PMON
Users Manual

algorithmics

© 1998 Algorithmics Ltd
© 1992,1993 LSI Logic Corporation

Revision: 1.4
Dated: 1998/01/30

Algorithmics' MIPS architecture single-board computers come equipped with the PMON PROM bootstrap/monitor program. This manual tells you how to work with PMON to load and run programs.

The original version of the PMON program was written by Phil Bunce for LSI Logic Inc, who have kindly made the sources freely reusable without royalty.

Algorithmics will provide any customer (on request) with sources of our version of PMON for a media charge only - free for those with internet *ftp* access. Call Algorithmics for support or assistance with porting to your hardware. US customers may also be able to get support from Phil Bunce: email pjb@carmel.portal.com or call (408) 625 1247.

The PMON program includes software developed by the University of California, Berkeley and its contributors. Specifically, the ethernet UDP/IP protocol stack and its support software come from the BSD4.3/net software release.

The following manual was derived from LSI Logic Corporation's PMON V4.0 manual, and thus constitutes a derivative work.

Algorithmics Ltd
3 Drayton Park
London N5 1NU
ENGLAND.

Phone: +44 71 700 3301
Fax: +44 71 700 3400
Email: sales@algor.co.uk, pmon@algor.co.uk

LSI Logic Corporation
1551 McCarthy boulevard
Milpitas
CA 95035
USA.

Contents

Summary of PMON

This manual describes PMON, a PROM Monitor originally written for LSI Logic and made available as re-usable source code. Algorithmics have extended and adapted PMON to run on products such as the P−4000i prototyping board, and will give you support and advice on applying PMON to your own hardware platform. This document contains the following sections:

- Section 1, Introduction
- Section 2, Monitor Environment
- Section 3, Download Record Types
- Section 4, Downloading Files via RS-232C
- Section 5, Connecting to Ethernet
- Section 6, Monitor Command Summary
- Section 7, Alphabetic Command Listing
- Section 8, Using PMON with SDE−MIPS
- Section 9, Power-on self-test description
- Section 10, Glossary of frequently-used words and acronyms
- Section 11, Reference list (other literature which may be relevant).

1. Introduction

PMON provides a fundamental set of commands for debugging software applications that run on MIPS architecture microprocessors. LSI originally commissioned it to run on the LR33000 Self-Embedding(TM) processor and its derivatives; Algorithmics adapted PMON to run on the 64-bit R4x00 family.

PMON is provided in PROM on the P−4000i.

Use of PMON involves three basic steps:

1. Users first develop code on their own workstations or PCs, and cross-compile the code on these host machines.
2. Users then download their code to the target board using either an RS−232C or Ethernet link.
3. PMON's debugging facilities permit users to:
   - start and stop program execution at any point via hardware and software breakpoints and single-step execution support.
   - read any register contents and set any register to a new value.
   - read data and disassemble code from main memory.
   - copy memory contents from one area to another, fill areas of memory with a particular byte or string, or search memory for a particular byte or string.

PMON provides a flexible environment for users to perform debugging. Command history, command-line editing, and downloadable symbols are only a few of PMON's features. A script feature lets users execute a specified sequence of PROM Monitor commands after reaching a breakpoint, or on reset. If the user enters an illegal command, on-line help provides syntactic information. PMON can also function as the target system back-end for a symbolic debugger running on a host machine; compatible debuggers include `gdb` (as provided in Algorithmics' SDE−MIPS toolkit, or in GNU toolchains obtained from the Free Software Foundation), or the MIPS/SGI version of `dbx`. In the rest of this document we will refer only to `gdb`.

Users can incorporate any of the code from the PMON source package into their own products with no redistribution or royalty fees. PMON software support, porting and installation help are available from Algorithmics.

Separate from PMON, but part of the PROM software provided with most of Algorithmics' board-level products, is the AlgPOST self-test suite. Section 9 tells you about the test software − in this manual, the version which runs on the P−4000i board.

1.1. PROM Monitor Commands

PMON supports 34 commands. The commands can be grouped into the following four categories:

- Execution Control

  `b` (breakpoint), `c` (continue), `call`, `db` (display or delete breakpoints), `g` (go), `t` (trace), and `to` (trace over).

- Display and Modify

  `boot` (network loader), `bt` (stack backtrace), `copy`, `d` (display), `dump`, `fill`, `l` (disassemble), `load` (download), `m` (memory display/modify), `r` (register display/modify), `search` (search memory for pattern), and `tlb` (display tlb entries).

- Environment

  `date` (display/set date & time), `hi` (history), `ls` (list symbols), `more` (screen-at-a-time display), `set` (set environment variables), `sh` (command shell), `stty` (set terminal options), `sym` (set symbolic name), and `unset` (remove environment variables).

- Miscellaneous

  `debug` (enter remote debug mode). `flush` (flush data and/or instruction cache), `h` (help), `mt` (memory test), `off` (switch off machine, if it has a soft off-switch), `ping` (network test), `reboot` (reboot PMON), and `tr` (transparent mode).

Table 1.1 lists the commands available in PMON, indicates their functional category, and refers the reader to more information about the command later in this document. Readers may wish to keep a photocopy of Table 1.1 as a loose-leaf marker in the book for fast reference.

Command	*Page No*	*Function*	*Execution Control*	*Display & Modify*	*Environment*	*Miscellaneous*
b	27	Breakpoint	×			
boot	29	Download binary via Ethernet		×		
bt	30	Stack backtrace		×		
c	31	Continue	×			
call	32	Call	×			
copy	33	Copy		×		
d	34	Display		×		
date	35	Display/set date and time			×	
db	36	Delete breakpoint	×			
debug	37	Debug				×
dump	38	Upload via RS232/Ethernet		×		
eset	40	Edit variable			×	
fill	41	Fill		×		
flush	42	Flush data and/or instruction cache				×
g	43	Go	×			
h	44	Help				×
hi	45	History			×	
l	46	List (disassemble)		×		
load	47	Download via RS232/Ethernet		×		
ls	49	List symbols			×	
m	50	Memory display/modify		×		
more	52	More			×	
mt	53	Memory test				×
off	56	Switch off				×
ping	54	Network setup test				×
r	57	Register display/modify		×		
reboot	59	reboot PMON				×
search	60	Search		×		
set	61	Set variable			×	
sh	63	Command shell			×	
stty	66	Display/set terminal options			×	
sym	68	Set symbolic name			×	
t	69	Trace (single step)		×		
tlb	70	Display TLB entries				×
to	69	Trace (step over)		×		
tr	71	Transparent mode				×
unset	72	Remove variable			×	

Table 1.1: PROM Monitor Commands

1.2. Features

PMON offers:

- No redistribution or royalty fees;
- Complete source code for PMON available on request (but note this does not include code for AlgPOST);
- Drivers for standard UARTs;
- Register display and set with named fields;
- Memory display and set in hexadecimal notation;
- Memory disassembly;
- Memory copy, fill, and search;
- Motorola S-record upload and download facility;
- Exclusive high-speed download format (the *FastLoad Format*);
- Download of ELF or ECOFF binary (object) files over Ethernet via *TFTP*;
- C Shell-style command history support;
- Emacs-style command line editing;
- Downloadable symbol tables to support symbolic addresses;
- On-line help;
- 32 software breakpoints;
- Hardware breakpoint (if supported by your CPU variant).
- User-defined command list execution on breakpoint;
- Single-step execution;
- Source-level debugging using MIPS-targeted versions of `gdb`;
- Local, remote, and transparent connection modes to a host;
- Entry points for user I/O service requests;
- Initialisation via NVRAM;

2. Monitor Environment

PMON defines a number of environment variables, which influence the interpretation and execution of various commands. These environment variables are listed in Table 2.1, together with the default values set for the variables in PMON and the possible values allowed for the variables.

The environment variables are stored in non-volatile memory, so that they are retained even when the board is switched off. In addition to those variables required by PMON, the user may define additional variables in which to save arbitrary strings, such as file names, command strings, etc.

Environment Variable	*Default Value*	*Contents*
autoboot		command list
bootaddr		internet address
bootdelay	20	1–60
bootfile		string
brkcmd	"l @epc 1"	command list
broadcast		internet address
datasz	-b	[-b|-h|-w|-d]
dlecho	off	[off|on|lfeed]
dlproto	EtxAck	[none|XonXoff|EtxAck]
ethaddr	00:40:bc:03:00:00	ethernet address
gateway		internet address
heaptop	80020000	string
hostname		string
hostport	tty1	tty0–9
inalpha	hex	[hex|symbol]
inbase	16	[auto|8|10|16]
loglevel	notice	[debug|info|notice|warning|err|crit|alert|emerg]
moresz	10	0–n
nameserver		internet address
netaddr		internet address
netmask		internet address
prompt	"PMON> "	string
regstyle	sw	[hw|sw]
regsize	32	[32|64]
rptcmd	trace	[off|on|trace]
tftphost		internet address
trabort	ˆK	char
tty0	9600	baudrate
tty1	9600	baudrate
ulcr	cr	[cr|lf|crlf]
uleof	%	string
validpc	"_ftext etext"	string

Table 2.1: PROM Monitor Environment Variables and Default Values

Environment variables can be displayed and specified using the `set` command (see the description of the `set` command later in this chapter). User-defined variables can be removed when no longer required by using the `unset` command.

Brief descriptions of each of the variables in Table 2.1 follow, together with references to their complete descriptions later in this chapter.

- *autoboot* – This variable, if defined, is a list of commands to be executed automatically when the board is reset. For example, it could load a program over the network, and start it running.
- *bootaddr* – This variable specifies an Internet host from which to load files, if one is not specified on the `boot` command line.
- *bootdelay* – This variable specifies how many seconds to wait after a reset before executing the command line in the *autoboot* variable (default: 20 seconds). The user can interrupt the delay by pressing any key. This variable is ignored if *autoboot* is not defined.
- *bootfile* – This variable gives the name of a file to be loaded by the network loader `boot` command, if no file name is specified on the command line. See page 29.
- *brkcmd* – This variable specifies a sequence of Monitor commands that are executed when a breakpoint halts program execution. See the `b` command on page 27.

- *broadcast* – This variable, if defined, overrides the default internet broadcast address for this board. See page 17 for more information about connecting the board to Ethernet.
- *datasz* – This variable controls whether data is displayed in byte, half-word, word or double-word groups. See the `d` command on page 34.
- *dlecho* – This variable controls whether the target board echoes on downloads. An entire line can be echoed, a single line-feed character can be echoed, or there can be no echo at all. See the `load` command on page 47 and the section on downloading on page 13.
- *dlproto* – This variable selects the download protocol for transfers via RS-232C. The Monitor supports Xon/Xoff and EtxAck download protocols. See the `load` command on page 47 and the section on downloading on page 13.
- *ethaddr* – This variable specifies the hardware Ethernet address. See the `boot` command on page 29 and the section on setting up Ethernet on page 17.
- *gateway* – This variable, if defined, is the internet host to which packets should be sent if they are addressed to other networks (the default gateway). See page 17 for more information about connecting the board to Ethernet.
- *heaptop* – This variable specifies the highest allowable address in the heap maintained by PMON. See the `load` command on page 47, and the `boot` command on page 29.
- *hostname* – This variable gives the symbolic Internet host name of this board. See page 17 for more information about connecting the board to Ethernet.
- *hostport* – This variable selects whether tty0 or tty1 is used as the default port for downloading, uploading and remote debugging. See the `load` command on page 47, the `dump` command on page 38, the `debug` command on page 37, and the section on downloading on page 13.
- *loglevel* – This variable selects how talkative the networking code should be. The above list of options are a set of priorities, in order of increasing severity. Any message of the specified priority or above will be displayed on the console.
- *inalpha* – This variable selects whether strings starting with the ASCII characters '`a`' through '`f`' are interpreted as symbols or hexadecimal numbers. See the `sh` command on page 63.
- *inbase* – This variable selects the default input base for numeric values. Users can input octal, decimal, or hexadecimal numbers by changing this variable. See the `sh` command on page 63.
- *moresz* – This variable specifies how many lines to display during screen-at-a-time display. See the `more` command on page 52.
- *nameserver* – This variable, if defined, specifies the numeric Internet address (in standard dot-notation) of the local domain's DNS name server. See page 17 for more information about connecting the board to Ethernet.
- *netaddr* – This variable specifies the numeric Internet address (in standard dot-notation) for this board. See page 17 for more information about connecting the board to Ethernet.
- *netmask* – This variable, if defined, overrides the default Internet *netmask* for this board. See page 17 for more information about connecting the board to Ethernet.
- *prompt* – This variable defines the Monitor prompt. See the `sh` command on page 63.
- *regsize* – This variable specifies whether CPU registers should be displayed as 32-bits or 64-bits wide. See the `r` command on page 57. Note that although PMON supports R4000 64-bit registers, it does not support 64-bit addressing mode.

- *regstyle* – This variable defines whether hardware or software names are displayed for the CPU registers in the `l` command. See the `l` command on page 46.
- *rptcmd* – When this variable is set to "`on`", the previous command is executed again when the user enters an empty line. When it is set to "`trace`", then the previous command is executed again only if it was `t` or `to`. See the `sh` command on page 63.
- *tftphost* – If set, this variable specifies a default internet hostname to use for TFTP network file accesses.
- *trabort* – This variable selects the character that terminates transparent mode and returns the Monitor to command mode. See the `tr` command on page 71.
- *tty0* – If specified, then this gives the default baud rate for the tty0 RS232-C port.
- *tty1* – If specified, then this gives the default baud rate for the tty1 RS232-C port.
- *ulcr* – This variable defines whether there is a carriage return, a line feed, or both at the end of the line during dumps. See the `dump` command on page 38.
- *uleof* – This variable specifies a string that is sent to the host after a dump to the target has completed. See the `dump` command on page 38.
- *validpc* –This variable specifies the range of valid PC values during program tracing. It consists of up to 5 pairs of addresses, and an EPC register must lie within the one of these pairs to be considered valid. See the `trace` command on page 69, and the `bt` command on page 30.

3. Download Record Types

PMON supports four download formats: LSI Logic's FastLoad Format, Motorola S–records, and (via Ethernet) ELF and ECOFF binary object files. All these formats can support the downloading of symbols, which is useful for debugging.

The FastLoad Format uses a compressed ASCII format that permits files to be downloaded in less than half the time taken for Motorola S-records. Motorola S-records have been extended to include a non-standard S4–record containing an address and a symbol.

4. Downloading Files via RS-232C

This section provides information on downloading programs and data from a host machine to a target board using a serial RS-232C link.

The next three subsections address the following topics:

- Choosing the Number of Target Ports
- Choosing and Setting Baud Rates
- Flow Control
- Examples

Choosing the Number of Target Ports

You may use one or two ports on the target board. In single-port mode, a single port on the host system is connected to tty0 on the target. Communication with the target system for issuing PROM Monitor commands and transferring files is performed using a terminal emulation program on the host.

In two-port mode, you can connect either a dumb terminal or a terminal emulation program to tty0 on the target board, with the transfer of files performed on the second serial port (tty1) using, for example, the SDE–MIPS `edown` program.

For single-port mode, set the environment variable *hostport* to "tty0". For two-port mode, set *hostport* to "tty1".

Choosing and Setting Baud Rates

To minimize download time, use the highest mutually acceptable baud rate for the host and target.

On the target, use the `stty` command to set the baud rate. For example, to set the baud rate on the target board's tty1 port to 19200 baud, enter the following command line on the target:

```
PMON> stty tty1 19200
```

You can also change the baudrate permanently, using the *tty0* and *tty1* environment variables:

```
PMON> set tty1 19200
```

On the host, the way the user sets the baud rate depends on the host type and the downloading method. If you are using `tip`, for example, you can specify the host's baud rate on the command line while initiating a download. For example, to download via `tip` at 19200 baud, enter the following command line on the host:

```
% tip -19200 hardwire
```

You can also specify the host's baud rate default value for `tip` in the file `/etc/remote`.

You can also specify the host's baud rate for the `edown` command on the command line while initiating a download.

Refer to your UNIX documentation for more information on the `tip` command. For more information on `edown`, see the description in the SDE–MIPS manual.

Flow Control

A flow-control protocol is selected to ensure that the host does not send data too fast for the target to receive. Although the target system may be able to read a record at 9600 baud, the target system may need time to process that record before it can read the next record.

The environment variables *dlproto* and *dlecho* specify the flow-control protocol. Table 4.1 summarizes the four protocols supported by the PROM Monitor.

Host Sends	*Target Returns*	*Set*	*Application*
Line terminated by carriage return.	Echoes same line.	dlecho = on dlproto = none	`tip` in single-port mode; UNIX host in single-port mode.
Line terminated by carriage return.	Returns line-feed character.	dlecho = lfeed dlproto = none	CrossTalk running on IBM PC.
Line terminated by carriage return.	Returns Xoff and Xon characters.	dlecho = off dlproto = XonXoff	`cat` in dual-port mode; UNIX host in dual-port mode.
Line terminated by ETX character.	Returns ACK character.	dlecho = off dlproto = EtxAck	`edown` in dual-port mode

Table 4.1: Flow control protocols

The following paragraphs describe the four supported flow-control protocols. In each case, the protocol itself is first described. An example showing how to set the *dlecho* and *dlproto* environment variables follows. Concluding each description is an indication of the suggested application cases for the protocol.

- The host sends one line terminated by a carriage-return character. The host then waits for the target to echo the line before sending the next line.

```
PMON> set dlecho on
PMON> set dlproto none
```

  This protocol is appropriate for use with `tip` in single-port mode and with UNIX host systems operating in single-port mode.

- The host sends one line terminated by a carriage-return character. The host then waits for the target to echo a line-feed character before sending the next line.

```
PMON> set dlecho lfeed
PMON> set dlproto none
```

  This protocol is appropriate for use with CrossTalk running on an IBM PC in single port-mode.

- The host sends each line terminated by a carriage-return character. Xoff (ˆS) and Xon (ˆQ) characters sent from the target are used to pause the host between lines.

```
PMON> set dlecho off
PMON> set dlproto XonXoff
```

  This protocol is appropriate for use with `cat` in two-port mode. The user can also use this mode for UNIX host systems operating in dual-port mode.

- The host sends one line terminated by ETX and waits for ACK before continuing.

```
PMON> set dlecho off
PMON> set dlproto EtxAck
```

  This protocol is appropriate for use with `edown` in two-port mode. The user can also use this mode instead of the last mode for UNIX and PC hosts systems operating in dual-port mode.

Examples

This section provides examples of downloading files using single-port and two-port modes.

Single-Port Mode

If the target board is operating in single-port mode, you can use the UNIX `tip` command to download compiled and linked files from your host machine.

The following example illustrates how to perform this procedure step by step.

`% make-sde ex4ram`	*Compile and link on host.*
`% tip hardwire`	*Establish connection with target.*
`PMON> set hostport tty0`	*Initial setup on target*
`PMON> set dlecho on`	*(only required once).*
`PMON> set dlproto none`	
`PMON> load`	*Prepare for download.*
`PMON> ˜> bubble.lsi`	*Start download.*
`PMON> g`	*Run the downloaded program.*

Two-Port Mode

If the target board is operating in two-port mode, you can use the `edown` command to download compiled and linked files from your host machine. The following example illustrates how to perform this procedure step by step.

`% make-sde ex4ram`	*Compile and link on host.*
`PMON> set hostport tty1`	*Initial setup on target.*
`PMON> set dlecho off`	*(only required once).*
`PMON> set dlproto XonXoff`	
`PMON> load`	*Prepare for download.*
`% edown -d /dev/ttyb bubble.lsi`	*Start download.*
`PMON> g`	*Run the downloaded program.*

5. Connecting to Ethernet

This section contains information on setting up your board on an Ethernet so that you can download programs from a host machine to the target board using a high-speed Ethernet link.

5.1. Configuring the Board

PMON uses several environment variables to control its access to Ethernet. Some of these must be set to fit in with your local network, and you may need to ask your network manager.

See the `set` and `eset` commands, on pages 61 and 40 respectively, for details of how to set and change environment variables. After you change any network related variables you must then reset the board before they will have any effect.

Minimum setup

The essential variables that will have to be set before you can perform any network communication are:

- *ethaddr* defines the board's Ethernet address, uniquely assigned to the board when it was made. This variable is configured at the factory and will only need to be reset if the NVRAM contents have got lost.

  For Algorithmics' boards the ethernet number always takes the form "**00:40:bc:*xx:yy:zz***"; "**00:40:bc**" is Algorithmics' allocated Ethernet block, and "***xx:yy:zz***" should be written on a label found on or near the ethernet controller logic. Ethernet addresses are allocated based on the board model and serial number, and in desperation can be confirmed from Algorithmics.

- *netaddr* holds the board's Internet address. This address will be allocated to you by your network manager. It should be in the standard dot-notation form, e.g. "***a.b.c.d***", where "***a***", "***b***", etc. are decimal numbers between 1 and 254, inclusive.

Most networks will use class C addressing, which means that the first three fields of the Internet address identify the network, and leaves only the last field for allocation to individual host IDs (i.e. a maximum of 254 hosts IDs). By default PMON works out what class of network addressing to use based on the value of *netaddr*, but occasionally you will come across a network setup which breaks the standard numbering rules, and you will need to force PMON to allow two or three fields for the host IDs. In this case you also have to set the *netmask* and *broadcast* variables. If these are required, then they should be specified in the standard dot-notation, as used for *netaddr* above.

The following would be valid on Algorithmics' network (though the setting of *netmask* and *broadcast* is redundant):

```
PMON> set ethaddr 00:40:bc:00:21:08
PMON> set netaddr 193.117.190.224
PMON> set netmask 255.255.255.0
PMON> set broadcast 193.117.190.255
```

Using a Name Server

If your local network is equipped with a ("DNS") name server, then PMON can use it. This allows you to enter the names of network systems, rather than the network number. To configure the board for DNS, follow these steps:

1) Set *nameserver* to the Internet (numeric) address of the host on which the name server runs.

2) The *hostname* variable should be set to the fully qualified hostname of this board, as allocated by your network manager. This means it must include the full domain name (e.g. "p4000.hwnet.xyzinc.com").

From now on, you can use symbolic names wherever a command requires an Internet address. Algorithmics' TFTP server (for loading software) is called "temple", so we can say:

```
PMON> boot temple:/usr/pmon/test
```

If you do not have a name server, then you must specify Internet addresses numerically (e.g. "193.117.190.222"). But you can always store commonly used addresses in environment variables, e.g.:

```
PMON> set temple 193.117.190.220
PMON> boot $temple:/usr/pmon/test
```

Selecting the Default Gateway

In a large organisation, your target board and your host development system may be on different *subnets*. In such cases there should an internet *gateway* which forwards packets between the subnets. To allow communication with other subnets, set the *gateway* variable to the internet name or address of the relevant gateway on the board's local subnet. PMON does not keep proper routing tables, but at least it will now send to that gateway any packets whose internet address does not match its own subnet.

Testing the Connection

At each stage of configuring the connection use the ping command, described on page 54, to test your current setup. Attempt to ping machines on your local subnet (using their numeric address); ping the name server; ping the gateway, and finally the remote machine(s) from which you intend to load your software.

5.2. Configuring the TFTP Server

PMON downloads and uploads by acting as a client in the TFTP (Trivial File Transfer Protocol); see the boot, load and dump for details. TFTP requires only the simple, connection-less UDP "transport" protocol. You need to make sure that there is a TFTP server up and running on your chosen host system.

Note that most systems already have some other mechanism (such as NFS) for sharing files; so the host system from which you download may not actually be the host where the files physically reside. You don't have to worry about that.

UNIX Workstation

Setting up the TFTP server on a Unix workstation is a job for the local system manager. The use of TFTP does not require an account or password on the remote system. Due to the lack of authentication information, the server will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling TFTP service. The server should have the user ID with the lowest possible privilege.

The TFTP server, *tftpd*(8), is normally started by the *inetd*(8) program. In many implementations, access to files may be restricted by invoking *tftpd* with a list of directories by including pathnames as server program arguments in /etc/inetd.conf. In this case access is restricted to files whose names are prefixed by the one of the given directories.

Because *tftpd* can be a security problem, systems are often shipped with it disabled by default. You'll often find the appropriate line commented out in `/etc/inetd.conf`. Rely on your Unix Network Guide for full details of *tftpd*(8) and *inetd.conf*(4).

DOS/Windows PC

DOS and Windows PCs provide a number of TCP/IP networking solutions. There are a number of commercial products: SunSoft's PC-NFS, FTP Software's PC/TCP, and many others. Alternatively there are "shareware" products such as Peter Tattam's TCP/IP for DOS and Windows (the author's email address is `peter@psychnet.psychol.utas.edu.au`).

The essential requirement for use with PMON is, of course, that the product includes a TFTP server. In most cases (under DOS, if not Windows) the TFTP server will not run as background task. You'll probably have to start it manually every time that you want to download or upload over the network. For example, using PC/TCP (and assuming the PC's Internet address is 193.117.189.8) would require a sequence of commands something like this:

`DOS C:\SDE\EXAMPLES\EX4>` **`make ex4ram`**	*Build example program*
`DOS C:\SDE\EXAMPLES\EX4>` **`tftp serve`**	*Start the TFTP server*
`PMON>` **`boot 193.117.189.8:ex4ram`**	*Load the program onto the board*
`PMON>` **`g`**	*Start the program*
`q` *(on the PC)*	*Terminate the TFTP server*

5.3. Downloading Files via Ethernet

In summary, to download and run a program via Ethernet, perform the following steps:

1. Setup your board and your host's TFTP server, as described earlier in this chapter.

2a. Use the `boot` command to load your program. If your network has a DNS name server, then you can specify the host name symbolically:

```
PMON> boot myhost:/usr/local/sde/examples/ex4/ex4ram
```

2b. If your network does not have a DNS name server, then you must specify the host numerically. To save having to remember it, you can store it in the `bootaddr` environment variable:

```
PMON> set bootaddr 193.117.190.21
PMON> boot /usr/local/sde/examples/ex4/ex4ram
```

2b. If you are repeatedly loading the same program, then you can store its name in the `bootfile` environment variable:

```
PMON> set bootaddr 193.117.190.21
PMON> set bootfile /usr/local/sde/examples/ex4/ex4ram
PMON> boot
```

3. Run the downloaded program.

```
PMON> g
```

6. Monitor Command Summary

This section summarizes all the commands supported by the PROM Monitor. Table 6.1 summarizes all PROM Monitor commands available on the P−4000i. Table 6.1 uses the following conventions:

- The caret symbol ("ˆ") indicates that the control key should be held while pressing the other keys in the command. For example, "ˆP" means hold the control key down and press the "P" key.
- Optional arguments are enclosed in square brackets. The square brackets are not part of the command.
- Arguments take spaces as delimiters.

For more information, see the Alphabetic Command Listing in Section 6.

Command	*Function and Options*	*Description and Comments*
b [-rw] [*adr*] [-s *str*]...	Set breakpoint(s)	Lists breakpoints if specified with no options. Up to 32 software breakpoints and a hardware break point are supported.
	[-r]	Hardware breakpoint for data read only.
	[-w]	Hardware breakpoint for data write only.
	[-s *str*]	Executes the command string when the breakpoint is reached.
	[*adr*]	Address for breakpoint.
boot [-bensy] [[host:]file]	Network bootstrap	Loads an executable object file over Ethernet using TFTP
	[-b]	Suppress breakpoint deletion.
	[-e]	Do not clear exception handlers.
	[-n]	Do not load symbols.
	[-s]	Do not clear symbol table.
	[-y]	Only load the symbol table.
	[[host:]file]	The internet hostname and filename.
bt [-v] [*cnt*]	Stack backtrace	Displays a function call stack backtrace.
	[-v]	Include stack frame address and size.
	[*cnt*]	Number of lines to display.
c [*bptadr*]	Continue execution	Continues from current address after updating shadow registers.
	[*bptadr*]	A single temporary breakpoint.
call *adr* [*val*|-s *str*]...	Call a function	Continues like the `c` command but does not update shadow registers.
	adr	Function starting address.
	[*val*]	Value to pass to function.
	[-s *str*]	String to pass to function.
copy *from to siz*	Copy memory	Copies from base up when copying to lower address and vice versa.
	from	Start address for source.
	to	Start address for destination.
	siz	Number of bytes to copy.

Command	*Function and Options*	*Description and Comments*
d [-b|h|w|s] adr [*cnt*|-r*reg*]	Display	`datasz` sets default word size. `moresz` sets default screen length.
	[-b]	Display bytes.
	[-h]	Display 16-bit words.
	[-w]	Display 32-bit words.
	[-d]	Display 64-bit words.
	[-s]	Display as a null terminated string.
	adr	Base address for display.
	[*cnt*]	Number of lines to display.
	[-r*reg*]	Display as register *reg*.
date	Display date and time	
	[yymmddhhmm.ss]	Sets new date and time.
db [numb|*] ...	Delete breakpoint(s)	Lists all breakpoints if no option specified.
	[*numb*]	Breakpoint number(s) to delete.
	[*]	Delete all breakpoints.
debug [-svV] [-- *args*]	Enter remote debug mode	Displays in terse mode by default.
	[-s]	Do not set client stack pointer.
	[-v]	Report protocol errors.
	[-V]	Set verbose mode.
	[-- *args*]	Pass remaining *args* to client.
dump *adr siz* [*port*]	Dump memory to host	Memory is dumped to hostport.
	adr	Dump from base address adr.
	siz	Dump a total of *siz* bytes.
	[*port*]	Send to this device or file.
eset *name*	Edit variable	Displays the named variable and allows it to be edited (see `sh` command for editing details).
	name]	Select variable named *name*.
fill *from to* {*val*|-s *str*}...	Fill memory	Fills memory block with numeric or string value. Note that *from* must be lower than *to*.
	from	Fill from base address *from*.
	to	Fill to end address *to*.
	[*val*]	Fill with hexadecimal byte *val*.
	[-s *str*]	Fill with ASCII string *str*. Enclose multiple words in double quotes.
flush [-di]	Flush caches	Flushes both caches by default.
	[-d]	Flush data cache only.
	[-i]	Flush instruction cache only.
g [-s] [-b *bptadr*] [-e *adr*] [-- *args*]	Go (start execution)	Starts at EPC address and sets stack pointer to beginning of stack by default.
	[-e *adr*]	Start at address *adr*.
	[-b *bptadr*]	Set temporary breakpoint at address *bptadr*.
	[-s]	Do not set client stack pointer.
	[-- *args*]	Pass remaining *args* arguments to client.
h [*|*cmd*...]	Help	Lists all available commands by default.
	[*]	List all help.
	[*cmd*]	Help on command *cmd*.
hi [*cnt*]	History display	Display last 200 commands.
	[*cnt*]	Display last *cnt* commands.

Command	*Function and Options*	*Description and Comments*
l [-bct] [*adr* [*cnt*]]	List (disassemble)	Disassembles from EPC address and pipes output to more command by default.
	[-b]	List only branches.
	[-c]	List only calls.
	[-t]	List trace buffer.
	adr	Start disassembly from address *adr*.
	cnt	Disassemble *cnt* lines.
load [-abeist] [-c *cmdstr*] [-o *offset*] [-u *baud*] [*port*]	Load memory from host	Uses current baud rate by default.
	[-a]	Do not add offset to symbols.
	[-b]	Suppress breakpoint deletion.
	[-c *cmdstr*]	Send *cmdstr* to host.
	[-e]	Do not clear exception handlers.
	[-i]	Ignore checksum errors.
	[-o *offset*]	Load at offset *offset*.
	[-s]	Do not clear symbol table.
	[-t]	Load at top of memory.
	[-u *baud*]	Set baud rate for transfer.
	[*port*]	Read from this device or file.
ls [-ln] [*sym*|-[va] *adr*]	List symbols	Lists all symbols in descending address order without showing addresses by default.
	[-l]	Show long listing with addresses or offsets.
	[-n]	List in numeric order.
	[*sym*]	Show symbols matching *sym* pattern filter. Wildcards * and ? supported.
	[-v]	Compute symbol values.
	[-a]	Show next lowest address in symbolic form.
	[*adr*]	Show symbols from address *adr*.
m [*adr* [*hexval*|-s *str*]...]	Modify memory	Enters interactive mode by default, displaying current address and value.
	adr	Modify address *adr* without entering interactive mode.
	hexval	Set current address to *hexval* and move forward one byte.
	-s *str*	Copy string *str* to current address.
	<CR>	In interactive mode, move forward one byte with no other change.
	=	In interactive mode, read current address again.
	ˆ|-	In interactive mode, move back one byte.
	.	Exit interactive mode.
more	Paginate to screen	(Embedded command) scroll MORESZ lines. Setting MORESZ to zero disables automatic scroll pauses.
	/*str*	Search for string *str*.
	n	Repeat last search.
	<SPACE>	Show next page.
	<CR>	Show next line.
	ˆs|ˆq	Pause scrolling.
	q|ˆc	Quit.
mt [-c] [[*addr*] *size*]	Memory test	Tests all memory by default.
	[-c]	Tests continuously.
	[*addr*]	Use address *addr* as base address.
	[*size*]	Perform test on *size* bytes.
off	off command	Switch off power supply on soft-switchable boards. Not implemented otherwise.

Command	*Function and Options*	*Description and Comments*
ping [-nqv] [-l *preload*] [-s *size*] *host*	Test net connection	Bounce ethernet packets back and forth between the board and another host.
	[-l *preload*]	Send the first *preload* packets fast, then revert to normal.
	[-n]	Numeric – display addresses numerically.
	[-q]	Quiet – nothing is displayed except summary.
	[-s *size*]	Use packets of *size* bytes (default=56).
	[-v]	Verbose – display all ICMP messages, not just echo replies.
	host	Internet host address to send packets to.
r [*reg*|* [*val*|*field val*]]	Display or set register	Lists general-purpose registers by default.
	*	Display all except floating-point registers.
	f*	Display all floating-point registers.
	reg val	Set specified register *reg* to value *val*.
	reg field val	Set specified field *field* in register *reg* to value *val*.
reboot	simulate reset	Jump to the MIPS start location 0xBFC0.0000, which is likely to restart PMON. But it won't do a complete hardware reset.
search *from to* {*val*|-s *str*}...	Search memory	To search for a multiple-word string, enclose the string in double quotation marks.
	from	Start search from address *from*.
	to	Stop search at address *to*.
	val	Search for value *val*.
	-s *str*	Search for string *str*.
set [*name* [*value*]]	Display or set variable	Lists all current variables by default. Entering a variable by itself displays the variable value.
	[*name*]	Select variable named *name*.
	[*value*]	Set variable to value *value*.

Command	*Function and Options*	*Description and Comments*
sh	Command shell	(Embedded command) process command prompt using ASCII character input and special characters listed below. INBASE sets the default numeric base for the shell. Setting INALPHA to “hex” makes the Monitor process the input as a hexadecimal number if possible. The PROMPT variable defines the command prompt string, with the metacharacter “!” replaced by the current history number. When RPTCMD is set to “on”, the previous command is repeated when the user enters a blank line. When RPTCMD is set to “trace”, only trace commands are repeated.
	ˆC	Abort execution of current command.
	ˆS	Pause output stream.
	ˆQ	Restart output stream after pause.
	ˆP	Recall previous command.
	ˆN	Recall next command.
	ˆF	Move cursor right.
	ˆB	Move cursor left.
	ˆA	Move cursor far left.
	ˆE	Move cursor far right.
	ˆD	Delete character at cursor.
	ˆH	Delete character before cursor.
	ˆK	Delete whole line to right of cursor
	;	Treat input after semicolon as a new command.
	!!	Repeat last command.
	!*str*	Recall and execute the last command that commenced with string *str*.
	!*num*	Recall and execute command *num*.
	+ - / ( )	Execute algebraic operator.
	ˆ*addr*	Substitute contents of address for address *addr*.
	@*name*	Substitute contents of register for named *register*.
	&*name*	Substitute value of symbol for symbol *name*.
	$*name*	Substitute value of environment variable *name*.
	0x*num*	Treat *num* as hexadecimal number.
	0o*num*	Treat *num* as octal number.
stty [*device*] [-va] [*baud*]	Set terminal options	Displays the terminal type and baud rate by default.
[sane] [*term*]	[*device*]	Use either "tty0" or "tty1" (tty0 is default).
[ixany|-ixany] [ixoff|-ixoff]	[-v]	List possible baud rates and terminal types.
	[-a]	List all settings.
	[*baud*]	Set baud rate.
	[sane]	Set sane settings.
	[*term*]	Set terminal type.
	[ixany]	Allow any char to restart output.
	[-ixany]	Allow only START to restart output.
	[ixoff]	Enable tandem mode.
	[-ixoff]	Disable tandem mode.
sym *name value*	Define symbol	Defines symbol value. Note that you can display symbols with the `ls` command.
	name	Change value for symbol *name*.
	value	Change symbol to value *value*.

Command	*Function and Options*	*Description and Comments*
t [-vbci] [-m *adr val*] [-M *adr val*] [-r *reg val*] [-R *reg val*] [*cnt*]	Trace (single step)	Execute command addressed by EPC by default.
	[-v]	List each step (verbose).
	[-b]	Capture only branches.
	[-c]	Capture only calls (jal instructions).
	[-i]	Stop on invalid program counter.
	[-m *adr val*]	Stop when memory at address *adr* is equal to value *val*.
	[-M *adr val*]	Stop when memory address *adr* is not equal to value *val*.
	[-r *reg val*]	Stop when register *reg* is equal to value *val*.
	[-R *reg val*]	Stop when register *reg* is not equal to value *val*.
	[*cnt*]	Trace *cnt* instructions.
tlb [entry]	display TLB	Show the contents of the "TLB", the memory management unit which stores translations for MIPS program addresses in the "mapped" regions. The display format depends on the CPU type. With no entry number, it will display all entries (typically 32 to 64 of them).
to [-vbci] [-m *adr val*] [-M *adr val*] [-r *reg val*] [-R *reg val*] [*cnt*]	Trace (step over)	Identical to the `t` command, except individual procedures are treated as a single step.
tr	Transparent mode	Copies keyboard characters to hostport, and copies characters from hostport to screen. Variable **`trabort`** sets the termination character.
unset *name* ...	Delete variable(s)	Makes all matching variables disappear.
	name	Variable name to delete. Wildcards * and ? supported.

Table 6.1: PROM Monitor Command Summary

7. Alphabetic Command Listing

This section contains an alphabetic listing of the commands supported by PMON. Each command description starts at the top of a new page. The name of the described command is in bold type in the top left hand corner of the first page. Each command description contains the following three sections:

- **Command Summary –** A single-sentence summary of the command function accompanies each description at the top of the first page, next to the command name.
- **Format –** A format description follows the single-sentence summary, with a brief description of each parameter and argument supported by the command. At the end of the format description, default values for optional variables are described where applicable.
- **Functional Description –** A complete description of the command function follows the format, with examples where appropriate. Related commands are listed at the end of the functional description where applicable.

Some command descriptions contain a final section that describes a variable specific to the command.

b — The `b` command sets and displays breakpoints.

Format

The format for this command is:

b [-r|w] ***adr...***

b ***adr*** **-s** ***str***

b [*adr...*]

where:

-r	set a hardware breakpoint which will trigger on reads.
-w	set a hardware breakpoint which will trigger on writes.
-s *str*	executes the command string when the breakpoint is hit.
adr	specifies an address for the breakpoint. Up to 32 software breakpoints addresses can be set, and one hardware breakpoint (on R4000, R4200, and R4400 processors only).

Invoking the `b` command with no options causes the Monitor to print a list of the current breakpoints. If neither the **-r** nor **-w** option is specified, then a software breakpoint is set.

Functional Description

The `b` command sets a hardware or software breakpoint at the specified address or addresses. Multiple addresses may be specified. Specified addresses must be word-aligned. For software breakpoints, the specified addresses must be in RAM.

The Monitor automatically assigns a number to each breakpoint. The Monitor allocates the lowest available breakpoint number from 0 to 31 to any new breakpoint.

The Monitor reports a new breakpoint's number immediately after the breakpoint is declared (see the examples at the end of this subsection for illustration of this). The assigned numbers can be used in the `db` (Delete Breakpoint) command.

If neither the **-r** nor the **-w** options are specified, software breakpoints are set by default. The Monitor implements software breakpoints by replacing the instruction at the specified address with a *break* instruction. Execution is then halted when the break instruction is executed.

The brkcmd Variable

When a breakpoint is reached, the command list specified in the environment variable *`brkcmd`* is executed. The default setting for *`brkcmd`* is "`l @epc 1`"

The first two words, "`l @epc`", specify that a breakpoint will occur at the address in the EPC register. The final "`1`" specifies that the Monitor will list one line when the breakpoint is reached. See the `l` command on page 46.

You can change the breakpoint command variable with the `set` command. For example, you can include additional monitor commands in the *`brkcmd`* variable. You should separate additional commands on the command line with a semicolon. For example, entering the following command lists one line after reaching a breakpoint, and then displays all the register values.

```
PMON> set brkcmd "l @epc 1;r *"
```

By default, breakpoints are cleared when the `load` or `boot` commands are executed. See pages 29 and 47 for details on how to override automatic breakpoint clearing.

Some examples illustrating the use of the `b` command follow.

`PMON>` **`b a002000c`**	*Set a software breakpoint at 0xA002.000C.*
`Bpt 0 = a002000c`	
`PMON>` **`b`**	*Display all breakpoints.*
`Bpt 0 = a002000c`	
`PMON>` **`b -r a0020020`**	*Set a hardware breakpoint on data read.*
`Bpt 32 = a0020020`	
`PMON>` **`b`**	*Display all breakpoints.*
`Bpt 0 = a002000c`	
`Bpt 32 = a0020020`	

See also the `db`, `set`, `load` and `boot` commands for more information on breakpoints.

boot — The `boot` command loads binary object files over Ethernet.

Format

The format for this command is:

boot [-bensy] [*host*:[*path*]]

where:

-b suppresses deletion of all breakpoints before the download.

-e suppresses clearing of the exception handlers.

-n suppresses the loading of symbols from the file.

-s suppresses clearing of the symbol table before the download.

-y loads only the symbols from the file.

host is the internet host from which to read the file.

path is the file name to be loaded from the host.

Invoking the `boot` command with no parameters or arguments clears the symbol table, deletes all current breakpoints, and attempts to load the program found in the host and file specified by the *bootaddr* and *bootfile* environment variables.

Functional Description

The `boot` command uses the TFTP (Trivial File Transfer Protocol) to load an executable binary file from a remote host over Ethernet. It can read files in ELF format (as used in Algorithmics' SDE−MIPS, newer SGI compilers, and systems compliant with the MIPS/ABI standard), and also the older MIPS ECOFF format. PMON extracts any symbol table information from these files, and adds it to the target symbol table.

The `boot` command normally clears the symbol table, exception handlers, and all breakpoints. The **-s** and **-b** options suppress the clearing of the symbol table and breakpoints, respectively. The value of the EPC register is set automatically to the entry point of the program. Therefore, to execute the downloaded program, only the `g` command is required.

The `boot` command may return a large number of different error messages, relating to network problems or file access permissions on the remote host. For a file to be loaded via TFTP it must be publicly readable, and it may have to be in a directory which is acceptable to the remote server. See page 18 for more information about setting up and using TFTP.

When reading the symbol table PMON may complain that it does not have enough room to store the program's symbols. To increase the size of the heap, use the `set heaptop` command to reserve more space and, if necessary, relink your program with a higher base address. The `boot` command will also detect cases where the program being loaded would overwrite PMON's crucial data or heap: again relinking your program at a different address will cure the problem.

Whilst it is loading each section of the file, `boot` displays the memory address (in hex) and size (in decimal) of that section. Typically these sections will be in the order *.text, .data* and *.bss*.

bt

The `bt` command displays a function call backtrace.

Format

The format for this command is:

bt [−v] [*cnt*]

where:

-v specifies that each function's stackframe base address and size should be displayed.

cnt specifies the number of lines to be displayed.

When invoking this command with no options, the backtrace displays the names and up to four arguments for each level of stackframe.

Functional Description

The `bt` command displays a list of function calls, starting with the function in which the *EPC* register currently lies, and finishing when a return address becomes "invalid". An address is deemed invalid if it does not lie within one of the ranges specified by the *`validpc`* environment variable.

Each line of output gives the current position in a function, and up to four of its arguments. The arguments can only be retrieved if they are saved within the function prologue, and this is unlikely to be the case for assembler functions and optimised C code. If you want to be able to see the arguments to C functions, then compile your program with optimisation disabled.

If the **-v** option is given, then the command additionally displays the stackframe base address and size for each function. It will also indicate the amount of dynamic stack space allocated using C's `alloca` function, or equivalent.

The output of this command is passed to the `more` command, letting the user view one screenful of output at a time. Optionally, the user can specify ***cnt***, which limits the number of lines to that number. An example illustrating the use of the `bt` command follows.

```
PMON> c write+10
     write+0x0010 3c09a07f lui      t1,0xa07f
PMON> bt
     write+0x0010 (0x00000001,0xa0030300,0x0000001c)
    flsbuf+0x0234 (0xa0030300,0xa0029030)
    printf+0x045c (0xa0025490,0xa0020000,0x000000001,0x00000010)
      main+0x0138 (0x00000001,0xa07ffffe0)
    _start+0x0040 ()
```

See also the `more` command on page 52.

c

The `c` command makes program execution continue after a breakpoint has stopped program execution.

Format

The format for this command is:

`c [`*`bptadr`*`]`

where:

bptadr specifies a single breakpoint. The breakpoint is removed when execution halts at this specified address.

Invoking the `c` command with no arguments causes the program execution to continue from the address specified in the *EPC* register.

Functional Description

When the user enters the `c` command, program execution starts at the address pointed to by the *EPC* register's current value. Use the `g` command to start program execution from an address specified on the command line.

As an option, a single temporary breakpoint may be specified. The temporary breakpoint is removed when execution halts. The temporary breakpoint is removed if another breakpoint stops program execution first.

An example of the `c` command follows.

`PMON>` **`c a0020104`** *Continue execution until 0xA002.0104.*

call	The `call` command executes a function.
Format	The format of the `call` command is:

call *adr* [*val*|-s *str*]

where:

adr	is the starting address of a function.
val	is the value to pass to the function.
-s *str*	is the string to pass to the function.

Functional Description

The `call` command executes the function whose address was specified as the first argument. Up to four optional arguments, if specified, are passed to the function in registers *a0* to *a3*.

The `call` command is similar to the `c` (continue) command, except the `call` command does not update the shadow registers with new values after the function is completed.

An example of the `call` function follows. In this example, the `call` command executes the function at 0x8002.0304, passing the single argument 0x8002.236C (converted into binary) to the function via register *a0*.

```
PMON> call 80020304 8002236c
```

copy — The `copy` command copies a specified number of bytes from one location in memory to another.

Format

The format of the `copy` command is:

`copy` ***`from to siz`***

where:

from declares the source address location.

to declares the target address location.

siz is the size of the block of memory to be moved. This quantity is specified in bytes.

If ***to*** is less than ***from***, then copying is performed in ascending order starting at ***from***. If ***from*** is less than ***to,*** then copying is performed in descending order starting at ***from+siz***.

Functional Description

The `copy` command replicates a specified number of bytes from one place in memory to another.

When moving a data block down, the source data is copied from the bottom of the block upwards: and when moving a data block up, the source data is copied from the top of the block downwards. By this technique, there is no risk of copying over data in overlapping block move operations; as the data in the overlapping area is copied first.

The following example shows how to copy a block of memory, 4 Kbytes in size, with a base address of 0x8002.0000, to another 4-Kbyte area starting at the address 0x8006.0000.

```
PMON> copy 80020000 80060000 4000
```

d — The `d` command displays memory contents in hex or ASCII format.

Format

The format for this command is:

d [-b|h|w|s] ***adr*** **[*cnt*|-r*reg*]**

where:

-b	displays the memory contents in groups of 8-bit bytes.
-h	displays the memory contents in 16-bit half-word groups.
-w	displays the memory contents in 32-bit word groups.
-d	displays the memory contents in 64-bit double-word groups.
-s	displays the memory contents as a null terminated string.
adr	specifies the base address from which data is displayed.
cnt	specifies the number of lines to be displayed.
-r*reg*	displays the contents of memory as register *reg*.

Functional Description

The `d` command displays memory, starting at the specified address, in hexadecimal or ASCII format. A **-b**, **-h**, **-w**, or **-s** option, if specified, sets how the data is displayed. See the examples at the end of this section for illustration of the possible display formats. The output of this command is passed to the `more` command, letting the user view one screenful of output at a time. Optionally, the user can specify ***cnt***, which limits the number of lines to that number.

The datasz Variable

If invoked without a **-b**, **-h**, **-w**, **-d** or **-s** option, the *`datasz`* variable sets the display format. Setting *`datasz`* to "`-b`", "`-h`", "`-w`" or "`-d`" has the same effect as the command line options of the same names described in this section. The *`datasz`* variable does not effect any other command displays.

The following example displays memory starting at 0x001.0000.

```
PMON> d a0010000
a0010000 bf c0 2b 00 bf c0 2b 00 bf c0 2b 00 bf c0 2b 3c ..+...+...+...+<
a0010010 bf c0 2b 3c bf c0 2b 3c bf c0 2b 20 bf c0 2b 20 ..+<..+<..+...+.
a0010020 bf c0 2b 20 bf c0 2b a8 bf c0 2b 78 bf c0 2b 60 ..+...+...+x..+`
a0010030 bf c0 2b 48 bf c0 2b a8 bf c0 2b a8 bf c0 2b a8 ..+H..+...+...+.
a0010040 bf c0 2b 78 bf c0 2b 60 bf c0 2b 48 bf c0 2e 78 ..+x..+`..+H...x
a0010050 bf c0 2f 08 bf c0 2e c4 bf c0 2e 80 bf c0 2f 90 ../.........../.
a0010060 bf c0 2f 90 bf c0 2f 90 bf c0 2e 78 bf c0 2e 78 ../.../....x...x
a0010070 bf c0 2e 78 00 00 00 00 00 00 00 00 00 00 00 00 ...x............
```

See also the `more` command on page 52.

date

The `date` command displays or sets the date and time.

Format

The format of the `date` command is:

`date [`*`yymmddHHMM.SS`*`]`

where:

yymmddHHMM.SS
is the new date and time.

Functional Description

The `date` command with no arguments displays the current date and time as stored in the board's battery-backed clock/calendar device. If an argument is given, then this sets the current date and time.

The optional argument is a string of pairs of digits, with the following meaning:

yy	year (modulo 100)
mm	month (January = 1)
dd	day of month
HH	hour (24 hour clock)
MM	minute
.SS	seconds

When setting the date and time, you only need to enter as much as needs changing, starting with the minutes, then hours, then day, etc. Any value which is omitted is unchanged, except for seconds, which will be set to zero if omitted.

Some examples of the `date` command follow.

```
PMON> date                          Display current time
Wed Feb 23 13:29:33 1994
PMON> date 32                       Change minutes
Wed Feb 23 13:32:00 1994
PMON> date 1405                     Change hours and minutes
Wed Feb 23 14:05:00 1994
PMON> date 9402241103               New date and time
Thu Feb 24 11:03:00 1994
```

db

The `db` command deletes the specified breakpoints.

Format

The format for this command is:

db [*numb*|*]

where:

numb	is the breakpoint number to be deleted.
*	deletes all breakpoints.

Entering `db` without any parameters lists all existing breakpoints. Entering an asterisk (“*”) instead of a breakpoint number deletes all the existing breakpoints.

Functional Description

The `db` command deletes one or more specified breakpoints.

Examples illustrating the use of the `db` command follow.

```
PMON> db 3                                  Delete breakpoint 3.
PMON> db 4 6                                Delete breakpoints 4 and 6.
PMON> db                                    Display all breakpoints.
Bpt 0 = a002000c
PMON> db *                                  Delete all breakpoints.
```

debug

The `debug` command initiates the Monitor's remote debug mode.

Format

The format for this command is:

debug [-svV] [-- *args*]

where:

-s	does not set client stack pointer.
-v	shows communication errors.
-V	sets the verbose option.
-- *args*	indicates that the remaining argument or arguments ***args*** are to be passed to the client program. If the first argument does not start with a "-", then the "--" separator can be omitted.

Functional Description

The `debug` command causes the Monitor to enter remote debugging mode. The **-V** option selects verbose mode. In verbose mode, each of the messages sent to and received from the remote debugger are displayed on the terminal screen. It is not possible to leave verbose mode without leaving remote debug mode and restarting it without the **-V** option. By default, the Monitor does not display any messages.

See the `g` command on page 43 for a detailed explanation of the optional ***args*** list.

Examples illustrating the use of the `debug` command with the `gdb` remote debugger follow.

PMON> **set hostport tty1**	*Specify protocol and port for download.*
PMON> **set dlproto EtxAck**	
PMON> **set dlecho off**	
PMON> **load**	*Prepare for download.*
% **edown -d /dev/ttyb test1.lsi**	*Start the download.*
% **gdb-sde test1**	*Invoke* `gdb`.
(gdb) **target dbgmon /dev/ttyb**	*Wait for connection from target.*
PMON> **debug**	*Start communication with* `gdb`.
(gdb) **break main**	*Optionally set breakpoint at main.*
(gdb) **cont**	*Prepare for execution.*

See also the `set` command for the setup of the environment variables.

dump — The `dump` command uploads data to the host.

Format

The format for this command is:

`dump [-B]` ***`adr siz`*** **`[`*****`port`*****`]`**

where:

-B	selects binary mode for network upload.
adr	is the base address of the data to be uploaded.
siz	is the number of bytes to be uploaded.
port	is the name of the device or remote filename to send the data to.

Functional Description

The `dump` command uploads S-records to the host. All uploaded S-records except the terminating S-record are S3-records. The terminating S-record is an S7-record.

By default, if ***port*** is not specified, then `dump` sends uploads to the device specified in the *`hostport`* variable.

On a networked board you can upload to a remote file using TFTP, by specifying ***port*** as "`host:filename`". When used in this way the **-B** option will cause `dump` to write a raw binary file, instead of S-records. Note that the TFTP protocol requires that the destination file must already exist, and be publicly writable. See page 18 for more information about setting up and using TFTP.

The uleof and ulcr Variables

After the dump is completed, the string specified in *`uleof`* will be transmitted. The default value for *`uleof`* is "`%`".

If the variable *`ulcr`* is set to "`cr`", then the lines will be terminated by a carriage return ('`\r`') character;

If *`ulcr`* is set to "`lf`", then the lines will be terminated by a linefeed ('`\n`') character;

If *`ulcr`* is set to "`crlf`", then the lines will be terminated by a carriage return and a linefeed character

The default value for *`ulcr`* is "`cr`".

The following example of the `dump` command uploads 128 bytes starting at 0x9FC0.0000 in S-record format to the serial port named in the *`hostport`* variable.

```
PMON> dump 9FC00000 80
S3159FC002403C09A07F3C08003C3529FF203508C62FB6
S3159FC00250AD2800003C09A07F3529FF102408002542
S3159FC00260AD2800003C020040408260004080680O8C
S3159FC002703C1D800127BD8B403C01A00003A1E82502
S3159FC002800FF005BC240400000FF005BC2404000138
S3159FC002903C0280003C03800124426AB024633C2018
S3159FC002A024420010AC40FFF00043082AAC40FFF444
S3159FC003308D28000025290004012A082A256B000460
S7030000FC
```

The following example uploads 256 bytes in binary format to a remote file over Ethernet.

```
myhost % touch /tmp/upload
myhost % chmod a+rw /tmp/upload
PMON> dump -B 9FC00000 100 myhost:/tmp/upload
```

eset — The `eset` command edits environment variables.

Format

The format for this command is:

eset ***name...***

where:

name is the name of the environment variable to edit.

Functional Description

The `eset` command is used to edit environment variable values. For each variable name given as an argument the `eset` command displays the variable name and its value. and then allows you to edit it using the same line-editing facilities available in the `sh` command, as described on page 63. When you press carriage-return, the new value is stored.

When using this command you should not place quotation marks around a multiple-word value; otherwise the quotation marks will be stored with the variable, which is probably not what you want.

See also the `set` and `unset` commands, on pages 61 and 72 respectively.

fill — The `fill` command writes a hexadecimal pattern or string to a block of memory.

Format

The format for this command is:

fill *from to* {*val*|-s *str*}...

where:

from is the base address for the fill operation.

to is the end address for the fill operation.

val is the hexadecimal value of the byte that is written to the area to be filled.

-s *str* specifies that the memory block should be filled with an ASCII string rather than a particular value. String ***str*** is the ASCII string to be written to the memory block during the fill operation if the **-s** parameter is specified.

Functional Description

The `fill` command fills an area of memory with a specified hexadecimal pattern or repeating string. The pattern can be a single byte or multiple bytes. For the `fill` command to work correctly, ***to*** must be greater than ***from***. If the **-s** option is specified, then the next parameter is interpreted as an ASCII string. Multiple-word strings may be specified by enclosing them in quotes.

For example, to clear an area of memory from 0xA002.0000 to 0xA002.1000, enter:

```
PMON> fill a0020000 a0021000 0
```

To fill an area of memory from 0xA002.0000 to 0xA002.1000 with the string of values 0x41, 0x42, 0x43, 0x44, and 0x45, enter:

```
PMON> fill a0020000 a00210000 41 42 43 44 45
```

To fill an area of memory from 0xA002.0000 to 0xA002.1000 with the ASCII string `"hello world"`, enter:

```
PMON> fill a0020000 a0021000 -s "hello world"
```

flush — The `flush` command flushes the data and/or instruction cache.

Format

The format for this command is:

`flush [-di]`

where:

-d flushes the data cache only.

-i flushes the instruction cache only.

Entering `flush` without any parameters flushes both caches.

Functional Description

The `flush` command flushes the data and/or instruction cache. On an R4x00 processor, flushing the instruction cache requires only that it be invalidated, whilst flushing the data cache performs both a write-back and invalidate.

g — The `g` command starts program execution.

Format

The format for this command is:

```
g [-s] [-b bptadr] [-e adr] [-- args...]
```

where:

-b *bptadr* is a breakpoint address where program execution is to be stopped. This breakpoint is removed the next time that execution halts.

-e *adr* is the address of the first instruction to be executed.

-s is a flag indicating that the stack pointer, *sp*, should not be set.

-- *args* indicates that the remaining argument or arguments ***args*** are to be passed to the client program. If the first argument does not start with a "-", then the "--" separator can be omitted.

By default, the `g` command starts program execution at the address in the *EPC* register, and sets the stack pointer, *sp*, to the top of the stack area.

Functional Description

The `g` command starts program execution. If the user does not specify the starting address with **-e**, then execution starts at the current value of the *EPC* register, otherwise it starts at ***adr***.

If the **-b** option is specified, then a temporary breakpoint is set at ***bptadr***. The temporary breakpoint remains in effect only until the next time that program execution is halted.

If the user specifies ***args***, then the Monitor passes them to the client program by the following method. It places the number of arguments (*argc*) in register *a0*. It also places the address of an array of pointers to the command-argument strings (*argv*) in register *a1*. The first array entry will point to the string "`g`" (this command). If you use start-up code which preserves registers *a0* and *a1*, then function *main* will receive *argc* and *argv* so that it can read options from the command line,

Examples illustrating the use of the `g` command follow.

`PMON> g`	*Start executing at the current value of the EPC register.*
`PMON> g -e a0020000`	*Start executing at 0xA002.0000.*
`PMON> g -b a0020008`	*Start executing at EPC break at 0xA002.0008.*
`PMON> g -e a0020000 -b a0020008`	*Start executing at 0xA002.0000 and break at 0xA002.0008.*

h

The `h` command provides on-line help.

Format

The format for this command is:

h [*|*cmd*...]

where:

*	provides detailed help on all the commands.
cmd	is a command. The Monitor then provides help on the stated command.

If the command is executed without any parameters, then the Monitor lists all the available commands.

Functional Description

The

command provides on-line help. If issued without arguments, all commands are listed. If issued with one or more command names as an option, it produces more detailed help on those commands.

The * option produces detailed help on all the commands, using the `more` command to control output on the screen.

Examples illustrating the use of the `h` command follow.

```
PMON> h
h       on-line help            hi      display command history
m       modify memory           r       display/set register
d       display memory          l       list (disassemble) memory
copy    copy memory             fill    fill memory
search  search memory           tr      transparent mode
g       go execute              c       continue execution
t       trace (single step)     to      trace (step over)
b       set breakpoint(s)       db      delete breakpoint(s)
load    load from hostport      dump    dump to hostport
set     display/set variable    stty    set terminal options
sym     define symbol           ls      list symbols
flush   flush cache             debug   enter remote debug mode
mt      memory test             call    call function
PMON> h stty
stty [tty] [-va] [baud] [sane] [term] set terminal options
```

hi

The `hi` command lists the command history.

Format

The format for this command is:

hi [*cnt*]

where:

cnt is the number of commands to list.

Entering the command with no parameters lists the last 200 executed command lines to the screen.

Functional Description

The `hi` command shows the command history, together with the history number for each command, in reverse order (the last command entered is listed first; the first command entered is listed last). The command numbers are reset to zero each time the system is reset.

Entering the `hi` command with no arguments lists the last 200 commands. This option is useful for determining the history number for a particular command.

The user can page through the output of the `hi` command, one screen at a time.

The optional ***cnt*** parameter selects a set number of lines to be output. The history list is intentionally in the reverse order to that used in a C shell, so that the latest entry is displayed first. If a command line is identical to the previous command, it is not added to the command history.

Examples illustrating the use of the `hi` command follow.

```
PMON> hi 3
14 hi 3
13 hi
12 l
```

Display the three last commands.

```
PMON> hi
13 hi
12 l
11 to
10 t
9  l
8  g start main
7  hi
6  g
5  ls -a @epc
4  d Pmon+200+0t13*4
more... q
```

Display the entire history, using `more` *to control the screen output.*

See also the `sh` command, which maintains a command history.

l

The `l` command disassembles instructions from memory.

Format

The format for this command is:

l [-b|c|t] [*adr* [*cnt*]]

where:

-b	lists only branches.
-c	lists only calls.
-t	lists the trace buffer.
adr	is the base address from which to disassemble instructions.
cnt	is the number of lines to disassemble.

When invoking this command with no options, disassembly starts at the address in the EPC register and is output to the `more` command.

Functional Description

The `l` command disassembles the memory contents, starting either at the EPC register's current value or at the specified address. The output of this command is passed to the `more` command, letting the user view one screenful of disassembled output at a time. Optionally, the user can specify a count value, which limits the number of disassembled lines to that number.

The regstyle Variable

The *regstyle* environment variable determines whether the Monitor displays hardware or software register names. Hardware register names are simply $0 through $31. Software registers are defined by the MIPS software conventions. Set *regstyle* to “hw” for hardware register names. Set *regstyle* to “sw” for software register names.

Examples illustrating the use of the `l` command follow.

```
PMON> set regstyle sw                              Select s/w names
PMON> l 9fc00240 4                                 Disassemble 4 instructions
Pmon+0x240 3c020040 lui     v0,0x40
Pmon+0x244 40826000 mtc0    v0,C0_SR
Pmon+0x248 3c048001 lui     a0,0x8001
Pmon+0x248 8c850080 lw      a1,128(a0)
PMON> set regstyle hw                              Select h/w names
PMON> l 9fc00240 4
Pmon+0x240 3c020040 lui     $2,0x40
Pmon+0x244 40826000 mtc0    $2,$12
Pmon+0x248 3c048001 lui     $4,0x8001
Pmon+0x248 8c850080 lw      $5,128($4)
```

See also the `more` command on page 52.

load The `load` command downloads programs and data from the host.

Format The format for this command is:

load [−abeist] [−c *cmdstr*] [−o *offset*] [−u *baud*] [*port*]

where:

-a suppresses addition of an offset to symbols.

-b suppresses deletion of all breakpoints before the download.

-c *cmdstr* sets a command string that the Monitor sends to the host to start a download operation. Note that ***cmdstr*** must be enclosed in quotation marks if it contains any spaces.

-e suppresses clearing of the exception handlers.

-i ignores checksum errors.

-o *offset* loads at the specified offset.

-s suppresses clearing of the symbol table before the download.

-t loads at the top of memory.

-u *baud* sets the baud rate for transfer.

port is the device or remote file to download from.

Invoking the load command with no parameters or arguments clears the symbol table, deletes all current breakpoints, allows the Monitor to receive programs or data from the host. via the device specified in the *`hostport`* variable.

Functional Description

The `load` command accepts programs and data from the host in LSI Logic's proprietary FastLoad format, Motorola S-record. The user can set environment variables to change the data port, the format, and the transfer protocol.

By default, if ***port*** is not specified, then `load` reads downloads from the device specified in the *`hostport`* variable.

On a networked board you can download from a remote S-record or FastLoad file using TFTP, by specifying ***port*** as "`host:filename`". See page 18 for more information about setting up and using TFTP.

The `load` command normally clears the symbol table, exception handlers, and all breakpoints. The **-s** and **-b** options suppress the clearing of the symbol table and breakpoints, respectively. The value of the *EPC* register is set automatically to the entry point of the program. Therefore, to execute the downloaded program, only the `g` command is required.

For RS232 download, the **-c** option permits a command string to be sent to the host when the load command is issued. This is intended for use in conjunction with the transparent mode. Note that if the command string contains multiple words, the command must be enclosed in double quotation marks, as shown in the example below.

The `load` command returns the error message "`out of memory`" if there is insufficient space in the heap for the program's global symbols. To increase the size of the heap, use the `set heaptop` command to reserve more space in the heap and, if necessary, relink your program with a higher start address.

The dlecho, dlproto, and hostport Variables

The *dlecho*, *dlproto* and *hostport* variables control operation of the download. Table 7.1 shows how these environment variables affect the operation of the `load` command.

See the section on downloading beginning on page 13 for more information on these variables and the use of the `load` command.

Table 7.1: Setting Variables for Download Operation

Variable	*Action*
dlecho off	Do not echo the lines
dlecho on	Echo the lines
dlecho lfeed	Echo only a linefeed for each line
dlproto none	Do not use a protocol
dlproto EtxAck	Send Xon and Xoff to control the host
dlproto XonXoff	Expect Etx as end of record, send Ack
hostport tty0	Select tty0 as the port to which the host is connected
hostport tty1	Select tty1 as the port to which the host is connected

Examples illustrating the use of the `load` command follow.

Two-Port Mode

```
PMON> set hostport tty1
PMON> set dlecho off
PMON> set dlproto EtxAck
PMON> load                                  Prepare for download.
% edown -d /dev/ttyb ex4ram.lsi             Start download on host.
Total = 0x00043C00 bytes
```

Single-Port Mode

```
PMON> set hostport tty0
PMON> set dlecho off
PMON> set dlproto XonXoff
PMON> load -c "cat ex4ram.lsi"              Send command "cat
Total = 0x00043C00 bytes                    ex4ram.lsi" to the host.
```

Network Mode

```
PMON> load myhost:ex4ram.lsi                Start network download
Total = 0x00043C00 bytes
```

See also the `set` command for the setup of the environment variables.

ls

The `ls` command lists the current symbols in the symbol table.

Format

The format for this command is:

ls [-ln] [*sym*|-v|-a *adr*]

where:

-l	provides a long listing, showing the address value for each symbol.
-n	lists the symbols in ascending order of address.
sym	is a pattern filter for the symbols to be shown. Both character wildcards ("`?`") and word wildcards ("`*`") are permitted.
-v	is the verbose option, showing the value in hexadecimal, decimal, and octal.
-a	shows the address in symbolic form.
adr	is the address for which a symbol or offset from a symbol is sought.

Invoking the `ls` command without any options or parameters lists the symbols in descending order of address without displaying the actual address for each symbol.

Functional Description

The `ls` command lists the symbols in the symbol table in alphabetical order.

The **-l** option produces a long listing, which includes the address value of each symbol. The **-n** option causes the symbols to be listed in ascending order of address. The **-a *adr*** option lists the symbol at the next lowest address. The **-v *adr*** option prints the result in hex, decimal, and octal. The **-v** option is useful for computing the value of an expression that may include registers, symbols, and absolute values.

Examples illustrating the use of the `ls` command follow.

```
PMON> ls                          List symbols in alphabetic order.
flush_cache start
PMON> ls -l                       List symbols in alphabetic order with
9fc016f0 flush_cache              addresses.
9fc00240 start
PMON> ls -ln                      List symbols and addresses in
9fc00240 start                    ascending order of address.
9fc016f0 flush_cache
PMON> ls s*                       List symbols starting with the letter "s".
start
PMON> ls -a 9fc00260              List symbol at the next lowest address.
9fc00240 start+0x20
PMON> ls -a @epc                  List symbol at the next lowest address
a0020020 = start+0x20             from EPC.
PMON> ls -v @t0+0t10*4            Display the value of the expression
0x800222e8 = 0t-2147343640 = 0o20000421350
```

m	The `m` command displays and modifies memory.

Format

The format for this command is:

m [*adr* [*hexval*|-s *str*]...]

where:

adr	is the memory address to display or modify without entering interactive mode.
hexval	is the value to insert at the specified address.
-s	is a flag signifying that the following parameter is a string value.
str	is a string value to copy to the specified address.
<CR>	enters interactive mode.
=	in interactive mode, reads current address again.
ˆ|-	in interactive mode, moves back one word.
.	exits interactive mode.

Entering no values with this command causes the command to operate in interactive mode.

Functional Description

This command can display and then modify memory locations interactively. This command can also set memory to a specified value directly.

If invoked with one or more values following the address, the command is executed immediately, without entering the interactive mode.

If the command is invoked without a value, the command enters the interactive memory mode. In interactive memory mode, the user enters a command at the cursor. The interactive memory mode first displays the address and its current value. Interactive memory mode then lets the user select one of the commands listed in Table 7.2.

Table 7.2: Interactive Memory Mode Commands

Command	*Action*
hex value	Set memory to hexadecimal value and then move forward one byte.
<CR>	Move forward one byte.
=	Stay at the same address and display the address again.
ˆ or -	Move backwards one byte.
.	Exit the `m` command.

If the **-s** option is specified, then the Monitor displays the memory contents as an ASCII string. A multiple-word string may be specified by enclosing the multiple-word string in quotation marks.

Examples illustrating the use of the `m` command follow.

`PMON>` **`m a0020000`**	*Display memory at address in interactive mode.*
`a0020000 00 _`	*User can enter command at cursor (_).*
`PMON>` **`m a0020000 1 2 3 4`**	*Set address 0xA002.0000 to 1, address 0xA002.0001 to 2, etc., in noninteractive mode.*

`PMON>` **`m a0020000`**	*Display memory at 0xA002.0000.*
`a0020000 01` **`<CR>`**	
`a0020001 02` **`<CR>`**	
`a0020002 03` **`<CR>`**	
`a0020003 04` **`.`**	
`PMON>` **`m a0020000 22`**	*Set address 0xA002.0000 to 0x22.*
`PMON>` **`m a0020000`**	*Display memory at 0xA002.0000.*
`a0020000 22` **`44`**	
`a0020001 00` **`<CR>`**	
`a0020002 00` **`55`**	
`a0020003 00` **`66`**	
`a0020004 00 ^`	
`a0020003 66` **`<CR>`**	
`a0020004 00` **`.`**	
`PMON>` **`m 80020000 -s even`**	*Set memory starting at 0x8002.0000 to the string "*`even`*".*
`PMON>` **`m 80030100 -s "PROM Monitor"`**	*Set memory starting at 0x8003.0100 to the string "*`PROM Monitor`*".*

more

The `more` command provides screen-at-a-time control for user input.

Format

The `more` command is an embedded command and is not accessible to the user on the command line.

Functional Description

The `more` command is not specified by the user on the command line, but is implicitly used by certain commands. After displaying the number of lines according to the value of the *moresz* environment variable, the `more` command displays the prompt "`more...`" Commands that use the `more` command include `h`, `d`, `l`, `search`, and `ls`.

The user can enter the following commands at the "`more...`" prompt:

Table 7.3: The more Commands

Command	*Action*
Space	Print one more page.
/str	Search forward for string str.
n	Repeat the last executed search.
<CR>	Show next line.
q	Quit from the more prompt and return to the Monitor prompt.

The moresz Variable

The *moresz* variable sets how many lines are displayed on one screen during screen-at-a-time output. If *moresz* is set to zero, then the screen scrolls continuously. The ˆS or ˆQ control sequence must be used to pause the output, and the ˆC control sequence must be used to terminate output.

For example, to set the default number of lines output by the `more` command to 12, enter:

```
PMON> set moresz 12
```

See also the `set` command for the setup of the environment variables.

mt	The `mt` command executes the memory test.

Format

The format for this command is:

mt [-c] [[*addr*] *size*]

where:

-c	implements a continuous test.
addr	is the base address from which to perform the memory test.
size	is the number of bytes, in hexadecimal, on which to execute the memory test.

Entering this command with no parameters tests all memory.

Functional Description

The `mt` command tests the available memory. By default, this command tests the memory at 0xA002.0000 to 0xA00F.FFFF.

If ***size*** is specified, then only that number of bytes are tested. If ***addr*** is also specified, then testing starts at the specified address.

Both ***addr*** and ***size*** are rounded down to the nearest word address. If the user specifies a ***size*** of zero, the test executes on the entire memory and does not terminate.

The `mt` memory test is not an exhaustive test. In the `mt` test, a single "walking one" is written to each word and cleared in turn. Then, to test other bits in the word, each word is loaded with its own address and then read back. Because this test writes an exclusive value to every word, it is sufficient to find most stuck-at faults and shorts. However, this test is not adequate to find pattern sensitivity and leakage faults.

Examples illustrating the use of the `mt` command follow.

PMON> **mt**	*Test from 0xA002.0000 to 0xA00F.FFFF.*
PMON> **mt 2000**	*Test 8 Kbytes starting at 0xA002.0000.*
PMON> **mt a0030000 4000**	*Test 16 Kbytes starting at 0xA003.0000.*

ping — The `ping` command "bounces" a packet to and from a specified network host.

Format

The format for this command is:

ping [-nqv] [-i *wait*]] [-s *size*] [-l *preload*] *host*

where:

-i *wait*	Wait ***wait*** seconds between sending each packet . The default is to wait for one second between each packet.
-l preload	If ***preload*** is specified, `ping` sends that many packets as fast as possible before falling into its normal mode of behavior.
-n	Numeric output only. No attempt will be made to lookup symbolic names for host addresses.
-q	Quiet output. Nothing is displayed except the summary lines at startup time and when finished.
-s *size*	Specifies the number of data bytes to be sent. The default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data.
-v	Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed. "Echo Replies" are displayed symbolically.

Functional Description

The `ping` command is used to verify ethernet network connections and setup. It makes use of a feature of the "ICMP" protocol, which is used by hosts and gateways for low-level administrative chores. Each ICMP host is required to respond to an ECHO_REQUEST datagram with an ECHO_RESPONSE. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a time and then an arbitrary number of "pad" bytes used to fill out the packet. The command continues pinging until interrupted by a Control-C.

When using `ping` for fault isolation, start by pinging "`127.0.0.1`" (a universal self-address, by internet convention.) This verifies that at least the onboard setup is workable. Then, hosts and gateways further and further away should be "pinged". Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the minimum/average/maximum round-trip time numbers. When the program is terminated by a Control-C a brief summary is displayed.

`Ping` will report duplicate and damaged packets. Duplicate packets "should never happen": they'd have to be gateway problems. Tell your network manager.

Damaged packets (data doesn't look like it should) are serious cause for alarm and often indicate broken hardware somewhere in the `ping` packet's path (in the network or in the hosts).

The “TTL” field of an IP packet is used to count the number of times the packet passes through a router; once it gets down to zero the packet is discarded, which prevents accidental internet loops from recycling the same old packets forever. It’s common practice for each router in the Internet to decrement the TTL field by exactly one. The biggest possible value of TTL is 255, and most Unix systems set the TTL field of ICMP ECHO_REQUEST packets to 255. This could, conceivably, mean that you can “ping” some hosts, but not reach them with `tftp`.

In normal operation ping prints the TTL value from the packet it receives. When a remote system receives a ping packet, it can do one of three things with the TTL field in its response:

- Not change it; this is what Berkeley Unix systems did before the *4.3 tahoe* release. In this case the TTL value in the received packet will be 255 minus the number of routers in the round-trip path.
- Set it to 255; this is what current Berkeley Unix systems do. In this case the TTL value in the received packet will be 255 minus the number of routers in the path *from* the remote system *to* the `ping` host.
- Set it to some other value. Some machines use the same value for ICMP packets that they use for TCP packets, for example either 30 or 60. Others may use completely wild values.

off

The `off` command switches off the power to the board, if your board and power supply have a soft switch. By convention this is the last command of a session.

Format

The format for this command is simply:

`off`

r	The `r` command sets or displays register values.

Format

The format for this command is:

r [*reg*|* [*val*|*field val*]]

where:

reg	is the name of the register or registers (specified by wildcard characters) to display or modify.
val	is the value to which the specified register or registers should be modified.
field val	is the value to which the specified field in the specified register should be modified.
*	displays the contents of all registers except floating-point registers.
f*	displays the contents of all floating-point registers.

Invoking the `r` command without any parameters or arguments displays a list of all the general-purpose registers.

Functional Description

The `r` command sets or displays register values. The character and word wildcards, "`*`" and "`?`", can be used in the register name. This command accepts both hardware and software names.

See also the `l` command for disassembling instructions from memory on page 46.

The regsize Variable

The *`regsize`* variable selects how many bits to display for the general-purpose registers (i.e. *$0* to *$31*), and some Coprocessor 0 registers (e.g. *EPC*). If *`regsize`* is set to "`32`" then 32-bits will be displayed, and if it is set to "`64`" then 64-bits will be displayed.

Note that *`regsize`* does not affect how the floating point registers are displayed. These will be displayed as 64-bit registers only if the *FR* bit is set in the CPU's *Status* register (called *sr* or *C0_SR*).

Examples illustrating the use of the `r` command follow.

PMON> **r**	*Display all general-purpose registers.*
PMON> **r ***	*Display all register values.*
PMON> **r 8**	*Display $8 (t0).*
PMON> **r t0**	*Display t0 ($8).*
PMON> **r t***	*Display t0 through t9.*
PMON> **r epc**	*Display EPC register.*
PMON> **r epc start**	*Set EPC register to the symbol start value.*
PMON> **r 4 45**	*Set register 4 to 45.*
PMON> **r t0 45**	*Set register t0 to 45.*
PMON> **r sr 0**	*Set SR to zero.*
PMON> **r sr bev 1**	*Set the BEV bit of SR to one.*

```
PMON> r epc a0020000     Set EPC to A002.0000.
```

The following illustration shows how the `r` command to display the register contents across the entire screen:

```
PMON> set regsize 32
PMON> r
       zero      at        v0        v1        a0        a1        a2        a3
 $0- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
        t0        t1        t2        t3        t4        t5        t6        t7
 $8- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
        s0        s1        s2        s3        s4        s5        s6        s7
$16- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
        t8        t9        k0        k1        gp        sp        s8       ra
$24- 00000000 00000000 00000000 00000000 00000000 80008b40 00000000 00000000

PMON> r *
 $0- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 $8- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$16- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
$24- 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
C0_EPC=a0020000 C0_BADADDR=00000000
   C0_SR:  CU BEV TS PE CM PZ SWC ISC  IM&SW   KUo IEo KUp IEp KUc IEc
          0000 0   0  0  0  0  0   0  00000000 0   0   0   0   0   0
C0_CAUSE: BD CE   IP   SW EXCODE
           0  0 000000 00  Int
 C0_PRID: IMP Rev
           0   0

PMON> r sr iec 1      Set IEC bit of SR to 1

PMON> r sr
C0_SR: CU BEV TS PE CM PZ SWC ISC IM&SW   KUo IEo KUp IEp KUc IEc
      0000 1   0  0  0  0  0   0 00000000 0   0   0   0   0   1

PMON> set regsize 64
PMON> r
           zero              at                v0                v1
 $0- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            a0               a1                a2                a3
 $4- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            t0               t1                t2                t3
 $8- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            t4               t5                t6                t7
$12- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            s0               s1                s2                s3
$16- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            s4               s5                s6                s7
$20- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            t8               t9                k0                k1
$24- 0000000000000000 0000000000000000 0000000000000000 0000000000000000
            gp               sp                s8               ra
$28- 0000000000000000 ffffffff80008b40 0000000000000000 0000000000000000
```

reboot

The `reboot` command attempts to restart the PMON monitor (and any other code which runs before PMON at bootstrap time) by jumping to 0xBFC0.0000 - the MIPS restart location. If your system initialisation depends on some device receiving a hardware reset, this may not work.

Format

The format for this command is just

`reboot`

search — The `search` command executes a search for a memory pattern.

Format

The format for this command is:

search *from to* {*val*|-s *str*}...

where:

from	is the start address for the search operation.
to	is the end address for the search operation.
val	is the hexadecimal value that is the object of the search.
-s *str*	specifies that the search operation is for a string ***str***.

Functional Description

The `search` command searches memory for a pattern. The pattern may be a single byte, multiple bytes, or an ASCII string.

If the **-s** option is specified, then the next parameter is interpreted as an ASCII string. To search for a multiple-word string, enclose the string in double quotation marks.

The output of this command is printed to the screen via the `more` command.

The following example searches for 3c and d4 from 0xA002.0000 to 0xA003.0000:

```
PMON> search a0020000 a0030000 3c d4
```

The following example searches or “ABC” from 0xA002.0000 to 0xA003.0000:

```
PMON> search a0020000 a0030000 -s "ABC"
```

See also the `more` command.

set	The `set` command sets and displays environment variables.

Format

The format for this command is:

`set [`*`name`*` [`*`value`*`]]`

where:

name	is the name of the environment variable to set.
value	is the string to which the environment variable is set.

Entering the `set` command with no arguments displays all the current environment variables.

Functional Description

The `set` command is used to set or display environment variable values, which are stored in NVRAM.

In some cases, when the Monitor displays a variable's current value, the Monitor prints a list of allowed values enclosed in square brackets; in other cases, no list is shown. In general, when the value is a numeric value, or when the value has an unlimited range of possible values, no list is shown.

Where a variable has a list of values, the `set` command will check that the specified value is in the list. Most other values will only be checked when a command uses a variable.

To set a variable to a multiple-word value, enclose the value in single or double quotation marks.

See also the `eset` and `unset` commands, on pages 40 and 72 respectively.

Examples illustrating the use of the `set` command follow.

```
PMON> set                              Display all current values.
brkcmd     "l @epc 1"
datasz     -b                    [-b -h -w]
dlecho     off                   [off on lfeed]
dlproto    EtxAck                [none XonXoff EtxAck]
ethaddr    00:40:bc:03:00:00
heaptop    80020000
hostport   tty1
inalpha    hex                   [hex symbol]
inbase     16                    [auto 8 10 16]
moresz     10
prompt     "PMON> "
regstyle   sw                    [hw sw]
rptcmd     trace                 [off on trace]
trabort    ^K
ulcr       cr                    [cr lf crlf]
uleof      %
validpc    "_ftext etext"
```

```
PMON> set moresz                                   Display current moresz.
moresz = 10
PMON> set moresz 20                                Set moresz to 20 decimal.

PMON> set brkcmd "l @epc 1;r cause;r"
                                          Set brkcmd to the string
                                          "l @epc 1;r cause;r"
```

sh

The `sh` command is an embedded command that executes the Monitor command typed following the prompt.

Format

The format for this command is:

sh

Functional Description

The following syntactic rules apply to all command lines entered at the Monitor prompt.

- Multiple commands can appear on one line if each command is separated by a semicolon (`;`).
- Register names are replaced by their contents if the register name is prefixed with an "at" symbol (`@`).
- Symbol names are replaced by their value if the symbol name is prefixed with an ampersand symbol (`&`).
- Environment variable names are replaced by their value if the symbol name is prefixed with a dollar symbol (`$`).
- Control-S pauses the output stream.
- Control-Q restarts the output stream.
- Control-C aborts the current command.

The shell also maintains a command history. Previous command lines are recalled either with Emacs-like commands or with C Shell "`!`" notation. Table 7.4 lists the commands that are supported by the Monitor.

Table 7.4: Command Shell Features

Command	*Action*
^P	Recall previous command.
^N	Recall next command.
^F	Move cursor one character to the right (forward).
^B	Move the cursor one character to the left (back).
^A	Move the cursor to the beginning of the line.
^E	Move the cursor to the end of the line.
^D	Delete character at cursor position.
^H	Delete character to the left of the cursor.
^K	Delete whole line of characters to right of cursor
!*str*	Recall and execute last command that started with string *str*.
!*num*	Recall and execute command *num*.
!!	Repeat last command.
+ – / ()	Execute algebraic operator.
^*addr*	Substitute contents of address for address *addr*.
@*name*	Substitute contents of named register.
&*name*	Substitute value of symbol for symbol *name*.
$*name*	Substitute value of named environment variable.
0x*num*	Treat *num* as a hexadecimal number.
0o*num*	Treat *num* as an octal number.

The inbase, inalpha, prompt and rptcmd variables

The following paragraphs describe the *inbase*, *inalpha*, *prompt,* and *rptcmd* environment variables:

inbase – This variable selects the default input base for numeric values. A value of 8, 10, or 16 selects that base as the assumed default. If "auto" is specified, the base is determined according to the usual C language rules (0x = hex, leading 0 = octal, otherwise decimal).

If *inbase* is set to 8, 10, or 16, then values starting with zero through nine are assumed to be values in the specified base. If *inbase* is set to "auto", then values starting with zero are assumed to be octal, and numbers starting with one through nine are assumed to be decimal.

Table 7.5 lists the rules that hold in setting the default numeric base.

Table 7.5: Setting the Default Numeric Base

Inbase	*Base*
0x	Hexadecimal
0t	Decimal
0o	Octal
[g-zG-z$_.]	Symbol
&	Symbol
@	Register

inalpha – This variable selects whether arguments starting with letters 'a' though 'f' are interpreted as symbols or as hexadecimal numbers.

Setting *inalpha* to "hex" causes the Monitor interpret the argument as a hexadecimal value, if possible. If the argument cannot be interpreted as a hexadecimal value, then the Monitor checks the symbol table to see if the argument is a known symbol.

Setting *inalpha* to "symbol" causes the Monitor to check the symbol table first.

It is also possible to specify values using simple expressions using the arithmetic operators +, –, *, and /. Expressions do not take spaces between the numerals and operators. For example,

```
PMON> b printf+4
```

sets a breakpoint at (printf+4). Any combination of register names, symbols, and values may be used. The precedence order of operators is the same as that defined by the C language. Two examples showing the use of simple arithmetic operators follow:

```
PMON> ls –v start+0x240          Show the actual address.
PMON> d map+0t10*4               Dump memory at (map+(10*4)).
```

prompt – This variable specifies the command prompt string. The metacharacter '!' is replaced by the current history number. For example,

```
PMON> set prompt "!> "
23> _
```

It is not possible to display system variables in the prompt.

rptcmd – When this environment variable is set to "`on`", the previous command is repeated when the user enters a blank line. When set to "`trace`", only trace commands (`t` or `to`) are repeated.

See also the `hi` (command history) and `set` (setup and display environment variables) commands.

stty

The `stty` command displays and sets terminal options.

Format

The format for this command is:

```
stty    [device][-av] [baud] [sane] [term]
        [ixany|-ixany] [ixoff|-ixoff]
```

where:

device	is either tty0 or tty1. The default is tty0.
-a	gives a long listing showing all current settings.
-v	displays the possible choices for baud rate and terminal type.
baud	sets the baud rate.
sane	resets terminal settings to the default.
term	sets the terminal emulation type.
ixany	allows any character to restart the output.
-ixany	allows only START to restart the output.
ixoff	enables the tandem mode.
-ixoff	disables the tandem mode.

When invoking the `stty` command with no parameters, the Monitor displays the terminal type and baud rate for the tty0 port.

Functional Description

The `stty` command displays and sets the terminal options, such as terminal emulation type, baud rate, and *ioctl* settings. First, to display the current terminal type, baud rate, and *ioctl* settings for tty0, enter:

```
PMON> stty -a
```

To display the same information for tty1, enter:

```
PMON> stty tty1 -a
```

To change the baud rate or terminal type for tty0, simply enter the new setting after `stty`. Precede the new setting with "`tty1`" to change the settings for tty1.

Examples illustrating the use of this command follow.

```
PMON> stty                    Display terminal type and baud rate for
term=tvi920 baud=9600         tty0.
PMON> stty -a                 Display terminal type, baud rate, and
term=tvi920 baud=9600         ioctl settings for tty0.
canon echo echoe onlcr icrnl istrip ixon
erase=^H stop=^S start=^Q eol=^J eol2=^C vintr=^C
PMON> stty 9600               Set baud rate for tty0 to 9600.
PMON> stty -v                 List available baud rates.
Baud rates:
         50   75  110  134   200   150 300 600 1200
```

```
        1800 2400 4800 9600 19200 38400
```

PMON> **stty tvi920**	*Set terminal type for tty0 to tvi920.*
PMON> **stty tvi920 9600**	*Set terminal type and baud rate for tty0 to tvi920 and 9600 baud.*
PMON> **stty tty1 sane**	*Reset* `ioctl` *settings for tty1.*
PMON> **stty tty1 19200**	*Set tty1 to 19200 baud.*

sym

The `sym` command sets a symbolic name for a variable.

Format

The format for this command is:

***sym** name value*

where:

name	is the name of the variable for which a value is to be set.
value	is the value to which the variable is set.

Functional Description

The `sym` command sets a symbolic name to the specified value. Normally the `load` and `boot` commands clears the symbol table. However, there is an option to override the clearing of the symbol table (see pages 29 and 47 for details).

Symbols can be displayed using the `ls` command.

Examples illustrating the use of this command follow.

```
PMON> sym start 9fc00240
PMON> sym flush_cache 9fc016f0
PMON> l start 4
start:
start+0x240 3c09a07f    lui     t1,0xa07f
start+0x244 3c08003c    lui     t0,0x3c
start+0x248 3529ff20    ori     t1,t1,0xff20
PMON> l 9fc0027c 5
start+0x27c 03a1e825    or      sp,sp,at
start+0x280 0ff005bc    jal     flush_cache
start+0x284 24040000    addiu   a0,zero,0x0
start+0x288 0ff005bc    jal     flush_cache
start+0x28c 24040001    addiu   a0,zero,0x1
```

See also the `ls`, `boot`, `load`, `l`, and `sh` commands.

t — The `t` command initiates a trace procedure.

Format

The format for this command is:

`t [-vbci] [-m` ***`adr val`*** **`] [-M`** ***`adr val`*** **`] [-r`** ***`reg val`*** **`]`**
`[-R` ***`reg val`*** **`] [`** ***`cnt`*** **`]`**

or:

`to [-vbci] [-m` ***`adr val`*** **`] [-M`** ***`adr val`*** **`] [-r`** ***`reg val`*** **`]`**
`[-R` ***`reg val`*** **`] [`** ***`cnt`*** **`]`**

where:

-v	lists each step (verbose).
-b	captures only branches.
-c	captures only calls (jal instruction).
-i	stops on invalid program counter.
-m *adr val*	stops when memory at address ***adr*** is equal to value ***val***.
-M *adr val*	stops when memory at address ***adr*** is not equal to value ***val***.
-r *reg val*	stops when register ***reg*** is equal to value ***val***.
-R *reg val*	stops when register ***reg*** is not equal to value ***val***.
cnt	traces ***cnt*** instructions.

Functional Description

The `t` command executes the instruction addressed by the current value of the *EPC* register. The `to` command is similar to the `t` command, except that the `to` command treats an entire procedure as a single step. For example, if the current instruction at *EPC* is a jump and link instruction, *jal*, the next stop is at *EPC*+8.

The command or commands that are executed on completion of the single step is determined by the value of the environment variable *`brkcmd`*.

An example illustrating the use of this command follows.

```
PMON> t
Pmon+0x240 3c09a07f lui t1,0xa07f
```

tlb The `tlb` command is used for displaying the contents of the MIPS CPU's memory management unit translation table, the TLB.

Format

The format for this command is

`tlb [`*`entryno`*`]`

where ***entryno*** is a number between 0 and the highest entry in your CPU's TLB (31, 47 or 63 on different CPU types). If you just invoke `tlb` with no parameters, you'll get a list of all entries in the TLB.

Functional Description

The `tlb` command shows TLB entries; either one you select, or the whole lot.

Examples illustrating the use of the `tlb` command follow.

`PMON>` **`tlb`**	*Display the whole TLB*
`PMON>` **`tlb 5`**	*Display the TLB entry whose index is 5*

Here's some example output from the `tlb` command:

```
PMON> tlb
  0: vpn=0xa005e000 asid=0x0 sz=4K 0x00000000 vdgc 0x00000000 vdgc
  1: vpn=0xa005c000 asid=0x0 sz=4K 0x00000000 vdgc 0x00000000 vdgc
  2: vpn=0xa005a000 asid=0x0 sz=4K 0x00000000 vdgc 0x00000000 vdgc
  3: vpn=0xa0058000 asid=0x0 sz=4K 0x00000000 vdgc 0x00000000 vdgc
  4: vpn=0xa0056000 asid=0x0 sz=4K 0x00000000 vdgc 0x00000000 vdgc
  ...
```

This is an R4000 or derivative CPU, since each entry has one virtual address (vpn/asid) and two output addresses. 'vpn" and "asid" are fields read through the *EntryHi* register; the "sz" is the page size, read through the *PageSize* register; and the output fields come from the two registers *EntryLo0* and *EntryLo1*.

The field shown as "vdgc" represents the flags stored with each possible translation, and the possible letters are these:

- `v/V` valid flag: upper-case for a valid mapping, lower-case otherwise.
- `d/D` writeable flag (called "dirty" for obscure reasons): lower-case for a read-only page.
- `g/G` global flag; upper-case is a translation which is "global" and matches an address irrespective of the setting of the ASID field in *EntryHi*.
- `c/U` cacheability field; "U" for uncacheable, and "c" for all varieties of cacheable.

tr	The `tr` command selects transparent mode.
Format	The format for this command is: **`tr`**
Functional Description	The `tr` command selects transparent mode. In transparent mode, the Monitor copies any characters typed on the keyboard to the hostport and then copies characters arriving at the hostport to the screen. The `tr` command lets the user run the Monitor on the same serial port that is used as a login line.
The trabort Variable	The environment variable *`trabort`* selects the character that terminates the transparent mode and returns the Monitor to the default command mode. See also the `set` command for the setup of the environment variables.

unset

The `unset` command removes environment variables.

Format

The format for this command is:

`unset` ***`name...`***

where:

name	is the name of an environment variable to remove. Both character wildcards "`?`" and word wildcards "`*`" are permitted.

Functional Description

The `unset` command removes environment variables that are no longer required. For each variable name given as an argument the `unset` command removes all variables which match that name.

If you attempt to remove an environment variables which has a pervasive standard use within PMON, then it is not removed, but is reset to its default value.

See also the `eset` and `set` commands, on pages 40 and 61 respectively.

Examples illustrating the use of the `unset` command follow.

`PMON>` **`unset bootfile`**	*Remove `bootfile` variable.*
`PMON>` **`unset datasz`**	*Reset `datasz` variable to default.*
`PMON>` **`unset it*`**	*Remove all variables starting with "`it`"*

8. Using PMON with SDE−MIPS

Most of the examples in the preceding sections of this manual assume that your are using the Algorithmics' SDE−MIPS cross-compiler and embedded system toolkit.

PMON makes it easy to load, debug and run programs developed using SDE−MIPS. Refer initially to the SDE−MIPS installation and programmers' guide for detailed information on the example programs, and then follow the steps described below to compile and run Example 1 (Hello World!). Note that these examples assume that you are using the Unix version of SDE−MIPS; if you are using DOS, then there will be minor differences, so check your SDE−MIPS programmer's guide for the equivalent commands.

Note that the exception handling facilities in the SDE−MIPS Release 1.4 toolkit will completely take over from the PMON exception handler, and this will disable PMON's debug facilities. If your program uses any of the SDE−MIPS exception, interrupt or floating-point trap handlers, then you have no choice but to use SDE−MIPS's own remote debug mechanism, as described in the Release 1.4 documentation. More recent releases of SDE−MIPS interwork with PMON's exception handling, so that you can use PMON's low-level or remote debugging, as described here, on all programs.

Compiling Example

On the host

`% cd /usr/local/sde/examples/ex1`	*Change to example directory*
`% make clean`	*Remove old binaries.*
`% make SBD=P4000 ram`	*Build downloadable* `ex1ram` *for P−4000i.*

Remote Debugging with a Network

These instructions assume that you have already set up a TFTP server on your Unix workstation. See page 18 for more information about setting up and using TFTP for both Unix and DOS. They also assume that your host is a Unix workstation, which has a link for its serial port `/dev/tty`*x* (where '*x*' is a one or two character port identifier), to the P−4000i's `tty1` port. On DOS the serial port would be called `COM`*n*, where '*n*' is digit 0 - 3.

On the host

`% gdb-sde ex1ram`	*Start* `gdb` *debugger.*
`(gdb) target dbgmon /dev/ttyx`	*Await response from P−4000i*
`Remote MIPS DBGMON debugging using /dev/ttyx`	

On the P−4000i

```
PMON> boot myhost:/usr/local/sde/examples/ex1ram
PMON> debug
```

On the host

`0x80020020 in __start()`	`gdb` *displays the current function.*
`(gdb) b main`	*Set an initial breakpoint.*
`Breakpoint 1 at 0x80021669: file ex1.c, line 80`	
`(gdb) c`	*Continue to breakpoint.*
`Breakpoint 1, main() at ex1.c:80`	`gdp` *stops and displays*
`80    volatile int a = 5`	*the current line.*

```
(gdb) s                                   Single-step one source line.
```

Remote Debugging Without a Networked

On the P−4000i

```
PMON> set hostport tty1                   Setup communications parameters
PMON> set dlproto EtxAck                  (only needed once)
PMON> set dlecho off
PMON> load                                Wait for download
```

On the host

```
% edown -d /dev/ttyx ex1ram.lsi           Download FastFormat file
% gdb-sde ex1ram                          Start gdb debugger.
(gdb) target dbgmon /dev/ttyx             Await response from P−4000i
Remote MIPS DBGMON debugging using /dev/ttyx
```

Now continue as for the networked board.

PMON Machine-level Debugging

Instead of remote debugging you can if you need, or prefer, use the machine-level debugging facilities of PMON described in this manual. SDE−MIPS's binary to ASCII conversion program (`convert`) will optionally include the program's symbol table in the downloadable file, so that symbolic addresses can be used however it is loaded.

9. AlgPOST - selftest code in boot PROMs

9.1. About this Chapter

This document tells you about the AlgPOST program. Major sections are:

- *§9.2 (Introduction and Overview)* : everyone should read this to get a grip on what AlgPOST is and how to understand it.
- *§9 (About AlgPOST)* : relatively short, so read this if anything is confusing you.
- *§9.6 (AlgPOST diagnostics)* : reference guide to error codes and messages. Look here when something goes wrong.

9.2. AlgPOST introduction and overview

AlgPOST is a program which runs when a board is first powered on or reset; it runs some hardware tests before handing over control to a higher-level program - in this case, to PMON.

AlgPOST is designed to make progress, and to get some information through to you, in the face of serious hardware problems. There is no direct user interface to control the test sequencing; instead this is done by sharing PMON's environment variables, stored in non-volatile memory. AlgPOST behaviour is tailored by a set of variables; if a problem is detected with the environment store itself, it defaults to running comprehensive (but slow) tests, and being rather verbose about them.

AlgPOST communicates to you through the "diagnostic display". In most Algorithmics boards this is a 4-character alphanumeric display, but in other applications it may be a single-character "hex" display or even a single LED, passing messages using a sequential morse-like code. If the board has a serial port which can be used as a console, you can ask AlgPOST to send longer and more easily comprehended messages to a dumb terminal.

You will need this manual in several circumstances:

- *Troubleshooting* : where AlgPOST has behaved unexpectedly; instead of the familiar quiet progress towards bootstrapping, you have a diagnostic indication of some kind.

  You will probably do all or some of the following:

  - look up the error code which you see on the diagnostic display: these are listed in §9.6.5.
  - select a higher "log level" and then reset the board to re-run the tests, to get more information. You will need a dumb RS232 terminal (or any kind of terminal emulator running on a PC) attached to the main console port. To set the log level you'll need to know about environment variables and how to change them (see the "set" command on page 61 of this manual).

    Once the higher log level is selected the tests should send diagnostic messages to your terminal. These messages are listed in §9.6.6.

- *Investigation* : where AlgPOST runs normally, but something else appears to be going wrong. In this case you will want to increase the test level (see §9.5) to get some additional useful information and then force AlgPOST to run all of its tests (see §9.5).
- *Configuring AlgPOST for your system* : AlgPOST is user-configurable: information in §9.5. You have some control over:

  - what is tested: you can suppress tests which will interfere with your other hardware.
  - how thoroughly tests are carried out: trade off your patience against your confidence level.

  - how progress, status information and diagnostic messages are communicated. AlgPOST always uses the diagnostic display if one is fitted, but you can control how much comes out of which RS232 port(s).
- *Building PROM applications for a board*: §9.7 tells programmers how to co-operate with AlgPOST so that your PROM code can benefit from the standard power-on tests. PMON is built like this.

9.3. The test sequence

The strategy used is to start with minimal assumptions about the board which allow it to run some code. The test software then climbs a “ladder” of functionality, avoiding stepping on any rungs which have not been seen to be sound.

The test code is pessimistic about the status of the hardware it uses, trying to ensure that tests cannot be hung-up by malfunctioning devices. For example, serial port routines do not wait forever for characters to be transmitted.

Where subsystems are confined to a single chip the test software will usually assume that they are either faulty or fully functional, with no further attempt made to narrow down the failure.

In many cases subsystems much larger than a single chip operate in such a way that it is impossible (or not cost-effective within a reasonable time budget) to attribute blame for a fault below the subsystem level.

What faults will AlgPOST miss?

Diagnostic software authors need humility, because diagnostics often fail when they are needed:

- the hardware is too broken to run the diagnostic software (at this stage, of course, you at least know the board is faulty).
- the fault is intermittent in nature; AlgPOST makes very few attempts to repeat tests.
- the fault is in some sense pattern-dependent. AlgPOST tries to vary data and address patterns, but thoroughness takes too long.

Then there are the tests AlgPOST does not even attempt.

What AlgPOST does not test

There are several reasons for leaving tests out of a simple power-on suite:

- it is pointless to test subsystems whose failure would prevent AlgPOST from starting up or reporting anything. The CPU, PROM, diagnostic display and IO bus subsystems must work before AlgPOST can get anywhere.
- some tests would require an external test harness: eg ethernet connector, expansion bus connector.
- some tests risk interference with the customer’s other equipment: eg external ethernet test, use of serial ports.
- some tests take too long to carry out.

In some cases, AlgPOST is configurable so that users can make their own trade-offs, or can increase the level of testing when a problem is suspected: see §9.5 below.

Using hardware testability features

The P−4000i, like most Algorithmics products, has some simple functions intended to assist diagnostic software:

- The control/status register allows certain subsystems to be independently reset under CPU control. Initial tests are carried out with unneeded subsystems held in reset.
- The memory system is parity protected.
- The onboard bus master devices have the ability to perform “loopback” tests allowing their bus interfaces to be checked.

9.4. How AlgPOST communicates with you

Test results appear on a number of different channels:

Diagnostic Display

The P−4000i is fitted with a software-controlled LED display that can show four alphanumeric characters. AlgPOST uses this display to show progress and to note problems.

The display is blanked from reset, and unblanked very early in the bootstrap sequence. A completely blank display probably denotes a very dead system (though, of course, it could be the display that has failed).

During normal running, when all tests are passing, the display will will intermittently flash the abbreviated name of the test which is currently running (listed in Table 9.5), in lower-case. If the display shows one of the test names in UPPER-CASE, then at least that one test has detected a problem.

The alphanumeric display can show up to four characters at a time. When a message comprises several parts, each part is shown for 500ms with a 100ms gap separating the components. At the end of a message there is a 250ms blank period.

This is much easier to see than it sounds!

The console or consoles

The P−4000i supports 2 RS232 ports. These are both accessible via PC-style 9-pin D connectors on the board.

AlgPOST can print messages to a dumb terminal or equivalent (“console”) attached to either or both ports. The port(s) used for the messages and AlgPOST’s verbosity are configurable through environment variables.

The “standard” way to set up a board (where you have a serial port where messages will cause no serious harm) is to set the log level to send errors only (this is log level 3), and to only one port.

Simple flow control is provided while messages are printed. If a Control-S (XOFF) is typed to any active console all output is suspended until any other character is typed.

Environment variable

After the tests have finished, the first error message will be found in `itfailure`.

Test equipment triggers

In certain fatal error conditions, where the problems are so acute that it is likely that information will not reach the diagnostic display, the software first performs a set of writes to ROM locations whose addresses encode the information that will be displayed on the display. This can be traced by test equipment in laboratory conditions; see §9.6.3 for details.

Reporting strategy

As each stage of a test is started, the diagnostic display will be blanked for a brief period.

Several things will happen when an error is detected:

- A message describing the error may appear on each active console. Error messages and a possible explanation for their causes are given in §9.6.6.
- A mnemonic for the test which found a problem appears on the diagnostic display in upper-case. The test mnemonics are listed in Table 9.5.
- For the first error to occur, the error message is kept in the *`itfailure`* environment variable (see §9.5.)
- The diagnostic display continues to show the test name that failed to indicate that an error has occurred. It will stay that way until the end of the tests, when control is transferred to PMON.

9.5. Controlling tests - environment variables used by AlgPOST

The environment variables used by the test code are summarised in Table 9.1.

Variable	*Default Value*	*Description*
`itconsole`	`a`	Determines which console(s) are active
`itloglevel`	6	How verbose are console messages? Higher = more output.
`itpkg`	6	PROM package to execute when tests complete
`itquick`		Run minimal tests only if set
`itquiet`		Suppress all but most desperate error messages if set
`ittstlevel`	5	Which tests should be skipped? Higher = more thorough.
`itfailure`	–	Test failure message

Table 9.1: Boot test environment variables

- *`itconsole`* – a string representing the serial channels that will be used for console IO. Each character of the string will enable a group of consoles. The characters ’0’, and ’1’ enable serial channels 0 and 1 respectively. The character ’a’ enables all available consoles.
- *`itfailure`* – a brief description of the first error to be detected. Where all tests passed, this variable will not be defined.
- *`itloglevel`* – a number between 0 and 7 selecting the level of messages printed. With loglevel *n*, only messages with priority-number *n* or less are shown; so the higher the number, the more output will result. Each message is printed preceded by a tag identifying the message level.

Level	Priority	Tag	Description
0	**EMERG**	Emergency	system is unusable
1	**ALERT**	Alert	nothing else expected to work
2	**CRIT**	Critical	serious error
3	**ERR**	Error	some other error
4	**WARNING**	Warning	warning conditions
5	**NOTICE**	Notice	normal but unusual condition
6	**INFO**	Info	informational
7	**DEBUG**	Debug	debug-level messages

Table 9.2: itloglevel settings

- `itpkg` – A value between 0 and 7 that picks the PROM package to be executed after test completion. Normally "6", used for the main PMON monitor.
- `itquick` – Overrides `ittstlevel` variable; when set, the system behaves as if the test level was 1 and performs minimal tests.
- `itquiet` – Overrides the `itloglevel` variable; when set, the system behaves as if the log level was 1 and tells you only of "Emergency" and "Alert" messages.
- `ittstlevel` – A value between 0 and 7 selecting the level of tests to be executed. The higher the value, the more will be tested (and the longer it will take). Note that the external serial port loopback test run at level 7 requires a special loopback plug to be fitted.

Level	Test Description
0	minimal tests only (low memory, cache and NVRAM)
1	all tests, as quickly as possible
3	extended memory tests; stop test on first error
4	extended memory tests; continue on error
6	as 4, and include internal loopback test on serial channels
7	as 4, and include external loopback test on serial channels

Forcing AlgPOST to execute all of the tests

You can force AlgPOST to execute all of its configured tests by holding the reset/debug button in its debug position, as soon as AlgPOST has started. This procedure is useful if the environment variables have been set to skip most of the tests but the board is behaving in an erratic way.

When test execution is forced in this way, the test level is set to 7 and the log level is set to 6. See §9.5 for the effect of these levels.

9.6. AlgPOST diagnostics in detail

9.6.1. Test Sequence

The test sequence is summarised in Table 9.3:

Notes on the test sequence

- *From Reset*: The CPU starts execution at the ROM reset vector `0xbfc00000`. The reset code inspects some data structures in the PROM which define a number of *PROM packages* (independently built programs sharing the ROM space). It should then start the boot-test code, which will be marked as PROM package 7. See §9.7.1 for a description of the PROM structure and packaging convention. The boot-test code will completely reinitialise the

Mnemonic	*Test summary*
led	display "*U*U" then "U*U*" on diagnostic display
endian	check consistency of ROM and board endianness, halt on error
can access byte variables now...	
mem-conf	size memory and check memory configuration
mem-min	uncached write/read address test on PROM data area
in C from here on...	
prom	checksum PROM packages
nvram	check battery, checksum environment region, set defaults if wrong
can use environment variables from here on; the remaining tests can be skipped by setting the test level to 0...	
cache	sizing and operation
nvram-rtc	check clock for reasonable value
mem-best	fast address-based confidence check
mem-soak	sequence of "thorough" memory tests
mpsc-reg	register write/read tests on MPSC
mpsc-loop	internal and external loopback
mpsc-int	check connection of interrupt line
eth-reg	register access tests on SONIC
eth-read	get SONIC to read memory and check
eth-write	get SONIC to write memory and check
eth-int	check connection of interrupt line

Table 9.3: Test Sequence in brief

board; all devices will be held in reset.

It is usually possible to restart the system by a programmed jump to `0xbfc00000`.

The boot-test code does initialisation of the board for its own purposes. You should not expect it to leave any part of your initialisation unchanged; but neither should you rely upon AlgPOST to perform initialisation for you. It will reset devices on the board and reprogram the interrupt switches. On completion it will attempt to clear all the memory it can find to ensure that it contains good parity.

- *led*: enable alphanumeric display and flash it from "*U*U" to "U*U*". If the display shows something else then there may be a problem with the board's IO bus and the remainder of the tests will probably fail.
- *endian*: check PROM endianness makes sense (up to this point the code is "bisexual", achieved by avoiding all partial-word loads and stores.) If the configuration link and PROM are mismatched, flash/print an error message and halt.
- *mem-conf*: size memory and save the value for later.
- *mem-min*: perform minimal memory test. In the event of any problems, report and carry on (no good can be accomplished by stopping.)

  This test only covers uncached accesses to memory made while running uncached from PROM. It is restricted to that portion of the memory used by the PROM software.

  Up to this point all the code has run using registers only. The code will now attempt to run compiled test code using memory.
- *prom*: compute and compare a simple 32-bit add/carry checksum on the PROM packages, intended to detect PROM corruption and misprogramming. A calculated checksum of

`0xffffffff` is converted to `0x00000000`. If the stored package checksum is `0xffffffff` (ie it is uninitialised) the correct checksum to use can be printed so that the checksum can be installed in the PROM.

The PROM checksum routine is largely useless (far more data is read in running the routine than in formulating the checksum). If it does go wrong, it is more likely an incorrectly programmed ROM than a hardware error.

- *nvram* : the *NVRAM* (non-volatile store) has a battery check feature which is checked during the first NVRAM write. The NVRAM environment area is checked. If it is wrong the environment is reinitialised and default values for environment strings are set. The default settings will cause tests to be more verbose and more thorough.

  Up to this point, the test and log levels have been implicitly set to their maximum values so that any error messages are sent to all of the serial devices. Normally no output will have been generated because all of the tests will have passed. Now the real values for these variables are extracted from the *`itloglevel`*, *`itquiet`*, *`itquick`* and *`ittstlevel`* environment variables.
- *cache* : find the cache sizes.

  If the caches appear reliable then the remaining tests will use the caches where necessary. In particular they will be used when initialising memory and when running memory tests, because it is impracticable to perform these functions in a reasonable amount of time without running cached.
- *nvram-rtc* : check that the clock is running and start it if necessary. Check for a plausible value in the real time clock registers, resetting the time if necessary.
- *mem-best* : a “best-efforts” test; is necessarily relative to the amount of time considered reasonable for testing. On the P−4000i a one pass address-in-address test, running cached, takes about 0.75s/Mbyte of memory.
- *mem-soak* : optionally (dependent on the *`ittstlevel`* variable) run several more thorough memory tests. Byte-address-in-address, short-address-in-address and random-data-in-address tests are performed. These tests take some time.
- *mpsc-reg* : check out 72001 UART write/read register access.
- *mpsc-loop* : perform an internal and external loop back test on both channels. The internal loopback test has the unfortunate side effect of outputting data on the transmit lines. The external loopback relies on a loopback connector being installed. You have to set *`ittstlevel`* to a high value to get these tests run.
- *mpsc-int* : check that the interrupt line is connected through to the processor.
- *eth-reg* : write/read test on SONIC registers.
- *eth-read* : make the SONIC perform master read cycles by issuing a “load CAM” command.
- *eth-write* : check that the SONIC can perform master write cycles by putting the SONIC into internal loopback mode and making it transmit and receive two packets of data. This test does not rely on the presence of a transceiver or network connection.
- *eth-int* : check that the interrupt line is connected through to the processor.

9.6.2. What happens when everything works

While the tests are running, the diagnostic display will display the mnemonic code for the currently running test (from Table 9.5) in lower-case. It will flash with a soothingly irregular rhythm. When the tests finish, AlgPOST will blank the display before invoking the customer's selected ROM package.

9.6.3. Catastrophic errors and unexpected exceptions

If something is really wrong with the board, the CPU will usually get some kind of exception (illegal instruction, illegal or unmapped address). These conditions are regarded as fatal. They are usually a sign of something very seriously wrong, so great care is taken to ensure that *something* will get reported.

It is not usually possible to pinpoint the precise cause of an unexpected exception. The reports are designed to collect together information which will be useful to a support engineer, and reflect the contents of those CPU control and status registers whose value is relevant to each particular type of exception. Users who aspire to this level of understanding should refer to [Architecture]. Table 9.4 shows the exception codes and registers displayed.

Exception Type	*Message*	*Error Code*	*Registers Displayed*
TlbMiss	`bevt`	38	epc,cr,sr,vaddr,ra
XTlbMiss	`bevx`	39	epc,cr,sr,vaddr,ra
Cache	`bevc`	3a	errpc,cr,sr,cach
General	`bevg`	3b	epc,cr,sr,vaddr,ra
Reserved	`bevb`	3c	epc,cr,sr,ra

Table 9.4: Catastrophic exception codes

Great pains are taken to ensure that some indication of the error is made available to the user. Because an exception can occur at any time the code takes the following very conservative approach to these errors:

- The error code and the register contents are dumped in a way that will allow a logic analyser connected to the system to interpret the values. This is a failsafe mechanism in case none of the following higher level approaches work.

  The information is made available to the analyser by performing writes to specific locations in ROM space (in normal operation no writes are made to the PROM area). Only the address written to carries any information (data lines are harder to see on a logic analyser). In order to display a 32 bit value `WWXXYYZZ`, four writes to the following locations will be produced:

  ```
  0x1fc0WW00
  0x1fc0XX00
  0x1fc0YY00
  0x1fc0ZZ00
  ```

- The error message and register contents are sent to the diagnostic display. Before starting each message, the display is blanked for 250ms. When displaying register contents, the high 4 bytes of the register are shown for 500ms then the display is blanked for 100ms and then the low 4 bytes of the register is displayed. When a message is completed there is a further pause of 250ms.

  First the 4 character exception type is displayed. Then the register names and contents are displayed in order.

- All serial channels are initialised. A short message and the contents of the registers are displayed on each console. Care is taken that the code should not hang at this point due to a faulty console device.

- The error code and register contents are repeatedly shown on the diagnostic display in the manner described above.

9.6.4. Interpreting AlgPOST outputs

The first sign of trouble from AlgPOST is likely to be codes flashed on the diagnostic display display, followed by a steady message.

As described in §9.4, diagnostic display characters are usually separated by a short blanking period, with a longer blanking period used to group sequences of characters into "words" of 2 or 8 digits. You should be able to comfortably write the codes down as they are output.

Most errors are reported by a group reported just once; if you miss it you should reset the board again. But note the following special cases:

- *Display flashes FORC* : You have forced AlgPOST to execute all of the tests regardless of the current environment variable settings (see §9.5). If you haven't touched the debug button then there may be a problem in the board's interrupt circuits or on its IO bus.
- *Display shows constant message* : an error was detected. Attach a terminal and try again, to read any diagnostic messages produced by AlgPOST. You may want to use the monitor to select a higher log level.
- *Display repeatedly cycles through a complex pattern* : this is probably a report of an unexpected exception, as described in §9.6.3. The report consists of an exception type message followed by register names and their contents.

  Such errors are usually catastrophic, so don't expect anything else to work. However, more information is probably being shown via any of the serial ports. Call an expert.
- *Interpreting console messages* : Console messages are listed below in §9.6.6, but at best are reasonably self-explanatory. Once again, really peculiar messages (full of strange acronyms) may result from unexpected errors. These are best told to your support engineer.

9.6.5. Status messages on the display

The following abbreviated messages, in the form of mnemonics of 4 or less characters, are displayed on the alphanumeric diagnostic display. When tests are running normally, the lower-case messages will be displayed, to allow you to monitor progress. If a test fails, then an UPPER-case error error mnemonic will be displayed and retained until the tetst complete

Depending on the setting of *`itloglevel`* the longer message may also be displayed on the console, and in the case of errors, the message will also stored in the *`itfailure`* variable in NVRAM for later examination. At a log level of 3 or above, the messages become more verbose.

If the log level is at least 1, the message itself will be output to any enabled console.

Mnemonic	*Message*
`BEEB`	Wrong endianess configured
`DCCH`	Dcache non functional
`FPA`	FPA test failed
`ICCH`	Icache non functional
`MEMC`	Memory configuration failed
`MEMF`	Minimal memory test failed
`MEMQ`	Quick memory test failed

Mnemonic	*Message*
MEMX	Extended memory test failed
MFIL	Dcache refill from memory failed
MPSC	MPSC failure
NONV	NVRAM failure
PFIL	Dcache refill from PROM failed
PRTY	Memory parity circuit failure
PSUM	PROM checksum incorrect
RTC	Real time clock failure
SONI	SONIC failure
bevb	Reserved boot exception
bevc	Cache error boot exception
bevg	General boot exception
bevt	TLBmiss boot exception
bevx	XTLBmiss boot exception
cach	Cache tests in progress
dcac	Dcache address test in progress
dref	Dcache refill test in progress
forc	Full test sequence forced by debug button
icac	Icache address test in progress
memb	Memory byte address test in progress
memh	Memory halfword address test in progress
memp	Memory parity operation test in progress
memq	Quick memory address test in progress
memr	Memory word random test in progress
mpsc	MPSC test in progress
net	SONIC test in progress
novr	NVRAM test in progress
npkg	Package unavailable
psum	PROM checksum in progress
rtc	Real-time clock test in progress
????	Unknown error code

Table 9.5: Mnemonic Displays and Associated Messages

9.6.6. Error messages from the console

This section lists all the messages may be printed by the boot-test code during execution. Note:

- to pause and read a message when there is too much output, press Control–S (restarts on any other character).
- which messages are output depends on the value of the log level environment variable (`itloglevel` described in §9.5). The higher the log level, the more messages get printed.

The messages are grouped under the following headings:

- *Alert messages* : these have something so urgent to convey that they are printed even at log level 1. They usually signify a failure which is likely to disrupt the execution of the power-on tests themselves. When you see one of these you may get fuller information by setting the log level to 3 or higher.

  All alert messages correspond to error codes/mnemonics and are listed in Table 9.5 above.

- *Activity messages* : these inform of the start of each test, and are printed only at log level 6 ("INFO") and above. This will normally only be selected if there is a problem - tests which pass are usually silent.
- *General messages* : a job lot of detailed errors and warnings, together with interesting information about the system.

Activity messages

The following messages are output at log level 6 (information) to describe the tests about to be performed. As each message is output the LED is toggled to give a visual indication of progress.

cache tests
Determine instruction and data cache sizes. For CPU's where all cache in on-chip, this probably doesn't really try to test the cache.

real time clock operation
Checking that the real time clock is running and has a reasonable date.

quick memory address test
About to start quick address-in-address test on the main block of memory. Takes approximately 12s for 16Mbytes of memory.

memory byte address test
Starting a thorough byte oriented memory test. This test is only run at higher test levels.

memory halfword address test
Starting a thorough halfword oriented memory test. This test is only run at higher test levels.

memory word random test
Starting a thorough random memory address test storing random data in random locations. This test is only run at higher test levels. These extended memory tests may take several minutes to run on 16Mbytes of memory, during which time there will be no apparent activity.

MPSC operation
Check operation of the NEC serial chip. This includes checks on the device, its ability to generate interrupts and internal/external loopback tests at higher test levels.

SONIC operation
Check operation of the SONIC ethernet chip. This includes checks on the device, its ability to generate interrupts and internal loopback tests checking master read/write memory cycles.

General messages

The following messages (dependent on the log level selected) may be printed on the console. These messages describe board status and errors.

Message formats

Variable parts of the message are described here with a sort of "printf" format which will be familiar to software engineers. The format types used here are:

%s	Some string (the context will give you a clue)
%x	Number in hexadecimal format
%02x,%08x	Hex number of fixed length (2 or 8 digits)

%c	single character
%d	Decimal number
[x]	character “x” is optional

Message **LogLevel**

[*]Add=0x%x Wnt=0x%x Got=0x%x Xor=0x%x [*]Rrd=0x%x [*]Urd=0x%x **ERR**
Error message from a memory test. If the message is preceded by an asterisk then a cache parity error has occurred. *Add* is the address being tested. *Wnt* gives the data expected at the address. *Got* gives the actual data read from the location. *Xor* is the binary xor of the expected and got data; this value is useful for recognising systematic errors. After the error is detected the location under test is read again, *Rrd* gives the subsequent value read; if this field is preceded by an asterisk then the *Rrd* value is different from *Got* indicating a memory read (as opposed to write) problem. The *Urd* field gives the value obtained by reading the uncached location. Again this will be preceded by an asterisk if it is different from the original value obtained indicating a possible cache problem.

[*]Add=0x%x Wnt=0x%x Got=0x%x Xor=0x%x [*]Rrd=0x%x **ERR**
See the description of the previous message. This message is used when an error has been detected in an uncached memory location.

Activity: %s **INFO**
A message indicating the type of test about to be performed. See §9.6.6 for messages in this category.

Ancient real time clock value **ERR**
The boot-tests know the Unix time at which they were built. If the real time clock is running but contains an earlier value than this, the clock will be reset. This situation may be caused by a faulty clock chip or rogue software reprogramming the clock.

Data cache line size %d **INFO**
Informational message giving the data cache line size. The line size is set by the AlgPOST startup code.

Data cache size %d bytes **INFO**
Informational message indicating the size of the data cache.

Date: %s %d/%d/%d %02d:%02d:%02d **INFO**
Informational message showing the current time and date stored in the real time clock.

Executing PROM package %d **NOTICE**
Printed immediately before transferring control from the test code to a PROM package.

Failed to start real time clock oscillator **ERR**
An attempt to start the real time clock oscillator has failed. This is probably caused by a faulty or worn out NVRAM chip.

Instruction cache size %d bytes **INFO**
Informational message indicating the size of the instruction cache.

Integrated Tests **NOTICE**
Informational message printed immediately after the basic tests have been performed and before the mainline tests are started.

Integrated Tests Completed **NOTICE**
Informational message printed immediately after the tests have been completed. The next step is to enter the monitor or execute a PROM package.

MPSC: chan%d %s Wnt=0x%02x(%c) Got=0x%02x(%c) **ERR**
Serial channel internal/external loopback failure. An internal loopback error indicates a problem with the serial chip. An external loopback error may be caused by a faulty cable.

MPSC: chan%d %s receiver busy before loop **ERR**

MPSC: chan%d %s receiver busy in loop **ERR**
The serial chip has returned an unexpected status. This is probably due to a faulty chip or problems with the associated data bus.

MPSC: chan%d %s transmitted %d but received %d **ERR**
The internal or external loopback test has received an unexpected number of characters. Receiving 0 characters on an external loopback test means that the a loopback connector is not installed. Other errors indicate problems with the serial chip.

MPSC: chan%d %s transmitter busy after loop **ERR**

MPSC: chan%d %s transmitter busy in loop **ERR**

MPSC: chan%d %s: transmitter busy before loop **ERR**
The serial chip has returned an unexpected status. This is probably due to a faulty chip or problems with the associated data bus.

MPSC: chan%d %sback test skipped **NOTICE**
A loopback test has been skipped because the serial channel is currently configured as a console.

MPSC: chan%d register read/write failure Wnt=0x55 Got=0x%02x **ERR**
A register access test on the serial channel has failed. This may be a problem with the chip or a problem with the associated data bus.

MPSC: failed to generate TxEmpty interrupt **ERR**

MPSC: failed to generate interrupt **ERR**
The serial device has been programmed to generate an interrupt but did not do so. Probably something wrong with the serial device.

NVRAM battery failure detected **ERR**
The NVRAM has failed its battery check test. Replace the NVRAM.

NVRAM contents are corrupted **ERR**
The NVRAM environment is corrupted. Either the NVRAM is failing or rogue software has modified the NVRAM environment area.

PROM package %d has checksum 0x%08x expected checksum 0x%08x **ERR**
A PROM package has an incorrect checksum. This package will not be executed if selected. This may indicate a problem with the PROM or simply a package installation error.

PROM package %d not installed **WARNING**
The `itpkg` environment variable specifies a package that is not installed; it should be reset.

PROM package %d requires checksum of 0x%08x **WARNING**
The package does not have a checksum installed. This is not an error. The checksum indicated should be installed in the package to prevent this message appearing in future.

Real time clock contains invalid information **ERR**
One of the real time clock locations in the NVRAM contains invalid information. This may be a problem with the NVRAM or the associated data bus.

Real time clock may have lost battery backup **WARNING**
A warning printed if the NVRAM battery ok test failed because the real time clock is probably incorrect.

Reinitialising NVRAM environment **NOTICE**
Message printed immediately before reinitialising the NVRAM environment.

SONIC: CAM enable pointer Wnt=0x%x Got=0x%x **ERR**

SONIC: CAM entry %d address port %d Wnt=0x%04x Got=0x%04x **ERR**
The SONIC has misread data from memory into its CAM. This may be caused by a problem in the bus arbitration circuits.

SONIC: command register when in reset Wnt=0x%x Got=0x%x **ERR**
When the SONIC is in software reset the command register contains an unexpected value, probably due to a problem with the chip.

SONIC: failed to generate receive interrupt **ERR**

SONIC: failed to generate transmit interrupt **ERR**
The SONIC has not generated an expected interrupt. This is probably a problem with the SONIC chip.

SONIC: failed to load CAM **ERR**
The SONIC has not completed a “load CAM” command. This may indicate problems with SONIC bus master operations or with the chip itself.

SONIC: failed to read receiver resource area **ERR**

SONIC: has bad receive buffer sequence number %d expect %d **ERR**

SONIC: has bad receive data Wnt=0x%02x Got=0x%02x **ERR**

SONIC: has bad receive packet length 0x%04x **ERR**

SONIC: has bad receive packet sequence number %d expect %d **ERR**

SONIC: has bad receive packet status 0x%04x **ERR**

SONIC: has bad transmit packet status 0x%04x **ERR**
An error has been detected on the SONIC internal loopback test. This may be due to a faulty device or a problem in the bus arbitration circuits.

SONIC: loaded CAM but failed to interrupt **ERR**
The command executed by the SONIC has apparently completed but it has failed to generate an interrupt in the chip.

SONIC: silicon revision %d **NOTICE**
SONIC revision number displayed for information.

SONIC: watchdog timer 0 register failure Wnt=0x5555 Got=0x%x **ERR**

SONIC: watchdog timer 0 register failure Wnt=0xaaaa Got=0x%x **ERR**
A register access test on the SONIC has failed. This may be a problem with the chip or a problem with the associated data bus.

Setting real time clock to %s(%d) %d/%d/%d %02d:%02d:%02d **NOTICE**
The clock has been reset; either the date was unacceptable or the clock has been restarted.

Starting real time clock oscillator **NOTICE**
The real time clock oscillator has been stopped (to conserve the battery.) This message should only appear once. If this message is always displayed there may be a problem with the NVRAM or some other software may be stopping the clock.

System halting **NOTICE**
The tests have completed and no valid PROM package is selected. The system will enter a halt state from which only an exception will exit.

User request to enter monitor **NOTICE**
The debug button was held down and the monitor will be entered. If this happens when the debug button was not held down, then there is a problem with the debug button connections.

Version: %s **INFO**
Identifies the boot-test release.

9.6.7. Asking AlgPOST for more detail

Increase the log-level. This means getting to the monitor prompt and setting the *`itloglevel`* variable to a higher level:

```
PMON> set itloglevel 3
```

1 will report serious problems only; 3 will report on all test failures, with reasons; 6 will tell you about every test as it starts. See the description of the environment variables in §9.5 for more details.

9.7. Programming AlgPOST

9.7.1. PROM packages

The PROM consists of some startup code and a set of packages (self-contained portions of code). Each package consists of a package record stored in a fixed area of the PROM and a region of code located somewhere else in the PROM.

All unused package records and other unused PROM locations will be initialised to contain words of `0xffffffff`. Due to the nature of PROM devices, these locations can be reprogrammed allowing new packages to be incorporated into an existing PROM. Areas of the PROM available for use by new packages can be determined by examining existing package records.

A small “package loader” is incorporated into the startup code. The loader checksums the selected package and will start execution of the package if it is satisfied with the information. The package loader may be called by jumping to the fixed location `0xbfc70000` with the CPU *a0* register containing the package number to be executed. By default, after reset, the PROM will attempt to execute package 7 (the tests). The default package may be changed by reprogramming the PROM; however it is the responsibility of the initial package to do some basic board initialisation.

Having completed the tests, the next package to be executed is selected by the *itpkg* environment variable, which defaults to 6.

Each package record consists of 32 bytes (8 words). The code fragment in Table 9.6 shows the format of a package record.

```
package_record:
        .word   magic           # indicating package format
        .word   pkg_first       # address of first location used by package
        .word   pkg_last        # address of last location used by package
        .word   pkg_csum        # add-with-carry sum of *pkg_first..*pkg_last
        .word   pkg_entry       # package entry point
        .word   reserve1        # reserved
        .word   reserve2        # reserved
        .word   reserve3        # reserved
```

Table 9.6: Format of package record

9.7.2. PROM layout

The first `0x400` bytes of the PROM are commonly used for jump table entry points.

The R3000/R4000 boot exception vectors are reserved for AlgPOST. Packages can define other entry points as required.

Table 9.7 shows the layout of the first section of the PROM consisting of the jump table, package records and package loader.

```
        j       boot_pkg        # 0xbfc00000
        li      a0,7            #  loads default package in branch delay slot
        ...                     #  available jump table entries
        j       it_bevutlb      # bfc00100
        nop
        ...                     #  available jump table entries
        j       it_bevgen       # bfc00180
        nop
        ...                     #  available jump table entries
        j       it_bevtlb       # bfc00200 (reserved for R4000)
        nop
        ...                     #  available jump table entries
        j       it_bevxtlb      # bfc00280 (reserved for R4000)
        nop
        ...                     #  available jump table entries
        j       it_bevcache     # bfc00300 (reserved for R4000)
        nop
        ...                     #  available jump table entries
        j       it_bevgen       # bfc00380 (reserved for R4000)
        nop
        ...                     #  available jump table entries
package_records:                # bfc00400
        ...                     # package information records
boot_pkg:
        ...
        /*
         * decide which bit of code to execute
         * based on package records and a0
         */
        ...
        la      v0,package_pointer
        lw      v0,16(v0)
        j       v0
```

Table 9.7: PROM structure

9.7.3. The NVRAM environment

The NVRAM consists of 2040 bytes of memory. The NVRAM is used for the non-volatile environment, controlling the boot tests and providing other board related information.

9.7.4. NVRAM structure

The first 64 locations of the environment area are special in that they are not checksummed. This allows low-level code that does not have the ability or time to recalculate the checksum to store values in the NVRAM. The remainder of the NVRAM (1076 bytes) is used to hold the environment strings. The environment consists of a set of strings preceded by their length.

Table 9.8 shows the detailed layout of the NVRAM environment area.

```
.half   magic
.half   sum              # checksum of environment area
.half   envsize          # total size of environment
.byte   tst              # NVRAM test byte
...                      # other unchecksummed locations
/* environment starts at offset 64 */
.byte   25               # length of env string
.ascii  "ethaddr=00:40:bc:00:01:00"
.byte   8
.ascii  "itquiet="
.byte   12
.ascii  "itloglevel=5"
...
```

Table 9.8: NVRAM environment structure

9.7.5. Exception vectors and re-vectoring from AlgPOST

AlgPOST only uses the "bootstrap exception vectors". This means that exceptions are vectored via a PROM location. In the MIPS architecture a CPU status register bit can be changed to cause exceptions to be vectored through low memory - as a real operating system will do.

AlgPOST provides a mechanism which permits other knowledgable PROM code to grab exceptions where appropriate - this mechanism is sometimes used internally by AlgPOST tests which expect to cause an exception. If the CPU *k0* register is non zero when AlgPOST catches an exception, then execution will be re-directed to the address in *k0*. No state is changed, so the code at the *k0* location can be just like any other MIPS exception handler. Before transferring control the *k0* register is cleared, so a double exception will be treated as "unexpected."

10. Glossary

72001: NEC serial port controller (used on many Algorithmics boards for peculiar historical reasons), sometimes also called “MPSC” for “multi-protocol serial controller”.

Big-Endian: see “Endianness”

Bisexual: used to describe a binary program which can run in either big-endian or little-endian mode.

Bitorder, Byteorder: when a little-endian CPU is connected to the big-endian VMEbus, complete compatibility of data representation is impossible. These describe two options: “Bitorder” is the result of preserving 32-bit words between the big- and little-endian worlds, so that aligned 32-bit integers are compatible but strings are disordered (“UNIX” on one side becomes “XINU” on the other). “Byteorder” describes the result of achieving compatibility in string order and byte addressing, but causes the representation of integers to be different.

Cache: a fast memory used to keep recently-referenced data in the hope that it will be used again soon. Any MIPS CPU has onchip data and instruction caches for high performance.

Checksum: a consistency check on a piece of memory, usually obtained by summing all the memory contents as if it was an array of integers and then storing the result at the end of the memory block. Used to ensure the validity of the non-volatile memory contents.

Console: a serial port used for communication with AlgPOST. Any or all of the P−4000i’s serial ports can be assigned as consoles, see §9.4.

CR register: the MIPS CPU “Cause” register; see [Architecture].

DMA: used here to mean any memory transfer which is not performed by the CPU.

Endian, Endianness: refers to how a CPU stores data items bigger than a byte in a byte-addressible memory. Big-Endian CPUs store the most significant bits of integers in the lowest memory locations; little-endian CPUs store the least significant bits of integers in the lowest memory locations. The name comes from “Gulliver’s Travels”. [Algorithmics] contains a section describing the awful consequences of that boards’ willingness to be configured in either way.

Environment variable: a piece of data stored under a mnemonic name in the non-volatile memory, see §9.5.

envname, envval: respectively the name and stored value of an environment variable.

EPC register: the MIPS CPU “exception PC” register; see [Architecture].

Exception: a MIPS word (see [Architecture]), meaning all interrupts and traps.

FPA: floating-point accelerator - hardware to carry out floating point arithmetic.

Halfword: a 2-byte quantity.

I-cache: instruction cache, see “cache”.

IO bus: the P−4000i is internally organised with a *local bus* which connects the CPU and onboard memory, which is in turn connected to an *IO bus* which connects all other controllers and memories. Since the PROM is on the IO bus, the IO bus control circuits must be functioning for AlgPOST to do anything useful.

Little-endian: see “endianness”

Loglevel, log level: refers informally to the environment variable *itloglevel* is crucial to determining how verbose are the reports from AlgPOST. *itloglevel* is described in §9.5.

Loopback: some controller ICs can be put into a mode where output from the interface can be internally wired straight back into the input. This allows the controller IC and its support logic to be checked regardless of what the interface is connected to.

MIPS: from MIPS Computer Systems Inc, often used as a description of their particular RISC architecture.

MPSC: multi-protocol serial interface controller: an acronym sometimes used for the NEC 72001 serial port controller used on the Algorithmics.

NVRAM: non-volatile RAM. The Algorithmics is equipped with a module which contains a low-power static RAM, battery and real-time clock in one unit. AlgPOST just requires a RAM store whose values persist when power is removed.

PROM, ROM: the Algorithmics read-only memory - the board has two 32-pin DIL sockets for JEDEC-standard uV-erasable PROMs. By association, used to describe any program which is designed to run from a read-only memory.

RA register: the MIPS CPU “return address” register $31; see [Architecture].

RS232: the CCITT interface standard for serial I/O, often used to describe the four serial ports of the Algorithmics.

SR register: the MIPS CPU “status” register; see [Architecture].

Timeout: the expiry of a maximum time permitted for something to happen, which ought to have happened. There is a hardware timeout provided for onboard accesses to IO devices (implemented by the VIC068).

UART: “universal asynchronous receiver/transmitter”: a name for a serial port controller, applicable to the one inside the VAC068.

tlbmiss: a particular sort of exception in the MIPS architecture, caused by a reference to a virtual address for which no memory-mapping information is available. It is a very common unexpected exception in a malfunctioning board.

VADDR register: the MIPS CPU “Bad Virtual Address” register; see [Architecture].

Variable: usually means an environment variable (see above).

XOFF: literally, another name for the ASCII character produced by Control-S. Used to refer to a flow control convention on serial ports: a person or computer being swamped with too much output can send an XOFF to ask the sender to pause. Used by AlgPOST.

11. References

Architecture: Gerry Kane, *MIPS R3000 RISC ARCHITECTURE*, Prentice Hall, 1987.

Architecture: Gerry Kane, *MIPS R4000 Microprocessor User's Manual*, MIPS Computer Systems, Inc, 1991

Architecture: MIPS Computer Systems Inc, *MIPS R-Series Architecture*.

P4000i: Algorithmics Ltd, *P−4000i User's Manual* (describes the P−4000i hardware) Algorithmics Ltd, 1994.

LSI-PMON: LSI Logic Inc, *MIPS PROM Monitor and C function Run Time Library User's Guide Release 4.0* (describes LSI's full PMON package, which includes more than just a PROM monitor). Published by LSI Logic, 1993.

MIPS prog: Erin Farquhar and Philip Bunce, *The MIPS Programmer's Handbook* (a useful backgrounder particularly for programmers new to the MIPS architecture). Published Morgan Kaufmann, ISBN 1-55860-297-6.