Origin200™ / 2000™ IP27 PROM Technical Reference Manual

Document Number 108-0170-001

Contributors

Written by Curt McDowell
Illustrated by Steve Whitney
Production by Cindy Stief
Engineering contributions by Uday Hegde

© Copyright 1997, Silicon Graphics, Inc.— All Rights Reserved

This document contains proprietary and confidential information of Silicon Graphics, Inc. The contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

Restricted Rights Legend

Use, duplication, or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and / or in similar or successor clauses in the FAR, or in the DOD or NASA FAR Supplement. Unpublished rights reserved under the Copyright Laws of the United States. Contractor / manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94043-1389.

Origin200™/2000™ IP27 PROM Technical Reference Manual
Document Number 108-0170-001

Silicon Graphics, Inc.
Mountain View, California

Silicon Graphics and the Silicon Graphics logo are registered trademarks of Silicon Graphics, Inc. Origin, Origin200, Origin2000, IRIX and IRISconsole are trademarks of Silicon Graphics, Inc. CrayLink is a trademark of Cray Research, Inc. R10000 is a trademark of MIPS Technologies, Inc.

Contents

Examples

Figures

Tables

Introduction

About This Document

This manual contains information needed to use the IP27PROM to boot or debug an Origin2000™ system. Coverage includes the following, but does not extend into the IO6PROM or IRIX™ Kernel:

- Module System Controller
- Multi-Module System Controller
- CrayLink™ Interconnect Topology Primer
- IP27PROMPROM Operation
- IP27PROM Command Set
- Debugging with the IP27PROM

A lot of basic information about the Origin2000™ is included to minimize the number of other documents required to use the IP27PROM. Information is presented in a reference format.

Intended Audience

This document contains no proprietary information and is suitable for use by Silicon Graphics Power Customers.

- Kernel developers
- IP27PROM and IO6PROM engineers
- Origin200™/2000™ hardware developers
- Manufacturing engineers
- Field engineers with need to know

Related Documents (Internal Access Only)

ASD Doc Depot (Engineering Access Only)

- Lego System Specification
- Hub Specification
- Hub Programming Manual
- CrayLink Interconnect Specification

Lego Firmware

Module System Controller (Engineering Access Only)

- Hardware Manual (PostScript)
- Firmware Manual (PostScript)

Multi-Module System Controller

- Command Language
- Network Addressing

FRU Analyzer (PostScript)

Serial Numbers

Global Customer Support

Chapter 1

Origin200/2000 System Overview

1.1 Slot Numbering

The two Router card slots in the front right of the machine are numbered R1 and R2, going left to right. Router cable ports are numbered 1, 2, and 3 from top to bottom. Router ports 4, 5, and 6 are internal to the system.

The four Node card slots in the back of the machine are numbered N1 through N4, going right to left. Each Node card contains two CPUs called A and B. Therefore, each module may contain up to 8 CPUs, which are called 1A, 1B, 2A, 2B, 3A, 3B, 4A, and 4B (or sometimes simply CPU 0 through CPU 7).

R1 is connected directly to N1 and N2, while R2 is connected directly to N3 and N4 (see Vector Addressing for a more thorough treatment of Router port connectivity).

IO1 through IO6 are connected directly to the Crossbows on N1 and N3, while IO7 through IO12 are connected directly to the Crossbows on N2 and N4.

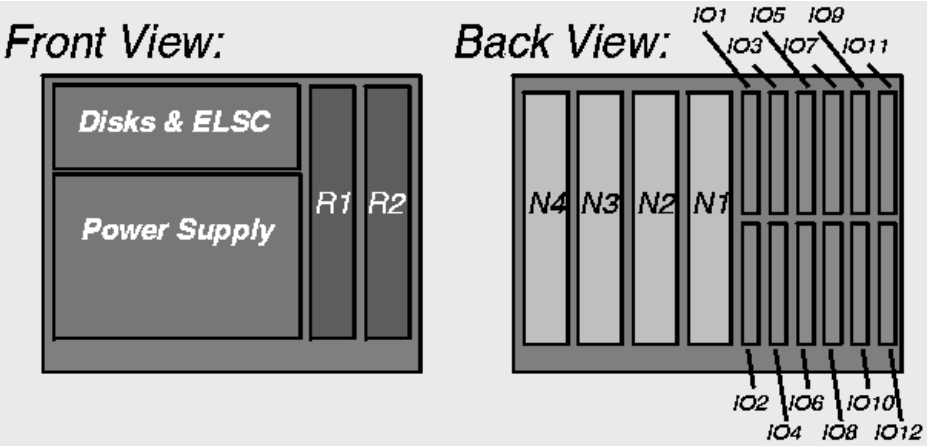

Figure 1-1 Slot Numbering

1.1.1 Part Numbers

Table 1-1 Common Part Numbers

Number	Part
013-1547	Origin2000 Midplane (8 CPU + 12 I/O)
013-1025	Origin200 Motherboard
030-0841	Origin2000 Router Card
030-0733	Origin2000 IP27 Node Card
030-1124	Server BaseIO Card
030-0872	MSCSI Card
030-0873	MENET Card
030-0880	MIO Card
030-0927	FibreChannel Card
030-0968	HIPPI_S Card
030-0956	HPCX Card
030-0948	ATM Card

1.2 Node IDs (NASIDs)

1.2.1 What is a Node ID?

During the boot process, the PROM assigns a Node ID to each Hub in the system. Node IDs are small integer values ranging from 0 to 255. Since there is one Hub on each node card, each Node card receives its own Node ID and the ID is shared between the CPUs on the same node card.

Node IDs are more correctly referred to as NASIDs (NUMA Address Space Identifiers). Hardware folks use the term Node ID, while software folks use the term NASID to avoid confusion with several other types of node identifiers used internal to the IRIX kernel.

1.2.2 What are Node IDs used for?

Node IDs allow any node to access the full address spaces of any other node. Addressing a remote node's space is done simply by placing the remote node's ID in bits 32 through 40 of the desired physical address. This is true whether talking to the node's memory, I/O devices, Hub chip, PROM, etc. For example, one could talk to various parts of remote node 7's address space using addresses as follows:

- **0x9000000700000000** - HSPEC (PROMs, back door directory memory)
- **0x9200000700000000** - I/O windows, including Hub and Bridge registers
- **0x9600000700000000** - Base of bank 0 memory, treated as uncached
- **0xa800000700000000** - Base of bank 0 memory, treated as cached

1.2.3 How do node IDs get assigned?

After a system reset, there is a period of time when no Node IDs are assigned. In this situation, all Node IDs are zero and it is not possible to access the address spaces of remote nodes.

A special Hub feature called Vector Routing provides a way to access a small subset of Hub registers on a remote node. It is through Vector Routing that the CrayLink Interconnect topology is discovered, a global master is arbitrated, node IDs are assigned and distributed to all nodes in the system, and the full address space access becomes defined.

After the Node IDs are assigned, routing tables in each Hub and Router in the system are programmed with information that allows nodes to find one another, so that a Node IDs is effectively used as a network address. The Node ID is used as an index into the routing tables at each network hop in order to determine to which router output port a request or response should be forwarded.

1.3 NICs (Number In a Can)

A tiny chip from Dallas Semiconductor called a NIC is used extensively throughout the Origin2000 system. There is a NIC on each type of board in the system, including the Node card, Router card, BaseIO card (with two NICs), module midplane, etc.

The NIC contains a 48-bit number that is permanently laser-burned in at the time the NIC is manufactured and is guaranteed to be unique from all other NICs. The Node card NICs help identify Nodes during the boot process when the system is probing the network. The NIC on the midplane is used for software licensing.

In addition to the 48-bit number, NICs also contain several pages of non-volatile memory that can be read or written by the system. This memory is used to store additional information, primarily for purposes of inventory tracking, such as:

- Board type
- Board revision number
- Board serial number (as it appears on the bar code)
- Board-specific information (e.g., on the BaseIO card, the Ethernet MAC address)

In general, Origin2000 software never writes to the NIC. Rather, it is programmed once by an external NIC programming device in the manufacturing plant, and subsequently reprogrammed by Manufacturing should a board be upgraded to a new revision level.

1.4 Memory Configurations

The Origin™ family memory consists of two parts, the system memory and the directory memory. Directory memory is used for storing cache coherence protocol information and is much smaller than system memory.

The Origin2000 supports from 64 MB to 4 GB of RAM per node card. Two types of directory memory are available for the Origin2000, referred to as standard and premium. Premium directory memory is required for systems that are not a subset of a standard 32-processor hypercube. They can also be used in smaller systems to provide greater process migration accuracy.

System memory must be populated one bank at a time. Each bank consists of two memory DIMM slots and one directory DIMM slot (be careful, because the three slots are separated from one another on the node card). The two memory DIMMs making up one bank must contain the same size DIMMs. Each bank may contain a different amount of memory. There should be memory in bank zero.

Standard directory memory resides in the same DIMMs in which the system memory resides, and is automatically populated when the system memory is populated. The directory DIMM slots are left empty.

Premium directory memory is populated by inserting additional smaller DIMMs into the directory DIMM slots, adding more precision to the standard directory memory.

The Origin200 supports from 32 MB to 4 GB of RAM per node card and does not support premium directory memory. Banks are populated from two outermost DIMM slots (making up bank 0) to the two innermost DIMM slots (making up bank 3).

1.5 Module Numbers

In an Origin2000 system, each Module (4 node cards, 8 processors) is assigned a unique number from 1 to 255, called the module number. The module number is the mechanism by which IRIX identifies particular modules. The module number is stored in MSC NVRAM. A backup copy of the module number is stored in each node card PROM Log, so that in the event that the MSC NVRAM becomes inaccessible, the module number is voted from the last value stored in the PROM Logs. A module number of 0 indicates that a module number has not yet been assigned.

The most important use of module numbers is device naming in the Irix Hardware Graph. Disks and other devices are referred to by the number of the module containing them, as well as the slot within that module. For example, the root disk device might be called

/hw/module/1/slot/io1/baseio/pci/0/scsi_ctlr/0/target/1/lun/0/disk/partition/0/block

indicating that the disk is in module 1, slot io1, BaseIO card, PCI slot 0, etc.

If a module number is changed, all affected devices in the hardware graph will change names. This would most likely require the system administrator to reconfigure filesystem mounts, among other things.

Maintenance of module numbers may be performed using the IP27PROM **module** command, or using the MSC **mod** command. When module numbers are changed in this manner, the change will not take place until the next time the system is rebooted.

The IO6PROM verifies the consistency of module numbers. To be consistent, every module must have a number assigned to it, and there must not be any duplicate numbers.

Modules found to be without numbers will have a number automatically assigned to them, and then the system will reboot. Therefore, if a new module is added to an existing system, a module number will be automatically assigned and any attached devices will show up in the hardware graph.

Duplicate module numbers will prevent the system from booting until the problem is rectified. A message indicating the problem will display on the system console.

1.6 Module System Controller

The MSC front panel is shown in Figure 1-2 and comprises the following elements:

- Reset momentary push-button switch, with associated LED
- NMI momentary push-button switch, with associated LED
- Eight-character alphanumeric dot matrix LED display
- Three-position rotary keyswitch (Off, On, Diagnostics)
- DIN-type RS-232 serial port connector (diagnostics port)
- A block of eight small DIP switches (called the Debug Switches)

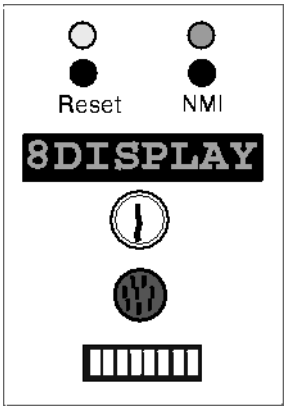

Figure 1-2 MSC front panel

Note: The MSC was at one time called the Entry Level System Controller, or ELSC. They are one and the same.

1.6.1 Alternate (Diagnostic) Console Port

The Origin2000 Module System Controller (MSC) front panel has a DIN-type RS-232 serial port, labelled the Diagnostic Port, and internally referred to as the Alternate Console Port (ACP). A second connector for the same serial port is provided in the back of the module for when it is to be connected to a Multi-Module System Controller (MMSC). The ACP is always available and is primarily used for debugging during manufacturing bring-up of a system, as well as system debugging when for some reason the regular serial port console is not available.

An RS-232 dumb terminal connected to the ACP can talk to the MSC, and through the MSC to the individual CPUs. All of the CPUs in a single module must share this console. For this reason, output from multiple CPUs appears on the console simultaneously, interleaving on a line by line basis, with CPU identification at the start of each line in the form of a slot (1 to 4) and slice (A or B). The MSC can be directed to send ACP keyboard input to a particular CPU. It can also be directed to show only the output from a particular CPU.

The MSC itself has a command set. Commands are sent to the MSC by typing the escape character ^T (Control-T), followed by the text of the command and `ENTER`. Ordinary ACP output is discarded between the time that the ^T and the `ENTER` are received. If echoing is enabled (default), the prompt `MSC>` will be displayed upon the reception of the ^T and characters typed will be visible.If echoing is not enabled (see the **ech** command), nothing will be visible as the command is typed.

1.6.2 Security

MSC commands which could potentially be destructive to system operation may only be executed when the MSC is in supervisor mode. If the MSC is not in supervisor mode, these commands result in the following error:

```
err perm
```

The MSC is automatically in supervisor mode when the front panel keyswitch is in the Diag position. It may also be placed in supervisor mode by issuing the MSC command *pas none* to enter the four-character MSC password, where **none** is the default MSC password. The command **pas s abcd** would change the MSC password to **abcd**.

1.6.3 Commands

Commands are typed to the MSC ACP by prefixing them with a ^T (Control-T) character. Commands are visible as they are typed only if echoing is turned on (which is true after power-on). Each command is three letters in length. Some of them take parameters. Numeric parameters are always in hexadecimal. All commands return responses consisting of ok and possibly some hexadecimal values, or one of the three error responses shown in Table 1-2.

Table 1-2 MSC Responses

Response	Reason
err perm	Permission denied. Keyswitch must be in diagnostic position, or password entered via the **pas** command
err cmd	Unrecognized command mnemonic
err arg	Invalid command argument(s)

The supported commands are listed in Table 1-3.

Table 1-3 MSC Commands

Response	Reason
aut 1	Turns on automatic power-on mode so the MSC automatically issues a **pwr u** when the MSC is powered on (Origin200 only).
aut 0	Turns off automatic power-on mode (Origin200 only).
clr	Resets all MSC options to their power-up defaults. This includes echo mode, no heartbeats, etc.
dbg	Displays the virtual and physical Debug Switch bytes.
dbg *V P*	Sets the virtual Debug Switch byte to V, and the physical Debug Switch byte to P.
dsp *M*	Displays message *M* on the 8-character alphanumeric display. M may contain up to 8 ASCII characters other than NUL. For example, **dsp testing** would overwrite the first seven characters with “testing,” while leaving the eighth character alone
dsc *N C*	Modifies only the Nth character on the alphanumeric display. *N* is a digit from 0 to 7, and *C* is any ASCII character other than NUL.
ech 0	Turns off echoing as MSC commands are entered.
ech 1	Turns on echoing.
ech	Toggles the echoing. Echoing is on by default after reset.
fan	If no fan has failed, returns **n** or **h**, according to whether the fans are currently operating at Normal or High speed. If a fan has failed on an Origin200, returns **f x**, where x is a bit map of the failed fans (fan 1 = 1, fan 2 = 2, fan 3= 4). If a fan has failed on an Origin2000, returns **f xyz**, where xyz are bit maps of the failed fans by row (fan 1 = 1, fan 2 = 2, fan 3= 4). When the MSC detects that a fan has failed, it speeds up the remaining fans to maintain cooling levels. The failed fan should be replaced because the increased speed reduces the life of the remaining fans. If more than one fan fails, the system is powered down.
fan n	Sets the fans to normal speed.
fan h	Sets the fans to high speed.
key	Returns the status of the key-switch as **off**, **on**, or **diag**.

Table 1-3 (continued) MSC Commands

Response	Reason
mod	Displays the number of the module containing the MSC.
mod *xx*	Sets the number of the module containing the MSC to *xx*, where *xx* is a hexadecimal module number from 01 to ff (a module number of 00 indicates no module number is yet assigned).
nmi	Sends a hardware NMI to all node cards in the MSCs module.
pas *xxx*	When *xxxx* is replaced with the correct four-character password, places the MSC into supervisor mode where various restricted commands may be used. If the MSC is not in supervisor mode, restricted commands will result in a permission error (**err perm**). The MSC is automatically in supervisor mode if the key-switch is in the diag position.
pas s *xxxx*	Set the password to *xxxx*. This is a restricted command. The password is stored in NVRAM and defaults to **none**.
pwr	Returns **u** or **d**, according to whether the system is powered up or down.
pwr u	Powers the system up.
pwr d	Powers the system down.
pwr d *N*	Waits N seconds and then powers the system down. N is a hex value from 5 to 258 (5 to 600 decimal).
pwr c *N*	Powers the system down, waits N seconds, then powers the system back up. *N* is a hex value from 5 to 258 (5 to 600 decimal).
rst	Sends a hardware reset to all node cards in the MSCs module.
rsw	Returns the current Debug Switch settings as an inverted hexadecimal byte. See section on Debug Switch Use for bit correspondence.
sel *cpu*	Selects which CPU is to receive input from the ACP. Anything typed on the ACP which is not an escaped command is sent through to the selected CPU. CPUs are named by slot and slice as described in Slot and CPU Numbering. For example, **sel 2a** would select CPU A in slot N2 for input.
sel	Displays which CPU is currently selected to receive ACP input, or none if no CPU is selected.
sel auto	Causes the last CPU to have output anything to be automatically selected to receive ACP input (this is the power-up default).
sel none	Causes no CPU to be selected to receive ACP input.
see *cpu*	Causes output from all CPUs other than a specific CPU to be discarded (ordinarily, the output from all CPUs is displayed intermixed by line). CPUs are named by slot and slice as described in Slot and CPU Numbering. For example, **see 2a** would cause only CPU 2A's output to be shown.
see	Displays which CPU is currently being shown, or **all** if all CPUs are being shown.
see all	Causes output from all CPUs to be displayed (this is the power-up default).

Table 1-3 (continued) MSC Commands

Response	Reason
tmp	Returns **o**, **h**, or **n**, indicating whether the system is over-temperature (automatic shutdown pending), high operating temperature, or normal operating temperature, respectively.
ver	Reports the MSC firmware revision number.

The following MSC commands are not documented here: **get**, **hbt**, **rcf**, **scf**, **tas**, and **vlm**.

Chapter 2

Multi-Module System Controller

The Multi-Module System Controller provides a way to manage Origin2000 systems consisting of more than one module, and also provides an enhanced graphical display of system activity. An MMSC is specified in the standard configuration for Origin2000 systems consisting of more than one module.

Multi-module systems can be operated without an MMSC. However, certain aspects of system management are less convenient. Moreover, if no MMSC is provided, the individual modules in the system must be manually powered on one at a time, with each switch being powered on less than 15 seconds after the previous one.

For detailed information about MMSC operations and command set, please refer to the Multi-Module System Controller document.

Note: The MMSC was at one time called the Full-Featured System Controller, or FFSC. They are one and the same.

2.1 Functions of the MMSC

The Multi-Module System Controller serves the following purposes:

- Simplifies partition management
- Manages a system from a single point of control (Important for remote modem control of systems)
  - Access to individual MSC command features
  - Access to individual MSC console I/O features
  - Powering up or down all modules in a system
  - Resetting all modules or one partition
  - Sending an NMI to all modules in a partition
- Enhanced graphical display of system activity
  - High resolution color display
- Retrieving information about the system while it is powered down

2.2 MMSC Connectivity

A typical multi-module installation consists of:

- Modules containing from 1 to 4 Node cards each
- One MSC per module, internally connected to each Node card
- One or two modules per rack
- One MMSC per rack, connected via 9600 baud serial port to both MSCs
- One designated master MMSC with a graphical display and input device
- One 115 kbaud serial port from each MMSC to one module's BaseIO console port
- An ethernet network connecting all MMSCs together
- An optional IRISconsole™ workstation (usually large systems only)

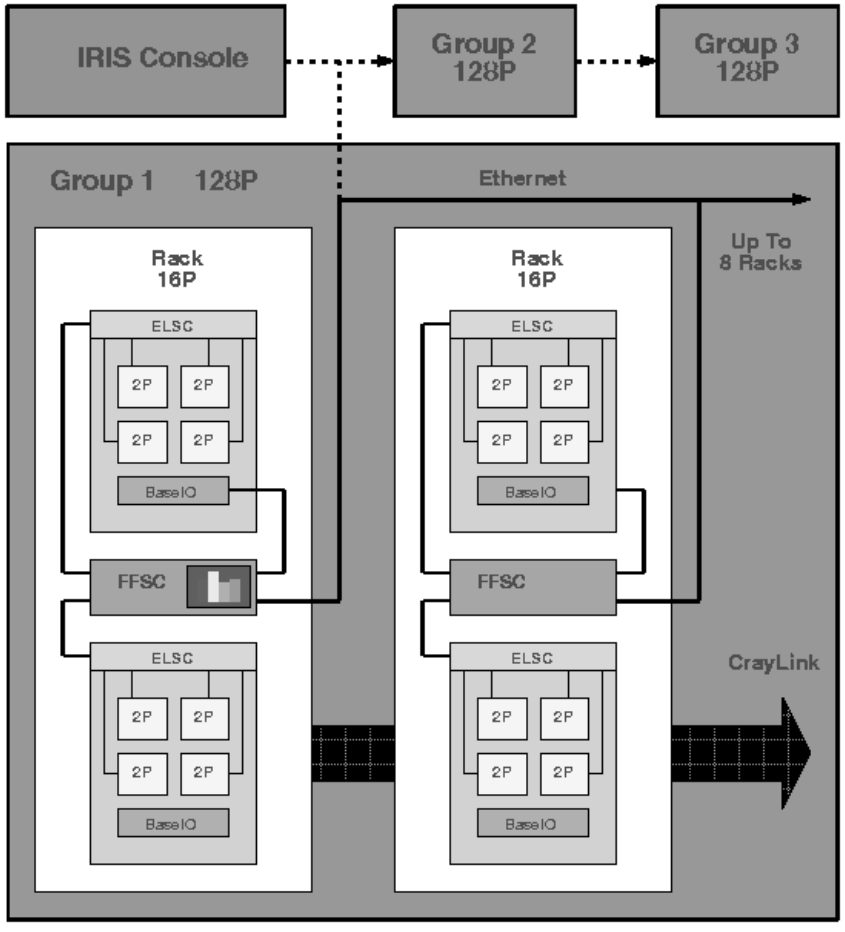

Figure 2-1 Origin2000 Hierarchical System Control

2.3 IP27PROM

There is one IP27PROM per node card, connected directly to the Hub chip. The device is physically an AMDF080 containing 1,048,576 bytes. It is possible to erase individual sectors consisting of 65,536 bytes, a process which sets all of the bits in the sector to 1. It is then possible to program any individual 1 bit into a 0 bit, but not vice versa.

The first 14 sectors of the PROM are used for the IP27PROM firmware. The last 2 sectors are used for the PROM Log which stores environment variables and log messages.

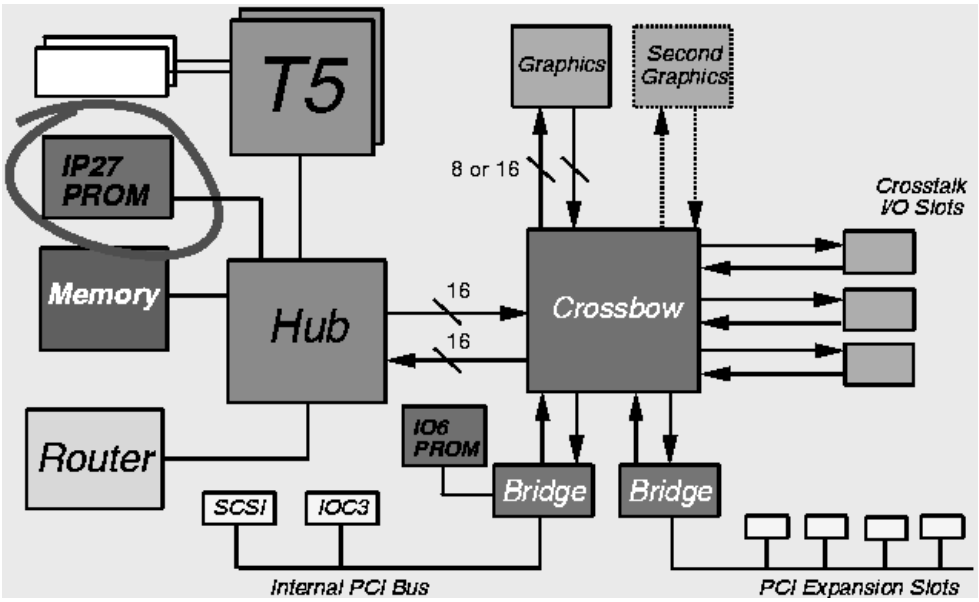

Figure 2-2 IP27PROM Fits Into the Origin2000 System

2.3.1 PROM Compatibility

It is illegal to run different versions of the IP27PROM on different nodes because the details of the boot sequence vary from release to release. Each node relies on the implicit actions of other nodes in order to maintain proper synchronization. The IP27PROM cross-checks version numbers with other IP27PROMs and displays a PROM revisionPROM mismatch warning if there is an incompatibility.

There are also restrictions on which versions of the IP27PROM may run with which versions of the IO6PROM, although they do not necessarily need to be synchronized on every release. Refer to the individual PROM release nodes for compatibility information.

2.3.1.1 System Boot Process

When the system is powered on or reset, the following processes take place.

1. Initialize CPU
   - Set up R10000™ status registers
   - Clear register files
2. Test CPU caches
   - Instruction cache
   - Data cache
3. Set up Dex mode
   - Begin treating the CPU data cache as memory
   - Initialize the stack
   - Execute the boot procedure (written in C)
4. Disable CPU A and / or CPU B
   - The DisableA and DisableB environment variables specify which CPUs to disable.
   - In emergencies, disabling can be overridden using Virtual Debug Switch 10.
   - CPUs are completely separated from the Hub by turning off PI_CPU_ENABLE.
5. Arbitrate local master (one per node card)
   - Check for presence of other CPU, if any
   - Time out if other CPU does not respond
   - Disable other CPU if not responding
6. Read Debug Switch settings from the MSC
7. Determine the initial console device, and initialize it
   - The MSC diagnostics serial port is normally used, but
   - If there is a Junk UART plugged in, it takes precedence
   - At this point, detailed boot status information can be printed
8. Record error information that may indicate the cause of a prior crash
   - Save state of Hub error registers in the cache
   - Clear state of Hub error registers
   - Enable SYSAD error checking
9. Initialize I / O
   - II section of the Hub
   - Bridge
   - PCI bus
10. Check if there is a BaseIO card with a console port
    - If so, switch the console to the BaseIO

11. Display the PROM boot banner on the console
    - IP27PROM version and size
    - Chip revisions
    - Slot ID and CPUs present
    - Other miscellaneous information
12. Display which Debug Switch settings are set to other than the default
13. Determine node's serial number and advertise it to other nodes
14. Run Hub Chip Self-Test if Heavy or Manufacturing diagnostics are selected
    - Runs the BIST (built-in self test) procedure, which automatically reboots the system
    - On reboot, if the test failed, stops with a failure LED value
15. Configure local memory
    - Initialize SIMM control registers
    - Probe amount and type (premium or standard) of memory
    - Program Hub memory configuration registers
16. Perform basic memory tests
    - Make sure address 0 of each bank is accessible
    - Disables banks that aren't accessible
    - If bank 0 is bad, swaps it with a good bank (see SwapBank environment variable)
17. Download PROM to memory
    - Perform memory test on small PROM area of memory
    - Download a copy of the PROM
    - Verify the download checksum
18. Transfer program counter to uncached RAM
    - Runs much, much faster than running out of the PROM
    - Still does not require the secondary cache
19. Switch crucial structures from the cache into uncached memory
20. Test and invalidate the secondary cache
21. Transfer the stack to uncached RAM
    - Discard the Dex contents of the data cache
22. Transfer the program counter and stack to cached RAM
    - Runs at top speed
23. Initialize the first 32 MB of bank 0 memory for use by the PROM

24. Initialize permanent low-memory system data structures
    - These structures persist across the IRIX kernel
    - **KLDIR** - Indirection table for accessing all other low-memory structures
    - **NMI** - Non-maskable Interrupt vector area
    - **KLCONFIG** - System topology and configuration information
25. Run diagnostics on the local CrayLink port
26. Discover the CrayLink Interconnect Topology
    - Depth-first search using Vector Routing operations
    - Builds **promcfg** data structure in PROM memory
27. Verify all PROMs are running the same firmware version
28. Arbitrate global master
    - Protocol uses Vector Routing operations
    - One global master per partition
29. Global Master configures CrayLink Interconnect
    - Determines Node ID for each node (not yet assigned)
    - Programs CrayLink routing tables in all Hubs and Routers
    - Tells each Node what their ID will be
30. All nodes switch back to uncached memory
    - Required for changing Node ID
    - Flush caches
    - Closely synchronize all nodes
    - Quiesce bus activity with tight loops internal to R10000
31. Change Node ID
    - Update data structures that were assuming Node ID was zero
    - Re-initialize coherence directories for first 32 MB of memory
32. Switch back to cached memory in new Node ID space
33. Test and initialize all of the rest of local memory (above 32 MB)
34. Initialize any headless nodes (nodes without functional CPUs)
    - Disable CPUs
    - Read NIC
    - Probe, configure, and initialize memory
    - Initialize low-memory data structures
    - Initialize I/O
35. Display error state information (if not cold power-on)
    - Hub error registers, decoded in detail

36. Transfer control to IO6PROM
    - May be inhibited by Debug Switch setting
    - Decompress IO6PROM image into memory from master BaseIO card
    - If no master BaseIO, decompress copy internal to IP27PROM
    - Jump to entry point of IO6PROM

As it boots, the IO6PROM displays several messages about booting and probing devices, then proceeds to the IO6PROM menu:

```
Numbering nodes...
2 node(s) found.
Clocks synchronized.
Modules numbered.
IO6 PROM Monitor SGI Version 2.1 Rev A IP27,   Sep 17, 1996 (BE64)
Sizing caches...
Sizing caches...
Initializing exception vectors.
Initializing environment
Initing environment
Initializing software and devices.
Reiniting caches..
Initing saio...
Installing Devices...

Walking SCSI Adapter 0
1- 2- 3- 4- 5- 6- 7- 8- 9- 10- 11- 12- 13- 14- 15- = 0 device(s)

Walking SCSI Adapter 1
1- 2+ 3- 4- 5- 6- 7- 8- 9- 10- 11- 12- 13- 14- 15- = 1 device(s)
Initializing devices...

System Maintenance Menu

1) Start System
2) Install System Software
3) Run Diagnostics
4) Recover System
5) Enter Command Monitor

Option?
```

2.3.2 Using the System Controller Debug Switches

There are two sets of Debug Switches maintained in NVRAM by the MSC:

- Eight Physical Debug Switches, numbered 1 through 8
- Eight Virtual Debug Switches, numbered 9 through 16

These switches are set by the MSC **dbg xx yy** command, where **xx** and **yy** are hexadecimal bytes. The Virtual Debug Switches are set to **xx** and the Physical Debug Switches are set to **yy**. The most significant bit of **xx** corresponds to Debug Switch 16, while the least significant bit of **yy** corresponds to Debug Switch 1. Using the **dbg** command without arguments displays what the current settings are. Ordinarily, both sets of Debug Switches should be set to zero. An equivalent **dbg** command is also available in IP27PROM POD mode (but be careful because it takes decimal by default).

There are also eight Hardware Debug Switches that directly correspond to the Physical Debug Switches. The Hardware Debug Switches are exclusive-ORed with the Physical Debug Switches so debug functions can be controlled via both MSC serial port commands and MSC Hardware Debug Switches. The exclusive-OR allows Hardware Debug Switches that are on to be turned off remotely, and switches that are off to be turned on. The Origin2000 Hardware Debug Switches are mounted on a blue block below the MSC keyswitch.

The Origin200 also has Hardware Debug Switches which accessible by removing a small EMI panel cover on the top of the chassis, and are slightly different than the Origin2000 switches in appearance (however, they are still `OFF` when switched away from the labelled numbers).

Examples:

- To turn off all boot diagnostics: **dbg 0 1**
  - If Hardware Debug Switch 1 is already on, this will turn on boot diagnostics.
- To stop at Memoryless POD mode on reset: **dbg 0 18**
  - The 4th and 5th bits in the hex value 0x18 are one, so this is the same as turning on Hardware Debug Switches 4 and 5.
- To override CPU disables: **dbg 1 0**
  - This turns on Virtual Debug Switch 9 (not available as a Hardware Debug Switch).

All switches should be `OFF` for normal system operation. Changing hardware debug switch settings requires using a sharp stylus to press the switch in on the top (switch `ON`) or bottom (switch `OFF`). The Debug Switch block shown here in Figure 2-3 has switches 1, 2, and 6 set to `ON` and all others `OFF`.

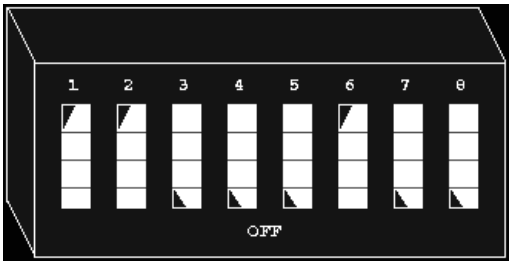

Figure 2-3 Hardware Debug Switch settings

When reading the raw binary value of the Hardware Debug Switches using the **rsw** command, a hexadecimal byte value is returned. The most significant bit is switch 8, and a bit is 1 when its switch is OFF. The bit values are reversed (1 when off, 0 when on). Reading the Hardware Debug Switch block shown in Figure 2-3 would return 0xdc.

2.3.3 System Controller Debug Switch Assignments

Switches 1 and 2 (see Table 2-1), select the kind of diagnostics that are run after a system reset before booting IRIX. Switches 1 and 2 apply only to the nodes in the module on which they're set.

Table 2-1 Dip Switches 1 and 2

1	2	Diagnostics Level
OFF	OFF	Normal
OFF	ON	Heavy
ON	OFF	None
ON	ON	Manufacturing

Normal — Tests each part of the system for basic functionality, only using relatively fast tests to expedite system boot while catching any blatantly troubled hardware.

Heavy — Runs the most thorough diagnostics available on each part of the system. They may take a very long time to complete, especially the memory tests. It may be desirable to run them after installing new hardware or if the system is having problems thought to be hardware-related.

Manufacturing — Runs heavy diagnostics and outputs special FRU (field replaceable unit) information. Console input and output are handled through the system controller port which must be connected to Silicon Graphics manufacturing equipment.

None — Performs no diagnostics and the system will boot as fast as possible. This might be used when debugging software such as kernel drivers, when there is complete confidence in the hardware.

Table 2-2 Dip Switch 3

3	Information Level

If Switch 3 is On, the PROM shows very detailed information messages during boot, interspersed with the normal boot status messages. The switch applies only to the nodes in the module on which it's set.

Table 2-3 Dip Switches 4 and 5

4	5	Boot Stop Point
OFF	OFF	Never
OFF	ON	Local
ON	OFF	Global
ON	ON	Memoryless

Switches 4 and 5 allow the boot process to be stopped at various stages, (see Table 2-3).

Never — Allows the boot to proceed all the way through to IRIX (default).

Local — allows the boot to proceed up to the point where it would normally load and jump to the IO6PROM. Instead of continuing, all nodes enter cached (Cac) POD Mode. If this switch is set on any module, it will be propagated to all modules.

Global — Allows the boot to proceed up to the point where it would normally load and jump to the IO6PROM. Instead of continuing, the master node enters cached (Cac) POD Mode and all of the slaves enter the Slave Loop. If this switch is set on any module, it will be propagated to all modules.

Memoryless — Stops as soon as possible after setting up just the bare minimum portion of the system required to enter POD mode. All nodes enter dirty exclusive (Dex) POD Mode even if there is no local memory.

Caution: If this switch is set on in one module, the system containing the module will not boot properly.

Table 2-4 Dip Switch 6

	**Default Environment**

If Switch 6 is On, the PROM ignores all PROM Log environment variables and IO6 NVRAM settings, and uses the system defaults. This may be useful for proceeding if any of the variable storage mechanisms contain data that is preventing the system from booting. This switch applies only to the module on which it is set.

Table 2-5 Dip Switch 7

	**Bypass IO6**

If Switch 7 is On, the PROM bypasses the first IO6 card that is found and tries to boot from the second one found. This may help to boot the system if the first IO6 card is not working, without having to physically remove the card. This switch applies only to the module on which it is set.

Table 2-6 Dip Switch 8

	**Bypass Global Master**

If Switch 8 is On, the node that would ordinarily become the global master will become a slave, and the next CPU in line will become the global master. This switch applies only to the module on which it is set.

Table 2-7 Dip Switch 9

	**Override CPU Disabling**

If Switch 9 is On, the all CPUs that would otherwise be disabled due to the DisableA or DisableB environment variables being set (see POD mode disable command) will no longer be disabled after a reset.

This is useful for getting out of the situation in which all CPUs in the system have accidentally been disabled simultaneously.

Table 2-8 Dip Switch 10

	**Override System Partitioning**

If Switch 10 is On, the all CPUs that would otherwise be disabled due to the DisableA or DisableB environment variables being set (see POD mode disable command) will no longer be disabled after a reset.

This is useful for getting out of the situation in which all CPUs in the system have accidentally been disabled simultaneously.

Table 2-9 Dip Switch 11

	**Use Default Console**

If no user-defined console can be located by means of the ConsolePath environment variable in the IO6PROM, and Switch 11 is turned On, then the first serial device found in each module will be treated as a console device. The module containing the Global Master CPU will become the overall system console.

Table 2-10 Dip Switch 12

12	**Router Oven Mode (manufacturing only)**

Switch 12 is a special function used by Silicon Graphics manufacturing that allows systems to boot part way even though some router(s) may have more than one express link. This is normally an invalid configuration, but is nevertheless used by Silicon Graphics to run router tests on all router ports in sparsely populated test fixtures.

Table 2-11 Dip Switch 13

13	**Show Error State**

Switch 13 causes the complete Hub chip error state to be dumped at system boot time. The state is still not dumped if the system was just powered on, since the error state at power-on is random. Sometimes, if a system crashes the Hub error state after reset is useful to developers. It is not displayed by default because there are often extraneous errors that would alarm users.

Table 2-12 Dip Switch 14

14	**Ignore Autoboot**

If Switch 14 is set, the IO6PROM will ignore the **autoboot** environment variable and go to the 5-item PROM menu. This is an alternative to pressing ESC on the console when the system says “Starting up the system.”

Table 2-13 Dip Switches 15 and 16

15	**Through**	16	**Not yet assigned**

Switches 15 and 16 are reserved for future use.

2.4 PROM Images

Note that this section might be of interest to developers only.

The IP27PROM build directory and the IRIX flash commands deal with PROM images in the **promgen** format. The file extension for such images is *.img*. Under IRIX, they reside in the directory */usr/cpu/firmware*.

To verify an image and view the version number of an image, use the command **promgen -h** *file***.img**. The promgen utility resides in *stand/arcs/tools/promgen*.

Chapter 3

Power-On Diagnostics

3.1 Boot Status LEDs

During system boot, the node board LEDs are constantly updated with values indicating the boot progress, so that if the system were to crash during any phase, the LEDs would indicate what it was doing at the time. Also, diagnostics values are displayed on the Origin2000/200 Module System Controller display during boot. Further into the boot process, a console becomes available to report more detailed information on failures.

If a single processor fails very early during boot, before a console is available, the PROM will present a non-flashing FLED (failure LED) value and completely disable that processor by setting its PI_CPU_ENABLE bit to zero. The system will continue to boot without that processor.

If a single process fails after a console is available, the PROM will flash a FLED value, and wait for ^C to be entered on the console, whereupon it will enter Dex POD mode. The system will continue to boot without that processor.

3.2 Reading the LEDs

Each node card has two side-by-side columns of 8 discreet LEDs (see Figure 3-1). The left column presents a status value from CPU (slice) A, and the right column presents a status value from CPU B.

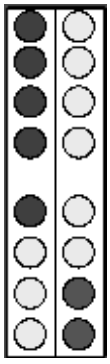

Figure 3-1 LED Values

During the boot process, the CPU changes the LEDs before each phase of initialization. If a CPU were to hang during any phase, the residual LED value would help to indicate which phase hung and perhaps pinpoint the failing component (for example, the R10000™ data cache). LED values from 0x00 to 0x7f, are used for this purpose, as shown in Table 3-1.

Table 3-1 Progress LED Values

LED	Name	Phase
0x00	RESET	-
0x01	INITCPU	Initializing R10000 GPRS, FPRS, and COP0
0x02	TESTCP1	Testing R10000 COP1 registers
0x03	RUNTLB	Switch to mapped mode
0x04	TESTICACHE	Test R10000 primary instruction cache
0x05	TESTDCACH	Test R10000 primary data cache
0x06	TESTSCACH	Test secondary cache
0x07	FLUSHCACHES	Flush all caches
0x08	CKHUBLOCAL	-
0x09	CKHUBCONFIG	-
0x0a	INVICACHE	Invalidate R10000 primary instruction cache
0x0b	INVDCACHE	Invalidate R10000 primary data cache
0x0c	INVSCACHE	Invalidate secondary cache
0x0d	INMAIN	Succeeded in jumping to main()
0x0e	SPEEDUP	About to increase PROM access speed
0x0f	SPEEDUPOK	Increased PROM access speed
0x10	INITDCACHE	-
0x11	INITICACHE	-
0x12	INITCOP0	-
0x13	FLUSHTLB	-
0x14	CLEARTAGS	-
0x15	CCLFAILED_INITUART	-
0x16	HUBINIT	-
0x17	HUBCFAILED_INITUART	-
0x18	NOCLOCK_INITUART	-
0x19	HUBINITDONE	-
0x1a	MSCPROBE	About to probe for presence of MSC
0x1b	JUNKPROBE	About to probe for presence of Junk UART

Table 3-1 (continued) Progress LED Values

LED	Name	Phase
0x1c	DONEPROBE	Done probing for presence of MSC
0x1d	UARTINIT	About to initialize selected UART
0x1e	UARTINITDONE	Done initializing selected UART
0x1f	CKHUBCHIP	-
0x20	PODMAIN	-
0x21	PODLOOP	About to enter POD mode, C portion
0x22	PODPROMPT	Just about to enter POD prompt loop
0x23	PODMODE	About to enter POD mode, assembler portion
0x24	LOCALARB	Performing local arbitration (CPU A/B)
0x25	SCINIT	-
0x26	BMARB	-
0x27	BMASTER	-
0x28	BARRIER	About to perform first local barrier
0x29	CKPDCACHE1	-
0x2a	MAKESTACK	About to configure Dex mode stack and data
0x2b	MAIN	Reached main()
0x2c	LOADPROM	-
0x2d	CKSCACHE1	-
0x2e	CKBT	-
0x2f	INSLAVE	-
0x30	PROMJUMP	-
0x31	NMI	-
0x32	INV_IDCACHES	-
0x33	INV_SCACHE	-
0x34	WRCONFIG	-
0x35	RTCINIT	About to initialize Hub Real Time Counter
0x36	RTCINITDONE	Done initializing Hub Real Time Counter
0x37	LOCK	-
0x38	BARRIEROK	First local barrier succeeded
0x39	LOCKOK	-
0x3a	FPROMINIT	-

Table 3-1 (continued) Progress LED Values

LED	Name	Phase
0x3b	FPROMINITDONE	-
0x3c	JUMPRAMU	About to jump to UALIAS space
0x3d	JUMPRAMUOK	Jumped to UALIAS space
0x3e	JUMPRAMC	About to jump to cached space
0x3f	JUMPRAMCOK	Jumped to cached space
0x40	STACKRAM	About to test stack area of memory
0x41	STACKRAMOK	Done testing stack area of memory
0x42	SLAVEINT	Slave saw command request interrupt
0x43	SLAVECALL	Slave about to call requested command
0x44	SLAVEREND	Slave command completed
0x45	LAUNCHLOOP	About to enter slave launch loop
0x46	LAUNCHINTR	Received launch interrupt
0x47	LAUNCHCALL	Calling launched function
0x48	LAUNCHDONE	Launched function returned
0x49	UARTBASE	-
0x4a	MDIRINIT	About to initialize Hub MD and SIMM controls
0x4b	MDIRCONFIG	About to probe and configure memory size
0x4c	I2CINIT	About to initialize PCF8584 I2C chip
0x4d	I2CDONE	Done initializing PCF8584 I2C chip
0x4e	CONFIG_INIT	-
0x4f	IODISCOVER	About to discover Hub I/O
0x50	HUB_CONFIG	-
0x51	ROUTER_CONFIG	About to write Router cfg info into KLCONFIG
0x52	INITII	About to initialize I/O section of Hub
0x53	CONSOLE_GET	About to probe I/O section for console
0x54	CONSOLE_GET_OK	Console probing completed
0x55	NOT_USED_55	-
0x56	INITIODONE	Done initializing I/O section of Hub
0x57	STASH2	Reset error state saved
0x58	STASH3	Hub error registers cleared
0x59	STASH4	Hub error checking enabled

Table 3-1 (continued) Progress LED Values

LED	Name	Phase
0x5a	IODISCOVER_DONE	Done discovering Hub I/O
0x5b	NMI_INIT	About to initialize NMI handler area
0x5c	TEST_INTS	About to test Hub interrupts
0x5d	IORESET	About to perform early reset of Hub I/O section

In addition to indicating boot progress, the LEDs are used to indicate fatal hardware problems found during diagnostics the PROM performs in each boot phase. If a fatal problem is found, the CPU sets the LEDs to a failure value between 0x80 and 0xff, as shown in Table 3-2, and automatically disables itself.

Table 3-2 Failure LED Values

LED	Name	Reason
0x81	CP1	R10000 COP1 register test failed
0x82	RESTART	Restart Master unable to load io6PROM
0x83	ICACHE	R10000 primary instruction cache test failed
0x84	DCACHE	R10000 primary data cache test failed
0x85	SCACHE	Secondary cache test failed
0x86	KILLED	CPU disabled by another node
0x87	RTC	Real-time counter not counting
0x88	ECC	-
0x89	XTLBMISS	-
0x8a	UTLBMISS	-
0x8b	KTLBMISS	-
0x8c	GENERAL	-
0x8d	NOTIMPL	-
0x8e	CACHE	-
0x8f	OS	-
0x90	HUBINTS	-
0x91	HUBLOCAL	-
0x92	HUBCONFIG	-
0x93	PREM_DIR_REQ	All nodes must have premium DIMMs for this configuration
0x94	UNUSED1	-
0x95	HUBUART	-

Table 3-2 (continued) Failure LED Values

LED	Name	Reason
0x96	HUBCCS	-
0x97	MAINRET	main() returned
0x98	NOMEM	Node card has no local memory
0x99	I2CFATAL	-
0x9a	DISABLED	CPU is disabled by environment variable
0x9b	DOWNLOAD	Error downloading IO6PROM into RAM
0x9c	COREDEBUG	Can't set CORE debug registers
0x9d	IODISCOVER	Hub I/O discovery failed
0x9e	HUB_CONFIG	Failed writing Hub info into KLCONFIG
0x9f	ROUTER_CONFIG	Failed writing Router info into KLCONFIG
0xa0	HUBIO_INIT	Hub I/O initialization failed
0xa1	CONFIG_INIT	Failed initializing KLCONFIG
0xa2	RTRCHIP	Router chip failed diags
0xa3	LINKDEAD	LLP link failed diags
0xa4	HUBBIST	Hub chip failed built-in self test (BIST)
0xa5	RTRBIST	Router chip failed built-in self test (BIST)
0xa6	RESETWAIT	Waiting for reset to go
0xa7	LLP_FAIL	LLP failed after reset
0xa8	LLP_NORESET	LLP never came up after reset
0xa9	BADMEM	No good local memory
0xaa	NOT_USED_AA	-
0xab	NET_DISCOVER	Hub Network discovery failed
0xac	NASID_CALC	NASID calculation failed
0xad	ROUTE_CALC	Route calculation failed
0xae	ROUTE_DIST	Route distribution failed
0xaf	NASID_DIST	NASID distribution failed
0xb0	NO_NASID	Master did not assign a NASID
0xb1	NO_MODULEID	Module ID arbitration failed
0xb2	MIXED_SN00	Origin200 mixed with Origin2000

The following codes in Table 3-3 blink and indicate that an exception occurred so early in the PROM that no TTY device was available print information about the exception.

Table 3-3 Early Exception LED Values

LED	Name	Exception Type
0xf2	EXC_GENERAL	(Blinking) General exception
0xf3	EXC_ECC	(Blinking) ECC exception
0xf4	EXC_TLB	(Blinking) TLB exception
0xf5	EXC_XTLB	(Blinking) XTLB exception
0xf6	EXC_UNIMPL	(Blinking) Unimplemented exception
0xf7	EXC_CACHE	(Blinking) Cache Error exception

3.3 POD Mode (PROM Command Interpreter)

POD mode is a command interpreter present in the PROM which is most often used for debugging of a crashed system. It can be used to examine the contents of CPU registers, support chip registers, and memory. It can also enable or disable certain node card features, such as CPUs and memory banks.

POD can be run in three modes: Dex, Cac, and Unc.

- When running in Dex mode, POD requires very few system resources to run. It does not require memory instead, it accesses its program text directly from the PROM and uses the R10000™ microprocessor's data cache as memory for its stack. The secondary cache is not used. NMI and uncaught exceptions typically result in a Dex mode POD prompt.

  Dex mode is generally very slow because PROM instruction fetches are very slow. You would want to avoid performing long memory tests or flashing remote PROMs from Dex mode. Certain commands may not be executed in Dex mode, especially ones that attempt to program the PROM. See the go cac command for getting out of Dex mode.

  Also, when loading or storing data or running memory tests, be sure not to access cached memory addresses while running in Dex mode. See the go dex command.

- When running in Cac mode, POD places its program text, data, and stack in cached memory. This may only occur after memory has been probed and configured. POD runs very quickly out of cached memory. When the PROM takes an exception or NMI and enters POD mode, it goes into Dex mode. In the right circumstances it is possible to get back to Cac mode using the go cac command.
- When running in Unc mode, POD places its program text, data, and stack in uncached memory. This is similar to Cac mode, except the cache is not used and it runs slower.

The PROM can be forced to enter POD mode at different stages of the boot process by setting Debug Switches.

The POD mode prompt is in the format:

```
POD Dex MSC 001   3A>

___ ___ ___ ___ ___ __
 1   2   3   4   5   6
```

where the fields are:

1. Always POD, indicating POD mode.
2. *Dex*, indicating POD mode is running memoryless out of the cache only, or *Cac*, indicating POD mode is running out of cached memory, or *Unc*, indicating POD mode is running out of uncached memory.
3. *MSC*, indicating the POD prompt is being displayed on the MSC ACP port, or *Junk*, indicating the POD prompt is being displayed on the Junk UART, or *IOC3*, indicating the POD prompt is being displayed on an IOC3 UART, or *Talk*, indicating the POD prompt is being displayed on the Net UART.
4. The NASID (node ID) of the current node.
5. Occasionally, status fields may be inserted here indicating errors or alternate operating modes. *MDerr* indicates MD errors are pending (use the **error** command to view them and **clear** to clear them). *AltRegs* indicates the kernel register set is in effect rather than the PROM register set.
6. The slot (1 to 4) and slice (A or B) of the CPU displaying the prompt.
7. Many POD mode commands, including memory tests, flashing PROMs, and **loop** or **repeat** statements can be aborted by typing ^C.

The following editing characters are provided at the POD prompt:

^C	Abort
^H, Backspace, or DEL	Backspace
^R	Redisplay line
^U	Erase line

Three handy command line editing features are borrowed from the C shell:

1. Wherever the symbol **!!** appears in a command line, it is substituted with the contents of the previous command line.
2. Wherever the symbol **!$** appears in a command line, it is substituted with the last word from the previous command line.
3. The command line **^xxx^yyy** repeats the previous command line, except substitutes the first occurrence of the string *xxx* with the string *yyy*.

In addition, environment variables (see **setenv** command) are substituted wherever the name of an environment variable appears in backquotes (``). This is useful as a form of aliases. For example:

```
POD Dex Hub 000 1A> setenv range
Set “range” to? n:1 u: 0x3ff0000 2m
POD Dex Hub 000 1A> dirtest `range`
POD Dex Hub 000 1A> dirinit `range`
POD Dex Hub 000 1A> memtest `range`
```

3.3.1 POD Consoles

POD mode will present a prompt on several possible I/O devices. It probes for devices in the order listed below and uses the first one found. When the CPU is at the POD prompt, it will blink an LED pattern at 0.5 Hz indicating on which UART input is expected.

junk	If a Junk UART is plugged in, it is used preferentially. Junk UARTs are available only in manufacturing and bring-up. LED pattern: 0x80/0xb3
ioc3	If a BaseIO (IO6) is found, and one of the ports on it is marked as the system console, and it can be initialized and accessed without causing an exception, the ioc3 uart is used. LED pattern: 0x80/0x8c
msc	If the Module System Controller is responsive, the MSC is used for I/O. (The MSC was at one time called the Entry Level System Controller, or ELSC). LED pattern: 0x80/0xbc
netuart	If no UART is available, the netuart is used (see below). This is not a real UART, but an emulation that allows one Hub to talk to another. LED pattern: 0x80/0xbf

3.3.1.1 Netuart

When a CPU is using the netuart, it monitors some shared memory locations and interrupt lines used for communication. From any node that has a UART and is at the POD prompt, you can use the talk command to communicate with CPUs that are using the netuart. Use of the netuart is not recommended.

3.3.1.2 Constants

PROM constants are similar to C constants. Hexadecimal numbers are prefixed with 0x; octal numbers are prefixed with 0; binary numbers are prefixed with 0b; all other numbers are in decimal.

Three special suffixes are provided to facilitate entering large constants:

```
g       Gigabytes               e.g.: 1g == (1 << 30)
m       Megabytes               e.g.: 32m == (32 << 20)
k       Kilobytes               e.g.: 100k == (100 << 10)
```

3.3.1.3 Expressions

Arithmetic expressions are permitted and are similar to C expressions in operators and order of evaluation. Many commands require expressions to be parenthesized to avoid ambiguity. For example:

```
px (0xdeadbeef >> 6) & 0xff
sd md_memory_config (ld md_memory_config | 077777777)
```

The supported operators are:

```
Unary: () ~ ! - + LD LW LH LB, and * (same as LD)
Binary: + - * / % | & ^ << >> == != < > <= >= && ||
```

Commands that require strings may sometimes require quotes on the strings. Within quotes, standard C escape sequences (\b, \t, \n, \r, \\, and \ddd) are permitted. There is a 31-character limit on string length.

3.3.1.4 Statements

More than one command may be performed on a command line by using a command list in which commands are separated by semicolons; e.g.:

```
sd u:0 1; ld u:0
```

Commands that take other commands as arguments, such as loop, may take a single command or a compound command consisting of a command list in curly braces; e.g.:

```
loop ld pi_rt_count
loop { ld pi_rt_count; sd pi_rt_count 0 }
```

A pound sign (#) begins a comment that lasts to the end of the input line. This may be useful when cutting and pasting into a terminal window from command files.

3.3.1.5 Address Modifiers

For convenience in entering constants which represent R10000 and Origin2000 addresses, a number of modifiers are available. The modifiers should precede the constants. Multiple modifiers may be used, as shown in Table 3-4.

Table 3-4 Address Modifiers

Modifier	Action	Example	Equivalent To
h:	Set top byte to 0x90 (HSPEC).	h:256m	0x9000000010000000 (LBOOT)
i:	Set top byte to 0x92 (IO).	i:(16m+32)	0x9200000001000020
m:	Set top byte to 0x94 (MSPEC).	m:0	0x9400000000000000
u:	Set top byte to 0x96 (UNCAC).	u:0	0x9600000000000000
c:	Set top byte to 0xa8 (CAC).	c:0	0xa800000000000000

Table 3-4 (continued) Address Modifiers

Modifier	Action	Example	Equivalent To
p:	Set top byte to 0x00 (PHYS).	p:c:0x100	0x100
s:	Sign extend word.	s:0xbfc00000	0xffffffffbfc00000
n:#	Add NASID, where # is NASID. May only be used if CrayLink routing is configured.	n:2 c:1	0xa800000200000001
w:#	Set top byte to 0x92. Sets bit 24. Adds Widget, where # is widget. If remote, adds bit 23.	w:1 0x200 n:1 w:1 0x200	0x9200000001000200 0x9200000101800200
b:#	Adds offset for memory bank (# times 512 MB).	n:3 b:5 u:2k	0x96000003a0000800
L:	Converts address to corresponding directory entry LO in HSPEC.	L:0x2100560	0x90000000c08402a0
H:	Converts address to corresponding directory entry HI in HSPEC.	H:u:0x2100560	0x90000000c08402a8
P:#	Converts address to corresponding protection entry, where # is the region.	P:5 0x2100560	0x90000000c0840028
E:	Converts address to corresponding ECC word address.	E:0x2100560	0x9000000080840158

3.4 Hardware Registers

The PROM knows the names and bit field assignments for approximately 500 hardware registers in the CPU, Hub, Router, and Crossbow, so you do not need to manually look up and type their addresses (or values in the case of CPU registers). Names of registers may appear in arithmetic expressions and are replaced with the registers' addresses.

The names of R10000 Cop0 registers are recognized, and include Cause, Status, etc. The names of the R10000 general purpose registers are recognized (v0, a1, ta0, gp, r5, etc). General purpose registers may also be referred to as $0, $1, $2, etc. Floating point registers may be referenced as $f0, $f1, $f2, etc.

The names of Hub registers are prefixed according to the five sections of the Hub with *PI_*, *MD_*, *II_*, *NI_*, and *CORE_*.

The names of Hub NI and Router registers can be used in place of the vector address in some vector commands. Router registers are prefixed with *RR_*. The names of the Crossbow (XBOW) registers are prefixed with *XB_*.

Register names are case-insensitive and may be truncated to the extent that they are unambiguous. As shown in the following examples.

Example 3-1 Case-insensitive Hub Register

```
POD Dex IOC3 000 4A> px (md_refresh_control + 8)
0x9200000001200028
POD Dex IOC3 000 4A> pr n:1 md_ref
Register: MD_REFRESH_CONTROL (0x9200000101a00020)
Value   : 0x0000000000000000 (loaded from remote register)
   <63> RW  ENABLE                    0x1
<23:12> RW  COUNTER                   0x2e0
<11:00> RW  CNT_THRESH                0x504
```

Example 3-2 Hub PI (Processor Interface) Registers

```
PI_BIST_COUNT_TARGET PI_BIST_ENTER_RUN PI_BIST_RDY  PI_BIST_READ_DATA
PI_BIST_SHIFT_LOAD

PI_BIST_SHIFT_UNLOAD PI_BIST_WRITE_DATA  PI_CALIAS_SIZE PI_CC_MASK PI_CC_PEND_CLR_A
PI_CC_PEND_CLR_B

PI_CC_PEND_SET_A PI_CC_PEND_SET_B PI_CPU_ENABLE_A PI_CPU_ENABLE_B PI_CPU_NUM
PI_CPU_PRESENT_A

PI_CPU_PRESENT_B PI_CPU_PROTECT PI_CRB_SFACTOR PI_CRB_TIMEOUT_A PI_CRB_TIMEOUT_B
PI_ERR_INT_MASK_A

PI_ERR_INT_MASK_B  PI_ERR_INT_PEND PI_ERR_STACK_ADDR_A PI_ERR_STACK_ADDR_B
PI_ERR_STACK_FORMAT

PI_ERR_STACK_SIZE PI_ERR_STATUS0_A PI_ERR_STATUS0_A_CLEAR  PI_ERR_STATUS0_B
PI_ERR_STATUS0_B_CLEAR

PI_ERR_STATUS1_A PI_ERR_STATUS1_A_CLEAR PI_ERR_STATUS1_B PI_ERR_STATUS1_B_CLEAR
PI_FORCE_BAD_CHECK_BIT_A

PI_FORCE_BAD_CHECK_BIT_B PI_GFX_BIAS_A PI_GFX_BIAS_B  PI_GFX_CREDIT_CNTR_A
PI_GFX_CREDIT_CNTR_B

PI_GFX_INT_CMP_A PI_GFX_INT_CMP_B  PI_GFX_INT_CNTR_A PI_GFX_INT_CNTR_B PI_GFX_PAGE_A
PI_GFX_PAGE_B

PI_INT_MASK0_A  PI_INT_MASK0_B PI_INT_MASK1_A PI_INT_MASK1_B PI_INT_PEND0
PI_INT_PEND1  PI_INT_PEND_MOD

PI_IO_PROTECT PI_MAX_CRB_TIMEOUT PI_NACK_CMP PI_NACK_CNT_A  PI_NACK_CNT_B PI_NMI_A
PI_NMI_B

PI_PROFILE_COMPARE PI_PROF_INT_EN_A  PI_PROF_INT_EN_B PI_PROF_INT_PEND_A
PI_PROF_INT_PEND_B PI_PROT_OVRRD

PI_REGION_PRESENT PI_REPLY_LEVEL PI_RTC_INT_EN_A PI_RTC_INT_EN_B PI_RT_COMPARE_A
PI_RT_COMPARE_B

PI_RT_COUNTER PI_RT_FILTER_CTRL PI_RT_INT_PEND_A PI_RT_INT_PEND_B PI_RT_LOCAL_CTRL
PI_SOFTRESET

PI_SPOOL_CMP_A  PI_SPOOL_CMP_B PI_SYSAD_ERRCHK_EN PI_UNUSED
```

Example 3-3 Hub MD (Memory / Directory) Registers

```
MD_DIR_DIMM_INIT MD_DIR_ERROR MD_DIR_ERROR_CLR  MD_FANDOP_CAC_STAT MD_HSPEC_PROTECT
MD_IO_PROTECT

MD_IO_PROT_OVRRD MD_LED0  MD_LED1 MD_MEMORY_CONFIG MD_MEM_DIMM_INIT MD_MEM_ERROR
MD_MEM_ERROR_CLR

MD_MIG_CANDIDATE MD_MIG_CANDIDATE_CLR MD_MIG_DIFF_THRESH  MD_MIG_VALUE_THRESH
MD_MISC_ERROR

MD_MISC_ERROR_CLR MD_MLAN_CTL MD_MOQ_SIZE  MD_PERF_CNT0 MD_PERF_CNT1 MD_PERF_CNT2
MD_PERF_CNT3

MD_PERF_CNT4  MD_PERF_CNT5 MD_PERF_SEL MD_PROTOCOL_ERROR MD_PROTOCOL_ERROR_CLR
MD_REFRESH_CONTROL

MD_SLOT_ID_USTATUS MD_UREG0_0 MD_UREG0_1 MD_UREG0_2 MD_UREG0_3  MD_UREG0_4 MD_UREG0_5
MD_UREG0_6

MD_UREG0_7 MD_UREG1_0 MD_UREG1_1  MD_UREG1_10 MD_UREG1_11 MD_UREG1_12 MD_UREG1_13
MD_UREG1_14 MD_UREG1_15

MD_UREG1_2 MD_UREG1_3 MD_UREG1_4 MD_UREG1_5 MD_UREG1_6 MD_UREG1_7  MD_UREG1_8
MD_UREG1_9
```

Example 3-4 Hub II (I / O) Registers

```
II_IBCT0 II_IBCT1 II_IBDA0 II_IBDA1 II_IBIA0 II_IBIA1 II_IBLS0 II_IBLS1 II_IBNA0
II_IBNA1 II_IBSA0

II_IBSA1 II_ICCR II_ICDR II_ICMR II_ICRB0_A II_ICRB0_B II_ICRB0_C II_ICRB0_D
II_ICRB1_A II_ICRB1_B

II_ICRB1_C II_ICRB1_D II_ICRB2_A II_ICRB2_B II_ICRB2_C II_ICRB2_D II_ICRB3_A
II_ICRB3_B II_ICRB3_C

II_ICRB3_D II_ICRB4_A II_ICRB4_B II_ICRB4_C II_ICRB4_D II_ICRB5_A II_ICRB5_B
II_ICRB5_C II_ICRB5_D

II_ICRB6_A II_ICRB6_B II_ICRB6_C II_ICRB6_D II_ICRB7_A II_ICRB7_B II_ICRB7_C
II_ICRB7_D II_ICRB8_A

II_ICRB8_B II_ICRB8_C II_ICRB8_D II_ICRB9_A II_ICRB9_B II_ICRB9_C II_ICRB9_D
II_ICRBA_A II_ICRBA_B

II_ICRBA_C II_ICRBA_D II_ICRBB_A II_ICRBB_B II_ICRBB_C II_ICRBB_D II_ICRBC_A
II_ICRBC_B II_ICRBC_C

II_ICRBC_D II_ICRBD_A II_ICRBD_B II_ICRBD_C II_ICRBD_D  II_ICRBE_A II_ICRBE_B
II_ICRBE_C II_ICRBE_D

II_ICTO II_ICTP II_IECLR II_IFDR II_IGFX0 II_IGFX1 II_IIAP II_IIDEM II_IIDSR II_IIWA
II_ILAPO II_ILAPR

II_ILCSR II_ILLR II_IMEM II_IOWA II_IPCA II_IPCR II_IPDR II_IPPR II_IPRB0 II_IPRB8
II_IPRB9 II_IPRBA
```

```
II_IPRBB II_IPRBC II_IPRBD II_IPRBE II_IPRBF II_IPRTE II_IPRTE0 II_IPRTE1 II_IPRTE2
II_IPRTE3 II_IPRTE4

II_IPRTE5 II_IPRTE6 II_IPRTE7 II_IPTP0 II_IPTP1 II_ITTE1 II_ITTE2 II_ITTE3 II_ITTE4
II_ITTE5 II_ITTE6

II_ITTE7 II_IXCC II_IXTT II_NOT_DONE II_WCR II_WID II_WSTAT
```

Example 3-5 Hub NI (Network Interface) Registers

```
NI_AGE_CPU0_MEMORY NI_AGE_CPU0_PIO NI_AGE_CPU1_MEMORY  NI_AGE_CPU1_PIO
NI_AGE_GBR_MEMORY NI_AGE_GBR_PIO

NI_AGE_IO_MEMORY  NI_AGE_IO_PIO NI_DIAG_PARMS NI_ERROR_CLEAR NI_GLOBAL_PARMS
NI_IO_PROTECT

NI_LOCAL_TABLE<00> NI_META_TABLE<00> NI_PORT_ERROR NI_PORT_PARMS NI_PORT_RESET
NI_PROTECTION_CONFIG

NI_PROT_OVRRD  NI_RETURN_VECTOR NI_SCRATCH_REG0 NI_SCRATCH_REG1 NI_STATUS_REV_ID
NI_VECTOR  NI_VECTOR_CLEAR

NI_VECTOR_DATA NI_VECTOR_PARMS NI_VECTOR_READ_DATA  NI_VECTOR_STATUS
```

Example 3-6 Hub CORE (Internal crossbar control) Registers

```
CORE_CACR CORE_CDBGSEL CORE_CFDR CORE_CMDAB CORE_CNDAB  CORE_CRPQD CORE_CRQQD
```

Example 3-7 Crossbow Registers (midplane)

```
XB_ARB_RELOAD XB_CTRL XB_ERR_CMDWORD XB_ERR_LOWER  XB_ERR_UPPER XB_ID
XB_INT_LOWER XB_INT_UPPER

XB_LINK_ARB_LOWER_8  XB_LINK_ARB_LOWER_9 XB_LINK_ARB_LOWER_A XB_LINK_ARB_LOWER_B
XB_LINK_ARB_LOWER_C

XB_LINK_ARB_LOWER_D XB_LINK_ARB_LOWER_E XB_LINK_ARB_LOWER_F XB_LINK_ARB_UPPER_8
XB_LINK_ARB_UPPER_9

XB_LINK_ARB_UPPER_A XB_LINK_ARB_UPPER_B XB_LINK_ARB_UPPER_C XB_LINK_ARB_UPPER_D
XB_LINK_ARB_UPPER_E

XB_LINK_ARB_UPPER_F XB_LINK_AUX_STAT_8 XB_LINK_AUX_STAT_9 XB_LINK_AUX_STAT_A
XB_LINK_AUX_STAT_B

XB_LINK_AUX_STAT_C XB_LINK_AUX_STAT_D XB_LINK_AUX_STAT_E XB_LINK_AUX_STAT_F
XB_LINK_CTRL_8 XB_LINK_CTRL_9

XB_LINK_CTRL_A XB_LINK_CTRL_B XB_LINK_CTRL_C  XB_LINK_CTRL_D XB_LINK_CTRL_E
XB_LINK_CTRL_F

XB_LINK_IBUF_FLUSH_8  XB_LINK_IBUF_FLUSH_9 XB_LINK_IBUF_FLUSH_A XB_LINK_IBUF_FLUSH_B
XB_LINK_IBUF_FLUSH_C

XB_LINK_IBUF_FLUSH_D XB_LINK_IBUF_FLUSH_E XB_LINK_IBUF_FLUSH_F XB_LINK_RESET_8
XB_LINK_RESET_9
```

```
XB_LINK_RESET_A XB_LINK_RESET_B XB_LINK_RESET_C XB_LINK_RESET_D XB_LINK_RESET_E
XB_LINK_RESET_F

XB_LINK_STAT_8 XB_LINK_STAT_9 XB_LINK_STAT_A XB_LINK_STAT_B  XB_LINK_STAT_C
XB_LINK_STAT_CLR_8

XB_LINK_STAT_CLR_9 XB_LINK_STAT_CLR_A  XB_LINK_STAT_CLR_B XB_LINK_STAT_CLR_C
XB_LINK_STAT_CLR_D

XB_LINK_STAT_CLR_E  XB_LINK_STAT_CLR_F XB_LINK_STAT_D XB_LINK_STAT_E XB_LINK_STAT_F
XB_LLP_CTRL XB_NIC

XB_PERF_CTR_A XB_PERF_CTR_B XB_PKT_TO XB_STAT XB_STAT_CLR
```

Example 3-8 Router Registers

```
RR_BIST_COUNT_TARGET RR_BIST_DATA RR_BIST_ENTER_RUN  RR_BIST_READY RR_BIST_SHIFT_LOAD
RR_BIST_SHIFT_UNLOAD

RR_DIAG_PARMS  RR_ERROR_CLEAR1 RR_ERROR_CLEAR2 RR_ERROR_CLEAR3 RR_ERROR_CLEAR4
RR_ERROR_CLEAR5

RR_ERROR_CLEAR6 RR_GLOBAL_PARMS RR_HISTOGRAM1 RR_HISTOGRAM2  RR_HISTOGRAM3
RR_HISTOGRAM4 RR_HISTOGRAM5

RR_HISTOGRAM6 RR_LOCAL_TABLE<00> RR_META_TABLE<00> RR_NIC_ULAN RR_PORT_PARMS1
RR_PORT_PARMS2

RR_PORT_PARMS3 RR_PORT_PARMS4 RR_PORT_PARMS5 RR_PORT_PARMS6 RR_PORT_RESET
RR_PROTECTION_CONFIG

RR_RESET_MASK1 RR_RESET_MASK2  RR_RESET_MASK3 RR_RESET_MASK4 RR_RESET_MASK5
RR_RESET_MASK6 RR_SCRATCH_REG0

RR_SCRATCH_REG1 RR_STATUS_ERROR1 RR_STATUS_ERROR2 RR_STATUS_ERROR3  RR_STATUS_ERROR4
RR_STATUS_ERROR5

RR_STATUS_ERROR6 RR_STATUS_REV_ID
```

Example 3-9 R10000 Coprocessor 0 Registers

```
C0_23 C0_24 C0_7 C0_BADVADDR C0_BRDIAG C0_CACHEERR C0_CAUSE  C0_COMPARE C0_CONFIG
C0_CONTEXT C0_COUNT

C0_ECC C0_ENTRYHI C0_ENTRYLO0  C0_ENTRYLO1 C0_EPC C0_ERROREPC C0_FRAMEMASK C0_INDEX
C0_LLADDR C0_PAGEMASK

C0_PERFCOUNT C0_PRID C0_RANDOM C0_STATUS C0_TAGHI C0_TAGLO C0_WATCHHI  C0_WATCHLO
C0_WIRED C0_XCONTEXT
```

3.5 Symbols

The PROM knows the names of symbols within the PROM (and within the kernel if one is loaded; see the **kern_sym** command). Symbols may appear in arithmetic expressions and are replaced with their addresses. For example:

```
POD Dex IOC3 000 4A> px main
0xc00000001fc07784
POD Dex IOC3 000 4A> dis (main+0x1c) 4
[main+0x001c, 0x1fc077a0]:      0001083c        dsll32  at,at,#0
[main+0x0020, 0x1fc077a4]:      64420000        daddiu  v0,v0,0x0
[main+0x0024, 0x1fc077a8]:      0039082c        dadd    at,at,t9
[main+0x0028, 0x1fc077ac]:      0002103c        dsll32  v0,v0,#0
POD Dex IOC3 000 4A> px “memcpy”
0xc00000001fc1db14
```

Note that certain symbols, such as **memcpy**, must be quoted because they are reserved words in the PROM (POD command).

Chapter 4

POD Mode Commands

4.1 Conventions

Data types shown in all caps are not to be input literally, but indicate the type of input expected. Items shown in square brackets ([]) are optional. Some of the types are shown in Table 4-1.

Table 4-1 Data Types

Type	Represents
EXPR	Arithmetic expression
GPRNAME	GPR register name (symbolic or $num)
REGNO	Register number, from 0 to 31
BITNO	Bit number, from 0 to 63
ADDR	Address, or arithmetic expression enclosed in parenthesis
START	Start address
LEN	Length in bytes
VAL	Number, or arithmetic expression enclosed in parenthesis
COUNT	Count or length
CMD	Command
NODE	Node ID (also called NASID: Numa Address Space Identifier)
SLICE	CPU, either A or B
VEC	CrayLink Vector path (e.g., 0x15)
VADDR	CrayLink Vector address (offset or NI or RR register name)

4.2 Command Summary

The **help** or **?** command, used by itself, prints a command summary similar to that shown in Table 4-2. The **help** or **?** command may also take one command name argument to print the summary for a specific command.

Parts of a command enclosed in square brackets ([]) are optional. Multiple possibilities are separated by vertical bars (|).

Table 4-2 Command Summary

Command	Purpose	Syntax
px	Print hex	px EXPR
pd	Print decimal	pd EXPR
po	Print octal	po EXPR
pb	Print binary	pb EXPR
pr	Print register	pr [GPRNAME [VAL]]
pf	Print fpreg	pf [REGNO]
pa	Print address	pa ADDR [BITNO]
lb	Load byte	lb ADDR [COUNT]
lh	Load half-word	lh ADDR [COUNT]
lw	Load word	lw ADDR [COUNT]
ld	Load double-word	ld ADDR [COUNT]
sb	Store byte	sb ADDR [VAL [COUNT]]
sh	Store half-word	sh ADDR [VAL [COUNT]]
sw	Store word	sw ADDR [VAL [COUNT]]
sd	Store double-word	sd ADDR [VAL [COUNT]]
sdv	sd with verify	sdv ADDR COUNT
sr	Store register	sr REG VAL
sf	Store fpreg	sf REGNO VAL
vr	Vector rea	vr VEC VADDR
vw	Vector write	vw VEC VADDR VAL
vx	Vector exchange	vx VEC VADDR VAL
repeat	Repeat count	repeat COUNT CMD
loop	Repeat forever	loop CMD
while	While loop	while (EXPR) CMD
for	For loop	for (CMD;EXPR;CMD) CMD

Table 4-2 (continued) Command Summary

Command	Purpose	Syntax
if	If statement	if (EXPR) CMD
delay	Delay	delay MICROSEC
sleep	Sleep	sleep SEC
time	Benchmark timing	time CMD
echo	Echo a string	echo “STRING”
jump	Inv. cache & jump	jump ADDR [A0 [A1]]
call	Call subroutine	call ADDR [A0 [A1]]
reset	Reset the system	reset
softreset	Softreset a node	softreset n:NODE
nmi	Send NMI to node	nmi n:NODE [a | b]
help	Display help	help [CMDNAME]
inval	Inv. cache(s)	inval [i][d][s]
flush	Flush+inv caches	flush
tlbc	Clear TLB	tlbc [INDEX]
tlbr	Read TLB	tlbr [INDEX]
dtag	Dump dcache tag	dtag line
itag	Dump icache tag	dtag line
stag	Dump scache tag	stag line
dline	Dump dcache line	dline line
iline	Dump icache lin	iline line
sline	Dump scache line	sline line
adtag	Dump dcache tag	adtag line
aitag	Dump icache tag	aitag line
astag	Dump scache tag	astag line
adline	Dump dcache line	adline line
ailine	Dump icache line	ailine line
asline	Dump scache line	asline line
go	Set memory mode	go dex | unc | cac
bist	Self-test Hub	bist le | ae | lr | ar [n:NODE]
rbist	Self-test Router	rbist le | ae | lr | ar VEC
disable	Disable unit	disable n:NODE [SLICE]

Table 4-2 (continued) Command Summary

Command	Purpose	Syntax
enable	Enable unit	enable n:NODE [SLICE]
tdisable	Temp. disable	tdisable n:NODE [SLICE]
dirinit	Dir / prot init	dirinit START LEN
meminit	Memory clear	meminit START LEN
dirtest	Dir. test / init	dirtest START LEN
memtest	Memory test / init	memtest START LEN
santest	Mem. sanity test	santest ADDR
slave	Goto slave mode	slave
segs	List segments	segs [FLAG]
exec	Load / exec segment	exec [SEGNAME [FLAG]]
why	Why are we here?	why
fru	Run FRU Analyzer	fru [1 | 2]
clear	Clear errors	clear
error	Display errors	error
ioc3	Use IOC3 UART	ioc3
junk	Use JunkBus UART	junk
elsc	Use MSC (SysCtlr UART)	elsc
talk	Use Net UART	talk [n:NODE a | b]
nm	Look up PROM addr	nm ADDR
disc	Discover CrayLink Interconnect	disc
pcfg	Dump pcfg struct	pcfg [n:NODE] [v]
node	Get / set node ID	node [[VEC] ID]
crb	Dump II CRBs	crb [n:NODE]
crbx	133-col wide crb	crbx [n:NODE]
route	Set up route	route [VEC NODE]
flash	Prgm remote PROM	flash NODE [...]
nic	Read Hub NIC	nic [n:NODE]
rnic	Read Router NIC	rnic [VEC]
sc	System ctlr cmd	sc [“STRING”]
scw	Wr sysctlr nvram	scw ADDR [VAL [COUNT]]
scr	Rd sysctlr nvram	scr ADDR [COUNT]

Table 4-2 (continued) Command Summary

Command	Purpose	Syntax
dips	Rd hardware debug switches	dips
dbg	Set / Rd debug switches	dbg [VIRT_VAL PHYS_VAL]
pas	Set / Rd MSC password	pas [“PASW”]
module	Set / Rd module number	module [VAL]
modnic	Get module NIC	modnic
qual	Quality mode	qual [1 | 0]
ecc	ECC mode	ecc [1 | 0]
cpu	Switch to cp	cpu [[n:NODE] a | b]
dis	Disassemble	dis ADDR [COUNT]
im	Set R10k int mask	im [BYTE]
dumpspool	Dump PI err spool	dumpspool [n:NODE SLICE]
error_dump	Dump Hub error info	error_dump
edump_bri	Dump Bridge error info	edump_bri [n:NODE]
maxerr	Test error limit	maxerr COUNT
rtab	Dump route table	rtab [VEC]
rstat	Dmp / clr rtr stat	rstat VEC
version	Show PROM version	version
memset	Fill mem w / byte	memset DST BYTE LEN
memcpy	Copy memory bytes	memcpy DST SRC LEN
memcmp	Cmp memory bytes	memcmp DST SRC LEN
memsum	Add memory bytes	memsum SRC LEN
scandir	Scan dir states	scandir ADDR [LEN]
dirstate	Directory stat	dirstate [STATE [BASE [LEN]]]
altregs	Use alt. regs	altregs NUM
traplog	Trap log	traplog ADDR
kern_sym	Use kernel symtab	kern_sym
kdebug	Enable kernel debug functions	kdebug [STACKADDR]
initlog	Init. PROM log	initlog [n:NODE]
setenv	Set variable	setenv [n:NODE] VAR [“STRING”]
unsetenv	Remove variable	unsetenv [n:NODE] VAR
printenv	Print variables	printenv [n:NODE] [VAR]

Table 4-2 (continued) Command Summary

Command	Purpose	Syntax
log	Print PROM log entries	log [n:NODE] [SKIP [COUNT]]
rlog	Print PROM log entries in reverse	rlog [n:NODE] [SKIP [COUNT]]
setpart	Set up partition	setpart RTR_LIST
hubsde	Send Hub data error	hubsde
rtrsde	Send Router data error	rtrsde
chklink	Check local link	chklink
dgxbow	Run Crossbow diagnostic	dgxbow [m<n | h | m> [n<NODE>]
dgbrdg	Run Bridge diagnostic	dgbrdg [m<n | h | m>] [n<NODE>] [s<SLOT>]
dgconf	Run BaseIO configuration space diagnostic	dgconf [m<n | h | m>] [n<NODE>] [s<SLOT>]
dgpci	Run PCI bus diagnostic	dgpci [m<n | h | m>] [n<NODE>] [s<SLOT>] [p<PCI#>]
dgspio	Run serial PIO diagnostic	dgspio [m<n | h | m | x>] [n<NODE>] [s<SLOT>] [p<PCI#]
dgsdma	Run serial DMA diagnostic	dgsdma [m<n | h | m | x>] [n<NODE>] [s<SLOT>] [p<PCI#]

4.3 Command Descriptions

4.3.1 Evaluate Expressions

- Print Expression (Calculator)
  - px EXPR
  - pd EXPR
  - po EXPR
  - pb EXPR

Prints the value of an arithmetic expression in hexadecimal, decimal, octal, or binary.

```
Example 1: px n:1 md_memory_config
Example 2: pd 0x3ea
Example 3: pd (0x12000 + 768) / 4
```

- Print Address
  - pa ADDR [BITNO]

Decodes an individual memory address and bit to print the name of the physical DIMM slot where the bit resides and the data line number of the bit. If only an address is specified, bit 0 is assumed. This works for both regular memory addresses and back door directory (HSPEC) addresses. For example:

```
POD Dex IOC3 000 4A> pa 0x40000 21
Memory address 0x0000000000040000 bit 21: MMXL0 DIMML line 39
```

- nm ADDR

Given an address, attempts to look up the function containing the address in the PROM symbol table (or Kernel table if using kernel symbols; see kern_sym) and if found, displays the function name and offset into the function. For example:

```
POD Dex Hub 000 1A> nm 0xc00000001fc07c04
main+0x488
```

4.3.2 Display Registers

- Print Register
  - pr [GPRNAME [VAL | ~]

Prints the value of R10000 register(s) or a Hub register. Without arguments, prints out the R10000 GPRs. With a register name argument, loads a value from the register and decodes

it into bit fields. With a register name and value, decodes the specified value into bit fields. With a register name and tilde (~), decodes the register's reset default value into bit fields. If a value or tilde (~) is specified, the command may also be used with router registers *(RR_*)*.

A register name may be truncated to its smallest unambiguous form. If the truncated name is ambiguous, all of the possible choices are listed.

```
Example 1: pr
        Prints all 32 R10000 GPRs with their number and symbolic names
Example 2: pr 2; pr $2; pr r2; pr r02; pr v0
        All of these print the value of GPR 2
Example 3: pr cause
        Prints the value of the R10000 COP0 CAUSE register
Example 4: pr pi
        This register name is ambiguous and causes all of the
        Hub PI registers to be printed.
Example 5: pr md_memory_config
        Prints the address, contents, and individually decoded
        bit fields for the specified Hub register
Example 6: pr n:1 md_memory_config
        Prints the address, contents, and individually decoded
        bit fields for the specified Hub register loaded
        from the remote node with NASID 1.
Example 7: pr md_memory_config 0x001010004fffffff
        Decodes the specified constant into individual bit
        fields for the specified Hub register
Example 8: pr md_memory_config ~
        Decodes the reset default value of the specified Hub
        register into individual bit fields for the register.
Example 9: pr rr_status_rev_id ~
        Decodes the reset default value of the specified Router
        register into individual bit fields for the register.
```

- Print Floating Point Register
  - pf [REGNO]

Prints the value of R10000 COP1 (floating point) register(s)

```
Example 1: pf

        Prints all 32 R10000 FPRs

Example 2: pf 3

        Prints $f3.

Example 3: px $f3

        Also prints $f3 (floating point register names can be

        used in expressions).
```

4.3.3 Load From and Store To Memory

- lb ADDR [COUNT]
- lh ADDR [COUNT]
- lw ADDR [COUNT]
- ld ADDR [COUNT]

Loads a byte, half-word, word, or double-word from a specified address, and displays the result in hexadecimal. If COUNT is specified, COUNT items are loaded and displayed in columns. If the specified address is not properly aligned, an exception may result. Note that cached addresses should not be accessed while running in Dex mode.

- la ADDR [COUNT]

Loads bytes similar to the lb command, except the output is displayed as ASCII characters instead of hexadecimal. Bytes outside the range 0x20 (space) through 0x7e (tilde) are displayed as octal byte codes. Note that cached addresses should not be accessed while running in Dex mode.

- sb ADDR [VAL [COUNT]]
- sh ADDR [VAL [COUNT]]
- sw ADDR [VAL [COUNT]]
- sd ADDR [VAL [COUNT]]

Stores a byte(s), half-word(s), word(s), or double-word(s) to a specified address. Note that cached addresses should not be accessed while running in Dex mode.

If neither VAL or COUNT is specified, the commands enter editing mode in which the current contents of each address is displayed and the user is allowed to modify it by typing in a hexadecimal value (without the x). Entering nothing goes on to the next address without modifying the location. Entering a period exits editing mode.

If VAL is specified, the value is stored at the specified address. If COUNT is also specified, the value is stored COUNT times starting at the specified address.

- sdv ADDR COUNT

Stores a double-word at a specified address, loads it back, and verifies the result. If what was read back was not what was written, it prints an error message. Note that cached addresses should not be accessed while running in Dex mode.

4.3.4 Set Registers

- Set General Purpose Register

- sr GPRNAME VAL
- GPRNAME = VAL

Assigns a R10000 GPR with the requested value. The modification is done in the register storage area where the registers are saved in the event of an exception or NMI.

- Set Floating Point Register
  - sf REGNO VAL
  - $f REGNO = VAL

Assigns a R10000 FPR with the requested value. The modification is done in the actual R10000 FPRs.

- Set Coprocessor 0 Register
  - COP0REG = VAL

Assigns the value of a specified R10000 Coprocessor 0 register to a requested value. The actual R10000 register is modified, except in the case of the following registers, where the value is written into the register storage area where the registers are saved in the event of an exception or NMI: Status, Cause, EPC, BadVAddr, and ErrorEPC. The names of all the registers are:

```
Index, Random, EntryLo0, EntryLo1, Context, PageMask,
Wired, Cop0_7, BadVAddr, Count, EntryHi, Compare,
Status, Cause, EPC, PRId, Config, LLAddr, WatchLo,
WatchHi, XContext, FrameMask, BrDiag, Cop0_23, Cop0_24,
PerfCount, ECC, CacheErr, TagLo, TagHi, ErrorEPC.
```

4.3.5 Vector Operation

- vr VEC VADDR
- vw VEC VADDR VAL
- vx VEC VADDR VAL

Performs a CrayLink vector read, write, or exchange. VEC is the CrayLink vector of the destination Hub or Router (see section Vector Addressing), VADDR is the address of which register to access in the Hub or Router, and VAL is the value to be written or exchanged.

VADDR, in addition to being a numeric offset, when talking to a Hub may be a symbolic Hub register name from the subset of registers that begin with *NI_*, or when talking to a Router may be a symbolic Router register name from the subset of registers that begin with *RR_*.

- route [VEC NODE]

Given a vector route VEC from the local Hub to a remote Hub, and a Node ID (or NASID) to be assigned to the remote Hub, this command programs a minimum subset of the router tables in the local Hub, remote Hub, and all of the intervening routers, as well as sets the

remote node's NASID. This will cause the remote node to crash. However, it renders the entire memory and I/O space of the remote node accessible to the local node.

```
Example:
    POD Cac Hub 000 1A> vr 0x53 ni_status_rev_id
    POD Cac Hub 000 1A> route 0x53 1
    POD Cac Hub 000 1A> pr n:1 ni_status_rev_id
    POD Cac Hub 000 1A> ld n:1 u:0 10
    POD Cac Hub 000 1A> flash 1
```

If no vector and node are given, the PROM consults the internal pcfg structure (see **disc** and **pcfg** commands), calculates the NASIDs and routing tables, and programs the NASIDs and routing tables for all Hubs and Routers in the system, using the same calculations that the normal boot sequence uses.

4.3.6 Control

- repeat COUNT CMD

Executes any POD command(s) a specified number of times. Repeat loops may be nested and may be broken by typing Control-C.

```
Example 1: repeat 20 vr 0 0
Example 2: repeat 4 { repeat 3 px 3; repeat 2 px 2 }
```

- loop CMD

Executes a command as long as expression is true.

```
Example 1: loop pr pi_rt_count
Example 2: loop { sd md_led0 0x55; sleep 1; sd md_led0 0xaa; sleep 1 }
```

- while (EXPR) CMD

Executes a command as long as expression is true.

```
Example: while ((ld md_ureg8 & 0x40) != 0) {}
```

- for (CMD;EXPR;CMD) CMD

Executes a "for" statement similar to the C "for" statement. The first command is executed, then the last command and the second command are executed as long as the expression is true.

```
Example: for (v0=0;v0<5;v0=v0+1) {px v0}
```

- if (EXPR) CMD

Executes the command if the expression is true.

```
Example: loop { if (ld n:1 u:32m) { px 1 }; sleep 1 }
```

- delay MICROSEC

Pauses for a specified number of microseconds. May not be interrupted by Control-C.

```
Example: loop { sd md_led0 v0; v0=v0+1; delay 25000 }
```

- sleep SEC

Pauses for a specified number of seconds. May not be interrupted by Control-C.

- time CMD

Benchmarks a specified command and displays the total execution time.

```
Example: time memtest u:32m 32m; time memtest c:32m 32m
```

- echo “STRING”

Echos a specified string.

- jump ADDR [A0 [A1]]

Loads the real R10000 a0 and a1 registers with values if specified, and sets the program counter to the specified address.

- call ADDR [A0 [A1 [A2 [A3]]]]

Loads the real R10000 a0, a1, a2, and a3 registers with values if specified, does a subroutine call to the specified address, and returns to POD prompt. The return value of the subroutine is stored in the register area as v0 and can be seen using pr v0.

```
Example: call hub_slot_get; pr v0
```

- reset

Causes the local node to reset. The reset signal propagates over CrayLinks to other nodes in the system, causing them to also reset, except the reset does not propagate past partition boundaries where the reset signal has been explicitly blocked by the operating system in the routers.

4.3.7 Non-Maskable Interrupts

- nmi NODE [a | b]

Sends a non-maskable interrupt to a specific CPU on a specific node. If the slice is not specified, sends an NMI is sent to both CPUs on the node.

- why

Displays the current exception and NMI status. If an NMI is issued or an exception occurs while the node is executing in the PROM, the node will save the exception / NMI status information, including all of the GPRs, program counter, some COP0 registers, and the exception vector address, and return to a POD prompt in Dex mode.

Example 4-1 POD Dex Hub

```
POD Dex Hub 000 1A> ld u:1      # (Misaligned memory address)
9600000000000001:
8A: *** General Exception on node 0
8A: *** EPC: 0xffffffffbfc2a9cc (load+0x48)
8A: *** Press ENTER to continue.
POD Dex Hub 000 1A> why
 EPC    : 0xffffffffbfc2a394 (load+0x48)
 ERREPC : 0x0000000000000000 (0x0)
 CACERR : 0x0000000000000000
 Status : 0x0000000024407c80
 BadVA  : 0x0000000000000001 (0x1)
 RA     : 0xffffffffbfc2a500 (exec_load+0x114)
 SP     : 0xa800000000103660
 A0     : 0x0000000000000001
 Cause  : 0xffffffff80000010 (BDSLOT INT:-------- <Load Addr Err>)
 BusAdr : 0x0000000000000001
 Reason : 242 (Unexpected General Exception.)
 POD mode was called from: 0xc00000001fc01910 (bevPanic+0xe0)
```

- fru [1 | 2]

Runs the FRU Analyzer, which reviews the current state of the error registers (see error_dump command) and attempts to determine which Field Replaceable Unit is the culprit. This may be a node card, memory SIMM, processor HIMM, router card, etc. If the argument 1 is given, the FRU Analyzer is run locally only. This should be used if it is believed the CrayLink network connection is unreliable or unconfigured. If the argument 2 is given, the FRU Analyzer takes into account the error data for all nodes.

4.3.8 Help

- help [CMDNAME]
- ? [CMDNAME]

If no argument is supplied, displays a summary of usage for all POD commands. If a command name is given as an argument, displays a summary of usage for that command only.

- version

Displays version string of the PROM that is running. For example:

```
POD Dex Hub 000 1A> version
IP27 PROM
SGI Version 1.4 Bring-up built 05:08:35 PM Jul 25, 1996
```

4.3.9 Cache

- inval [i][d][s]

Invalidates the instruction, data, and or secondary caches. Any combination of the three letters may be supplied. The contents of the caches is not written back to memory.

- flush

Causes the contents of the primary and secondary data caches to be flushed to memory.

- Cache Line Operations
  - dtag line
  - stag line
  - dline line
  - iline line
  - sline line
  - adtag line
  - aitag line
  - astag line
  - adline line
  - ailine line
  - asline line

These functions display instruction, data, and secondary cache lines, addresses, or tags.

4.3.10 TLB

- tlbc [INDEX]

Clear TLB entry, or entire TLB if no index is specified.

- tlbr [INDEX]

Read TLB entry, or entire TLB if no index is specified.

4.3.11 Operating Mode

- go dex | unc | cac

go dex flushes the data cache, invalidates all of the caches, initializes the primary data cache as a stack and data area, then returns to the POD prompt in memoryless (dirty exclusive) mode.

go unc returns to the POD prompt with all POD code, data structures, and stack in uncached memory. The area of memory to be used by the PROM is first tested, then the contents of the PROM are copied into it and verified before jumping to it.

go cac returns to the POD prompt with all POD code, data structures, and stack in cached memory. The area of memory to be used by the PROM is first tested, then the contents of the PROM are copied into it and verified before jumping to it.

See POD Mode (PROM Command Interpreter) for more information on operating modes.

4.3.12 Built-in Self Test (BIST)

- bist le | ae | lr | ar [NODE]

Performs a Hub built-in self test. If a NODE is specified, BIST is run on a remote Hub instead of the local Hub. One of four possible arguments must be specified:

```
le      Logic BIST Execute
ae      Address BIST Execute
lr      Logic BIST Results
ar      Address BIST Results
```

Executing a Hub BIST using the le or ae arguments will cause the Hub to perform the self-test, then automatically reset the node since all state in the Hub is randomized. When the node eventually returns to the POD prompt, the results can be obtained using the **lr** or **ar** arguments.

- rbist le | ae | lr | ar VEC

Performs a Router built-in self test on a Router specified by a CrayLink vector (see section Vector Addressing). One of four possible arguments must be specified:

```
le      Logic BIST Execute
ae      Address BIST Execute
lr      Logic BIST Results
ar      Address BIST Results
```

The Router BIST should be executed using the **le** or **ae** arguments. Following the test, the Router will resets itself. Then the **lr** or **ar** arguments should be used to retrieve the results of the BIST.

Enable and Disable Hardware

- disable NODE [SLICE]

Sets the environment variable DisableA or DisableB in the specified node's PROM log. If no slice is specified, both DisableA and DisableB are set. Once disabled, a CPU will not participate in the boot process and will completely prevent itself from making any Hub requests by storing a 1 into its PI_CPU_ENABLE register as soon as possible.

- enable NODE [SLICE]

Removes the environment variable DisableA or DisableB from the specified node's PROM log. If no slice is specified, both DisableA and DisableB are removed.

- tdisable NODE [SLICE]

Temporarily disables a specified CPU until the next system reset. If no slice is specified, both slice A and B are temporarily disabled. CPUs are disabled by storing 1 into their PI_CPU_ENABLE register, which completely prevents them from making any Hub requests. When the node is reset, the CPU(s) will be re-enabled.

4.3.13 Memory Test

- dirinit START LEN

Initializes a range directory memory to the Unowned state. START is an ordinary memory address that will be masked for the lower 40 bits and translated to the corresponding directory address (see L: modifier), and LEN is the length in bytes of the area to initialize. Both START and LEN must be multiples of 4096 (0x1000).

- meminit START LEN

Initializes a range of memory to zeroes. START specifies the address, which may be cached or uncached, and LEN specifies the length of the region in bytes. Both START and LEN must be multiples of 4096 (0x1000).

- dirtest START LEN

Tests a range of directory memory. START is an ordinary memory address that will be masked for the lower 40 bits and translated to the corresponding directory address (see L: modifier), and LEN is the length in bytes of the area to test. Both START and LEN must be multiples of 4096 (0x1000).

See Memory Tests for a description of the tests performed. Following the tests, the directory memory will be in an invalid state and must be re-initialized via the dirinit command before the corresponding real memory can be used.

- memtest START LEN

Tests a range of memory. START specifies the address, which may be cached or uncached, and LEN specifies the length of the region in bytes. Both START and LEN must be multiples of 4096 (0x1000).

See Memory Tests for a description of the tests performed.

- santest ADDR

Performs a sanity test of a memory address. ADDR specifies the address to be tested, which must be double-word aligned. The address is checked for readability and writability, with full error register monitoring and exception catching.

See Memory Tests for a description of the sanity test.

- maxerr COUNT

Sets the maximum number of errors that a memory test may print before it gives up and goes onto the next phase of the test. The default is 32.

- qual [1 | 0]

Turns the data quality checking features of the R10000 on or off in the PI_SYSAD_ERRCHK_EN register. This may be useful during memory testing. See section Memory Tests for more information.

- ecc [1 | 0]

ecc 1 turns the IGNORE_ECC bit off and thereby enables ECC checking. When checking is enabled, exceptions will occur when the ECC for directory memory or regular memory do not match the data stored in directory memory or regular memory.

ecc 0 turns the IGNORE_ECC bit on and thereby disables ECC checking. When ECC checking is disabled, exceptions will not occur due to ECC bad being. However, the Hub cannot be made to stop error correcting.

- error

Checks for any error bits turned on in the Hub MD section, and decodes and displays them. You might want to run this whenever the prompt contain the string MDerr, indicating there are memory / directory errors pending in the Hub.

- clear

Clears any pending error bits in the Hub MD section.

- im [BYTE]

Sets the R10000 interrupt mask field in the Status COP0 register. This affects whether certain errors cause an interrupt to be taken, especially when running memory test.

4.3.14 Memory Region

- memset DST BYTE LEN

Fills a range of memory with a byte value. The fill uses double-word stores where possible to speed the operation. DST is the start byte address, BYTE is the byte value to use, and LEN is the length in bytes.

- memcpy DST SRC LEN

Copies a range of memory. The copy uses double-word loads and stores where possible to speed the operation. DST is the destination byte address, SRC is the source byte address, and LEN is the length in bytes.

- memcmp DST SRC LEN

Compares two ranges of memory and prints whether or not they are the same. DST and SRC are the byte address of the two ranges, and LEN is the length in bytes.

- memsum SRC LEN

Adds up the bytes in a range of memory and prints the result. Uses double-word loads where possible to speed up the operation. This may be used to compare ranges of memory on different machines by comparing their byte sums. SRC is the starting byte address, and LEN is the length in bytes.

4.3.15 Directory Region

- scandir ADDR [LEN]

Scans a range of memory, showing the directory states of each location. To reduce the amount of output, only the start of each region where the directory state changes is printed. ADDR is an ordinary cache-line aligned memory address that will be masked for the lower 40 bits and translated to the corresponding directory address (see L: modifier), and LEN is the length in bytes of the area to test, defaulting to one cache line.

- dirstate [STATE [BASE [LEN]]]

Scans a range of addresses for a particular directory state.

4.3.16 Slave (Launch) Loop

- slave

Causes a CPU to jump to the IP27PROM Slave Launch Loop where it waits to be assigned a task and awoken by an interrupt sent from another CPU.

- cpu [[NODE] a | b]

Launches a specified CPU that is in the slave loop into POD Mode, and puts the launcher into the Slave Launch Loop, thereby exchanging the active CPU.

4.3.17 IO6PROM

- segs [FLAG]

Reads the IO6PROM and prints information about the segments it contains, their sizes, checksums, etc. If no flag is specified, a default flag of 1 is used. The possible flags are:

```
0       Use the copy of the IO6prom internal to the IP27prom
        instead of going out to the real IO6prom.
1       Use the real IO6prom.
```

- exec [SEGNAME [FLAG]]

Reads the IO6PROM and jumps to a specified named segment. If no SEGNAME is specified, the default io6PROM is used. If no flag is specified, a default flag of 1 is used. The possible flags are:

```
0       Use the copy of the IO6prom internal to the IP27prom
        instead of going out to the real IO6prom.
1       Use the real IO6prom.
2       Load the internal IO6prom, but don't jump to it.
        Displays the entry point address and returns to POD mode.
3       Load the real IO6prom, but don't jump to it.


Example: exec io6prom
```

4.3.18 Console Selection

- ioc3
- junk
- elsc
- talk [NODE a | b]

These commands switch the POD prompt to alternate consoles. Before switching, they probe to make sure the selected console is accessible. See POD Consoles for information on the types of consoles available. Note that **elsc** refers to the MSC.

The **talk** command, used with no arguments, switches over to the virtual netuart. With arguments, the **talk** command attempts to act as a terminal emulator and contact another node that is using the virtual netuart.

4.3.19 CrayLink Interconnect

- disc

Runs the CrayLink Interconnect discovery algorithm. This may only be run in Cac mode where memory is initialized. The internal promcfg data structure is built. This structure is a connectivity graph of all Hubs and Routers in the system. The **pcfg** command can be used to view the results. See also: **route** command (below).

- pcfg [n:NODE] [v]

Displays the contents of the promcfg data structure. This structure is a connectivity graph of all Hubs and Routers in the system. This command should only be used in Cac mode after discovery has been performed. If the v argument is supplied, only abbreviated hub information will be displayed. See also: route command (below).

- node [[VEC] ID]

Sets a node ID (or NASID). If VEC is specified, it uses a CrayLink vector write to set a remote node's NASID. This will crash the remote node, but may be useful for accessing the remote node's memory and I/O space. If VEC is not specified, the local Hub's NASID is set. This will crash the local node, but it will return to POD mode.

- rtab [VEC]

Displays the router tables in the local Hub if a vector VEC is not supplied. Otherwise, displays the router tables in the device addressed by vector VEC, whether it is a Hub or Router. See Vector Addressing for more information on vectors.

The routing table information in a Hub consists of 16 local table entries and 32 meta table entries, each containing a single 4-bit direction. The routing table information in a Router consists of 16 local table entries and 32 meta table entries, each containing 24 bits of routing information organized as 6 4-bit directions each.

See the Origin2000 Hub2 Programming Manual and the Origin2000 Router Manual for a detailed description of hardware registers.

- rstat VEC

Displays the link error and packet histogram information for a Router addressed by a specified vector VEC. See Vector Addressing for more information about vectors. See the Origin2000 Router Manual for a detailed description of hardware registers.

- hubsde

Sends a Hub data error for diagnostic testing purposes.

- rtrsde

Sends a Router data error for diagnostic testing purposes.

- chklink

Reports on the status of the local Hub's CrayLink.

4.3.20 I/O Debug

- crb [NODE]

Dumps the CRB registers from the IO section of the Hub chip, as raw hex values.

- crbx [NODE]

Dumps the CRB registers from the IO section of the Hub chip in a decoded wide format that requires 133 columns to display.

4.3.21 PROM Flash

- flash NODE [...]

Flashes a remote node's IP27 Flash PROM with a copy of the local node's IP27 Flash PROM, over the CrayLink Interconnect. See section on Flashing PROMs for more details.

The flash command must access the remote node through memory space, and as such can be used only if the CrayLink routing tables are programmed and the remote Node ID is assigned. The **route command** is often useful for achieving this.

In emergencies, the **flash** command can be used from Dex mode, but it's extremely slow. If are running in Dex mode (for example, by means of an NMI), you may want to try running from cached memory (**go cac**) before flashing remote PROM(s).

Example:

```
POD Dex Hub 000 1A> go cac
POD Cac Hub 000 1A> route 0x53 1
POD Cac Hub 000 1A> flash 1
```

4.3.22 NIC (Number In a Can)

- nic [NODE]

Reads the unique laser tag from the NIC (Number In a Can) of the local node if no Node ID is specified; otherwise reads the NIC of a remote node by accessing its memory space.

Example:

```
POD Dex Hub 000 1A> nic
0x000000000001fcba
```

- rnic [VEC]

Reads the unique laser tag from the NIC (Number In a Can) of a router, given a CrayLink vector to the router (default 0). See Vector Addressing for information about vectors.

Example:

```
POD Dex Hub 000 1A> rnic 1
0x0000000000020300
```

4.3.23 System Controller

- scw ADDR [VAL [COUNT]]

Stores byte(s) to the Origin2000 Module System Controller non-volatile RAM. The NVRAM address must be between 0 and 2047, inclusive. Addresses 1792 through 2047 are reserved for MSC use.

If neither VAL or COUNT is specified, scw enters editing mode in which the current contents of each address is displayed and the user is allowed to modify it by typing in a hexadecimal value (without the 0x). Entering nothing goes on to the next address without modifying the location. Entering a period exits editing mode.

If VAL is specified, the value is stored at the specified address. If COUNT is also specified, the value is stored COUNT times starting at the specified address.

- scr ADDR [COUNT]

Reads byte(s) from the Origin2000 Module System Controller non-volatile RAM. The NVRAM address must be between 0 and 2047, inclusive. Addresses 1792 through 2047 are reserved for MSC use.

If COUNT is specified, COUNT bytes are loaded and displayed in columns.

- sc [“command”]

Sends an arbitrary command string to the system controller for execution. The result returned by the command is displayed. See Module System Controller Commands for a description of commands that might be sent. The dbg and pas commands cannot be executed in this manner since they access the I2C bus during operation. Instead, corresponding POD commands are provided.

```
Example: sc “dsp HiWorld”
```

- dips

Displays the settings of the Hardware Debug Switches, exactly as they are set on the MSC front panel.

- dbg [VIRT_VAL PHYS_VAL]

If no arguments are specified, the values of the Physical and Virtual Debug Switches are displayed.

```
Example: dbg 0 0x18
```

This command is very similar to the MSC **dbg** command, except allows the MSC Physical and Virtual Debug Switches to be set or viewed from IP27PROM POD mode. Care should be taken because the arguments are decimal by default, rather than hex as in the MSC **dbg** command. Use the 0x prefix to set values in hex.

The above example turns off all of the Virtual Debug Switches, and turns on Physical Debug Switches 4 and 5 (since 0x18 has the fourth and fifth bits set, numbering from 1). Note that the system XORs the Physical and Hardware Debug Switches to determine the actual Physical Debug Switches, so this feature can be used to override remotely the Hardware Debug Switches.

- pas [“PASW”]

This command is very similar to the MSC **pas** command, except allows the MSC password to be set or viewed from IP27PROM POD mode.

If no arguments are specified, the current four-character system controller password is displayed.

If a password is specified, the MSC password is replaced by the specified string, which must be four characters in length. The default MSC password is none.

- module [VAL]

This command is very similar to the MSC **mod** command, except allows the module number of the module containing the current IP27 card to be set or viewed from IP27PROM POD mode. Care should be taken because the argument is decimal by default, rather than hex as in the MSC **mod** command. Use the 0x prefix to set values in hex.

If no arguments are specified, the current module number is displayed.

If a module number from 0 to 255 is specified, the module number is changed. See section Module Numbering for important information about module numbers. The **module** command does not verify the consistency of module numbers; however, the IO6PROM will repair inconsistent module numbers before allowing the system to boot.

- modnic

Displays the value of the mid-plane NIC of the module containing the current node.

4.3.24 Disassembler

- dis ADDR [COUNT]

Disassembles a region of memory into R10000 instructions beginning at the specified start address. If COUNT is specified, then COUNT instructions will be disassembled (not counting branch delay slots). If COUNT is not specified, then the disassembly will stop at the end of the function containing the start address, if applicable. Otherwise, 12 instructions will be disassembled.

- kern_sym

Toggles the set of symbols the PROM uses when translating addresses to symbols and vice-versa. The two sets of symbols available are the IP27PROM symbols and the Kernel symbols (which are not available unless a Kernel has been loaded since system boot).

- kdebug [STACKADDR]

Turns on kernel debugging functions (see Kernel Debugging) and returns to POD in a modified Dex mode in which the cache is flushed and the stack pointer is moved into memory, but the CPU continues to execute out of the PROM.

- altregs NUM

If NUM is 1, causes the PROM to stop using the set of R10000 registers that were saved on PROM entry, and instead use the set of registers saved by the Kernel debugger Symmon. If NUM is 0, causes the PROM to switch back to its regular saved registers. See Kernel Debugging.

- traplog ADDR

Displays entries from the Kernel trap log.

4.3.25 Error Dump

- dumpspool [NODE SLICE]

Dumps the PI error spool on the local Hub if NODE and SLICE are not specified. Dumps the PI error spool(s) on a remote Hub if a NODE and SLICE are specified. SLICE indicates the CPU(s), and may be A, B, or AB.

- error_dump

Dumps the error state of the local Hub. All of the error bits in the various sections of the Hub are decoded and displayed in detail.

- edump_bri [n:NODE]

Dumps the error state of a local or remote Bridge. All of the error bits in the various sections of the Bridge are decoded and displayed in detail.

PROM Log and Environment Variables

4.3.26 PROM Log and Environment Variables

- initlog [n:NODE]

Initializes the PROM Log in a new or corrupt PROM. Clears all variable entries and error log entries. If a node is specified, the operation is done on a remote node's PROM instead of the local PROM. This command may not be used on the local PROM in Dex mode since the PROM cannot be programmed while running from it.

Caution: Executing initlog on a remote PROM will crash the remote node if it is running in Dex mode (out of the PROM).

- initalllogs

Initializes the PROM Log on every node in the system. This command may only be run when the system is in Cac mode and the CrayLink network has already been fully and properly configured.

- setenv [n:NODE] VAR [STRING

Sets an environment variable in the PROM Log. If a node is specified, the operation is done on a remote node's PROM instead of the local PROM. VAR is the name of the variable and must be from 1 to 15 ASCII characters. STRING is the value, which must be from 0 to 127

ASCII characters, and usually must be enclosed in quotes. If STRING is omitted, the PROM will prompt for a string (this is useful because the PROM parser may only handle strings up to 31 characters in length). Certain variable names that are reserved words in the PROM, such as “ld” or “v0,” may also have to be enclosed in quotes. Numeric values must also be enclosed in quotes. This command may not be used on the local PROM in Dex mode since the PROM cannot be programmed while running from it.

Caution: Executing setenv on a remote PROM will crash the remote node if it is running in Dex mode (out of the PROM).

```
Example: setenv SwapBank “1”
```

See Reserved PROM Log Variables for information on specific environment variables.

- unsetenv [n:NODE] VAR

Removes an environment variable from the PROM Log. If a node is specified, the operation is done on a remote node’s PROM instead of the local PROM. VAR is the name of the variable and must be from 1 to 15 ASCII characters. Certain variable names that are reserved words in the PROM, such as **ld** or **v0**, must be enclosed in quotes. This command may not be used on the local PROM in Dex mode since the PROM cannot be programmed while running from it.

Caution: Executing unsetenv on a remote PROM will crash the remote node if it is running in Dex mode (out of the PROM).

- printenv [n:NODE] [VAR]

Displays environment variable(s) in the PROM Log. If a node is specified, the operation is done on a remote node’s PROM instead of the local PROM. If VAR is not specified, all environment variables are printed. If VAR is specified, it is the name of the variable and must be from 1 to 15 ASCII characters. Certain variable names that are reserved words in the PROM, such as **ld** or **v0**, must be enclosed in quotes.

- log [n:NODE] [SKIP [COUNT]]

Displays log entries from the PROM Log. If a node is specified, the operation is done on a remote node’s PROM instead of the local PROM. If SKIP is specified, the first SKIP entries are not displayed. If COUNT is specified, COUNT entries are displayed (otherwise, 24 entries are displayed). The output is paged to prevent it from scrolling away.

- rlog [n:NODE] [SKIP [COUNT]]

Displays log entries from the PROM Log in reverse order. If a node is specified, the operation is done on a remote node’s PROM instead of the local PROM. If SKIP is specified, the first SKIP entries are not displayed. If COUNT is specified, COUNT entries are displayed (otherwise, 24 entries are displayed). The output is paged to prevent it from scrolling away.

4.3.27 Partitioning

- setpart RTR_LIST

Sets reset partitions for partition (testing only).

4.3.28 I/O Diagnostics

- dgxbow [m<n | h | m> [n<NODE>]

Runs Crossbow diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. The second optional argument specifies a remote NASID on which to run the test.

- dgbrdg [m<n | h | m>] [n<NODE>] [s<SLOT>]

Runs Bridge diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. The second optional argument specifies a remote NASID on which to run the test. The third optional argument specifies a slot on which to run the test.

- dgconf [m<n | h | m>] [n<NODE>] [s<SLOT>]

Runs BaseIO configuration space diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. The second optional argument specifies a remote NASID on which to run the test. The third optional argument specifies a slot on which to run the test.

- dgpci [m<n | h | m>] [n<NODE>] [s<SLOT>] [p<PCI#>]

Runs PCI bus diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. The second optional argument specifies a remote NASID on which to run the test. The third optional argument specifies a slot on which to run the test.

- dgspio [m<n | h | m | x>] [n<NODE>] [s<SLOT>] [p<PCI#]

Runs serial PIO diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. If x is present, external loopback is used. The second optional argument specifies a remote NASID on which to run the test. The third optional argument specifies a slot on which to run the test.

- dgsdma [m<n | h | m | x>] [n<NODE>] [s<SLOT>] [p<PCI#]

Runs serial DMA diagnostic. The first optional two-character argument specifies the test mode as Normal, Heavy, or Manufacturing. If x is present, external loopback is used. The second optional argument specifies a remote NASID on which to run the test. The third optional argument specifies a slot on which to run the test.

4.4 Debugging with the IP27PROM

4.4.1 Reset

The hardware reset signal is generated by pressing the left button on the front panel of the Origin2000 Module System Controller, or by using the MSC rst command. All CPUs and other hardware in the module are reset. Reset may also be sent by means of software, such as when the reset command is used in POD mode.

The reset signal is propagated by hardware over the CrayLinks into other modules in the same partition. On system power-up, there are no partitions and the reset propagates throughout the entire CrayLink network. Subsequently, reset barriers are put in place to separate the system into separately resettable partitions.

4.4.2 Non-Maskable Interrupt (NMI)

NMIs are used for debugging a system that is hung, usually because of a software (PROM or Kernel) bug. An NMI to a CPU causes the CPU to stop what it's doing and jump to a fixed address in the PROM. The code at that address, called the NMI handler, attempts to bring the system to a PROM or Kernel monitor where the state of the system can be examined to determine the cause of the crash. The NMI handler is written carefully to avoid making assumptions about the state of the system and to avoid using system resources unnecessarily.

NMIs can be sent by means of software or by hardware. Software can send an NMI over the CrayLink Interconnect to any individual CPU. Pressing the right button on the front panel of the Origin2000 Module System Controller, or using the MSC nmi command, sends a hardware NMI to all of the CPUs plugged into the same module -- these CPUs will attempt to propagate the NMI to other modules in the same partition by means of a software NMI.

The NMI handler determines if the system was running in the Kernel or in the PROM at the time of the NMI by examining the NMI program counter. If the system was running in the PROM, the CPU immediately enters POD mode. Otherwise, the handler checks if a Kernel debugging facility (such as symmon) is available, and if so, transfers control to it. Sometimes the Kernel debugging facility may subsequently hang due to data corruption. For this reason, if another NMI is received (called a second NMI), the PROM will ignore the Kernel debugging facility and enter POD mode.

Occasionally, a hung Node may not respond to an NMI if the Hub or R10000 is completely hung. When a Node does respond to an NMI, a four-stage pattern flashes along the Node LEDs.

4.4.3 Kernel Debugging

There is support in the IP27PROM to dump kernel data structures info via idbg. This is especially useful if symmon does not respond to the first NMI and you use the second NMI to drop into the IP27PROM.

On NMI, all the kernel registers are saved and you enter PROM **Dex** mode. To switch to kernel debug mode, you just type **kdebug**. This will enable kernel symbols in the PROM, switch the stack to memory so that we can do cached references, and switch your register set to the one that was saved on NMI. The stack is automatically placed somewhere near 64 MB on each node, but this can be overridden with an optional address to **kdebug**.

If any exceptions are taken while printing/inspecting kernel data structures, the PROM will re-enter **Dex** mode. However, the kernel register state is still intact so just typing **kdebug** will get you back in business.

You can use **call idbg_func *args*** to dump kernel data structures. Some of these may take TLB exceptions, since TLB misses will not be processed normally. Any idbg function that does not use qprintf (and uses printf or other printing routines) to dump the information will fail.

Here are some useful commands:

- **kdebug** [optional stack address] - switch to kdebug mode.
- **altregs <num> or <Address>** - Switch to an alternate register set. A number is interpreted as (nasid << 1 | slice). An address is used as an address to the register set.
- **call idbg_xxxxx [arg1 [arg2 [arg3 [arg4]]]]**

Here are some examples:

Example 4-2 kdebug (or kdebug 0xa800000001000000 to Force sp to an Address)

```
1A 000: POD MSC Dex>kdebug
1A 000:
1A 000: *** Turning on kernel symbol table lookup
1A 000: Using Alternate register set at 0x9600000000011400
1A 000:
1A 000: *** Entering POD mode with new stack pointer on node 0
1A 000: POD MSC Dex AltRegs 0x9600000000011400> why
1A 000: EPC : 0xc000000000534378 (wait_for_interrupt+0x28)
1A 000: ERREPC : 0xc0000000004b44f4 (local_idle+0x8)
1A 000: CACERR : 0x00000000dfffffff
1A 000: Status : 0x0000000024400080
1A 000: BadVA : 0x0000000010008848 (0x10008848)
1A 000: RA : 0xc000000000534378 (wait_for_interrupt+0x28)
1A 000: SP : 0xa800000000deff88
1A 000: A0 : 0x0000000000000000
1A 000: Cause : 0x0000000000009000 (INT:8--5----)
1A 000: Reason : 253 (POD mode switch set or POD key pressed.)
1A 000: POD mode was called from: 0x0000000000000000 (0x0)
```

Example 4-3 altregs (Switches the Register Set to NASID 1, CPU 1's Register State at NMI)

```
1A 000: POD MSC Dex AltRegs 0x9600000000011400> altregs 3
1A 000: Using Alternate register set at 0x9600000100011800
1A 000: POD MSC Dex AltRegs 0x9600000100011800> why
1A 000: EPC : 0xc0000000004b44fc (local_idle+0x10)
1A 000: ERREPC : 0xc000000000534378 (wait_for_interrupt+0x28)
1A 000: CACERR : 0x00000000dfffffff
1A 000: Status : 0x0000000024400080
```

```
1A 000: BadVA : 0x000000001013fff0 (0x1013fff0)
1A 000: RA : 0xc000000000534378 (wait_for_interrupt+0x28)
1A 000: SP : 0xa800000100e03f88
1A 000: A0 : 0x0000000000000000
1A 000: Cause : 0x0000000000009000 (INT:8--5----)
```

Example 4-4 Something That symmon Cannot Do

```
1A 000: POD MSC Dex AltRegs 0x9600000000011400> a0=0x400; while(a0<0x800) {call
1A 000: pfdat: addr 0xa800000001ff0000 [1024 0x400] flags 0x2800 <dump cstale >
1A 000: pageno 0x0 tag 0x0 use 1 rawcnt 1069
1A 000: next 0x0 [-1] prev 0x0 [-1] hchain 0x0 [-1]
1A 000: Bitmap base (pure) a800000001ffff00 (tainted) a800000001fffe00
1A 000: pdep1: 0x0 pdep2: 0x0
1A 000: PFMS: 0x0
1A 000:
1A 000: pfdat: addr 0xa800000001ff0040 [1025 0x401] flags 0x2800 <dump cstale >
1A 000: pageno 0x0 tag 0x0 use 1 rawcnt 1031
1A 000: next 0x0 [-1] prev 0x0 [-1] hchain 0x0 [-1]
1A 000: Bitmap base (pure) a800000001ffff00 (tainted) a800000001fffe00
1A 000: pdep1: 0x0 pdep2: 0x0
1A 000: PFMS: 0x0
1A 000:
1A 000: pfdat: addr 0xa800000001ff0080 [1026 0x402] flags 0x2800 <dump cstale >
1A 000: pageno 0x0 tag 0x0 use 1 rawcnt 1031
1A 000: next 0x0 [-1] prev 0x0 [-1] hchain 0x0 [-1]
1A 000: Bitmap base (pure) a800000001ffff00 (tainted) a800000001fffe00
1A 000: pdep1: 0x0 pdep2: 0x0
1A 000: PFMS: 0x0
1A 000:
1A 000: pfdat: addr 0xa800000001ff00c0 [1027 0x403] flags 0x2800 <dump cstale >
1A 000: pageno 0x0 tag 0x0 use 1 rawcnt 1031
1A 000: next 0x0 [-1] prev 0x0 [-1] hchain 0x0 [-1]
1A 000: Bitmap base (pure) a800000001ffff00 (tainted) a800000001fffe00
1A 000: pdep1: 0x0 pdep2: 0x0
1A 000: PFMS: 0x0
1A 000:
1A 000: pfdat: addr 0xa800000001ff0100 [1028 0x404] flags 0x2800 <dump cstale >
1A 000: pageno 0x0 tag 0x0 use 1 rawcnt 1031
1A 000: next 0x0 [-1] prev 0x0 [-1] hchain 0x0 [-1]
1A 000: Bitmap base (pure) a800000001ffff00 (tainted) a800000001fffe00
1A 000: pdep1: 0x0 pdep2: 0x0
```

Example 4-5 idbg_runq

```
1A 000: POD MSC Dex AltRegs 0x9600000000011400> call idbg_runq 1
1A 000: Runq(0xc000000001cfb7f8): active(0xf)
1A 000:
1A 000: CPU 0:
1A 000: CPU 1:
1A 000: CPU 2:
1A 000: CPU 3:
1A 000:
1A 000: vmp 0 at 0xa800000000f34000
1A 000: vmp 1 at 0xa800000000f34080
1A 000: vmp 2 at 0xa800000000f34100
```

```
1A 000: vmp 3 at 0xa800000000f34180
1A 000:
```

Example 4-6 pb

```
1A 000: POD MSC Dex AltRegs 0x9600000000011400> call idbg_print putbuf 0xffffff:
fffffffffff
1A 000: Memfit Initialized...
1A 000: <5>NOTICE: [memsched_init]: There are 2 <= 2^1 memory nodes.
1A 000: b.0.210
1A 000: <4>WARNING: Nasid 0 interrupt bit 31 set in INT_PEND1
1A 000: <4>WARNING: Nasid 0 interrupt bit 63 set in INT_PEND1
1A 000: <4>WARNING: Nasid 1 interrupt bit 31 set in INT_PEND1
1A 000: <4>WARNING: Nasid 1 interrupt bit 63 set in INT_PEND1
1A 000: <4>WARNING: CPU 0 (/hw/module/1/slot/n1/node/cpu/0) is downrev  (925)
1A 000: <4>WARNING: CPU 1 (/hw/module/1/slot/n1/node/cpu/1) is downrev  (925)
1A 000: <4>WARNING: CPU 2 (/hw/module/1/slot/n2/node/cpu/0) is downrev  (925)
1A 000: <4>WARNING: CPU 3 (/hw/module/1/slot/n2/node/cpu/1) is downrev  000: Realtime
overflow q:
```

Example 4-7 nodenuma (Ultimately Fails Since It Uses printf)

```
1A 000: POD MSC Dex AltRegs 0x9600000000011400> call idbg_nodepda_numa 1
1A 000: Dumping node numa parameters at 0xa800000100c0c000
1A 000: memoryd semaphore @ 0xa800000100c0c008 ((noname)) count -1 flags:
1A 000: wq: 0xa800000001aa0000; no meter info
1A 000: Migration Control Parameters, (mcd)@: 0xa800000100c108b0
1A 000: Migration Per Address Space Default Kernel Parameters:
1A 000: migration_base_enabled: 0x1
1A 000: migr_base_threshold: 0x3c
.......
1A 000: migr_unpegger_npages: 27
1A 000: migr_unpegger_lastnpages: 0
1A 000:
1A 000: *** General Exception on node 0
1A 000: *** EPC: 0xc000000000532664 (0xc000000000532664)
1A 000: *** Press ENTER to continue.
1A 000: POD MSC Dex> kdebug
1A 000:
1A 000: POD MSC Dex AltRegs 0x9600000000011400
```

4.4.4 Use of the MSC

The MSC alternate console port is shared between CPUs. Upon taking an NMI or exception, each CPU prints a brief three-line message on the MSC indicating what happened, and waits for the user to press ENTER.

The user can then use the MSC **sel** command to select a particular CPU, press ENTER to get a POD prompt on that CPU, and use the **why** and **pr** commands to obtain detailed information on where the system was executing, why it stopped, and what state the CPU registers and memory were in.

4.4.5 Fast loops

Since PROM command lines are precompiled, then executed by a fast stack-based engine, a loop of register stores or loads (such as for testing I/O devices) can be executed at relatively high speed. For example:

```
loop { sd ADDR1 value1; sd ADDR2 value2 }

loop v0 = ld ADDR
```

4.5 Memory Tests

4.5.1 Bank Organization

Memory is organized in up to 8 banks of up to 512 MB each. The base address of each bank is separated by 512 MB regardless of the actual size of the bank. Thus, there is a hole in the memory address space wherever one bank follows another that's less than 512 MB. Care should be taken to avoid testing a region of memory that crosses a hole.

For example, if there are four banks of 64 MB in the system, there would be memory in the following address ranges (assuming cached space):

```
c:0     through c:(64m-1)     or    c:b:0 0 through c:b:0 (64m-1)

c:512m  through c:(576m-1)    or    c:b:1 0 through c:b:1 (64m-1)

c:1024m through c:(1088m-1)   or    c:b:2 0 through c:b:2 (64m-1)

c:1536m through c:(1600m-1)   or    c:b:3 0 through c:b:3 (64m-1)
```

4.5.2 Running the Tests

Memory tests clear the error registers before they start. During the test, if a data miscompare occurs, the contents of the error registers are decoded and displayed along with the test address and data.

If the qual on command is used, the data quality checking features of the R10000 will be enabled (PI_SYSAD_ERRCHK_EN). If data quality mode is on, then events such as ECC errors and SysAD parity errors will cause the R10000 to take exceptions during a memory test. On any exception during a memory test, the contents of the error registers are decoded and displayed along with the test address and data.

The ecc off command must be used prior to testing directory memory, or when testing regular memory but the directory memory is uninitialized, to prevent false ECC errors from occurring during the test.

Note that even when ECC checking is turned off, ECC correcting remains enabled. It is not possible to turn off error correcting in the Hub. For this reason, memory tests must be performed in the following order: test directory memory; initialize directories; test regular memory.

The standard memory test performs multiple phases:

Store / load — For each double-word address in the memory range, do a store of all 5's, and immediately load it back and compare, then do a store of all A's, and immediately load it back and compare. This test may find problems with RAM cells that take too long to change state.

A5 fill / verify — The memory range is filled with alternating double-word patterns of all 5's and all A's, then the range is read back and verified.

5A fill / verify — The memory range is filled with alternating double-word patterns of all A's and all 5's, then the range is read back and verified.

C3 fill / verify — The memory range is filled with alternating double-word patterns of 0xc3c3c3c3c3c3c3c3 and 0x3c3c3c3c3c3c3c3c, then the range is read back and verified.

Random fill / verify — The memory range is filled with pseudo-random double-words (same pattern every time), then read back and verified. This is an address test that verifies that all memory addresses are unique.

Address fill / verify — The memory range is filled with double-words corresponding to the address being written, then read back and verified. This is an address test that verifies that all memory addresses are unique. Since this is the last test, memory is left filled with incrementing double-words.

The commands for performing memory initialization and tests are:

- `dirinit`
- `dirtest`
- `meminit`
- `memtest`

It's not recommended to run memory tests in Dex mode since they so slow; they take very long and don't stress the memory. Also, tests cannot be run to cached memory while in Dex mode since the cache is already being used for POD memory.

Examples:

```
memtest u:32m 1m
```

```
memtest 0x9600000002000000 0x100000
```

These are equivalent and test one megabyte of uncached memory space starting at an offset of 32 megabytes. The first argument is the start address, uncached 32 megabytes, and the second is the length.

```
memtest c:32m 32m
```

Tests the second 32 megabytes of cached memory. Tests on cached memory are very fast. However, it doesn't make sense to test a very small region of memory since you would only be testing out of the cache.

```
memtest u:b:3 0 64m
```

Tests 64 megabytes of uncached memory in bank 3. The b: is a shorthand way to add multiples of the bank size, 512 megabytes, to the address.

```
memtest n:7 b:3 c: 1m 63m
```

```
memtest (n:7 b:3 c: 1m) 63m
```

```
memtest 0xa800000760100000 0x3f00000
```

These are equivalent and test 63 megabytes of cached memory on node (NASID) 7, beginning at the one megabyte point of bank 3.

When testing directory / protection memory, as opposed to regular memory, the dirtest command is used. It doesn't matter whether one uses a physical, cached, or uncached address. ECC checking should be turned off when testing directory memory.

The **`maxerr`** command controls the number of errors the memory test prints before it gives up and goes on to the next phase of the memory test. The default is to print 32 errors.

A final memory test available is the sanity test (**`santest`**) which tests a single address according to the following scheme:

1. Clear the error registers and verify that they clear.
2. Test that no bits in the corresponding directory memory entry LO / HI are stuck at 1 or 0, with error register checking.
3. Initialize the corresponding directory memory entry LO / HI to the Unowned state.
4. Test that no bits in the memory location are stuck at 1 or 0, with error register checking.

4.5.3 Problem Reporting

There are four general types of problems that may occur in the PROM:

- The PROM takes an unexpected exception and enters POD mode.
- The PROM hangs, but can be made to enter POD mode by sending an NMI.
- The PROM hangs, but does not respond to NMI.
- The PROM waits for something to happen that never occurs.

If either of the first two cases occur, useful debugging information can be dumped that may aid a PROM developer in quickly pin-pointing the problem. At a minimum, the output of the following POD mode commands should be captured:

```
why          <-- Prints the exception type and status registers
pr           <-- Prints all R10000 general purpose registers
error_dump   <-- Displays detailed Hub error register dump
crbx         <-- If it appears to be I/O-related
edump_bri    <-- If it appears to be Bridge-related
```

```
fru 1        <-- Runs the FRU Analyzer on the error dump data
```

The third case usually indicates the Hub chip has been hung, typically because of I/O problems, and is very difficult to debug. The only available information is the last thing the PROM printed (or the last progress LED value).

The fourth case usually indicates a CrayLink communications problem. One node may be waiting on another, but never hear from it due to link errors. Multiple nodes may believe they are the global master. It is very useful to check for extinguished Router card LEDs to see if any of the CrayLink connections have gone down, effectively isolating a CPU while the system was running (see Reading the Router LEDs).

4.5.4 Flashing PROMs

The flashing method described here should not be necessary except in case of blank systems or special emergencies. Ordinarily, the IO6PROM and IRIX **flashcpu**, **flashio**, and **flash** commands are all that is needed for flashing, but if a system can't come up to the IO6PROM or IRIX due to a PROM problem, this may come in handy.

It is possible to use one node to flash another node's PROM using the IP27PROM **flash** command. The contents of the PROM of the node doing the flash is copied into the remote node.

Caution: The remote node must perfectly match the node doing the flashing. Never copy a PROM to another node if the other node has a different secondary cache size or clock speed. The IP27PROM flash command does not automatically reconfigure PROMs as the IO6PROM and IRIX flash commands do.

1. Get to a POD prompt on the node from which the flash will be done. Try setting Hardware Debug Switches 4 and 5 to enter memoryless POD mode. Alternately, use **^Tnmi** from the MSC alternate console port part way through system initialization. Either way will result in Dex POD mode.
2. Enter cached mode using the **go cac** command. Flashing should be done from cached mode if possible because it's deathly slow in Dex mode, plus the route command sometimes has trouble changing remote nodes' NASIDs when in Dex mode.
3. Establish NASIDs and routes for the node(s) to be flashed. If the system was previously booted past global arbitration, NASIDs may already be assigned. Check the NASIDs that appear in the POD prompts coming out on the MSC ports to see if they may have already been assigned; if they are non-zero, route command(s) may be unnecessary and it may become self-evident which nodes have which NASIDs assigned.

   To establish routes manually, use the **route** command, which takes a CrayLink vector and a NASID to assign. Any NASID less than 32 will do. Multiple nodes may be routed at the same time for convenience. For example, if you're talking to node N1 and wish to flash N2, N3, and N4 within the same module, the following commands may suffice:

   - `route 0x4 1`
   - `route 0x56 2`
   - `route 0x46 3`

The **route** command may sometimes be used without arguments to automatically determine all nodes present in the system, and assign routes and NASIDs automatically in one step.

The **route** command will most likely cause the remote nodes to crash since their NASIDs are changed out from under them, but this is okay.

4. Execute the **flash** command, specifying the NASID of the node to be flashed. More than one NASID may be flashed in parallel to save time; for example,

   ```
   flash 1 2 3
   ```

After flashing, the system may be rebooted, or additional **route** and **flash** commands may be executed.

Chapter 5

PROM Log Overview

The IP27 Flash PROM is an AMD29F080 containing 1 MB of data, organized as 16 pages of 64 KB. The first 14 pages are used for storing the IP27PROM code, and the last 2 pages are used for storing the PROM log. Sectors can be flashed (erased) independently, so when new code is flashed into the IP27PROM, the last two sectors are left intact, and when the PROM log is initialized, only the last two sectors are affected.

The PROM Log is used to store three types of data:

- PROM environment variables, which consist of a key / value pair of strings.
- PROM environment lists, which consist of multiple value strings stored under the same key string.
- Error log messages, which can be dumped and viewed from the PROM and get copied into /var/adm/SYSLOG each time IRIX boots.

Variables can be set and cleared, and log entries can be added to the first (primary) sector. Space is used up each time anything is done to the PROM Log (since data can't be erased except by flashing). When the primary sector becomes full, the second (alternate) sector is flashed clean and all persistent data (variables and lists) is compacted and copied into it. The new sector then becomes the primary sector, and the old sector becomes the alternate sector. Swapping back and forth in this manner minimizes the chance that all data will be lost in case of a problem.

The **setenv**, **unsetenv**, and **printenv** POD commands are used to view and modify environment variables stored in the PROM Log. Variable keys may be up to 15 ASCII characters in length, and variable values may be up to 127 ASCII characters.

5.1 Reserved PROM Log Variables

Where the value field is numeric, it is generally stored as a decimal number, or a hex number if prefixed by 0x. The word # appears where the data is numeric.

Table 5-1 Reserved PROM Log Variables

Key	Value	Purpose
DisableA	reason	CPU A disabled if variable defined. The value is generally a description of the reason the CPU is disabled.
DisableB	reason	CPU B disabled if variable defined. The value is generally a description of the reason the CPU is disabled.
DisableMem	#...	List of memory banks (in the range 0 to 7) to disable
DisableMemSz	########	Positional list of the sizes of all disabled banks, generated automatically when memory is disabled by POD command.
DisableMemMask	#...	List of disabled banks, generated automatically when memory is disabled by POD command.
MemDisReason	reason	String indicating why memory bank(s) were disabled.
DisableIO	reason	I/O probing disabled if present. Hub I/O section will not be initialized.
SwapBank	#	Alternate bank swapped with bank 0 if bank 0 is bad. Automatically incremented if bank 0 is found to be bad.
OverrideNIC	#	Override Hub NIC, useful if NIC broken (this NIC is not the one used for software licensing).
GlobalMaster	reason	If so, node wants to be global master (if multiple have this, lowest NIC).
LastModule	#	Number of the module this node card was last plugged into, updated automatically (Origin2000 only).
Origin200Module	#	Module number for master/slave configuration (Origin200 only).
ForceConsole	-	Temporary variable automatically set if no console is found. After automatic reboot, a default console will be selected.
DisableExpress	reason	If this variable is set, express links are disabled on subsequent reboots. This is mostly useful for benchmarking the effects of express links.
NASIDOffset	#	Adds a value to the calculated NASID. May prevent successful booting -- for developer use only.
NASID	#	Variable automatically assigned during partitioning. Used to reassign NASID if one partition is rebooted.
FencePorts	#...	List of router ports to fence off. Automatically assigned and used during partitioning.

By convention, in Log entries the first part of the value should consist of the date in local time, formatted in ASCII as follows: *YY/MM/DD hh:mm:ss*. This applies only if the date is available when the entry is made. The PROM may not have a real-time clock, but the kernel may.

5.2 Error Log Keys

Log entries are to be used very sparingly to avoid filling up the log. Any given message should generally not be logged more than once per reboot. Additional message types may be added later.

When the IRIX kernel boots, it dumps the contents of the error log since the last reboot into the */var/adm/SYSLOG* file, and sets a marker entry in the log for the next reboot.

Table 5-2 Error Log Keys

Key	Value	Meaning
Fatal	string	Fatal error
Error	string	Non-fatal error
Info	string	Important info message

5.3 List Keys

Table 5-3 List Keys

Key	Value	Meaning
-	-	There are currently no list keys in use.

Chapter 6

Origin2000 Routers

Each router card has two side-by-side columns of 6 discreet LEDs, as shown in Figure 6-1.

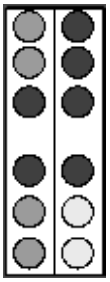

Figure 6-1 Router LEDs

The left side (green) LEDs indicate the connection status of router ports 1 through 6. They light up if a port has successfully negotiated and maintained a connection with another device, and turn off if the port is disconnected (physically or due to severe connection errors). These lights are very useful for pinpointing a cable or router Field Replaceable Unit (FRU) and should be one of the first things inspected if link(s) appear to be down. Note that although ports 4 through 6 connections are on the Origin2000 module midplane, the connection lights are still useful because an improperly seated card may not connect to the router properly.

The right side (yellow) LEDs are controlled by system software and typically light up when there is traffic across the corresponding link.

6.1 Vector Addressing

Early in the system boot, before the PROM has configured the CrayLink Interconnect, Node IDs are not yet assigned (they're all zero) and Router tables are not yet programmed. In this state, it is not yet valid to access the memory and I/O spaces of remote nodes using regular remote memory accesses (where the Node ID has been placed into physical address bits 40:32, such as by using the **n:** modifier).

The Origin2000 Hub and Routers have a feature called vector addressing which allows access to the any remote Hub's Network Interface (*NI_*) register subset, and to all Router (*RR_*) registers. To read or write a remote Hub or Router register requires specifying a path called a vector. A vector contains a string of output port numbers from 1 to 6 (corresponding directly to the hardware labeling A through F), one per nybble, appearing in reverse order. For example, the vector 0x512 indicates the request should leave the current Hub into a Router, leave port 2 of that Router into another Router, leave port 1 of that Router into another Router, and finally leave port 5 of that Router into either a Hub or Router, depending on which appears on that last port. The response automatically follows the reverse path and arrives back at the local Hub. The vector 0x0 accesses the device immediately connected to the local Hub, which is usually a Router but may be a Hub if a Null Router is being used.

POD contains several commands for performing vector operations, among them vector read (**vr**) and vector write (**vw**). Here are several examples:

vr 0 0 — Reads address 0 from the device immediately connected to the local Hub. Address 0 is the NI_STATUS_REV_ID on a Hub, and RR_STATUS_REV_ID on a Router. The status register was deliberately placed at address 0 on both Hubs and Routers, and contains a bit field that can be used to tell which type is connected.

vr 1 rr_scratch_reg0

Goes out the Hub into the connected Router, out port 1 of that Router to another Router, and reads one of that Router's scratch registers.

vw 0x512 ni_port_reset 0x80

Goes through a string of 3 Routers to reach a Hub and writes 0x80 (warm reset code) into that hub's NI_PORT_RESET register.

loop vr 0 0 — Loops repeatedly reading the status register of the immediately connected device. May be useful for getting a rough idea of the link quality since the number of polling iterations for each read is printed (if this number is not constant or is very high, the link may be performing excessive retries).

6.2 System Configurations

6.2.1 Full Router Configuration

Consider the example CrayLink Interconnect configuration shown in Figure 6-2, which is a typical fully-populated single-module system having two Routers (slots R1-R2) and four node cards (slots N1-N4). The five links shown are permanent and internal to the system mid-plane, and nothing is connected to either Router's external cable ports (1, 2, and 3). Any source node can talk to any destination node or router using vectors shown in Table 6-1.

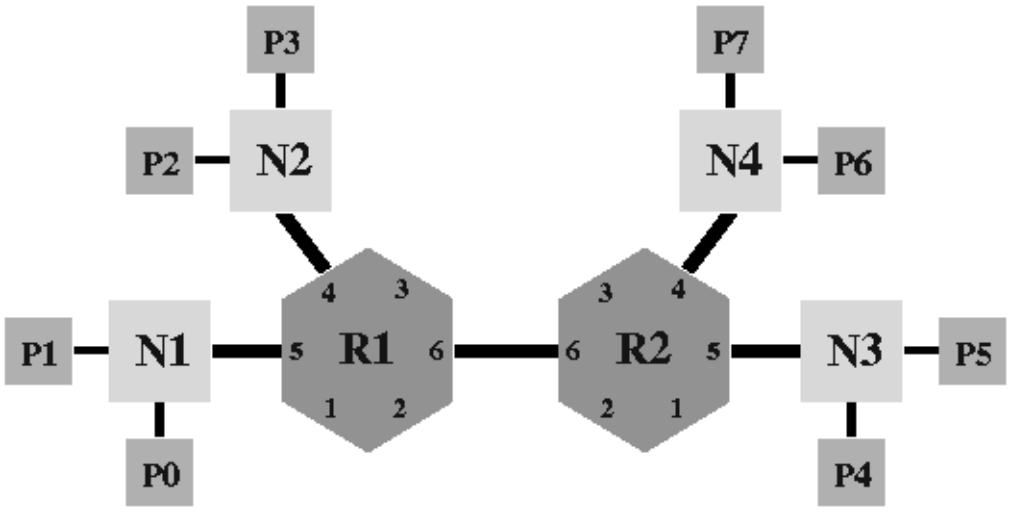

Figure 6-2 Fully Populated Origin2000 Module Connectivity

Table 6-1 Vector Routes

	Destination					
Source	**N1**	**N2**	**N3**	**N4**	**R1**	**R2**
N1	5	4	0x56	0x46	0	6
N2	5	4	0x56	0x46	0	6
N3	0x56	0x46	5	4	6	0
N4	0x56	0x46	5	4	6	0

6.2.2 Star Router Configuration (First Possibility)

A Star Router configurations is another way of populating an Origin2000 module. Figure 6-3 shows the connectivity of a fully populated Star Router configuration with the Router in slot 1 and the Star Router in slot 2. It is also possible to place the Star Router in slot 1 and the Router in slot 2 (see Figure 6-4).

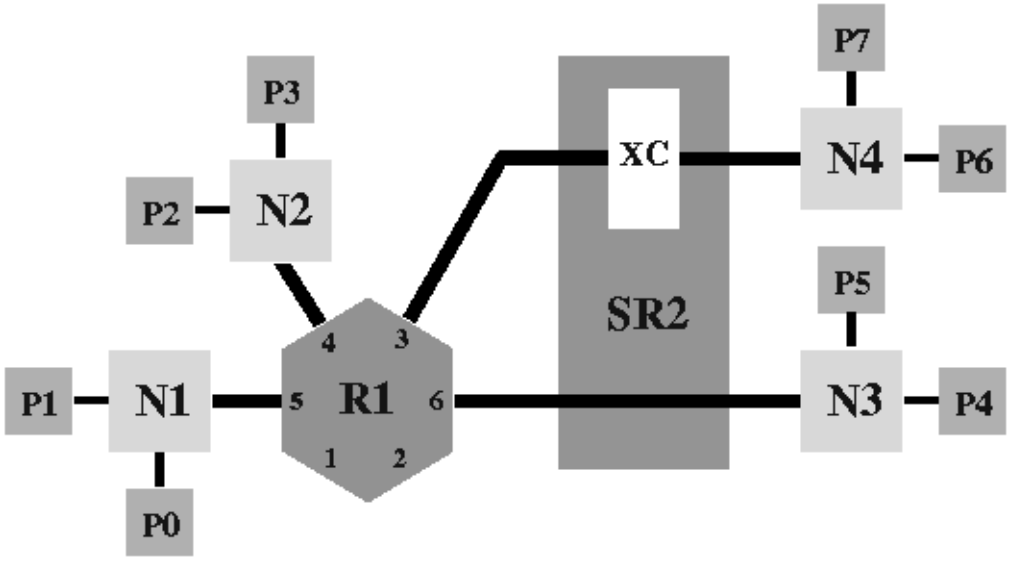

Figure 6-3 Fully-Populated Star Router Connectivity (First Possibility)

The midplane connection from R1 port 6 is internal to the system and connects directly to N3. The connection from R1 port 3 is provided by an external jumper (or cable) connecting to SR2, and is converted from differential to single-ended by the XC chip in the Star Router. The remaining ports on R1 (1 and 2) may be connected externally.

This configuration offers two advantages:

- One of the router cards is replaced by a slightly less expensive Star Router card.
- Interconnect latency is at most one hop for all messages within the module.

The disadvantage is limited scalability.

Any source node can talk to any destination node (or R1) using vectors shown in Table 6-2.

Table 6-2 Vector Routes (R1)

	Destination				
Source	**N1**	**N2**	**N3**	**N4**	**R1**
N1	5	4	6	3	0
N2	5	4	6	3	0
N3	5	4	6	3	0
N4	5	4	6	3	0

6.2.3 Star Router Configuration (Second Possibility)

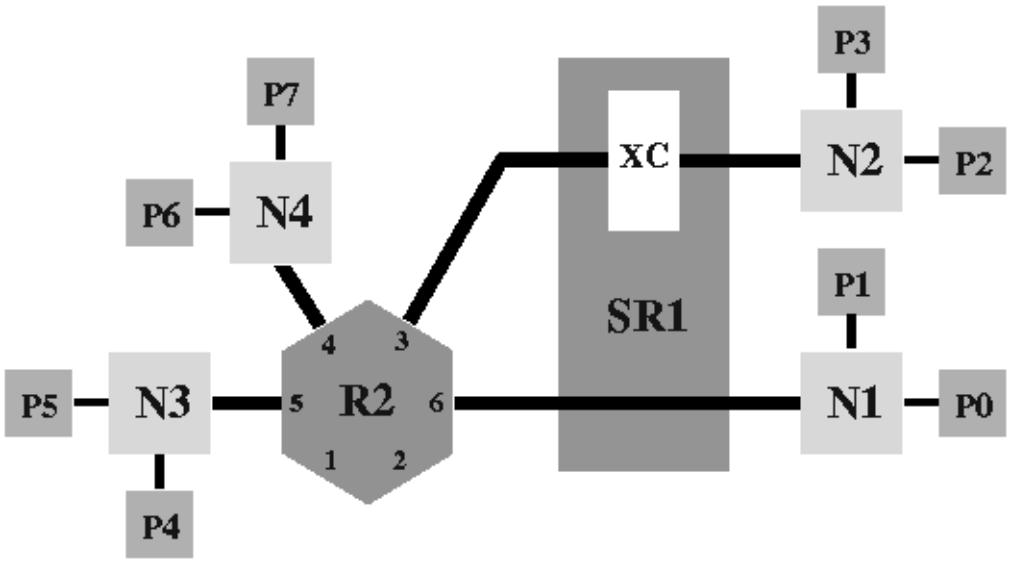

Figure 6-4 Fully-Populated Star Router Connectivity (Second Possibility)

The second Star Router configuration has the Router in slot 2 and the Star Router in slot 1. See the previous configuration for more details on the Star Router configurations.

Any source node can talk to any destination node (or R2) using vectors shown in Table 6-3.

Table 6-3 Vector Routes (R2)

	Destination				
Source	**N1**	**N2**	**N3**	**N4**	**R2**
N1	6	3	5	4	0
N2	6	3	5	4	0
N3	6	3	5	4	0
N4	6	3	5	4	0

6.2.4 Null Router Configuration

The Null Router configuration is the simplest way to create a 4-CPU system. Figure 6-5 shows the two possible connectivities of a module using the Null Router configuration.

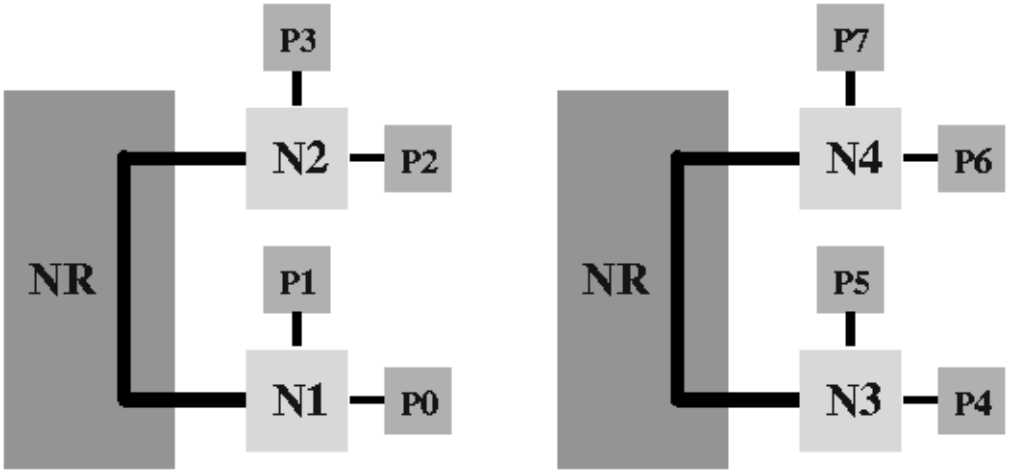

Figure 6-5 Null Router Connectivity (Two Possibilities)

In this configuration, only one inexpensive Null Router card is needed. Null Routers contain very little componentry and simply connect two Hub slots together. The system is not at all scalable, and may be only half-populated (populating both halves as two separate systems is not supported).

Either node may talk to the other using the vector 0. No other vector routes are possible.