

PMON Reference Manual (Getting Started with PMON – Version 6.50)

1.0 Introduction

The purpose of this manual is to introduce the BRECIS boot loader for operating systems such as Linux and proprietary operating systems.

BRECIS PMON is derived from the original PMON. The original sources are at <http://www.carmel.com/pmon/>. Please refer to this web site for more information.

2.0 Development Environment

The following components are required for software development:

- a. Host system (Linux Operating System on a PC, RedHat 7.2/7.3)
- b. MIPS cross-compiler (Available from BRECIS)
- c. RS232 Console/Terminal
- d. Target EVM board

3.0 Build PMON under Linux host

The following steps build/rebuild PMON:

Step 1: Copy PMON from CD to hard drive with directory named “pmon”.

Step 2: Make the pmon directory writeable by issue the following command:

```
chmod -R 777 pmon
```

Step 3: Clean up the object code: (clean is a script file)

```
./clean
```

Step 4: Build the object code: (bld is a script file)

```
./bld
```

4.0 Flash memory organization

The following is the flash memory organization defined by BRECIS. Users can change it according to their needs, however, it is your responsibility to make sure no conflict exist between the blocks.

<i>address</i>	<i>description</i>
bfc00000	pmon code in flash
bfc70000	pmon startup script
bfc80000	Applications (e.g. uClinux)

5.0 SDRAM memory organization

The following is the SDRAM memory organization defined by PMON.

<i>address</i>	<i>description</i>
80000000	Exception vectors in SDRAM
80000200	pmon data area
80040000	Default start of Applications SDRAM area
End-pmon size - 4000	pmon relocated code
End - 4000	Unused by pmon

1. Start the RTOS at or above 0x80040000.
2. Do NOT use the last 0x30000 of SDRAM, where PMON resides in SDRAM,
3. PMON reserves that last 0x4000 of SDRAM for RTOS private storage.
PMON will not touch this memory.

6.0 Set up RS232 Console

Set the baud rate to 57600 bps. Under Linux, one can use Linux terminal application minicom.

7.0 PMON command listing

The following lists the commands used for PMON.

Detail descriptions are listed in Appendix A.

Command Category	Command	Status
Downloading	Load a File (load)	Supported
	Define a symbol (sym)	Supported
	List a symbol (ls)	Supported
Display/Set Registers/Memory	Display/Set Register (r)	Supported
	Modify Memory (m)	Supported
	Display Memory (d)	Supported
	Disassemble Memory (l)	Supported
	Fill Memory (fill)	Partial (no sanity check)
	Copy Memory (copy)	Supported
	Compare Memory (compare)	Supported
	Search Memory (search)	Partial (no sanity check)
Execution Control	Dump Memory (dump)	Supported
	Start Execution (g)	Supported
	Display/Set Breakpoints (b)	Supported
	Set Complex Breakpoint (when)	
	Delete Breakpoint (db)	
	Single Step (t/to)	
	Continue Execution (c)	

	Execute Subroutine (call)	
Miscellaneous	Help (h)	Supported
	History (hi)	Supported
	Execute a script (exec)	Supported
	Display/Set environment variable (set)	Supported
	Set Terminal Parameters (stty)	
	Erase flash (erase)	Supported
	Flush the Caches (flush)	Supported
	Display the version number (vers)	?? (need to verify data)
	The Command Shell	Supported
	Paginate (more command)	Supported

8.0 Load application

The application (or RTOS) can be loaded through Ethernet into SDRAM. The HOST is assumed to be running Linux (Redhat 7.2/7.3) and PMON has been burned into FLASH before the target is powered up.

The following is an example to set up and download uClinux.

8.1 Set the Target's MAC and IP address for MAC0

Under the PMON console, type the following commands:

```
set etheraddr 00:02:d8:00:00:17
```

```
set ipaddress 192.168.1.1
```

The variable hostport controls what ethernet MAC will be used.

To use MAC0, set hostport to ethernet -or- ethernet0

```
set hostport ethernet
```

-or-

```
set hostport ethernet0
```

To use MAC1,

```
set hostport ethernet1
```

If your EVM board has MAC2, you can use MAC2

```
set hostport ethernet2
```

8.2 Download script file to SDRAM and save it to flash.

Step 1: Create a file similar to the following:

(The following example script is from the uClinux Release in uClinux/documentation/pmon-script. There is an example script in pmon/examples/pmon-script.)

```
#!/bin/pmon
# set some initial boot parameters
set ipaddr 192.168.15.2
set etheraddr 00:02:d8:00:00:50
set heaptop 80100000
#      Load binary image at 80100000
load -B
#
# set parameters for kernel
set modetty0 "57600,n,8,1,hw"
set bootserport tty0
set cpuconfig ""
set ethaddr0 00:02:d8:00:00:50
set ethaddr1 00:02:d8:00:00:51
# set ethaddr2 00:02:d8:00:00:52
set baseboardserial 0000000086
set bootprot tftp
set bootserver 0.0.0.0
set bootfile ""
#
# The following is how two interfaces might be configured.
# The following starts the kernel invoking kgdb using either scc0 or console.
# g 80100000 -c kgdb=scc0
ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:100f
ip=192.168.16.2::::eth1:none:an
# g 80100000 -c kgdb=console
ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:100f
ip=192.168.16.2::::eth1:none
#
# The following is one way to start the kernel with CONFIG_IP_PNP=y
# (parsing "ip=" arguments).
g 80100000 -c ip=192.168.15.2::192.168.15.1::Brecis:eth0:none
ip=192.168.16.2::::eth1:none
#
# Set three ethernet addresses via:
# g 80100000 -c ip=192.168.15.2::192.168.15.1::Brecis:eth0:none
ip=192.168.16.2::::eth1:none ip=192.168.17.2::::eth2:none
#
# Format of the above "g" commands are as follows:
# "g <kernel_entry_address> -c <arguments to the uClinux kernel>
```

```

#
# The format of the "ip=argument" is as follows:
#
# <client-ip>:<server-ip>:<gw-ip>:<netmask>:<host
name>:<device>:<PROTO>:<PHY CONFIG>
# Any of the fields can be empty which means to use a default value:
#   <client-ip>   - address given by BOOTP or RARP
#   <server-ip>   - address of host returning BOOTP or RARP packet
#   <gw-ip>       - none, or the address returned by BOOTP
#   <netmask>     - automatically determined from <client-ip>, or BOOTP
#   <host name>   - <client-ip> in ASCII notation, or returned by BOOTP
#   <device>      - device
#   <PROTO>:
#     off|none    - don't do autoconfig at all (DEFAULT)
#     on|any      - use any configured protocol
#     dhcp|bootp|rarp - use only the specified protocol
#     both        - use both BOOTP and RARP (not DHCP)
#   <PHY CONFIG>:
#     an          - auto-negotiate (DEFAULT)
#     10h         - set to 10 Mbit, Half Duplex
#     10f         - set to 10 Mbit, Full Duplex
#     100h        - set to 100 Mbit, Half Duplex
#     100f        - set to 100 Mbit, Full Duplex
#!eof

```

Lines beginning with “#” are comments.

Step 2: Set up PMON to load the script by typing the command
load -B

Step 3: ping the target to make sure the Ethernet connection is up.

Step 4: Invoke tftp and download the pmon-script.

Step 5: The console shows “loaded data from ...” when downloading is complete

Step 6: Save script from SDRAM into Flash.

The following screen shot shows a PMON session doing the initial loading of the pmon-script.

```
Copyright (c) 2001-2002 BRECIS Communications Corporation.  
No manufacturing flash config at bfc00020  
  
PMON version 6.5.2 [EB], Mon Jul  8 14:49:31 CDT 2002  
  
Chip select 0, Flash chip TC58FVT321, address 0xbfc00000, size 0x400000  
Chip select 1, Flash chip TC58FVT321, address 0xbe000000, size 0x400000  
Ethernet MAC address 00:02:d8:00:00:00, IP address 192.168.254.214  
CPU type BRECIS 4000, ErrorEPC (may be PC before reset): bfc1fd60.  
CPU clock frequency 150 MHz Assumed. Avail RAM 65136 KBytes.  
Type 'h' for on-line help.  
  
No script at address bfc70000 beginning with <#!/bin/pmon> found  
PMON> # The above line indicates there was no script for pmon to run.  
PMON> # Also, the "No manufacturing flash config at bfc00020" indicates  
PMON> # there was no manufacturing info record.  
PMON>  
PMON> # First, we need to set the ipaddr pmon will use for loading  
PMON> # a script, the ethernet mac address, optionally the interface  
PMON> # pmon should use, and finally do the load command to cause  
PMON> # pmon to act as a tftp server.  
PMON> set ipaddr 192.168.15.2  
PMON> # The following ethernet MAC address is an example. Please use  
PMON> # a unique ethernet MAC address.  
PMON> set etheraddr 00:02:d8:00:00:50  
PMON> # select the ethernet interface to use. by default pmon will  
PMON> # use ethernet MAC0. The variable, hostport, controls which  
PMON> # interface. To display the value of hostport, type the following:  
PMON> set hostport  
    hostport = ethernet  
PMON> # if hostport is ethernet or ethernet0, pmon uses MAC0  
PMON> # if hostport is ethernet1, pmon uses MAC1  
PMON> # if hostport is ethernet2, and MAC2 exists, pmon uses MAC2.  
PMON> set hostport ethernet2  
PMON> # issue the command so pmon will act as a tftp server  
PMON> # while this command is executing, pmon will respond to  
PMON> # ping commands. pmon will not respond to ping commands otherwise.  
PMON> load -B  
Downloading from ethernet2, ^C to abort  
Loaded data from 80040000 to 80040981, length 982 (2434 decimal)  
PMON> # When the tftp command finishes, the address where the data  
PMON> # is loaded and the length in hexadecimal is printed.  
PMON> # This data is a pmon script, so copy it to the location where  
PMON> # pmon expects the pmon startup script to be located.  
PMON> copy -f -f 80040000 bfc70000 982  
Updating flash block which starts at bfc70000  
PMON> # The script is now copied to flash.  
PMON> # Each time the EVM board is powered up, pmon will execute this script.  
PMON> █
```

Note: The address bfc70000 is the fixed location for the startup script. The startup script will be executed automatically during next power-up.

The following screen shot shows a ping of the EVM board followed by the tftp of the pmon script to PMON.

```
rsewill@rsewill:~ <4:12> $ # ping pmon to see if it is alive
rsewill@rsewill:~ <4:13> $ ping -n 192.168.15.2
PING 192.168.15.2 (192.168.15.2) from 192.168.15.1 : 56(84) bytes of data.
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=1.79 ms
64 bytes from 192.168.15.2: icmp_seq=2 ttl=64 time=0.849 ms
64 bytes from 192.168.15.2: icmp_seq=3 ttl=64 time=0.838 ms
64 bytes from 192.168.15.2: icmp_seq=4 ttl=64 time=0.831 ms

--- 192.168.15.2 ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 3006ms
rtt min/avg/max/mdev = 0.831/1.077/1.791/0.412 ms
rsewill@rsewill:~ <4:14> $ # transfer the pmon script to pmon
rsewill@rsewill:~ <4:15> $ tftp 192.168.15.2
tftp> bin
tftp> rex 1
tftp> put pmon-script
Sent 2434 bytes in 0.0 seconds
```

8.3 Load RTOS from Ethernet

This step assumes the RTOS we wish to load is a uClinux kernel.

Step 7: Prepare a Linux script for loading the uClinux image similar to the uClinuxload script file found in the uClinux documentation folder:

```
#!/bin/bash
EVMIPADDR=192.168.15.2
echo "Under pmon."
echo "                load -B"
echo "                g <options...>"

echo " IP <$EVMIPADDR> put </tftpboot/${USER}/image.bin> 80100000"
tftp $EVMIPADDR <<EOF
bin
rex 1
put /tftpboot/${USER}/image.bin 80100000
EOF
```

Note: Assume the file name of the uClinux image is image.bin which is found in the /tftpboot/\${USER} directory. It will be loaded into the SDRAM starting from 80100000 because this is where the image.bin is built to execute.

```

Copyright (c) 2001-2002 BRECIS Communications Corporation.
No manufacturing flash config at bfc00020

PMON version 6.5.2 [EB], Mon Jul  8 14:49:31 CDT 2002

Chip select 0, Flash chip TC58FVT321, address 0xbfc00000, size 0x400000
Chip select 1, Flash chip TC58FVT321, address 0xbe000000, size 0x400000
Ethernet MAC address 00:02:d8:00:00:00, IP address 192.168.254.214
CPU type BRECIS 4000. ErrorEPC (may be PC before reset): bfc1fd58.
CPU clock frequency 150 MHz Assumed. Avail RAM 65136 KBytes.
Type 'h' for on-line help.

<Executing script at address bfc70000>
# <#!/bin/pmon>
# <# set some initial boot parameters>
# <set ipaddr 192.168.15.2>
# <set etheraddr 00:02:d8:00:00:50>
# <set heaptop 80100000>
# <#   Load binary image at 80100000>
# <load -B>
Downloading from ethernet, ^C to abort

```

Step 8: Power off and power on the EVM board. The following screen shows the EVM board powered up and waiting for the uClinux image to be downloaded.

```

rsewill@rsewill:~ <4:24> $ ./uClinuxload
Under pmon:
                                load -B
                                g <options...>
IP <192.168.15.2> put </tftpboot/rsewill/image.bin> 80100000
tftp> tftp> tftp> Sent 4123232 bytes in 12.8 seconds
tftp> rsewill@rsewill:~ <4:25> $ █

```

Step 9: Execute the uClinuxload script file from the host:

Note: The file size of uClinux image is 4123232 bytes decimal and is equal to hex 3EEA60.

Step 10: The following screen shows up downloading complete and the downloaded uClinux kernel executing (instead of PMON).

```

Terminal
File Edit Settings Help

# <load -B>
Downloading from ethernet, ^C to abort
Loaded data from 80100000 to 804eea5f, length 3eea60 (4123232 decimal)
# <#>
# <# set parameters for kernel>
# <set modetty0 "57600,n,8,1,hw">
# <set bootserport tty0>
# <set cpuconfig "">
# <set ethaddr0 00:02:d8:00:00:50>
# <set ethaddr1 00:02:d8:00:00:51>
# <# set ethaddr2 00:02:d8:00:00:52>
# <set baseboardserial 0000000086>
# <set bootprot tftp>
# <set bootserver 0.0.0.0>
# <set bootfile "">
# <#>
# <# The following is how two interfaces might be configured.>
# <# The following starts the kernel invoking kgdb using either scc0 or console>
# <# g 80100000 -c kgdb=scc0 ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:10>
# <# g 80100000 -c kgdb=console ip=192.168.15.2::192.168.15.1::Brecis:eth0:none>
# <#>
# <# The following is one way to start the kernel with CONFIG_IP_PNP=y (parsing>
# <g 80100000 -c ip=192.168.15.2::192.168.15.1::Brecis:eth0:none ip=192.168.16.>

Linux started...
YAMON MEMORY DESCRIPTOR dump:
prom_memsize = 0x03fdc100
prom_heaptop = 80100000
[0,8050da40]: base<00000000> size<00100000> type<Dont use memory>
[0,8050da4c]: base<00100000> size<0040e000> type<Dont use memory>
[0,8050da58]: base<0050e000> size<03ace100> type<Free memory>
prom_memsize = 0x03fdc100
prom_heaptop = 80100000
Config serial console: ttyS0,57600
CPU revision is: 00018305
Primary instruction cache 16kb, linesize 16 bytes (4 ways)
Primary data cache 16kb, linesize 16 bytes (4 ways)

Linux version 2.4.14 (rsewill@rsewill.brecis.com) (gcc version 3.1) #1 Thu Jul 2
Determined physical RAM map:
  memory: 00100000 @ 00000000 (reserved)
  memory: 0040e000 @ 00100000 (reserved)
  memory: 03ace100 @ 0050e000 (usable)
On node 0 totalpages: 16348
zone(0): 16348 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: ip=192.168.15.2::192.168.15.1::Brecis:eth0:none ip=192.168e
calculating r4koff... 000b71b0(750000)
CPU frequency 150.00 MHz
Calibrating delay loop... 149.91 BogoMIPS
Memory: 59188k/60216k available (959k kernel code, 1028k reserved, 53k data, 0k)
Dentry-cache hash table entries: 8192 (order: 4, 65536 bytes)
Inode-cache hash table entries: 4096 (order: 3, 32768 bytes)
Mount-cache hash table entries: 1024 (order: 1, 8192 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Serial driver version 5.05c (2001-07-08) with no serial options enabled
ttyS00 at 0xbc000100 (irq = 19) is a 16550A
block: 128 slots per queue, batch=32
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
Entering MSPPHYGSetup() for MAC#: 0
PHY is set for Auto-Negotiation
AUTO NEGOTIATION COMPLETE
MSPPHYGGetSpeedAndDuplex: for iMAC# 0
PHY is thSetDuplex(): Setting MAC#: 0 for Full Duplex Mode
Entering MSPPHYGSetup() for MAC#: 1
PHY is set for Auto-Negotiation
**** ERROR *** NO LINK DETECTED
ethaddr2 not set in boot prom
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 1 disk images:
  0: 801FEA60-804EEA5F [VIRTUAL A01FEA60-A04EEA5F] (RO)
PPP generic driver version 2.4.1

```

8.4 Break the application download process and return back to PMON

```
<Executing script at address bfc70000>
# <#!/bin/pmon>
# <# set some initial boot parameters>
# <set ipaddr 192.168.15.2>
# <set etheraddr 00:02:d8:00:00:50>
# <set heaptop 80100000>
# <# Load binary image at 80100000>
# <load -B>
break!
PMON> █
```

After the script file is burned into flash, PMON will execute the script and may try to load the application (most likely). If you want to get into PMON instead of downloading an application, use control-C to terminate the loading process.

8.5 Save uClinux in Flash

Generally, the uClinux image.bin is too large to fit in flash. The uClinux build simultaneously creates a file, image.flash, which is a self extracting gzip file. One loads this file to flash at the address 0xBFC20000. One sets up a bbload script and uses bbload to load image.flash.

A screen capture of the command performed under PMON:

```
Terminal
File Edit Settings Help

# <load -B>
Downloading from ethernet, ^C to abort
break!
PMON> # we will load bbload, the bbload-script, and image.flash (the kernel)
PMON> # to the flash chip in chip select 1, address 0xbe000000.
PMON> # we then move the chip selects causing the flash chip we just loaded
PMON> # to be 0xbfc00000. When we power cycle, we then boot bbload
PMON> load -B
Downloading from ethernet, ^C to abort
Loaded data from 80100000 to 80103647, length 3648 (13896 decimal)
PMON> copy -f -f 80100000 be000000 3648
Are you sure that you want to overwrite the manufacturing record (y/n)?y
Updating flash block which starts at be000000
PMON> # We had a manufacturing record on the flash chip in chip select 1.
PMON> # By saying y, we overwrote it. If we wanted to keep the manufacturing
PMON> # record, we would need to copy the manufacturing record from
PMON> # the flash chip in be000000 to SDRAM, and later copy it back.
PMON> # next load the bbload script.
PMON> load -B
Downloading from ethernet, ^C to abort
Loaded data from 80100000 to 801009a2, length 9a3 (2467 decimal)
PMON> copy -f -f 80100000 be010000 9a3
Updating flash block which starts at be010000
PMON> # next load image.flash, the uClinux kernel.
PMON> load -B
Downloading from ethernet, ^C to abort
Loaded data from 80100000 to 8040a98a, length 30a98b (3189131 decimal)
PMON> # copy image.flash to flash.
PMON> copy -f -f 80100000 be020000 30a98b
Updating flash block which starts at be020000
Updating flash block which starts at be030000
Updating flash block which starts at be040000
Updating flash block which starts at be050000
Updating flash block which starts at be060000
Updating flash block which starts at be070000
Updating flash block which starts at be080000
Updating flash block which starts at be090000
Updating flash block which starts at be0a0000
Updating flash block which starts at be0b0000
Updating flash block which starts at be0c0000
Updating flash block which starts at be0d0000
Updating flash block which starts at be0e0000
Updating flash block which starts at be0f0000
Updating flash block which starts at be100000
Updating flash block which starts at be110000
Updating flash block which starts at be120000
Updating flash block which starts at be130000
Updating flash block which starts at be140000
Updating flash block which starts at be150000
Updating flash block which starts at be160000
Updating flash block which starts at be170000
Updating flash block which starts at be180000
Updating flash block which starts at be190000
Updating flash block which starts at be1a0000
Updating flash block which starts at be1b0000
Updating flash block which starts at be1c0000
Updating flash block which starts at be1d0000
Updating flash block which starts at be1e0000
Updating flash block which starts at be1f0000
Updating flash block which starts at be200000
Updating flash block which starts at be210000
Updating flash block which starts at be220000
Updating flash block which starts at be230000
Updating flash block which starts at be240000
Updating flash block which starts at be250000
Updating flash block which starts at be260000
Updating flash block which starts at be270000
Updating flash block which starts at be280000
Updating flash block which starts at be290000
Updating flash block which starts at be2a0000
Updating flash block which starts at be2b0000
Updating flash block which starts at be2c0000
Updating flash block which starts at be2d0000
Updating flash block which starts at be2e0000
Updating flash block which starts at be2f0000
Updating flash block which starts at be300000
Updating flash block which starts at be310000
Updating flash block which starts at be320000
PMON> █
```

A screen capture of the commands performed on the Linux host:

```
rsewill@rsewill:/tftpboot/rsewill <4:53> $ tftp 192.168.15.2
tftp> bin
tftp> rex 1
tftp> put bbload
Sent 13896 bytes in 0.0 seconds
tftp> put bbload-script
Sent 2467 bytes in 0.0 seconds
tftp> put image.flash
Sent 3189131 bytes in 9.9 seconds
tftp> █
```

Following is the bbload-script for our example:

(An example bbload-script can be found in file bbload/example-script -or- in the file, uClinux/documentation/bbload-script).

```
#!/bin/bbload
# set parameters for kernel
set modetty0 "57600,n,8,1,hw"
set bootserport tty0
set cpuconfig ""
set ethaddr0 00:02:d8:00:00:50
set ethaddr1 00:02:d8:00:00:51
# set ethaddr2 00:02:d8:00:00:52
set baseboardserial 0000000086
set bootprot tftp
set bootserver 0.0.0.0
set bootfile ""
#
# The following is how two interfaces might be configured.
# The following starts the kernel invoking kgdb using either scc0 or console.
# setargs progname kgdb=scc0
ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:100f
ip=192.168.16.2:::eth1:none:an
# setargs progname kgdb=console
ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:100f
ip=192.168.16.2:::eth1:none
#
# The following is one way to start the kernel with CONFIG_IP_PNP=y (parsing
"ip=" arguments).
```

```

setargs progname ip=192.168.15.2::192.168.15.1::Brecis:eth0:none
ip=192.168.16.2:::eth1:none
#
# Set three ethernet addresses via:
# setargs progname ip=192.168.15.2::192.168.15.1::Brecis:eth0:none
ip=192.168.16.2:::eth1:none ip=192.168.17.2:::eth2:none
#
# Format of the "setargs" bbload command is as follows:
# setargs <program name-argv[0]> <arguments to the uClinux kernel>
#
# The format of the "ip=argument" is as follows:
#
# <client-ip>:<server-ip>:<gw-ip>:<netmask>:<host
name>:<device>:<PROTO>:<PHY CONFIG>
# Any of the fields can be empty which means to use a default value:
#   <client-ip>   - address given by BOOTP or RARP
#   <server-ip>   - address of host returning BOOTP or RARP packet
#   <gw-ip>       - none, or the address returned by BOOTP
#   <netmask>     - automatically determined from <client-ip>, or BOOTP
#   <host name>   - <client-ip> in ASCII notation, or returned by BOOTP
#   <device>      - device
#   <PROTO>:
#     off|none    - dont do autoconfig at all (DEFAULT)
#     on|any      - use any configured protocol
#     dhcp|bootp|rarp - use only the specified protocol
#     both        - use both BOOTP and RARP (not DHCP)
#   <PHY CONFIG>:
#     an          - auto-negotiate (DEFAULT)
#     10h         - set to 10 Mbit, Half Duplex
#     10f         - set to 10 Mbit, Full Duplex
#     100h        - set to 100 Mbit, Half Duplex
#     100f        - set to 100 Mbit, Full Duplex
# Use setpc for address to start execution if flash-based initialization
# Note: no attempt is made to copy a kernel from flash to SDRAM in this case.
setpc bfc20000
#!eof

```

A screen capture after switching chip selects showing bbload booting from flash:

```

Terminal
File Edit Settings Help

UART initialized

Copyright (c) 2001-2002 BRECIS Communications Corporation
bbload version 1.0
Processing flash configuration.
#!/bin/bbload
# set parameters for kernel
set modetty0 "57600,n,8,1,hw"
set bootserport tty0
set cpuconfig ""
set ethaddr0 00:02:d8:00:00:50
set ethaddr1 00:02:d8:00:00:51
# set ethaddr2 00:02:d8:00:00:52
set baseboardserial 0000000086
set bootprot tftp
set bootserver 0.0.0.0
set bootfile ""
#
# The following is how two interfaces might be configured.
# The following starts the kernel invoking kgdb using either scc0 or console.
# setargs progname kgdb=scc0 ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:100
# ip=192.168.16.2:::eth1:none:an
# setargs progname kgdb=console ip=192.168.15.2::192.168.15.1::Brecis:eth0:none:
# 100f ip=192.168.16.2:::eth1:none
#
# The following is one way to start the kernel with CONFIG_IP_PNP=y (parsing "ip
# =" arguments).
setargs progname ip=192.168.15.2::192.168.15.1::Brecis:eth0:none ip=192.168.16.2
:::eth1:none
#
# Set three ethernet addresses via:
# setargs progname ip=192.168.15.2::192.168.15.1::Brecis:eth0:none ip=192.168.16
.2:::eth1:none ip=192.168.17.2:::eth2:none
#
# Format of the "setargs" bbload command is as follows:
# setargs <program name-argv[0]> <arguments to the uClinux kernel>
#
# The format of the "ip=argument" is as follows:
#
# <client-ip>:<server-ip>:<gw-ip>:<netmask>:<host name>:<device>:<PROTO>:<PHY C
ONFIG>
# Any of the fields can be empty which means to use a default value:
# <client-ip> - address given by BOOTP or RARP
# <server-ip> - address of host returning BOOTP or RARP packet
# <gw-ip> - none, or the address returned by BOOTP
# <netmask> - automatically determined from <client-ip>, or BOOTP
# <host name> - <client-ip> in ASCII notation, or returned by BOOTP
# <device> - device
# <PROTO>:
# off|none - dont do autoconfig at all (DEFAULT)
# on|any - use any configured protocol
# dhcp|bootp|rarp - use only the specified protocol
# both - use both BOOTP and RARP (not DHCP)
# <PHY CONFIG>:
# an - auto-negotiate (DEFAULT)
# 10h - set to 10 Mbit, Half Duplex
# 10f - set to 10 Mbit, Full Duplex
# 100h - set to 100 Mbit, Half Duplex
# 100f - set to 100 Mbit, Full Duplex
# Use setpc for address to start execution if flash-based initialization
# Note: no attempt is made to copy a kernel from flash to SDRAM in this case.
setpc bfc20000
#!eof
No manufacturing flash config at bfc00020
heaptop=80001000 dynamically added
clkfreq=150 dynamically added
memsize=0x4000000 dynamically added
cputype=4000 dynamically added
Finished processing flash configuration.
Inflating image at bfc23844 to 80100000
block 103
Total blocks 103
original crc 0xe5887be9 and length 0x3eea60
last memory location used = 80016700, status = 0

LINUX started...
YAMON MEMORY DESCRIPTOR dump:
prom_memsize = 0x4000000
prom_heaptop = 80001000
[0.8050da40]: base<00000000> size<00001000> type<Dont use memory>

```

Appendix A. Detailed Descriptions of the PMON Commands

B.1. Downloading load

The load command downloads programs and data from the host.

Format

The format for the load command is:

```
load [-abeistvBS] [-baud] [offset] [-c cmdstr]
```

where:

- m Only load the symbol information.
- a suppresses addition of an offset to symbols.
- b suppresses deletion of all breakpoints before the download.
- e suppresses clearing of the exception handlers.
- i ignores checksum errors.
- s suppresses clearing of the symbol table before the download.
- t loads at the top of memory.
- v verbose mode
- B Interprets input from host as binary data for Ethernet transfers.
- S Don't load symbols.
- baud sets the baud rate for transfer.
- offset loads at the specified offset.
- c sets a command string that pmon sends to the host to start a download
- cmdstr operation. String cmdstr is the string that starts the download. Note that the command string must be enclosed in double quotation marks if the string contains any spaces.

Invoking the load command with no parameters or arguments clears the symbol table, deletes all current breakpoints, allows PMON to receive programs or data from the host.

Functional Description

The load command accepts programs and data from the host port in LSI Logic's proprietary FastLoad format, Motorola S-record, or binary files. The user can set environment variables to change the data port, the format, and the transfer protocol. Programs and data are always transferred as binary files over ethernet.

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 UNLESS the program is loaded as a binary file.

Please see the ld command if the program being loaded as a binary file is in the ELF image format or the srec format.

The -c option permits a command string to be sent to the host when the load command is issued and the hostport is tty0. 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 when PMON is reading an srec format or ELF image format file. PMON will still attempt to finish loading the image, but the symbol table will be truncated. To increase the size of the heap, use the set heaptop command to reserve more space in the heap.

The dlecho, dlproto, heaptop, and hostport Variables

The dlecho, dlproto, and hostport variables control operation of the download. The dlecho and dlproto variables only have meaning when downloading through the tty0 hostport. The following table shows how these environment variables affect the operation of the load command.

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 XonXoff	Send Xon and Xoff to control the host
dlproto EtxAck	Expect Etx as end of record, send Ack
heaptop	The heaptop variable determines the default load address when ethernet is used for downloading. Any data must be loaded at or above this address or pmon data will be destroyed.
hostport tty0	Select tty0 as the port to which the host is connected
hostport ethernet	Select ethernet (MAC0) as the port to which the host is connected
hostport ethernet0	Select ethernet (MAC0) as the port to which the host is connected
hostport ethernet 1	Select ethernet (MAC1) as the port to which the host is connected
hostport ethernet2	Select ethernet (MAC2) as the port to which the host is connected

See the section on downloading beginning on page 1-9 for more information on these variables and the use of the load command.

Examples

Download to tty0 using a terminal emulator.

```
PMON> set dlecho off
PMON> set hostport tty0
PMON> set dlproto none
PMON> load
-- use terminal emulator's "send text file" command
Downloading from tty0, ^C to abort
Entry address is 80040000

Total = 0x00043C00 bytes
PMON>
```

Download to ethernet0 using tftp:

```
PMON> # set some initial boot parameters
PMON> set ipaddr 10.26.3.55
PMON> set etheraddr 00:02:d8:00:00:52
PMON> set hostport ethernet0
PMON> set heaptop 80100000
PMON> load -B
Downloading from ethernet0, ^C to abort
Loaded data from 80100000 to 8012d4c7, length 2d4c8
(185544 decimal)
PMON>
On the host computer use tftp as follows:
$ tftp 10.26.3.55
tftp> bin
tftp> rex 1
tftp> put pmon.bin
Sent 185544 bytes in 0.6 seconds
tftp> quit
$
```

Files

The code for the load command is located in pmon/load.c.

See Also

set command for the setup of the environment variables.
ld command.

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 command clears the symbol table. However, there is an option to override the clearing of the symbol table (see the load command for details).

Symbols can be displayed using the ls command.

Examples illustrating the use of this command follow.

```
PMON> sym start bfc00578
PMON> l bfc00578
      start      3c08bfc0 lui      t0,0xbfc0          # 49088
      start+0x4   25080594 addiu   t0,t0,0x594         # 1428
      start+0x8   3c01a000 lui      at,0xa000          # 40960
      start+0xc   01014025 or       t0,t0,at
      start+0x10  01000008 jr       t0
      start+0x14  00000000 nop
      start+0x18  00000000 nop
      start+0x1c  40809000 mtc0     zero,C0_BPC
      start+0x20  00000000 nop
      start+0x24  40809800 mtc0     zero,C0_BDA
PMON> l start 4
      start      3c08bfc0 lui      t0,0xbfc0          # 49088
      start+0x4   25080594 addiu   t0,t0,0x594         # 1428
      start+0x8   3c01a000 lui      at,0xa000          # 40960
      start+0xc   01014025 or       t0,t0,at
PMON> sym flush_cache bfc00b70
PMON> l flush_cache 5
      flush_cache 3c098001 lui      t1,0x8001          # 32769
      flush_cache+0x4 2529cd58 addiu t1,t1,0xcd58       # -12968
      flush_cache+0x8 3c01a000 lui      at,0xa000          # 40960
      flush_cache+0xc 01214825 or       t1,t1,at
      flush_cache+0x10 8d280000 lw      t0,0(t1)
PMON>
```

See Also

ls, load, l, ld, and sh commands.

ls

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

Format

The format for the ls 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 alphabetical order without displaying the actual address for each symbol.

Functional Description

The ls command lists the symbols in the symbol table.

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
flush_cache start
PMON> ls -l
bfc00b70 flush_cache
bfc00578 start
PMON> ls -ln
bfc00578 start
bfc00b70 flush_cache
PMON> ls s*
start

PMON> ls -a bfc00598
bfc00598 start+0x20

PMON> ls -a @epc
bfc00598 = start+0x20

PMON> ls -v @t0+0t10*4
0x800222e8 = 0t-2147343640 =
0o20000421350
```

List symbols in alphabetic order.

List symbols in alphabetic order with addresses.

List symbols and addresses in ascending order of address.

List symbols starting with the letter "s."

List symbol at the next lowest address.

List symbol at the next lowest address from EPC.

Display the value of the expression "@t0+0t10*4":

B.2. Display/Set Registers/Memory

r

The r command sets or displays register values.

Format

The format for the r 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.

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. The '?' character matches any single character, while the '*' character matches any number of any characters. This command accepts both hardware and software names. Examples illustrating the use of the r command follow.

r	Display all General-purpose registers.
r *	Display all register values.
r 8	Display \$8 (t0).
r t0	Display t0 (\$8).
r t*	Display t0 through t9.
r pc	Display PC register.
r pc start	Set PC register to the symbol start value.
r 4 45	Set register 4 to 45.
r t0 45	Set register t0 to 45.
r sr 0	Set SR to zero.
r sr bev 1	Set the BEV bit of SR to one.
r pc a0040000	Set PC to a0040000.
r a	Display the value of pseudo register 'a'.
r ?	Display the value of all pseudo registers.

There are 26 pseudo registers (named a thru z). These can be referenced in the same way as real registers, and are typically used to implement complex breakpoint conditions (see the when command).

Examples

Display all General Purpose registers.

```
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 00000001 80000208
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 83fdb7e8
00000000
00000000
      PC=80040000      HI=00000000      LO=00000000
      C0_SR:  CU  BEV TS  PE  CM  PZ  SWC  ISC  IM&SW  KUo IEo KUp
IEp KUp IEc
      0000  0  0  0  0  0  0  0  0  11111111  0  0  0
0  1  0
      C0_CAUSE: BD CE  IP  SW EXCODE
      0  0 000000 00  Int
      C0_CONFIG: M K23  KU  -  MDU - MM BM BE AT  AR  MT  -  K0
      1 010 011 0000  0  0 00  0  1 00 000 011 0000 011
      C0_PRID: IMP Rev
      131  5
      C0_EPC=80040000      C0_BADVA=dcae2dae
      C0_CCC: EVI CMP IIE DIE MUL MAD TMR BGE IE0 IE1 IS DE0 DE1
DS IPWE
IPWS
      1  0  0  0  0  0  0  0  0  0  0  0  2  0  0
0  0  3
      C0_CCC: TE WB SR0 SR1 ISC TAG INV
      0  0  0  0  0  1  1
      C0_DCS: TR UD KD TE DW DR DAE PCE DE T W R DA PC DB
      0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
M_SCBUS_ST: BEDE BERR
      0  0
M_SCBUS_EA=3c1abfc0
M_EXVI: HEVI SEVI
```

```

          0      0
C0_INDEX=00000000  C0_RANDOM=00000000  C0_ENTRYLO=00000000
C0_CONTEXT=3f6e5710 C0_PAGEMASK=00000000  C0_WIRED=00000000
C0_COUNT=293e6f84  C0_ENTRYHI=00000000  C0_COMPARE=ffffffff
C0_LLADR=0a87bd1c  C0_BPC=00000000      C0_BDA=00000000
C0_BPCM=00000000  C0_BDAM=00000000      C0_ROTATE=0a009c31
C0_CMASK=fd683a11  C0_ERREPC=bf02070c

```

Set the IEC bit of the Status Register.

```
PMON> r sr iec 1
```

Display the Status Register.

```
PMON> r sr
```

```

C0_SR: CU BEV TS PE CM PZ SWC ISC IM&SW KUo IEo KUp IEp KUp IEC
      0000 1  0  0  0  0  0  0  0 00000000 0  0  0  0  0  1

```

You can use the ls command to display the Status Register as a hex value.

This technique can be used for any register.

```
PMON> ls -v @sr
```

```
0x00400001 = 0t4194305 = 0o20000001
```

Files

The r command is located in mon/regs.c.

See Also

l command for disassembling instructions from memory.

m

The m command displays and modifies memory.

Format

The format for the m command is:

```
m [-bhw] [adr [hexval|-s str]..]
```

where:

- b is a flag signifying that byte accesses are to be used.
- h is a flag signifying that halfword accesses are to be used.
- w is a flag signifying that word accesses are to be used.
- 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.
enters interactive mode.
- = in interactive mode, reads current address again.
- ^|- 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 the following table.

If the -s option is specified, pmon 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 a0040000          Display memory at address in interactive
mode.
```

```
a0040000 00 _ User can enter command at cursor (_).
```

```
PMON> m a0040000 1 2 3 4    Set address 0xa0040000 to 1,
address
```

```
                                0xa0020001 to 2, etc., in
noninteractive mode.

PMON> m a0020000          Display memory at 0xa0040000.
a0040000 01 CR
a0040001 02 CR
a0040002 03 CR
a0040003 04 .

PMON> m a0040000 22      Set address 0xa0040000 to 0x22.
PMON> m a0040000          Display memory at 0xa0040000.
a0040000 22 44
a0040001 00
a0040002 00 55
a0040003 00 66
a0040004 00 ^
a0040003 66
a0040004 00 .

PMON> m 80040000 -s even   Set memory starting at 0x80040000
to                          the string "even."

PMON> m 80050100 -s "BRECIS Communications"
Set memory starting at 0x80050100 to
the multiple-word string "BRECIS Communications"
```

Files

The m command is located in mon/modify.c.

See Also

fill command, l command, d command, and dump command.

d

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

Format

The format for the d command is:

```
d [-b|h|w|s|S] adr [cnt|-rreg]
```

where:

- b displays the memory contents in groups of bytes.
- h displays the memory contents in halfword groups.
- w displays the memory contents in word groups.
- s displays the memory contents as a null terminated string.
- S cnt is the number of data items to be displayed.
- adr specifies the base address from which data is displayed.
- cnt specifies the number of lines displayed.
- rreg 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 datasz and moresz Variables

If invoked without a -b, -h, -w, or -s option, the datasz variable sets the display format. Setting datasz to -b, -h, -w, or -s 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.

If invoked without cnt, the d command first displays the number of lines specified in the environment variable moresz. Next, pmon pauses and displays the more prompt. See the more command for commands available with the more prompt. Also see the more command for more information on the moresz variable.

The following example displays memory starting at 0xa0010000.

```
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.....
```

Files

The d command is located in mon/dump.c.

See Also

l command, search command, m command and dump command.

l

The l command disassembles instructions from memory.

Format

The format for the l command is:

```
l [-bctT] [adr [cnt]]
```

where:

- b lists only branches.
- c lists only calls.
- t lists the trace buffer.
- T assume TinyRISC (MIPS16) instructions.
- 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 pmon 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                      Normally the default.

PMON> l bfc00578
bfc00578 3c08bfc0 lui          t0,0xbfc0          # 49088
bfc0057c 25080594 addiu       t0,t0,0x594         # 1428
bfc00580 3c01a000 lui          at,0xa000          # 40960
bfc00584 01014025 or          t0,t0,at
bfc00588 01000008 jr          t0
bfc0058c 00000000 nop
bfc00590 00000000 nop
bfc00594 40809000 mtc0        zero,C0_BPC
bfc00598 00000000 nop
bfc0059c 40809800 mtc0        zero,C0_BDA
PMON> set regstyle hw
PMON> l bfc00578
bfc00578 3c08bfc0 lui          $8,0xbfc0          #
49088
```

```
bfc0057c 25080594 addiu    $8,$8,0x594    #
1428
bfc00580 3c01a000 lui      $1,0xa000      #
40960
bfc00584 01014025 or        $8,$8,$1
bfc00588 01000008 jr         $8
bfc0058c 00000000 nop
bfc00590 00000000 nop
bfc00594 40809000 mtc0       $0,C0_BPC
bfc00598 00000000 nop
bfc0059c 40809800 mtc0       $0,C0_BDA
```

Files

The `l` command is located in `mon/dis.c`.

See Also

`d` command, `m` command, `dump` command, `more` and `rdsrec` commands.

fill

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

Format

The format for the fill 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.

specifies that the memory block should be filled with an ASCII string rather

-s str 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, 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 0xa0040000 to 0xa0041000, enter:

```
PMON> fill a0040000 a0041000 0
```

To fill an area of memory from 0xa0040000 to 0xa00410000 with the string of values 0x41, 0x42, 0x43, 0x44, and 0x45, enter:

```
PMON> fill a0040000 a00410000 41 42 43 44 45
```

To fill an area of memory from 0xa0040000 to 0xa00410000 with the ASCII string "hello world," enter:

```
PMON> fill a0040000 a0041000 -s "hello world"
```

Files

The fill command is located in mon/fill.c.

See Also

m command.

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 [-f] from to size
```

where:

from declares the source address location.

to declares the target address location.

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

-f Specifies that the destination is flash memory.

Functional Description

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

The -f option is used when copying to flash memory. This requires that the board supports writes to that address range, and that the flash memories support sector erase.

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

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. However, if the -f option is specified, copying is always performed in ascending address order.

Examples

- This example shows how to copy a block of memory, 8 Kbytes in size, with a base address of 0x80040000, to another 8-Kbyte area starting at the address 0x80060000.

```
•PMON> copy 80040000 80060000 2000
```

- This example shows how to copy a block of memory, 4 Kbytes in size, with a base address of 0x80040000, to a 4-Kbyte area of flash memory starting at the address 0xbfc80000.

```
•PMON> copy -f 80040000 bfc80000 1000
```

- This example shows how an small application program may be copied into flash.

```
•PMON> load -B
```

```
•Downloading from ethernet2, ^C to abort
•Loaded data from 80100000 to 8010fd4e, length fd4f (64847
  decimal)
•PMON> copy -f -f 80100000 bfd00000 fd4f
•Updating flash block which starts at bfd00000
•PMON> ld bfd00000
•section 1) .text, copying bfd00058 to 80040000, len 2238
•section 2) .data, copying bfd02290 to 80042238, len 1078
•section 3) .sbss, clearing 800432b0, len 28
•section 4) .bss, clearing 800432d8, len 0
•section 8) .symtab, processing global symbols
•      96 symbols of 96 symbols loaded
•Setting pc to 80040000
•PMON> g
•Welcome to TINY caves, you may direct me with commands:
•N, NE, E, SE, S, SW, W, NW, UP, DOWN, or QUIT
•
•You're at road near house.
•
•>quit
•PMON>
•
•
```

Files

The copy command is located in mon/copy.c.

See Also

ld command, g command

compare

The compare command compares the contents of two memory areas.

Format

The format of the compare command is:

```
compare adr1 adr2 siz
```

where:

adr1 specifies the start address of the first memory area.

adr2 specifies the start address of the second memory area.

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

Functional Description

The compare command compares each byte in the two memory areas. Any bytes that are different are displayed with the address, the expected value, and the actual value read.

Examples

- This example shows how to compare two blocks of memory, 8 Kbytes in size. The first block has a base address of 0x80040000, the second block starts at 0x80060000.

```
•PMON> compare 80040000 80060000 2000
```

Files

The compare command is located in mon/compare.c.

See Also

copy command

search

The search command executes a search for a memory pattern.

Format

The format for the search 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, 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 0xa0040000 to 0xa0050000:

```
PMON> search a0040000 a0050000 3c d4
```

The following example searches for "ABC" from 0xa0040000 to 0xa0050000:

```
PMON> search a0040000 a0050000 -s "ABC"
```

Files

The source for the search command is located in mon/search.c.

See Also

d command and more command.

dump

The dump command uploads S-records or binary to the host port.

Format

The format for the dump command is:

```
dump [-B] adr siz
      -B      select binary (default if host port is an ethernet port)
```

where:

adr

is the base address of the data to be uploaded.

siz

the number of bytes to be uploaded.

Functional Description

The dump command uploads Motorola S-records or binary to the host port. For S-records, all uploaded S-records except the terminating S-record are S3-records. The terminating S-record is an S7-record.

When uploading Motorola S-Records:

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 "off", the lines will be terminated by a carriage return ("\r") and a linefeed character ("\n").

If ulcr is set to "on", each line will be terminated by a linefeed character ("\n") only.

The default value for ulcr is "off".

To upload an area of memory using ethernet, type:

```
PMON> dump -B bfc70000 1000
```

Uploading to ethernet2, ^C to abort

Dumped data from bfc70000 to bfc70fff, length 1000 (4096 decimal)

```
PMON>
```

On the host computer, use tftp to get the data being uploaded:

```
$ tftp 10.26.3.55
```

```
tftp> bin
```

```
tftp> rex 1
```

```
tftp> get pmon-scriptarea
```

```
Received 4096 bytes in 0.0 seconds
```

```
tftp> quit
```

```
$
```

Files

The dump command is located in mon/sdump.c.

See Also

l command, d command, and m command.

g

The g command starts program execution.

Format

The format for the g command is:

```
g [-st] [-p progname ] [adr [bptadr]] [-c args]  start execution (go)
    -p  specify program name progname in argv[0]
    -s  don't set client sp
    -t  time execution
    <adr>  start address
    <bptadr>  temporary breakpoint
    -c <args>  args to be passed to client
```

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

Functional Description

The g command starts program execution. If the user does not specify the starting address adr, execution starts at the current value of the EPC register. This command must only be used once after downloading a new program. Use the c command to continue execution after a breakpoint.

If the user specifies the -c option, pmon passes all arguments after -c to the client program by the following method:

1) If -c is not specified,
a0 will be set to 1 (argc),

a1 will point to an array of pointers (argv)
where the first pointer points to progname
if the -p progname option was specified
and "g" otherwise
This array will be terminated by a NULL pointer

a2 will point to an array of pointers (envp)
where each pointers points to a zero terminated string,
"variable=value"
This array will be terminated by a NULL pointer

a3 will be the NULL pointer

2) if -c is specified,
a0 will be set to the number of arguments (argc),

a1 will point an array of pointers (argv)

where the first pointer points to progname
if the -p progname option was specified
and "g" otherwise
This array will be terminated by a NULL pointer

a2 will point to an array of pointers (envp)
where each pointers points to a zero terminated string,
"variable=value"
This array will be terminated by a NULL pointer

a3 will be the NULL pointer

If adr is specified, a temporary breakpoint (bptadr) may also be specified. The temporary breakpoint remains in effect only until the next time that program execution is halted. The character '.' may be used as a placeholder for the adr if you wish to specify a temporary breakpoint without specifying a start address.

Examples illustrating the use of the g command follow.

```
PMON> g                                Start executing at the
current value of                        the EPC register.

PMON> g 80040000                        Start executing at 0x80040000.

PMON> g 80040000 80040008              Start executing at 0x80040000 and
break at 0x80040008.
```

Files

The g command is located in mon/go.c.

See Also

c command

b

The **b** command sets and displays breakpoints.

Format

The format for the **b** command is:

```
b
b adr..
b adr -s str
```

where:

- adr* specifies an address for the breakpoint. Up to 32 breakpoints addresses can be set.
- s *str* executes the command string when the breakpoint is hit.

Invoking the **b** command with no options causes the Monitor to print a list of the current breakpoints.

Functional Description

The **b** command sets a breakpoint at the specified address or addresses. Multiple addresses may be specified. Specified addresses must be word-aligned.

This command is effectively a "pass-thru" to the **when** command. Issuing the command, "**b main**" is equivalent to issuing the command, "**when @pc==main stop**". However, unlike the **when** command, the **b** command can take multiple addresses.

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

pmon reports a new breakpoint's number immediately after the breakpoint is set (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.

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:

```
brkcmd = "l @pc 1"
```

This command "**l @pc 1**", specifies that when the breakpoint occurs, **pmon** will disassemble one line starting at the address of the program counter.

You can change the breakpoint command variable with the **set** command. For example, you can include additional monitor commands in the **brkcmd** variable. You must 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.

```
set brkcmd "l @pc 1;r *"
```

By default, breakpoints are cleared when the load command is executed. See the section on the load command later in this document for details on how to override automatic breakpoint clearing after a download operation.

Some examples illustrating the use of the b command follow.

```
PMON> b 8004000c    Set a breakpoint at 0x8004000c.
Bpt 1 = 8004000c
PMON> b             Display all breakpoints.
Bpt 0 = 8004022c
Bpt 1 = 8004000c
PMON> b 80041248    Set a breakpoint at 0x80041248. Display registers when
-s "r"              the breakpoint is encountered.
```

See Also

db, when, .d, ld, and load commands.

when

The when command sets complex breakpoints

Format

The format for the when command is:

```
when condition action
```

where:

condition Specifies the conditions under which the breakpoint will occur.

action Specifies the action to be taken when the breakpoint occurs.

Functional Description

The when command provides a very flexible way to set breakpoints and to specify the action that should be taken when they are encountered.

Each when command takes two arguments, a condition, and an action. The condition can be something as simple as @pc==main (when the program counter is equal to value of the symbol main), or it can be a complex expression that might include specifying the contents of memory locations.

Actions can be any pmon command string. But it can also include the pseudo command stop. For example, the command, "when @pc==main stop" specifies that when the program counter has the value of the symbol main, execution should stop.

Conditional expressions can be combined in any arbitrary manner to provide extremely complex breakpoint conditions. For example,

```
PMON> when @pc==main2 && (^tcbchn==task1 || ^tcbchn==task2)
stop
```

which will stop execution when the pc is equal to main2, and either tcbchn is equal to task1 or task2.

In the preceeding examples we have chosen to stop execution when the condition is met. However, it is possible to specify a list of commands. For example,

```
PMON> when @pc==sort "r;d -w dat3 1"
```

specifes that when the pc is equal to 'sort', the commands 'r' and 'd -w dat3 1' will be executed. The ';' is used to separate commands, and the double quotes are necessary because the argument contains spaces.

pmon is responsible for deciding when to use hardware breakpoint registers (if present). In general, pmon uses software breakpoints unless there is a specific reason that a hardware breakpoint is required.

If you issue a command that requires a hardware breakpoint register. But that pmon finds that there is not one available. pmon will automatically use trace mode when you issue the c command to continue execution. Because trace

mode is not real-time, pmon will warn you at the time that you set the breakpoint that this will require non real-time execution.

Examples

- Stop when a memory location changes.

You can specify data locations by using the dereferencing operator '^'. For example,

```
PMON> when '^tcbchn!=0' stop
```

means that execution should stop when the contents of the memory location specified by the symbol tcbchn is not equal to zero. Note that expressions that include the character '!' must be enclosed within single- or double-quotes to suppress the normal history substitution mechanism. Single- or double-quotes must also be used if the expression contains spaces.

- Break on the 20th time around a loop.

- `when @pc==0x80041234 "r a @a+1"`
- `when @a==0t20 stop`

These commands specify that execution should pause each time the pc has the value 0x80041234. Each execution pauses, the command `"r a @a+1"` will be executed. This command increments the value of pseudo register 'a'. See the `r` command for more information on the pseudo registers.

If during one of these pauses in execution pseudo register 'a' has the value 20 (decimal), execution will stop.

- Break when `fred()` is executed after `jim()`, but not after `mike()`.

- `when @pc==jim "r a 1"`
- `when @pc==mike "r a 0"`
- `when "@pc==fred && @a==1" stop`

- Stop if `dat1` ever gets set back to zero.

- `when "^dat1 != 0" "r a 1"`
- `when "^dat1 == 0 && @a == 1" stop`

This requires a hardware data breakpoint register in order to execute this in real-time. In this mode a brief pause will occur each time a write to `dat1` occurs.

If you don't have a hardware data breakpoint register, you can maintain real-time performance by only checking the value at specific points in the program.

```
when "@pc == fred && ^jim != 0" "r a 1"
when "@pc == fred && ^jim == 0 && @a == 1" stop
```

- Find the maximum value written to a specified memory location.

- `when "^dat1 > @a" "r a ^dat1"`

Use the command "r a" to examine the value when the program stops.

- Stop if jim() is ever called with a 2nd argument of zero.

- `when "@pc == jim && @a1 == 0" stop`

- Count how many times jim() is executed.

- `when "@pc == jim" "r a @a+1"`

Use the command "r a" to examine the value when the program stops.

See Also

b, db, ld, and load commands.

db

The db command deletes the specified breakpoints.

Format

The format for the db 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 *	Delete all breakpoints.
PMON> db	Display all breakpoints.
Bpt 0 = 8004000c	

See Also

d, when, ld, and load commands.

t/to

The **t** command performs a trace (single step) operation.

Format

The format for this command is:

t [-vbci]

or:

to [-vbci]

where:

- v lists each step (verbose).
- b captures only branches.
- c captures only calls (jal instruction).
- i stops on invalid program counter.
- 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.

A branch instruction and the instruction in its delay slot are executed as a single step. This also means that two-instruction loops are treated as a single step. This command is implemented by setting a breakpoint at the following instruction.

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
start+0x240 80040240 lui t1,0xa07f
```

Files

The **t/to** commands are located in **mon/go.c**.

c

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

Format

The format for the **c** 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.

Examples of the **c** command follow.

PMON> **c**

Continue execution until exit or a regular breakpoint is encountered.

PMON> **c** 80040104

Continue execution until 0x80040104 or a regular breakpoint is encountered.

Files

The **c** command is located in **mon/go.c**.

See Also

g command

call

The call command executes a subroutine.

Format

The format for the call command is:

```
call adr [-s str|val]..
```

where:

adr is the address of the subroutine to be executed.

-s str is a string argument.

val is a value to be passed.

The call command calls a function using the standard C calling convention.

The "-s *str*" and *val* options permit arguments to be passed to the subroutine.

Functional Description

The call command executes a downloaded subroutine, using a jalr instruction to pass control to the specified address. This does not affect the existing value of the saved registers. Instead the subroutine is called directly from C code without restoring the saved registers. Control returns to PMON via the usual subroutine return mechanism.

If the user specifies arguments, these are passed using the standard C calling convention. If the "-s" option is specified, the following argument is assumed to be a string. In this case the address of the string is passed to the subroutine. If a numerical value is specified in place of the "-s", it will be evaluated according to the existing rules and passed to the function. Up to five arguments may be passed.

This command is usually used to provide a method of displaying application-specific data structures. For example, if your application has a complex, linked-list data structure, you might find it helpful to add a function to your program that can display the structure. The *call* command can then be used to invoke this function from pmon prompt at any time in the execution, even between two single-step operations.

Examples illustrating the use of the call command follow.

```
PMON> call prstat          Call the function whos name is
'prstat'.
```

```
PMON> call prrec a0040000+8  Call the function 'prrec' and
pass it                      the value 0xa0040008 as the first
argument.
```

```
PMON> call printf -s "hello world"
                             Call the function printf and pass
it the                       address of the string "hello
world".
```

Files

The call command is located in mon/call.c.

See Also

g command

h

The h command provides on-line help.

Format

The format for the h command is:

```
h [*|cmd-]
```

where:

* provides detailed help on all the commands.

cmd is a command. pmon then provides help on the stated command.

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

Functional Description

The h 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  The help command          help  The help command
      ?  The help command          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          compare compare memory
      when  complex breakpoint      g   start execution (go)
      c     continue execution       t   trace (single-step)
      to    trace over (single-step) b   set breakpoint at
address
      db    delete breakpoint       sym  define symbol
      ls    list symbols             set  display/set variable
      vers  display version info     sh   The command shell
      more  paginator               debug enter gdb mode
      ab    set access (data) bpt    stty Set/display
terminal options
      load  load memory from hostport dump send srecs/binary
to hostport
      mt    memory test             call  call function
      ld    load elf or srec         erase  erase flash
      exec  execute script from address flush flush caches

PMON> h stty

      stty  [opt][<baud>][<term>]    Set/display terminal s
          -v    list possible baud rates and terminal types
```

-a	list all settings
<baud>	set baud rate
<term>	set terminal type
sane	set sane settings
ixany	allow any char to restart output
-ixany	allow only <start> to restart output
ixoff	enable tandem mode
-ixoff	disable tandem mode
echo	enable echo
-echo	disable echo

hi

The hi command lists the command history.

Format

The format for the hi 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      Display the three last commands.
```

```
14 hi 3
```

```
13 hi
```

```
12 l
```

```
PMON> hi      Display the entire history, using more
```

```
13 hi to control the screen output.
```

```
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)
```

See Also

sh command, which maintains a command history.

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.

In some cases, when pmon displays a variable's current value, pmon 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.

The set command does not evaluate the specified value or check the specified value against a list of allowed values. Value checking is only performed when a command uses a variable.

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

Examples illustrating the use of the set command follow.

```
PMON> set
brkcmd = "l @pc 1"
datasz = -b                [-b -h -w]
inalpha = hex              [hex symbol]
inbase = 16                [auto 8 10 16]
moresz = 10
regstyle = sw              [hw sw]
rptcmd = trace             [off on trace]
trabort = ^K
uleof = %
ulcr = off                 [off on]
validpc = "_ftext etext"
dlecho = off               [off on lfeed]
dlproto = none             [none XonXoff EtxAck]
hostport = ethernet
prompt = "PMON> "
etheraddr = 00:02:d8:00:00:52
ipaddr = 10.26.3.55
heaptop = 80040000
diag = 0                   [N[:dev]]
clkfreq = 150
memsize = 0x03fdb800
```

```
cputype = 4000 #
```

```
PMON> set moresz
moresz = 10
```

```
PMON> set moresz 20
```

Display current value of Cause Register and display all general-purpose registers:

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

Environment Variables and Default Values

Environment Variable	Default Value	Options
brkcmd	"l @epc 1"	command list
datasz	-b	[-b -h -w]
dlecho	off	[off on lfeed]
dlproto	none	[none XonXoff EtxAck]
etheraddr	00:02:d8:00:00:52	string
ipaddr	10.26.3.55	string
heaptop	80040000	string
hostport	ethernet	tty0,ethernet,ethernet0,ethernet1,ethernet2
inalpha	hex	hex symbol
inbase	16	[auto 8 10 16]
moresz	10	0-n
prompt	"PMON> "	string
regstyle	sw	[hw sw]
rptcmd	trace	[off on trace]
trabort	^K	unused
ulcr	off	[off on]
uleof	%	string
validpc	"_fext etext"	string
diag	0	[N[:dev]]
clkfreq	150	cpu clock frequency
memsize	0x3fdb800	size of SDRAM excluding SDRAM used by pmon
cputype	4000	BRECIS chip type

Environment variables can be set and displayed using the set command.

Brief descriptions of each of the variables follow, together with references to their complete descriptions.

- **brkcmd** - This variable specifies a sequence of pmon commands that are executed when a breakpoint halts program execution. See the **b** command.

- datasz** - This variable controls whether data is displayed in byte, half-word, or word groups. See the **d** command.
- 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 and the section on flow control.
- dlproto** - This variable selects the download protocol for transfers via RS-232C. **pmon** supports Xon/Xoff and EtxAck download protocols. See the **load** command and the section on flow control.
- etheraddr** - This variable specifies the hardware Ethernet address. See the **load** command and the section on downloading via Ethernet.
- ipaddr** - This variable specifies the Internet Protocol address. See the **load** command and the section on downloading via Ethernet.
- heaptop** - This variable specifies the highest allowable address in the heap maintained by **pmon** from low (0x8000000) memory. **pmon** code is also placed near the end of memory. See the **load** command.
- hostport** - This variable selects whether **tty0**, **ethernet**, **ethernet0**, **ethernet1**, **ethernet2** is the host port. See the **load** command and the section on flow control.
- inalpha** - This variable selects whether strings starting with the ASCII characters **a**, **b**, **c**, **d**, **e**, and **f** are interpreted as symbols or hexadecimal numbers. See the **sh** command.
- 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.
- moresz** - This variable specifies how many lines to display during screen-at-a-time display. See the **more** command.
- prompt** - This variable defines the **pmon** prompt. An example of using this command is when you need to set the prompt to "**PMON>** " for compatibility with a source-level debugger. To do this use the following commands.

This will set the prompt to "**PMON>** " (note the space) and save this new value in the non-volatile memory (if supported).

- regstyle** - This variable defines whether hardware or software names are displayed for the MIPS 4Km registers in the **l** command. See the **l** command.
- rptcmd** - When this variable is set to "on," the previous command is executed again when the user enters an empty line. See the **sh** command.

- ulcr - This variable defines whether there is a carriage return or both a carriage return and a linefeed character at the end of the line during dumps. See the dump command.
- uleof - This variable specifies a string that is sent to the host after a dump to the target has completed. See the dump command.
- validpc - This variable specifies the range of valid PC values during program tracing. See the trace command.
- to the console. However, you may direct it to another port. For
exampldiag - This variable specifies the diagnostic level. zero is off, non-zero is on. Valid values are 0 thru 9. By default the diagnostic output is sent e,
 - PMON> set diag 2:/dev/tty1
- clkfreq is the MIPS CPU clock frequency set by pmon.
- memsize is the amount of available SDRAM excluding SDRAM used by pmon.
- cputype is the BRECIS chip type.

ld

The ld command reads an elf image or srec image in SDRAM or flash and loads the image to SDRAM.

Format

The format for this command is:

ld [-sbeS] addr load elf or srec

- s don't clear symbols
- b don't clear breakpoints
- e don't clear exception handlers
- S don't load symbols

Functional Description

The ld command accepts programs and data found in SDRAM or flash in Motorola S-record or ELF format and places the program in SDRAM ready for execution.

The ld command normally clears the symbol table, exceptions handlers, and all breakpoints. The value of the EPC register is set automatically to the entry point of the program. Therefore, to execute the program, only the g command is required.

See Also

load command, g command.

stty

The stty command displays and sets terminal options.

Format

The format for this command is:

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

where:

- 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, pmon 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, enter:

```
PMON> stty -a
```

To change the baud rate or terminal type, simply enter the new setting after stty.

Examples illustrating the use of this command follow.

```
PMON> stty
```

```
baud=57600
```

```
PMON> stty -a
```

```
baud=57600
```

```
istrip ixon -ixany -ixoff icanon echo echoe icrnl onlcr
```

```
erase=^H stop=^S start=^Q eol=^J eol2=^C vintr=^C
```

```
PMON> stty -v
```

```
Baud rates:
```

```
50 75 110 134 200 150 300 600
```

```
1200 1800 2400 4800 9600 19200 38400 57600 76800 115200
```

```
1 Terminal types:
```

```
tvi920 vt100
```

```
baud=57600
```

flush

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

Format

The format for the flush 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 performs a hard flush of the data and/or instruction cache. All entries will be flushed, even those that had been locked.

Files

f5000cfl.s

vers

The vers command prints the version number of pmon.

Format

The format for the vers command is:

```
vers [-a]
```

where:

-a List all version numbers

Functional Description

Entering vers without any parameters prints the primary version number of pmon. This is the value of the file 'version' in either the pmon directory. The -a option lists the version numbers of all the components that were used to build pmon.

Examples

```
PMON> version
6.5.0
PMON> vers -a
pmon:6.5.0 mon:5.3.8 lib:5.3.17
```

Command shell

The “command shell” is an embedded command that executes the pmon command typed following the prompt.

Functional Description

The following syntactic rules apply to all command lines entered at the pmon 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 pre fixed with an ampersand 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. the following table lists the commands that are supported by pmon.

Command	Action
^P	Recall previous command
^N	Recall next command
^F	Move cursor once 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
! <i>str</i>	Recall and execute the last command that started with the string <i>str</i>
! <i>num</i>	Recall and execute command number <i>num</i>
!!	Recall and execute last command
+-(/)	Algebraic operators
^ <i>addr</i>	Substitute with contents of address <i>addr</i>
@ <i>name</i>	Substitute with contents of named register
& <i>name</i>	Substitute with value of symbol <i>name</i>
0x <i>num</i>	Treat <i>num</i> as a hexadecimal number
0o <i>num</i>	Treat <i>num</i> as an octal number
0t <i>num</i>	Treat <i>num</i> as an decimal 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.

The following lists the rules that hold in setting the default numeric base.

Inbase	Base
0x	Hexadecimal
0t	Decimal
0o	Octal
[g-zA-Z@_.]	Symbol
&	Symbol
@	Register

inalpha - This variable selects whether arguments starting with a, b, c, d, e, or f are interpreted as symbols or as hexadecimal numbers.

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

Setting inalpha to "symbol" causes pmon 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
PMON> d map+0t10*4
PMON> d @a0+0t56
PMON> d ^tcbchn
```

Show the actual address.

Dump memory at (map+(10*4)).

Dump memory at 56(a0)

Dump memory at contents of tcbchn

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

hi (command history) and set (setup and display environment variables) commands.

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, hi, d, l, search, and ls.

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

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

The moresz Variable

moresz sets how many lines are displayed on one screen during screen-at-a-time output. If moresz is set to zero, the screen scrolls continuously. The ^S or ^Q control sequence must be used to pause the output, and the ^C 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

set command for the setup of the environment variables.