Open Boot PROM Toolkit User's Guide

Open Boot PROM Toolkit User's Guide

Contents

About This Book

This book describes how to use the Open Boot PROM available in many SPARC products. This PROM is significantly different from PROMs in other Sun systems. It can perform many new functions and has a new user interface.

Who Should Read This Book

This guide is for Sun system administrators and field service technicians who need to use the boot PROM to do the following:

- ❑ Boot the operating system
- ❑ Run the Sun Diagnostic Executive
- ❑ Modify system start-up configuration parameters
- ❑ Perform field service troubleshooting.

Software and hardware developers may also find the information in this book useful.

Before Reading This Book

If you need to set up a system from scratch, you should read your system's installation guide.

The boot PROM has a new Forth-based command interpreter. To effectively use this new interface, it is helpful to be familiar with the Forth programming language, except for simple operations such as booting the system. See "For Further Reference " in Chapter 4 for a list of recently published Forth Language reference materials.

How to Use This Book

This book contains seven chapters and two appendixes:

- ❑ Chapter 1 is an overview of the boot PROM and the user interfaces.
- ❑ Chapter 2 describes what happens during the system start-up self-test and auto-booting sequence.
- ❑ Chapter 3 describes the Sun-Compatible Monitor interface. Refer to this chapter when you need to boot from the > prompt.
- ❑ Chapter 4 describes the basics of how the Forth Toolkit interface works. Anyone who intends to use the Forth Toolkit should read this chapter.
- ❑ Chapter 5 describes Forth tools. You will use this chapter when you are working with the Forth programming language.
- ❑ Chapter 6 describes basic machine functions. You will use this chapter when you are working with machine-specific programming issues.
- ❑ Chapter 7 describes working with NVRAM configuration parameters. You will use this chapter when you are performing typical system administration tasks.
- ❑ Appendix A compares other Sun System Open Boot PROM commands with commands used with the open boot PROM.

- Appendix B is a list of the Power-On Self-Tests (POST) with brief descriptions.

Typographic Conventions

This book uses the following typographic conventions:

- *This font* is used for emphasis, for a command argument, and for the title of a book. For example:

  You must type the *filename* argument as described in the *SunOS Reference Manual.*

- `This font` indicates a program listing, a command name, a program name, or text the machine displays on the screen, as in a tutorial session. For example:

  ```
  You have new mail.
  ```

- **`This font`** indicates what you type. Pressing the $\boxed{\text{Return}}$ key after typing the command line is an assumed action. For example:

  ```
  tutorial% date
  ```

- A rectangular box around text indicates a key name. For example:

  Press the $\boxed{\text{Return}}$ key.

  When you see two key names within one rectangular box, press and hold the first key down and then press the second key. For example:

  To press $\boxed{\text{Control-d}}$, press and hold $\boxed{\text{Control}}$, then press $\boxed{\text{d}}$.

- In a command line, square brackets indicate an optional entry and italics indicate an argument that you must replace with the appropriate text. For example:

  `cd` *[directory]*

- ❑ Toolkit commands may be typed in either upper or lower case characters. Many Toolkit commands are single character symbols. When these occur in text they are set off with quotation marks. For example:

  The "+" command adds two numbers.

Related Books

This book is part of the *SBus Developer's Kit*, a set of publications available from Sun Microsystems, Inc. The kit includes the following books:

- ❑ *SBus Specification*
- ❑ *SBus Hardware Application Notes*
- ❑ *Writing SBus Device Drivers*
- ❑ *Writing FCode Programs for SBus Cards*
- ❑ *L64853 SBus DMA Controller Technical Manual*
- ❑ *Open Boot PROM Toolkit User's Guide* (this manual).

In addition, you might find the following books useful:

- ❑ Your system's system administration & network guide
- ❑ Your system's installation guide.

1

Overview

This chapter provides an overview of:

- ❑ Open boot PROM (Programmable Read Only Memory)
- ❑ NVRAM (Non-Volatile Random Access Memory)
- ❑ PROM user interfaces.

The Open Boot PROM is very different from boot PROMs in other Sun systems. One significant change is in the user interface. The user interface and other changes are described later in this chapter. You might also see Appendix A for a list of commands that perform equivalent functions to the Sun Monitor command set provided with other boot PROMs.

Programmable Read Only Memory

The primary function of any boot PROM is to interact with the system hardware and to provide the software foundation necessary to run programs. The SunOS Operating System, the Sun Diagnostic Executive, and standalone programs all depend on the boot PROM for their initial program loading.

Another function of the boot PROM is to provide a versatile set of tools for testing the system hardware.

Non-Volatile Random Access Memory

The NVRAM contains information that is used during system boot to set up the basic machine configuration. Unlike the information contained in the open boot PROM, you can change NVRAM parameters. These changes remain in effect even when the system is turned off.

PROM User Interfaces

The boot PROM interface works in two modes: the Sun-Compatible Monitor and the Forth Toolkit (new command mode).

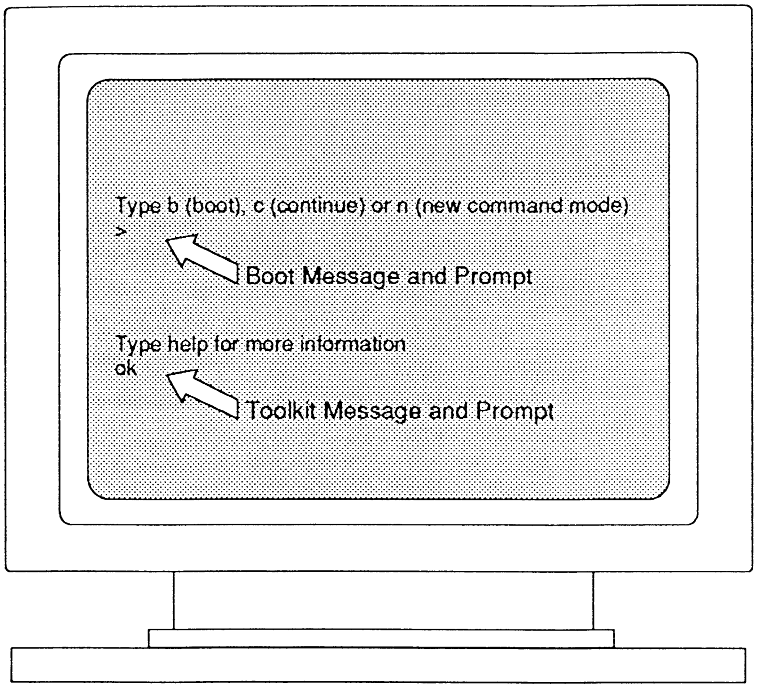
The Sun-Compatible Monitor mode is provided to present a compatible interface to the most common PROM use — booting the system.

Sun-Compatible Monitor

When you start the Sun-Compatible Monitor, the boot prompt > appears on the display screen. From the boot prompt you may execute an abbreviated set of commands. These commands allow you to boot the system, continue the execution of a halted program, or enter the Forth Toolkit.

Forth Toolkit

The Toolkit is an interactive command interpreter based on the Forth programming language. The ok prompt indicates that you are in the Toolkit. The Toolkit gives you access to an extensive set of functions for performing hardware development, problem determination (fault isolation), software development, and debugging. All functions available through the Sun-Compatible Monitor mode are also available through the Forth Toolkit.



```
Type b (boot), c (continue) or n (new command mode)
>

Type help for more information
ok
```

Where to Find What You Need

The following shows where you can find the important information in this manual:

Where	What You Will Find
Chapter 2	*System Start-Up and Auto-Booting* Overview of power-up and auto-booting sequences.

For basic booting and using the Monitor, go to Chapter 3.

Where	What You Will Find
Chapter 3	*Using the Sun-Compatible Monitor* Procedures for: 1. Starting the Monitor 2. Booting the system 3. Entering and leaving the Toolkit 4. Performing a power cycle.

For an introduction to the Forth Toolkit, go to Chapter 4.

Chapter 4 is for anyone who wishes to use the Forth Toolkit Interface.

Where	What You Will Find
Chapter 4	*Forth Toolkit Fundamentals* Forth Interface basics as they apply to the PROM Toolkit implementation of the Forth programming Language.

For Forth tools, go to Chapter 5.

For basic machine functions, go to Chapter 6.

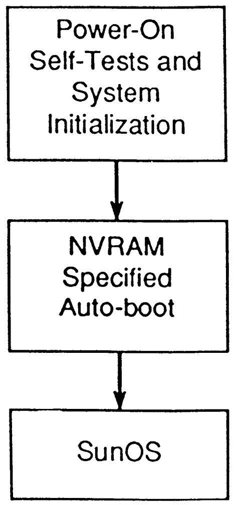
For NVRAM Configuration Parameters, go to Chapter 7.

Where	What You Will Find
Chapter 5 describes Forth programming capabilities.	**Chapter 5** *Using Forth Tools* 1. Manipulating the stack 2. Using arithmetic 3. Accessing memory 4. Searching the Forth Dictionary 5. Controlling input and output 6. Conditional testing and execution 7. Using case statements 8. Using the disassembler 9. Using loops 10. Displaying registers 11. Using breakpoints.
Chapter 6 describes basic machine control functions.	**Chapter 6** *Using Machine Functions* Procedures for: 1. Resetting the system 2. Running Diagnostics 3. Displaying system information 4. Booting from the ok prompt 5. Input/output and display modes 6. Setting up a TIP window 7. Downloading files 8. Controlling disk drives.
Chapter 7 describes working with NVRAM configuration parameters.	**Chapter 7** *Using Configuration Parameters* Procedures for: 1. Displaying and changing parameters 2. Setting security 3. Changing the power-on banner 4. Input/output control 5. Boot options 6. Controlling POST.

System Start-Up and Auto-Booting

This chapter describes the default start-up and auto-booting sequence.

Power-On Self-Test (POST)


Power-On Self-Tests and System Initialization
NVRAM Specified Auto-boot
SunOS

The power-up sequence assumes that the system Integer Unit (IU) is functional and able to fetch instructions from the Open Boot PROM. Turning on the power switch to the system unit, powering-up, resets the IU. Execution of the Power-On Self-Test (POST) sequence begins immediately.

The Open Boot PROM contains the programs for the power-on self-tests and system initialization sequence. The overall objectives of POST are to quickly verify that the system functions, initialize the system hardware, and boot the SunOS Operating System.

The POST and component initialization occur somewhat simultaneously with each component being initialized as testing completes. See Appendix B for a list of POST with brief descriptions.

Note: The POST normally performs minimum-confidence tests (not comprehensive hardware examination) prior to attempting to boot the specified software program.

For more comprehensive tests, see "diag-switch?" in Chapter 7, "Using Configuration Parameters."

Auto-Boot Procedure

When the system test and initialization are completed, the auto-boot procedure begins. Normally, the PROM attempts to auto-boot `vmunix` from the system's internal hard disk drive.

Auto-boot defaults are contained in the NVRAM configuration parameters. These parameters can be modified using the PROM Toolkit. You can change the default parameter settings to specify another program to be booted or another `boot-from` device. See Chapter 7 for procedures for modifying NVRAM configuration parameters.

You can use the L1-A key combination to access the Monitor from the login prompt.

As the power-up sequence executes, you will see status messages on the display. At the completion of an uninterrupted power-up sequence, the system's login prompt is displayed.

```
login:
```

When the system is unable to successfully complete one or more of the POSTs or auto-boot, the boot PROM outputs an error message or messages to ttya and/or the console display and attempts to start the Sun-Compatible Monitor program. If a fatal error is encountered, the program will attempt to display a message on ttya and/or the console display and will then loop on the error location. If enough of the system is functional so that the Sun-Compatible Monitor can execute, the PROM displays a brief message and the boot prompt.

```
Type b (boot), c (continue), or n (new command mode)
>
```

Chapter 3 describes the functions available from the > prompt. Chapter 4 is an introduction to the Forth Toolkit interface. Chapters 5, 6, and 7 describe using the Forth Toolkit.

3

Using the Sun-Compatible Monitor

This chapter explains how to access the boot PROM interface and use the Sun-Compatible interface — commonly called the Monitor. The Monitor supports three commands that allow you to boot the system, continue a halted program, and enter the Forth Toolkit.

Starting the Monitor

The boot PROM interface operates independently from the SunOS Operating System. Figure 3-1 lists the three ways you can start the interface.

Because boot PROM commands can modify any location in memory, it is possible to enter commands incorrectly so that the PROM is unable to execute what you've entered and becomes *hung*. The system stops responding to input from the keyboard. In that case, your only alternative is to perform a power cycle to bring the system back to normal operation. Once you perform the power cycle, you can interrupt the power-up sequence to return to the command interpreter.

When performed as described on the following pages, a power cycle will not produce any adverse effects on your system.

Figure 3-1. Starting the Boot PROM Interface

Method	Procedure
Performing a Power-Cycle and Interrupting Power-Up Sequence	1. If necessary, turn the power to the system unit off and wait 10 seconds 2. Turn on the power to the display (if necessary) 3. Turn on the power to the system unit, and wait several seconds 4. When the word `Testing` appears on the screen, press $\boxed{\text{L1-A}}$.
Halting SunOS Operating System (normal method)	1. Save and quit all open files 2. Quit all applications 3. In a shell window, become the system superuser and type: **`/etc/halt`**
Aborting SunOS Operating System (if hung system)	1. Press $\boxed{\text{L1-A}}$. 2. At > prompt, type **`n`** 3. At `ok` prompt type **`sync`** 4. Press $\boxed{\text{L1-A}}$ again when you see the word `rebooting`. 5. At `the ok` prompt type **`old-mode`** to return to the > prompt (if desired)

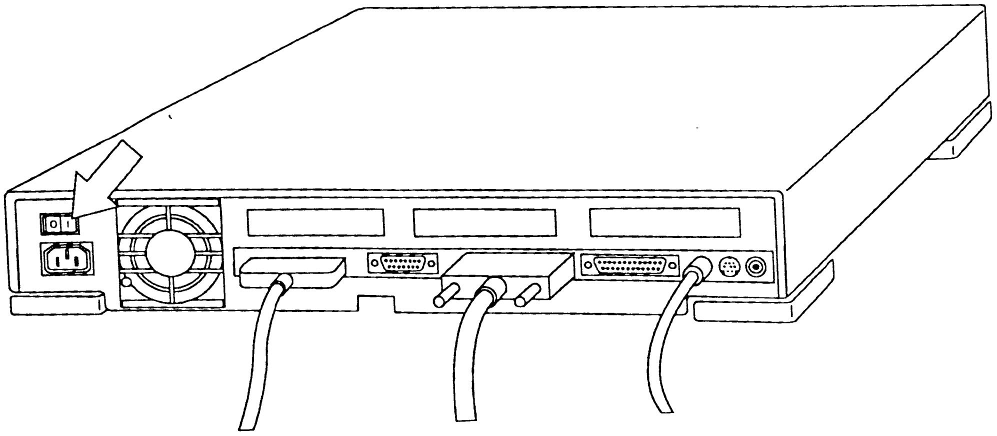
Performing a Power Cycle

When your system becomes *hung,* a power cycle is necessary to return the system to normal operation.

To perform a power cycle:

1. Turn off the power to the system unit (use the main power switch on the back of the system unit).

   The following drawing shows the location of the power switch on the SPARCstation 1. The location of the power switch for other systems might be different.



2. Wait a minimum of 10 seconds.
3. Turn the power back on.

Caution: Always allow 10 seconds between turning off the power and turning it back on again. This pause prevents possible damage to power supply components in your system unit.

Interrupting Power-Up Sequence

The most common way to start the PROM interface is to interrupt the power-up sequence. You can interrupt the power-up sequence anytime you turn the system unit on, or when you reset the system from the keyboard.

To interrupt the power-up sequence (assuming the system is powered off):

1. Turn on the power to the display.
2. Turn on the power to the system unit.

   Locate the power toggle switch on the back of the system unit. Press the side of the switch labeled 1.
3. After the word "Testing" appears on the display, press the L1-A keys simultaneously.

   The power-up sequence halts and the system displays a brief message and the boot prompt > .

```
Type b (boot), c (continue), or n (new command mode)
>
```

Halting the Operating System

To start the boot PROM interface when the SunOS Operating System is running, you must first halt the execution of SunOS. Halting SunOS should be done carefully. When you halt the SunOS Operating System, the Monitor program starts automatically.

When the system is running the SunOS Operating System, you should see a machine prompt in an open shell window that looks something like this:

```
hostname%
```

To halt the operating system and start the user interface:

1. Save and quit all open files. See the *Sun System User's Guide* for information about ending a work session.
2. Quit all open applications.
3. Become superuser as described in the *Sun System Network Manager's Guide,* Chapter 2. Type `/bin/su` and press Return.
4. Type `/etc/halt` and press Return.

The system displays system halt messages followed by the boot prompt.

```
hostname% /bin/su
Password:
hostname# /etc/halt
Syncing file systems . . . done
Halted

Type b (boot), c (continue), or n (new command mode)
>
```

Aborting a Hung System

When the operating system appears to be running but the system does not respond to the mouse and/or keyboard, the system is hung. When you abort a hung system, the PROM user interface automatically starts. If the following sequence does not work — that is, if the system does not respond to the abort attempt — perform a power cycle to return the system to normal operation. If a power cycle does not restore normal system function, call your field service representative for further assistance.

To abort a hung system and start the PROM user interface:

On some keyboards, [L1] appears on the front face of the [Stop] key.

1. Press [L1-A].
2. Type n and press [Return].

   The system displays a help message and an ok prompt.
3. Type sync and press [Return].
4. Press [L1-A] again when you see the word rebooting.
5. Type old-mode and press [Return], to return to the > prompt (if desired).

```
Press [L1-A]
Type b (boot), c (continue) or n (new command mode)
> n
Type help for more information
ok sync
When you see the word rebooting, press [L1-A] again
ok old-mode
Type b (boot), c (continue) or n (new command mode)
>
```

The `sync` command helps prevents the system from losing data that was not preserved when the system hung.

Caution: When the operating system or any other standalone program has already booted, it is preferable not to use L1-A to halt the machine. Aborting program execution with L1-A can cause damage to currently open data files.

Compatible Monitor Functions

The boot PROM Sun-Compatible Monitor mode presents a compatible interface to the most common PROM use, booting the system. All functions available through this mode are also available through the Forth Toolkit.

You can disable Sun-Compatible Monitor mode using NVRAM parameters. See Chapter 7 for information about modifying NVRAM configuration parameters.

Three commands are supported by the Sun-Compatible Monitor mode. These commands are `b` for booting the system, `c` for continuing the execution of a halted program, and `n` for entering the new command mode called the Forth Toolkit. The `c` and `n` are single character commands only. However `b` supports the standard booting command syntax.

Booting From the > Prompt

The boot command loads the SunOS Operating System or another executable program into memory and executes that program when the program load completes.

All booting operations function identically, whether you are in Sun-Compatible Monitor mode or in the Forth Toolkit. The only difference is that you must type out the entire word "boot" (with a following space if options are used) when you are in the Toolkit.

To boot your system, enter a boot command. See the next section "Boot Command Syntax" for the boot command format and the options summary in Figure 3-2 for further details. Syntax for both the > prompt and the Toolkit `ok` prompt is shown in the following examples.

Sun Compatible Monitor >	PROM Toolkit `ok`	Description
`b`	`boot`	Boot system using defaults
`b -as`	`boot -as`	Boot sd0 with flags `-a` (interactive flag) and `-s` (single-user operation)
`b le()`	`boot le()`	Boot `vmunix` from the network
`b net`	`boot net`	Boot `vmunix` from the network
`b sd(0,0,2)mydiag`	`boot sd(0,0,2)mydiag`	Boot `mydiag` from SCSI drive partition 2

Note: Boot defaults can be changed using NVRAM configuration parameters. The NVRAM defaults are only used if the boot command has no arguments. See Chapter 7 for information about changing defaults.

Boot Command Syntax

The syntax of the boot command follows. Spaces and tabs typed in the command line are ignored. All arguments shown in italics are optional. When using command options, the command word `boot` must be followed by a space.

> **b** *[device (c,u,p) filename options]*

ok **boot** *[device (c,u,p) filename options]*

Figure 3-2 lists the boot commands and their syntax.

Figure 3-2. Boot Option Commands

Option	Description
device is one of:	`net` or `le` *(c,u,p)* LANCE Ethernet `disk` or `sd` *(c,u,p)* SCSI hard disk `tape` or `st` *(c,u,p)* SCSI tape `fd` *(c,u,p)* 3-1/2" diskette drive *c* Controller Number, default value = 0. *u* Unit Number, default value = 0; when booting from a hard disk the range may be from 0-3. *p* Partition Number, default value = 0; when booting from a hard disk the range may be from 0-7. When using `le`, `sd` and `fd` as device identifiers, the parentheses are required in the command line. Example: `ble()` or `ble(0,0,0)`. The contents of the parentheses depends on the specified device.
filename	Default = `vmunix`. The name of the program to be booted, such as `stand/diag` or `vmunix`. *filename* is relative to the root of the selected device and partition (if specified). *filename* never begins with '/'. If *filename* is not given, the boot program uses the default file name `vmunix`.
options	-a Prompts interactively for the device and name of the file to boot. -b Pass the -b flag through the kernel to init (8) to skip execution of the `/etc/rc.local` script. -h Halt after loading the program. -s Pass the -s flag through the kernel to init (8) for single-user operation. -i *initname* Pass the -i *initname* to the kernel to tell it to run *initname* as the first program rather than the default `/single/init`.

Continuing a Halted Program

The c command is useful if you have halted your SunOS Operating System or another program by pressing L1-A. To resume execution of a halted program:

```
Type b (boot), c (continue), or n (new command mode)
> c
```

Program execution resumes. Once execution has resumed, you can choose Redisplay All from the SunView menu to refresh the display and remove any screen artifacts.

Note: From the ok prompt, the command go performs the same function as typing c at the > prompt.

Entering The Forth Toolkit

To enter boot PROM Forth Toolkit mode from the > prompt, type:

```
Type b (boot), c (continue), or n (new command mode)
> n
Type help for more information
ok
```

The monitor enters the Forth Toolkit and displays the ok prompt and help message.

Returning to the > Prompt

After entering the Toolkit, all the functions available from the boot prompt are available from the ok prompt. However, should you wish to exit the Toolkit and return to the > prompt, enter:

```
ok
ok old-mode
Type b (boot), c (continue), or n (new command mode)
>
```

The Sun-Compatible Monitor supports a very abbreviated set of functions. From the > prompt you can boot the system, enter the Forth Toolkit, or continue the execution of a halted program.

After entering the Forth Toolkit, you can work closely with your system's hardware.

The remaining chapters describe using the Forth Toolkit. Even if you are familiar with the operation of the Forth programming language, we recommend that you read Chapter 4 which describes how Forth is implemented in the Open Boot PROM.

Forth Toolkit Fundamentals

This chapter introduces Forth as it is implemented in the Open Boot PROM. Even if you are already familiar with the Forth programming language, we recommend that you read this chapter because it contains useful information that relates specifically to your system.

While it is impossible to provide a complete tutorial on the Forth language here, this chapter covers enough of the basics to enable you to use the Toolkit. To use this chapter to its fullest advantage, work through the examples shown in the gray screens. These examples will help you understand how the interface operates.

For additional information, see "For Further Reference" at the end of this chapter. In addition, Chapter 5 "Using Forth Tools" describes the Forth programming language capabilities.

Note: As mentioned previously, it is possible to enter commands at the `ok` prompt that cause the system to become hung. If this happens, you might need to perform a power cycle to return the system to normal operation.

This chapter assumes that you have read Chapter 3 and are familiar with how to enter and leave the Forth Toolkit from the Sun-Compatible Monitor.

Forth Commands (Words)

Forth has a very simple command structure. Forth commands, also called Forth *words*, consist of any combination of printable characters — for example, letters, digits, or punctuation marks. All of the following are examples of legitimate words:

```
@     dump     .     0<     +      test-memory
```

In this chapter, the terms word and command are used interchangeably.

Words must always be separated by one or more spaces (blanks) in order to be recognized. Press Return at the end of any command line to execute the typed command(s). In all examples shown, a Return at the end of the line is assumed.

Multiple words on a line are simply executed one at a time, from left to right, in the order in which they were entered (from left to right). For example:

```
ok testa testb testc
ok
```

is exactly equivalent to:

```
ok testa
ok testb
ok testc
ok
```

In this implementation of Forth, upper-case and lower-case letters are equivalent. Therefore, `testa`, `TESTA`, and `TesTa` all invoke the same command.

Commands that may generate large amounts of output, such as `dump` or `words`, can be interrupted by pressing any key. At that point, output is suspended and the following message appears:

```
More [<space>,<cr>,q] ?
```

Press the space bar to continue, press Return to output one more line and pause again, or type `q` to abort the command. When you are generating more than one page of output, the system will automatically enter this prompt after every page.

Getting Help

Whenever you see the ok prompt on the display, you can ask the system for help by typing one of the help commands. For example:

```
ok help dump
       Category: Memory access
dump ( addr length -- ) display memory at addr for length bytes
ok
```

The help command displays instructions on how to use the help system and lists the available help categories.

ok **help** *category*

This command shows the help messages for all commands available in the selected category, or possibly a list of sub-categories.

ok **help** *name*

This command shows the help for the named command.

Note: Because there are a very large number of command words, help is available for the most frequently used commands only.

Figure 4-1 lists commands which describe the help and change functions.

Figure 4-1. Help and Mode Commands

Command	Stack Diagram	Description
help	(--)	List main help categories.
help *category*	(--)	Show help for all commands in the category. Use only the first word of the category description.
help *name*	(--)	Show help for individual command (where available).

Numbers

Numbers are entered simply by typing in the value, for example, 55 or -123. Forth accepts only integer (whole) numbers; fractional values such as 2/3 or 5.77 are not allowed. Be sure to use one or more spaces to separate numbers from words or from each other.

The Forth toolkit performs 32-bit integer arithmetic, and all numbers are 32-bit values unless otherwise specified. Because hexadecimal (base 16) numbers are so commonly used, the Forth Toolkit automatically interprets all numbers in hexadecimal, not decimal. Therefore, adding 8 and 7 returns the value `f`, not `15`. However, you can change the operating number base.

To operate in decimal (base 10), type the following command:

```
ok decimal
ok
```

To change back to hexadecimal (base 16) type:

```
ok hex
ok
```

To find out what number base is currently active, type:

```
ok 10 .d
16
ok
```

See "Numeric Input and Output in Different Bases" in Chapter 5 for more information and additional commands regarding hexadecimal versus decimal numeric conversion.

The Stack

The Forth *stack* is a last-in, first-out buffer used for temporarily holding numeric information. Think of it as a stack of books: the last one you put on the top is the first one you take off. *Understanding the stack is essential to using the Forth Toolkit.*

To place a number on the stack, simply type its value.

```
ok 44   The value 44 is now on top of the stack
ok 7    The value 7 is now on top, with 44 just underneath
ok
```

Showing the Stack With showstack

The contents of the stack are normally invisible until needed. However, properly visualizing the current stack contents is important for achieving the desired result.

To show the stack contents with every ok prompt, type:

```
ok showstack
44 7 ok  8
47 7 8 ok
```

Top of the Stack

Remember, the topmost stack item is always shown on the right side of the list.

Once invoked, showstack will remain in effect until a machine reset takes place.

Nearly all words that require numeric parameters will fetch those parameters from the top of the stack. Any values returned are generally left on top of the stack, where they may be viewed or consumed by another command.

For example, the Forth word + removes two numbers from the stack, adds them together, and leaves the result on the stack.

To add two numbers from the top of the stack, type the addition operator + like this:

```
44 7 8 ok +
44 f ok +
53 ok                    Remember, all arithmetic is in hex
```

Once the two values are added together, the result is put onto the top of the stack. The Forth word . removes the top stack item and displays that value on the screen. For example:

```
53 ok 12
53 12 ok .
12
53 ok .
53
ok                       The stack is now empty
ok 3 5 + .
8
ok                       The stack is now empty
```

Stack Diagram

Because knowing the stack usage is vital to the proper operation of all Forth words, there is an associated *stack diagram* in the form (--) for every defined word. The stack diagram specifies what happens to the stack with the execution of the command word.

For example, the stack effect diagram for the + word is (n1 n2 -- n3). The stack effect diagram for . is (n --).

Any entries *before* the -- show stack items that are consumed, that is removed from the stack and used by the operation of that word. Any entries *after* the -- show stack items that are left on top of the stack after the word is finished executing.

Therefore, + removes two numbers and then leaves the sum on the stack. The word . simply removes one number and displays it.

Any word that has no effect on the contents of the stack, such as `showstack` or `decimal`, will have a (--) stack effect diagram. These words may be executed at any time, with no effect on the contents of the stack.

Occasionally, a word will require another word or other text immediately following, such as the `see` word, used in the form `see` *anyword*. The word `see` has no stack effect. The stack diagram would be:

`see` *anyword* (--)

Colon Definitions

Forth provides an easy means to create custom definitions for new command words. These are called *colon definitions*, named after the : word used to create them. For example, suppose you wish to create a new word `add4` that will add any four numbers together and display the result. The definition could be created as follows:

```
ok : add4  + + + .  ;
ok
```

The ; (semi-colon) marks the end of the definition that defines `add4` to have the behavior (+ + + .). The three pluses reduce the four stack items to a single sum on the stack, and then the . removes and displays that result.

```
ok 1 2 3 3 + + + .
9
ok 1 2 3 3 add4
9
ok
```

Definitions are stored in local memory, which means they are forgotten if a machine reset takes place. To keep useful definitions, either jot them down (for short ones), or create a text file (using your favorite text editor under SunOS) containing the definitions. This text file can then be downloaded whenever it is needed. See "Downloading Files" in Chapter 6 for more information.

When you type a definition in the Toolkit, the `ok` prompt becomes a `]` (right square bracket) prompt after you type the : (colon) and before you type the ; (semi-colon). For example, you could type the definition for add4 like this:

```
ok : add4
]  + + +
]  .
]  ;
ok
```

Every definition you create (in a text file) should have a stack effect diagram shown with that definition, even if the stack effect is nil (--). This is vital because the stack diagram tells you how that word is properly used. We also recommend that you use generous stack comments, within the middle of complex definitions, to help trace the flow of execution.

For example, when creating add4, it might be defined as:

```
: add4  ( n1 n2 n3 n4 -- )  + + + .  ;
```

or

```
: add4  ( n1 n2 n3 n4 -- )
   + + +  ( sum )
   .
;
```

Note: The (open parenthesis is a Forth word meaning to ignore the following text, up to the closing parenthesis). Like any other Forth word, the open parenthesis must have one or more following spaces.

Keyboard Editor

An EMACS-style (one of the text editors available on Sun systems) keyboard line editor and history mechanism is also provided with the Forth Toolkit. This powerful tool enables you to re-execute previous commands without retyping them, and allows editing of the current command line to fix typing errors or to edit previous commands.

The line editing commands listed in Figure 4-2 are available for your use when you are typing commands to the Forth Toolkit `ok` prompt.

These commands are control and escape key combinations.

Using Control Key Combinations

To execute a control key combination:

1. Press and hold down the Control key.
2. Type the desired character key.

Using Escape Key Combinations

To execute an escape key combination:

1. Press *and release* the Esc key.
2. Type the desired character key.

As you review the list of commands, notice that there are commands for the following:

- ❑ Moving forward and backward on the command line
- ❑ Erasing characters, words, all or a portion of the command line
- ❑ Recalling the most recently typed command lines; repeatedly pressing Control-p will recall previous commands (at least 8 are remembered).

To insert text at the cursor, simply type normally. Pressing Return sends the line (as it currently appears) out for execution.

While a small effort is required to learn this function, it will save you time and effort every time you use the Forth Toolkit.

Figure 4-2. Line Editor Commands

Command	Description
Control-b	Backward one character
Esc b	Backward one word
Control-f	Forward one character
Esc f	Forward one word
Control-a	Beginning of line
Control-e	End of line
Control-h	Erase previous character (also Del or Back Space)
Esc h	Erase previous portion of word (also Control-w)
Control-d	Erase this character
Esc d	Erase this portion of word, from here to end of word
Control-k	Erase forward, from here to end of line
Control-u	Erase entire line
Control-l	Retype line
Control-q	Quote next character (to type a control-character)
Control-p	Recall previous command line
Control-n	Recall subsequent command line

For Further Reference

For further reading, see one or more of the following reference materials:

Mastering Forth
Anita Anderson and Martin Tracy
Brady Communication Company, Inc.
1230 Avenue of the Americas
New York, New York

Mastering Forth is particularly useful, because the Forth dialect it describes quite closely resembles the implementation of Forth in the boot PROM.

Starting Forth
Leo Brodie/Forth, Inc.
Prentice-Hall Software Series
Englewood Cliffs, New Jersey 07632

Starting Forth is a popular and well-written book. The second edition describes the current Forth standard dialect, Forth 83.

Note: There are several differences between the versions of Forth as described in the reference materials and the version described in this guide. Specifically, the boot PROM Forth Toolkit uses 32-bit numbers instead of 16-bit numbers. The editors, described in these books, do not apply.

This chapter presented a brief overview of how to use the Forth Toolkit interface. The next three chapters describe many useful Forth commands. Chapter 5 provides information about using the Toolkit and the Forth language. Chapter 6 describes machine-specific issues for writing programs that interact with your system's hardware. Chapter 7 describes NVRAM configuration parameters and how to change them.

5

Using Forth Tools

This chapter provides an overview of how to use the many functions provided by the Open Boot PROM's Forth Toolkit. These descriptions are intended to help you get started using this Forth implementation to its fullest capacity. However, you may find that you need more specific information concerning the Forth programming language. For further information, consult any Forth tutorial or reference book, or see "For Further Reference" at the end of Chapter 4 for a short list of Forth Language publications.

In this chapter you will find information about:

- ❑ Manipulating the stack
- ❑ Using numeric input and output in different bases
- ❑ Using arithmetic
- ❑ Accessing memory
- ❑ Searching the Forth dictionary
- ❑ Controlling text input and output
- ❑ Using conditional testing
- ❑ Controlling conditional execution

- ❑ Using conditional and counted loops
- ❑ Using case statements
- ❑ Using defining words
- ❑ Compiling the dictionary
- ❑ Using the disassembler
- ❑ Displaying registers
- ❑ Using breakpoints.

This chapter assumes that you are familiar with the boot PROM's Forth Toolkit interface. With the exception of the NVRAM parameter commands, which should only be used with caution, all the commands that are described in this guide can be freely executed at any time. Remember, you can either enter commands at the `ok` prompt or type them into ASCII text files for downloading and execution.

The *SPARC Open PROM Toolkit Reference Summary*, which comes with your *SBus Developer's Kit*, provides a list of all commands explained in this book.

Showing the Stack

For all examples shown in this chapter, `showstack` is enabled. Every `ok` prompt is immediately preceded by a display of the current contents of the stack. Every example will work just the same if `showstack` were not enabled, except that the values immediately before each `ok` will not be shown. See "The Stack" in Chapter 4 for information about the `showstack` command.

Using 32-Bit Numbers

The Forth interpreter implemented in the boot PROM adheres closely to the Forth 83-Standard in most respects. One major exception is that the boot PROM Forth implementation uses 32-bit numbers instead of 16-bit numbers. In most cases, this difference will be transparent to the user. For example, `@` and `!` (described later in this chapter) work with variables as expected. If you explicitly want a 16-bit fetch or a 32-bit fetch, use `w@` or `L@` instead of `@`. Other commands also follow this convention.

Manipulating the Stack

Stack manipulation commands allow you to add, delete and reorder items on the stack. In most cases, the stack effect diagram fully defines the behavior of the word. A typical use of stack manipulation might be to display the top stack item while preserving all stack items as shown in the example below:

```
5 77 ok dup    Duplicates the top item on the stack
5 77 77 ok .   Removes and displays  the top stack item
77
5 77 ok        The stack is now the same as before
```

Figure 5-1 on the next page lists stack manipulation commands.

Figure 5-1. Stack Manipulation Commands

Command	Stack Diagram	Description
`clear`	(??? --)	Empties the stack
`depth`	(-- +n)	Returns the number of items on the stack
`drop`	(n –)	Removes top item from the stack
`2drop`	(n1 n2 --)	Removes 2 items from the stack
`dup`	(n – n n)	Duplicates the top stack item
`2dup`	(n1 n2 -- n1 n2 n1 n2)	Duplicates 2 stack items
`3dup`	(n1 n2 n3 -- n1 n2 n3 n1 n2 n3)	Duplicates 3 stack items
`?dup`	(n – n n | 0)	Duplicates 1 top stack item if it is non-zero
`nip`	(n1 n2 -- n2)	Discards the second stack item
`over`	(n1 n2 -- n1 n2 n1)	Copies second stack item to top of stack
`2over`	(n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)	Copies second 2 stack items
`pick`	(+n -- n2)	Copies +n-th stack item (1 pick = over)
`>r`	(n –)	Moves a stack item to the return stack (use with caution)
`r>`	(-- n)	Moves an item from the return stack to the stack (use with caution)
`r@`	(--n)	Copies the top of the return stack to the stack (use with caution)
`roll`	(+n --)	Rotates +n stack items (2 roll = rot)
`rot`	(n1 n2 n3 -- n2 n3 n1)	Rotates 3 stack items
`-rot`	(n1 n2 n3 -- n3 n1 n2)	Inversely rotate 3 stack items
`2rot`	(n1 n2 n3 n4 n5 n6 -- n3 n4 n5 n6 n1 n2)	Rotates 3 pairs of stack items
`swap`	(n1 n2 -- n2 n1)	Exchanges the top 2 stack items
`2swap`	(n1 n2 n3 n4 -- n3 n4 n1 n2)	Exchanges 2 pairs of stack items
`tuck`	(n1 n2 -- n2 n1 n2)	Copies the top stack item underneath the second item

Numeric Input and Output in Different Bases

The commands `hex` and `decimal` cause all subsequent numeric input and output to be performed in base 16 or base 10, respectively. `d#` and `h#` are useful for inputting a number in the other base, without having to explicitly change the base. For example:

```
ok decimal                   Change base to decimal
ok 4 h# ff 17 2
4 255 17 2 ok
```

`.d` and `.h` act like `.` but display the value in decimal or hex, respectively, regardless of the current base setting. For example:

```
ok hex
ok ff .  ff .d
ff 255
ok
```

`.s` displays the entire stack contents without disturbing them. It can be safely used at any time for debugging purposes. This is the function that `showstack` performs automatically.

Large numbers such as `fffffff` are sometimes shown with a decimal point — that is, `fff.ffff`. This is done solely for readability. The Toolkit ignores the decimal point.

Figures 5-2, 5-3, and 5-4 list commands to control numeric input and output.

Figure 5-2. Changing the Numeric Base

Command	Stack Diagram	Description
`base`	(-- adr)	Variable containing number base
`d# number`	(-- n)	Interpret the next number in decimal; base is unchanged.
`decimal`	(--)	Set number base to 10
`h# number`	(-- n)	Interpret the next number in hex; base is unchanged.
hex	(--)	Set the number base to 16

Figure 5-3. Displaying Output

Command	Stack Diagram	Description
`.`	(n --)	Display a number in the current base
`.d`	(n --)	Display n in decimal without changing base
`.h`	(n --)	Display n in hex without changing base
`.r`	(n size --)	Display a number in a fixed width field
`.s`	(--)	Display contents of data stack
`showstack`	(--)	Automatically shows stack items before ok prompt
`u.`	(u --)	Display an unsigned number
`u.r`	(u size --)	Display an unsigned number in a fixed width field

Figure 5-4 lists "primitives" used to create numeric display words, such as `.` `u.` or `.r`. They are not normally needed.

Figure 5-4. Output Display Primitives

Command	Stack Diagram	Description
`<#`	(--)	Initializes pictured numeric output
`#`	(+n1 -- +n2)	Converts next digit
`#s`	(+n1 -- 0)	Converts remaining digits
`hold`	(char --)	Inserts character into pictured output
`sign`	(n --)	Inserts sign into pictured output
`#>`	(n -- adr len)	Ends pictured output, leaving string ready to `type`.
`(.)`	(n -- adr len)	Converts a number into a string, ready to `type`.
`(u.)`	(u -- adr len)	Converts an unsigned number into a string, ready to `type`.

Using Arithmetic

Forth provides a variety of basic arithmetic functions. The commands listed in Figure 5-5 on the next page perform basic arithmetic operations on items in the data stack.

Figure 5-5. Using Arithmetic

Command	Stack Diagram	Description
`*`	(n1 n2 -- n3)	Multiplies n1 * n2
`+`	(n1 n2 -- n3)	Adds n1 + n2
`-`	(n1 n2 -- n3)	Subtracts n1 - n2
`/`	(n1 n2 -- quot)	Divides n1 / n2
`<<`	(n1 +n -- n2)	Left shift n1 by +n places
`>>`	(n1 +n -- n2)	Right shift n1 by +n places
`>>a`	(n1 +n -- n2)	Arithmetic right shift n1 by +n places
`*/`	(n1 n2 n3 -- n4)	n1*n2/n3
`1+`	(n1 -- n2)	Adds 1
`1-`	(n1 -- n2)	Subtracts 1
`2*`	(n1 -- n2)	Multiplies by 2
`2+`	(n1 -- n2)	Adds 2
`2-`	(n1 -- n2)	Subtracts 2
`2/`	(n1 -- n2)	Divides by 2
`abs`	(n -- u)	Absolute value
`aligned`	(n1 -- n2)	Round n1 up to the next multiple of 4
`and`	(n1 n2 -- n3)	Bitwise logical AND
`max`	(n1 n2 -- n3)	n3 is maximum of n1 and n2
`min`	(n1 n2 -- n3)	n3 is minimum of n1 and n2
`mod`	(n1 n2 -- rem)	Remainder of n1 /n2
`/mod`	(n1 n2 -- rem quot)	Remainder, quotient of n1 / n2
`*/mod`	(n1 n2 n3 -- rem quot)	Remainder, quotient of n1 * n2 /n3
`negate`	(n1 -- n2)	Changes the sign of n1
`not`	(n1 -- n2)	Bitwise ones complement
`or`	(n1 n2 -- n3)	Bitwise logical OR
`xor`	(n1 n2 -- n3)	Bitwise exclusive OR

Accessing Memory

The PROM Toolkit provides interactive commands for examining and setting memory. You can use the Toolkit to:

- ❑ read and write to any virtual address
- ❑ map virtual addresses to physical addresses.

Memory operators allow you to read from and write to any desired memory location. All memory addresses shown in the examples that follow are virtual addresses.

"L " is sometimes printed here in uppercase to avoid confusion with the number one.

A variety of 8-bit, 16-bit, and 32-bit operations are provided. In general, a `c` (character) prefix indicates an 8-bit (one byte) operation; a `w` (word) prefix indicates a 16-bit (two byte) operation; and an `L` (longword) prefix indicates a 32-bit (four byte) operation.

You can use the commands listed in Figures 5-6, 5-7, 5-8, and 5-9 on the following pages to access, modify, map, and test memory locations. Except for commands listed in Figure 5-6, most of these are rarely needed.

Figure 5-6. Memory Accessing Commands (continued on next page)

Command	Stack Diagram	Description
`@`	(adr -- n)	Fetches a 32-bit number from adr, must be 16-bit aligned
`c@`	(adr -- byte)	Fetches a byte from adr
`w@`	(adr -- word)	Fetches a 16-bit number from adr, must be 16-bit aligned
`L@`	(adr -- long)	Fetches a 32-bit number from adr, must be 32-bit aligned
`!`	(n adr --)	Stores a 32-bit number at adr, must be 16-bit aligned
`c!`	(n adr --)	Stores low byte of n at adr
`w!`	(n adr --)	Stores a 16-bit number at adr, must be 16-bit aligned
`L!`	(n adr --)	Stores a 32-bit number at adr, must be 32-bit aligned
`blank`	(adr u --)	Sets u bytes of memory to space (decimal 32)
`cmove`	(adr1 adr2 u --)	Copies u bytes from adr1 to adr2, starting at lo byte
`cmove>`	(adr1 adr2 u --)	Copies u bytes from adr1 to adr2, starting at high byte
`comp`	(adr1 adr2 len -- n)	Compare two byte arrays, n = 0 if arrays are identical, n = 1 if first byte that is different is greater in array#1, n = -1 otherwise
`dump`	(adr len --)	Displays len bytes of memory starting at adr
`erase`	(adr u --)	Sets u bytes of memory to 0
`fill`	(adr size byte --)	Sets size bytes of memory to byte
`cfill`	(adr size byte --)	Sets size bytes of memory to byte (same as `fill`)
`wfill`	(adr size word --)	Sets size bytes of memory to 16-bit word, addr 16-bit aligned
`Lfill`	(adr size long --)	Sets size bytes of memory to 32-bit long, addr 32-bit aligned
`move`	(adr1 adr2 u --)	Copies u bytes from adr1 to adr2, handles overlap properly
`?`	(adr --)	Displays the 32-bit number at adr, must be 16-bit aligned
`c?`	(adr --)	Displays the byte at adr
`w?`	(adr --)	Displays the 16-bit number at adr, must be 16-bit aligned
`L?`	(adr --)	Displays the 32-bit number at adr, must be 32-bit aligned
`+!`	(n adr --)	Adds n to the 32-bit number stored at adr, must be 16-bit aligned
`2!`	(n1 n2 adr --)	Stores 2 numbers at adr; n2 at lower address, must be 32-bit aligned

Figure 5-6. Memory Accessing Commands (continued)

Command	Stack Diagram	Description
`2@`	(adr -- n1 n2)	Fetches 2 numbers from adr; n2 from lower address, must be 32-bit aligned
`unaligned-w@`	(adr -- word)	Fetches a 16-bit number, any alignment
`unaligned-L@`	(adr -- long)	Fetches a 32-bit number, any alignment
`unaligned-W!`	(word adr --)	Stores a 16-bit number, any alignment
`unaligned-L!`	(long adr --)	Stores a 32-bit number, any alignment

Figure 5-7. Memory Mapping Commands (continued on next page)

Command	Stack Diagram	Description
`allocate-dma`	(size -- virt)	Present in boot PROM versions 1.0 and 1.1. Only supports a single allocation at a time, and should no longer be used. This function is replaced in boot PROM version 1.2 and later with `dma-alloc`.
`dma-alloc`	(size -- virt)	Allocate and map size bytes of available memory in DMA space. Release allocated memory with `free-virtual`. This word is only available in boot PROM version 1.2 and later.
`allocate-virtual`	(phys size -- virt)	Assign a virtual address to be used for later mapping
`alloc-mem`	(size -- virt)	Allocate and map size bytes of available memory, return the virtual address
`map-sbus`	(physoffset size – virt)	Map a region of SBus space. Physoffset is the offset for the desired SBus slot – that is, slot#1=200.0000, slot#2=400.0000, slot#3=600.0000.

Figure 5-7. Memory Mapping Commands (continued)

Command	Stack Diagram	Description
`memmap`	(phys space size -- virt)	Map a region of physical addresses, return the allocated virtual address. The region is unmapped with `free-virtual`. This word is only available for boot PROM version 1.2 and later.
`free-dma`	(virt size --)	Free memory allocated by `allocate-dma`. Present in boot PROM versions 1.0 and 1.1. Only supports a single allocation at a time. These two words are obsolete for boot PROM version 1.2 and later.
`free-virtual`	(virt size --)	Undo mappings created with `map-sbus`, `allocate-virtual`, `memmap`, or `dma-alloc`
`free-mem`	(virt size --)	Free memory allocated by `alloc-mem`
`map?`	(virt --)	Display memory map information for the virtual address
`cprobe`	(adr -- flag)	Test for data exception using `c@`
`wprobe`	(adr -- flag)	Test for data exception using `w@`
`Lprobe`	(adr -- flag)	Test for data exception using `L@`

The SBus slot offsets for SPARCstations 1 and 1+ are:

- ❑ SBus slot #0 - 0 (internal slot)
- ❑ SBus slot #1 - 200.0000
- ❑ SBus slot #2 - 400.0000
- ❑ SBus slot #3 - 600.0000

Figure 5-8 lists memory mapping primitives to control page and segment maps. These commands are rarely needed.

Figure 5-8. Memory Mapping Primitives

Command	Stack Diagram	Description
`obio`	(-- space)	Specify the *device* address space for mapping
`obmem`	(-- space)	Specify the *onboard memory* address space for mapping
`sbus`	(-- space)	Specify the *sbus* address space for mapping
`allocate-physical`	(size -- phys)	Return physical address of some available memory
`free-physical`	(phys size --)	Free memory allocated by `allocate-physical`
`map-page`	(phys space virt --)	Map one page (4K) of memory starting at address *phys* onto virtual address *virt* in the given address space *space*. All addresses are truncated to lie on a page boundary
`map-pages`	(phys space virt size --)	Perform consecutive `map-pages` to map a region of memory to the given size
`pgmap!`	(pmentry virt --)	Store a new page map entry for the virtual address
`pgmap@`	(virt -- pmentry)	Return the page map entry for the virtual address
`pagesize`	(-- size)	Return the size of a page, 4K (hex 1000)
`segmentsize`	(-- size)	Return the size of a segment, 256K (hex 40000)
`smap!`	(smentry virt --)	Store a new segment map entry for the virtual address
`smap@`	(virt -- smentry)	Return the segment map entry for the virtual address
`smap?`	(virt --)	Formatted display of the segment map entry for the virtual address
`map-segments`	(smentry virt len --)	Consecutive `smap!`s to map a region of memory

Figure 5-9 lists commands to access alternate address space.

Figure 5-9. Alternate Address Space Accessing Commands

Command	Stack Diagram	Description
`spacec!`	(byte adr asi --)	Store the byte into the given asi and address
`spacew!`	(byte adr asi --)	Store the 16-bit word into the given asi and address
`spaceL!`	(byte adr asi --)	Store the 32-bit word into the given asi and address
`spacec@`	(adr asi -- byte)	Fetch the byte from the given asi and address
`spacew@`	(adr asi -- word)	Fetch the 16-bit word from the given asi and address
`spaceL@`	(adr asi -- longword)	Fetch the 32-bit word from the given asi and address
`spacec?`	(adr asi --)	Display the byte at the given asi and address
`spacew?`	(adr asi --)	Display the 16-bit word at the given asi and address
`spaceL?`	(adr asi --)	Display the 32-bit word at the given asi and address

Some useful values for asi are:

2	System space (onboard devices)
8	User instruction space
9	Supervisor instruction space
10	User data space
11	Supervisor data space.

Examples

The following examples show how you might use the Toolkit for memory mapping and testing operations.

The dump command is particularly useful. It displays a region of memory as both bytes and ASCII values.

The following example displays the contents of 20 bytes of memory starting at virtual address 10000. This example also demonstrates reading from and writing to a memory location.

```
ok 10000 20 dump        Display 20 bytes of memory starting at virtual address 10000
        \/ 1  2  3  4  5  6  7   8  9  a  b  c  d  e  f v123456789abcdef
 10000 05 75 6e 74 69 6c 00 40  4e d4 00 00 da 18 00 00 .until.@NT..Z...
 10010 ce da 00 00 f4 f4 00 00  fe dc 00 00 d3 0c 00 00 NZ..tt..~\..S...
ok 22 10004 c!          Change 8-bit byte at location 10004 to 22
ok 123 10006 w!         Change 16-bit word at location 10006 to 0123
ok 10004 L@ .           Retrieve and display 32-bit longword at location 10004
226c0123
ok
```

If you try to access (with @ for example) an invalid memory location, the operation will immediately abort and the PROM will display an error message, such as Data Access Exception or Bus Error.

To test if a location is valid or to write a loop to repeatedly access a location known to generate an exception, you will need the cprobe command.

```
ok f0000000 c@
Data Access Exception
ok f0000000 cprobe
0                           False (0) indicates error
ok
```

The Toolkit ignores decimal points in numbers. In the following example, decimal points are inserted in numbers to help count zeros.

```
ok 4000 alloc-mem .            Allocate 4000 bytes of memory and display the starting
   ffec21e0                    address of the area reserved
ok
ok ffec21e0 4000 free-mem      Return the 4000 bytes of memory at ffec21e0
ok
ok 200.0000 4000 map-sbus      Map in addresses on an SBus device in slot #1* and create
ok constant slot1              a name for the virtual address that is generated
ok
ok slot1 100 dump
ok (memory dump of FCode PROM - not shown)
ok 5000 1000 55 fill           Fill in a region of memory 5000-6000 with a fixed pattern
ok
```

The SBus slot offsets for SPARCstations 1 and 1+ are:

- ❑ SBus slot #0 - 0 (internal slot)
- ❑ SBus slot #1 - 200.0000
- ❑ SBus slot #2 - 400.0000
- ❑ SBus slot #3 - 600.0000

The following examples describe how to use the `map-page` and `map-pages` commands.

`ok 80.0000 obmem 700.0000 map-page`	*Map one page of on-board memory starting at physical address 80.0000 to virtual address 700.0000*
`ok 80.0000 obio 700.0000 map-page`	*Map one page of on-board I/O space at address 80.0000 to virtual address 700.0000*
`ok 80.0000 obmem 700.0000 4.0000 map-pages`	*Map multiple pages of on-board memory starting at physical address 80.0000 to virtual address 700.0000 until 4.0000 bytes of memory are mapped*

Using Defining Words

The defining word `variable` assigns a name to a 32-bit region of memory which you can use to hold values as needed. Later execution of that name leaves the address of the memory on the stack. Typically, `@` and `!` are used to read or write at that address. For example:

```
ok variable bar
ok 33 bar !
ok bar @ 2 + .
35
ok
```

Although `variables` are fully supported, we encourage you to use `values` (described next).

The defining word value allows you to assign a name to any number. Later execution of that name leaves the assigned value on the stack. The following example assigns a value of 22 to a word named foo, and then calls foo to use its assigned value in an arithmetic operation.

```
ok 22 value foo
ok foo 3 + .
25
ok
```

The value can be changed with the dictionary compiling word is. For example:

```
ok 43 value thisval
ok thisval .
43
ok 10 is thisval
ok thisval .
10
ok
```

Commands created with value are convenient, since you do not have to use the @ every time you want the number. This is more consistent with most other commands, whose execution leaves the desired result directly on the stack.

You can use the defining word constant like value, but only to create a name whose value will not change. A simple colon definition `: foo 22 ;` accomplishes a similar result.

The defining word `defer` allows you to change the execution of previously defined commands, by creating a slot which can be loaded with different behaviors at different times. For example:

```
ok hex
ok defer printit
ok ' .d  is  printit
ok ff printit
255
ok : myprint ( n -- ) ." It is " .h
] ." in hex " ;
ok ' myprint is printit
ok ff printit
It is ff in hex
ok
```

Figure 5-10 lists the defining words that you can use for creating dictionary entries.

Figure 5-10. Defining Words

Command	Stack Diagram	Description
: *name*	(--)	Start the creation of a new colon definition
;	(--)	Finish the creation of a new colon definition
`alias` *new-name old-name*	(--)	Create a new name with the same behavior as old-name
`buffer:` *name*	(size --)	Create a named array in temporary storage
`constant` *name*	(n --)	Define a constant (example: `3 constant bar`)
`2constant` *name*	(n1 n2 --)	Define a 2-number constant
`create` *name*	(--)	Generic defining word
`defer` *name*	(--)	Defining word for forward references or execution vectors using code field address
`does>`	(-- adr)	Start the run-time clause for defining words
`variable` *name*	(--)	Define a variable
`value` *name*	(n --)	Create a changeable named constant

Searching the Dictionary

The *dictionary* lists all available Forth commands. This section describes some useful tools you can use to search the dictionary.

The command `words` displays all word (command) names in the dictionary, starting with the most recent definitions.

The command `see`, used in the form `see` *thisword,* will decompile the specified command (*thisword*). This means that it shows the definition used to create that command word.

Figure 5-11 lists the commands you can use to search the contents of the dictionary.

Figure 5-11. Dictionary Searching Commands

Command	Stack Diagram	Description
' *name*	(-- acf)	Finds a word in the dictionary. Returns the *code field address.* Use outside of definitions
['] *name*	(-- acf)	Acts similar to ' but is used inside of definitions
`find`	(pstr -- acf n)	Searches for a word in the dictionary. The word to be found is indicated by *pstr. n* is 0 if not found, 1 if immediate, -1 otherwise
`words`	(--)	Displays all visible words in the dictionary
`see` *name*	(--)	Decompiles the named word*
`(see)`	(acf --)	Decompiles the word indicated by the *code field address**

*The decompiled definition may sometimes be confusing, because some internal names may have been omitted from the PROM's symbol table to save space.

Compiling into The Dictionary

These commands compile data into the dictionary. Most of these are not normally needed. Figure 5-12 lists these commands.

Figure 5-12. Dictionary Compiling Commands

Command	Stack Diagram	Description
`,`	(n --)	Place a number in the dictionary
`c,`	(n --)	Place a byte in the dictionary
`w,`	(w --)	Place a 16-bit word in the dictionary
`L,`	(n --)	Place a 32-bit number in the dictionary
`[`	(--)	Begin interpreting
`]`	(--)	Begin compilation
`allot`	(n --)	Allocate *n* bytes in the dictionary
`compile`	(--)	Compile next word at run time
`[compile]` *name*	(--)	Compile the next (immediate) word
`forget` *name*	(--)	Remove word from dictionary and all subsequent words
`here`	(-- adr)	Address of top of dictionary
`immediate`	(--)	Mark the last definition as immediate
`is` *name*	(acf --)	Install a new action in a `defer` word or `value`
`literal`	(n --)	Compile a number
`state`	(-- adr)	Variable that is nonzero in compile state
`npatch` *word-to-patch*	(new-n old-n --)	Replace first old-n with new-n in the word *word-to-patch*. Note that the values 0, 1, 2, and 3 are actually defined as words, not numbers. Use `patch`.
`patch` *new-word old-word word-to-patch*	(--)	Replace first *old-word* with *new-word* in *word-to-patch*.
`(patch`	(new-n old-n acf --)	Replace first old-n with new-n in word indicated by acf.

Controlling Text Input and Output

This section describes text input and output commands, which are listed in Figures 5-13, 5-14, and 5-15. These commands control strings or character arrays, and allow you to enter comments and control keyboard scanning.

Comments are used with Forth source code (generally in a text file) to describe the function of the code. The `(` (open parenthesis) is a command that begins a comment. Any character up until the closing parenthesis `)` is ignored by the Forth interpreter. Remember to follow the `(` with a space, so it will be recognized. Stack effect diagrams are one example of comments using `(`. The `\` (backslash) also indicates a comment, terminated by the end of the line of text.

The `key?` command looks at the keyboard to see whether the user has recently typed any key. It returns a flag on the stack: `true` if a key has been pressed and `false` otherwise. See the next section, "Using Conditional Testing" for a discussion of the use of flags.

The command `key` waits for a key to be pressed, then returns the ASCII value of that key on the stack.

The command `ascii`, used in the form `ascii` *x*, returns on the stack the numerical ASCII code of the letter following.

The `emit` command displays the letter whose ASCII value is on the stack. For example:

```
ok ascii a
61 ok 42
61 42 ok emit emit
Ba
ok
```

The cr command sends a carriage-return to the output. For example:

```
ok 3 . 44 . cr 5 .
3 44
5
ok
```

The ." command used in the form ." *string*" outputs text when needed. This command only works inside of a definition. A " (double quotation mark) is used to mark the end of the text string.

For example:

```
ok  : testing 34 .  ." This is a test"  55 . ;
ok
ok testing
34 This is a test55
ok
```

Finally, some string commands specify an *address* (the location in memory where the characters reside) and a *length* (how many characters). Other commands use a *packed string* or pstr, which is a location in memory containing a byte for the length, immediately followed by the characters. The stack effect comment for the command will indicate which form is used. The count command converts a packed string to an address-length string.

Interpreting Source Code

The command `eval` takes a string off of the stack (specified as an address and a length). That string is then interpreted, just as if those characters were entered from the keyboard. If a Forth text file has been loaded into memory (for example, with `dload`. See "Downloading Files" in Chapter 6), then `eval` can be used to compile whatever definitions were contained in the file. Figures 5-13, 5-14, and 5-15 list commands to control text input and keyboard scanning, text output display, and strings or character arrays.

Figure 5-13. Inputting Text

Command	Stack Diagram	Description
(*ccc*)	(--)	Begin a comment
\ *rest-of-line*	(--)	Skip the rest of the line
`key`	(-- char)	Read a character from the keyboard
`key?`	(-- flag)	True if a key has been typed on the keyboard
`ascii` *ccc*	(-- char)	Numerical value of first ascii character of next word
`bl`	(-- n)	The ASCII code for the space character; decimal 32

Figure 5-14. Displaying Text Output

Command	Stack Diagram	Description
`cr`	(--)	Terminates a line on the display and go to the next line
`emit`	(char --)	Displays the character
`exit?`	(-- flag)	True if the user wants the output to be terminated. This command enables the scrolling control prompt: `More [<space>,<cr>,q] ?`
`space`	(--)	Displays a space character
`spaces`	(+n --)	Displays +*n* spaces

Figure 5-15. Manipulating Text Strings

Command	Stack Diagram	Description
`"` *ccc*`"`	(-- adr len)	Collect an input stream string, either interpreted or compiled
`."` *ccc*`"`	(--)	Compile a string for later display
`.(` *ccc*`)`	(--)	Display a string immediately
`eval`	(adr len --)	Interpret Forth source from an array
`p"` *ccc*`"`	(-- pstr)	Collect a string from the input stream, store as a packed string
`type`	(adr +n --)	Displays characters
`count`	(pstr -- adr +n)	Unpack a packed string
`-trailing`	(adr +n1 -- adr +n2)	Remove trailing spaces

Using Conditional Testing

Forth conditionals use flags to indicate true/false values. A flag can be generated in any number of ways based on some criteria for testing. The flag can then simply be displayed off of the stack with the word . or can be used as an input to a conditional control command. Control commands can cause one behavior if a flag is true, and another behavior if the flag is false. Thus, execution can be altered based on the result of a test.

A `0` value indicates the flag value is `false`. A `-1` (or any other nonzero number) indicates the flag value is `true`. In hexadecimal, the value `-1` is displayed as `ffffffff`.

For example, the > command takes two numbers off of the stack, and returns true (-1) on the stack if the first number was greater than the second number, or returns false (0) otherwise. For example:

```
ok 3 6 > .
0                    3 is not greater than 6
ok
```

The `0=` command takes one number off of the stack, and returns true if that number was 0, or returns false otherwise. This word inverts any flag to its opposite value.

Figure 5-16 lists commands that perform relational tests, and leave a true or false flag result on the stack.

Figure 5-16. Comparison Commands

Command	Stack Diagram	Description
`0<`	(n -- flag)	True if n < 0
`0<=`	(n -- flag)	True if n < = 0
`0<>`	(n -- flag)	True if n <> 0
`0=`	(n -- flag)	True if n = 0 (also inverts any flag)
`0>`	(n -- flag)	True if n > 0
`0>=`	(n -- flag)	True if n > = 0
`<`	(n1 n2 -- flag)	True if n1 < n2
`<=`	(n1 n2 -- flag)	True if n1 <= n2
`<>`	(n1 n2 -- flag)	True if n1 <> n2
`=`	(n1 n2 -- flag)	True if n1 = n2
`>`	(n1 n2 -- flag)	True if n1 > n2
`>=`	(n1 n2 -- flag)	True if n1 >= n2
`between`	(n min max -- flag)	True if min <= n <= max
`false`	(-- 0)	The value FALSE, which is 0
`true`	(-- -1)	The value TRUE, which is -1
`u<`	(u1 u2 -- flag)	True if u1 < u2 , unsigned
`u<=`	(u1 u2 -- flag)	True if u1 <= u2, unsigned
`u>`	(u1 u2 -- flag)	True if u1 > u2, unsigned
`u>=`	(u1 u2 -- flag)	True if u1 > = u2, unsigned
`within`	(n min max -- flag)	True if min < = n < max

Controlling Conditional Execution

The commands `if`, `else`, and `then` provide a simple *if-then-else* control structure.

The format for using these commands is:

```
flag    if      do this if true
        else    do this if false
        then    continue normally
```

or

```
flag    if      do this if true
        then    continue normally
```

The `if` consumes a flag off of the stack. If the flag is true (non-zero), the commands just after the `if` are performed. Otherwise, the commands (if any) just after the `else` are performed.

```
ok : testit  ( n -- )
] 5 >  if   ." good enough "
] else   ." too small "
] then
] ." Done. "   ;
ok
ok 8 testit
good enough Done.
ok 2 testit
too small Done.
ok
```

Note: The `]` prompt reminds you that you are part way through creating a new colon definition. It reverts back to `ok` after you finish the definition with a semicolon.

The commands listed in Figure 5-17 control the flow of conditional execution.

Figure 5-17. Conditional Program Execution Commands

Command	Stack Diagram	Description
`else`	(--)	Execute the following code if `if` failed
`if`	(flag --)	Execute following code if flag is true
`then`	(--)	Terminate `if...else...then`

Using Conditional Loops

Conditional loops execute the same commands repeatedly until a certain condition is satisfied. There are two general forms:

`begin` *any commands...* *flag* `until`

and

`begin` *any commands...* *flag* `while`
more commands `repeat`

In both cases, the commands within the loop will be executed repeatedly until the proper flag value causes the loop to be terminated. Once terminated, execution continues normally with the next command after the closing command word (`until` or `repeat`).

In the `begin...until` case, the `until` command removes a flag from the top of the stack and inspects it. If the flag is false, execution continues just after the `begin` and the loop repeats. If the flag is true, the loop is exited.

In the `begin...while...repeat` case, the `while` command removes a flag from the top of the stack and inspects it. If the flag is true, the loop continues by executing the commands just after the `while`. The `repeat` command automatically sends control back to the `begin` to continue the loop. If the flag is false when `while` is encountered, then the loop is exited immediately. Control goes to the first command after the closing `repeat`.

The following is a simple example:

```
ok begin 4000 c@ .  key? until     repeat until any key is pressed
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
ok
```

The loop starts by fetching a byte from location 4000 and displaying the value. Then, the `key?` command is called, which leaves a true on the stack if the user has pressed any key, false otherwise. This flag is consumed by the `until` and, if the value is false, then the loop continues. Once a key is pressed, the next call to `key?` returns true and the loop terminates.

Unlike many versions of Forth, the Toolkit allows interactive use of loops and conditionals — that is, without first creating a definition.

Figure 5-18 lists commands to control the execution of conditional loops.

Figure 5-18. Conditional Loop Commands

Command	Stack Diagram	Description
`again`	(--)	Ends a `begin...again` infinite loop
`begin`	(--)	Begin a `begin...while...repeat` loop or `begin...until` loop
`repeat`	(--)	Ends a `begin...while...repeat` loop
`until`	(flag --)	Continues executing a `begin...until` loop until flag is true
`while`	(flag --)	Continues executing a `begin...while...repeat` loop while flag is true

Using Counted Loops

Counted loops, called *do loops,* are used when the number of iterations of the loop can be calculated in advance.

Figure 5-19 lists commands to control the execution of counted loops.

Note: A *do loop* normally exits just *before* the specified ending value is reached.

Figure 5-19. Counted Loop Commands

Command	Stack Diagram	Description
`do`	(end start --)	Begin a `do...loop`. Index goes from start to end-1 inclusive. Example: `10 0 do i . loop`
`?do`	(end start --)	Begin `?do...loop` to be executed 0 or more times. Index goes from start to end-1 inclusive.
`i`	(-- n)	Loop index
`j`	(-- n)	Loop index for next enclosing loop
`leave`	(--)	Exit from `do...loop`
`?leave`	(flag --)	Exit from a `do...loop` if flag is non-zero
`loop`	(--)	End of `do...loop`
`+loop`	(n --)	End a `do...+loop` construct; adds n to loop index and returns to `do` (if n < 0, index goes from start to end inclusive).

Several loop examples follow:

```
ok 10 5 do  i .  loop
5 6 7 8 9 a b c d e f
ok
ok 2000 1000 do i .  i c@ . cr   i c@ ff = if leave then  4 +loop
1000 23
1004 0
1008 fe
100c 0
1010 78
1014 ff
ok : scan ( byte -- )
]    6000 5000                                Scan memory (5000-6000) for
]    do dup i c@ <> (  byte error? )          bytes not equal to the pattern (55)
]      if i . then   ( byte )
]    loop
]    drop ( the original byte was still on the stack, discard it )
]  ;
ok 55 scan
ok 5005 5224 5f99
ok 6000 5000 do i i c! loop                   Fill a region of memory
ok                                            with a stepped pattern
ok                                            (0-1-2-3-...)
ok 500 value testloc
ok : test16 ( -- ) 1.0000 0 ( do 0-ffff )     Write different 16-bit values to
]     do i testloc w! testloc w@ i <> ( error? ) a location and check.
]       if ." Error - wrote " i . ." read " testloc w@ . cr
]        leave ( exit after first error found ) This line is optional
]       then
]     loop
]  ;
ok test16
ok 6000 is testloc
ok test16
ok
```

Using Case Statements

A high-level case command is provided for selecting alternatives with multiple possibilities. It is easier to read than deeply nested if-then commands. A simple example follows:

```
ok : testit  ( testvalue -- )
]  case  0  of  ." It was zero "  endof
]    1  of  ." It was one "  endof
]    ff of  ." Correct "  endof
]    -2 of  ." It was minus-two "  endof
]    ( default )  ." It was this value: "  dup .
]  endcase   ." All done."  ;
ok
ok 1 testit
It was one All done.
ok ff testit
Correct All done.
ok 4 testit
It was this value: 4 All done.
ok
```

Note: The (optional) default clause can use the test value which is still on the stack, but should *not* remove it (use the dup . phrase instead of .). A successful of clause automatically removes the test value from the stack.

Figure 5-20 on the next page lists the conditional case statement commands you can use.

Figure 5-20. Case Statement Commands

Command	Stack Diagram	Description
case	(selector -- selector)	Begins a case. . .endcase conditional
endcase	(selector --)	Terminates a case. . .endcase conditional
endof	(--)	Terminates an of. . .endof clause withing a case. . .endcase
of	(selector test-value -- selector | {empty}	Begins an of. . .endof clause within a case conditional

Additional Control Commands

The abort command causes immediate termination and returns control to the keyboard. abort" is similar to abort but is different in two respects. abort" removes a flag from the stack and only aborts if the flag is true. Also, abort" prints any desired message when the abort takes place. Figure 5-21 contains descriptions of the various program execution control commands.

Figure 5-21. Program Execution Control Commands

Command	Stack Diagram	Description
`abort`	(--)	Abort current execution and interpret keyboard commands
`abort"` *ccc*"	(flag --)	If flag is true, abort and display message
`execute`	(acf --)	Execute the word whose code field address is on the stack
`exit`	(--)	Return from the current word
`quit`	(--)	Abort, but leave stack intact

Using the Disassembler

The PROM's built-in disassembler translates the contents of memory into equivalent SPARC assembly language. The `dis` command will begin to disassemble the data content of any desired location. A pause occurs if any key is pressed while disassembly is taking place or after every page of output. At that point, disassembly might be continued or stopped. Disassembly stops automatically when a `call` or `jmp` opcode is encountered.

You can use the `+dis` command to continue disassembling at the location where the last disassembly stopped.

Memory addresses are normally shown in hex. However, if a symbol table is present, memory addresses will be displayed symbolically whenever possible.

Figure 5-22 lists commands that disassemble memory into equivalent opcodes.

Figure 5-22. Disassembler Commands

Command	Stack Diagram	Description
`dis`	(addr --)	Begin disassembling at the given address
`+dis`	(--)	Continue disassembling where the last disassembly left off

Displaying Registers

You can enter the Toolkit from the middle of an executing program as a result of a program crash, a user abort with L1-A, or an encountered breakpoint (breakpoints are discussed in the next section). In all these cases, the Toolkit automatically saves all of the CPU data register values into a buffer area. These values can then be inspected for debugging purposes.

After inspection, program execution can be continued by entering the `go` command. The saved register values are copied back into the CPU, and then execution resumes (at the location specified by the saved `%pc`).

These saved register values can be changed, if desired, by using the `to` command. When execution is resumed, the changed values will be copied back into the CPU and used.

If you change `%pc` with the `to` command, you should also change `%npc`. It is easier to use `set-pc`, which changes both registers automatically.

For the `w` and `.window` commands, a window value of 0 usually specifies the current window — that is, the active window for the subroutine where the program was interrupted. A value of 1 specifies the window for the caller of this subroutine, and 2 specifies the caller's caller, and so on, up to the number of active stack frames. The default starting value is 0.

Note: In some earlier versions of the Open Boot PROM, floating-point registers can only be read, but not written. In these versions, the command `save-fregs` must be executed before you can read floating-point registers. Floating-point registers are not supported in some boot PROMs.

Figure 5-23 lists the SPARC register reading and writing commands.

Figure 5-23. SPARC Register Commands

Command	Stack Diagram	Description
`%g0` through `%g7`	(-- value)	Return the value in the given register
`%o0` through `%o7`	(-- value)	Return the value in the given register
`%L0` through `%L7`	(-- value)	Return the value in the given register
`%i0` through `%i7`	(-- value)	Return the value in the given register
`%pc %npc %psr`	(-- value)	Return the value in the given register
`%y %wim %tbr`	(-- value)	Return the value in the given register
`%f0` through `%f31`	(-- value)	Return the value in the given floating point register
`%fsr`	(-- value)	Return the value in the given floating point register
`to` *regname*	(value --)	Change the value stored in any of the above registers Use in the form: *value* `to` *regname*
`set-pc`	(value --)	Set `%pc` to the given value, and set `%npc` to (value+4)
`w`	(window# --)	Set the current window, for displaying `%ix` `%lx` or `%ox`
`ctrace`	(--)	Display the return stack showing C subroutines
`.locals`	(--)	Display the values in the `i`, `L` and `o` registers
`.psr`	(--)	Formatted display of the `%psr` data
`.registers`	(--)	Display values in `%g0` through `%g7`, plus `%pc`, `%npc`, `%psr`, `%y`, `%wim`, `%tbr`

Using Breakpoints

The Toolkit provides a robust breakpoint capability to assist in the development and debugging of standalone programs.

Standalone programs do not run under the SunOS Operating System. Typical programs running under the SunOS Operating System do not normally use this feature, but use other debuggers designed to run under theSunOS Operating System.

The breakpoint feature lets you pause execution of the test program at any desired point(s). After execution has stopped, registers or memory can be inspected and/or changed, and new breakpoints can be set or cleared. Then, execution can be resumed with the `go` command.

To debug a program using breakpoints:

1. Load the test program into memory at location 4000 (hex).

   See "Downloading Files" in Chapter 6 for more information. Using `dload` is generally best, since the symbol table for the program is preserved. `boot -h` also works if the program is not available over Ethernet.

   The values for `%pc` and all other registers will be initialized automatically.

2. Disassemble the downloaded program, if desired, to verify a properly downloaded file.

3. At this point you can begin single-stepping the test program using the `step` command. Or you can set a breakpoint(s) and then execute (for example, using the commands `4020 +bp` and `go`), or perform other variations.

Figure 5-24 lists the breakpoint commands that control and monitor program execution.

Figure 5-24. Breakpoint Commands

Command	Stack Diagram	Description
`go`	(--)	Continue the execution of a halted program
`.bp`	(--)	Display all curently set breakpoints
`+bp`	(addr --)	Add a breakpoint at the given address
`-bp`	(addr --)	Remove the breakpoint at the given address
`--bp`	(--)	Remove the most recently set breakpoint
`bpoff`	(--)	Remove all breakpoints
`step`	(--)	Single-step one instruction
`steps`	(n --)	Execute n `steps`
`hop`	(--)	Like the `step` command, but treats a subroutine call as a single instruction
`hops`	(n --)	Execute n `hops`
`skip`	(--)	Skip (do not execute) the current instruction
`till`	(addr --)	Executes until the given address is encountered; equivalent to `+bp go`
`return`	(--)	Execute until the end of this subroutine
`returnL`	(--)	Execute until the end of this leaf subroutine
`finish-loop`	(--)	Execute until the end of this loop
`.instruction`	(--)	Display the address, opcode for the last encountered breakpoint
`.breakpoint`	(--)	Defer word, for display behavior after every breakpoint. Default is `.instruction`. Change with: `' .registers is .breakpoint`

The examples shown in this chapter illustrate some of the tools available to you through the boot PROM's Toolkit interface. If you require more information about Forth, consult some of the previously cited reference books.

Using Machine Functions

This chapter provides machine-specific information about how to control your system using the Open Boot PROM's Forth Toolkit. The machine functions described in this chapter allow you to perform the following tasks:

- ❑ Resetting the system
- ❑ Running the diagnostics
- ❑ Displaying system information
- ❑ Booting from the `ok` prompt
- ❑ Redirecting input and output
- ❑ Setting up a `tip` window
- ❑ Downloading files
- ❑ Ejecting a floppy diskette
- ❑ Preserving data after a system crash.

The procedures in this chapter assume that you have started the Sun-Compatible Monitor, and have entered the Forth Toolkit. This chapter also assumes that you have read Chapter 4 and are familiar with the Forth Toolkit interface.

Resetting the System

Occasionally you will find it necessary to reset the system. The `reset` command, listed in Figure 6-1, resets the system without actually having to turn the power off and on.

To reset the system, type:

```
ok reset
```

The power-on self-test and initialization procedure begin immediately. This system reset is very similar to a power cycle. All Forth definitions you entered are forgotten.

Figure 6-1. System Resetting Commands

Command	Stack Diagram	Description
`reset`	(--)	Resets the entire system (very similar to power-cycle)

Diagnostic Routines

Several diagnostic routines are available through the Toolkit. These on-board tests allow you to test the control registers, the network controller, the floppy disk system, memory, the cache, and the system clock. Figure 6-2 lists commands to perform diagnostic tests.

Figure 6-2. Diagnostic Test Commands

Command	Stack Diagram	Description
`probe-scsi`	(--)	Determine the attached SCSI devices
`test-control-regs`	(--)	Test registers (context, sync, sync virt, async, async virt, enable)
`test-net`	(--)	Test Lance Ethernet controller with internal & external loopback
`test-cache`	(--)	Test cache data and tag fields
`test-memory`	(--)	Test main memory (number of megabytes indicated in NVRAM configuration parameter `selftest-#megs`). If the `diag-switch?` NVRAM configuration parameter is set to true, all of memory is tested.
`test-floppy`	(--)	Test the floppy drive
`watch-clock`	(--)	Test the clock function

Testing Control Registers

Control registers reside in hardware on the main-logic board on the SPARCstation 1. The registers that are tested include the context register, the synchronous error register, the synchronous error virtual address register, the asynchronous error register, the asynchronous error virtual address register and the enable register. Other machines might have a different set of control registers.

To test control registers, type:

```
ok test-control-regs
ok
```

If the system fails this test, a message appears on the screen. If the system passes this test, the system displays the `ok` prompt.

Testing The Ethernet Controller

To test the on-board Ethernet controller, type:

```
ok test-net
Internal Loopback test - (result)
External Loopback test - (result)
ok
```

The system responds with a testing message that indicates the result of the test.

Testing The Diskette Drive

The diskette drive test determines whether the diskette drive is functioning properly. A formatted (HD) disk must be inserted into the diskette drive for this test to complete successfully.

To test the diskette drive system, type:

```
ok test-floppy
Testing the floppy disk system.  A formatted
disk should be in the drive.
It appears to be okay.
ok
```

If the test fails, you will see an error message. Eject the diskette by typing `eject-floppy` at the `ok` prompt.

Testing Memory

When you use the memory testing routine, the system will test the number of megabytes specified in NVRAM parameter `selftest-#megs`. One megabyte of memory is tested as the default. When the diagnostic switch NVRAM parameter `diag-switch?` is enabled, all memory is tested.

To test memory, type:

```
ok test-memory    There will be a delay while the PROM tests
                  the system before the prompt returns to the
                  display
ok
```

If the system fails this test, you will see an error message. Otherwise, the `ok` prompt returns to the display.

Testing Cache

The cache test routine exercises the cache buffers.

To test the cache, type:

```
ok test-cache     There will be a delay while the PROM tests
                  the system before the prompt returns to the
                  display
ok
```

If the system fails this test you will see an error message. Otherwise, the ok prompt returns to the display.

Testing The Clock

To test the clock function, type:

```
ok watch-clock
Watching the 'seconds' register of the real
time clock chip.
It should be ticking once a second.
Type any key to stop.
1                 Press any key to stop test
ok
```

The system responds by incrementing a number once a second. Press any key to return to the ok prompt.

Displaying System Information

The Toolkit provides several commands you can use to display pertinent system information. These commands, listed in Figure 6-3, allow you to display the system banner, the Ethernet address for the Ethernet controller, the contents of the IDPROM, and the version number of the PROM. The IDPROM contains information specific to each individual machine, including the serial number, date, and Ethernet address assigned to the machine.

Figure 6-3. System Information Display Commands

Command	Stack Diagram	Description
`banner`	(--)	Displays power-on banner
`.enet-addr`	(--)	Displays the current Ethernet address
`.idprom`	(--)	Displays IDPROM contents, formatted
`.version`	(--)	Display the version and date of boot PROM
`.traps`	(--)	Display a list of SPARC trap types

Booting the System From the Toolkit Prompt

The `boot` command loads the SunOS Operating System kernel or another executable program into memory, and executes that program when the program load completes.

All booting operations function the same, whether you are in the Sun-Compatible Monitor or in the Forth Toolkit. The only difference is that you must type out the entire word `boot` (followed by a space when using any command options) when you are in the Toolkit.

To boot your system from the `ok` prompt, type the `boot` command using the standard boot syntax. See Chapter 3 for information about booting.

Input, Output, and Display Modes

Normally, your system uses a standard Sun keyboard for all user input, and a frame buffer with a connected display screen for most display output. It is possible to redirect the input or output, or both, to either one of the system's serial ports. This might be useful, for example, when debugging a frame buffer. See your system's Installation Guide for information about connecting a terminal to the system unit.

Redirecting Input and Output

The commands `input` and `output` change the current sources of input and output. The change takes place immediately without a system reset. The `input` command must be preceded by one of the following: `keyboard`, `ttya`, or `ttyb`.

For example, if input is currently accepted from the keyboard, and you wish to make a change so that input is accepted from a terminal connected to the serial port ttya, type:

```
ok ttya input
ok
```

At this point, the Sun keyboard will be non-functional (except for L1-A), but any text entered from the terminal connected to ttya will be processed as input. All commands will be executed as usual. To resume using the keyboard as the input device, type from the terminal keyboard:

```
ok keyboard input
ok
```

Similarly, the `output` command must be preceded by one of the following: `screen`, `ttya`, or `ttyb`.

If you wish to send output to ttya instead of the normal display screen, type:

```
ok ttya output
```

The screen will not show the answering `ok` prompt, but the terminal connected to ttya will show the `ok` prompt, and all further output as well.

Note that either **screen io** or **keyboard io** is equivalent to **keyboard input** plus **screen output**.

The command `io` is used in the same way, except that it changes both the input and output to the specified place.

The commands `input`, `output`, and `io` have a temporary effect only. A system reset or power cycle causes the input and output sources to revert back to the default settings specified in the configuration parameters. The NVRAM parameters `input-device` and `output-device` control the default input and output sources, and can be changed if desired. See "Changing a Parameter's Value" in Chapter 7 for information about changing defaults.

The standard baud rate and digital signal transmission settings for both ttya and ttyb are: 9600 baud, 8 data bits, 1 stop bit, no parity, and no handshaking. These settings can be changed if desired, using the `ttya-mode` and `ttyb-mode` NVRAM parameters.

Emergency Procedure

There is also an *emergency procedure,* in case a specified input source is unavailable. For example, suppose you typed `ttya io` and then discovered that your terminal connected to ttya has the wrong baud rate and cannot be easily changed. Or worse, suppose you set the NVRAM parameters incorrectly, so that even a power cycle leaves you without a usable source of input.

Even when the Sun keyboard is inactive (because the serial port is being used for input), the L1-A key combination from the Sun keyboard will still be detected. When L1-A is pressed, the system resets the input source back from the current setting and accepts input to the keyboard.

Note: L1-A does not change the output source. If output is incorrect, then you might also need to restore the output to the screen connection by typing `screen output` and pressing Return. Characters are not echoed as you type. If you make a mistake, error messages are not displayed. If this procedure does not work correctly, you might need to type `n` and press Return to enter the Toolkit, and then type `screen output` and press Return.

Figure 6-4 lists commands you can use to redirect input and output.

Figure 6-4. Input, Output, and Display Commands

Command	Stack Diagram	Description
`input`	(source --)	Select source for subsequent input (ttya, ttyb, or keyboard)
`output`	(source --)	Select source for subsequent output (ttya, ttyb, or screen)
`io`	(source --)	Select source for subsequent input and output
L1-A (from keyboard)	(--)	Redirect input to come from keyboard

Setting Up a `tip` Connection

You can use the ttya or ttyb ports on your SPARC system to connect to another Sun Workstation (either the same type of SPARC system or a different type of Sun Workstation or server system). This connection allows you to use a shell window on the Sun Workstation as a terminal to your SPARC system being tested. See the on-line `tip` man-page documentation for detailed information about terminal connection to a remote host.

We highly recommend the `tip` method, since it allows you to use the SunOS Operating System windowing and operating system features to assist you in your interactions with the boot PROM. The SunOS Operating System must first be loaded. A communications program or another non-Sun computer can be used in the same way, if the program can keep up with the output baud rate used by the PROM tty port. A simple setup procedure follows.

To set up a `tip` connection:

1. Connect the Sun Workstation (ttyb serial port) to your SPARC system ttya serial port using a serial connection cable. This connection is made with a 3-wire Null Modem Cable. Connect wires 3-2, 2-3, and 7-7. Refer to your Installation Guide for specifications on null modem cables.
2. At the Sun Workstation, add the following lines to the file `/etc/remote`:

```
hardwire:\
        :dv=/dev/ttyb:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
```

3. In a shell window on the Sun Workstation, type `tip hardwire` and press Return.

   The system will reply, `connected`.

   ```
   hostname% tip hardwire
   connected
   ```

   The shell window is now a `tip` window directed to the Sun Workstation ttyb.

4. At your SPARC system, start the Sun-Compatible Monitor and enter the Toolkit. You see the `ok` prompt.

   Note: When you **do not have a video monitor** attached to your SPARC system unit, connect the SPARC system unit to the Sun Workstation and turn the power on to your SPARC system. Wait 10 or 15 seconds and press L1-A to interrupt the power-up sequence and start the Monitor. Type `n` and press Return. Unless the system is totally inoperable, the Toolkit is open and you can continue with the next step in this procedure.

5. To redirect the standard input and output to ttya, if needed, type `ttya io` and press Return.

   ```
   ok ttya io
                                   No echoed response
   ```

6. Press [Return] on the Sun workstation keyboard. The `ok` prompt appears in the `tip` window.

Caution: Do not type an [L1-A] from a Sun Workstation being used as a `tip` window to your SPARC system. Doing so will abort the SunOS Operating System on the Sun Workstation. If you forget and accidentally do so, you can recover by immediately typing the letter `c` then pressing [Return].

Ending the `tip` Session

When you are finished using the `tip` window, you need to end your tip session and exit the `tip` window.

To end the `tip` session:

1. Redirect the input and output to the screen and keyboard, if needed.
2. In the `tip` window, type the **~.** command.
3. The tip window session is closed, and you see the host prompt.

```
ok ~.

hostname%
```

Note: When entering ~ commands in the `tip` window, the ~ (tilde character) must be the first character entered on the line. When in doubt, press [Return] first.

Common Problems With `tip`

Common problems with `tip` might occur if:

1. The lock directory is missing or incorrect. There must be a directory `/usr/spool/uucp`. The owner must be `uucp` and the group must be `staff`. The mode is `drwxr-sr-x`.
2. `ttyb` is not enabled for logins. The status field for `ttyb` (or the serial port you are using) must be set to `off` in `/etc/ttytab`. Be sure to execute `kill -HUP 1` (see **init(8)**) as root if you have to change this entry.
3. `/dev/ttyb` is inaccessible. Sometimes, a program will have changed the protection of `/dev/ttyb` (or the serial port you are using) so that it is no longer accessible. Make sure that `/dev/ttyb` has the mode set to `crw-rw-rw-`.
4. The serial line is in tandem mode. If the tip connection is in tandem mode, the operating system will sometimes send XON (^S) characters, particularly when programs in other windows are generating lots of output. The XON characters will be detected by the Forth `key?` word, and can cause confusion. The solution is to turn off tandem mode with the `tip` command `~s !tandem`.

Downloading Text Files Over a Serial Line

File downloading commands allow you to download and interpret a Forth text file over a serial connection made between your SPARC system and a Sun Workstation (or another SPARC system the same type as your SPARC system). You can also download Forth or binary files over an Ethernet connection, or from a locally-attached diskette or the SCSI disk drive.

To download a Forth text file over a serial connection, you must have a Sun Workstation connected to a serial port on your SPARC system. The Sun Workstation must have a `tip` window connection set up. The following procedure assumes that you have made the serial connection described in "Setting up a `tip` Connection" earlier in this chapter and that you have a `tip` window open on the Sun Workstation. Input and output must be directed to that connection.

To download a Forth file from a Sun Workstation to your SPARC system:

1. In the Sun Workstation `tip` window, type `dl` and press [Return].
2. Type `~C` and `cat` *myfile.fth*

   Note: The C must be capitalized.

   The file will be searched for relative to the current working directory when the `tip` window was activated.
3. Wait several seconds for download to complete.

4. Type Control-d.

```
ok dl
~CLocal command?cat myfile.fth

away for 2 seconds
!
^D
ok
```

5. If the requested file is not found, the following message is displayed.

```
ok dl
~CLocal command?cat myfile.fth
myfile.fth: No such file or directory
away for 2 seconds
!
^D
ok
```

6. Type Control-d to return to the `ok` prompt.

After the downloading is complete, the contents of the Forth text file are automatically interpreted. You can download files up to 32K using this method. If you need to interpret a larger file, break it into pieces and download each piece with a separate `dl` command.

Downloading Binary Files Over a Serial Line

To download a binary file over a serial connection, you must have a Sun Workstation connected to a serial port on your SPARC system. The Sun Workstation must have a `tip` window set up. The following procedure assumes that you have made the serial connection explained in "Setting up a `tip` Connection" earlier in this chapter, and that you have a `tip` window open on the Sun Workstation. Input and output must be directed to that connection.

Note: Before beginning this procedure, you must have the shell script *sendbin* defined in your directory on the Sun Workstation.

If your SunOS version is 3.9 or earlier, create a *sendbin* script file with these statements:

```
# !/bin/sh
sleep 5 >/dev/tty &
stty -even -odd litout pass8
cat $1
```

If your SunOS version is 4.0 or later, create a *sendbin* script file with these statements:

```
# !/bin/sh
sleep 5 >/dev/tty &
stty -parenb cs8 raw -echo
cat $1
```

After creating a *sendbin* script file for your SunOS version, you can download your binary file.

Note: You must make your *sendbin* script executable by typing `chmod 777` *sendbin* at the `%` prompt.

To download a binary file from a Sun Workstation to your SPARC system:

1. In the Sun Workstation `tip` window, type `dlbin` and press Return.
2. Type `~C` and `sendbin` *myfile.ext* and press Return.

   The file will be searched for relative to the current working directory when the `tip` window was activated.
3. Wait several seconds for download to complete.

```
ok dlbin
~CLocal command?sendbin myfile.ext

away for 2 seconds
!
ok
```

4. If the requested file is not found, the following message is displayed.

```
ok dlbin
~CLocal command?sendbin myfile.ext
myfile.ext: No such file or directory
away for 2 seconds
!
```

5. Type `~#` to return to the `ok` prompt.

Your file will be placed in a memory location determined by the *a.out* header of the downloaded file (usually hex 4000). Type `init-program go` to execute the downloaded binary file, if desired.

If the downloaded file is an FCode binary file, interpret it by typing `4000 1 byte-load` at the `ok` prompt.

Downloading Files Over Ethernet

You can download any file over Ethernet with the `dload` command. `dload` requires that you specify the address at which you wish to download the file. In general, we recommend storing the file at location 4000, which is a known and well behaved address. The address 4000 is used in the following example.

For binary files, `dload` is superior to other downloading methods, because the symbol table (useful in debugging) is also downloaded automatically.

`dload` uses the `tftp` protocol to transfer a file over the network. You must have permission for `tftp` to access files on your server. Ask your system administrator to remove the # (pound sign) at the beginning of the line "`tftp...`" in the server's file `/etc/inetd.conf`, and to put in a pound sign before the `-s` flag (if present). This allows `tftp` access to any file.

To download and execute a file at address 4000:

1. At the `ok` prompt type `4000 dload` */path/filename.ext* and press Return.

   Specify the full pathname relative to the server's root.

2. If your downloaded file:

   ❑ is a binary file, type `init-program go` to execute that program.

   ❑ contains Forth text, type `4000 file-size @ eval` to interpret the file.

   ❑ contains FCode binary, type `4000 1 byte-load` to interpret the file.

You can also download files over Ethernet using `boot -h` (see next section for information). In this case, specify the filename relative to the client's partition on the server, not relative to the server's root. For example, typing `boot net myfile.ext -h` at the `ok` prompt specifies the following pathname:
`/export/root/<client>/myfile.ext`

Downloading Files From a Hard Disk

To download and execute a file from your local hard disk:

1. If you are downloading a Forth or FCode file, you must convert the file into an *a.out* format of the required length.

   You can use `fakeboot`, available from the SBus Support Group, to perform the conversion.

2. At the `ok` prompt type `boot disk` *filename.ext* `-h` and press [Return].

   This command downloads the file from the root directory `/filename.ext` of your hard disk.

3. If your downloaded file:

   - ❑ is a binary file, type `init-program go` to execute that program.
   - ❑ contains Forth text and was converted using `fakeboot`, type `4010 4004 @ eval` to interpret the file.
   - ❑ contains FCode binary and was converted using `fakeboot`, type `4030 1 byte-load` to interpret the file.

Downloading Files From a Floppy Disk

To download and execute a file from your floppy disk:

1. Build a bootable floppy:
   - ❑ Use a 1.4MB High-Density (HD) floppy, not a Double-Density (DD) floppy.
   - ❑ Format floppy. Type `/bin/fdformat` and press [Return].
   - ❑ Create a file system on your floppy. Type `/usr/etc/newfs /dev/rfd0a` and press [Return].
   - ❑ Mount your floppy. Type `mount /dev/fd0a /mnt` and press [Return].
   - ❑ Copy the `/boot` program from your local disk or server to the `/mnt` directory on your system.
   - ❑ Install a boot block. Type `cd /usr/mdec` and press [Return]. Then type `installboot /mnt/boot bootfd /dev/rfd0a` and press [Return].
2. If you are downloading a Forth or FCode file, you must convert the file into an *a.out* format of the required length.

   You can use `fakeboot`, available from the SBus Support Group, to perform the conversion.
3. After converting a Forth or FCode file using the `fakeboot` program, copy the file to your floppy. Type `cp` *filename.ext* `/mnt` and press [Return].

   A file copied into `/mnt/filename.ext` will appear in the root directory `/filename.ext` on your floppy.
4. At the `ok` prompt type `boot` fd() *filename.ext* `-h` and press [Return].

   This command downloads the file from the root directory `/filename.ext` of your floppy.

5. If your downloaded file:

   - ❑ is a binary file, type `init-program go` to execute that program.
   - ❑ contains Forth text and was converted using `fakeboot`, type `4010 4004 @ eval` to interpret the file.
   - ❑ contains FCode binary and was converted using `fakeboot`, type `4030 1 byte-load` to interpret the file.

Figure 6-5 lists the file downloading commands.

Figure 6-5. File Downloading Commands

Command	Stack Diagram	Description
`boot` *[specifiers]* `-h`	(--)	Download file from specified source
`dl`	(--)	Download a Forth file over a serial line with `tip` and interpret with: `~C cat` *filename.fth* `^D`
`dlbin`	(--)	Download a binary file over a serial line with `tip`. Type `~C sendbin` *filename.fth*
`dload` *filename*	(addr --)	Load the specified file over Ethernet at the given address
`go`	(--)	Begin execution of previously loaded program or continue execution of an interrupted program
`init-program`	(--)	Initialize to execute binary file
`byte-load`	(addr 1 --)	Interpret downloaded FCode binary. Addr is 4030 for a file converted with `fakeboot`, but usually 4000 otherwise
`eval`	(addr len --)	Interpret downloaded Forth text file. Type `4010 4004 @ eval` for a file converted with `fakeboot`, otherwise type `4000 file-size @ eval`

Ejecting the Floppy Diskette

Your SPARC system might have two types of locally-attached disk drives: a diskette drive and one or more hard disks. Two basic commands provide disk drive control.

The `eject-floppy` command causes the floppy diskette to be ejected from the diskette drive. If this command fails, you can insert a paper clip into the little hole on the drive and physically eject the diskette.

Preserving Data After a System Crash

The `sync` command forces any information on its way to the hard disk to be written out immediately. This is useful if the SunOS Operating System has crashed, or has been interrupted without preserving all data first.

The `sync` command actually returns control to the SunOS Operating System, which then performs the data saving operations. After the disk data has been `synced`, the SunOS Operating System begins to save a *core image* of the operating system. This *core dumping* procedure is preceeded by the following message:

```
dumping to vp xxxxxxxx offset xxxxxx
```

If you do not need this core dump, you can interrupt the operation with L1-A. Figure 6-6 lists commands to control your disk.

Figure 6-6. Disk Control Commands

Command	Stack Diagram	Description
`eject-floppy`	(--)	Ejects the diskette from the floppy drive
`sync`	(--)	Call SunOS Operating System to write any pending information to the hard disk. Also boots after syncing file systems.

Symbolic Names

You can use the commands listed in Figure 6-7 for symbolic debugging. For correct execution, the symbol table needs to be loaded before these commands are invoked. See `dload` in "Downloading Files Over Ethernet" earlier in this chapter.

Figure 6-7. Symbolic Name Commands

Command	Stack Diagram	Description
`.adr`	(adr --)	Display the symbolic name (plus offset) for the given address
symname	(-- adr)	Type any valid symbolic name to get the equivalent address

SunOS Operating System Calls

You can use the command listed in Figure 6-8 to call SunOS Operating System functions.

Figure 6-8. SunOS Operating System Call Commands

Command	Stack Diagram	Description
`wector` *string*	(value --)	Call SunOS with the given value and string

Manipulating the Cache

You can use the commands listed in Figure 6-9 to manipulate the cache.

Note: Use commands ending with ! with caution. Otherwise your system might become hung.

Figure 6-9. Cache Manipulation Commands

Command	Stack Diagram	Description
`flush-cache`	(--)	Invalidate all cache entries. Present in boot PROM versions 1.0 and 1.1.
`clear-cache`	(--)	Invalidate all cache entries. Present in boot PROM version 1.2 and later.
`cache-off`	(--)	Disable the cache
`cache-on`	(--)	Enable the cache
`cdata!`	(data offset --)	Store the 32-bit data at the cache offset
`cdata@`	(offset -- data)	Fetch (return) data from the cache offset
`ctag!`	(value offset --)	Store the tag value at the cache offset
`ctag@`	(offset -- value)	Return the tag value at the cache offset

Reading and Writing Machine Registers

You can use the commands listed in Figure 6-10 to read from and write to machine registers.

Note: Use commands ending with ! with caution. Otherwise your system might become hung.

Figure 6-10. Machine Register Reading and Writing Commands

Command	Stack Diagram	Description
`aerr!`	(data --)	Write asynchronous error register
`aerr@`	(-- data)	Read asynchronous error register
`averr!`	(data --)	Write asynchronous virtual address register
`averr@`	(-- data)	Read asynchronous error virtual address register
`aux!`	(data --)	Write auxiliary register
`aux@`	(-- data)	Read auxiliary register
`context!`	(data --)	Write context register
`context@`	(-- data)	Read context register (MMU context)
`dcontext@`	(-- data)	Read context register (cache context)
`dmaaddr!`	(data --)	Write DMA address register
`dmaaddr@`	(-- data)	Read DMA address register
`enable!`	(data --)	Write system enable register
`enable@`	(-- data)	Read system enable register
`interrupt-enable!`	(data --)	Write interrupt enable register
`interrupt-enable@`	(-- data)	Read interrupt enable register
`serr!`	(data --)	Write synchronous error register
`serr@`	(-- data)	Read synchronous error register
`sverr!`	(data --)	Write synchronous error virtual address register
`sverr@`	(-- data)	Read synchronous error virtual address register

This chapter described ways to control your machine using the boot PROM Toolkit. Chapter 7 describes the special commands you can use to view and change system configuration parameters. If you wish to use the Forth Toolkit to its fullest capacity, please read Chapter 5 for a detailed explanation of the PROM's Forth capabilities, if you have not already done so.

7

Using Configuration Parameters

The system configuration parameters are stored in the system NVRAM. These parameters determine the basic start-up machine configuration and related communication characteristics. This chapter describes how to access and change these parameters.

The procedures described in this chapter assume that you have started the Monitor, entered the Forth Toolkit mode, and the `ok` prompt is displayed on your screen. See Chapter 3 for information about entering the Forth Toolkit.

NVRAM configuration parameters can be viewed and changed using the Toolkit commands listed in Figure 7-1.

Figure 7-1. Configuration Parameter Commands

Command	**Description**
`printenv`	Displays all current parameters and current default values (numbers are shown as decimal values)
`setenv` *parameter value*	Sets the *parameter* to the given decimal or text *value* (Changes are permanent, but usually only take effect after a reset)
`set-default` *parameter*	Resets the value of the named *parameter* to the factory default
`set-defaults`	Resets most parameter values to the factory defaults

Displaying Parameters

To display a list of the current parameter settings, type:

```
ok printenv
```

The system responds by displaying a formatted list of the current parameter settings, similar to the partial list shown in Figure 7-2.

Note: Numeric parameters are displayed in decimal.

Figure 7-2. Partial Configuration Display

```
Parameter Name          Value                               Default Value

sunmon-compat?          true                                true
oem-logo
oem-logo?               false                               false
oem-banner
oem-banner?             false                               false
ttyb-mode               9600,8,n,1,-                        9600,8,n,1,-
ttya-mode               9600,8,n,1,-                        9600,8,n,1,-
```

Figure 7-3 on the next page lists all NVRAM configuration parameters in current SPARCstation 1 and SPARCstation 1+s. This might change with future systems.

Figure 7-3. NVRAM Configuration Parameters

Parameter	Description	Default
`auto-boot?`	If true, boot automatically after power up	True
`diag-switch?`	If true, run in diagnostic mode	True*
`fcode-debug?`	If true, include name fields for plug-in device Fcodes	False
`keyboard-click?`	If true, enable keyboard click	False
`mfg-switch?`	If true, perform repeated system self-tests	False
`oem-banner?`	If true, use custom OEM banner	False
`oem-logo?`	If true, use custom OEM logo (else use SUN Logo)	False
`sunmon-compat?`	If true, come up with old-style monitor prompt '>'	True
`ttya-ignore-cd`	If true, SunOS ignores carrier-detect on ttya	True
`ttyb-ignore-cd`	If true, SunOS ignores carrier-detect on ttyb	True
`ttya-rts-dtr-off`	If true, SunOS does not assert DTR and RTS on ttya	False
`ttyb-rts-dtr-off`	If true, SunOS does not assert DTR and RTS on ttyb	False
`watchdog-reboot?`	If true, reboot after watchdog reset	False
`screen-#columns`	Number of on-screen columns (characters/line)	80†
`screen-#rows`	Number of on-screen rows (lines) used	34†
`scsi-initiator-id`	SCSI bus address of host adapter, range 0-7	7†
`security-#badlogins`	Number of incorrect security password attempts	0†
`selftest-#megs`	Megabytes of RAM to test on power-up or on test-memory	1†
`input-device`	Power-on input device (keyboard, ttya or ttyb)	keyboard
`output-device`	Power-on output device (screen, ttya or ttyb)	screen
`boot-from`	Boot source (default device is sd)	vmunix
`boot-from-diag`	Diagnostic boot source	le()vmunix
`hardware-revision`	System version information	no default
`last-hardware-update`	System update information	no default
`oem-banner`	Custom OEM banner (enabled by oem-banner? true)	empty
`sbus-probe-list`	Which SBus slots are probed and in what order	0123
`ttya-mode`	ttya (baud rate, #bits, parity, #stop, handshake)	9600, 8, n, 1, -†
`ttyb-mode`	ttyb (baud rate, #bits, parity, #stop, handshake)	9600, 8, n, 1, -†
`oem-logo`	Byte array custom OEM logo (enabled by oem-logo? true)	empty
`sd-targets`	Map SCSI disk units, e.g. unit #0 = target #3, etc.	31204567
`st-targets`	Map SCSI tape units, e.g. unit #0 = target #4, etc.	45670123
`testarea`	One-byte scratch field, available for read/write test	0†
`security-mode`	System security level (none, command, full)	none
`security-passwd`	System security password (never displayed)	empty

*The default is true for boot PROM version 1.1 and later, but false for boot PROM version 1.0.
†Values shown in decimal.

Changing a Parameter's Value

Use the `setenv` command to change a parameter setting. The `setenv` command has the following format:

`setenv` *parametername value*

where

parametername is one of the listed parameters.

value is a numeric value or text string appropriate to the named parameter. Numeric values are entered in decimal.

To change the setting of the `auto-boot?` parameter from true to false, enter:

```
ok setenv auto-boot? false
ok
```

This command sets the `auto-boot?` parameter flag to false. This means that the next time the system is powered on or reset the auto-boot feature is turned off. The system will not attempt to boot the SunOS Operating System after self-tests and initialization have completed.

Resetting Default Values

You can reset one or most of the parameters back to the original defaults using the `set-default` and `set-defaults` commands. These commands have the following format:

`set-default` *parametername*

`set-defaults`

where

parametername is one of the listed parameters.

To reset the `auto-boot?` parameter to its original default setting (true), type:

```
ok set-default auto-boot?
ok
```

To reset most parameters to their default settings, type:

```
ok set-defaults
ok
```

Once the default for a parameter is changed or reset, a system reset is usually required for the parameter setting to actually take effect. A system reset (which is very similar to a power cycle) does not necessarily include booting, depending on how the configuration parameters are specified. The parameters that relate to system booting require a system boot for the parameter to take effect. You can use the `reset` command to reset the system when you have changed a parameter.

Figure 7-4 lists additional NVRAM configuration commands, which are used only rarely.

Figure 7-4. Configuration Parameter Command Primitives

Command	Stack Diagram	Description
parameter	(-- value)	Return the (current) field value
`show` *parameter*	(--)	Display the (current) field value (numbers shown in decimal)
`to` *parameter*	(value --)	Change a (current) field value
		Examples: `false to auto-boot?`
		" *Text string*" `to oem-banner`

Security

The security feature of the boot PROM is available on version 1.1 and later boot PROM versions. Setting the `security-mode` parameter to `full` or `command` security restricts the set of actions others are allowed to perform, thus making it more difficult for them to break into your computer network. Figure 7-5 lists the security parameters.

Figure 7-5. Security Parameters

Parameter	Default	Description
`security-mode`	None	System security level (none, command, full)
`security-passwd`	Empty	System security password
`security-#badlogins`	0*	Number of incorrect security password attempts
	*Value shown in decimal	

There are three security modes:

1. No security
2. Command security
3. Full security.

With no security, any command can be executed at the boot prompt > with no password required. `Command` security is the next level of security and `full` security is the most secure. With both `command` and `full` security, passwords are required to execute certain commands at the boot prompt >.

A password is never required from the `ok` prompt, regardless of security mode. However, a password is required to get to the `ok` prompt in either `command` or `full` security mode.

No Security

With no security (default), no password is required for any command at the boot prompt >. Anyone can execute the three commands at the boot prompt > without a password:

- b (boot)
- n (new)
- c (continue).

If you previously set the security to `command` or `full` security and wish to set the system with no security, enter the following:

```
ok setenv security-mode none
```

The next time the system checks the boot PROM's security, it will determine that no security (`security-mode none`) has been set for the user. It is also possible to change the PROM security mode using the `/etc/eeprom` SunOS Operating System command.

Command Security

With the security set to `command` mode, a password is not required if you type the `b` command at the boot prompt >. However, if you follow the `b` command with a parameter, a password is required.

To execute the `n` command from the boot prompt >, a password is required. The `c` command never asks for a password. Examples follow:

```
> b                  (no password required)
> c                  (no password required)
> b filename         (password required)
PROM Password:       (password is not echoed as it is typed)
> n                  (password required)
PROM Password:       (password is not echoed as it is typed)
```

To set the security password and `command` security, enter the following at the `ok` prompt:

```
ok setenv security-password passwd
ok setenv security-mode command
ok old-mode
```

The security password you assign follows the same rules as the `root` password, a combination of six to eight letters and numbers. The security password can be the same as the `root` password, or you can assign a security password different from the `root` password.

Caution: The security password is important to remember. If you forget your security password, your system will be unbootable and you must call Sun's customer support service to make your machine bootable again.

It is not necessary to reset the system. The security feature takes effect as soon as the Sun-Compatible mode (> prompt) is entered.

Note: After setting your security password, it is a good idea to remove the password from the screen to prevent someone from seeing it. Press the Return key several times to remove the password from the screen.

If you enter an incorrect security password, there will be approximately a 10 second delay before the next boot prompt > appears. The number of times that an incorrect security password is typed is stored in the `security-#badlogins` parameter. This parameter is a 32-bit signed number (680 years worth of attempts at 10 seconds per attempt). This parameter can be set to 0 with the `setenv` command. Its value can be displayed with the `printenv` command. An example of setting the number of badlogins to 0 follows:

```
ok setenv security-#badlogins 0
```

Note: If you enter the `boot` command in command or full security mode, the PROM will revert to the > prompt the next time that the PROM command interpreter is entered.

Full Security

The `full` security mode is the most restrictive. With the security set to `full` mode, a password is required any time you type the `b` command at the boot prompt > (either `b` alone or `b` followed by a parameter).

A password is required to execute the `n` command from the boot prompt >. The `c` command never asks for a password. Examples follow:

```
> c                     (no password required)
> b                     (password required)
PROM Password:          (password is not echoed as it is typed)
> b filename            (password required)
PROM Password:          (password is not echoed as it is typed)
> n                     (password required)
PROM Password:          (password is not echoed as it is typed)
```

To set the security password and `full` security, enter the following at the `ok` prompt:

```
ok setenv security-password passwd
ok setenv security-mode full
ok old-mode
```

Changing the Power-On Banner

You can use the `banner` command to view the power-on banner. Figure 7-6 lists the configuration parameters that control the power-on system banner.

Figure 7-6. Banner Configuration Parameters

Parameter	Default	Description
`oem-banner?`	False	When true, the default Sun banner message displayed during system power up is replaced with whatever text string is present in the `oem-banner` parameter text field
`oem-logo?`	False	When true, the data array specified in the `oem-logo` field is substituted for the Sun logo in the power-on banner
`oem-banner`	Empty	Custom banner (enabled by `oem-banner?` true)
`oem-logo`	Empty	Byte array custom logo (enabled by `oem-logo?` true)

To display the system power-on banner, enter:

```
ok banner
```

The PROM displays the system banner. The following banner is the SPARCstation 1 banner. The banner for your SPARC system might be different.

```
SPARCstation 1: Type 4 Keyboard
ROM Rev. 1.0, 8MB memory installed, Serial # 312
Ethernet Address 8:0:20:6:5:16 , Host ID:51000174
```

The banner consists of two parts: the text field and the logo (over serial ports, only the text field is displayed). You can replace the existing text field with a custom text message using the `oem-banner` and `oem-banner?` configuration parameters.

To insert a custom text field in the power-on banner, enter:

```
ok setenv oem-banner Hello Mom and Dad
ok setenv oem-banner? true
ok banner
ok
```

The system displays the banner with your new message.

```
Hello Mom and Dad
```

However, the graphic logo must be handled somewhat differently. The `oem-logo` field is a 512-byte array, containing a total of 4096 bits arranged in a 64 x 64 array. Each bit controls one pixel. The most significant bit (MSB) of the first byte controls the upper-left corner pixel. The next bit controls the next pixel to the right and so on.

To create a new logo, you must first create a Forth array containing the correct data and then copy this array into the `oem-logo` field. For the following example, the array is created using Forth Toolkit commands. This command could also be done under SunOS Operating System using the `/etc/eeprom` command. This array is then copied using the `to` command, which is an NVRAM primitive. The following example fills the top half of the `oem-logo` field with an ascending pattern, and leaves the bottom half unchanged.

```
ok create logoarray d# 512 allot
ok logoarray d# 256 0 do i over i + c! loop drop
ok logoarray d# 256 to oem-logo
ok setenv oem-logo? true
ok banner
```

The system displays the power-on banner with the new logo array.

New Logo Array display

```
Hello Mom and Dad
```

To restore the original Sun power-on banner, set the oem-logo? and oem-banner? parameters to false.

```
ok setenv oem-logo? false
ok setenv oem-banner? false
ok
```

Because the oem-logo array is so large, printenv only displays the first 8 bytes or so (in hex). To display the entire array, use the oem-logo dump command. The oem-logo array is not erased by set-defaults, since it might be difficult to restore the data. However, oem-logo? will be set to false by execution of set-defaults, so the custom logo will no longer be displayed.

Input and Output Control

Figure 7-7 lists the configuration parameters related to the control of system input and output. You can use these parameters to assign the power-on defaults for input and output, and to adjust the communication characteristics of the ttya and ttyb serial ports. These values do not take effect until the next system reset.

Figure 7-7. Input and Output Control Parameters

Parameter	Default	Description
input-device	keyboard	Power-on input device (keyboard, ttya, or ttyb)
output-device	screen	Power-on output device (screen, ttya, or ttyb)
ttya-mode	9600, 8, n, 1, -*	ttya (baud, #bits, parity, #stop, handshake)
ttyb-mode	9600, 8, n, 1, -*	ttyb (baud, #bits, parity, #stop, handshake)
screen-#columns	80*	Number of on-screen columns (characters/line)
screen-#rows	34*	Number of on-screen rows (lines)
	*Values in decimal	

Setting Serial Port Characteristics

The default settings for both ttya and ttyb are:

9600 baud
8 data bits
no parity
1 stop bit
no handshake

The communications characteristics for the two serial ports, ttya and ttyb, are set using the following values for the parameters `ttya-mode` and `ttyb-mode`.

```
baud, #bits, parity, #stop, handshake
```

where:

baud	110, 300, 1200, 2400, 4800, 9600, 19200, 38400 (bits/second)
#bits	5, 6, 7, 8 (data bits)
parity	n=none, e=even, o=odd, m=mark, s=space (parity bit)
#stop	1=1, . =1.5, 2=2 (stop bits)
handshake	-=none, h=hardware (rts/cts), s=software (xon/xoff)

To set ttya to 1200 baud, seven data bits, one stop bit, even parity, and no handshake, type:

```
ok setenv ttya-mode 1200,7,e,1,-
ok
```

Note: rts/cts and xon/xoff handshaking are not implemented on all systems. In this case, the handshake parameter is silently ignored.

Selecting Input and Output Device Options

The `input-device` and `output-device` parameters control the system's selection of input and output devices after a power-on reset. The default `input-device` value is `keyboard` and the default `output-device` value is `screen`. Input and output can be set to the following values:

input-device	**output-device**
`keyboard*`	`screen**`
`ttya`	`ttya`
`ttyb`	`ttyb`

* `keyboard` implies standard Sun keyboard

**`screen` implies frame buffer video display

When the system is reset, the named device becomes the default input or output device.

If you wish to temporarily change the input or output device, use the `input` or `output` commands described in Chapter 6.

To set ttya as the power-on default input device, type this command:

```
ok setenv input-device ttya
ok
```

Note: If `keyboard` is selected for `input-device` but is not plugged in, or if `screen` is selected for `output-device` but no on-board frame buffer is available, then both input and output will be sent via ttya after the next power cycle or system reset.

Selecting Boot Options

You can use the configuration parameters to determine whether the system will automatically boot after the system start-up tests and initialization. In addition, the parameters can be used to select the boot device and the program to be booted. Figure 7-8 lists the parameters that control boot options.

Figure 7-8. Boot Options Parameters

Parameter	Default	Description
`auto-boot?`	True	Determines whether the system will automatically boot after the power-on self-test and system initialization. If true and `diag-switch?` is set to false, the Open PROM attempts to boot whatever file is specified by the `boot-from` parameter.
`boot-from`	vmunix	Boot source filename (default device is sd)

The `boot-from` parameter defaults to the filename `vmunix`. The `boot-from` parameter is used during auto-boot or when you boot the system manually without specifying a filename. If no device is specified, the default device is assumed to be the system's internal hard disk. However, you can use the `boot-from` parameter to specify a different device and file. For example, to specify the file `myunix` to be auto-booted single-user from the Ethernet server, type:

```
ok setenv boot-from le()myunix -s
ok boot                    Specified booting begins immediately
```

Controlling Power-On Self-Test

Enabling the diagnostic switch parameter `diag-switch?` causes the system to perform a more thorough self-test during power-on. After `diag-switch?` is enabled, additional status messages are sent out (some to ttya and some to the specified output device) and *all* of memory is tested.

Figure 7-9 lists the power-on testing parameters. The default value of `diag-switch?` is true, but the actual value is set to false at the factory. If the default values are restored with `set-defaults`, the `diag-switch?` value becomes true.

Figure 7-9. Power-On Testing Parameters (continued on next page)

Parameter	Default	Description
`diag-switch?`	True	When `diag-switch?` is true, the system calls out diagnostic tests as they are executed (at power-on time) and performs complete memory tests (all of memory is tested). Each Power-On Self-Test prints its name (either to ttya or to the default output device) as it begins to execute, and the boot PROM attempts to boot the program specified by the `boot-from-diag` parameter. When `diag-switch?` is false, the system will not call out the diagnostic tests as they are run, unless a test fails, and will not run any additional tests. **Note:** The default value is true for boot PROM version 1.1 and later, but false for boot PROM version 1.0. Regardless of the default value, all systems are shipped from the factory with a current value of false for `diag-switch?`

Figure 7-9. Power-On Testing Parameters (continued)

Parameter	Default	Description
`mfg-switch?`	False	When true, the system repeats the power-on self-test and initialization sequence until interrupted with the L1-A key sequence.
`selftest-#megs`	1*	Number of megabytes of RAM to test on power-up or on `test-memory`. This value is ignored if `diag-switch?` is true.
`boot-from-diag`	le()vmunix	Diagnostic boot source filename
	*Value shown in decimal.	

The `selftest-#megs` parameter determines how much of the RAM will be tested during the power-on self tests. The default for this parameter is one megabyte.

When the `mfg-switch?` parameter is set to true, the system repeats power-on self-test and initialization until interrupted with an L1-A key sequence. For example:

To power-up in diagnostic mode if `diag-switch?` is set to false:

1. Set the `diag-switch?` parameter to true.
2. Reset the system.

```
ok setenv diag-switch? true
ok reset
```

See your system's Field Service Manual for more information about using diagnostics.

This chapter described the configuration parameters contained in NVRAM. Changes made to these parameters are permanent. The configuration parameter commands listed in Figure 7-1 have been created to simplify using these parameters.

However, configuration parameters should always be adjusted cautiously. When correctly used, these configuration parameters allow you flexibility in working with your system's hardware.

A

Sun Monitor Command Equivalents

Figure A-1 lists the most commonly used Sun Monitor commands, with the Forth Toolkit commands that perform equivalent functions.

Figure A-1. Command Equivalents (continued on next page)

Sun Monitor Command	**Forth Toolkit Command**
^C *source-addr dest-addr len*	*source-addr dest-addr len* `move`
^I	`.version`
^T *address*	*address* `map?`
b *[boot spec]*	`boot` *[boot spec]*
	addr `dload` *pathname*
c *address*	`go` or *address* `set-pc go`
d *window#*	`.registers`
	`.locals`
	window# `.window`
e *address action*	*address* `w?`
	value address `w!`

Figure A-1. Command Equivalents (continued on next page)

Sun Monitor Command	Forth Toolkit Command
`f` *addr 1 addr 2 pattern size*	*addr #bytes byte* `cfill` *addr #bytes shortword* `wfill` *addr #bytes longword* `lfill`

`g` *vector argument*	`go`
	`sync`
	value `wector` *argument*

`h`	`help` `help` *name* `help` *category*
`i` *cache-data-offset action*	*address* `cdata@ .` *value address* `cdata!`
`j` *cache-tag-offset action*	*address* `ctag@ .` *value address* `ctag!`
`k1` or `k2`	`reset`
`kb`	`banner`
`l` *address action*	*address* `L?` *value address* `L!`
`m` *address action*	*address* `smap?` *value address* `smap!`
`o` *address action*	*address* `c?` *value address* `c!`
`p` *address action*	*address* `pgmap@` *address* `map?` *pte address* `pgmap!` *physical space# virtual* `map-page` *physical space# virtual size* `map-pages` *space#*: `obmem`, `obio`, `sbus`

`q` *offset action*	`printenv`
	`setenv` *parametername value*
	`set-default` *parametername*
	`set-defaults`

Figure A-1. Command Equivalents (continued)

Sun Monitor Command	**Forth Toolkit Command**
s *step-count*	step *step-count* steps
r *register-number action*	*registername* . *value* to *register-name*
s *space#*	*address space#* spacec? *address space#* spacew? *address space#* spaceL? *value address space#* spacec! *value address space#* spacew! *value address space#* spaceL!
t *program*	*n* steps
u *various options*	*device* input *device* output *device* io *devices*: ttya,ttyb,screen,keyboard
v *addr1 addr2*	*address size* dump
w address argument	*value* wector *argument*
x	test-control-regs test-net test-cache test-memory test-floppy watch-clock
z *number address type len*	*address* +bp *address* -bp
	return returnL .bp till finish-loop

B

Power-On Self-Test

Figure B-1 lists the Power-on Self-Test (POST) for the SPARCstation 1, with brief descriptions.

Figure B-1. POST Descriptions (continued on next page)

Test Name	Description
PROM Checksum	Calculates the checksum of the PROMs and compares the calculated value with the expected value.
Segment Map Address	Writes the number of each segment map location to that location, then reads back the value and compares it with the expected value. The entire range of segment map addresses is written prior to the read and compare operation. When an error is detected, the test loops on the error location.
Page Map Address	Writes the number of each page map location to that location, then reads back the value and compares the observed value with the expected value. The entire range of Page Map addresses is written prior to the read and compare operations. When an error is detected, the test loops on the error location.
Context Register	Performs write-read-compare cycles on the context register using patterns 0x08, 0x07, through 0x00.

Figure B-1. POST Descriptions (continued)

Test Name	Description
Synchronous Error Register	Performs write-read-compare cycles on the synchronous error register, using patterns 0xff, 0xfe, through 0x00, and also patterns 0x80ff, 0x80fe, through 0x800.
Synchronous Error Virtual Address Register	Performs write-read-compare cycles on the synchronous error virtual address register using patterns 0xffffffff, 0x0, 0x00000001, 0x00000002, 0x00000004, 0x00000008, through 0x80000000.
Asynchronous Error Register	Performs write-read-compare cycles on the asynchronous error register, using patterns 0xb0, 0xa0, 0x90, 0x80, 0x30, 0x20, 0x10, and 0x00.
Asynchronous Error Virtual Address Register	Performs write-read-compare cycles on the asynchronous error virtual address register using patterns 0xffffffff, 0x0, 0x00000001, 0x00000002, 0x00000004, 0x00000008, through 0x20000000.
System Enable Register	Examines the system enable register for correct bits set.
System Memory	Tests main memory. The number of megabytes tested is indicated by the NVRAM parameter `selftest-#megs`.

Index

Note: Page references for commands appear in bold.

sun
microsystems

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
FAX 415 969-9131

Part Number: 800-5279-10
Revision B of January 1991