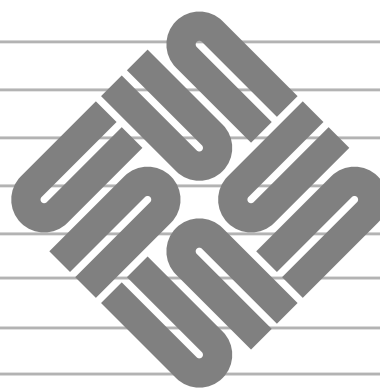




Open Boot PROM Toolkit User's Guide



Part No: 800-4251-10
Revision A (FCS) of 5 December 1989

Copyright ©1989 Sun Microsystems, Inc.—Printed in U.S.A.

The Sun logo, Sun Microsystems, and Sun Workstation are registered trademarks of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, Sun386i, SunInstall, SunOS, SunView, NFS, SunLink, NeWS, SPARC, and SPARCstation 1 are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T.

The Sun Graphical User Interface was developed by Sun Microsystems, Inc., for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox for the Xerox Graphical User Interface, which license also covers Sun's licensees.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Sun Microsystems, Inc., disclaims any responsibility for specifying which marks are owned by which companies or organizations.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means—graphic, electronic, or mechanical—including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

This product is protected by one or more of the following U.S. patents: 4,777,485; 4,688,190; 4,527,232; 4,745,407; 4,679,014; 4,435,792; 4,719,569; 4,550,368 in addition to foreign patents and applications pending.

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 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.

How to Use This Book

This book consists of seven chapters and four appendices.

- ❑ 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 the Toolkit functions. You will use this chapter when you are performing typical field service tasks.
- ❑ Chapter 6 describes working with NVRAM configuration parameters. You will use this chapter if you are performing typical system administration tasks.
- ❑ Chapter 7 describes advanced Toolkit functions. You will use this chapter when you are performing field service troubleshooting.
- ❑ Appendix A is a command reference for Toolkit commands.
- ❑ Appendix B is a list of the NVRAM Configuration Parameters.
- ❑ Appendix C compares other Sun System PROM commands with commands used with the open boot PROM.
- ❑ Appendix D is a list of the Power-On Self-Tests (POST) with brief descriptions.

Related Books

This book is part of the SBus Developer's Kit, a set of publications available from Sun Microsystems, Inc.:

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

In addition, the following books may be useful to you:

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

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. In order 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.

Typographical Conventions

This book follows a number of typographical 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 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 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 Control-d, press and hold Control, then press 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.

Contents

About This Bookix

How to Use This Bookx

Related Booksxi

Before Reading This Bookxi

Chapter 1. Overview1

Programmable Read Only Memory1

Non-Volatile Random Access Memory2

PROM User Interfaces2

Sun-Compatible Monitor2

Forth Toolkit3

Where to Find What You Need4

Chapter 2. System Start-Up and Auto-Booting7

Power-On Self-Test (POST)7

Auto-Boot Procedure8

Chapter 3. Using the Sun-Compatible Monitor9

Starting the Monitor9

Performing a Power Cycle11

Interrupting Power-Up Sequence12

Halting the Operating System13

Aborting a Hung System14

	Compatible Monitor Functions	15
	Booting From the > Prompt	16
	Boot Command Syntax	17
	Continuing a Halted Program	19
	Entering the Forth Toolkit	19
	Returning to the > Prompt	19
Chapter 4.	Forth Toolkit Fundamentals	21
	Forth Commands (Words)	22
	Getting Help	23
	Numbers	24
	The Stack	25
	Showing the Stack With showstack	25
	Stack Diagram	26
	Colon Definitions	27
	Keyboard Editor	29
	Using Control Key Combinations	29
	Using Escape Key Combinations	29
	For Further Reference	30
Chapter 5.	Using the Forth Toolkit	33
	Resetting the System	34
	Diagnostic Routines	35
	Displaying System Information	38
	Booting the System From the Toolkit Prompt	38
	Input, Output, and Display Modes	38
	Emergency Procedure	40
	Setting Up a tip Connection	41
	Sending a “Break”	43
	Ending the tip Session	43
	Common Problems With tip	44
	Downloading Files	45
	Downloading a File Over a Serial Line	45
	Downloading a File Over Ethernet	46
	Ejecting the Floppy Diskette	48
	Preserving Data After a System Crash	48

Chapter 6. Using Configuration Parameters	51
Displaying Parameters	52
Changing a Parameter's Value	53
Resetting Default Values	53
Security	55
No Security	55
Command Security	56
Full Security	57
Changing the Power-On Banner	59
Input and Output Control	62
Setting Serial Port Characteristics	62
Selecting Boot Options	64
Controlling Power-On Self-Test	65
Miscellaneous Parameters Descriptions	66
Chapter 7. More Forth Tools	69
Showing the Stack	70
Using 32-Bit Numbers	70
Manipulating the Stack	71
Numeric Input and Output in Different Bases	72
Using Arithmetic	74
Accessing Memory	75
Examples	78
Using Defining Words	80
Searching the Dictionary	82
Controlling Text Input and Output	83
Interpreting Source Code	85
Using Conditional Testing	87
Controlling Conditional Execution	89
Using Conditional Loops	90
Using Counted Loops	92
Using Case Statements	94
Additional Control Commands	95
Using the Disassembler	96
Displaying Registers	97
Using Breakpoints	99

Appendix A. Toolkit Command Reference	101
Basic System Commands	103
System Resetting Commands	103
Diagnostic Tests	103
System Information	104
Disk Drive Control	104
Help and Mode Commands	104
Booting the System	105
Basic Forth Commands	107
Manipulating the Stack	107
Accessing Memory	108
Using Arithmetic	109
Changing the Numeric Base	110
Displaying Output	110
Output Display Primitives	111
Line Editor Commands	112
Advanced Forth Programming Commands	113
Defining Words	113
Performing Comparisons	114
Inputting Text	115
Displaying Text Output	115
Conditional Loops	116
Counted Loops	116
Controlling Program Execution	117
Conditional Case Statements	117
Manipulating Text Strings	118
Compiling the Dictionary	119
Searching the Dictionary	120
Advanced System Commands	120
Input Output Display Modes	120
File Downloading	121
Using the Disassembler	121
Using Breakpoints	122
Reading and Writing SPARC Registers	123
Symbolic Names	124
Traps	124

Contents

Mapping Memory	125
Mapping Memory Primitives	126
Accessing Alternate Address Space	127
Manipulating the Cache	128
SunOS Operating System Calls	128
Reading and Writing Machine Registers	129
Appendix B. NVRAM Configuration Parameters Summary	131
Appendix C. Sun Monitor Command Equivalents	135
Appendix D. Power-On Self Test	139



Contents

Overview

This chapter is a brief overview of the following:

- ❑ 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 may also see Appendix C 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 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 may 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).

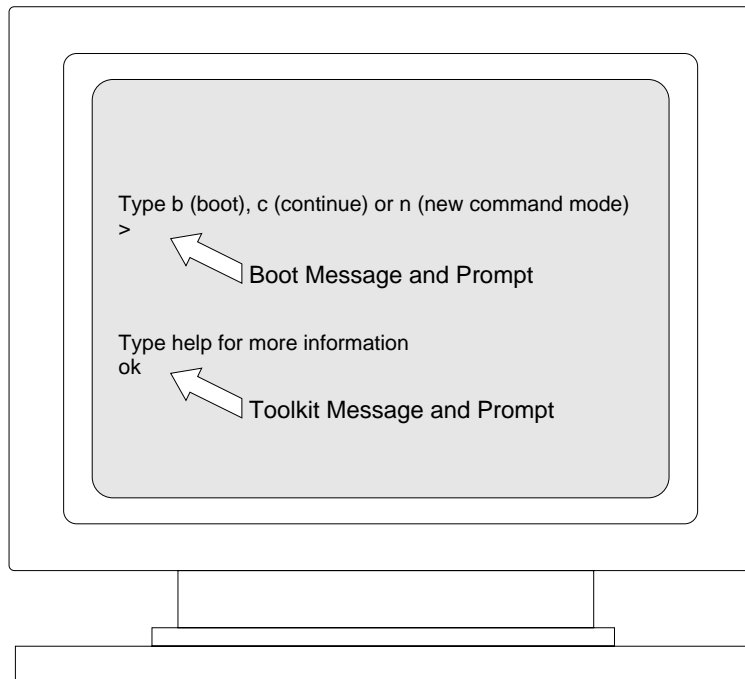
The Sun-Compatible Monitor mode is provided to present a compatible interface to the most common PROM use, booting the system.

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.

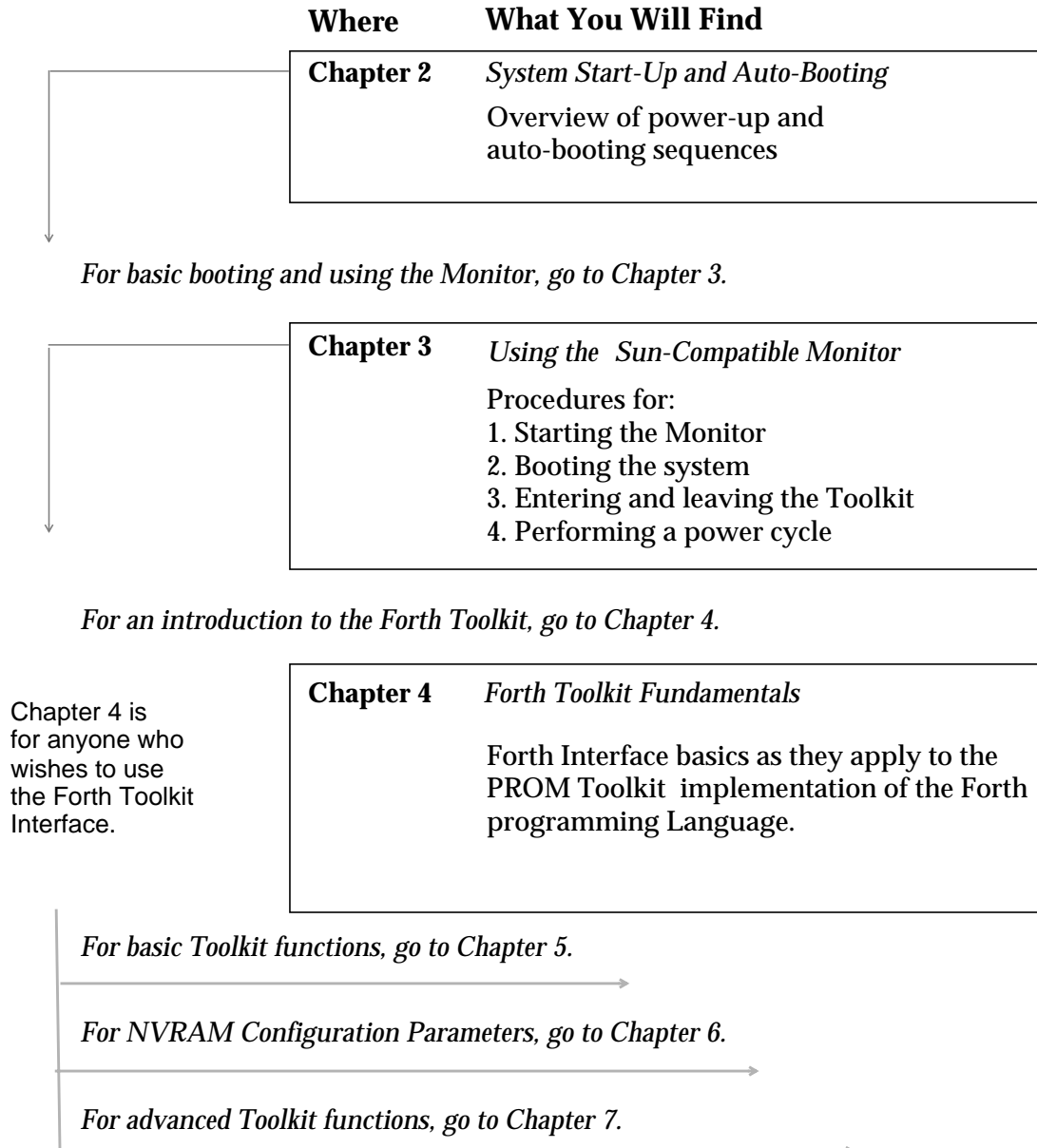
The Toolkit is an interactive command **Forth Toolkit** Forth programming language. When you enter the toolkit, you see the ok prompt. The Toolkit gives you access to an extensive set of functions for performing hardware development, problem determination (fault isolation), software development, and debugging. All functions available through

the Sun-Compatible Monitor mode are also available through the Forth Toolkit.



Where to Find What You Need

The following shows where you will find the important information in this manual.



Chapter What You Will Find

Chapter 5 describes the basic control functions.

Chapter 5 *Using the Forth Toolkit*

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 6 describes working with the NVRAM configuration parameters. You will have to use the Toolkit to do this.

Chapter 6 *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

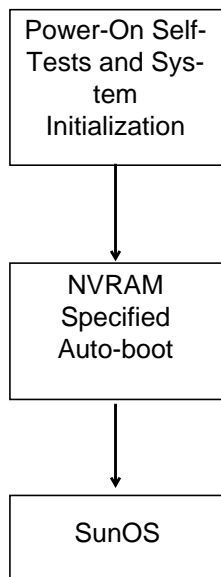
Chapter 7 describes more complex Forth operations. The functions provide advanced programming capabilities.

Chapter 7 *More 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

System Start-Up and Auto-Booting

Power-On Self-Test (POST)



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

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, to initialize the system hardware, and to boot the SunOS Operating System.

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

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

Auto-Boot Procedure



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

When the system test and initialization are completed, the auto-boot procedure begins. By default, 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 may be modified using the PROM Toolkit. You may change the default parameter settings to specify another program to be booted or another `boot-from` device. See Chapter 6 for procedures for modifying NVRAM configuration parameters.

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.

Using the Sun-Compatible Monitor

This chapter describes accessing the boot PROM interface and using 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. The three ways to start the interface are summarized in Figure 3-1.

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*. That is, it 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	<ol style="list-style-type: none"> 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 <code>Testing</code> appears on the screen, press <code>(L1-A)</code> (or the <code>(Break)</code> key for an ASCII terminal)
Aborting SunOS Operating System	<ol style="list-style-type: none"> 1. Press <code>(L1-A)</code> (or the <code>(Break)</code> key for an ASCII terminal) 2. At <code>></code> prompt, type <code>n</code> 3. At <code>ok</code> prompt type <code>sync</code> 4. Press <code>(L1-A)</code> again when you see the word <code>rebooting</code> (or the <code>(Break)</code> key for an ASCII terminal) 4. At the <code>ok</code> prompt type <code>old-mode</code> to return to the <code>></code> prompt
Halting SunOS Operating System	<ol style="list-style-type: none"> 1. Save and quit all open files 2. Quit all applications 3. In a shell window, become the system superuser and type: <code>/etc/halt</code>

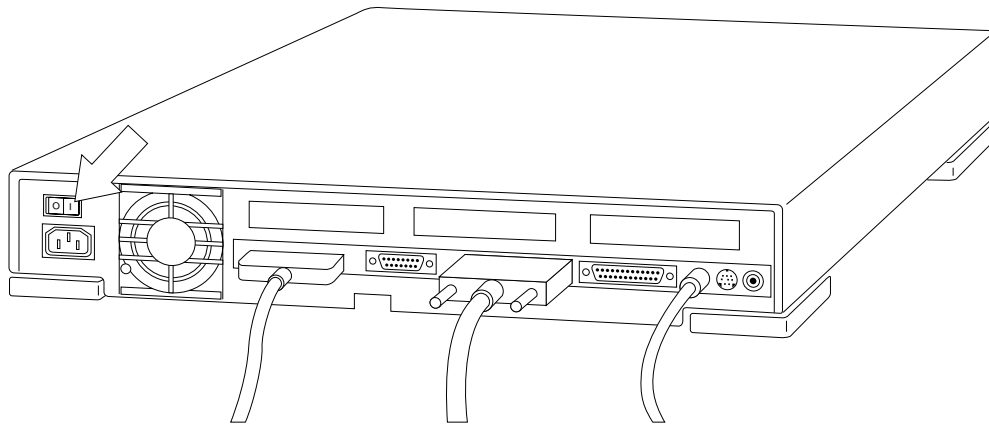
Performing a Power Cycle

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 may 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 **(LI-A)** keys simultaneously. Or, if your console device is a terminal, press the **(Break)** key.

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 more 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)
>
```

When the operating system appears to be hung, the system does not respond to the machine monitor (mc) command. 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

Aborting a Hung System

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:



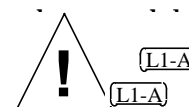
On some keyboards, **[L1]** appears on the front face of the **[Stop]** key. On a system that has a terminal as a console, rather than a Sun keyboard and bitmapped monitor, you must press **[Break]** instead of **[L1-A]** to obtain a boot prompt.

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 program has already booted, it is possible to halt the machine. Aborting program execution 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 may choose to disable Sun-Compatible Monitor mode using NVRAM parameters. See Chapter 6 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 examples below.

The following boot commands include boot commands invoked from the Sun-Compatible Monitor prompt, >, and the Forth Toolkit prompt, ok.

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

Examples of boot commands follow.

```

> b                               Boot system using defaults
ok boot

> b -as                           Boot sd0 with flags a (interactive
                                flag) and s
                                (single-user operation)
ok boot -as

> b le()                          Boot vmunix from the network
> b net
ok boot net
ok boot le()

> b sd(0,0,2)mydiag              Boot mydiag from SCSI drive
                                partition 2
ok boot sd(0,0,2)mydiag

```

Note: Boot defaults may be changed using NVRAM configuration parameters. The NVRAM defaults are only used if the boot command has no arguments. See Chapter 6 for more 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 shows a list of the boot commands and their syntax.

Figure 3-2. Boot Command Options Summary

Option	Description
<i>device</i> is one of:	<p> <code>leX (c,u,p)</code> LANCE Ethernet <code>sdX (c,u,p)</code> SCSI Hard Disk <code>stX (c,u,p)</code> SCSI Tape <code>fdX (c,u,p)</code> 3 1/2" Floppy Disk Drive </p> <p><i>X</i> is the device number, such as <code>le0</code>, <code>sd0</code>, or <code>fd0</code></p> <p><i>c</i> Controller Number, default value = 0</p> <p><i>u</i> Unit Number, default value = 0; when booting from a SCSI disk the range may be from 0-3.</p> <p><i>p</i> Partition Number, default value = 0; when booting from a SCSI disk the range may be from 0-7.</p> <p>When using <code>le</code>, <code>sd</code> and <code>fd</code> as device identifiers, the parentheses are required in the command line. Example: <code>ble()</code> or <code>ble(0,0,0)</code>. The contents of the parentheses depends on the specified device.</p>
<i>filename</i>	<p>Default = <code>vmunix</code></p> <p>The name of the program to be booted, such as <code>stand/diag</code> or <code>vmunix</code>. <i>filename</i> is relative to the root of the selected device and partition (if specified). <i>filename</i> never begins with <code>'/'</code>. If <i>filename</i> is not given, the boot program uses the default file name <code>vmunix</code>.</p>
<i>options</i>	<p>-a Prompts interactively for the device and name of the file to boot.</p> <p>-b Pass the <code>-b</code> flag through the kernel to <code>init (8)</code> to skip execution of the <code>/etc/rc.local</code> script.</p> <p>-h Halt after loading the program.</p> <p>-s Pass the <code>-s</code> flag through the kernel to <code>init (8)</code> for single-user operation.</p> <p>-i <i>initname</i></p> <p>Pass the <code>-i initname</code> to the kernel to tell it to run <i>initname</i> as the first program rather than the default <code>/single/init</code>.</p>

Continuing a Halted Program

The `c` command is useful when you've halted the SunOS Operating System or some other program. To resume execution of a halted program enter:

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

Program execution resumes. Once execution has resumed, you may wish to 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

Once you have entered the Toolkit all the functions available from the boot prompt are also 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.

Once you have entered the Forth Toolkit, you can work closely with your system's hardware.

The remaining chapters describe using the Forth Toolkit. Even if you are already familiar with the operation of the Forth programming language, it is recommended that you read Chapter 4 which describes how Forth is implemented in the boot PROM.

4

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, it is recommended 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 7 “More Forth Tools” describes using many of the Toolkit’s advanced functions.

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 may 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)



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

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
```

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
ok testa testb testc
ok
```

is exactly equivalent to:

```
ok
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`, may 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
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.

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 section “Selecting the Numeric Base” in Chapter 7 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 “+” as shown below:

```
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
ok 3 5 + .
8
ok
```

The stack is now empty

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. “.” 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`. `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
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
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 may then be downloaded whenever it is needed. See “Downloading Files” in Chapter 5 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 as shown here:

```
ok
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. It is also recommended that you use generous stack comments within the middle of complex definitions, to help trace the flow of execution.

For example, when creating add4 in a text file, 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 “)”. And, 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 4-1 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 4-1. Line Editor Commands

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

For Further Reference

For further reading, refer to 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 (not 16-bit), and 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 contains information about how to perform specific tasks using the Toolkit. Chapter 6 describes working with the NVRAM configuration parameters. Chapter 7 describes how you can use advanced Forth functions for writing programs that interact with your system's hardware.

Using the Forth Toolkit

This chapter describes how to use the open boot PROM's Forth Toolkit. The functions described in this chapter include how 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 generally familiar with how the Forth Toolkit interface operates.

Resetting the System

Occasionally you will find it necessary to reset the system. The `reset` command, listed in Figure 5-1, 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 begins immediately. This system reset is very similar to a power cycle. All Forth definitions you entered are forgotten.

Figure 5-1. System Resetting Command Summary

Command	Stack Diagram	Description
<code>reset</code>	(--)	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. See the command summary in Figure 5-2 for a list of the available diagnostic tests.

Figure 5-2. Diagnostic Routines Command Summary

Command	Stack Diagram	Description
probe-scsi	(--)	Determine the attached SCSI devices
test-control-regs	(--)	Test registers (context, sync, sync vert, 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)
test-floppy	(--)	Test the floppy drive
watch-clock	(--)	Test the clock function
watch-net	(--)	Watch the Ethernet for valid packets

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 may 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 System

The diskette drive test determines whether the diskette drive is functioning properly. A formatted 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.

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 5-3, 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 5-3. System Information Command Summary

Command	Stack Diagram	Description
<code>banner</code>	(--)	Displays power-on banner
<code>.enet-addr</code>	(--)	Displays the current Ethernet address
<code>.idprom</code>	(--)	Displays IDPROM contents, formatted
<code>.version</code>	(--)	Display the version and date of boot PROM

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 identically 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 more 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 `LI-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 may be changed if desired. See “Changing a Parameter’s Value” in Chapter 6 for more information about changing defaults.

The standard baud rate and digital signal transmission settings for both `ttya` and `ttyb` are as follows: 9600 baud, 8 data bits, 1 stop bit, no parity, and no handshaking. These settings may 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 can not 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 may also need to restore the output to the screen connection as well by typing `screen output` and press `[Return]`. Of course, you won’t be able to see any characters being echoed as you type. And if you make a mistake, you won’t be able to see the error message. If this doesn’t work correctly, maybe you need to type `n` and press `[Return]` to enter the Toolkit, and then type `screen output` and press `[Return]`.

Similarly, a break sent over a serial line will grab the input source to that serial line.

Figure 5-4 shows a summary of the commands you can use to redirect input and output.

Figure 5-4. Input, Output, and Display Commands

Command	Stack Diagram	Description
<code>input</code>	(source --)	Select source for subsequent input (ttya, ttyb, or keyboard)
<code>output</code>	(source --)	Select source for subsequent output (ttya, ttyb, or screen)
<code>io</code>	(source --)	Select source for subsequent input and output
<code>[LI-A]</code> (from keyboard)	(--)	Redirect input to come from keyboard
<code>[Break]</code> (from serial port)	(--)	Redirect input to come from serial port

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). 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.

The `tip` method is highly recommended because 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 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 should be made with a 3-wire Null Modem Cable. Refer to your Installation Guide for specifications on null modem cables.
2. At the Sun Workstation, add the following lines to the file `/etc/remotefs`:

```
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 should 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 may continue with the next step in this procedure.

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

```
ok tttya io
```

No echoed response

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

Sending a “Break”

Special commands are sent from the `tip` window to your system using the tilde “~” character. When you wish to send a break to your system from the `tip` window, use the `~#` command. This command acts similarly to an `[LI-A]` from your system’s keyboard. Using `~#` will interrupt any activity in progress, and cause the subsequent input to come from the `tip` window.

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 then `~#`.



Caution: Do not type an `[LI-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’re finished using the `tip` window, you need to end the `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 should see the host prompt.

```
ok
ok ~.

hostname%
```

Common Problems With `tip`

Following are common problems with `tip`:

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` must be 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`.

File downloading commands let you **Downloading Files**
Forth text file over a serial connect

Downloading a File Over a Serial Line

SPARC system and a Sun Workstation (or another SPARC system the same type as your SPARC system). You may also download Forth or binary files over the 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.

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. Files downloaded in this manner should be no larger than 32K. If you need to interpret a larger file, break it into pieces and download each piece with a separate `dl` command.

Downloading a File Over Ethernet

You may download any file over Ethernet with the `dload` command. `dload` requires that you specify the address into which you want to download the file. It is generally best to direct the file into location 4000, a “known and well behaved address.” In the following procedure example, the address 4000 is used.

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 filename.ext` and press `[Return]`.
2. If the downloaded file is a binary file, then `go` will execute that program.
3. If the downloaded file contains Forth text beginning with a backslash and a space (`\ [Space]`) then the word `?go` will interpret the file correctly.

```
ok 4000 dload filename.ext
ok ?go
ok
```

An alternative to `?go` would be to use `eval` as in:
`4000 file-size @ eval`. See the section "Interpreting Source Code" in Chapter 7 for more information.

Note: To download a Forth or binary file from a floppy disk or from the hard disk, use the `boot` command with the `-h` flag. This leave the file at location 4000, just like the `dload` example shown above. You may then use the `go` command (for a binary file), or the `?go` command (for a Forth file) as desired. In order for this method to work, the file must begin with an `a.out` header.

Figure 5-5 is a summary of file downloading commands.

Figure 5-5. File Downloading Command Summary

Command	Stack Diagram	Description
<code>boot [specifiers] -h</code>	(--)	Download file from specified source
<code>dl</code>	(--)	Download a Forth file over serial line with "tip" and interpret with: ~C cat { <i>filename.fth</i> } ^D
<code>dload filename</code>	(addr --)	Load the specified file over Ethernet, at the given address
<code>go</code>	(--)	Begin execution of a previously loaded program or continue execution of an interrupted program
<code>?go</code>	(--)	Interpret downloaded Forth source file. (The file must begin with: \ (Space))

Ejecting the Floppy Diskette

Your SPARC system may 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 may 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 have no need for this core dump, you can interrupt the operation with **(L1-A)** or **(Break)**.

Figure 5-6 is a summary of the disk control commands.

Figure 5-6. Disk Control Command Summary

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.

This chapter described some of the fundamental tasks you may need to perform using the boot PROM Toolkit. Chapter 6 describes the special commands you can use to view and change system configuration parameters. And, if you wish to use the Forth Toolkit to its fullest capacity, turn to Chapter 7 for a more in-depth description of the PROM's Forth capabilities.

6

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

NVRAM configuration parameters may be viewed and changed using the Toolkit commands listed in Figure 6-1.

Figure 6-1. Configuration Parameter Commands

Command	Description
<code>printenv</code>	Displays all current parameters and current default values (numbers are shown as decimal values)
<code>setenv <i>parameter value</i></code>	Sets the <i>parameter</i> to the given decimal <i>value</i> (Changes are permanent but usually only take effect after a reset.)
<code>set-default <i>parameter</i></code>	Resets the value of the named <i>parameter</i> to the factory default
<code>set-defaults</code>	Resets all parameter values to the factory defaults
<code>show <i>parameter</i></code>	Displays the current value of the named parameter

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 list shown in Figure 6-2. For a reference list of parameters with descriptions, see Appendix B.

Figure 6-2. Typical Configuration Display

Parameter Name	Value	Default Value
sunmon-compatible?	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,-
ttyb-ignore-cd	true	true
ttyb-rts-dtr-off	false	false
ttya-ignore-cd	true	true
ttya-rts-dtr-off	false	false
sbus-probe-list	0123	0123
fcode-debug?	false	false
screen-#columns	80 (decimal value)	80
screen-#rows	34 (decimal value)	34
boot-from-diag	le()vmunix	le()vmunix
boot-from	vmunix	vmunix
auto-boot?	true	true
input-device	keyboard	keyboard
output-device	screen	screen
sd-targets	31204567	31204567
st-targets	45670123	45670123
keyboard-click?	false	false
scsi-initiator-id	7	7
hardware-revision	VersionNumber1	
last-hardware-update	30MAR89	
watchdog-reboot?	false	false
selftest-#megs	1	1
testarea	0	0
mfg-switch?	false	false
diag-switch?	true	true* (see following note)

Note: The default value of `diag-switch?` is `false` in one early version of the boot PROM. To determine the version of the boot PROM, enter:

```
ok .version
ok
```

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.

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 completes.

Resetting Default Values

You can reset one or all 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 all the parameters to the 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 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've changed a parameter.

Security

The security feature of the boot PROM is available on version 1.1 as well as later boot PROM versions. Setting the `security-mode` parameter to `full` or `command` security restricts the set of actions that you are allowed to perform thus making it more difficult for individuals to break into your computer network.

There are three security modes:

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

With no security, any command may 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 want 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 superuser. 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 6 to 8 letters and numbers). The security password can be the same as the `root` password or you can assign a security password which is different from the `root` password.

Caution: The security password is you forget your security password unbootable and you will need to `c:` 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 the security password in this manner, it is a good idea to do something to remove the password from the screen, lest someone see 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 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).

To execute the `n` command from the boot prompt, `>`, a password is required. 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
```

The security password you assign follows the same rules as the `root` password (a combination of 6 to 8 letters and numbers). The security password can be the same as the `root` password or you can assign a security password which is 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 will need to 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 the security password in this manner, it is a good idea to do something to remove the password from the screen, lest someone see 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 full security mode, the PROM will revert to the `>` prompt the next time that the PROM command interpreter is entered.

Changing the Power-On Banner

You can use the `banner` command to view the power-on banner. The configuration parameters that control the power-on system banner are listed in Figure 6-3.

Figure 6-3. Banner Control Parameters

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

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 may 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 banner 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
```

The system displays the banner with your new message.



```
Hello Mom and Dad
```

The graphic logo must be handled a somewhat differently, however. 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 : filllit ( -- ) d# 256 0 do i c, loop ;
ok create logoarray d# 512 allot
ok filllit
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 Ar-
ray display

Hello Mom and Dad

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

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

The configuration parameters related to **Input and Output** input and output are listed in Figure 6-1. **Control** 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 6-4. 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 (keyboard, 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 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:

==

The default settings for both ttya and ttyb are:

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

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 may 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 5.

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 or not the system will automatically boot after the system start-up tests and initialization. In addition, the parameters may be used to select the boot device and the program to be booted. Figure 6-5 shows the parameters that control boot options.

Figure 6-5. Boot Options Parameters

Parameter	Default	Description
auto-boot?	True	Determines whether or not the system will automatically boot after the power-on self-test and system initialization. When true, the Open PROM attempts to boot whatever file is specified by the boot-from parameter.
boot-from	vmunix	Boot source filename (default device is sd0)

The `boot-from` parameter defaults to the filename `vmunix`. The `boot-from` parameter is used either during auto-boot or if 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
ok setenv boot-from le()myunix -s
ok boot
```

Specified booting begins immediately

Controlling Power-On Self-Test

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.

Enabling the diagnostic switch parameter, `diag-switch?`, causes the system to perform a more thorough self-test during power-on. When `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. The power-on testing parameters are listed in Figure 6-6.

Figure 6-6. Power-On Testing Parameters

Parameter	Default	Description
<code>diag-switch?</code>	True	<p>Note: The default value of <code>diag-switch?</code> is false in some early versions of the boot PROM.</p> <p>When <code>diag-switch?</code> 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 <code>ttya</code> or to the default output device) as it begins to execute, and the boot PROM attempts to boot the program specified by the <code>boot-from-diag</code> parameter.</p> <p>When <code>diag-switch?</code> 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.</p>
<code>mfg-switch?</code>	False	When true, the system repeats the power-on self-test and initialization sequence until interrupted with the <code>(L-A)</code> key sequence.
<code>selftest-#megs</code>	1*	Number of megabytes of RAM to test on power-up or on <code>test-memory</code> . This value is ignored if <code>diag-switch?</code> is true.
<code>boot-from-diag</code>	<code>le()vmunix</code>	Diagnostic boot source filename
* Value 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 `[LI-A]` key sequence.

For example, the `diag-switch?` is set to false and you want to power-up in diagnostic mode:

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

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

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

Miscellaneous Parameters

Figure 6-7 shows the remaining configuration parameters that don't readily fall into the categories discussed previously in this chapter. These parameters control various aspects of system function and should generally be used with caution.

Figure 6-7. Miscellaneous Configuration Parameters

Parameter	Default	Description
fcode-debug?	False	If true, includes name fields for SBus cards
keyboard-click?	False	If true, enables the keyboard click sound
ttya-ignore-cd	True	If true, carrier-detect signal is ignored by SunOS on ttya
ttyb-ignore-cd	True	If true, carrier-detect signal is ignored by SunOS on ttyb
ttya-rts-dtr-off	False	If true, SunOS does not assert DTR and RTS on ttya
ttyb-rts-dtr-off	False	If true, SunOS does not assert DTR and RTS on ttyb
watchdog-reboot?	False	Determines whether or not the system will attempt to reboot in the event of a system watchdog timeout
scsi-initiator-id	7	SCSI bus address of host adapter, range 0-7
hardware-revision	no default	System version information
last-hardware-update	no default	System update information
testarea	0	One-byte scratch field, available for read/write test
sbus-probe-list	0123	SBus slot probe order
sunmon-compatible?	True	Indicates whether or not the Sun -Compatible Monitor mode interface is presented. When set to false, the Monitor starts in Toolkit mode with the ok prompt unless security is set. With security set, you will not see the Toolkit prompt even if sunmon-compatible? is set to false.
sd-targets	31204567	Map SCSI disk units, e.g. unit #0 = target #3
st-targets	45670123	Map SCSI tape units, e.g. unit #0 = target #4

This chapter described the configuration parameters contained in NVRAM. Changes made to these parameters are permanent. The Configuration Parameter commands listed in Figure 6-1 have been created to simplify using these parameters. However, configuration parameters should always be adjusted cautiously. When used properly, these configuration parameters allow you flexibility when working with the system hardware.

Appendix B contains a quick reference to the parameters and the Configuration Parameter commands.

More Forth Tools

This chapter provides a brief 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 in-depth 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 generally 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 may 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.

All the available commands are not listed in the reference tables in this chapter. Appendix A is a more thorough command summary.

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 memory access 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
```

Commonly used stack manipulation commands are listed in Figure 7-1.

Figure 7-1. Common Stack Manipulation Commands

Command	Stack Diagram	Description
clear	(??? --)	Empties the data stack
depth	(-- +n)	Returns the number of items that are on the stack
drop	(n --)	Removes the top item from the stack
dup	(n -- n n)	Duplicates the top item on the stack
over	(n1 n2 -- n1 n2 n1)	Copies second stack item to top of stack
pick	(+n -- n2)	Copies +n -th stack item (1 pick = over)
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 rotates 3 stack items
swap	(n1 n2 -- n2 n1)	Exchanges the top 2 stack items

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 may be safely used at any time for debugging purposes. This is the function that `showstack` performs automatically.

The commands listed in Figure 7-2 control numeric input and output.

Figure 7-2. Commands for Changing the Numeric I/O

Command	Stack Diagram	Description
d# <i>item</i>	(-- n)	Interpret the next number in decimal; base is unchanged
decimal	(--)	Set number base to 10
h# <i>item</i>	(-- n)	Interpret the next number in hex; base is unchanged
hex	(--)	Set the number base to 16
.	(n --)	Display n 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 the stack
u.	(u --)	Display an unsigned number
u.r	(u size --)	Display an unsigned number in a fixed width field

Using Arithmetic

Forth provides a variety of basic arithmetic functions. The commands listed in Figure 7-3 perform basic arithmetic operations on items in the data stack.

Figure 7-3. Using Arithmetic

Command	Stack Diagram	Description
*	(n1 n2 -- n3)	Multiply n1 * n2
+	(n1 n2 -- n3)	Add n1 + n2
-	(n1 n2 -- n3)	Subtract n1 - n2
/	(n1 n2 -- quot)	Divide n1 / n2, quotient is truncated
<<	(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
abs	(n -- u)	Absolute value
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
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. See Figures 7-4 and 7-5 for command summaries. You can use the Toolkit to do the following:

- ❑ Reads and writes to any virtual address
- ❑ Maps 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. Generally, 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.

The commands shown in the following two figures can be used to access, modify, map, and test memory locations.

Figure 7-4. Memory Mapping Commands

Command	Stack Diagram	Description
obio	(-- space)	Specify the "devices" address space for mapping
obmem	(-- space)	Specify the "onboard memory" address space for mapping
sbus	(-- space)	Specify the "sbus" address space for mapping
allocate-dma	(size -- virt)	Allocate and map size bytes of memory in DMA space
alloc-mem	(size -- virt)	Allocate and map size bytes of available memory, return the virtual address
map-sbus	(phys size -- virt)	Map a region of SBus space
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 --)	Performs consecutive map-page's to map a region of memory of the given size.
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@

Figure 7-5. Memory Accessing Commands

Command	Stack Diagram	Description
@	(adr -- n)	Fetch a 32-bit number from adr, must be 16-bit aligned
c@	(adr -- byte)	Fetch a byte from adr
w@	(adr -- word)	Fetch a 16-bit number from adr, must be 16-bit aligned
L@	(adr - -n)	Fetch a 32-bit number from adr, must be 32-bit aligned
!	(n adr --)	Store a 32-bit number at adr, must be 16-bit aligned
c!	(n adr --)	Store low byte of n at adr
w!	(word adr --)	Store a 16-bit number at adr, must be 16-bit aligned
L!	(n adr --)	Store a 32-bit number at adr, must be 32-bit aligned
dump	(adr len --)	Display len bytes of memory starting at adr
fill	(adr size byte --)	Set size bytes of memory to byte
wfill	(adr size word --)	Set size bytes of memory to 16-bit word, addr must be 16-bit aligned
Lfill	(adr size long --)	Set size bytes of memory to 32-bit long, addr must be 32-bit aligned
move	(adr1 adr2 u --)	Copy u bytes from adr1 to adr2, handles overlap properly
?	(adr --)	Display the 32-bit number at adr, must be 16-bit aligned
c?	(adr --)	Display the byte at adr
w?	(adr --)	Display the 16-bit number at adr, must be 16-bit aligned
L?	(adr --)	Display the 32-bit longword at adr, must be 32-bit aligned

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.

In order 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 this following example, decimal points are inserted in numbers to help count zeros.

```
ok
ok 1000 alloc-mem           Map in 1000 bytes of physical memory
ok
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 - not shown)
ok 5000 1000 55 fill         Fill in a region of memory 5000-6000 with a fixed pattern
ok
```

*The SBus slot offsets are shown below:

- ☐ SBus slot #0 - 0
- ☐ 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		
ok	<code>80.0000 obmem 700.0000 map-page</code>	<i>Map one page of on- board memory starting at physical address 80.0000 to virtual address 700.0000</i>
ok	<code>80.0000 obio 700.0000 map-page</code>	<i>Map one page of on- board I/O space at address 80.0000 to virtual address 700.0000</i>
ok	<code>80.0000 obmem 700.0000 4.0000 map-pages</code>	<i>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</i>

Using Defining Words

The defining word `variable` assigns a name to a 32-bit region of memory which can then be used 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
```

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 shows assigning a value of 22 to a word named `foo` and then calling `foo` to use its assigned value in an arithmetic operation.

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

A simple colon definition, `: foo 22 ;` also accomplishes a similar result.

The value may be changed with the 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 because you don't have to bother with 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.

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 7-6 shows the defining words that you can use for creating dictionary entries.

Figure 7-6 Common Defining Words

Command	Stack Diagram	Description
<code>: name</code>	<code>(--)</code>	Start the creation of a new colon definition
<code>;</code>	<code>(--)</code>	Finish the creation of a colon definition.
<code>value name</code>	<code>(n --)</code>	Define a value
<code>defer name</code>	<code>(--)</code>	Defining word for forward references or execution vectors
<code>variable name</code>	<code>(--)</code>	Define a variable
<code>is name</code>	<code>(acf --)</code>	Install a new action in a <code>value</code> word or a <code>defer</code> word

Searching the Dictionary

The *dictionary* is the list of all available Forth commands. This section describes some useful dictionary-searching tools.

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 7-7 lists the commands you can use to search the contents of the dictionary.

Figure 7-7. Selected Dictionary Searching Commands

Command	Stack Diagram	Description
<code>' name</code>	<code>(-- acf)</code>	Finds a word in the dictionary. Returns the "code field address".
<code>find</code>	<code>(pstr -- acf n)</code>	Searches for a word in the dictionary. The word to be found is indicated by <code>pstr</code> . <code>n</code> is 0 if not found.
<code>words</code>	<code>(--)</code>	Displays all visible words in the dictionary.
<code>see name</code>	<code>(--)</code>	Decompiles the named word.*
<code>(see)</code>	<code>(acf --)</code>	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 PROMs symbol table, in order to save space.

Controlling Text Input and Output

This section describes the text input and output commands. The commands listed in Figure 7-8 may be used for general purpose text display. These commands control strings or arrays of characters and provide a means to enter comments and control keyboard scanning.

Comments are used with Forth source code (generally in a text file) to describe what the code is attempting to do. The “(” open parenthesis is a command that begins a comment. Anything 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.

`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
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
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
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 and 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 in from the keyboard. If a Forth text file has been loaded into memory (for example, with `dload`), see “Downloading Files” in Chapter 5, then `eval` can be used to compile whatever definitions were contained in the file.

Figure 7-8. General Purpose Text Manipulation Commands

Command	Stack Diagram	Description
<code>ascii ccc</code>	(-- char)	Numerical value of first ASCII character of next word
<code>bl</code>	(-- n)	The ASCII code for the space character; decimal 32
<code>count</code>	(pstr -- adr len)	Convert a packed string to unpacked form
<code>cr</code>	(--)	Terminate a line on the display and goes to the next line
<code>emit</code>	(char --)	Display the character
<code>eval</code>	(adr len --)	Interpret Forth source from an array
<code>exit?</code>	(-- flag)	True if the user wants the output to be terminated. Enables the scrolling prompt: More [<space> , <cr> q] ?
<code>key</code>	(-- char)	Read a character from the keyboard
<code>key?</code>	(-- flag)	True if a key has been typed on the keyboard
<code>p" ccc"</code>	(-- pstr)	Collect a string from the input stream, store as a packed string
<code>type</code>	(adr +n --)	Display +n characters
<code>" ccc"</code>	(-- adr len)	Collect an input stream string; either interpreted or compiled
<code>(ccc)</code>	(--)	Begin and end a comment
<code>\ rest-of-line</code>	(--)	Skip the rest of the line
<code>. " ccc"</code>	(--)	Display a string when definition is executed

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 may then simply be displayed off of the stack (with “.”), or may 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 `false`. A -1 (or any other nonzero number) indicates the flag value `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  
ok
```

3 is not greater than 6

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.

The commands shown in Figure 7-9 perform relational tests and leave a true or false flag result on the stack.

Figure 7-9. 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
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* capability.

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 7-10 control the flow of conditional execution.

Figure 7-10. Conditional Program Execution Commands

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

Using Conditional Loops

Conditional loops execute the same commands over and over 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 of these 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` 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
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.

Figure 7-11 shows the commands you can use to perform conditional loops.

Figure 7-11. Conditional Loop Commands

Command	Stack Diagram	Description
<code>begin</code>	(--)	Begin a <code>begin...while...repeat</code> loop or <code>begin...until</code> loop
<code>repeat</code>	(--)	End a <code>begin...while...repeat</code> loop
<code>until</code>	(flag --)	Continue executing a <code>begin...until</code> loop until flag is true
<code>while</code>	(flag --)	Continue executing a <code>begin...while...repeat</code> 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 7-12 shows the various commands you can use to perform counted loop operations.

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

Figure 7-12. Counted Loops Commands

Command	Stack Diagram	Description
do	(end+1 start --)	Begin a do...loop
?do	(end+1 start --)	Begin ?do...loop to be executed 0 or more times
i	(-- n)	Loop index
j	(-- n)	Loop index for next enclosing loop
leave	(--)	Exit from do...loop
loop	(--)	End of do...loop. Adds 1 to the loop index
+loop	(n --)	End a do...+loop construct; adds n to loop index

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
] do dup i c@ <> ( byte error? )
] 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
ok
ok
ok
ok 500 value testloc
ok : test16 ( -- ) 1.0000 0 ( do 0-ffff )
] 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

```

Scan memory (5000-6000) for bytes not equal to the pattern (55)

Fill a region of memory with a stepped pattern (0-1-2-3-...)

Write different 16-bit values to a location and check. This line is optional

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 statements. 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 that the (optional) default clause may use the test value which is still on the stack, but should *not* remove it (using the "dup ." phrase instead of "."). A successful of clause automatically removes the test value from the stack.

Figure 7-13 lists the conditional case statement commands.

Figure 7-13. 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 out any desired message when the abort takes place. Figure 7-14 contains descriptions of the various program execution control commands.

Figure 7-14. Program Execution Control Commands

Command	Stack Diagram	Description
<code>abort</code>	(--)	Abort current execution and interpret keyboard commands
<code>abort " ccc"</code>	(flag --)	If flag is true, abort and display message
<code>execute</code>	(acf --)	Execute the word whose code field address is on the stack. See <code>find</code> .
<code>exit</code>	(--)	Return from the current word
<code>quit</code>	(--)	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 may be continued or stopped. Disassembly stops automatically when a `call` or `jmp` opcode is encountered.

The `+dis` command may be used 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.

The commands listed in the figure below disassemble memory into equivalent opcodes.

Figure 7-15. Disassembler Command Summary

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

Displaying Registers

There are several ways to enter the Toolkit from the middle of an executing program. These include a program crash, a user abort with `(J1-A)`, or an encountered breakpoint. (Breakpoints are discussed in the next section.) In all of these cases, the Toolkit automatically saves all of the CPU data register values into a buffer area. These values may then be inspected for debugging purposes.

After inspection, program execution may 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 may be altered if desired, by using the `to` command. When execution is resumed, the altered values will be copied back into the CPU and used.

The SPARC register reading and writing commands are listed in Figure 7-16.

If you change `%pc` with the `to` command, you should also change `%npc` as well. It is easier to simply 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, 2 specifies the caller's caller, and so on, up to the number of active stack frames. The default starting value is always 0.

Note: In some earlier versions of the boot PROM, floating-point registers may only be read, but not written. In these versions, the command `save-fregs` must be executed before reading is possible.

Figure 7-16. SPARC Registers Command Summary

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 %yim %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 <i>regname</i>	(value --)	Change the value stored in any of the above registers Use in the form: <i>value to regname</i>
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, %yim, %tbr

Using Breakpoints



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

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

The breakpoint feature lets you pause execution of the test program at any desired point(s). After execution has stopped, registers or memory may be inspected and/or altered, and new breakpoints may be set or cleared. Then, execution may 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 5 for more information. Using `dload` is generally best (as the symbol table for the program is preserved), but `boot -h` will also work if the program is not available over the 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 set a breakpoint(s) and then execute (for example, using the commands `4020 +bp` and `go`), or perform other variations.

The breakpoint commands that control and monitor program execution are listed in the following figure.

Figure 7-18. Breakpoint Command Summary

Command	Stack Diagram	Description
go	(--)	Continue the execution of a halted program.
.bp	(--)	Display all curenly 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. Appendix A contains additional reference information about Forth commands. If you require still more information about Forth, consult some of the previously cited reference books.

A

Toolkit Command Reference

This appendix lists commands supported by the open boot PROM's Forth Toolkit. These commands are grouped into the following four major categories.

❑ Basic System Commands	
System Resetting	Page 103
Diagnostic Tests	Page 103
System Information	Page 104
Disk Drive Control	Page 104
Help and Mode Commands	Page 104
Booting the System	Page 105
❑ Basic Forth Commands	
Manipulating the Stack	Page 107
Accessing Memory	Page 108
Using Arithmetic	Page 109
Changing the Numeric Base	Page 110
Displaying Output	Page 110
Output Display Primitives	Page 111
Line Editor Commands	Page 112

❑ Advanced Forth Programming Commands	
Defining Words	Page 113
Performing Comparisons	Page 114
Inputting Text	Page 115
Displaying Text Output	Page 115
Conditional Loops	Page 116
Counted Loops	Page 116
Controlling Program Execution	Page 117
Conditional Case Statements	Page 117
Manipulating Text Strings	Page 118
Compiling the Dictionary	Page 119
Searching the Dictionary	Page 120
❑ Advanced Programming Commands	
Input/Output Display Modes	Page 120
File Downloading	Page 121
Using the Disassembler	Page 121
Using Breakpoints	Page 122
Reading and Writing SPARC Registers	Page 123
Symbolic Names	Page 124
Traps	Page 124
Mapping Memory	Page 125
Mapping Memory Primitives	Page 126
Accessing Alternate Address Space	Page 127
Manipulating the Cache	Page 128
SunOS Operating System Calls	Page 128
Reading and Writing Machine Registers	Page 129

Basic System Commands

Basic system control commands are listed in the following tables.

System Resetting Commands

The command in the following table performs system resetting.

Command	Stack Diagram	Description
<code>reset</code>	(--)	Reset the entire system (very similar to power-cycle)

Diagnostic Tests

The commands listed in the following table invoke the specified diagnostic test routine.

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

System Information

The following commands provide system information formatted displays.

Command	Stack Diagram	Description
<code>banner</code>	(--)	Displays power-on banner
<code>.enet-addr</code>	(--)	Displays the current Ethernet address
<code>.idprom</code>	(--)	Displays IDPROM contents, formatted
<code>.version</code>	(--)	Displays version and date of boot PROM

Disk Drive Control

The following commands provide floppy and SCSI disk drive control.

Command	Stack Diagram	Description
<code>eject-floppy</code>	(--)	Ejects the diskette from the drive
<code>sync</code>	(--)	Calls the SunOS to write any pending information to the hard disk

Help and Mode Commands

The following commands describe the `help` and mode change functions.

Command	Stack Diagram	Description
<code>help</code>	(--)	List main help categories
<code>help category</code>	(--)	Show help for all commands in the category. Use only the first word of the category description.
<code>help name</code>	(--)	Show help for individual command (where available)
<code>old-mode</code>	(--)	Return to the boot (>) prompt

Booting the System

Boot command syntax are shown in the following figure. Spaces and tabs typed in the command line are ignored. All arguments shown in italics are optional. 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]*

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

Basic Forth Commands

The commands listed in the figures in this section are common Forth command words.

Manipulating the Stack These commands delete, add, and reorder items on the stack.

Command	Stack Diagram	Description
clear	(??? --)	Empties the stack
depth	(-- +n)	Returns the number of items on the stack
drop	(n --)	Removes n from the stack
2drop	(n1 n2 --)	Removes 2 items from the stack
dup	(n -- n n)	Duplicates n
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 n 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
>r	(n --)	Moves a stack item to the return stack
r>	(-- n)	Moves an item from the return stack to the stack
r@	(--n)	Copies the top of the return stack to the stack
roll	(+n --)	Rotates +n stack items
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

Accessing Memory These commands access, modify and test memory locations.

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!	(word 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
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 cnt 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 --)	Set 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
+!	(n adr --)	Adds n to the 32-bit number stored at adr
2!	(n1 n2 adr --)	Stores 2 numbers at adr; n2 at lower address
2@	(adr -- n1 n2)	Fetches 2 numbers from adr; n2 from lower address
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

Using Arithmetic

These commands perform basic arithmetic operations on items in the data stack.

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*n2/3
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
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

Changing the Numeric Base These commands control the interpretation of numeric input.

Command	Stack Diagram	Description
<code>base</code>	(-- adr)	Variable containing number base
<code>d# item</code>	(-- ?)	Interpret the next number in decimal; base is unchanged.
<code>decimal</code>	(--)	Set number base to 10
<code>h# item</code>	(-- ?)	Interpret the next number in hex; base is unchanged.
<code>hex</code>	(--)	Set the number base to 16

Displaying Output These commands convert numbers into text for display.

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

Output Display Primitives

The following primitives are used to create numeric display words, such as `.u.`, or `.r.`

Command	Stack Diagram	Description
<code><#</code>	<code>(--)</code>	Initializes pictured numeric output
<code>#</code>	<code>(tn1 -- tn2)</code>	Converts next digit
<code>#s</code>	<code>(t n1 -- 0)</code>	Converts remaining digits
<code>HOLD</code>	<code>(char --)</code>	Inserts character into pictured output
<code>SIGN</code>	<code>(n --)</code>	Inserts sign into pictured output
<code>#></code>	<code>(n -- adr len)</code>	Ends pictured output, leaving string ready to <code>type</code> .
<code>(.)</code>	<code>(n -- adr len)</code>	Converts a number into a string, ready to <code>type</code> .
<code>(u.)</code>	<code>(u -- adr len)</code>	Converts an unsigned number into a string, ready to <code>type</code> .

Line Editor Commands You can use these line editor commands whenever you are typing commands to the `ok` prompt. When you see two keys in one box, press and hold the first key while pressing the second key, for example to type `(Control-b)`, press and hold the `(Control)` key while pressing the `(b)` key.

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

Advanced Forth Programming Commands

The advanced programming commands can be used to write Forth programs. You may need to refer to a Forth reference book for more information about how to use some of these commands.

Defining Words

These commands are defining words for creating dictionary entries.

Command	Stack Diagram	Description
<code>: <i>name</i></code>	(--)	Start the creation of a new colon definition
<code>;</code>	(--)	Finish the creation of a new colon definition
<code>buffer: <i>name</i></code>	(size --)	Create a named array in temporary storage
<code>value <i>name</i></code>	(n --)	Define a constant (example: 5 value foo)
<code>2constant <i>name</i></code>	(n1 n1 --)	Define a 2-number constant
<code>create <i>name</i></code>	(--)	Generic defining word
<code>defer <i>name</i></code>	(--)	Defining word for forward references or execution vectors
<code>does></code>	(--adr)	Start the run-time clause for defining words
<code>variable <i>name</i></code>	(--)	Define a variable

Performing Comparisons

These commands perform relational tests and leave a true or false flag result.

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
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

Inputting Text

These commands control text input and keyboard scanning.

Command	Stack Diagram	Description
(<i>ccc</i>)	(--)	Begin a comment
\ <i>rest-of-line</i>	(--)	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 <i>ccc</i>	(-- char)	Numerical value of first ascii character of next word
bl	(-- n)	The ASCII code for the space character; decimal 32

Displaying Text Output

These commands control text output display.

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 spaces

Conditional Loops These commands control the execution of conditional loops.

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

Counted Loops These commands control the execution of counted loops.

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

Controlling Program Execution

These commands control the flow of program execution.

Command	Stack Diagram	Description
<code>abort</code>	(--)	Abort current execution and interpret keyboard commands
<code>abort " ccc"</code>	(flag --)	Conditional abort with message
<code>else</code>	(--)	Execute the following code if <code>if</code> failed
<code>execute</code>	(acf --)	Execute the word whose code field address is on the stack
<code>exit</code>	(--)	Return from the current word
<code>if</code>	(flag --)	Execute following code if flag is true
<code>then</code>	(--)	Terminate an <code>if...else...then</code>
<code>quit</code>	(--)	Abort, but leave stack intact

Conditional Case Statements

These commands perform conditional statements.

Command	Stack Diagram	Description
<code>case</code>	(selector -- selector)	Begins a <code>case...endcase</code> conditional
<code>endcase</code>	(selector --)	Terminates a <code>case...endcase</code> conditional
<code>endof</code>	(--)	Terminates an <code>of...endof</code> clause within a <code>case...endcase</code>
<code>of</code>	(selector test-value -- selector { empty})	Begins an <code>of...end</code> within case

Manipulating Text Strings

These commands manipulate strings or arrays of characters.

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

Compiling the Dictionary

These commands compile data into the dictionary.

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

Searching the Dictionary

These commands search the dictionary for information.

Command	Stack Diagram	Description
' <i>name</i>	(-- acf)	Find a word in the dictionary (while executing)
find	(pstr -- acf n)	Search for a word in the dictionary
words	(--)	Display all words
see <i>name</i>	(--)	Decompile or disassemble the named word
(see)	(acf --)	Decompile or disassemble the word indicated by acf

Advanced System Commands

The advanced system commands provide the ability to interact closely with your system's hardware.

Input Output Display Modes

The following commands control temporary assignment of input and output display modes.

Command	Stack Diagram	Description
input	(source --)	Select input source (ttya, ttyb, or keyboard)
output	(source --)	Select output source (ttya, ttyb, or screen)
io	(source --)	Select input and output source

File Downloading

The commands listed below provide various file downloading capabilities.

Command	Stack Diagram	Description
<code>dl</code>	(--)	Download a Forth file over serial line with <code>tip</code> and interpret with: ~C <code>cat filename.fth</code> ^D
<code>dload filename</code>	(addr --)	Load the specified file over Ethernet, at the given address If a binary file, execute with <code>go</code> . If a Forth source file, interpret with: <code>?go</code> (For <code>?go</code> to work, Forth source file must begin with \<space>.)
<code>go</code>	(--)	Begin execution of previously loaded program or continue execution of an interrupted program
<code>?go</code>	(--)	Process recently downloaded file (interpret Forth source, execute binary, or interpret SBus

Using the Disassembler

These commands disassemble memory into equivalent opcodes.

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

Using Breakpoints

Breakpoints may be set to control and monitor program execution.

Command	Stack Diagram	Description
go	(--)	Continue the execution of a halted program
.bp	(--)	Display all currently 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 step, but treats a subroutine call as a single instruction
hops	(n --)	Execute n hops
skip	(--)	Skip (do not execute) the current instruction
till	(addr --)	Execute until the given address is encountered
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

**Reading and Writing
SPARC Registers**

These command provide SPARC register reading and writing capability.

Command	Stack Diagram	Description
%g0 through %g7	(-- value)	Return the value in the given register
%o0 though %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 %yim %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 <i>regname</i>	(value --)	To change the value stored in any of the above registers Use in the form: <i>value to regname</i>
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. 0 is current window, 1 is caller's window, etc.
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, %yim, %tbr

Symbolic Names

These commands can be used for symbolic debugging. For correct execution, the symbol table needs to be loaded before these commands are invoked.

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

Traps

The command listed below can be used to examine SPARC traps.

Command	Stack Diagram	Description
<code>.traps</code>	(--)	Display a list of SPARC trap types

Mapping Memory

The memory mapping commands inspect and alter mapping between virtual and physical memory addresses.

Command	Stack Diagram	Description
<code>allocate-dma</code>	(size -- virt)	Allocate 'size' bytes of memory in DMA space
<code>allocate-virtual</code>	(phys size -- virt)	Assign a virtual address to be used for later mapping
<code>alloc-mem</code>	(size -- virt)	Allocate and map size bytes of memory, return the virtual address
<code>map-sbus</code>	(phys size -- virt)	Map a region of SBUS space (free with <code>free-virtual</code>)
<code>free-dma</code>	(virt size --)	Free memory allocated by <code>allocate-dma</code>
<code>free-virtual</code>	(virt size --)	Free virtual address allocated by <code>allocate-virtual</code>
<code>free-mem</code>	(virt size --)	Free memory allocated by <code>alloc-mem</code> or <code>allocate-virtual</code>
<code>map?</code>	(virt --)	Display memory map information for the virtual address
<code>cprobe</code>	(adr -- flag)	Test for data exception using <code>c@</code>
<code>wprobe</code>	(adr -- flag)	Test for data exception using <code>w@</code>
<code>Lprobe</code>	(adr -- flag)	Test for data exception using <code>L@</code>

Mapping Memory Primitives

Memory mapping primitives are low-level words for controlling page and segment maps.

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

Accessing Alternate Address Space

These commands access alternate address space.

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

Manipulating the Cache

These commands provide cache manipulation capabilities.

Command	Stack Diagram	Description
<code>flush-cache</code>	(--)	Invalidate all cache entries
<code>cache-off</code>	(--)	Disable the cache
<code>cache-on</code>	(--)	Enable the cache
<code>cdata!</code>	(data offset --)	Store the 32-bit data at the cache offset
<code>cdata@</code>	(offset -- data)	Fetch (return) data from the cache offset
<code>ctag!</code>	(value offset --)	Store the tag value at the cache offset
<code>ctag@</code>	(offset -- value)	Return the tag value at the cache offset

SunOS Operating System Calls

This command allows you to call SunOS Operating System functions.

Command	Stack Diagram	Description
<code>wector <i>string</i></code>	(value --)	Call SunOS with the given value and string

Reading and Writing Machine Registers

These commands allow you to read and write the machine registers.

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

B

NVRAM Configuration Parameters Summary

The figures in this appendix provide a quick reference to the NVRAM configuration parameters and commands. Figure B-1 shows the configuration parameter viewing and setting commands.

Figure B-1. Configuration Parameter Command Summary

Command	Description
<code>printenv</code>	Displays all current parameters and current default values (numbers are shown as decimal values)
<code>setenv <i>parameter value</i></code>	Sets the <i>parameter</i> to the given decimal <i>value</i> (Changes are permanent but usually take effect after a reset)
<code>set-default <i>parameter</i></code>	Resets the named <i>parameter</i> value to the factory default
<code>set-defaults</code>	Resets all parameter values to the factory defaults

Figure B-2 shows the command primitives.

Figure B-2. Configuration Parameter Command Primitives

Command	Stack Diagram	Description
<i>parameter</i>	(-- value)	Return the (current) field value
show <i>parameter</i>	(--)	Display the (current) field value, symbolically
show-default <i>parameter</i>	(--)	Display the default field value, symbolically
to <i>parameter</i>	(value --)	Change a (current) field value Examples: false to auto-boot? " <i>Text string</i> " to oem-banner

Appendix B: NVRAM Configuration Parameters Summary

Figure B-3 is a list of the NVRAM configuration parameters.

Figure B-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*
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, #bits, parity, #stop, handshake)	9600, 8, n, 1, -*
ttyb-mode	ttyb (baud, #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 in the boot PROM version 1.1 but false in the boot PROM version 1.0
 ** Values in decimal

Appendix B: NVRAM Configuration Parameters

C

Sun Monitor Command Equivalents

This appendix lists of the most commonly used Sun Monitor commands and the Forth Toolkit commands which perform equivalent functions.

Sun Monitor Command	Forth Toolkit Command
<i>^C source-addr dest-addr len</i>	<i>source-addr dest-addr len move</i>
<i>^I</i>	<i>.version</i>
<i>^T address</i>	<i>address map?</i>
<i>b [boot spec]</i>	<i>boot [boot spec]</i>
	<i>addr dload pathname</i>
<i>c address</i>	<i>go or address set-pc go</i>
<i>d window#</i>	<i>.registers</i>
	<i>.locals</i>
	<i>window# .window</i>
<i>e address action</i>	<i>address w?</i>
	<i>value address w!</i>
<i>f addr 1 addr 2 pattern size</i>	<i>addr #bytes byte cfill</i>
	<i>addr #bytes shortword wfill</i>

Appendix c: Sun Monitor Command Equivalents

Sun Monitor Command	Forth Toolkit Command
<i>g</i> <i>vector argument</i>	go sync <i>value</i> <i>wector argument</i>
<i>h</i>	help help <i>name</i> help <i>category</i>
<i>i</i> <i>cache-data-offset action</i>	<i>address</i> <i>cdata</i> @. <i>value</i> <i>address</i> <i>cdata</i> !
<i>j</i> <i>cache-tag-offset action</i>	<i>address</i> <i>ctag</i> @ . <i>value</i> <i>address</i> <i>ctag</i> !
<i>k1</i> or <i>k2</i>	<i>k1</i> = reset <i>k2</i> = soft-reset
<i>kb</i>	banner
<i>l</i> <i>address action</i>	<i>address</i> <i>l</i> ? <i>value</i> <i>address</i> <i>l</i> !
<i>m</i> <i>address action</i>	<i>address</i> <i>smap</i> ? <i>value</i> <i>address</i> <i>smap</i> !
<i>o</i> <i>address action</i>	<i>address</i> <i>c</i> ? <i>value</i> <i>address</i> <i>c</i> !
<i>p</i> <i>address action</i>	<i>address</i> <i>pgmap</i> @ <i>address</i> <i>pgmap</i> ? <i>pte</i> <i>address</i> <i>pgmap</i> ! <i>physical space#</i> <i>virtual</i> <i>map-page</i> <i>physical space#</i> <i>virtual size</i> <i>map-pages</i> <i>space#</i> : <i>obmem</i> , <i>obio</i> , <i>sbus</i>
<i>q</i> <i>offset action</i>	printenv setenv <i>parametername</i> <i>value</i> set-default <i>parametername</i> set-defaults
<i>s</i> <i>step-count</i>	step <i>step-count</i> <i>steps</i>
<i>r</i> <i>register-number action</i>	<i>registername</i> . <i>value</i> <i>to</i> <i>register-name</i>

Sun Monitor Command	Forth Toolkit Command
<i>s space#</i>	<i>address space# spacec?</i> <i>address space# spacew?</i> <i>address space# spacel?</i> <i>value address space# spacec!</i> <i>value address space# spacew!</i> <i>value address space# spacel!</i>
<i>t program</i>	<i>n steps</i>
<i>u various options</i>	<i>device input</i> <i>device output</i> <i>device io</i> <i>devices: ttya, ttyb, screen, keyboard</i>
<i>v addr1 addr2</i>	<i>address size dump</i>
<i>w address argument</i>	<i>value wector argument</i>
<i>x</i>	<i>test-control-regs</i> <i>test-net</i> <i>test-cache</i> <i>test-memory</i> <i>test-floppy</i> <i>watch-clock</i>
<i>z number address type len</i>	<i>address +bp</i> <i>address -bp</i> <i>return</i> <i>returnL</i> <i>.bp</i> <i>till</i> <i>finish</i>

D

Power-On Self Test

The following figure lists of the Power-on Self-Test (POST) for the SPARCstation 1 with brief descriptions.

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.

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 <code>selftest-#megs</code>

Index

Symbols

! 77
111
#> 111
#s 111
%f0 through %f31 123
%fsr 98
%g0 through %g7 123
%i0 through %i7 123
%L0 through %L7 123
%NPC 97
%npc 98
%o0 through %o7 123
%PC 97, 99
%pc 98
%pc %npc %psr 123
%psr 98
%tbr 98
%y 98
%y %yim %tbr 123
%yim 98
(28, 83
(ccc) 86, 115
(.) 111
(patch 119
(see) 83
(see) 120
(u.) 111
) 28, 83
* 74, 109
*/ 109
*/mod 109
+ 25, 26, 74, 109
+! 108
+bp 99, 100, 122, 138
+di 121
+dis 96
+loop 92, 116
, 119
- 74, 109
-- 26
--bp 100, 122
-bp 100, 122, 138
-rot 107
-trailing 118
. 73, 110, 137
."ccc" 86
.(118
." 118
.bp 100, 122, 138
.breakpoint 100
.breakpoint 122
.d 24, 72, 73, 110

Index

.enet-addr 38, 104
.h 72, 73, 110
.idprom 38, 104
.instruction 100, 122
.locals 98, 123, 136
.psr 98, 123
.r 73, 110
.registers 98, 123, 136
.s 72, 73, 110
.version 38, 53, 104, 136
.window 97, 136
/ 74, 109
/etc/remote 42
/mod 74, 109
;comments in Forth code 83
< 88, 114
<# 111
<< 74, 109
<= 88, 114
<> 88, 114
= 88, 114
> 87, 88, 114
>= 88, 114
> prompt x, 8, 12, 14, 16, 19, 20
 booting from 16
 returning to 19, 104
>> 74, 109
>>a 74, 109
>r 107
? 77, 108
?do 116
?do 92
?dup 107
?go 48, 121
?leave 116
@ 77, 108
\ 86, 115
] 89
] prompt 89
~ 43
~. 44
~C 45
 119
' 83, 120
" ccc" 86

Numerics

0< 88, 114
0<= 88, 114
0<> 88, 114
0= 87, 88, 114
0> 88, 114
0>= 88, 114
1+ 109
1- 109
2! 108
2* 109
2+ 109
2- 109
2/ 109
2@ 108
2drop 107
2dup 107
2over 107
2rot 107
2swap 107
32-bit integer 24
32-bit numbers 31, 70
3dup 107

A

abort 95, 117
aborting a hung system 10, 14
abs 74, 109
absolute value 74, 109
acf 82, 83, 95, 117, 119, 120
addition 74, 109
address
 code field 83, 95, 119
adds 1 109
adds 2 109
aerr! 129
aerr@ 129
again 116
alloc-mem 76, 125
allocate-dma 76, 125
allocate-physical 126
allocate-virtual 76, 125
and 74, 109

Index

arithmetic
 right shift n1 by +n places 109
 using 74, 109
ASCII 86
ascii command 84, 86, 115
asi 127
asynchronous error register 140
asynchronous error virtual address register 140
auto-boot 8
auto-boot? 52, 64, 133
aux! 129
aux@ 129
averr! 129
averr@ 129

B

b command 15
back space
 erase previous character 112
backwards
 one character 112
 one word 112
banner 59, 104, 137
 displaying 38, 60
base 110
 displaying the number in the current base 110
 setting the number base to 10 110
 variable containing number base 110
base 10 24, 110
base 16 110
base 16 numbers 24
baud rate 40
begin 90, 91, 116
beginning of line 112
between 88, 114
bl 86, 115
blank 108
boot
 command 16, 38, 47, 99
 syntax 105
 prom
 displaying the version and date 38, 104
 version 53

boot prompt 2
boot source filename 64
boot-from 52, 64, 133
boot-from-diag 52, 65, 133
booting 38
 after power-on self-test 64
 after start-up tests and initialization 64
 from the > prompt 16
 from the ok prompt 16
bpoff 100, 122
break 43
break key 10, 12, 14, 41
breakpoint commands 99, 122
byte
 copying u bytes from adr 1 to adr 2 77, 108
 copying u bytes from adr 1 to adr 2 starting at high byte 108
 copying u bytes from adr 1 to adr 2 starting at lo byte 108
 displaying len bytes of memory starting at adr 108
 displaying the byte at adr 77, 108
 fetching a byte from adr 77, 108
 low
 storing low byte of n at adr 77
 storing low byte of n at adr 108

C

c command 15, 19
c! 77, 108, 137
c, 119
c? 77, 108, 137
c@ 77, 108
cable
 null modem 42
cache 37, 128
 data and tag fields
 testing 35, 103
 testing 37
cache-off 128
cache-on 128
carriage-return 84
case 94, 95, 117
cat 45

Index

cdata! 128, 137
cdata@ 128
cdata@. 137
cfill 108
clear 71, 107
clock
 function
 testing 35, 37, 103
 system 37
cmove 108
cmove> 108
code field address 83, 95, 119
 address
 code field 117
colon definition 27
 finishing the creation of a colon definition 82
 starting the creation of a new colon definition 82
command line
 erasing characters or words 29
 moving forward and backwards 29
command security 56
command syntax 22
comments in Forth code 83
compile 119
compiling data into the dictionary 119
configuration parameters 67
 NVRAM 131
context register 139
context! 129
context@ 129
control key 29
control registers 35
 testing 35
Control-D 45
constants
 defining word 81
count 85, 86, 118
cprobe 76, 78, 125
CPU data register 97
cr 84, 86, 115
ctag! 128, 137
ctag@ 128, 137
ctrace 98, 123
CTRL A
 beginning of line 30, 112
CTRL B
 backwards one character 30, 112
CTRL D
 erase this character 30, 112
CTRL E
 end of line 30, 112
CTRL F
 forward one character 30, 112
CTRL H
 erase previous character 30, 112
CTRL K
 erase forward 30, 112
CTRL L
 retype line 30, 112
CTRL N
 recall subsequent command line 30, 112
CTRL P
 recall previous command line 30, 112
CTRL Q
 quote next character 30, 112
CTRL U
 erase entire line 30, 112
CTRL W
 erase previous portion of word 30, 112

D
d# 72, 73, 110
data stack
 displaying the contents 110
 emptying 71
dcontext@ 129
decimal 24, 72, 73, 110
decimal number 72
default values 51
defer 82
defining word constants 81, 113
defining word for forward references or
 execution vectors 82
Del
 erase previous character 30, 112
depth 71, 107
diag-switch? 52, 65, 133
diagnostic

Index

- boot from source filename 65, 133
- routines 35, 133
- switch
 - setting 65
- dictionary
 - compiling data into 119
 - displaying all the visible words in the dictionary 83
 - finding a word 83
 - placing a number in the dictionary 119
 - searching for a word in the dictionary 83, 120
- dis 96, 121
- disassembler 96
- diskette
 - drive
 - system 36
 - test 35, 103
 - ejecting 48, 104
- dividing by 2 109
- division 74, 109
- dl 45, 48, 121
- dlbin 48
- dload 46, 47, 48, 85, 99, 121, 136
- DMA
 - allocate and map size of memory in DMA space 76
- dmaaddr 129
- dmaaddr! 129
- dmaaddr@ 129
- do 92, 116
- do loops 92
- downloading 45
- downloading files 28, 45, 85, 99, 121
- drop 71, 107
- dump 22, 77, 78, 108, 138
- dup 71, 107
- duplicate
 - the top item on the stack 71
- duplicating n 107

E

- editor 29
- editor commands 112

- eject-floppy 48, 104
- else 89, 90, 117
- EMACS 29
- emit 84, 86, 115
- enable! 129
- enable@ 129
- end of line 112
- endcase 95, 117
- endof 95, 117
- entering the Forth Toolkit 19
- erase 108
 - entire line 30, 112
 - forward 30, 112
 - previous character 30, 112
 - this character 30, 112
 - this portion of word 30
- ESC B
 - backwards one word 30, 112
- ESC D
 - erase this portion of word 30, 112
- ESC F
 - forward one word 30, 112
- ESC H
 - erase previous portion of word 30, 112
- escape key 29
- Ethernet 18, 106
 - address
 - displaying 38, 104
 - controller
 - testing 35, 36, 103
 - loading the specified file over 121
- eval 85, 118
- exchanging the top 2 stack items 107
- exclusive or 74, 109
- execute 95, 117
- exit 95, 117
- exit? 86, 115

Index

F

- false 87, 88, 114
- Fcode 121
- fcode-debug? 52, 67, 133
- files
 - downloading 28
- fill 77, 108
- find 83, 120
- finish 138
- finish-loop 100, 122
- flag 87
- floppy
 - ejecting 48, 104
- floppy disk drive
 - system 36
 - test 35, 103
- flush-cache 128
- forget 119
- Forth
 - file 121
 - reference materials 30, 31
 - source 118, 121
 - text 45, 46, 47
 - Toolkit
 - entering 19
 - mode 2
- Forth 83-Standard 70
- Forth text 45
- Forth Toolkit 3
- forward
 - one character 112
 - one work 112
- frame buffer 38
- free-dma 125
- free-mem 125
- free-physical 126
- free-virtual 125
- full security 57

G

- go 48, 97, 99, 100, 121, 122, 137

H

- h# 72, 73, 110
- halt 10
- halting the SunOS Operating System 10, 13
- hardware-revision 52, 67, 133
- help 23, 104, 137
- here 119
- hex 24, 72, 73, 110
 - setting the number base to base 16 110
- hex number 24, 72, 110
 - interpreting the next number in hex 110
- history mechanism 29
- HOLD 111
- hop 100, 122
- hops 100, 122
- hung system 9, 11, 14, 21
 - aborting 10, 14

I

- i 92, 116
- ID PROM
 - displaying contents 38, 104
- if 89, 90, 117
- immediate 119
- input 38, 39, 41, 62, 138
 - select source for input 120
 - text 115
- input device
 - keyboard 63
 - ttya 63
 - ttyb 63
- input-device 40, 52, 62, 63, 133
- input/output 40, 41
 - select source for input and output 120
- inserting text 29
- integer 24
 - 32-bit 24
- integer unit 7
 - 119
- interrupt-enable! 129
- interrupt-enable@ 129
- interrupting the power-up sequence 10, 12
- io 138

Index

is 82, 119
IU 7

J

j 92, 116

K

key 86, 115
key? 84, 86, 91, 115
keyboard 38, 39, 115, 120, 138
 input device 63
keyboard click 67
keyboard-click? 52, 67, 133

L

L! 77, 108
l! 137
l? 137
L@ 77, 108
L1-A 8, 10, 12, 14, 39, 40, 42, 43, 66, 97
last-hardware-update 52, 67, 133
leave 92, 116
left shift n1 by +n places 74, 109
Lfill 77, 108
line editor 29
line editor commands
 backward one character 30, 112
 backward one word 30, 112
 beginning of line 112
 end of line 30, 112
 erase entire line 30, 112
 erase forward 30, 112
 erase previous character 30, 112
 erase previous portion of word 30, 112
 erase this character 30, 112
 erase this portion of word 30, 112
 forward one character 30, 112
 forward one word 30, 112
 quote next character 30, 112
 recall previous command line 30, 112
 recall subsequent command line 30, 112
 retype line 30, 112

literal 119
loading the file over Ethernet 121
logical and 74, 109
logical or 74, 109
loop 92, 93, 116
loops 90, 92
 conditional 116
Lprobe 76, 125

M

main memory
 testing 35, 65
manufacturing switch
 setting 65
map-page 76, 126, 137
map-pages 76, 126, 137
map-sbus 76, 125
map-segments 126
map? 76, 125, 136
max 74, 109
maximum 74, 109
memory 96, 99, 121
 accessing 75
 allocate and map size bytes of available
 memory 76
 allocate and map size bytes of memory 76
 displaying len bytes of memory starting at
 adr 77
 main
 testing 103
 map
 displaying memory map information 76
 mapping 78
 setting cnt bytes of memory to byte 108
 setting size bytes of memory to 16-bit word
 108
 setting size bytes of memory to 32-bit long
 108
 setting size bytes of memory to a 16-bit word
 77
 setting size bytes of memory to a 32-bit long
 address 77
 setting size bytes of memory to byte 77, 108
 setting u bytes of memory to 0 108

Index

- setting up u bytes of memory to space 108
- testing 35, 36, 65
- mfg-switch? 52, 65, 133
- min 74, 109
- minimum 74, 109
- mod 74, 109
- monitor 3, 8, 15, 21, 33, 38, 67
 - mode 2
 - starting 9
- monitor mode 2
- move 77, 108, 136
- multiplication 74, 109
- multiplies by 2 109

N

- n command 14, 15, 43
- n1*n1/3 109
- nip 107
- no security 55
- Non-Volatile Random Access Memory 2, 8, 15, 17, 31, 35, 40, 51
- Non-Volatile Read Only Memory 1
- not 74
- npatch 119
- null modem cable 42
- number 24, 73
 - 16-bit
 - displaying the 16-bit number at adr 77, 108
 - fetching a 16-bit number from adr 77, 108
 - storing a 16-bit number at adr 77, 108
 - 32-bit 24, 31, 70
 - adding n to the 32-bit number stored at adr 108
 - displaying the 32-bit number at adr 77, 108
 - fetching a 32-bit number from adr 77, 108
 - storing a 32-bit number at address 108
 - storing a 32-bit number at adr 77
- decimal 72
 - displaying n in decimal without changing the base 73, 110

- interpreting the next number in decimal 73, 110
- setting number base to 10 73
- displaying
 - a number from the current stack 73
 - a number in a fixed width field 73, 110
 - an unsigned number 110
 - the number in the current base 110
- fetching two numbers from adr 108
- hex 72
 - displaying a number in hex without changing the base 73
 - setting the number base to 16 73
- hexidecimal
 - displaying n in hex without changing the base 110
- interpreting the next number in hex with the base unchanged 73
- storing two numbers at adr 108
- unsigned
 - displaying an unsigned number 73, 110
- NVRAM 1, 2, 8, 15, 17, 31, 35, 40, 51
- NVRAM configuration parameters 131

O

- obio 76, 126, 137
- obmem 76, 126, 137
- oem-banner 52, 59, 133
- oem-banner? 52, 60, 61, 133
- oem-logo 52, 59, 133
- oem-logo? 52, 59, 61, 133
- of 95, 117
- ok prompt 3, 16, 19, 21, 29, 40, 112
 - booting from 16
- old-mode 14, 104
- ones compliment 74
- operating system 1, 7, 9, 10, 13, 38, 48
 - halting 10
- or 74, 109
- output 38, 39, 41, 62, 138
 - select source for output 120
 - text 115
- output-device 40, 52, 62, 63, 133
 - screen 63

Index

over 71, 107

P

p" 86, 118

packed string 85, 86

page

- map one page of memory 76

- performs consecutive map-pages 76

page map 139

pagesize 126

parameter

- configuration 67

- displaying 51, 52

- displaying all current parameters 131

- power-on testing 65

- resetting all parameters to the default 51, 54, 131

- resetting the named parameter to the default 51, 54, 131

- setting 51, 53

- setting the parameters to the given decimal value 131

- system configuration 51

parentheses

- left 83

- right 83

password

- security 56

patch 119

pgmap! 126, 137

pgmap? 137

pgmap@ 126, 137

physical address 75

physical memory 125

pick 71, 107

ports 38

POST 7, 139

power cycle 9, 11, 21, 40

Power-On Self-Test 7, 139

power-on testing parameters 65

power-up sequence, interrupting 10, 12

printenv 51, 52, 131, 137

probe-scsi 35, 103

PROM

Programmable Read Only Memory 1

prompt

- > 8, 14, 16, 19, 20

- boot 2

- ok 3, 16, 19, 21, 29, 40

pstr 83, 85, 86, 118, 120

Q

quit 95

R

r> 107

RAM

- testing 65

recall

- previous command line ion 112

- subsequent command ion 112

redirecting input and output 39

registers 99

- test control 35, 103

remainder

- of $n1/n2$ 74, 109

- quotient of $n1*n2/n3$ 109

- quotient of $n1/n2$ 74, 109

repeat 90, 91, 116

reset 34, 103, 137

resetting

- parameter defaults 53

- the system 34

return 100, 122, 138

returning to the > prompt 19

returnL 122, 138

returnl 100

retype line 112

right shift $n1$ by $+n$ places 74, 109

roll 71, 107

rot 71, 107

Index

S

- save-fregs 97
- SBus 67, 121
- sbus 76, 126, 137
- SBus
 - card 67
 - map a region of SBus space 76
 - slot offsets 79
- sbus-probe-list 52, 67, 133
- screen 39, 41, 120, 138
 - output device 63
- screen-#columns 52, 62, 133
- screen-#rows 52, 62, 133
- SCSI 18, 67, 106
- SCSI devices
 - determining 35, 103
- scsi-initiator-id 52, 67, 133
- sd-targets 52, 67, 133
- searching the dictionary 120
- security 55
 - command 56
 - full 57
 - none 55
 - password 56
- security-mode 55, 133
- security-passwd 56, 133
- see 27, 83, 120
- segment map 139
- segmentsize 126
- selftest-#megs 52, 65, 133
- serial 38
 - cable 42
 - line 41
 - port a 8, 39, 41, 120
 - port b 39, 41, 120
 - ports 62
- serr! 129
- serr@ 129
- set-default 51, 54, 131, 137
- set-defaults 51, 54, 131, 137
- set-pc 97, 98, 123
- setenv 51, 53, 131, 137
 - ttya-mode 63
- show 51, 132
- show-default 132
- showing the stack 70
- showstack 25, 70, 110
- SIGN 111
- skip 100, 122
- slot 79
- slot offsets 79
- smap! 126, 137
- smap? 126, 137
- smap@ 126
- soft-reset 137
- space 115
- spacec! 127, 138
- spacec? 127, 138
- spacec@ 127
- spacel! 127, 138
- spacel? 127
- spacel? 138
- spacel@ 127
- spaces 115
- spacew! 127, 138
- spacew? 127, 138
- spacew@ 127
- st-targets 52, 67, 133
- stack 25, 107
 - copying +nth stack items 71, 107
 - copying the second 2 stack items 107
 - copying the second item to the top of the stack 107
 - copying the second stack item to the top of the stack 71
 - copying the top stack item underneath the second item 107
- data
 - displaying the contents 73, 110
 - emptying 71
- diagram 26
- discarding the second stack item 107
- displaying the contents 73
- duplicating 2 stack items 107
- duplicating 3 stack items 107
- duplicating n if it is non-zero 107
- duplicating the top item on the stack 71
- exchanging the top 2 stack items 71, 107
- inversely rotating 3 stack items 71, 107

Index

- moving a stack item to the return stack 107
- moving an item from the return stack to the data stack 107
- removing 2 items from the stack 107
- removing n from the stack 107
- removing the top item from the stack 71
- returning the number of items on the stack 71, 107
- rotating +n stack items 71, 107
- rotating 3 pairs of stack items 107
- rotating 3 stack items 71, 107
- showing 70
- showing stack items before the ok prompt 110
- standalone 99
- starting the monitor 9
- state 119
- step 99, 100, 122, 137
- steps 100, 122, 137, 138
- string
 - packed 85
- subtraction 74, 109
- subtracts 2 109
- Sun-Compatible Monitor 3, 8, 15, 21, 33, 38, 67
 - mode 2
- Sun-Compatible Monitor mode 2
- sunmon-compat? 52, 67, 133
- SunOS Operating System 1, 7, 9, 10, 13, 38, 48, 99, 128
 - halting 10, 13
- sverr! 129
- sverr@ 129
- swap 71, 107
- switch
 - setting 133
- symbol table 46, 96, 124
- symbolic debugging 124
- sync 14, 48, 104, 137
- synchronous error register 140
- synchronous error virtual address register 140
- system clock 37
- system configuration parameters 51
- system enable register 140

T

- terminal 38
- test control registers 103
- test-cache 35, 37, 103, 138
- test-control-regs 35, 103, 138
- test-floppy 35, 36, 103, 138
- test-memory 35, 37, 103, 138
- test-net 35, 36, 103, 138
- testarea 52, 67, 133
- text
 - input 115
 - inserting 29
 - output 115
- tftp protocol 46
- then 89, 90, 117
- till 100, 122, 138
- tip command 41
- tip window 42, 45
- to 61, 98, 132, 137
- to regname 123
- true 87, 88, 114
- ttya 8, 39, 41, 120, 138
 - input device 63
- ttya-ignore-cd 52, 67, 133
- ttya-mode 40, 52, 62, 133
- ttya-rts-dtr-of 67
- ttya-rts-dtr-off 52, 133
- ttyb 39, 41, 120, 138
 - input device 63
- ttyb-ignore-cd 52, 67, 133
- ttyb-mode 40, 52, 62, 133
- ttyb-rts-dtr-off 52, 67, 133
- tuck 107
- type 86, 118

U

- u. 73, 110
- u.r 73, 110
- u< 88, 114
- u<= 88, 114
- u> 88, 114
- u>= 88, 114
- until 90, 91, 116

Index

V

- value 81
 - defining 82
- variable 80
 - defining 82
- version
 - boot PROM 53
- virtual 125
 - address 75

W

- w 97, 98, 123
- w! 108
- w! 77, 136
- w. 119
- w? 77, 108, 136
- w@ 77, 108
- watch-clock 35, 37, 103, 138
- watch-net 35
- watchdog-reboot? 52, 67, 133
- wector 128, 137, 138
- wfill 77, 108, 136
- while 90, 91, 116
- within 88, 114
- words 22, 83, 120
- wprobe 76, 125

X

- xor 74, 109