate** command loads and executes the firmware update utility (fwupdate.exe) from diskette. The utility gets loaded into physical address 900000_{16}, (physical location 9 MB), and get executed in PALmode.

This command expects the diskette to be formatted with a FAT file structure.

Example

```
AlphaPC64> fwupdate

   ...follow instructions to update firmware for
   Windows NT Firmware, the debug monitor, or the Alpha SRM Console ...
```

go

The **go** command begins execution of instructions at the specified address.

Format

go [start_address]

Parameters

start_address
Specifies the address at which to start executing the instructions.

Description

The **go** command is used to jump to a location in memory and begin executing instructions. If no address is specified, then the execution of instructions begins at the boot address.

Example

This example starts executing instructions at address 100000_{16}.

```
EB64> go 100000
```

help

The **help** command displays a list of commands currently available. If you specify a command keyword, information about the specified command is displayed.

Format

h[elp] [command_keyword]

Parameters

command_keyword
Any command name that appears in the list when you type the **help** command. An asterisk (*) displays help for all commands.

Description

The **help** command displays a list of command keywords implemented in the current release. The command can be abbreviated to one letter **h**. If you specify a command with a command keyword, then a brief description and syntax for the specified command is displayed. An asterisk (*) can be specified in the place of a command keyword to display all help information.

Example

The **help** command without a parameter displays a list of all commands implemented in the current version of the software. When specified with a parameter, it displays more information about that command keyword.

help

```
AlphaPC64> help

A brief help description is available for each of the
following commands.

 load boot netload netboot flcd flcopy fldir flboot flload flread flwrite
 flsave romload romboot romlist bootadr bootopt go jtopal init eml emq dml dmq
 pq pl pw pb cq cl cw cb fill copy compare dis sum rl rw rb wl ww wb mrl mrw
 mrb mwl mww mwb sq sl sw sb pcishow prl prw prb pwl pww pwb flash fwupdate
 bcon bcoff date apropos help h ident version sysshow preg pfreg ppreg creg
 cfreg cpreg stop bpstat step s cont delete ladebug rabox wabox rbiu wbiu
 riccsr wiccsr iack rmode setty setbaud tip edevice eshow ereg estat einit
 estop ebuff edmp eprom arpshow beep memtest

examples:

help command_keyword
Displays help for command_keyword, where command_keyword
is any command name that appears in the command list.

AlphaPC64> h date
date: date ["yymmddhhmmss"]
arguments: <opt str>

AlphaPC64>  h go
go: start execution at <adr>
arguments: <opt hex>
```

iack

The **iack** command performs an interrupt acknowledge cycle.

Format

iack

Parameters

None.

Description

The **iack** command allows you to perform an interrupt acknowledge cycle. Two **iack** commands are required to read the interrupt vector.

Example

```
EB64> iack
FF
EB64> iack
07
```

ident

The **ident** command displays revision control system (RCS) ID strings found in the specified memory range.

Format

ident [start_address [end_address]]

Parameters

start_address
A hexadecimal number that represents a legal address at which to start searching for RCS keywords. The default value is the boot address.

end_address
A hexadecimal number that represents a legal address at which to end the search for RCS keywords. The default value is the boot address plus 70_{16}.

Description

The **ident** command can identify the revision of files used to build images that were loaded into memory by searching for all occurrences of the pattern $keyword: ...$ in the specified memory range. This command is based on the assumption that RCS was used for version control on the source files on the host development system. RCS is supplied with the Digital UNIX operating system.

Example

```
EB64> ident 0 80000
Id: crt_startup.s,v 1.3 1993/06/18 20:30:03 fdh Rel $
Id: crt.c,v 1.1 1993/06/08 19:56:39 fdh Rel $
Id: dis.c,v 1.1 1993/06/08 19:56:40 fdh Rel $
Id: ffexec.c,v 1.2 1993/06/09 20:23:05 fdh Rel $
Id: ffsrec.c,v 1.1 1993/06/08 19:56:41 fdh Rel $
Id: cmd.c,v 1.6 1993/06/18 17:32:36 fdh Rel $
Id: pReg.c,v 1.1 1993/06/08 19:56:41 fdh Rel $
Id: rw.c,v 1.1 1993/06/08 19:56:42 fdh Rel $
Id: netboot.c,v 1.1 1993/06/08 19:56:30 fdh Rel $
Id: amd.c,v 1.2 1993/06/08 22:32:57 berent Rel $
Id: tftp.c,v 1.1 1993/06/08 19:56:31 fdh Rel $
Id: netutil.c,v 1.1 1993/06/08 19:56:31 fdh Rel $
Id: boots.c,v 1.2 1993/06/08 22:32:57 berent Rel $
Id: listener.c,v 1.2 1993/06/08 22:32:57 berent Rel $
Id: kernel.c,v 1.5 1993/06/18 17:49:34 fdh Rel $
Id: bptable.c,v 1.1 1993/06/08 19:56:33 fdh Rel $
Id: kutil.s,v 1.1 1993/06/08 19:56:36 fdh Rel $
Id: comms.c,v 1.2 1993/06/08 22:32:06 berent Rel $
Id: server_read_loop.c,v 1.1 1993/06/08 19:56:38 fdh Rel $
Id: packet-handling.c,v 1.2 1993/06/08 22:32:06 berent Rel $
Id: printf.c,v 1.1 1993/06/08 19:56:24 fdh Rel $

        Hit any key to continue. Control-C to quit...
```

init

The **init** command reinitializes the debug monitor.

Format

init

Parameters

None.

Description

The **init** command restarts the debug monitor by jumping to the PALcode base address in PALmode. It is analogous to using the **jtopal** command with the PALbase address.

Example

```
AlphaPC64> init
Jumping to 0x000000...

Stopping network device 0 in PCI slot 18:


================  Starting Debug Monitor!!! ===================
```

jtopal

The **jtopal** command sets the environment to PALmode and begins execution of instructions at the specified address.

Format

jtopal [start_address]

Parameters

start_address
Specifies the address at which to start executing instructions. The default is the boot address.

Description

The **jtopal** command is used to emulate the hardware mechanism for entering PALcode. When instructions contain PALcode, you must set the environment to PALmode to properly execute instructions. This command is required for executing downloaded images entered in PALmode, such as a serial ROM or debug ROM image. The **jtopal** command sets the environment to PALmode and then jumps to the specified location in memory to begin executing instructions.

Example

This example starts executing instructions at address 100000_{16}.

```
EB64> jtopal 100000
```

ladebug

The **ladebug** command starts the Ladebug server for a remote debug session.

Format

ladebug

Parameters

None.

Description

The **ladebug** command configures the board as a remote debugger target. You can connect to the board from the Ladebug source-level debugger running on a Digital UNIX host. Communication is performed through the Ethernet connection. The Ladebug software provides the full source-level debugging capabilities of most programs running on the board, including the debug monitor.

To debug a program running on an board using Ladebug running on a remote host, follow these steps:

1. Load the program into memory on the board.
2. Set a breakpoint in the program.
3. Execute the program. The program will stop at the breakpoint and print the instruction line at that location.
4. Issue the **ladebug** command. This causes the board to wait for a connection from Ladebug.
5. From the host system, enter the command to start up Ladebug and cause it to connect to the board.

Refer to the Ladebug documentation for more information.

Example

```
EB64> netload size
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 100000
Init Done.
Ethernet BA-98-76-54-32-10
Attempting BOOTP...success.
     my IP address: 16.123.45.67
 server IP address: 16.123.45.69
gateway IP address: 16.123.45.69
Loading from /users/eval/boot/size ...
####
EB64> stop 200000
EB64> go
Executing at 0x200000...

00200000:  23DEFFF0         lda    sp, -16(sp)
EB64> ladebug
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 100000
Init Done.
Client connected : client is FFFFFFFFA0107F10
```

The following command, entered from the host system, starts Ladebug and causes it to connect to the EB64:

```
% ladebug size.out -rn eb64 -pid 0
```

The following information is displayed on the host system:

```
Welcome to the Ladebug Debugger Version 1.3.1
------------------
object file name: size.out
machine name: eb64
process id: 0
Reading symbolic information ...done
Connected to remote debugger
(ladebug)
```

The (ladebug) in the previous example is the Ladebug prompt. You are now ready to debug a process that is running on the EB64.

load

The **load** command downloads S-records through the active serial communication port.

Format

load *use communication software command to access file*

Parameters

Dependent on communication software.

Description

The **load** command downloads an S-record file through a serial communication port. The S-records contained in the specified file determine the location in memory where the program is loaded. The S-record is received via communication software as a text file (no special communication software is required). The program can then be executed with the **go** or **jtopal** command.

Example

In this example, the host system connects to the board through the Digital UNIX `tip` command. The ~> is a `tip` command option to copy files from local to remote systems. The S-records contained in the `size.sr` file are loaded into memory.

```
% tip kdebug
EB64> load
 Send File now...
~>Local file name? /users/eval/demo2/size.sr
```

memtest

The **memtest** command tests a memory range.

Format

memtest [iterations [start_address [end_address [increment]]]]

Parameters

iterations
Specifies the number of times the memory range test will run. The default iteration is 1.

start_address
Specifies the address at which to start the memory test. The default is the current address.

end_address
Specifies the address at which to end the memory test.

increment
Defines the step size. The default is longword access (4).

Description

The **memtest** command performs a set of memory tests on the specified address range. This test uses longword accesses to memory. The tests include walking 1's and walking 0's as well as alternating 1's and 0's.

Example

```
EB64> memtest 2 8000000 8ffffff
```

mrb

The **mrb** command reads a byte from memory in the register port in I/O address space.

Format

mrb address [iterations [silent]]

Parameters

address
Specifies the address in memory I/O space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **mrb** command displays the byte from the specified memory location in the memory I/O space. For example, on the EB64 the byte is read from the ISA extension slot.

Example

```
EB64> mrb d0000
FF
```

mrl

The **mrl** command reads a longword from memory in the register port in I/O address space.

Format

mrl address [iterations [silent]]

Parameters

address
Specifies the address in memory I/O space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **mrl** command displays the longword from the specified memory location in the memory I/O space. For example, on the EB64 the longword is read from the ISA extension slot.

Example

```
EB64> mrl d0000
FC04FF00
```

mrw

The **mrw** command reads a word from memory in the register port in I/O address space.

Format

mrw address [iterations [silent]]

Parameters

address
Specifies the address in memory I/O space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **mrw** command displays the word from the specified memory location in the memory I/O space. For example, on the EB64 the word is read from the ISA extension slot.

Example

```
EB64> mrw d0000
FF00
```

mwb

The **mwb** command writes a byte to memory in the register port in I/O address space.

Format

mwb address data [iterations]

Parameters

address
Specifies the address in memory I/O space where the byte is written.

data
Specifies byte data.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **mwb** command specifies the memory location in I/O memory space to write data in byte format.

Example

```
EB64> mrb d0000
FF
EB64> mwb d0000 0
EB64> mrb d0000
00
```

mwl

The **mwl** command writes a longword to memory in the register port in I/O address space.

Format

mwl address data [iterations]

Parameters

address
Specifies the address in memory I/O space where the longword is written.

data
Specifies longword data.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **mwl** command writes a longword to memory in I/O address space. For example, on the EB64 the longword is written to the ISA extension slot.

Example

```
EB64> mwl d0000 fc04ff00
```

mww

The **mww** command writes a word to memory in the register port in I/O address space.

Format

mww address data [iterations]

Parameters

address
Specifies the address in memory I/O space where the word is written.

data
Specifies word data.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **mww** command writes a word to memory I/O space. For example, on the EB64 a word is written in the ISA extension slot.

Example

```
EB64> mrw d0000
FF00
EB64> mww d0000 a5a5
EB64> mrw d0000
A5A5
```

netboot

The **netboot** command downloads the specified file through the Ethernet port and begins execution of that file.

Format

netboot [file [address]]

Parameters

file
Specifies a legal file name to be downloaded to the board. The default is to load the file specified in the bootptab file.

address
Specifies the address at which to download the file. The default is the boot address.

Description

The **netboot** command uses BOOTP to download the specified file through the Ethernet port. The Ethernet port is selected through the **edevice** command. The downloaded file automatically begins execution in PALmode. This command has the same effect as using the **netload** command followed by the **jtopal** command.

A default file and directory path may be defined in the bootptab file. See Section 2.3.4.2 for more information.

If you specify an address, this address becomes the default boot address. However, this value is not set in battery-backed RAM.

Example

This example downloads and begins execution of a file called size.

netboot

```
EB64> netboot size
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 80000
Init Done.
Ethernet BA-98-76-54-32-10
Attempting BOOTP...success.
     my IP address: 16.123.45.67
 server IP address: 16.123.45.69
gateway IP address: 16.123.45.69
Loading from /users/eval/boot/size ...
###
Jumping to 0x100000...

 char   =  1
 short  =  2
 int    =  4
 long   =  8
 float  =  4
 double =  8

Alpha 21064 Evaluation Board (EB64) Debug Monitor
 Version: Wed Feb 10 19:52:24 EST 1993
 Bootadr: 0x100000, memSize: 0x2000000
```

netload

The **netload** command downloads the specified file through the Ethernet port to the default boot address.

Format

netload [file [address]]

Parameters

file
Specifies a legal file name to be downloaded to the board. The default is to load the file specified in the bootptab file.

address
Specifies the address at which to download the file. The default is the boot address.

Description

The **netload** command uses BOOTP to download the specified file through the Ethernet port. The Ethernet port is selected using the **edevice** command. The program is loaded into the default boot address. You can set up or change the boot address with the **bootadr** command. The program can then be executed with the **go** or **jtopal** command.

A default file and directory path may be defined in the bootptab file. See Section 2.3.4.2 for more information.

If you specify an address, this address becomes the default boot address. However, this value is not set in battery-backed RAM.

Example

In this example, a file called size is loaded into the default boot address.

netload

```
EB64> netload size
Ethernet Base Address: 360, DMA Mask: 1 = DRQ5
Init Block Address 80000
Init Done.
Ethernet BA-98-76-54-32-10
Attempting BOOTP...success.
     my IP address: 16.123.45.67
 server IP address: 16.123.45.69
gateway IP address: 16.123.45.69
Loading from /users/eval/boot/size ...
###
```

pb

The **pb** command displays the specified memory byte (8-bit).

Format

pb [start_address [end_address [iterations [silent]]]]

Parameters

start_address
A hexadecimal number that represents a legal address at which to start the display. The default is the current address.

end_address
A hexadecimal number that represents a legal address at which to end the display. The default is the current address plus 127 bytes.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **pb** command displays the specified memory in byte format. If no address is specified, then the current memory byte and the 127 bytes that follow it are displayed. The field displayed after the bytes represents the translation of the memory contents in ASCII characters. If the memory contents can be translated to an ASCII character, then that character is displayed; otherwise, a dot is displayed.

The silent and iterations fields are often used together to continuously perform read operations without being slowed down by having to be displayed. The repeating cycles can be monitored with test equipment.

Example

This example displays 128 bytes from memory starting with 100000 in byte format.

```
EB64> pb 100000
00100000: 03 00 00 C1 00 00 00 00 10 D9 10 00 00 00 00 00 ................
00100010: 07 00 88 2F 00 00 9E A4 05 14 C1 43 06 14 A1 40 .../.......C...@
00100020: 22 77 80 48 06 04 C2 40 F0 82 DC B4 F8 82 9C B4 "w.H...@........
00100030: 00 83 BC B4 3E 15 C5 43 20 00 FE B7 08 83 FC B3 ....>..C .......
00100040: 07 00 00 D0 04 04 E2 47 19 10 00 D0 80 00 00 00 .......G........
00100050: 1F 04 FF 47 00 00 00 00 00 00 00 00 00 00 00 00 ...G............
00100060: 3E 15 C6 43 28 00 1E B4 36 01 00 D0 18 80 9C A4 >..C(...6.......
00100070: 05 34 E0 43 09 03 00 D0 20 80 9C A4 05 54 E0 43 .4.C.... ....T.C
```

pcishow

The **pcishow** command displays the contents of each PCI slot and the current PCI-to-system address space mapping.

Format

pcishow

Parameters

None.

Description

The **pcishow** command applies only to PCI boards (for example, EB64+ and EB66). This command does not exist on the EB64.

Example

```
EB164> pcishow

PCI Address Mapping windows are:
        (1) PCI Base = 0x00100000, Size = 0x00100000
                Translated Base = 0x00100000

Bus = 0
        primary = 0, secondary = 0, subordinate = 0
        PCI I/O space = 1000, PCI Mem space = 3F00000
        PCI I/O base = B000, PCI Mem base = 200000
PCI slot 18, vendor = 0x1011, device = 0x4
        PCI IO Base = 0x0, PCI IO Size = 0x0
        PCI Mem Base = 0x2000000, PCI Mem Size = 0x2000000
        Display controller
PCI slot 19, vendor = 0x8086, device = 0x484
        PCI IO Base = 0x0, PCI IO Size = 0x0
        PCI Mem Base = 0x0, PCI Mem Size = 0x0
        Non-VGA compatible device
PCI slot 17, vendor = 0x1011, device = 0x2
        PCI IO Base = 0xB000, PCI IO Size = 0x80
        PCI Mem Base = 0x4000000, PCI Mem Size = 0x80
        Ethernet controller
PCI slot 20, vendor = 0x1000, device = 0x1
        PCI IO Base = 0xB400, PCI IO Size = 0x100
        PCI Mem Base = 0x4001000, PCI Mem Size = 0x100
        Non-VGA compatible device

EB164>
```

pfreg

The **pfreg** command displays the saved CPU floating-point register state.

Format

pfreg [address]

Parameters

address
Specifies an alternate address for the saved-state area.

Description

The **pfreg** command displays the contents of the CPU floating-point registers stored in the saved-state area. A register state is stored when a breakpoint is encountered or the PALcode reset flow is entered.

Example

```
EB64> pfreg
Floating Point Registers
register file @: 0000C840
f00: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f04: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f08: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f12: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f16: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f20: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f24: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
f28: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
PC:  000000000000000D   PS:  000000000000000D
```

pl

The **pl** command displays the specified memory longword (32-bit).

Format

pl [start_address [end_address [iterations [silent]]]]

Parameters

start_address
A hexadecimal number that represents a legal address at which to start the display. The default is the current address.

end_address
A hexadecimal number that represents a legal address at which to end the display. The default is the current address plus 127 bytes.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **pl** command displays the specified memory in longword format. If no address is specified, then the current memory longword and the following 31 longwords are displayed. The field displayed after the longword represents the translation of the memory contents in ASCII characters. If the memory contents can be translated to an ASCII character, then that character is displayed; otherwise, a dot is displayed.

The silent and iterations fields are often used together to continuously perform read operations without being slowed down by having to be displayed. The repeating cycles can be monitored with test equipment.

Example

This example displays memory longwords.

```
EB64> pl 0
00000000: E7E01021 00000000 00000000 00000000 !...............
00000010: 00000000 00000000 00000000 00000000 ................
00000020: E7E01095 00000000 00000000 00000000 ................
00000030: 00000000 00000000 00000000 00000000 ................
00000040: 00000000 00000000 00000000 00000000 ................
00000050: 00000000 00000000 00000000 00000000 ................
00000060: 00000000 00000000 00000000 00000000 ................
00000070: 00000000 00000000 00000000 00000000 ................

EB64> pl
00000090: 00000000 00000000 00000000 00000000 ................
000000A0: 00000000 00000000 00000000 00000000 ................
000000B0: 00000000 00000000 00000000 00000000 ................
000000C0: 00000000 00000000 00000000 00000000 ................
000000D0: 00000000 00000000 00000000 00000000 ................
000000E0: 74420082 644200A9 74210081 64210024 ..Bt..Bd..!t$.!d
000000F0: 48405682 F0400013 E4400003 77DE009E .V@H..@...@....w
00000100: 67DE009F 44205401 47E09402 744200A9 ...g.T D...G..Bt

EB64> pl 100000
00100000: C1000003 00000000 0010D910 00000000 ................
00100010: 2F880007 A49E0000 43C11405 40A11406 .../.......C...@
00100020: 48807722 40C20406 B4DC82F0 B49C82F8 "w.H...@........
00100030: B4BC8300 43C5153E B7FE0020 B3FC8308 ....>..C .......
00100040: D0000007 47E20404 D0001019 00000080 .......G........
00100050: 47FF041F 00000000 00000000 00000000 ...G............
00100060: 43C6153E B41E0028 D0000136 A49C8018 >..C(...6.......
00100070: 43E03405 D0000309 A49C8020 43E05405 .4.C.... ....T.C

EB64>pl 100000 100000
00100000: C1000003 00000000 0010D910 00000000 ................
```

pq

The **pq** command displays the specified memory quadword (64-bit).

Format

pq [start_address [end_address [iterations [silent]]]]

Parameters

start_address
A hexadecimal number that represents a legal address at which to start the display. The default is the current address.

end_address
A hexadecimal number that represents a legal address at which to end the display. The default is the current address plus 127 bytes.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **pq** command displays the specified memory in quadword format. If no address is specified, then the current memory quadword and the following 15 quadwords are displayed. The field displayed after the quadword represents the translation of the memory contents in ASCII characters. If the memory contents can be translated to an ASCII character, then that character is displayed; otherwise, a dot is displayed.

The silent and iterations fields are often used together to continuously perform read operations without being slowed down by having to be displayed. The repeating cycles can be monitored with test equipment.

Example

This example displays memory quadwords.

```
EB64> pq
00000000: 00000000E7E01021 0000000000000000 !...............
00000010: 0000000000000000 0000000000000000 ................
00000020: 00000000E7E01095 0000000000000000 ................
00000030: 0000000000000000 0000000000000000 ................
00000040: 0000000000000000 0000000000000000 ................
00000050: 0000000000000000 0000000000000000 ................
00000060: 0000000000000000 0000000000000000 ................
00000070: 0000000000000000 0000000000000000 ................

EB64> pq 100000
00100000: 00000000C1000003 000000000010D910 ................
00100010: A49E00002F880007 40A1140643C11405 .../.......C...@
00100020: 40C2040648807722 B49C82F8B4DC82F0 "w.H...@........
00100030: 43C5153EB4BC8300 B3FC8308B7FE0020 ....>..C .......
00100040: 47E20404D0000007 00000080D0001019 .......G........
00100050: 0000000047FF041F 0000000000000000 ...G............
00100060: B41E002843C6153E A49C8018D0000136 >..C(...6.......
00100070: D000030943E03405 43E05405A49C8020 .4.C.... ....T.C
```

prb

The **prb** command reads a byte (8 bits) from the specified address in the PCI configuration space.

Format

prb pci_address id bus

Parameters

pci_address
Specifies the address in PCI space.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

bus
Specifies which bus to read from. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **prb** command reads a byte from the specified address in the PCI configuation space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to read.

Example

```
EB66> prb 0 19
86
```

preg

The **preg** command displays the saved CPU general-purpose register state.

Format

preg [address]

Parameters

address
Specifies an alternate address for the saved-state area.

Description

The **preg** command displays the contents of the CPU general-purpose registers stored in the saved-state area. A register state is stored when a breakpoint is encountered or the PALcode reset flow is entered.

Example

```
EB64> preg
General Purpose Registers
register file @: 0000C040
r00: 0000000000000020 0000000000000005 000000000000C000 000000000000000D
r04: 00000000000003F8 0000000000000000 0000000000000000 000000000000000D
r08: FFFFFC000005F470 0000000000027340 0444306453605341 0A110C485F6EA26E
r12: 208090EA6024C19C 882C08AA92065B2D 4100610AE100244F 9E2891ACA8A9D984
r16: 0000000000100000 000000000000000D 0000000000000006 0000000000000030
r20: 0000000E20026335 5619A46B2B1A5125 0000000000000000 000000000000000D
r24: 0000000000000003 0000000000000000 FFFFFC0000042C3C 0000000000100000
r28: FFFFFC02C0000000 FFFFFC000006C1E0 0000000000FFDF40 0000000000000003
PC:  000000000000000D   PS:  000000000000000D
```

prl

The **prl** command reads a longword (32 bits) from the specified address in the PCI configuration space.

Format

prl pci_address id bus

Parameters

pci_address
Specifies the address in PCI space.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

bus
Specifies which bus to read from. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **prl** command reads a longword from the specified address in the PCI configuation space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to read.

Example

```
EB66> prl 0 19
04848086
```

prw

The **prw** command reads a word (16 bits) from the specified address in the PCI configuration space.

Format

prw pci_address id bus

Parameters

pci_address
Specifies the address in PCI space.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

bus
Specifies which bus to read from. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **prw** command reads a word from the specified address in the PCI configuation space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to read.

prw

Example

```
AlphaPC64> pcishow

PCI Address Mapping windows are:
        (1) PCI Base = 0x00100000, Size = 0x00100000
                Translated Base = 0x00100000
Bus = 0
        primary = 0, secondary = 0, subordinate = 1
        PCI I/O space = 1000, PCI Mem space = 100000
        PCI I/O base = B000, PCI Mem base = 200000
PCI slot 17, vendor = 0x1011, device = 0x1
        PCI IO Base = 0x0, PCI IO Size = 0x0
        PCI Mem Base = 0x0, PCI Mem Size = 0x0
        PCI-PCI bridge
PCI slot 19, vendor = 0x8086, device = 0x484
        PCI IO Base = 0x0, PCI IO Size = 0x0
        PCI Mem Base = 0x0, PCI Mem Size = 0x0
        Non-VGA compatible device

Bus = 1
        primary = 0, secondary = 1, subordinate = 1
        PCI I/O space = 1000, PCI Mem space = 100000
        PCI I/O base = B000, PCI Mem base = 200000
PCI slot 6, vendor = 0x1011, device = 0x2
        PCI IO Base = 0xB000, PCI IO Size = 0x80
        PCI Mem Base = 0x200000, PCI Mem Size = 0x80
        Ethernet controller

AlphaPC64> prw 0 6 1
1011
AlphaPC64> prw 0 19
8086
```

pw

The **pw** command displays the specified memory word (16-bit).

Format

pw [start_address [end_address [iterations [silent]]]]

Parameters

start_address
A hexadecimal number that represents a legal address at which to start the display. The default is the current address.

end_address
A hexadecimal number that represents a legal address at which to end the display. The default is the current address plus 127 bytes.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **pw** command displays the specified memory in word format. If no address is specified, then the current memory word and the 63 words that follow it are displayed. The field displayed after the word represents the translation of the memory contents in ASCII characters. If the memory contents can be translated to an ASCII character, then that character is displayed; otherwise, a dot is displayed.

The silent and iterations fields are often used together to continuously perform read operations without being slowed down by having to be displayed. The repeating cycles can be monitored with test equipment.

Example

This example displays eight memory addresses starting with 100000_{16} in word format.

```
EB64>pw 100000
00100000: 0003 C100 0000 0000 D910 0010 0000 0000 ................
00100010: 0007 2F88 0000 A49E 1405 43C1 1406 40A1 .../.......C...@
00100020: 7722 4880 0406 40C2 82F0 B4DC 82F8 B49C "w.H...@........
00100030: 8300 B4BC 153E 43C5 0020 B7FE 8308 B3FC ....>..C .......
00100040: 0007 D000 0404 47E2 1019 D000 0080 0000 .......G........
00100050: 041F 47FF 0000 0000 0000 0000 0000 0000 ...G............
00100060: 153E 43C6 0028 B41E 0136 D000 8018 A49C >..C(...6.......
00100070: 3405 43E0 0309 D000 8020 A49C 5405 43E0 .4.C.... ....T.C
```

pwb

The **pwb** command writes a byte (8 bits) to an address in PCI configuration space.

Format

pwb pci_address id data bus

Parameters

pci_address
Specifies which address to write to.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

data
Specifies the value that is written to the pci_address.

bus
Specifies which bus to write to. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **pwb** command writes a byte to the specified address in the PCI configuration space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to write.

Example

```
EB66> prb 4f 19
3F
EB66> pwb 4f 19 2f
EB66> prb 4f 19
2F
```

pwl

The **pwl** command writes a longword (32 bits) to an address in PCI configuration space.

Format

pwl pci_address id data bus

Parameters

pci_address
Specifies which address to write to.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

data
Specifies the value that is written to the pci_address.

bus
Specifies which bus to write to. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **pwl** command writes a longword to the specified address in the PCI configuration space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to write.

Example

```
EB66> pwl 4f 19 0000a6f3
```

pww

The **pww** command writes a word (16 bits) to an address in PCI configuration space.

Format

pww pci_address id data bus

Parameters

pci_address
Specifies which address to write to.

id
Specifies a decimal number that represents the idsel pin on the PCI device. The EB64+ and EB66 support an id value from 11 to 23.

data
Specifies the value that is written to the pci_address.

bus
Specifies which bus to write to. The default value is 0. Use the **pcishow** command to view available buses.

Description

The **pww** command writes a word to the specified address in the PCI configuration space for a device specified by the id. If the board does not support PCI, then this command is not implemented. The EB64 does not support PCI. If your system configuration supports multiple PCI buses, you can use the bus option to specify which bus to write.

Example

```
EB66> pww 4f 19 4
EB66> prw 4f 19
0004
```

rabox

The **rabox** command reads the CPU ABOX_CTL register.

Format

rabox

Parameters

None.

Description

The **rabox** command reads the CPU ABOX_CTL register and displays the value in hexadecimal format.

This command applies only to Alpha boards based on the 21064 and 21066 microprocessors. This command is not implemented for the EB164.

Example

```
EB64> rabox
0000000000000428
```

rb

The **rb** command reads a byte (8 bits) from a register port in I/O address space.

Format

rb register [iterations [silent]]

Parameters

register
Specifies the register in I/O address space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **rb** command reads a byte from the specified register in I/O address space.

Example

```
EB64> rb 370
04
```

rbcfg

The **rbcfg** command reads the backup cache configuration register.

Format

rbcfg

Parameters

None.

Description

The **rbcfg** command reads the shadow copy of the backup cache configuration register.

If you manually change this register by writing to its architected address, the change will not be reflected for this command. You must use the **wbctl** or **wbcfg** command to make any changes.

This command is implemented only for the EB164.

Example

```
EB164> rbcfg
0000000001E22772
```

rbctl

The **rbctl** command reads the backup cache control register.

Format

rbctl

Parameters

None.

Description

The **rbctl** command reads a shadow copy of the backup cache control register.

If you manually change this register by writing to its architected address, the change will not be reflected for this command. You must use the **wbctl** or **wbcfg** command to make any changes.

This command is implemented only for the EB164.

Example

```
EB164> rbctl
0000000000028051
```

rbiu

The **rbiu** command reads the CPU BIU_CTL register.

Format

rbiu

Parameters

None.

Description

The **rbiu** command reads the CPU BIU_CTL register and displays the value in hexadecimal format.

This command applies only to board designs based on the Alpha 21064 microprocessor.

Example

```
EB64> rbiu
0000000E2001C645
```

riccsr

The **riccsr** command reads the CPU ICCSR register.

Format

riccsr

Parameters

None.

Description

The **riccsr** command reads the CPU ICCSR register and displays the value in the CPU write format. For more information about the write format, see the hardware reference manual that corresponds to your CPU chip.

Example

```
EB64> riccsr
000006F800000000
```

rl

The **rl** command reads a longword (32 bits) from a register port in I/O address space.

Format

rl register [iterations [silent]]

Parameters

register
Specifies the register from the I/O address space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **rl** command reads a longword from the specified register in I/O address space.

Example

```
EB64> rl 370
0000A6F3
```

rmode

The **rmode** command sets the **dis** command register display mode.

Format

rmode [status]

Parameters

status
Determines the mode. If set (1), the software register names are displayed. If cleared (0), the hardware register names are displayed. The default is 0.

Description

The **rmode** command specifies whether hardware register names, such as r16, or software register names, such as a0, are displayed with the **dis** command.

The following table displays the Digital UNIX Alpha microprocessor register usage.

rmode

Register Name	Software Name	Use and Linkage
r0	v0	Used for expression and to hold integer function results.
r1..r8	t0..t7	Temporary registers; not preserved across procedure calls.
r9..r14	s0..s5	Saved registers; their values must be preserved across procedure calls.
r15	FP or s6	Frame pointer or a saved register.
r16..r21	a0..a5	Argument registers; used to pass the first six integer type arguments; their values are not preserved across procedure calls.
r22..r25	t8..t11	Temporary registers; not preserved across procedure calls.
r26	ra	Contains the return address; used for expression evaluation.
r27	pv or t12	Procedure value or a temporary register.
r28	at	Assembler temporary register; not preserved across procedure calls.
r29	GP	Global pointer.
r30	SP	Stack pointer.
r31	zero	Always has the value 0.

If you enter the command without a parameter, then the current status is displayed. The **rmode** setting is stored in battery-backed RAM.

Example

```
EB64> rmode
rmode = 0
EB64> dis 243a0
000243A0:  43020122        subl    r24, r2, r2
000243A4:  48441722         sll    r2, 0x20, r2
000243A8:  74420050          mt    r2, cc
000243AC:  64630082          mf    r3, pt2
000243B0:  209F07E1         lda    r4, 2017(zero)
000243B4:  48855724         sll    r4, 0x2A, r4
000243B8:  44640103         bic    r3, r4, r3
000243BC:  47203019         and    r25, 0x1, r25
EB64>dis
000243C0:  4B037698         srl    r24, 0x1B, r24
000243C4:  4703F118         bic    r24, 0x1F, r24
000243C8:  47190418         bis    r24, r25, r24
000243CC:  4B055738         sll    r24, 0x2A, r24
000243D0:  44780403         bis    r3, r24, r3
000243D4:  746300A2          mt    r3, A2
000243D8:  77FF0055          mt    zero, flushIc
000243DC:  77FF0000          mt    zero, 0
EB64>

EB64> rmode 1
EB64> dis 243a0
000243A0:  43020122        subl    t10, t1, t1
000243A4:  48441722         sll    t1, 0x20, t1
000243A8:  74420050          mt    t1, cc
000243AC:  64630082          mf    t2, pt2
000243B0:  209F07E1         lda    t3, 2017(zero)
000243B4:  48855724         sll    t3, 0x2A, t3
000243B8:  44640103         bic    t2, t3, t2
000243BC:  47203019         and    t11, 0x1, t11
EB64> dis
000243C0:  4B037698         srl    t10, 0x1B, t10
000243C4:  4703F118         bic    t10, 0x1F, t10
000243C8:  47190418         bis    t10, t11, t10
000243CC:  4B055738         sll    t10, 0x2A, t10
000243D0:  44780403         bis    t2, t10, t2
000243D4:  746300A2          mt    t2, A2
000243D8:  77FF0055          mt    zero, flushIc
000243DC:  77FF0000          mt    zero, 0
EB64>
```

romboot

The **romboot** command loads the specified image from ROM and begins execution.

Format

romboot [type] [address]

Parameters

type
Specifies the image to load into ROM. If the type is specified as #0, any header information is ignored and the entire contents of the ROM is loaded. The default is to load and execute the first image in the System ROM.

address
Specifies the starting address for loading the image in ROM.

Description

The **romboot** command loads and executes the operating system and associated firmware from the system ROM. Use the **romlist** command to display the images contained in the ROM. You can specify the type as a number or a name.

Type_number	Type_name	Description
0	DBM	Alpha Board Debug Monitor
1	NT	Windows NT
2	VMS	OpenVMS
3	UNIX	Digital UNIX
7	LINUX	Linux, Milo

The **romboot** command can also be used to select a ROM image based on its position in the ROM. Specifying the type as "#0" selects the whole ROM. Specifying the type as "#1" selects the first image; "#2" selects the second image, and so on. You can specify an address to override what is in the image file header. You may also use the **bootadr** command. Use the system reset to reset the board to the initial booted state.

Example

```
AlphaPC64> romboot
Searching for ROM image #1
  Header Size......... 52 bytes
  Image Checksum...... 0x581A (22554)
  Image Size (Uncomp). 117160 (114 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 117160 (114 KB)
  Firmware ID (Opt.).. 0000000000000000  ASCII: ........
  Header Checksum..... 0x8F5C
Loading ROM to address 00300000
Image checksum verified. 0x581A

Loaded 117160 bytes starting at 300000 to 31C9A8
Jumping to 0x300000...
```

romboot

```
AlphaPC64> romboot #2
Searching for ROM image #2
  Header Size......... 52 bytes
  Image Checksum...... 0xD38C (54156)
  Image Size (Uncomp). 211728 (206 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 1 - Windows NT Firmware
  ROM Image Size...... 211728 (206 KB)
  Firmware ID (Opt.).. 0305109502131030  ASCII: 0.......
  Header Checksum..... 0xCED2
Loading ROM to address 00300000
Image checksum verified. 0xD38C

Loaded 211728 bytes starting at 300000 to 333B10
Jumping to 0x300000...

AlphaPC64> romboot unix
Searching for the "Alpha SRM Console".
The specified ROM image was not found

AlphaPC64> romboot nt
Searching for the "Windows NT Firmware".
  Header Size......... 52 bytes
  Image Checksum...... 0xD38C (54156)
  Image Size (Uncomp). 211728 (206 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 1 - Windows NT Firmware
  ROM Image Size...... 211728 (206 KB)
  Firmware ID (Opt.).. 0305109502131030  ASCII: 0.......
  Header Checksum..... 0xCED2
Loading ROM to address 00300000
Image checksum verified. 0xD38C

Loaded 211728 bytes starting at 300000 to 333B10
Jumping to 0x300000...
```

romlist

The **romlist** command lists the ROM image headers contained in ROM.

Format

romlist

Parameters

None.

Description

The **romlist** command searches the system ROM for any ROM image headers that might be present. It then prints a summary for each header found.

Example

```
AlphaPC64> romlist

ROM image header found at offset: 0x000000
  Header Size......... 52 bytes
  Image Checksum...... 0x8111
  Image Size (Uncomp). 129552 (126 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 129552 (126 KB)
  Firmware ID (Opt.).. 0000000000000000  ASCII: ........
  Header Checksum..... 0xA839

ROM image header found at offset: 0x040000
  Header Size......... 52 bytes
  Image Checksum...... 0xD38C
  Image Size (Uncomp). 211728 (206 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 1 - Windows NT Firmware
  ROM Image Size...... 211728 (206 KB)
  Firmware ID (Opt.).. 0305109502131030  ASCII: 0.......
  Header Checksum..... 0xCED2
AlphaPC64>
```

romload

The **romload** command loads the specified image from ROM to the specified address.

Format

romload [type] [address]

Parameters

type
Specifies the image to load into ROM. If the type is specified as #0, then any header information is ignored and the entire contents of the ROM is loaded. The default is to load the first image in the system ROM.

address
Specifies the starting address for loading the image in ROM.

Description

The **romload** command loads the operating system and associated firmware from the system ROM. Use the **romlist** command to display the images contained in the ROM. You can specify the type as a number or a name.

Type_number	Type_name	Description
0	DBM	Alpha Board Debug Monitor
1	NT	Windows NT
2	VMS	OpenVMS
3	UNIX	Digital UNIX
7	LINUX	Linux, Milo

The **romload** command can also be used to select a ROM image based on its position in the ROM. Specifying the type as "#0" selects the whole ROM. Specifying the type as "#1" selects the first image; "#2" selects the second image, and so on.

You can specify an address to override what is in the image file header. You may also use the **bootadr** command. Use the **jtopal** command to execute the image.

Example

```
AlphaPC64> romload #0
Loading entire ROM.
Loading ROM to address 00200000

Loaded 1048576 bytes from 200000 to 300000
AlphaPC64>

AlphaPC64> romload #1
Searching for ROM image #1
  Header Size......... 52 bytes
  Image Checksum...... 0x581A (22554)
  Image Size (Uncomp). 117160 (114 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 117160 (114 KB)
  Firmware ID (Opt.).. 0000000000000000  ASCII: ........
  Header Checksum..... 0x8F5C
Loading ROM to address 00300000
Image checksum verified. 0x581A

Loaded 117160 bytes from 300000 to 31C9A8
AlphaPC64>

AlphaPC64> romload
Searching for ROM image #1
  Header Size......... 52 bytes
  Image Checksum...... 0x581A (22554)
  Image Size (Uncomp). 117160 (114 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 0 - Alpha Evaluation Board Debug Monitor
  ROM Image Size...... 117160 (114 KB)
  Firmware ID (Opt.).. 0000000000000000  ASCII: ........
  Header Checksum..... 0x8F5C
Loading ROM to address 00300000
Image checksum verified. 0x581A

Loaded 117160 bytes from 300000 to 31C9A8
AlphaPC64>

AlphaPC64> romload unix
Searching for "Alpha SRM Console".
The specified ROM image was not found
AlphaPC64>
```

romload

```
AlphaPC64> romload nt
Searching for "Windows NT Firmware".
  Header Size......... 52 bytes
  Image Checksum...... 0xD38C (54156)
  Image Size (Uncomp). 211728 (206 KB)
  Compression Type.... 0
  Image Destination... 0x0000000000300000
  Header Version...... 1
  Firmware ID......... 1 - Windows NT Firmware
  ROM Image Size...... 211728 (206 KB)
  Firmware ID (Opt.).. 0305109502131030  ASCII: 0.......
  Header Checksum..... 0xCED2
Loading ROM to address 00300000
Image checksum verified. 0xD38C

Loaded 211728 bytes from 300000 to 333B10
AlphaPC64>
```

rsys

The **rsys** command reads the EB64 system control register.

Format

rsys

Parameters

None.

Description

The **rsys** command displays the current value of the system register. This command applies only to the EB64.

Example

```
EB64> rsys
840000
```

rw

The **rw** command reads a word (16 bits) from a register port in I/O address space.

Format

rw register [iterations [silent]]

Parameters

register
Specifies the register from the I/O address space.

iterations
Specifies how many times the data is read. The default is 1.

silent
Specifies whether or not the data is displayed. Setting this field to 1 causes the data to be read but not displayed. The default is 0 (data is displayed).

Description

The **rw** command reads a word from the specified register in I/O address space.

Example

```
EB64> rw 372
0000
EB64> rw 370
A6B3
```

sb

The **sb** command searches memory by bytes (8-bit).

Format

sb start_address end_address string [inverse]

Parameters

start_address
Specifies the address at which to begin the search.

end_address
Specifies the address at which to end the search.

string
Specifies the search string.

inverse
Specifies whether to search for a matching string (0) or a nonmatching string (1). The default is 0 (search for matching string).

Description

The **sb** command searches memory by byte chunks for the specified string. An asterisk (*) can be used as a wildcard character for single character matching.

sb

Example

```
EB64> pl 100000 100080
00100000: C3E00007 00000000 00000000 00000000 ................
00100010: 00000000 00000000 00000000 00000000 ................
00100020: 221F0000 26100012 6BF00000 00000000 ..."...&...k....
00100030: 00000000 00000000 00000000 00000000 ................
00100040: 00000000 00000000 00000000 00000000 ................
00100050: 00000000 00000000 00000000 00000000 ................
00100060: 00000000 00000000 00000000 00000000 ................
00100070: 00000000 00000000 00000000 00000000 ................
00100080: 00000000 00000000 00000000 00000000 ................

EB64> sb 100000 100080 2*
val = 20  mask = F0
occurrence at 00100023 22
occurrence at 00100027 26

EB64> sb 100000 100080 1*
val = 10  mask = F0
occurrence at 00100022 1F
occurrence at 00100024 12
occurrence at 00100026 10

EB64> sb 100000 100080 1f
val = 1F  mask = FF
occurrence at 00100022 1F
```

setbaud

The **setbaud** command sets the baud rate for the specified communication port connection.

Format

setbaud port baud_rate

Parameters

port
Specifies the number identifier for the keyboard or serial port.

baud_rate
Specifies the baud rate for the specified port. The default is 9600.

Description

The **setbaud** command sets the baud rate for the specified keyboard or serial communication port. The baud rate can be set to 1200, 2400, 9600, 19200, or 38400.

The following table shows the port identifier numbers.

Port ID	Port Name
0	Keyboard port
1	Serial communication port 1
2	Serial communication port 2

Example

```
EB64> setbaud 1 2400
```

setty

The **setty** command sets the debug monitor to the specified port.

Format

setty port

Parameters

port
Specifies the number identifier for the keyboard or serial port.

Description

The **setty** command specifies the port used for debug monitor interaction. The following table shows the port identifier numbers.

Port ID	Port Name
0	Keyboard port
1	Serial communication port 1
2	Serial communication port 2

Example

```
EB64> setty 1
```

sl

The **sl** command searches memory by longwords (32-bit).

Format

sl start_address end_address string [inverse]

Parameters

start_address
Specifies the address at which to begin the search.

end_address
Specifies the address at which to end the search.

string
Specifies the search string.

inverse
Specifies whether to search for a matching string (0) or a nonmatching string (1). The default is 0 (search for matching string).

Description

The **sl** command searches memory by longword chunks for the specified string. An asterisk (*) can be used as a wildcard character for single character matching.

sl

Example

```
EB64> pl 100000
00100000: C3E00007 00000000 00000000 00000000 ................
00100010: 00000000 00000000 00000000 00000000 ................
00100020: 221F0000 26100012 6BF00000 00000000 ..."...&...k....
00100030: 00000000 00000000 00000000 00000000 ................
00100040: 00000000 00000000 00000000 00000000 ................
00100050: 00000000 00000000 00000000 00000000 ................
00100060: 00000000 00000000 00000000 00000000 ................
00100070: 00000000 00000000 00000000 00000000 ................

EB64> sl 100000 100070 2*******
val = 20000000  mask = F0000000
occurrence at 00100020 221F0000
occurrence at 00100024 26100012
EB64> sl 100000 100070 2*1*****
val = 20100000  mask = F0F00000
occurrence at 00100020 221F0000
occurrence at 00100024 26100012
```

sq

The **sq** command searches memory by quadwords (64-bit).

Format

sq start_address end_address string [inverse]

Parameters

start_address
Specifies the address at which to begin the search.

end_address
Specifies the address at which to end the search.

string
Specifies the search string.

inverse
Specifies whether to search for a matching string (0) or a nonmatching string (1). The default is 0 (search for matching string).

Description

The **sq** command searches memory by quadword chunks for the specified string. An asterisk (*) can be used as a wildcard character for single character matching.

sq

Example

```
EB64> pq
00000000: 00000000C3E00007 0000000000000000 ................
00000010: 0000000000000000 0000000000000000 ................
00000020: 26100002221F0000 000000006BF00000 ..."...&...k....
00000030: 0000000000000000 0000000000000000 ................
00000040: 0000000000000000 0000000000000000 ................
00000050: 0000000000000000 0000000000000000 ................
00000060: 0000000000000000 0000000000000000 ................
00000070: 0000000000000000 0000000000000000 ................

EB64> sq 100000 100080 2
val = 2  mask = FFFFFFFFFFFFFFFF
value not found

EB64> sq 100000 100080 26100002221F0000
value = 26100002221F0000  mask = FFFFFFFFFFFFFFFF
occurrence at 00000020 26100002221F0000
```

step

The **step** command executes the next instruction after a breakpoint.

Format

step

Parameters

None.

Description

The **step** command allows the user to single step through the code after a breakpoint.

Example

```
EB64> stop 100000
EB64> go
Executing at 0x100000...
00100000:  C1000003          br    r8, 100010
EB64> step
00100010:  2F880007       ldq_u    r28, 7(r8)
EB64> step
00100014:  A49E0000         ldq    r4, 0(sp)
EB64> step
00100018:  43C11405        addq    sp, 0x8, r5
EB64> step
0010001C:  40A11406        addq    r5, 0x8, r6
```

stop

The **stop** command sets a breakpoint.

Format

stop address

Parameters

address
Specifies the address at which the breakpoint is set.

Description

The **stop** command sets a breakpoint at the specified address. When a breakpoint is encountered, all current register values are stored in memory and can be viewed with the **preg** and **pfreg** commands.

Example

```
EB64> stop 100000
EB64> go
Executing at 0x100000...
00100000:  C1000003          br    r8, 100010


EB64> stop 100200
EB64> go
Executing at 0x100000...
00100200:  4A671793         sra    r19, 0x38, r19
EB64> cont
00100200:  4A671793         sra    r19, 0x38, r19

This simple program prints the size of
various data types in bytes.
 char   =  1
 short  =  2
 int    =  4
 long   =  8
 float  =  4
 double =  8

Alpha 21064 Evaluation Board (EB64) Debug Monitor
 Version: Fri Apr 09 20:50:11 EDT 1993
 Bootadr: 0x100000, memSize: 0x2000000
```

sum

The **sum** command computes the checksum of the data in the specified range.

Format

sum start_address end_address

Parameters

start_address
Specifies the address at which the checksum check begins.

end_address
Specifies the address at which the checksum check ends.

Description

The **sum** command prints the checksum of the data contained in the specified memory range. The algorithm used computes a 16-bit checksum, and is compatible with the standard BSD4.3 algorithm provided in most implementations of UNIX (sum), thus allowing easy comparisons of images in the board's memory with those on the UNIX host.

Example

```
AlphaPC64> netload pc64dbm.rom
Digital Semiconductor 21040 (0): Initializing
        Hardware address = BA-98-76-54-32-10
        Trying 10 Base T
        Switching to AUI
MAC address:  BA-98-76-54-32-10
Attempting BOOTP...
Loading /sae_share/boot/user1/pc64/pc64dbm.rom at 0x300000
 My IP address:     16.123.45.67
 Server IP address: 16.123.45.69
##################
File loaded successfully.  Size = 0x30B80 (199552)

AlphaPC64> sum 300000 330B7F
0xe5cc 58828
```

sw

The **sw** command searches memory by words (16-bit).

Format

sw start_address end_address string [inverse]

Parameters

start_address
Specifies the address at which to begin the search.

end_address
Specifies the address at which to end the search.

string
Specifies the search string.

inverse
Specifies whether to search for a matching string (0) or a nonmatching string (1). The default is 0 (search for matching string).

Description

The **sw** command searches memory by word chunks for the specified string. An asterisk (*) can be used as a wildcard character for single character matching.

Example

```
EB64> pl 100000 100080
00100000: C3E00007 00000000 00000000 00000000 ................
00100010: 00000000 00000000 00000000 00000000 ................
00100020: 221F0000 26100012 6BF00000 00000000 ..."...&...k....
00100030: 00000000 00000000 00000000 00000000 ................
00100040: 00000000 00000000 00000000 00000000 ................
00100050: 00000000 00000000 00000000 00000000 ................
00100060: 00000000 00000000 00000000 00000000 ................
00100070: 00000000 00000000 00000000 00000000 ................
00100080: 00000000 00000000 00000000 00000000 ................

EB64> sw 100000 100080 2*1*
val = 2010  mask = F0F0
occurrence at 00100022 221F
occurrence at 00100026 2610
```

sysshow

The **sysshow** command displays all SROM parameters.

Format

sysshow

Parameters

None.

Description

The **sysshow** command displays the system status passed from the SROM at initialization or reset. Refer to your boards user's guide for more information about the SROM parameters displayed.

Example

```
EB66> sysshow
abox_ctl :      428
bcr0     :     64C0     bcr1     :  10064C0
bcr2     :        0     bcr3     :        0
bmr0     :   F00000     bmr1     :   F00000
bmr2     :        0     bmr3     :        0
srom_rev :     1805     proc_id  :        4
mem_size :  2000000     cycle_cnt:     1771
signature: DECB0001     proc_mask:        1
sysctx   :        0     valid    :        1
```

tip

The **tip** command connects to the specified serial communication port.

Format

tip port

Parameters

port
Specifies the serial port.

Description

The **tip** command is a subset of the Digital UNIX `tip` command. It allows you to connect directly from the board to the specified serial communication port. You can specify 1 for serial port 1, or specify 2 for serial port 2.

Example

In this example, the host system is connected to serial port 1.

```
EB64> tip 1
```

version

The **version** command displays the current debug monitor version information.

Format

version

Parameters

None.

Description

The **version** command displays the current debug monitor version information. This information is also displayed in the banner when you power on the board.

Example

```
EB64> version
Wed Feb 10 19:52:24 EST 1993
```

wabox

The **wabox** command writes to the CPU ABOX_CTL register.

Format

wabox data

Parameters

data
Specifies the new value written to the register.

Description

The **wabox** command writes to the CPU ABOX_CTL register. The board does not check for valid register values.

This command applies only to Alpha boards based on the 21064 and 21066 microprocessors. This command is not implemented for the EB164.

Example

```
EB64> rabox
0000000000000428
EB64> wabox 418
EB64> rabox
0000000000000418
```

wb

The **wb** command writes a byte (8 bits) to a register port in I/O address space.

Format

wb register data [iterations]

Parameters

register
Specifies which register to write to.

data
Specifies the value that is written to the register.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **wb** command writes a byte to the specified register in I/O address space.

Example

```
EB64> rb 280
28
EB64> wb 280 68
EB64> rb 280
68
```

wbcfg

The **wbcfg** command writes the backup cache configuration register.

Format

wbcfg bcfg_data [bctl_data]

Parameters

bcfg_data
Specifies the new backup cache configuration register value.

bctl_data
Specifies the new backup cache control register value. If not supplied, the current value remains unchanged.

Description

The **wbcfg** command allows you to write to the backup cache configuration register and the backup cache control register in the same command. If you are making a change to the configuration register that requires a change to the control register, specify both values in a single write to prevent the CPU from being in an inconsistent state. If the change you are making to the backup cache configuration register does not require a change to the control register, the second parameter is optional. On the EB164, the memory controller registers are automatically changed to reflect the new state of the backup cache.

This command is implemented only for the EB164.

Example

```
EB164> wbcfg 1e22772 28051
Old BC_CTL = 0x00008051  &  BC_CFG = 0x01E21772
New BC_CTL = 0x00028051  &  BC_CFG = 0x01E22772
    CIA_CACK_EN = 0x8  &  CIA_MCR = 0x0001FE21
```

wbctl

The **wbctl** command writes the backup cache control register.

Format

wbctl bctl_data [bcfg_data]

Parameters

bctl_data
Specifies the new backup cache control register value. If not supplied, the current value remains unchanged.

bcfg_data
Specifies the new backup cache configuration register value.

Description

The **wbctl** command allows you to write to the backup cache control register and the backup cache configuration register in the same command. If you are making a change to the control register that requires a change to the configuration register, specify both values in a single write to prevent the CPU from being in an inconsistent state. If the change you are making to the backup cache control register does not require a change to the configuration register, the second parameter is optional. On the EB164, the memory controller registers are automatically changed to reflect the new state of the backup cache.

This command is implemented only for the EB164.

Example

```
EB164> wbctl 8051
Old BC_CTL = 0x00028051  &  BC_CFG = 0x01E21772
New BC_CTL = 0x00008051  &  BC_CFG = 0x01E21772
    CIA_CACK_EN = 0x8  &  CIA_MCR = 0x0001FE21
```

wbiu

The **wbiu** command writes to the CPU BIU_CTL register.

Format

wbiu data

Parameters

data
Specifies the new value written to the register.

Description

The **wbiu** command writes to the CPU BIU_CTL register. The board does not check for valid register values.

Caution

Bit 2 of the BIU_CTL register cannot be cleared with this command. Setting the OE could damage the EB64. If you are writing your own software and accessing the BIU_CTL register, set bit 2 of this register to 1 to avoid potential damage to your hardware.

This command applies only to board designs based on the Alpha 21064 microprocessor.

Example

```
EB64> rbiu
0000000E2001C645
EB64> wbiu E2001c545
EB64> rbiu
0000000E2001C545
```

wiccsr

The **wiccsr** command writes to the CPU ICCSR register.

Format

wiccsr data

Parameters

data
Specifies the new value written to the register.

Description

The **wiccsr** command writes to the CPU ICCSR register. The board does not check for valid register values.

Example

```
EB64> riccsr
000006F800000000
EB64> wiccsr 6f900000000
EB64> riccsr
000006F900000000
```

wl

The **wl** command writes a longword (32 bits) to a register port in I/O address space.

Format

wl register data [iterations]

Parameters

register
Specifies which register to write to.

data
Specifies the value that is written to the register.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **wl** command writes a longword to the specified register port in I/O address space.

Example

```
EB64> wl 370 0000a6f3
```

wsys

The **wsys** command writes to the EB64 system control register.

Format

wsys data

Parameters

data
The specified value becomes the new value of the system register.

Description

The **wsys** command allows you to modify the contents of the EB64 system register. This command applies only to the EB64.

Example

```
EB64> rsys
840000
EB64> wsys 177700
EB64> rsys
177700
```

ww

The **ww** command writes a word (16 bits) to a register port in I/O address space.

Format

ww register data [iterations]

Parameters

register
Specifies which register to write to.

data
Specifies the value that is written to the register.

iterations
Specifies how many times the data is read. The default is 1.

Description

The **ww** command writes a word to the specified register in the I/O address space. For example, on the EB64 the word is written to the ISA extension slot.

Example

```
EB64> ww 370 4
EB64> rw 370
0004
```

A

Technical Support

A.1 Technical Support

If you need technical support with your Alpha PCI Motherboard, contact your local Digital representative. Please provide your local representative with the model number and if possible a brief description of the problem you are encountering.

Additional technical documentation is available from Digital on the major Digital semiconductor components used on your PCI Motherboard. A complete list of these documents can be obtained from your local representative.

And be sure to visit Digital Equipment's home page at UIC:

```
http://www.digital.com
```

Select the Semiconductor InfoCenter for pointers to relevant technical documentation.

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W