

# OpenBoot™ 2.x Command Reference

2550 Garcia Avenue  
Mountain View, CA 94043  
U.S.A.

Part No: 802-3241-10  
Revision A, November 1995



© 1995 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX<sup>®</sup> system. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19. The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

#### TRADEMARKS

Sun, Sun Microsystems, the Sun logo, SunSoft, the SunSoft logo, Solaris, SunOS, OpenWindows, DeskSet, ONC, ONC+, and NFS are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. OPEN LOOK is a registered trademark of Novell, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc.

All SPARC trademarks are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. SPARCcenter, SPARCcluster, SPARCcompiler, SPARCdesign, SPARC811, SPARCengine, SPARCprinter, SPARCserver, SPARCstation, SPARCstorage, SPARCworks, microSPARC, microSPARC-II, and UltraSPARC are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK<sup>®</sup> and Sun<sup>™</sup> Graphical User Interfaces were 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 to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

X Window System is a trademark of X Consortium, Inc.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAMS(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



# *Contents*

---

Preface.....	xvii
Ordering Sun Documents.....	xxi
<b>1. Overview .....</b>	<b>1</b>
OpenBoot Features.....	1
The User Interface .....	2
The Restricted Monitor .....	3
The Forth Monitor .....	3
The Default Mode .....	4
The Device Tree .....	4
Device Path Names, Addresses, and Arguments .....	5
Device Aliases .....	7
Displaying the Device Tree.....	8
Getting Help.....	10
A Caution About Using Some OpenBoot Commands .....	11
<b>2. Booting and Testing Your System .....</b>	<b>13</b>

---

Booting Your System . . . . .	13
Running Diagnostics . . . . .	16
Testing the SCSI Bus . . . . .	17
Testing Installed Devices . . . . .	17
Testing the Diskette Drive. . . . .	18
Testing Memory . . . . .	18
Testing the Ethernet Controller . . . . .	19
Testing the Clock . . . . .	20
Monitoring the Network. . . . .	20
Displaying System Information . . . . .	20
Resetting the System . . . . .	21
<b>3. Setting Configuration Parameters . . . . .</b>	<b>23</b>
Displaying and Changing Parameter Settings . . . . .	25
Setting Security Parameters . . . . .	27
Command Security . . . . .	28
Full Security. . . . .	30
Changing the Power-on Banner . . . . .	30
Input and Output Control. . . . .	32
Selecting Input and Output Device Options . . . . .	33
Setting Serial Port Characteristics . . . . .	34
Selecting Boot Options . . . . .	34
Controlling Power-on Self-test . . . . .	35
Using NVRAMRC . . . . .	36
Editing the Contents of NVRAMRC . . . . .	37

---

Activating an NVRAMRC File . . . . .	38
<b>4. Using Forth Tools . . . . .</b>	<b>41</b>
Forth Commands . . . . .	41
Using Numbers . . . . .	43
The Stack . . . . .	44
Displaying Stack Contents . . . . .	44
The Stack Diagram. . . . .	45
Manipulating the Stack . . . . .	47
Creating Custom Definitions . . . . .	49
Using Arithmetic Functions . . . . .	51
Accessing Memory. . . . .	53
Mapping An SBus Device . . . . .	58
Using Defining Words . . . . .	59
Searching the Dictionary. . . . .	61
Compiling Data into the Dictionary . . . . .	63
Displaying Numbers . . . . .	64
Changing the Number Base . . . . .	65
ff 255. . . . .	65
Controlling Text Input and Output . . . . .	66
Redirecting Input and Output . . . . .	69
Command Line Editor. . . . .	71
Conditional Flags. . . . .	73
Control Commands . . . . .	75
The if...else...then Structure. . . . .	75

---

The case Statement .....	76
The begin Loop .....	77
The do Loop .....	79
Additional Control Commands .....	81
<b>5. Loading and Executing Programs.....</b>	<b>83</b>
Using dload to Load from Ethernet.....	84
Forth Programs .....	84
FCode Programs.....	84
Binary Executables.....	85
Using boot to Load from Hard Disk, Floppy Disk, or Ethernet	85
Forth Programs .....	86
FCode Programs.....	86
Binary Executables.....	87
Using dl to Load Forth Over a Serial Port .....	87
Using dlbin to Load FCode or Binary Over a Serial Port .....	88
<b>6. Debugging .....</b>	<b>89</b>
Using the Disassembler.....	89
Displaying Registers .....	90
Breakpoints.....	91
The Forth Source-level Debugger .....	93
Using ftrace.....	94
<b>A. Testing with a Terminal Emulator .....</b>	<b>97</b>
Common Problems with tip .....	99
<b>B. Building A Bootable Floppy Disk .....</b>	<b>101</b>

---

Procedure for the Pre-Solaris 2.0 Operating Environment . . . .	101
Procedure for the Solaris 2.0 or 2.1 Operating Environment . .	102
<b>C. Unsupported Commands</b> . . . . .	<b>105</b>
<b>D. Troubleshooting Guide</b> . . . . .	<b>109</b>
Power-on Initialization Sequence . . . . .	109
Emergency Procedures . . . . .	111
Preserving Data After a System Crash . . . . .	111
Common Failures . . . . .	111
Blank Screen - No Output . . . . .	112
System Boots From the Wrong Device . . . . .	112
System Will Not Boot From Ethernet . . . . .	114
System Will Not Boot From Disk . . . . .	114
SCSI Problems . . . . .	115
Setting the Console to a Specific Monitor . . . . .	115
<b>E. Forth Word Reference</b> . . . . .	<b>117</b>
<b>Index</b> . . . . .	<b>151</b>





## *Tables*

---

Table 1	Restricted Monitor Commands .....	3
Table 2	Device Path Name Parameters .....	5
Table 3	Examining and Creating Device Aliases .....	7
Table 4	Commands for Browsing the Device Tree.....	8
Table 5	Help Commands .....	10
Table 6	Common Options for the <code>boot</code> Command .....	14
Table 7	Typical Device Aliases .....	15
Table 8	Alias Names in the Solaris Operating Environment.....	16
Table 9	Diagnostic Test Commands .....	16
Table 10	System Information Display Commands .....	21
Table 11	NVRAM Configuration Parameters .....	23
Table 12	Viewing/Changing Configuration Parameters .....	25
Table 13	I/O Device Parameters .....	33
Table 14	NVRAMRC Editor Commands .....	37
Table 15	<code>nvedit</code> Keystroke Commands .....	38
Table 16	Stack Item Notation .....	46

---

Table 17	Stack Manipulation Commands . . . . .	48
Table 18	Color Definition Words . . . . .	49
Table 19	Arithmetic Functions. . . . .	51
Table 20	Memory Access Commands . . . . .	53
Table 21	Memory Mapping Commands . . . . .	56
Table 22	Defining Words . . . . .	59
Table 23	Dictionary Searching Commands . . . . .	61
Table 24	Dictionary Compilation Commands . . . . .	63
Table 25	Basic Number Display . . . . .	64
Table 26	Changing the Number Base . . . . .	65
Table 27	Controlling Text Input . . . . .	66
Table 28	Displaying Text Output . . . . .	67
Table 29	Manipulating Text Strings . . . . .	68
Table 30	I/O Redirection Commands . . . . .	69
Table 31	Required Command Line Editor Keystroke Commands . . .	71
Table 32	Optional Command Line Editor Keystroke Commands . . .	71
Table 33	Optional Command Line History Keystroke Commands . . .	72
Table 34	Optional Command Completion Keystroke Commands . . .	73
Table 35	Comparison Commands. . . . .	74
Table 36	<code>if..else..then</code> Commands. . . . .	75
Table 37	<code>case</code> Statement Commands . . . . .	76
Table 38	<code>begin</code> (Conditional) Loop Commands . . . . .	77
Table 39	<code>do</code> (Counted) Loop Commands . . . . .	79
Table 40	Program Execution Control Commands . . . . .	81
Table 41	File Loading Commands . . . . .	83

---

Table 42	Disassembler Commands . . . . .	89
Table 43	SPARC Register Commands . . . . .	90
Table 44	Breakpoint Commands . . . . .	91
Table 45	Forth Source-level Debugger Commands . . . . .	93
Table C-1	Workarounds for Unsupported Commands . . . . .	105
Table D-1	Emergency Keyboard Commands . . . . .	111
Table E-1	Stack Item Notation . . . . .	117
Table E-2	Restricted Monitor Commands . . . . .	118
Table E-3	Examining and Creating Device Aliases . . . . .	118
Table E-4	Commands for Browsing the Device Tree . . . . .	119
Table E-5	Help Commands . . . . .	119
Table E-6	Common Options for the <code>boot</code> Command . . . . .	120
Table E-7	Diagnostic Test Commands . . . . .	120
Table E-8	System Information Display Commands . . . . .	121
Table E-9	NVRAM Configuration Parameters . . . . .	121
Table E-10	Viewing/Changing Configuration Parameters . . . . .	123
Table E-11	Configuration Parameter Command Primitives . . . . .	123
Table E-12	NVRAMRC Editor Commands . . . . .	123
Table E-13	<code>nvedit</code> Keystroke Commands . . . . .	124
Table E-14	Stack Manipulation Commands . . . . .	124
Table E-15	Colon Definition Words . . . . .	125
Table E-16	Arithmetic Functions . . . . .	125
Table E-17	Conversion Operators . . . . .	127
Table E-18	Memory Access Commands . . . . .	128
Table E-19	Memory Mapping Commands . . . . .	129

---

Table E-20	Memory Mapping Primitives . . . . .	130
Table E-21	Cache Manipulation Commands . . . . .	131
Table E-22	Reading/Writing Machine Registers in Sun-4D Machines . . .	132
Table E-23	Reading/Writing Machine Registers in Sun-4M Machines . .	133
Table E-24	Reading/Writing Machine Registers in Sun-4C Machines . . .	134
Table E-25	Alternate Address Space Access Commands . . . . .	134
Table E-26	Defining Words . . . . .	135
Table E-27	Dictionary Searching Commands . . . . .	136
Table E-28	Dictionary Compilation Commands . . . . .	136
Table E-29	Assembly Language Programming . . . . .	137
Table E-30	Basic Number Display . . . . .	138
Table E-31	Changing the Number Base . . . . .	138
Table E-32	Numeric Output Word Primitives . . . . .	139
Table E-33	Controlling Text Input . . . . .	139
Table E-34	Displaying Text Output . . . . .	140
Table E-35	Formatted Output . . . . .	140
Table E-36	Manipulating Text Strings . . . . .	140
Table E-37	I/O Redirection Commands . . . . .	141
Table E-38	ASCII Constants . . . . .	141
Table E-39	Line Editor Commands . . . . .	141
Table E-40	Comparison Commands . . . . .	142
Table E-41	<code>if-then-else</code> Commands . . . . .	143
Table E-42	<code>case</code> Statement Commands . . . . .	143
Table E-43	<code>begin</code> (Conditional) Loop Commands . . . . .	144
Table E-44	<code>do</code> (Counted) Loop Commands . . . . .	144

---

Table E-45	Program Execution Control Commands . . . . .	144
Table E-46	File Loading Commands . . . . .	145
Table E-47	Disassembler Commands . . . . .	145
Table E-48	SPARC Register Commands . . . . .	146
Table E-49	Breakpoint Commands . . . . .	146
Table E-50	Forth Source-level Debugger Commands . . . . .	147
Table E-51	Time Utilities . . . . .	148
Table E-52	Miscellaneous Operations . . . . .	148
Table E-53	Multiprocessor Commands . . . . .	149
Table E-54	Emergency Keyboard Commands . . . . .	149









## *Preface*

---

The *OpenBoot 2.x Command Reference* manual describes the OpenBoot 2.x firmware that is part of the boot PROM in Sun™ systems.

### *Audience*

The features of the OpenBoot firmware allow it to be used by end users as well as by system administrators and developers. This manual is for all such users who want to use the OpenBoot 2.x firmware to configure and debug their systems.

### *Contents*

In this manual, you will find information about using the OpenBoot firmware to perform tasks such as:

- Booting the operating system
- Running diagnostics
- Modifying system start-up configuration parameters
- Loading and executing programs
- Troubleshooting

If you want to write Forth programs or use the more advanced features of this firmware (such as its debugging capabilities), this manual also describes the commands of the OpenBoot Forth Interpreter.

---

## Assumptions

This manual assumes that you are working on a SPARC® system with a version 2.x OpenBoot PROM. Some of the tools and capabilities described in this manual do not exist on the pre-2.x PROM SPARC systems. If you are using a SPARCstation™ 1, SPARCstation IPC™, or other system with a pre-2.x version PROM, refer to an earlier version of this manual: *Open Boot PROM Toolkit User's Guide*, part number 800-5279-10. Also see Appendix C in this manual for a list of unsupported commands.

## Organization

The *OpenBoot 2.x Command Reference* is organized as follows:

Chapter 1, “Overview”, describes the user interface and other main features of the firmware.

Chapter 2, “Booting and Testing Your System”, explains the most common tasks for which the OpenBoot firmware is used.

Chapter 3, “Setting Configuration Parameters”, details how to perform system administration tasks with NVRAM parameters.

Chapter 4, “Using Forth Tools”, describes both basic and advanced functions of the OpenBoot Forth language.

Chapter 5, “Loading and Executing Programs”, describes how to load and execute programs from various sources (such as Ethernet, disk, or a serial port).

Chapter 6, “Debugging”, describes the firmware’s debugging capabilities, including the Disassembler, the Forth Source-level Debugger, and breakpoints.

Appendix A, “Testing with a Terminal Emulator“, describes how to connect your system to another Sun™ system using serial ports.

Appendix B, “Building A Bootable Floppy Disk”, tells you how to create a bootable floppy diskette from which you can load programs or files.

Appendix C, “Unsupported Commands”, lists commands that may not be available in earlier OpenBoot systems and possible workarounds for them.

---

Appendix D, “Troubleshooting Guide”, discusses solutions for typical situations where you cannot boot the operating system.

Appendix E, “Forth Word Reference”, contains all currently-supported OpenBoot Forth commands.

## *Related Documentation*

- Companion documents to this manual:
  - *OpenBoot 2.x Quick Reference Card*, part number 802-1958-01.  
This fold-out card is a summary of often-used OpenBoot Forth commands.
- For information about FCode, the version of Forth implemented in the OpenBoot 2.x firmware for using SBus cards, refer to the Sun manual:
  - *Writing FCode 2.x Programs*, part number 802-1941-01.
- For more information on the Forth language, read:
  - *Starting Forth*
  - Leo Brodie/Forth, Inc.  
Prentice-Hall Software Series  
Englewood Cliffs, New Jersey 07632

The second edition of *Starting Forth* describes the current Forth standard dialect, Forth 83.

---

**Note** – There are several differences between the versions of Forth described in the above document and the version described in this manual. Specifically, the boot PROM Forth Monitor uses 32-bit numbers instead of 16-bit numbers. Also, the text editor described in the referenced book is not the same as the Forth Monitor editor.

---

---

## What Typographic Changes and Symbols Mean

The following table describes meanings of the type changes and symbols used in this book.

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. system% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	system% <b>su</b> Password:
AaBbCc123	Variable: replace with a real name or value	To delete a file, type <code>rm filename</code> .
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.
Code samples are included in boxes and may display the following:		
ok	The OpenBoot Forth Monitor prompt	ok
>	The OpenBoot Restricted Monitor prompt	>
%	UNIX C shell prompt	system%
\$	UNIX Bourne and Korn shell prompt	system\$
#	Superuser prompt, all shells	system#

- Keys are indicated by their name. For example:  
Press the Return key.
- When you see two keys named, press and hold the first key down and then press the second key. For example:  
For 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.

---

## Ordering Sun Documents

The SunDocs Order Desk is a distribution center for Sun Microsystems technical documentation. You can use major credit cards and company purchase orders. You can order documentation in the following ways:

---

**In the U.S.A.**

Call 1-800-247-0250

Fax 1-801-373-6798

---

**Outside the U.S.A.**

Call 1-801-342-3450

Fax 1-801-373-6798

---

**World Wide Web:** <http://www.sun.com/sundocs/catalog.html>

---

## Sun Welcomes Your Comments

Please use the *Reader Comment Card* that accompanies this document. We are interested in improving our documentation and welcome your comments and suggestions.

You can also email or fax your comments to us. Please include the part number of your document in the subject line of your email or fax message.

- **Email:** [smcc-docs@sun.com](mailto:smcc-docs@sun.com)
- **Fax:** SMCC Document Feedback  
1-415-786-6443



# Overview

---



This chapter introduces the OpenBoot firmware, the standard firmware for Sun systems.

The OpenBoot Version 1 firmware was introduced on the Sun SPARCstation 1. It also was the firmware for the SPARCstation 1+, SPARCstation IPC, and SPARCstation SLC™ systems. This manual describes Version 2 of the firmware, which first appeared on the SPARCstation 2 system.

The OpenBoot firmware is stored in the boot PROM (programmable read-only memory) of a system so that it is executed immediately after you turn on your system. The primary task of the OpenBoot firmware is to boot the operating system from either a mass storage device or from a network. The firmware also provides extensive features for testing hardware and software interactively.

## OpenBoot Features

The OpenBoot architecture provides a significant increase in functionality over the boot PROMs in earlier Sun systems. Although this architecture was first implemented on SPARC systems, its design is processor-independent. Some notable features of the OpenBoot firmware include:

- *Plug-in device drivers.* A plug-in device driver is usually loaded from a plug-in device such as an SBus card. The plug-in device driver can be used to boot the operating system from that device or to display text on the

device before the operating system has activated its own drivers. This feature allows the input and output devices supported by a particular system to evolve without changing the system PROM.

- *FCode interpreter.* Plug-in drivers are written in a machine-independent interpreted language called *FCode*. Each OpenBoot system PROM contains an FCode interpreter. Thus, the same device and driver can be used on machines with different CPU instruction sets.
- *Device tree.* The device tree is an OpenBoot data structure describing the devices (permanently installed and plug-in) attached to a system. Both the user and the operating system can determine the hardware configuration of the system by inspecting the device tree.
- *Programmable user interface.* The OpenBoot user interface is based on the interactive programming language *Forth*. Sequences of user commands can be combined to form complete programs. This provides a powerful capability for debugging hardware and software.

## The User Interface

You can enter the OpenBoot environment in the following ways:

- By halting the operating system.
- By using the `stop-A` key sequence from the keyboard. (This abruptly breaks execution of the operating system and should be used with caution.)
- By power-cycling the system. (If your system is configured to boot automatically, you can enter the OpenBoot environment by pressing `stop-A` after the display console banner appears but before the system starts booting the operating system. If automatic booting is not enabled, the system will enter the OpenBoot environment on its own instead of booting the operating system.)
- When the system hardware detects an error from which it cannot recover. (This is known as a Watchdog Reset.)

The OpenBoot firmware provides three external interfaces: an interface for the operating system or other standalone programs, an interface for expansion bus plug-in boards (for example, SBUS), and a command line interface for the user at the system console. This manual describes the third of these interfaces: the system console command line interface.



The command line interface has two modes:

- The Restricted Monitor
- The Forth Monitor

## *The Restricted Monitor*

The Restricted Monitor provides a simple set of commands to initiate booting of the system, resume system execution, or enter the Forth Monitor. The Restricted Monitor is also used to implement system security. (See Chapter 3, “Setting Configuration Parameters”, for information on system security.)

The Restricted Monitor prompt is `>`. When you enter the Restricted Monitor, the following screen is displayed:

---

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

---

The Restricted Monitor commands are summarized in Table 1.

*Table 1* Restricted Monitor Commands

<b>Command</b>	<b>Description</b>
<code>b</code> [ <i>specifiers</i> ]	Boot the operating system.
<code>c</code>	Resume the execution of a halted program.
<code>n</code>	Enter the Forth Monitor.

## *The Forth Monitor*

The Restricted Monitor functions `b` (for booting the system) and `c` (for resuming execution of a halted program) are available as the `boot` (see Chapter 2, “Bootting and Testing Your System”) and `go` (see Chapter 5, “Loading and Executing Programs”) commands, respectively, in the Forth Monitor.

The Forth Monitor is an interactive command interpreter that gives you access to an extensive set of functions for hardware and software development, fault isolation, and debugging. A variety of system users, from end-users to system administrators to system developers, can use these functions.

The Forth Monitor prompt is `ok`. When you enter the Forth Monitor, the following screen is displayed:

---

```
Type help for more information
ok
```

---

### *The Default Mode*

The default mode in early OpenBoot systems is the Restricted Monitor. This was done mainly to provide a default look and feel similar to pre-OpenBoot systems.

The SPARCserver™ 690 system was the first to have the Forth Monitor as the default mode. All systems introduced thereafter also default to this mode. For such systems, the Restricted Monitor’s only real function is to support system security. (Chapter 3, “Setting Configuration Parameters”, discusses system security.)

If you want to leave the Forth Monitor and get into the Restricted Monitor, type:

---

```
ok old-mode
```

---

### *The Device Tree*

Devices are attached to a SPARC-based system on a set of interconnected buses. The OpenBoot firmware represents the interconnected buses and their attached devices as a tree of nodes. Such a tree is called the device tree. A node representing the whole machine forms the tree’s root node.

Each device node can have:

- *Properties*, which are the data structures describing the node and its associated device
- *Methods*, which are the software procedures used to access the device
- *Children*, which are other device nodes “attached” to that node, that lie directly below it in the device tree
- A *parent*, which is the node that lies directly above it in the device tree.

Nodes with children usually represent buses and their associated controllers, if any. Each such node defines a physical address space that distinguishes the devices connected to the node from one another. Each child of that node is assigned a physical address within the parent's address space.

The physical address generally represents a physical characteristic unique to the device (such as the bus address or the slot number where the device is installed). This prevents device addresses from changing when another device is installed in the system.

### *Device Path Names, Addresses, and Arguments*

The firmware deals directly with hardware devices in the system. Each device has a unique name representing the type of device and where that device is located within the system addressing structure. The following example shows a full device path name:

```
/sbus@1,f8000000/esp@0,40000/sd@3,0:a
```

A full device path name is a series of node names separated by slashes (/). The root of the tree is the machine node, which is not named explicitly but is indicated by a leading slash (/). Each node name has the form:

*name@address:arguments*

Table 2 describes each of these parameters.

Table 2 Device Path Name Parameters

Path Name Parameter	Description
name	A text string that, ideally, has some mnemonic value. (For example, <i>sd</i> represents "SCSI disk".) Many names, especially names of plug-in modules, include the name or stock symbol of the device's manufacturer (for example, <i>SUNW</i> , <i>esp</i> ).
@	Must precede the <i>address</i> parameter.
address	A text string representing an address, usually of the form <i>hex_number</i> , <i>hex_number</i> . (Numbers are given in hexadecimal format.)
:	Must precede the <i>arguments</i> parameter.
arguments	A text string, whose format depends on the particular device. It can be used to pass additional information to the device's software.

The full device path name mimics the hardware addressing used by the system to distinguish between different devices. Thus, you can specify a particular device without ambiguity.

In general, the *address* part of a node name represents an address in the address space of its parent. The exact meaning of a particular address depends on the bus to which the device is attached. Consider the same example:

```
/sbus@1,f8000000/esp@0,40000/sd@3,0:a
```

- 1 , f8000000 represents an address on the main system bus, because the SBus interface is directly attached to the main system bus.
- 0 , 40000 is an SBus slot number and an offset within that slot, because the `esp` device is in SBus slot 0 at offset 40000. (In this example, the device is a SCSI host adapter, although the name does not say so directly.)
- 3 , 0 is a SCSI target and logical unit number, because the `sd` device is attached to a SCSI bus at target 3, logical unit 0.

When specifying a path name, either the *@address* or *name* part of a node name is optional, in which case the firmware tries to pick the device that best matches the given name. If more than one equally-good selection exists, the firmware makes a selection (but it may not be the one you want).

For example, using `/sbus/esp@0,40000/sd@3,0` assumes that the system in question has exactly one SBus interface on the main system bus, making `sbus` as unambiguous an address as `sbus@1,f8000000`. On the same system, however, `/sbus/esp/sd@3,0` might or might not be ambiguous. Since SBus accepts plug-in cards, there could be more than one `esp` device on the same SBus. If there were more than one on the system, using `esp` alone would not specify which one, and the firmware might not select the one you intended.

As another example, `/sbus/@0,40000/sd@3,0` would normally be acceptable while `/sbus/esp@0,40000/@3,0` usually would not, since both a SCSI disk device driver (`sd`) and a SCSI tape device driver (`st`) can use the SCSI target,logical unit address 3,0.

The *:arguments* part of the node name is also optional. Once again, in the example:

```
/sbus@1,f8000000/esp@0,40000/sd@3,0:a
```

the argument for the `sd` device is the string `a`. The software driver for `sd` interprets its argument as a disk partition, so the device path name refers to partition `a` on that disk.

## Device Aliases

There are two kinds of device names:

- Full device path names (discussed in the previous section), such as `/sbus@1,f8000000/esp@0,40000/sd@3,0:a`
- Device aliases, such as `disk`

A device alias, or simply, alias, is a way of representing a device path name. An *alias represents an entire device path name, not a component of it*. For example, the alias `disk` may represent the device path name:

```
/sbus@1,f8000000/esp@0,40000/sd@3,0:a
```

Systems have predefined device aliases for most commonly-used devices, so you rarely need to type a full device path name.

Table 3 describes the `devalias` command, which is used to examine, create, and change aliases.

Table 3 Examining and Creating Device Aliases

Command	Description
<code>devalias</code>	Display all current device aliases.
<code>devalias alias</code>	Display the device path name corresponding to <i>alias</i> .
<code>devalias alias device-path</code>	Define an alias representing <i>device path</i> . If an alias with the same name already exists, the new value supersedes the old.

User-defined aliases are lost after a system reset or power cycle. If you want to create permanent aliases, you can either manually store the output of the `devalias` command in a portion of non-volatile RAM (NVRAM) called NVRAMRC, or use the `nvalias` and `nvunalias` commands. (See Chapter 3, “Setting Configuration Parameters”, for more details.)

## *Displaying the Device Tree*

You can browse the device tree to examine and modify individual device tree nodes. The device tree browsing commands are similar to the UNIX® commands for changing the working directory within the UNIX directory tree. Selecting a device node makes it the current node.

Examine the device tree with the commands shown in Table 4.

*Table 4* Commands for Browsing the Device Tree

<b>Command</b>	<b>Description</b>
<code>.attributes</code>	Display the names and values of the current node's properties.
<code>cd <i>device-path</i></code>	Select the indicated device node, making it the current node.
<code>cd <i>node-name</i></code>	Search for a node with the given name in the subtree below the current node, and select the first such node found.
<code>cd ..</code>	Select the device node that is the parent of the current node.
<code>cd /</code>	Select the root machine node.
<code>device-end</code>	De-select the current device node, leaving no node selected.
<code>ls</code>	Display the names of the current node's children.
<code>pwd</code>	Display the device path name that names the current node.
<code>show-devs [<i>device-path</i>]</code>	Display all the devices known to the system directly beneath a given level in the device hierarchy. <code>show-devs</code> used by itself shows the entire device tree.
<code>words</code>	Display the names of the current node's methods.

If you have been browsing the device tree, and want to reset the system, type:

---

```
ok device-end
ok reset
```

---

---

The following example shows the use of `.attributes`:

---

```
ok cd /zs@1,f0000000
ok .attributes
address                ffee9000
port-b-ignore-cd
port-a-ignore-cd
keyboard
device_type            serial
slave                  00000001
intr                   0000000c 00000000
interrupts              0000000c
reg                    00000001 f0000000 00000008
name                   zs
ok
```

---

`show-devs` lists all the devices in the OpenBoot device tree, as shown in the following example:

---

```
ok show-devs
/fd@1,f7200000
/virtual-memory@0,0
/memory@0,0
/sbus@1,f8000000
/auxiliary-io@1,f7400003
/interrupt-enable@1,f5000000
/memory-error@1,f4000000
/counter-timer@1,f3000000
/eprom@1,f2000000
/audio@1,f7201000
/zs@1,f0000000
/zs@1,f1000000
/openprom
/aliases
/options
/packages
/sbus@1,f8000000/cgsix@3,0
/sbus@1,f8000000/le@0,c00000
/sbus@1,f8000000/esp@0,800000
ok
```

---

The following is an example of the use of words:

```
ok cd /zs
ok words
selftest      ring-bell      read           remove-abort?
install-abort close          open           abort?        restore
clear         reset         initkbdmouse  keyboard-addr mouse
1200baud     setbaud      initport      port-addr
ok
```

## 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 shown in Table 5.

Table 5 Help Commands

Command	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 command</code>	Show help for individual command (where available).

`help`, without any specifier, displays instructions about using the help system and lists the available help categories. Because of the large number of commands, help is available only for commands that are used frequently.

If you want to see the help messages for all the commands in a selected category, or, possibly, a list of sub-categories, type:

```
ok help category
```

If you want help for a specific command, type:

```
ok help command
```



---

For example, when you ask for information on the `dump` command, you see the following message:

---

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

---

The above help message first shows that `dump` is a command from the Memory access category. The message also shows the format of the command.

---

**Note** – In some newer systems, descriptions of additional machine-specific commands are available with the `help` command.

---

## *A Caution About Using Some OpenBoot Commands*

If you boot the operating system, exit it with either the `Stop-A` or `halt` commands, and then use some OpenBoot commands, the commands might not work as expected.

For example, suppose you boot the operating system, *exit* it with `Stop-A`, then execute the `probe-scsi` command (described in Chapter 2, “Booting and Testing Your System”). You may find that `probe-scsi` fails, and you may not be able to resume the operating system. When this happens, type the following commands:

---

```
ok sync
ok boot
```

---

To re-execute an OpenBoot command which fails because the operating system has halted, reset the system, then invoke the command, as shown:

---

```
ok reset
ok probe-scsi
ok
```

---



## *Booting and Testing Your System*

---



This chapter describes the most common tasks that you perform using the OpenBoot firmware. These tasks let you:

- Boot your system.
- Run diagnostics.
- Display system information.
- Reset the system.

### *Booting Your System*

The most important function of the OpenBoot firmware is to boot the system. Booting is the process of loading and executing a standalone program such as the operating system. Once it is powered on, the system usually boots automatically, without user intervention. If necessary, you can explicitly initiate the boot process from the OpenBoot command interpreter. Automatic booting uses the default boot device specified in non-volatile RAM (NVRAM); user-initiated booting uses either the default boot device or one specified by the user.

If you want to boot the system from the default boot device, type the following command at the Forth Monitor prompt:

```
ok boot
```

If you are at the Restricted Monitor prompt, and you want to boot your system, type:

```
> b
```

The `boot` command has the following format:

```
boot [device-specifier] [filename] [options]
```

The optional parameters for the `boot` command are described in Table 6.

Table 6 Common Options for the `boot` Command

Parameter	Description
[ <i>device-specifier</i> ]	The name (full path name or alias) of the boot device. Typical values include: cdrom (CD-ROM drive) disk (hard disk) floppy (3-1/2" diskette drive) net (Ethernet) tape (SCSI tape)
[ <i>filename</i> ]	The name of the program to be booted (for example, <code>stand/diag</code> ). <i>filename</i> is relative to the root of the selected device and partition (if specified). If <i>filename</i> is not specified, the boot program uses the value of the <code>boot-file</code> NVRAM parameter (see Chapter 3).
[ <i>options</i> ]	-a - Prompt interactively for the device and name of the boot file. -h - Halt after loading the program. (These options are specific to the operating system, and may differ from system to system.)

**Note** – Many commands (such as `boot` and `test`) that require a device name, accept either a full device path name or a device alias. In this manual, the term *device-specifier* is used to indicate that either a device path name or a device alias is acceptable for such commands.

To explicitly boot from the internal disk (for diskfull systems), type:

```
ok boot disk
```

To explicitly boot from Ethernet, type:

```
ok boot net
```

To specify a boot device at the Restricted Monitor prompt, use the `b` command with the name of the boot device as shown in the examples below.

```
> b disk (to explicitly boot from the internal disk for diskfull systems)
> b net (to explicitly boot from Ethernet)
```

Device alias definitions vary from system to system. Use the `devalias` command, described in Chapter 1, “Overview”, for definitions of your system’s aliases. Table 7 is an example of device aliases and their definitions based on SPARCstation 2 and SPARCstation IPX systems. The heading “Old Path” refers to the OpenBoot Version 1.x usage for the equivalent SBus device.

Table 7 Typical Device Aliases

Alias	Boot Path	Old Path	Description
disk	/sbus/esp/sd@3,0	sd(0,0,0)	Default disk (1st internal).
disk0	/sbus/esp/sd@3,0	sd(0,0,0)	First internal disk sd0.
disk1	/sbus/esp/sd@1,0	sd(0,1,0)	Second internal disk sd1.
disk2	/sbus/esp/sd@2,0	sd(0,2,0)	External disk sd2.
disk3	/sbus/esp/sd@0,0	sd(0,3,0)	External disk sd3.
tape	/sbus/esp/st@4,0	st(0,0,0)	First tape drive st0.
tape0	/sbus/esp/st@4,0	st(0,0,0)	First tape drive st0.
tape1	/sbus/esp/st@5,0	st(0,1,0)	Second tape drive st1.
cdrom	/sbus/esp/sd@6,0:c	sd(0,6,2)	CD-ROM partition c.
cdrom a	/sbus/esp/sd@6,0: a	sd(0,6,0)	CD-ROM partition a.
net	/sbus/le	le(0,0,0)	Ethernet.
floppy	/fd	fd(0,0,0)	Floppy drive.

Note that in Table 7 the names `sd0`, `sd1`, and so on, are terms used in the Solaris® 1.x operating environment to describe these devices. The Solaris 2.x operating environment names are different, as shown in Table 8 below.

Table 8 Alias Names in the Solaris Operating Environment

Alias	Solaris 1.x Name	Solaris 2.x Name
disk and disk0	sd0	c0t3d0s0
disk1	sd1	c0t1d0s0
disk2	sd2	c0t2d0s0
disk3	sd3	c0t0d0s0

## Running Diagnostics

Several diagnostic routines are available from the Forth Monitor. These on-board tests let you check devices such as the network controller, the floppy disk system, memory, installed SBUS cards and SCSI devices, and the system clock. User-installed devices can be tested if their firmware includes a self-test feature.

Table 9 lists diagnostic test commands. Remember: *device-specifier* refers to either a device path name or a device alias.

Table 9 Diagnostic Test Commands

Command	Description
<code>probe-scsi</code>	Identify devices attached to the built-in SCSI bus.
<code>probe-scsi-all [device-path]</code>	Perform <code>probe-scsi</code> on all SCSI buses installed in the system below the specified device tree node. (If <i>device-path</i> is absent, the root node is used.)
<code>test device-specifier</code>	Execute the specified device's self-test method. For example: <code>test floppy</code> - test the floppy drive, if installed <code>test /memory</code> - test number of megabytes specified in the <code>selftest-#megs</code> NVRAM parameter; or test all of memory if <code>diag-switch?</code> is true <code>test net</code> - test the network connection
<code>test-all [device-specifier]</code>	Test all devices (that have a built-in self-test method) below the specified device tree node. (If <i>device-specifier</i> is absent, the root node is used.)
<code>watch-clock</code>	Test the clock function.
<code>watch-net</code>	Monitor the network connection.

## Testing the SCSI Bus

To check the built-in SCSI bus for connected devices, type:

```
ok probe-scsi
Target 1
  Unit 0 Disk SEAGATE ST1480 SUN04246266 Copyright (C) 1991 Seagate All rights reserved
Target 3
  Unit 0 Disk SEAGATE ST1480 SUN04245826 Copyright (C) 1991 Seagate All rights reserved
ok
```

To test all SCSI buses installed in the system, type:

```
ok probe-scsi-all
/iommu@f,e0000000/sbus@f,e0001000/esp@3,200000
Target 6
  Unit 0 Disk Removable Read Only device SONY CD-ROM CDU-8012 3.1d

/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000
Target 1
  Unit 0 Disk SEAGATE ST1480 SUN04246266 Copyright (C) 1991 Seagate All rights reserved
Target 3
  Unit 0 Disk SEAGATE ST1480 SUN04245826 Copyright (C) 1991 Seagate All rights reserved
ok
```

The response depends on the devices on the SCSI bus.

## Testing Installed Devices

To test a single installed device, type:

```
ok test device-specifier
```

This executes the device method (named `selftest`) of the specified device node. Response depends on the self-test of the device node.

To test a group of installed devices, type:

```
ok test-all
```

All devices below the root node of the device tree are tested. The response depends on the devices that have a self-test method. If you use the *device-specifier* option with the `test-all` command, all devices below the specified device tree node are tested.

### *Testing the Diskette Drive*

The diskette drive test determines whether the diskette drive is functioning properly. A formatted, high-density (HD) disk must be in the diskette drive for this test to be successful.

To test the diskette drive, type:

```
ok test floppy
Testing floppy disk system. A formatted
disk should be in the drive.
Test succeeded.
ok
```

If the test fails, you see an error message.

To eject the diskette, type:

```
ok eject-floppy
ok
```

If this command fails, you can physically eject the diskette by inserting a straightened paper clip into the little hole near the diskette slot.

### *Testing Memory*

When you use the memory testing routine, the system tests the number of megabytes of memory specified in the NVRAM parameter `selftest-#megs`. (See Chapter 3, “Setting Configuration Parameters”, for information about



---

NVRAM parameters.) One megabyte of memory is tested as the default. If either the hardware diagnostic switch (if the system has one) or the NVRAM parameter `diag-switch?` is enabled, all the memory is tested.

To test memory, type:

```
ok test /memory
Testing 16 megs of memory at addr 4000000 11
ok
```

In the preceding example, the first number (4000000) is the base address of the testing, and the following number (11) is the number of megabytes to go.

There will be a delay while the PROM tests the system. If the system fails this test, you see an error message.

### *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 message indicating the result of the test.

---

**Note** – The external loopback portion of this test will fail unless the system is connected to Ethernet.

---

## 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
ok
```

The system responds by incrementing a number once a second. Press any key to stop the test.

## Monitoring the Network

To monitor the network connection, type:

```
ok watch-net
Internal Loopback test - succeeded
External Loopback test - succeeded
Looking for Ethernet packets.
'.' is a good packet. 'X' is a bad packet.
Type any key to stop
.....X.....X.....
ok
```

The system monitors network traffic, displaying “.” each time it receives an error-free packet and “X” each time it receives a packet with an error that can be detected by the network hardware interface.

---

**Note** – Not all OpenBoot 2.x systems include this test word.

---

## Displaying System Information

The Forth Monitor provides several commands to display system information. These commands, listed in Table 10, let you display the system banner, the Ethernet address for the Ethernet controller, the contents of the ID PROM, and

the version number of the OpenBoot firmware. (The ID PROM contains information specific to each machine, including the serial number, date of manufacture, and Ethernet address assigned to the machine.)

Table 10 System Information Display Commands

Command	Description
banner	Display power-on banner.
show-sbus	Display list of installed and probed SBus devices.
.enet-addr	Display current Ethernet address.
.idprom	Display formatted ID PROM contents.
.traps	Display a list of SPARC trap types.
.version	Display version and date of the boot PROM.

Also see the device tree browsing commands in Table 4 on page 8.

**Note** – If you halt the operating system, type `banner`, then resume the system, you may find that your color tables have been altered. To restore these tables on pre-Solaris 2.0 operating environments, type `clear_colormap`, then select Refresh from the Utilities menu. To restore these tables on Solaris 2.0 or 2.1 operating environments, select Color Chooser from the Properties... menu.

## Resetting the System

Occasionally, you may need to reset your system. The `reset` command resets the entire system and is similar to performing a power cycle.

To reset the system, type:

```
ok reset
```

If your system is set up to run the power-on self-test (POST) and initialization procedures on reset, these procedures begin executing when you initiate this command. (On some systems, POST is only executed after power-on.) Once POST completes, the system either boots automatically or enters the Forth Monitor, just as it would have after a power cycle.

---

**Note** – If you were browsing the device tree, you may need to use the `device-end` command (see Chapter 1, “Overview”) before you reset the system.

---

## Setting Configuration Parameters



This chapter describes how to access and modify non-volatile RAM (NVRAM) configuration parameters.

System configuration parameters are stored in the system NVRAM. These parameters determine the start-up machine configuration and related communication characteristics. You can modify the default values of the configuration parameters, and any changes you make remain in effect even after a power cycle. Configuration parameters should always be adjusted cautiously. When correctly used, these parameters give you flexibility in working with your system's hardware.

The procedures described in this chapter assume that the `ok` prompt is displayed on your screen. See Chapter 1, "Overview", for information about entering the Forth Monitor.

Table 11 lists current NVRAM configuration parameters.

*Table 11* NVRAM Configuration Parameters

Parameter	Typical Default	Description
auto-boot?	true	If true, boot automatically after power on or reset.
boot-device	disk	Device from which to boot.
boot-file	empty string	File to boot (an empty string lets secondary booter choose default).
boot-from	vmunix	Boot device and file (1.x only).
boot-from-diag	le()vmunix	Diagnostic boot device and file (1.x only).
diag-device	net	Diagnostic boot source device.

Table 11 NVRAM Configuration Parameters (Continued)

Parameter	Typical Default	Description
diag-file	empty string	File from which to boot in diagnostic mode.
diag-switch?	false	If true, run in diagnostic mode.
fcode-debug?	false	If true, include name fields for plug-in device FCodes.
hardware-revision	no default	System version information.
input-device	keyboard	Power-on input device (usually keyboard, ttya, or ttyb).
keyboard-click?	false	If true, enable keyboard click.
keymap	no default	Keymap for custom keyboard.
last-hardware-update	no default	System update information.
local-mac-address?	false	If true, network drivers use their own MAC address, not system's.
mfg-switch?	false	If true, repeat system self-tests until interrupted with Stop-A.
nvrामrc	empty	Contents of NVRAMRC .
oem-banner	empty string	Custom OEM banner (enabled by oem-banner? true).
oem-banner?	false	If true, use custom OEM banner.
oem-logo	no default	Byte array custom OEM logo (enabled by oem-logo? true). Displayed in hexadecimal.
oem-logo?	false	If true, use custom OEM logo (else, use Sun logo).
output-device	screen	Power-on output device (usually screen, ttya, or ttyb).
sbus-probe-list	0123	Which SBus slots are probed and in what order.
screen-#columns	80	Number of on-screen columns (characters/line).
screen-#rows	34	Number of on-screen rows (lines).
scsi-initiator-id	7	SCSI bus address of host adapter, range 0-7.
sd-targets	31204567	Map SCSI disk units (1.x only).
security-#badlogins	no default	Number of incorrect security password attempts.
security-mode	none	Firmware security level (options: none, command, or full).
security-password	no default	Firmware security password (never displayed). <i>Do not set this directly.</i>
selftest-#megs	1	Megabytes of RAM to test. Ignored if diag-switch? is true.
skip-vme-loopback?	false	If true, POST does not do VMEbus loopback tests.
st-targets	45670123	Map SCSI tape units (1.x only).
sunmon-compat?	false	If true, display Restricted Monitor prompt (>).
testarea	0	One-byte scratch field, available for read/write test.
tpe-link-test?	true	Enable 10baseT link test for built-in twisted pair Ethernet.

Table 11 NVRAM Configuration Parameters (Continued)

Parameter	Typical Default	Description
ttya-mode	9600,8,n,1,-	TTYA (baud rate, #bits, parity, #stop, handshake).
ttyb-mode	9600,8,n,1,-	TTYB (baud rate, #bits, parity, #stop, handshake).
ttya-ignore-cd	true	If true, operating system ignores carrier-detect on TTYA.
ttyb-ignore-cd	true	If true, operating system ignores carrier-detect on TTYB.
ttya-rts-dtr-off	false	If true, operating system does not assert DTR and RTS on TTYA.
ttyb-rts-dtr-off	false	If true, operating system does not assert DTR and RTS on TTYB.
use-nvramrc?	false	If true, execute commands in NVRAMRC during system start-up.
version2?	true	If true, hybrid (1.x/2.x) PROM comes up in version 2.x.
watchdog-reboot?	false	If true, reboot after watchdog reset.

**Note** – Not all OpenBoot systems support all parameters. Defaults may vary depending on the type of system and the PROM revision.

## Displaying and Changing Parameter Settings

NVRAM configuration parameters can be viewed and changed using the commands listed in Table 12.

Table 12 Viewing/Changing Configuration Parameters

Command	Description
printenv	Display all current parameters and current default values. (Numbers are usually shown as decimal values.) <code>printenv parameter</code> shows the current value of the named parameter.
setenv <i>parameter value</i>	Set <i>parameter</i> to the given decimal or text <i>value</i> . (Changes are permanent, but usually only take effect after a reset.)
set-default <i>parameter</i>	Reset the value of the named parameter to the factory default.
set-defaults	Reset parameter values to the factory defaults.

The following pages show how these commands can be used.

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

```

ok printenv
Parameter Name      Value                                     Default Value

oem-logo            2c 31 2c 2d 00 00 00 00 ...
oem-logo?          false                                    false
oem-banner
oem-banner?        false                                    false
output-device      ttya                                     screen
input-device       ttya                                     keyboard
sbus-probe-list    03                                       0123
keyboard-click?   false                                    false
keymap
ttyb-rts-dtr-off   false                                    false
ttyb-ignore-cd     true                                     true
ttya-rts-dtr-off   false                                    false
ttya-ignore-cd     true                                     true
ttyb-mode          9600,8,n,1,-                             9600,8,n,1,-
ttya-mode          9600,8,n,1,-                             9600,8,n,1,-
diag-file
diag-device        net                                       net
boot-file
boot-device        disk                                     disk
auto-boot?         false                                    true
watchdog-reboot?   false                                    false
fcode-debug?       true                                     false
local-mac-address? false                                    false
use-nvramrc?       false                                    false
nvramrc
screen-#columns    80                                       80
screen-#rows       34                                       34
sunmon-compat?     false                                    true
security-mode      none                                    none
security-password
security-#badlogins 0
scsi-initiator-id  7                                       7
version2?          true                                     true
hardware-revision
last-hardware-update
testarea          0                                       0
mfg-switch?        false                                    false
diag-switch?       true                                     false
ok

```



In the displayed, formatted list of the current settings, numeric parameters are shown in decimal, except where otherwise noted.

To change a parameter setting, type:

```
setenv parameter value
```

*parameter* is the name of the parameter. *value* is a numeric value or text string appropriate to the named parameter. A numeric value is typed as a decimal number, unless preceded by 0x, which is the qualifier for a hexadecimal number. *Most parameter changes do not take effect until the next power cycle or system reset.*

For example, to change the setting of the `auto-boot?` parameter from true to false, type:

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

You can reset one or most of the parameters to the original defaults using the `set-default parameter` and `set-defaults` commands.

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

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

To reset most parameters to their default settings, type:

```
ok set-defaults
ok
```

## Setting Security Parameters

The NVRAM system security parameters are:

- security-mode

- `security-password`
- `security-#badlogins`

`security-mode` can restrict the set of actions that unauthorized users are allowed to perform from the Forth Monitor. The three security modes, listed in order of least to most secure, are:

- `none`
- `command`
- `full`

The Restricted Monitor is used to implement the `command` and `full` modes. When security is set to `command` or `full` mode, the OpenBoot firmware will come up in the Restricted Monitor. In `none` security mode, it will come up in either the Forth Monitor or the Restricted Monitor, depending on which one is the default.

In `none` security mode, any command can be typed in the Restricted Monitor, and no password is required. In `command` and `full` security modes, passwords are required to execute certain commands. For example, a password is required to get to the Forth Monitor. Once you enter the Forth Monitor, however, a password is never required.

`security-mode` can be changed with the operating system `EEPROM` utility.

## *Command Security*

With `security-mode` set to `command`, the system comes up in the Restricted Monitor. In this monitor mode,

- A password is not required if you type the `b` command, unless you use the command *with a parameter*.
- The `c` command never asks for a password.
- A password is required to execute the `n` command.

Examples are shown in the following screen.

```
> 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 mode, type the following at the `ok` prompt:

```
ok password
ok New password (only first 8 chars are used):
ok Retype new password:
ok setenv security-mode command
ok
```

---

**Note** – Although this example works, you should normally set the two security parameters with the `EEPROM` command from the operating system.

---

The security password you assign follows the same rules as the `root` password: a combination of six to eight letters and numbers. The security password can be the same as the `root` password, or different from it. You do not have to reset the system; the security feature takes effect as soon as you type the command.



---

**Caution** – It is important to remember your security password. If you forget this password, you cannot use your system; you will have to call Sun's customer support service to make your machine bootable again.

---

If you enter an incorrect security password, there will be a delay of about 10 seconds 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).

## *Full Security*

The full security mode is the most restrictive. With `security-mode` set to `full`, the system comes up in the Restricted Monitor. In this mode:

- A password is required when you type the `b` command.
- The `c` command never asks for a password.
- A password is required to execute the `n` command.

Examples are shown below.

```
> 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, type the following at the `ok` prompt:

```
ok password
ok New password (only first 8 chars are used):
ok Retype new password:
ok setenv security-mode full
ok
```

## *Changing the Power-on Banner*

The banner configuration parameters are:

- `oem-banner`
- `oem-banner?`
- `oem-logo`
- `oem-logo?`

To view the power-on banner, type:

```
ok banner
```

The PROM displays the system banner. The following example shows a SPARCstation 2 banner. The banner for your SPARC system may be different.



```
SPARCstation 2, Type 4 Keyboard  
ROM Rev. 2.0, 16MB memory installed, Serial # 289  
Ethernet address 8:0:20:d:e2:7b, Host ID: 55000121
```

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

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

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

The system displays the banner with your new message, as shown in the following screen.



```
Hello Mom and Dad
```

However, the graphic logo must be handled somewhat differently. `oem-logo` 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 pixel to the right of it, and so on.

To create a new logo, first create a Forth array containing the correct data; then copy this array into `oem-logo`. In the following example, the array is created using Forth commands. (It could also be done under the operating system using the `eeeprom` command.) The array is then copied using the `to` command. The example below fills the top half of `oem-logo` with an ascending pattern.

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

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

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

Because the `oem-logo` array is so large, `printenv` displays approximately the first 8 bytes (in hexadecimal). Use the `oem-logo dump` command to display the entire array. The `oem-logo` array is not erased by `set-defaults`, since it might be difficult to restore the data. However, `oem-logo?` is set to `false` when `set-defaults` executes, so the custom logo is no longer displayed.

## *Input and Output Control*

The configuration parameters related to the control of system input and output are:

- `input-device`
- `output-device`
- `screen-#columns`
- `screen-#rows`
- `ttya-mode`
- `ttyb-mode`

You can use these parameters to assign the power-on defaults for input and output and adjust the communication characteristics of the TTYA and TTYB serial ports. Except for the `ttya-mode` and `ttyb-mode` results, these values do not take effect until the next power cycle or system reset.

## Selecting Input and Output Device Options

The `input-device` and `output-device` parameters control the system's selection of input and output devices after a power-on reset. The default `input-device` value is `keyboard` and the default `output-device` value is `screen`. Input and output can be set to the values in Table 13.

Table 13 I/O Device Parameters

Options	Description
<code>device-specifier</code>	Device identified by that device path name or alias.
<code>keyboard</code>	(Input only) Default system keyboard.
<code>screen</code>	(Output only) Default graphics display.
<code>ttya</code>	Serial port A.
<code>ttyb</code>	Serial port B.

When the system is reset, the named device becomes the default input or output device. (If you want to temporarily change the input or output device, use the `input` or `output` commands described in Chapter 4, “Using Forth Tools.”)

To set TTYA as the power-on default input device, type:

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

If you select `keyboard` for `input-device`, and the device is not plugged in, input is accepted from `ttya` after the next power cycle or system reset. If you select `screen` for `output-device`, but no frame buffer is available, output is sent to `ttya` after the next power cycle or system reset.

To specify an SBus `bwtwo` frame buffer as the default output device (especially if there are multiple frame buffers in the system), type:

```
ok setenv output-device /sbus/bwtwo
ok
```

### *Setting Serial Port Characteristics*

The default settings for both TTYA and TTYB for most Sun systems are:

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

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

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

For example, to set TTYA to 1200 baud, seven data bits, even parity, one stop bit, and no handshake, type:

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

Changes to these parameter values take effect immediately.

---

**Note** – rts/cts and xon/xoff handshaking are not implemented on some systems. When a selected protocol is not implemented, the handshake parameter is accepted but ignored; no messages are displayed.

---

### *Selecting Boot Options*

You can use the following configuration parameters to determine whether or not the system will boot automatically after a power cycle or system reset.



- auto-boot?
- boot-device
- boot-file

If `auto-boot?` is true, then the system boots automatically (using the `boot-device` and `boot-file` values).

These parameters can also be used during manual booting to select the boot device and the program to be booted. For example, to specify auto-booting from the Ethernet server, type:

```
ok setenv boot-device net
ok boot
```

Specified booting usually begins immediately.

---

**Note** - `boot-device` and `boot-file` are specified differently with `diag-switch?` set to true. See the next section for more information.

---

## *Controlling Power-on Self-test*

The power-on testing parameters are:

- diag-device
- diag-file
- diag-switch?
- mfg-switch?
- selftest-#megs

Most systems have a factory default of false for the `diag-switch?` parameter. To set `diag-switch?` to true, type:

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

Enabling `diag-switch?` causes the system to perform more thorough self-tests during any subsequent power-on process. Once `diag-switch?` is enabled, additional status messages are sent out (some to TTYA and some to

the specified output device), *all* of memory is tested, and different default boot options are used. The boot PROM tries to boot the program specified by the `diag-file` parameter, from the device specified by `diag-device`.

---

**Note** – Some SPARC systems have a hardware diagnostic switch. The system runs the full tests on power-on if either the hardware switch or `diag-switch?` is set.

---

You can also force `diag-switch?` to true by using the Stop-D key sequence during power-on.

To set `diag-switch?` to false, type:

```
ok setenv diag-switch? false
ok
```

When `diag-switch?` is false, the system does not call out the diagnostic tests as they are run (unless a test fails) and runs a reduced set of diagnostics.

## Using NVRAMRC

A portion of NVRAM, whose size depends on the particular SPARC system, is called NVRAMRC. It is reserved to store user-defined commands that are executed during start-up.

Typically, NVRAMRC would be used by a device driver to save start-up configuration parameters, to patch device driver code, or to define installation-specific device configuration and device aliases. It also could be used for bug patches or for user-installed extensions. Commands are stored in ASCII, just as the user would type them at the console.

There are two NVRAMRC-related configuration parameters:

- `nvrामrc`
- `use-nvrामrc?`

Commands in NVRAMRC are executed during system start-up if `use-nvrामrc?` is set to true. Almost all Forth Monitor commands can be used here. *The following are exceptions:*

- `banner` (use with caution)

- boot
- go
- nvedit
- password
- reset
- setenv security-mode

### *Editing the Contents of NVRAMRC*

The NVRAMRC editor, `nvedit`, lets you create and modify the contents of NVRAMRC using the commands listed in Table 14.

*Table 14* NVRAMRC Editor Commands

Command	Description
<code>nvalias alias device-path</code>	Store the command " <code>devalias alias device-path</code> " in NVRAMRC. The alias persists until the <code>nvunalias</code> or <code>set-defaults</code> commands are executed.
<code>nvedit</code>	Enter the NVRAMRC editor. If data remains in the temporary buffer from a previous <code>nvedit</code> session, resume editing those previous contents. If not, read the contents of NVRAMRC into the temporary buffer and begin editing them.
<code>nvquit</code>	Discard the contents of the temporary buffer, without writing it to NVRAMRC. Prompt for confirmation.
<code>nvrecover</code>	Recover the contents of NVRAMRC if they have been lost as a result of the execution of <code>set-defaults</code> ; then enter the editor as with <code>nvedit</code> . <code>nvrecover</code> fails if <code>nvedit</code> is executed between the time that the NVRAMRC contents were lost and the time that <code>nvrecover</code> is executed.
<code>nvrn</code>	Execute the contents of the temporary buffer.
<code>nvstore</code>	Copy the contents of the temporary buffer to NVRAMRC; discard the contents of the temporary buffer.
<code>nvunalias alias</code>	Delete the corresponding alias from NVRAMRC.

---

**Note** – Not all OpenBoot 2.x systems include the `nvalias` and `nvunalias` commands.

---

The editing commands shown in Table 15 are used within the NVRAM

Table 15 `nvedit` Keystroke Commands

Keystroke	Description
Control-B	Move backward one character.
Control-C	Exit the editor and return to the OpenBoot command interpreter. The temporary buffer is preserved but is not written back to NVRAMRC. (Use <code>nvstore</code> afterwards to write back the temporary buffer.)
Control-F	Move forward one character.
Control-K	If at the end of a line, join the next line to the current line (that is, delete the new line).
Control-L	List all lines.
Control-N	Move to the next line of the NVRAMRC editing buffer.
Control-O	Insert a new line at the cursor position and stay on the current line.
Control-P	Move to the previous line of the NVRAMRC editing buffer.
Delete	Delete the previous character.
Return	Insert a new line at the cursor position and advance to the next line.

Other standard line editor commands are described in Chapter 4, “Using Forth Tools.”

### Activating an NVRAMRC File

Use the following steps to activate an NVRAMRC command file:

1. **At the `ok` prompt, type `nvedit`**  
 Edit the contents of NVRAMRC using editor commands.
2. **Type `Control-C` to get out of the editor and back to the `ok` prompt.**
3. **Type `nvstore` to save your changes.**
4. **Enable the interpretation of NVRAMRC by typing:**  

```
setenv use-nvramrc? true
```
5. **Type `reset` to reset the system and execute the NVRAM contents, or type `nvramrc eval` to execute the contents directly. If you have not yet typed `nvstore` to save your changes, type `nvrn` to execute the contents of the temporary edit buffer.**

---

The following example shows you how to create a simple colon definition in NVRAMRC.

```
ok nvedit
0: : hello ( -- )
1: ." Hello, world. " cr
2: ;
3: ^-C
ok nvstore
ok setenv use-nvramrc? true
ok reset
....
ok hello
Hello, world.
ok
```

Notice the `nvedit` line number prompts (0:, 1:, 2:, 3:) in the above example. These prompts may be different on some systems.



## Using Forth Tools

---



This chapter introduces Forth as it is implemented in OpenBoot. Even if you are familiar with the Forth programming language, work through the examples shown in this chapter; they provide specific, OpenBoot-related information.

The version of Forth contained in OpenBoot is based on ANS Forth. Appendix E, “Forth Word Reference” lists the complete set of available commands. Words that are specifically used for writing OpenBoot FCode programs for SBus devices are described in the manual, *Writing FCode 2.x Programs*.

---

**Note** – This chapter assumes that you know how to enter and leave the User Interface. At the `ok` prompt, if you type commands that hang the system and you cannot recover using a key sequence, you may need to perform a power cycle to return the system to normal operation.

---

### *Forth Commands*

Forth has a very simple command structure. Forth commands, also called Forth *words*, consist of any combination of characters that can be printed—for example, letters, digits, or punctuation marks. Examples of legitimate words are shown below:

```
@  
dump
```

```
.
```

```
0<
+
probe-pci
```

To be recognized as commands, *Forth words must be separated by one or more spaces* (blanks). Pressing Return at the end of any command line executes the typed commands. (In all the examples shown, a Return at the end of the line is assumed.)

A command line can have more than one word. Multiple words on a line are executed one at a time, from left to right, in the order in which they were typed. For example:

---

```
ok testa testb testc
ok
```

---

is equivalent to:

---

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

---

In OpenBoot, uppercase and lowercase letters are equivalent. Therefore, `testa`, `TESTA`, and `TeStA` all invoke the same command. However, words are conventionally written in lowercase.

Some commands generate large amounts of output (for example, `dump` or `words`). You can interrupt such a command by pressing any key except `q`. (If you press `q`, the output is aborted, not suspended.) Once a command is interrupted, output is suspended and the following message appears:

---

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

---

Press the space bar (`<space>`) to continue, press Return (`<cr>`) 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 automatically displays this prompt at the end of each page.



## Using Numbers

Enter a number by typing its value, for example, 55 or -123. Forth accepts only integers (whole numbers); fractional values (for example, 2/3) are not allowed. A period at the end of a number signifies a double number. Periods or commas embedded in a number are ignored, so 5.77 is understood as 577. By convention, such punctuation usually appears every four digits. Use one or more spaces to separate a number from a word or from another number.

OpenBoot performs 32-bit integer arithmetic, and all numbers are 32-bit values unless otherwise specified.

Although OpenBoot implementations are encouraged to provide a hexadecimal conversion radix, they are not required to do so. So, you must establish such a radix if your code depends on a given base for proper operation.

You can change the operating number base with the commands `octal`, `decimal` and `hex` which cause all subsequent numeric input and output to be performed in base 8, 10 or 16, respectively.

For example, to operate in decimal, type:

---

```
ok decimal
ok
```

---

To change to hexadecimal type:

---

```
ok hex
ok
```

---

Two simple techniques for identifying the active number base are:

---

```
ok 10 .d
16
ok 10 1- .
f
ok
```

---

The 16 and the f on the display show that you are operating in hexadecimal. If 10 and 9 showed on the display, it would mean that you are in decimal base. 8 and 7 would indicate octal.

## 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 of the stack is the first one you take off. *Understanding the stack is essential to using Forth.*

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
```

---

## Displaying Stack Contents

The contents of the stack are normally invisible. 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 showstack
ok
```

---

The topmost stack item is always shown as the last item in the list, immediately before the ok prompt. In the above example, the topmost stack item is 8.

If showstack has been previously executed, noshowstack will remove the stack display prior to each prompt.

---

**Note** – In some of the examples in this chapter, `showstack` is enabled. In those examples, each `ok` prompt is immediately preceded by a display of the current contents of the stack. The examples work the same if `showstack` is not enabled, except that the stack contents are not displayed.

---

Nearly all words that require numeric parameters fetch those parameters from the top of the stack. Any values returned are generally left on top of the stack, where they can 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. In the example below, all arithmetic is in hexadecimal.

---

```
44 7 8 ok +
44 f ok +
53 ok
```

---

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

---

```
53 ok 12
53 12 ok .
12
53 ok .
53
ok (The stack is now empty)
ok 3 5 + .
8
ok (The stack is now empty)
ok .
Stack Underflow
ok
```

---

## *The Stack Diagram*

To aid understanding, conventional coding style requires that a *stack diagram* of the form ( `--` ) appears on the first line of every definition of a Forth word. The stack diagram specifies what happens to the stack with the execution of the word.

Entries to the left of -- show stack items that are consumed (i.e. removed) from the stack and used by the operation of that word. Entries to the right of -- show stack items that are left on the stack after the word finishes execution. For example, the stack diagram for the word + is: ( nu1 nu2 -- sum ), and the stack diagram for the word . is: ( nu -- ). Therefore, + removes two numbers (nu1 and nu2), then leaves their sum (sum) on the stack. The word . removes the number on the top of the stack (nu) and displays it.

Words that have no effect on the contents of the stack (such as showstack or decimal), have a ( -- ) stack diagram.

Occasionally, a word will require another word or other text immediately following it. For example, the word see, used in the form see *thisword* ( -- ).

Stack items are generally written using descriptive names to help clarify correct usage. See Table 16 for stack item abbreviations used in this manual.

Table 16 Stack Item Notation

Notation	Description
	Alternate stack results shown with space, e.g. ( input -- addr len false   result true ).
	Alternate stack items shown without space, e.g. ( input -- addr len   0 result ).
???	Unknown stack item(s).
...	Unknown stack item(s). If used on both sides of a stack comment, means the same stack items are present on both sides.
< > <space>	Space delimiter. Leading spaces are ignored.
a-addr	Variable-aligned address.
addr	Memory address (generally a virtual address).
addr len	Address and length for memory region
byte bxxx	8-bit value (low order byte in a 32-bit word).
char	7-bit value (low order byte), high bit unspecified.
cnt len size	Count or length.
dxxx	Double (extended-precision) numbers. 2 stack items, hi quadlet on top of stack.
<eol>	End-of-line delimiter.
false	0 (false flag).
ihandle	Pointer for an instance of a package.
n n1 n2 n3	Normal signed values (32-bit).
nu nu1	Signed or unsigned values (32-bit).

Table 16 Stack Item Notation (Continued)

Notation	Description
<nothing>	Zero stack items.
phandle	Pointer for a package.
phys	Physical address (actual hardware address).
phys.lo phys.hi	Lower / upper cell of physical address
pstr	Packed string.
quad qxxx	Quadlet (32-bit value).
qaddr	Quadlet (32-bit) aligned address
{text}	Optional text. Causes default behavior if omitted.
"text<delim> "	Input buffer text, parsed when command is executed. Text delimiter is enclosed in <>.
[text<delim> ]	Text immediately following on the same line as the command, parsed immediately. Text delimiter is enclosed in <>.
true	-1 (true flag).
uxxx	Unsigned value, positive values (32-bit).
virt	Virtual address (address used by software).
waddr	Doublet (16-bit) aligned address
word wxxx	Doublet (16-bit value, low order two bytes in a 32-bit word).
x x1	Arbitrary stack item.
x.lo x.hi	Low/high significant bits of a data item
xt	Execution token.
xxx?	Flag. Name indicates usage (e.g. done? ok? error?).
xyz-str xyz- len	Address and length for unpacked string.
xyz-sys	Control-flow stack items, implementation-dependent.
( C: -- )	Compilation stack diagram
( -- ) ( E: -- )	Execution stack diagram
( R: -- )	Return stack diagram

## Manipulating the Stack

Stack manipulation commands (described in Table 17) allow you to add, delete,

and reorder items on the stack.

Table 17 Stack Manipulation Commands

Command	Stack Diagram	Description
-rot	( x1 x2 x3 -- x3 x1 x2 )	Inversely rotate 3 stack items.
>r	( x -- ) ( R: -- x )	Move a stack item to the return stack. (Use with caution.)
?dup	( x -- x x   0 )	Duplicate the top stack item if it is non-zero.
2drop	( x1 x2 -- )	Remove 2 items from the stack.
2dup	( x1 x2 -- x1 x2 x1 x2 )	Duplicate 2 stack items.
2over	( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 )	Copy second 2 stack items.
2rot	( x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2 )	Rotate 3 pairs of stack items.
2swap	( x1 x2 x3 x4 -- x3 x4 x1 x2 )	Exchange 2 pairs of stack items.
3drop	( x1 x2 x3 -- )	Remove 3 items from the stack.
3dup	( x1 x2 x3 -- x1 x2 x3 x1 x2 x3 )	Duplicate 3 stack items.
clear	( ??? -- )	Empty the stack.
depth	( -- u )	Return the number of items on the stack.
drop	( x -- )	Remove top item from the stack.
dup	( x -- x x )	Duplicate the top stack item.
nip	( x1 x2 -- x2 )	Discard the second stack item.
over	( x1 x2 -- x1 x2 x1 )	Copy second stack item to top of stack.
pick	( xu ... x1 x0 u -- xu ... x1 x0 xu )	Copy u-th stack item (1 pick = over).
r>	( -- x ) ( R: x -- )	Move a return stack item to the stack. (Use with caution.)
r@	( -- x ) ( R: x -- x )	Copy the top of the return stack to the stack.
roll	( xu ... x1 x0 u -- xu-1 ... x1 x0 xu )	Rotate u stack items (2 roll = rot).
rot	( x1 x2 x3 -- x2 x3 x1 )	Rotate 3 stack items.
swap	( x1 x2 -- x2 x1 )	Exchange the top 2 stack items.
tuck	( x1 x2 -- x2 x1 x2 )	Copy top stack item below second item.

A typical use of stack manipulation might be to display the top stack item while preserving all stack items, as shown in this example:

```
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
```

## Creating Custom Definitions

Forth provides an easy way to create custom definitions for new command words. Table 18 shows the Forth words used to create custom definitions.

Table 18 Color Definition Words

Command	Stack Diagram	Description
: <i>new-name</i>	( -- )	Start a new colon definition of the word <i>new-name</i> .
;	( -- )	End a colon definition.

Definitions for new commands are called *colon definitions*, named after the word `:` that is used to create them. For example, suppose you want to create a new word, `add4`, that will add any four numbers together and display the result. You could create the definition as follows:

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

The `;` (semicolon) marks the end of the definition that defines `add4` to have the behavior `(+ + + .)`. The three addition operators `(+)` reduce the four stack items to a single sum on the stack; then `.` removes and displays that result. An example follows.

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

Definitions are stored in local memory, which means they are erased when a system resets. To keep useful definitions, put them into a text file (using a text editor under your operating system or using the NVRAMRC editor). This text file can then be loaded as needed. (See Chapter 5, “Loading and Executing Programs”, for more information on loading files.)

When you type a definition from the User Interface, the `ok` prompt becomes a `] (right square bracket)` prompt after you type the `:` (colon) and before you type the `;` (semicolon). For example, you could type the definition for `add4` like this:

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

Every definition you create (in a text file) should have a stack effect diagram shown with that definition, even if the stack effect is `nil ( -- )`. This is vital because the stack diagram shows the proper use of that word. Also, use generous stack comments within complex definitions; this helps trace the flow of execution. For example, when creating `add4`, you could define it as:

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

Or you could define `add4` as follows:

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

---

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

---



## Using Arithmetic Functions

The commands listed in Table 19 perform basic arithmetic with items on the data stack.

Table 19 Arithmetic Functions

Command	Stack Diagram	Description
+	( nu1 nu2 -- sum )	Add $nu1 + nu2$ .
-	( nu1 nu2 -- diff )	Subtract $nu1 - nu2$ .
*	( nu1 nu2 -- prod )	Multiply $nu1 * nu2$ .
/	( n1 n2 -- quot )	Divide $n1$ by $n2$ ; remainder is discarded.
/mod	( n1 n2 -- rem quot )	Remainder, quotient of $n1 / n2$ .
<<	( x1 u -- x2 )	Synonym for <code>lshift</code> .
>>	( x1 u -- x2 )	Synonym for <code>rshift</code> .
>>a	( x1 u -- x2 )	Arithmetic right-shift $x1$ by $u$ bits.
*/	( n1 n2 n3 -- quot )	$n1 * n2 / n3$ .
*/mod	( n1 n2 n3 -- rem quot )	Remainder, quotient of $n1 * n2 / n3$ .
1+	( nu1 -- nu2 )	Add 1.
1-	( nu1 -- nu2 )	Subtract 1.
2*	( nu1 -- nu2 )	Multiply by 2.
2+	( nu1 -- nu2 )	Add 2.
2-	( nu1 -- nu2 )	Subtract 2.
2/	( nu1 -- nu2 )	Divide by 2.
abs	( n -- u )	Absolute value.
aligned	( n1 -- n1   a-addr )	Round $n1$ up to the next multiple of 4.
and	( n1 n2 -- n3 )	Bitwise logical AND.
bounds	( startaddr len -- endaddr startaddr )	Convert <i>startaddr len</i> to <i>endaddr startaddr</i> for <code>do</code> loop.
bljoin	( b.low b2 b3 b.hi -- quad )	Join four bytes to form a 32-bit quadword.
bwjoin	( b.low b.hi -- word )	Join two bytes to form a 16-bit word.
d+	( d1 d2 -- d.sum )	Add two 64-bit numbers.
d-	( d1 d2 --d.diff )	Subtract two 64-bit numbers.
even	( n -- n   n+1 )	Round to nearest even integer $\geq n$ .
fm/mod	( d n -- rem quot )	Divide $d$ by $n$ .

Table 19 Arithmetic Functions (Continued)

Command	Stack Diagram	Description
invert	( x1 -- x2 )	Invert all bits of <i>x1</i> .
lbflip	( quad1 -- quad2 )	Swap the bytes within a 32-bit quadword
lbsplit	( quad -- b.low b2 b3 b.hi )	Split a 32-bit quadword into four bytes.
lwflip	( quad1 -- quad2 )	Swap halves of a 32-bit quadword.
lwsplit	( quad -- w.low w.hi )	Split a 32-bit quadword into two 16-bit words.
lshift	( x1 u -- x2 )	Left-shift <i>x1</i> by <i>u</i> bits. Zero-fill low bits.
max	( n1 n2 -- n3 )	<i>n3</i> is maximum of <i>n1</i> and <i>n2</i> .
min	( n1 n2 -- n3 )	<i>n3</i> is minimum of <i>n1</i> and <i>n2</i> .
mod	( n1 n2 -- rem )	Remainder of <i>n1</i> / <i>n2</i> .
negate	( n1 -- n2 )	Change the sign of <i>n1</i> .
not	( x1 -- x2 )	Synonym for invert.
or	( n1 n2 -- n3 )	Bitwise logical OR.
rshift	( x1 u -- x2 )	Right-shift <i>x1</i> by <i>u</i> bits. Zero-fill high bits.
s>d	( n1 -- d1 )	Convert a number to a double number.
sm/rem	( d n -- rem quot )	Divide <i>d</i> by <i>n</i> , symmetric division.
u2/	( x1 -- x2 )	Logical right shift 1 bit; zero shifted into high bit.
u*	(u1 u2 -- uprod )	Multiply 2 unsigned numbers yielding an unsigned product.
u/mod	( u1 u2 -- urem uquot )	Divide unsigned 32-bit number by an unsigned 32-bit number; yield 32-bit remainder and quotient.
um*	( u1 u2 -- ud )	Multiply 2 unsigned 32-bit numbers; yield unsigned double number product.
um/mod	( ud u -- urem uprod )	Divide <i>ud</i> by <i>u</i> .
wbflip	( word1 -- word2 )	Swap the bytes within a 16-bit word.
wbsplit	( word -- b.low b.hi )	Split 16-bit word into two bytes.
wljoin	( w.low w.hi -- quad )	Join two words to form a quadword.
xor	( x1 x2 --x3 )	Bitwise exclusive OR.

## Accessing Memory

The User Interface provides interactive commands for examining and setting memory. Use the User Interface to:

- Read and write to any virtual address.
- Map virtual addresses to physical addresses.

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

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

---

**Note** – “l” is sometimes printed in uppercase to avoid confusion with 1 (the number one).

---

*waddr*, *qaddr*, and *addr64* indicate addresses with alignment restrictions. For example, *qaddr* indicates 32-bit (4 byte) alignment; so this address must be evenly divisible by 4, as shown in the following example:

```
ok 4028 L@
ok 4029 L@
Memory address not aligned
ok
```

The Forth interpreter implemented in OpenBoot adheres closely to the ANS Forth Standard. If you explicitly want a 16-bit fetch or a 32-bit fetch, use *w@* or *L@* instead of *@*. Other commands also follow this convention.

Table 20 lists the commands used to access memory.

Table 20 Memory Access Commands

Command	Stack Diagram	Description
!	( x a-addr -- )	Store a number at <i>a-addr</i> .
+!	( nu a-addr -- )	Add <i>nu</i> to the number stored at <i>a-addr</i> .
<w@	( waddr -- n )	Fetch doublet <i>w</i> from <i>waddr</i> ; sign-extended.

Table 20 Memory Access Commands (Continued)

Command	Stack Diagram	Description
@	( a-addr --x )	Fetch a number from <i>a-addr</i> .
2!	( x1 x2 a-addr -- )	Store 2 numbers at <i>a-addr</i> , <i>x2</i> at lower address.
2@	( a-addr -- x1 x2 )	Fetch 2 numbers from <i>a-addr</i> , <i>x2</i> from lower address.
blank	( addr len -- )	Set <i>len</i> bytes of memory beginning at <i>addr</i> to space (decimal 32).
c!	(byte addr --)	Store <i>byte</i> at <i>addr</i> .
c@	( addr -- byte )	Fetch a <i>byte</i> from <i>addr</i> .
cmove	( addr1 addr2 u -- )	Copy <i>u</i> bytes from <i>addr1</i> to <i>addr2</i> , starting at low byte.
cmove>	( addr1 addr2 u -- )	Copy <i>u</i> bytes from <i>addr1</i> to <i>addr2</i> , starting at high byte.
cpeek	( addr -- false   byte true )	Fetch the byte at <i>addr</i> . Return the data and <code>true</code> if the access was successful. Return <code>false</code> if a read access error occurred.
cpoke	( byte addr -- okay? )	Store the <i>byte</i> to <i>addr</i> . Return <code>true</code> if the access was successful. Return <code>false</code> if a write access error occurred.
comp	( addr1 addr2 len -- diff? )	Compare two byte arrays. <i>diff?</i> = 0 if arrays are identical, <i>diff?</i> = -1 if first byte that is different is lesser in string at <i>addr1</i> , <i>diff?</i> = 1 otherwise.
dump	( addr len -- )	Display <i>len</i> bytes of memory starting at <i>addr</i> .
erase	( addr len -- )	Set <i>len</i> bytes of memory beginning at <i>addr</i> to 0.
fill	( addr len byte -- )	Set <i>len</i> bytes of memory beginning at <i>addr</i> to the value <i>byte</i> .
l!	( n qaddr -- )	Store a quadlet <i>q</i> at <i>qaddr</i> .
l@	( qaddr -- quad )	Fetch a quadlet <i>q</i> from <i>qaddr</i> .
lbflips	( qaddr len -- )	Reverse the bytes within each quadlet in the specified region.
lwflips	( qaddr len -- )	Swap the doublets within each quadlet in specified region.
lpeek	( qaddr -- false   quad true )	Fetch the 32-bit quantity at <i>qaddr</i> . Return the data and <code>true</code> if the access was successful. Return <code>false</code> if a read access error occurred.
lpoke	( quad qaddr -- okay? )	Store the 32-bit quantity at <i>qaddr</i> . Return <code>true</code> if the access was successful. Return <code>false</code> if a write access error occurred.
move	( src-addr dest-addr len -- )	Copy <i>len</i> bytes from <i>src-addr</i> to <i>dest-addr</i> .
off	( a-addr -- )	Store <code>false</code> at <i>a-addr</i> .
on	( a-addr -- )	Store <code>true</code> at <i>a-addr</i> .
unaligned-l!	( quad addr -- )	Store a quadlet <i>q</i> , any alignment

Table 20 Memory Access Commands (Continued)

Command	Stack Diagram	Description
unaligned-l@	( addr -- quad )	Fetch a quadlet <i>q</i> , any alignment.
unaligned-w!	( w addr -- )	Store a doublet <i>w</i> , any alignment.
unaligned-w@	( addr -- w )	Fetch a doublet <i>w</i> , any alignment.
w!	( w waddr -- )	Store a doublet <i>w</i> at <i>waddr</i> .
w@	( waddr -- w )	Fetch a doublet <i>w</i> from <i>waddr</i> .
wbflips	( waddr len -- )	Swap the bytes within each doublet in the specified region.
wpeek	( waddr -- false   w true )	Fetch the 16-bit quantity at <i>waddr</i> . Return the data and <code>true</code> if the access was successful. Return <code>false</code> if a read access error occurred.
wpoke	( w waddr -- okay? )	Store the 16-bit quantity to <i>waddr</i> . Return <code>true</code> if the access was successful. Return <code>false</code> if a write access error occurred.

The `dump` command is particularly useful. It displays a region of memory as both bytes and ASCII values. The example below displays the contents of 20 bytes of memory starting at virtual address 10000. It also shows you how to read from and write 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
```

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

Table 21 lists memory mapping commands.

*Table 21* Memory Mapping Commands

<b>Command</b>	<b>Stack Diagram</b>	<b>Description</b>
alloc-mem	( size -- virt )	Allocate and map <i>size</i> bytes of available memory; return the virtual address.
free-mem	( virt size -- )	Free memory allocated by <code>alloc-mem</code> .
free-virtual	( virt size -- )	Undo mappings created with <code>memmap</code> .

The following screen is an example of the use of `alloc-mem` and `free-mem`.

- `alloc-mem` allocates 4000 bytes of memory, and the starting address (`ffef7a48`) of the reserved area is displayed.
- `dump` displays the contents of 20 bytes of memory starting at `ffef7a48`.
- This region of memory is then filled with the value 55.

- Finally, `free-mem` returns the 4000 allocated bytes of memory starting at `ffef7a48`.

```

ok
ok 4000 alloc-mem .
ffef7a48
ok
ok ffef7a48 constant temp
ok temp 20 dump
      0 1 2 3 4 5 6 7 \ / 9 a b c d e f 01234567v9abcdef
ffef7a40 00 00 f5 5f 00 00 40 08 ff ef c4 40 ff ef 03 c8
..u_..@..oD@.o.H
ffef7a50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
ffef7a60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
ok temp 20 55 fill
ok temp 20 dump
      0 1 2 3 4 5 6 7 \ / 9 a b c d e f 01234567v9abcdef
ffef7a40 00 00 f5 5f 00 00 40 08 55 55 55 55 55 55 55 55
..u_..@.UUUUUUUU
ffef7a50 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
UUUUUUUUUUUUUUUU
ffef7a60 55 55 55 55 55 55 55 55 00 00 00 00 00 00 00 00
UUUUUUUU.....
ok
ok temp 4000 free-mem
ok

```

An example of using `memmap` is shown below.

```

ok 200.0000 sbus 1000 memmap ( virt )
ok

```

## Mapping An SBus Device

Here is a general method for mapping an SBus device from the `ok` prompt, without the necessity of knowing system-dependent device addresses. This method does not depend on the presence of a valid FCode PROM on the SBus device. The method will work on any OpenBoot system version 2.0 or higher.

```
ok " /sbus" select-dev
ok (offset) (slot#) (size) map-in ( virt )
ok
```

For example, to inspect the FCode PROM for a device in slot #3 of a system, enter:

```
ok " /sbus" select-dev
ok 0 3 1000 map-in .s
ffed3000
ok dup 20 dump
(Dump of first 20 bytes of FCode PROM)
ok
```

Here are some variations to the method:

1. On some systems, the pathname for the system SBus may vary. For example, “/iommu/sbus” (for Sun4m) or “/io-unit/sbi” (for Sun4d). The `show-devs` command from the `ok` prompt (which lists all system devices) is one way to determine the correct path.
2. Direct placement of (offset size) on the stack may or may not work in the most general cases on future systems. If you encounter problems, try the following, more general approach:

```
ok " /sbus" select-dev
ok " 3,0: decode-unit ( offset space )
ok 1000 map-in ( virt )
ok
```



## Using Defining Words

The *dictionary* contains all the available Forth commands. Defining words are used to create new Forth commands.

Defining words require two stack diagrams. The first diagram shows the stack effect when the new command is created. The second (or “Usage:”) diagram shows the stack effect when that command is later executed.

Table 22 lists the defining words that you can use to create dictionary entries.

Table 22 Defining Words

Command	Stack Diagram	Description
<code>: name</code>	( -- ) Usage: ( ??? -- ? )	Start creating a new colon definition.
<code>;</code>	( -- )	Finish creating a new colon definition.
<code>alias new-name old-name</code>	( -- ) Usage: ( ??? -- ? )	Create <i>new-name</i> with the same behavior as <i>old-name</i> .
<code>buffer: name</code>	( size -- ) Usage: ( -- a-addr )	Create a named array in temporary storage.
<code>constant name</code>	( n -- ) Usage: ( -- n )	Define a constant (for example, <code>3 constant bar</code> ).
<code>2constant name</code>	( n1 n2 -- ) Usage: ( -- n1 n2 )	Define a 2-number constant.
<code>create name</code>	( -- ) Usage: ( -- waddr )	Generic defining word.
<code>defer name</code>	( -- ) Usage: ( ??? -- ? )	Define a word for forward references or execution vectors using execution token.
<code>does&gt;</code>	( -- waddr )	Start the run-time clause for defining words.
<code>field name</code>	( offset size -- offset+size ) Usage: ( addr -- addr+offset )	Create a named offset pointer.
<code>struct</code>	( -- 0 )	Initialize for <code>field</code> creation.
<code>value name</code>	( n -- ) Usage: ( -- n )	Create a changeable, named 32-bit quantity.
<code>variable name</code>	( -- ) Usage: ( -- waddr )	Define a variable.

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

```
ok 72 constant red
ok
ok red .
72
ok
```

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

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

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

```
ok 43 value thisval
ok thisval .
43
ok 10 to thisval
ok thisval .
10
ok
```

Commands created with `value` are convenient, because you do not have to use `@` every time you want the number.

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

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

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

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

## Searching the Dictionary

The *dictionary* contains all the available Forth commands. Table 23 lists tools you can use to search the dictionary.

Table 23 Dictionary Searching Commands

Command	Stack Diagram	Description
<code>' name</code>	( -- xt )	Find the named word in the dictionary. Returns the execution token. Use outside definitions.
<code>['] name</code>	( -- xt )	Similar to <code>'</code> but is used either inside or outside definitions.
<code>.calls</code>	( xt -- )	Display a list of all words that call the word whose execution token is <code>xt</code> .

Table 23 Dictionary Searching Commands

Command	Stack Diagram	Description
\$find	( addr len -- addr len false   xt n )	Find a word. <i>n</i> = 0 if not found, <i>n</i> = 1 if immediate, <i>n</i> = -1 otherwise.
find	( pstr -- pstr false   xt n )	Search for a word in the dictionary. The word to be found is indicated by <i>pstr</i> . <i>n</i> = 0 if not found, <i>n</i> = 1 if immediate, <i>n</i> = -1 otherwise.
see <i>thisword</i>	( -- )	Decompile the named command.
(see)	( xt -- )	Decompile the word indicated by the execution token.
sift	( pstr -- )	Display names of all dictionary entries containing the string pointed to by <i>pstr</i> .
sifting <i>ccc</i>	( -- )	Display names of all dictionary entries containing the sequence of characters. <i>ccc</i> contains no spaces.
words	( -- )	Display all visible words in the dictionary.

see, used in the form see *thisword*, decompiles the specified command (that is, it shows the definition used to create *thisword*). The decompiled definition may sometimes be confusing, because some internal names may have been omitted from the PROM's symbol table to save space.

The following screen is an example of how to use `sifting`.

```
ok sifting input
input-device input restore-input line-input input-line input-file
ok
```

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

## Compiling Data into the Dictionary

The commands listed in Table 24 control the compilation of data into the dictionary.

Table 24 Dictionary Compilation Commands

Command	Stack Diagram	Description
,	( n -- )	Place a number in the dictionary.
c ,	( byte -- )	Place a byte in the dictionary.
w ,	( word -- )	Place a 16-bit number in the dictionary.
l ,	( quad -- )	Place a 32-bit number in the dictionary.
[	( -- )	Begin interpreting.
]	( -- )	End interpreting, resume compilation.
allot	( n -- )	Allocate <i>n</i> bytes in the dictionary.
>body	( xt -- a-addr )	Find the data field address from the execution token.
body>	( a-addr -- xt )	Find the execution token from the data field address.
compile	( -- )	Compile next word at run time.
[compile] <i>name</i>	( -- )	Compile the next (immediate) word.
forget <i>namep</i>	( -- )	Remove word from dictionary and all subsequent words.
here	( -- addr )	Address of top of dictionary.
immediate	( -- )	Mark the last definition as immediate.
to <i>name</i>	( n -- )	Install a new action in a defer word or value.
literal	( n -- )	Compile a number.
origin	( -- addr )	Return the address of the start of the Forth system.
patch <i>new-word old-word word-to-patch</i>	( -- )	Replace <i>old-word</i> with <i>new-word</i> in <i>word-to-patch</i> .

Table 24 Dictionary Compilation Commands (Continued)

Command	Stack Diagram	Description
(patch	( new-n old-n xt -- )	Replace <i>old-n</i> with <i>new-n</i> in word indicated by <i>xt</i> .
recursive	( -- )	Make the name of the colon definition being compiled visible in the dictionary, and thus allow the name of the word to be used recursively in its own definition.
state	( -- addr )	Variable that is non-zero in compile state.

## Displaying Numbers

Table 25 shows basic commands to display stack values.

Table 25 Basic Number Display

Command	Stack Diagram	Description
.	( n -- )	Display a number in the current base.
.r	( n size -- )	Display a number in a fixed width field.
.s	( -- )	Display contents of data stack.
showstack	( ??? -- ??? )	Execute .s automatically before each ok prompt.
noshowstack	( ??? -- ??? )	Turn off automatic display of the stack before each ok prompt
u.	( u -- )	Display an unsigned number.
u.r	( u size -- )	Display an unsigned number in a fixed width field.

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

## Changing the Number Base

You can change the operating number base using the commands in Table 26.

Table 26 Changing the Number Base

Command	Stack Diagram	Description
.d	( n -- )	Display <i>n</i> in decimal without changing base.
.h	( n -- )	Display <i>n</i> in hex without changing base.
base	( -- addr )	Variable containing number base.
decimal	( -- )	Set the number base to 10.
d# <i>number</i>	( -- n )	Interpret <i>number</i> in decimal; base is unchanged.
hex	( -- )	Set the number base to 16.
h# <i>number</i>	( -- n )	Interpret <i>number</i> in hex; base is unchanged.
octal	( -- )	Set the number base to 16.
o# <i>number</i>	( -- n )	Interpret <i>number</i> in hex; base is unchanged.

The d# , h# and o# commands are useful when you want to input a specific number in another base without explicitly changing the current base. For example:

```
ok decimal      (Changes base to decimal)
ok 4 h# ff 17 2
4 255 17 2 ok
```

The .d and .h commands act like “.” but display the value in decimal or hexadecimal, respectively, regardless of the current base setting. For example:

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

*ff 255*

## Controlling Text Input and Output

This section describes text input and output commands. These commands control strings or character arrays, and allow you to enter comments and control keyboard scanning.

Table 27 lists commands to control text input.

Table 27 Controlling Text Input

Command	Stack Diagram	Description
( <i>ccc</i> )	( -- )	Begin a comment. Conventionally used for stack diagrams.
\ <i>rest-of-line</i>	( -- )	Treat the rest of the line as a comment.
ascii <i>ccc</i>	( -- char )	Get numerical value of first ASCII character of next word.
expect	( addr +n -- )	Get a line of edited input from the assigned input device's keyboard; store at <i>addr</i> .
key	( -- char )	Read a character from the assigned input device's keyboard.
key?	( -- flag )	True if a key has been typed on the input device's keyboard.
span	( -- waddr )	Variable containing the number of characters read by <i>expect</i> .
word	( char -- pstr )	Collect a string delimited by <i>char</i> from input string and place in memory at <i>pstr</i> .

Comments are used with Forth source code (generally in a text file) to describe the function of the code. The ( (open parenthesis) is the Forth word that begins a comment. Any character up to the closing parenthesis ) is ignored by the Forth interpreter. Stack diagrams are one example of comments using ( .

**Note** – Remember to follow the ( with a space, so that it is recognized as a Forth word.

\ (backslash) indicates a comment terminated by the end of the line of text.

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

ascii, used in the form `ascii x`, returns on the stack the numerical code of the character *x*.



`key?` looks at the keyboard to see if the user has recently pressed any key. It returns a flag on the stack: `true` if a key has been pressed and `false` otherwise. See “Conditional Flags” on page 73 for a discussion on the use of flags.

Table 28 lists general-purpose text display commands.

Table 28 Displaying Text Output

Command	Stack Diagram	Description
<code>. " ccc"</code>	( -- )	Compile a string for later display.
<code>( cr</code>	( -- )	Move the output cursor back to the beginning of the current line.
<code>cr</code>	( -- )	Terminate a line on the display and go to the next line.
<code>emit</code>	( char -- )	Display the character.
<code>exit?</code>	( -- flag )	Enable the scrolling control prompt: <code>More [&lt;space&gt;,&lt;cr&gt;,q] ?</code> The return flag is <code>true</code> if the user wants the output to be terminated.
<code>space</code>	( -- )	Display a space character.
<code>spaces</code>	( +n -- )	Display <code>+n</code> spaces.
<code>type</code>	( addr +n -- )	Display the <code>+n</code> characters beginning at <code>addr</code> .

`cr` sends a carriage-return character to the output. For example:

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

`emit` displays the letter whose ASCII value is on the stack.

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

Table 29 shows commands used to manipulate text strings.

Table 29 Manipulating Text Strings

Command	Stack Diagram	Description
" ,	( addr len -- )	Compile an array of bytes from <i>addr</i> of length <i>len</i> , at the top of the dictionary as a packed string.
" ccc"	( -- addr len )	Collect an input stream string, either interpreted or compiled. Within the string, "(00, ff...)" can be used to include arbitrary byte values.
.( ccc)	( -- )	Display a string immediately.
-trailing	( addr +n1 -- addr +n2 )	Remove trailing spaces.
bl	( -- char )	ASCII code for the space character; decimal 32.
count	( pstr -- addr +n )	Unpack a packed string.
lcc	( char -- lowercase-char )	Convert a character to lowercase.
left-parse-string	( addr len char -- addrR lenR addrL lenL )	Split a string at <i>char</i> (which is discarded).
pack	( addr len pstr -- pstr )	Make a packed string from <i>addr len</i> ; place it at <i>pstr</i> .
p" ccc"	( -- pstr )	Collect a string from the input stream; store as a packed string.
upc	( char -- uppercase-char )	Convert a character to uppercase.

Some string commands specify an address (the location in memory where the characters reside) and a length (the number of characters in the string). Other commands use a packed string or *pstr*, which is a location in memory containing a byte for the length, immediately followed by the characters. The stack diagram for the command indicates which form is used. For example, *count* converts a packed string to an address-length string.

The command `.` is used in the form: `. " string"`. It outputs text when needed. A `"` (double quotation mark) marks 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
```

## Redirecting Input and Output

Normally, your system uses a keyboard for all user input, and a frame buffer with a connected display screen for most display output. (Server systems may use an ASCII terminal connected to a system serial port. For more information on how to connect a terminal to the system unit, see your system’s installation manual.) You can redirect the input, the output, or both, to either one of the system’s serial ports. This may be useful, for example, when debugging a frame buffer.

Table 30 lists commands you can use to redirect input and output.

Table 30 I/O Redirection Commands

Command	Stack Diagram	Description
<code>input</code>	( device -- )	Select device ( <code>keyboard</code> , or <i>device-specifier</i> ) for input.
<code>io</code>	( device -- )	Select device for input and output.
<code>output</code>	( device -- )	Select device ( <code>screen</code> , or <i>device-specifier</i> ) for output.

The commands `input` and `output` temporarily change the current devices for input and output. The change occurs when you enter a command; you do not have to reset your system. A system reset or power cycle causes the input and output devices to revert to the default settings specified in the NVRAM configuration parameters `input-device` and `output-device`. These parameters can be modified, if needed (see Chapter 3, “Setting Configuration Parameters” for information about changing defaults).

`input` must be preceded by one of the following: `keyboard`, `ttya`, `ttyb`, or *device-specifier* text string. For example, if input is currently accepted from the keyboard, and you want 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 keyboard becomes non-functional (except for `Stop-A`), but any text entered from the terminal connected to TTYA is processed as input. All commands are executed as usual.

To resume using the keyboard as the input device, *use the terminal keyboard* to type:

```
ok keyboard input
ok
```

Similarly, `output` must be preceded by one of the following: `screen`, `ttya`, or `ttyb`. For example, if you want to send output to TTYA instead of the normal display screen, type:

```
ok ttya output
```

The screen does *not* show the answering `ok` prompt, but the terminal connected to TTYA shows the `ok` prompt and all further output as well.

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

Generally, `input`, `output`, and `io` take a *device-specifier*, which can be either a device path name or a device alias. *The device must be specified as a Forth string, using double quotation marks ("), as shown in the two examples below:*

```
ok " /sbus/cgsix" output
```

or:

```
ok " screen" output
```

In the preceding examples, `ttya`, `screen`, and `keyboard` are Forth words that put their corresponding device alias string on the stack.

## Command Line Editor

OpenBoot specifies a required command line editor (similar to EMACS, a common text editor), some optional extensions and an optional history mechanism for the User Interface. Use these powerful tools to re-execute previous commands without retyping them, to edit the current command line to fix typing errors, or to recall and change previous commands.

Table 31 lists the required line-editing commands available at the `ok` prompt.

*Table 31* Required Command Line Editor Keystroke Commands

Keystroke	Description
Delete	Erases previous character.
Backspace	Erases previous character.
Control-U	Erases the line.
Return (Enter)	Finishes editing of the line and submits the entire visible line to the interpreter regardless of the current cursor position.

The OpenBoot standard also describes three groups of extensions of these capabilities. Table 32 lists the command line editing extension group.

*Table 32* Optional Command Line Editor Keystroke Commands

Keystroke	Description
Control-B	Moves backward one character.
Escape B	Moves backward one word.
Control-F	Moves forward one character.
Escape F	Moves forward one word.
Control-A	Moves backward to beginning of line.

Table 32 Optional Command Line Editor Keystroke Commands (Continued)

Keystroke	Description
Control-E	Moves forward to end of line.
Delete	Erases previous character.
Backspace	Erases previous character.
Control-H	Erases previous character.
Escape H	Erases from beginning of word to just before the cursor, storing erased characters in a save buffer.
Control-W	Erases from beginning of word to just before the cursor, storing erased characters in a save buffer.
Control-D	Erases next character.
Escape D	Erases from cursor to end of the word, storing erased characters in a save buffer.
Control-K	Erases from cursor to end of line, storing erased characters in a save buffer.
Control-U	Erases entire line, storing erased characters in a save buffer.
Control-R	Retypes the line.
Control-Q	Quotes next character (allows you to insert control characters).
Control-Y	Inserts the contents of the save buffer before the cursor.

The command line history extension enables previously-typed commands to be saved in an EMACS-like command history ring that contains at least 8 entries. Commands may be recalled by moving either forward or backward around the ring. Once recalled, a command may be edited and/or re-submitted (by typing the Return key). The command line history extension keys are:

Table 33 Optional Command Line History Keystroke Commands

Keystroke	Description
Control-P	Selects and displays the previous command in the command history ring.
Control-N	Selects and displays the next command in the command history ring.
Control-L	Displays the entire command history ring.

The command completion extension enables the system to complete long Forth word names by searching the dictionary for one or more matches based upon the already-typed portion of a word. After you type in a portion of a word followed by the command completion keystroke, `Control-Space`, the system responds as follows:

- If the system finds exactly one matching word, the remainder of the word is automatically displayed.
- If the system finds several possible matches, it displays all characters common to all possibilities.
- If the system cannot find a match for the already-typed characters, it deletes characters from the right until there is at least one match for the remaining characters.
- The system beeps if it cannot determine an unambiguous match.

The command completion extension keys are:

*Table 34* Optional Command Completion Keystroke Commands

Keystroke	Description
Control-Space	Complete the name of the current word.
Control-?	Display all possible matches for the current word.
Control-/	Display all possible matches for the current word.

## *Conditional Flags*

Forth conditionals use flags to indicate true/false values. A flag can be generated in several ways, based on testing criteria. The flag can then be displayed from the stack with the word “.”, or it can be used as input to a conditional control command. Control commands can cause one response if a flag is true and another if it is false. Thus, execution can be altered based on the result of a test.

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

Table 35 lists commands that perform relational tests, and leave a `true` or `false` flag result on the stack.

Table 35 Comparison Commands

Command	Stack Diagram	Description
<code>&lt;</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1 < n2$ .
<code>&lt;=</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1 \leq n2$ .
<code>&lt;&gt;</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1$ is not equal to $n2$ .
<code>=</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1 = n2$ .
<code>&gt;</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1 > n2$ .
<code>&gt;=</code>	( <i>n1</i> <i>n2</i> -- flag )	True if $n1 \geq n2$ .
<code>0&lt;</code>	( <i>n</i> -- flag )	True if $n < 0$ .
<code>0&lt;=</code>	( <i>n</i> -- flag )	True if $n \leq 0$ .
<code>0&lt;&gt;</code>	( <i>n</i> -- flag )	True if $n <> 0$ .
<code>0=</code>	( <i>n</i> -- flag )	True if $n = 0$ (also inverts any flag).
<code>0&gt;</code>	( <i>n</i> -- flag )	True if $n > 0$ .
<code>0&gt;=</code>	( <i>n</i> -- flag )	True if $n \geq 0$ .
<code>between</code>	( <i>n</i> <i>min</i> <i>max</i> -- flag )	True if $min \leq n \leq max$ .
<code>false</code>	( -- 0 )	The value <code>FALSE</code> , which is 0.
<code>true</code>	( -- -1 )	The value <code>TRUE</code> , which is -1.
<code>u&lt;</code>	( <i>u1</i> <i>u2</i> -- flag )	True if $u1 < u2$ , unsigned.
<code>u&lt;=</code>	( <i>u1</i> <i>u2</i> -- flag )	True if $u1 \leq u2$ , unsigned.
<code>u&gt;</code>	( <i>u1</i> <i>u2</i> -- flag )	True if $u1 > u2$ , unsigned.
<code>u&gt;=</code>	( <i>u1</i> <i>u2</i> -- flag )	True if $u1 \geq u2$ , unsigned.
<code>within</code>	( <i>n</i> <i>min</i> <i>max</i> -- flag )	True if $min \leq n < max$ .

`>` takes two numbers from the stack, and returns `true` (-1) on the stack if the first number was greater than the second number, or returns `false` (0) otherwise. An example follows:

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



0= takes one item from the stack, and returns `true` if that item was 0 or returns `false` otherwise. This word inverts any flag to its opposite value.

## Control Commands

The following sections describe words used within a Forth program to control the flow of execution.

### *The if...else...then Structure*

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

The commands listed in Table 36 control the flow of conditional execution.

Table 36 `if...else...then` Commands

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

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` command consumes a flag from the stack. If the flag is `true` (non-zero), the commands following the `if` are performed. Otherwise, the commands (if any) following 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 to `ok` after you finish the definition with a semicolon.

### The case Statement

A high-level `case` command is provided for selecting alternatives with multiple possibilities. This command is easier to read than deeply-nested `if...then` commands.

Table 37 lists the conditional case commands.

Table 37 case Statement Commands

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

Here is a simple example of a case command:

```

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

```

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

## *The begin Loop*

A begin loop executes the same commands repeatedly until a certain condition is satisfied. Such a loop is also called a conditional loop.

Table 38 lists commands to control the execution of conditional loops.

*Table 38* begin (Conditional) Loop Commands

Command	Stack Diagram	Description
again	( -- )	End a begin...again infinite loop.
begin	( -- )	Begin a begin...while...repeat, begin...until, or begin...again loop.

Table 38 begin (Conditional) Loop Commands

Command	Stack Diagram	Description
repeat	( -- )	End a begin...while...repeat loop.
until	( flag -- )	Continue executing a begin...until loop until <i>flag</i> is true.
while	( flag -- )	Continue executing a begin...while...repeat loop while <i>flag</i> is true.

There are two general forms:

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

and

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

In both cases, the commands within the loop are executed repeatedly until the proper flag value causes the loop to be terminated. Then execution continues normally with the command following the closing command word (*until* or *repeat*).

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

An easy aid to memory for either of these loops is: If true, fall through.

Here 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, and `false` otherwise. This flag is consumed by `until` and, if the value is `false`, then the loop continues. Once a key is pressed, the next call to `key?` returns `true`, and the loop terminates.

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

## The `do` Loop

A `do` loop (also called a counted loop) is used when the number of iterations of the loop can be calculated in advance. A `do` loop normally exits just *before* the specified ending value is reached.

Table 39 lists commands to control the execution of counted loops.

Table 39 `do` (Counted) Loop Commands

Command	Stack Diagram	Description
<code>+loop</code>	( <code>n --</code> )	End a <code>do...+loop</code> construct; add <code>n</code> to loop index and return to <code>do</code> (if <code>n &lt; 0</code> , index goes from <i>start</i> to <i>end</i> inclusive).
<code>?do</code>	( <code>end start --</code> )	Begin <code>?do...loop</code> to be executed 0 or more times. Index goes from <i>start</i> to <i>end-1</i> inclusive. If <code>end = start</code> , loop is not executed.
<code>?leave</code>	( <code>flag --</code> )	Exit from a <code>do...loop</code> if <i>flag</i> is non-zero.
<code>do</code>	( <code>end start --</code> )	Begin a <code>do...loop</code> . Index goes from <i>start</i> to <i>end-1</i> inclusive. Example: <code>10 0 do i . loop</code> (prints 0 1 2...d e f).
<code>i</code>	( <code>-- n</code> )	Leaves the loop index on the stack.
<code>j</code>	( <code>-- n</code> )	Leaves the loop index of the next outer enclosing loop on the stack.
<code>leave</code>	( <code>--</code> )	Exit from <code>do...loop</code> .
<code>loop</code>	( <code>--</code> )	End of <code>do...loop</code> .

This screen shows several examples of the ways in which loops are used.

```

ok 10 5 do i . loop
5 6 7 8 9 a b c d e f
ok
ok 2000 1000 do i . i c@ . cr i c@ ff = if leave then 4 +loop
1000 23
1004 0
1008 fe
100c 0
1010 78
1014 ff
ok : scan ( byte -- )
] 6000 5000 ( Scan memory 5000 - 6000 for bytes not equal to the specified byte )
] 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
5005 5224 5f99
ok 6000 5000 do i i c! loop ( Fill a region of memory with a stepped pattern )
ok
ok 500 value testloc
ok : test16 ( -- ) 1.0000 0 ( do 0-ffff ) ( Write different 16-bit values to a location )
] do i testloc w! testloc w@ i <> ( error? ) ( Also check the location )
] if ." Error - wrote " i . ." read " testloc w@ . cr
] leave ( exit after first error found ) ( This line is optional )
] then
] loop
] ;
ok test16
ok 6000 to testloc
ok test16
Error - wrote 200 read 300
ok

```

## Additional Control Commands

Table 40 contains descriptions of additional program execution control commands.

Table 40 Program Execution Control Commands

Command	Stack Diagram	Description
<code>abort</code>	( -- )	Abort current execution and interpret keyboard commands.
<code>abort "</code> <code>ccc"</code>	( abort? -- )	If <code>abort?</code> is true, abort and display message.
<code>eval</code>	( addr len -- )	Interpret Forth source from an array.
<code>execute</code>	( xt -- )	Execute the word whose execution token is on the stack.
<code>exit</code>	( -- )	Return from the current word. (Cannot be used in counted loops.)
<code>quit</code>	( -- )	Same as <code>abort</code> , but leave stack intact.

`abort` causes immediate termination and returns control to the keyboard. `abort "` is similar to `abort` but is different in two respects. `abort "` removes a flag from the stack and only aborts if the flag is `true`. Also, `abort "` prints any desired message when the abort takes place.

`eval` takes a string from the stack (specified as an address and a length). The characters in that string are then interpreted as if they were entered from the keyboard. If a Forth text file has been loaded into memory (see Chapter 5, "Loading and Executing Programs"), then `eval` can be used to compile the definitions contained in the file.





## Loading and Executing Programs



The User Interface provides several methods for loading and executing a program. These methods load a file into memory from Ethernet, a hard disk, a floppy disk, and serial port A, and support the execution of Forth, FCode and binary executable programs.

Table 41 lists commands for loading files from various sources.

Table 41 File Loading Commands

Command	Stack Diagram	Description
?go	( -- )	Execute Forth, FCode or binary programs.
boot [ <i>specifiers</i> ] -h	( -- )	Load file from specified source.
byte-load	( addr span -- )	Interpret loaded FCode binary file. <i>span</i> is usually 1.
d1	( -- )	Load a Forth file over a serial line with a terminal emulator and interpret. Using <code>tip</code> as an example, type: ~C cat <i>filename</i> ^-D
dlbin	( -- )	Load a binary file over a serial line with a terminal emulator. Using <code>tip</code> as an example, type: ~C cat <i>filename</i>
dload <i>filename</i>	( addr -- )	Load the specified file over Ethernet at the given address.
eval	( addr len -- )	Interpret loaded Forth text file.
go	( -- )	Begin executing a previously-loaded binary program, or resume executing an interrupted program.

Table 41 File Loading Commands (Continued)

Command	Stack Diagram	Description
init-program	( -- )	Initialize to execute a binary file.
load <i>device-specifier argument</i>	( -- )	Load data from specified device into memory at the address given by load-base.
load-base	( -- addr )	Address at which load places the data it reads from a device.

## Using *dload* to Load from Ethernet

*dload* loads files over Ethernet at a specified address, as shown below.

```
ok 4000 dload filename
```

In the example above, *filename* must be relative to the server's root. Use 4000 (hex) as the address for *dload* input. *dload* uses the trivial file transfer protocol (TFTP), so the server may need to have its permissions adjusted for this to work.

## Forth Programs

Forth programs loaded with *dload* must be ASCII files beginning with the two characters “\ ” (backslash and blank). To execute the loaded Forth program, type:

```
ok 4000 file-size @ eval
```

In the above example, *file-size* contains the size of the loaded image.

## FCode Programs

FCode programs loaded with *dload* must be *a.out* files. To execute the loaded FCode program, type:

```
ok 4000 1 byte-load
```

`byte-load` is used by OpenBoot to interpret FCode programs on expansion boards such as SBus. The `1` in the example is a specific value of a parameter that specifies the separation between FCode bytes in the general case. Since `dload` loads into system memory, `1` is the correct spacing.

### *Binary Executables*

Executable binary programs loaded with `dload` are `a.out` files and must be linked to run `dload`'s input address (4000) or be position independent. To execute the binary program, type:

```
ok go
```

To run the program again, type:

```
ok init-program go
```

`dload` does not use intermediate booters (unlike the `boot` command). Thus, any symbol information included in the `a.out` file is available to the User Interface's symbolic debugging capability. (See Chapter 6, "Debugging" for more information on symbolic debugging.)

### *Using boot to Load from Hard Disk, Floppy Disk, or Ethernet*

You can also load and execute a program with `boot`, the command normally used to boot the operating system. `boot` has the following format:

```
ok boot [ device-specifier ] [ filename ] -h
```

*device-specifier* is either a full device path name or a device alias. (See Chapter 1, "Overview" for information on device path names and aliases.)

For a hard disk or floppy partition, *filename* is relative to the resident file system. (See Appendix B, "Building A Bootable Floppy Disk", for information on creating a bootable floppy disk.) For Ethernet, *filename* is relative to the system's root partition on its root server. In both cases, the leading `/` must be omitted from the file path.

The `-h` flag specifies that the program should be loaded, but not executed.

`boot` uses intermediate booters to accomplish its task. When loading from a hard disk or floppy disk, OpenBoot first loads the disk's boot block, which in turn loads a second-level booter. When loading over Ethernet, the firmware uses TFTP to load the second-level booter. *filename* and `-h` are passed to these intermediate booters.

### *Forth Programs*

Forth programs are ASCII source files that must be converted to the file format required by the secondary boot program. A utility called `fakeboot` is available from the SBUS Support Group at Sun to perform this conversion. After the file is loaded into memory, it can be executed using the command `eval`.

For instance, if the file is loaded to address `0x4010`, and runs for 934 bytes, type:

```
ok 4010 d# 934 eval
```

### *FCode Programs*

FCode programs produced by a Tokenizer (which creates FCode programs) may need to be converted to the file format of the secondary boot program. `fakeboot` may be useful in this process. Once the file is in memory, execute it using the `byte-load` command.

For example, assuming the file is loaded to address `0x4030`, type:

```
ok 4030 1 byte-load
```

## Binary Executables

A binary program other than the operating system can also be loaded and executed as follows:

```
ok go
```

`go` is needed since the `boot` command includes `-h`.

## Using `d1` to Load Forth Over a Serial Port

Forth programs loaded with `d1` must be ASCII files.

To load the file over the serial line, connect the system-under-test's serial port to a machine that is able to transfer a file on request, and start a terminal emulator on that system. The terminal emulator is then used to download the file using `d1`.

The following example assumes the use of the Unix terminal emulator `tip`. (See Appendix A, "Testing with a Terminal Emulator", for information on this procedure.)

**1. At the `ok` prompt, type:**

```
ok d1
```

**2. In the `tip` window of the other system, send the file, and follow it with a `Control-D` to signal the end of the file.**

```
~C (local command) cat filename  
(Away two seconds)  
^D
```

The file is automatically interpreted after it is loaded, and the `ok` prompt reappears on the screen of the system to which the file was loaded.

## *Using `dlbin` to Load FCode or Binary Over a Serial Port*

FCode and binary programs loaded with `dlbin` must be `a.out` files. `dlbin` loads the files at the entry point indicated in the `a.out` header. Link binary files for 4000 (hex). Recent versions of the FCode Tokenizer create an `a.out` file with entry point 4000.

To load the file over the serial line, connect the system's serial port A to a machine that is able to transfer a file on request. The following example assumes a `tip` window setup. (See Appendix A, "Testing with a Terminal Emulator", for information on this procedure.)

**1. At the `ok` prompt, type:**

```
ok dlbin
```

**2. In the TIP window of the other system, send the file:**

```
~C (local command) cat filename  
(Away two seconds)
```

The `ok` prompt appears on the screen of the system to which the file is loaded.

To execute an FCode program, type:

```
ok 4000 1 byte-load  
ok
```

To execute a binary program, type:

```
ok go
```

OpenBoot provides debugging tools that include a disassembler, register display commands, and breakpoint commands.

## Using the Disassembler

The built-in disassembler translates the contents of memory into equivalent SPARC assembly language.

Table 42 lists commands that disassemble memory into equivalent op codes.

Table 42 Disassembler Commands

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

`dis` begins to disassemble the data content of any desired location. The system pauses when:

- Any key is pressed while disassembly is taking place.
- The disassembler output fills the display screen.
- A `call` or `jump` op code is encountered.

Disassembly can then be stopped or the `+dis` command can be used to continue disassembling at the location where the last disassembly stopped.

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

## Displaying Registers

You can enter the User Interface from the middle of an executing program as a result of a program crash, a user abort with `stop-A`, or an encountered breakpoint. (Breakpoints are discussed on page 91.) In all these cases, the User Interface automatically saves all the CPU data register values in a buffer area. You can then inspect or alter these values for debugging purposes.

Table 43 lists the SPARC register commands.

Table 43 SPARC Register Commands

Command	Stack Diagram	Description
<code>%f0 through %f31</code>	( -- value )	Return the value in the specified floating point register.
<code>%fsr</code>	( -- value )	Return the value in the floating point status register.
<code>%g0 through %g7</code>	( -- value )	Return the value in the specified global register.
<code>%i0 through %i7</code>	( -- value )	Return the value in the specified input register.
<code>%l0 through %l7</code>	( -- value )	Return the value in the specified local register.
<code>%o0 through %o7</code>	( -- value )	Return the value in the specified output register.
<code>%pc %npc %psr %y %wim %tbr</code>	( -- value )	Return the value in the specified register.
<code>.fregisters</code>	( -- )	Display the values in <code>%f0 through %f31</code> .
<code>.locals</code>	( -- )	Display the values in the <code>i</code> , <code>l</code> and <code>o</code> registers.
<code>.psr</code>	( -- )	Formatted display of the program status register.
<code>.registers</code>	( -- )	Display values in <code>%g0 through %g7</code> , plus <code>%pc</code> , <code>%npc</code> , <code>%psr</code> , <code>%y</code> , <code>%wim</code> , <code>%tbr</code> .
<code>.window</code>	( window# -- )	Same as <code>w .locals</code> ; display the desired window.
<code>ctrace</code>	( -- )	Display the return stack showing C subroutines.
<code>set-pc</code>	( new-value -- )	Set <code>%pc</code> to <i>new-value</i> , and set <code>%npc</code> to ( <i>new-value</i> +4).
<code>to regname</code>	( new-value -- )	Change the value stored in any of the registers above. Use in the form: <i>new-value to regname</i> .
<code>w</code>	( window# -- )	Set the current window for displaying <code>%ix</code> , <code>%Lx</code> , or <code>%ox</code> .



After the values have been inspected and/or modified, program execution can be continued with the `go` command. The saved (and possibly modified) register values are copied back into the CPU, and execution resumes at the location specified by the saved program counter.

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

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

## Breakpoints

The User Interface provides a breakpoint capability to assist in the development and debugging of stand-alone programs. (Programs that run under the operating system generally do not use this feature, but use other debuggers designed to run under the operating system.) The breakpoint feature lets you stop the test program at desired points. After program execution has stopped, registers or memory can be inspected or changed, and new breakpoints can be set or cleared. You can resume program execution with the `go` command.

Table 44 lists the breakpoint commands that control and monitor program execution.

Table 44 Breakpoint Commands

Command	Stack Diagram	Description
<code>+bp</code>	( addr -- )	Add a breakpoint at the specified address.
<code>-bp</code>	( addr -- )	Remove the breakpoint at the specified address.
<code>--bp</code>	( -- )	Remove the most-recently-set breakpoint.
<code>.bp</code>	( -- )	Display all currently set breakpoints.

Table 44 Breakpoint Commands (Continued)

Command	Stack Diagram	Description
<code>.breakpoint</code>	( -- )	Perform a specified action when a breakpoint occurs. This word can be altered to perform any desired action. For example, to display registers at every breakpoint, type: ['] <code>.registers is .breakpoint</code> . The default behavior is <code>.instruction</code> . To perform multiple behaviors, create a single definition which calls all desired behaviors, then load that word into <code>.breakpoint</code> .
<code>.instruction</code>	( -- )	Display the address, opcode for the last-encountered breakpoint.
<code>.step</code>	( -- )	Perform a specified action when a single step occurs (see <code>.breakpoint</code> ).
<code>bpoff</code>	( -- )	Remove all breakpoints.
<code>finish-loop</code>	( -- )	Execute until the end of this loop.
<code>go</code>	( -- )	Continue from a breakpoint. This can be used to go to an arbitrary address by setting up the processor's program counter before issuing <code>go</code> .
<code>gos</code>	( n -- )	Execute <code>go</code> <i>n</i> times.
<code>hop</code>	( -- )	(Like the <code>step</code> command.) Treat a subroutine call as a single instruction.
<code>hops</code>	( n -- )	Execute <code>hop</code> <i>n</i> times.
<code>return</code>	( -- )	Execute until the end of this subroutine.
<code>returnL</code>	( -- )	Execute until the end of this leaf subroutine.
<code>skip</code>	( -- )	Skip (do not execute) the current instruction.
<code>step</code>	( -- )	Single-step one instruction.
<code>steps</code>	( n -- )	Execute <code>step</code> <i>n</i> times.
<code>till</code>	( addr -- )	Execute until the given address is encountered. Equivalent to <code>+bp go .</code>

To debug a program using breakpoints, use the following procedure.

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

See Chapter 5, "Loading and Executing Programs" for more information. Using `dload` is generally best, since the symbol table for the program is preserved. `boot -h` also works if the program is not available over Ethernet.

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

**2. (Optional) Disassemble the downloaded program to verify a properly-loaded file.**

### 3. Begin single-stepping the test program using the `step` command.

You can also set a breakpoint, then execute (for example, using the commands `4020 +bp` and `go`) or perform other variations.

## The Forth Source-level Debugger

The Forth Source-level Debugger allows single-stepping and tracing of Forth programs. Each step represents the execution of one Forth word.

The debugger commands are shown in Table 45.

Table 45 Forth Source-level Debugger Commands

Command	Description
<code>c</code>	“Continue”. Switch from stepping to tracing, thus tracing the remainder of the execution of the word being debugged.
<code>d</code>	“Down a level”. Mark for debugging the word whose name was just displayed, then execute it.
<code>f</code>	Start a subordinate Forth interpreter. When that interpreter exits (with <code>resume</code> ), control returns to the debugger at the place where the <code>F</code> command was executed.
<code>q</code>	“Quit”. Abort the execution of the word being debugged and all its callers and return to the command interpreter.
<code>u</code>	“Up a level”. Un-mark the word being debugged, mark its caller for debugging, and finish executing the word that was previously being debugged.
<code>debug name</code>	Mark the specified Forth word for debugging. Enter the Forth Source-level Debugger on all subsequent attempts to execute <i>name</i> . After executing <code>debug</code> , the execution speed of the system may decrease until debugging is turned off with <code>debug-off</code> . (Do not debug basic Forth words such as “.”.)
<code>debug-off</code>	Turn off the Forth Source-level Debugger so that no word is being debugged.
<code>resume</code>	Exit from a subordinate interpreter, and go back to the stepper (see the <code>F</code> command in this table).
<code>stepping</code>	Set “step mode” for the Forth Source-level Debugger, allowing the interactive, step-by-step execution of the word being debugged. Step mode is the default.
<code>tracing</code>	Set “trace mode” for the Forth Source-level Debugger. This traces the execution of the word being debugged, while showing the name and stack contents for each word called by that word.
<code>&lt;space-bar&gt;</code>	Execute the word just displayed and proceed to the next word.

Every Forth word is defined as a series of one or more words that could be called “component” words. While debugging a specified word, the debugger displays information about the contents of the stack while executing each of the word’s “component” words. Immediately before executing each component word, the debugger displays the contents of the stack and the name of the component word that is about to be executed.

In trace mode, that component word is then executed, and the process continues with the next component word.

In step mode (the default), the user controls the debugger’s execution response. Before the execution of each component word, the user is prompted for one of the keystrokes specified in Table 45.

## *Using ftrace*

The `ftrace` command shows the sequence of Forth words that were being executed at the time of the last exception. An example of `ftrace` follows.

```
ok : test1 1 ! ;
ok : test2 1 test1 ;
ok test2
Memory address not aligned
ok ftrace
!      Called from test1 at ffeacc5c
test1  Called from test2 at ffeacc6a
(ffeb8b574) Called from (interpret at ffe8b6f8
execute  Called from catch at ffe8a8ba
    ffeff0
    0
    ffeebdc
catch    Called from (fload) at ffe8ced8
    0
(fload)  Called from interact at ffe8cf74
execute  Called from catch at ffe8a8ba
    ffeffd4
    0
    ffeebdc
catch   Called from (quit at ffe8cf98
```

---

In this example, `test2` calls `test1`, which tries to store a value to an unaligned address. This results in the exception: `Memory address not aligned`.

The first line of `ftrace`'s output shows the last command that caused the exception to occur. The next lines show locations from which the subsequent commands were being called.

The last thirteen lines are usually the same in any `ftrace` output, because that is the calling sequence in effect when the Forth interpreter interprets a word from the input stream.



## *Testing with a Terminal Emulator*

---



You can use the serial port(s) on the system that you are testing to connect to a second system which will act as a file server. This file server may or may not be the same type of system provided that:

- The capabilities of the file server's serial port are compatible with the system being tested.
- The file server has a terminal emulator that is capable of correctly setting the file server's output baud rate to match that of the system that you are testing.

By connecting two systems in this way, you can use the terminal emulator on the file server as a terminal into the system that you are testing. (For UNIX systems, see the on-line `tip` manpage for detailed information about terminal connection to a remote host. For Windows systems, see the documentation for the Terminal accessory. For Macintosh® systems, see the documentation for MacTerminal®.)

This terminal emulation method is recommended (over simply connecting to a dumb terminal), since it lets you use your normal editor and operating system features when working with the boot ROM.

---

**Note** – In the following pages, *system* refers to the system that you are testing, and *server* refers to the file server system that you are connecting to the system being tested.

---

The procedures given in this chapter assume the use of the UNIX `tip` terminal emulator. Other terminal emulators will use similar procedures.

**1. Connect a serial port from the server to a serial port on your system with a 3-wire "null modem" cable (i.e. a cable that connects Pin 3 to Pin 2, Pin 2 to Pin 3, and Pin 7 to Pin 7). For the following examples, we will assume the use of Port A on the system and Port B on the server.**

**2. To set up the `tip` session on the server, type:**

```
hostname% tip -9600 /dev/ttyb
connected
```

---

**Note** – On Sun workstations, use a Shell Tool window, not a Command Tool window; some `tip` commands may not work properly in a Command Tool window.

---

**3. At your system, enter the User Interface so that the `ok` prompt is displayed.**

If you do not have a video monitor attached to your system, connect the system's TTYA to the server's TTYB and turn on the power to your system. Wait for a few seconds, and press `stop-A` to interrupt the power-on sequence and start the User Interface. Type `n` to get to the `ok` prompt. Unless the system is completely inoperable, the User Interface is enabled, and you can continue with the next step in this procedure.

**4. If you need to redirect the standard input and output to TTYA, type:**

```
ok ttya io
```

There will be no echoed response.

**5. Press Return on the Sun workstation keyboard. The `ok` prompt appears in the TIP window.**

Typing `~#` in the TIP window is equivalent to typing `stop-A` at the SPARC system.



---

**Note** – *Do not* type `Stop-A` from a Sun workstation being used as a server to your system. Doing so will abort the operating system on the server. (If you accidentally type `Stop-A`, you can recover by immediately typing either `c` at the `>` prompt or `go` at the `ok` prompt.)

---

6. When you are finished using the `tip` window, end your `tip` session and exit the window:
  - a. Redirect the input and output to the screen and keyboard, if needed.
  - b. In the `tip` window, type:

```
ok ~.  
hostname%
```

---

**Note** – When entering `~` (tilde) commands in the `tip` window, `~` must be the first character entered on the line. To ensure that you are at the start of a new line, press Return first.

---

## Common Problems with `tip`

This section describes solutions for `tip` problems occurring in pre-Solaris 2.0 operating environments.

Problems with `tip` may occur if:

- The lock directory is missing or incorrect.

There should be a directory named `/usr/spool/uucp`. The owner should be `uucp` and the mode should be `drwxr-sr-x`.

- TTYB is 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.

- `/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-`.

- The serial line is in tandem mode.

If the `tip` connection is in tandem mode, the operating system sometimes sends XON (^S) characters (particularly when programs in other windows are generating lots of output). The XON characters are detected by the Forth word `key?`, and can cause confusion. The solution is to turn off tandem mode with the `~s !tandem tip` command.

- The `.cshrc` file generates text.

`tip` opens a sub-shell to run `cat`, thus causing text to be attached to the beginning of your loaded file. If you use `dl` and see any unexpected output, check your `.cshrc` file.

## *Building A Bootable Floppy Disk*

---



The instructions in this appendix show how to build a floppy diskette from which you can boot programs. You should use a high density (HD, not DD) diskette. Two sets of instructions are provided:

- Page 101 describes the procedure for systems using pre-Solaris 2.0 operating environments.
- Page 102 describes the procedure for systems using the Solaris 2.0 or 2.1 operating environments.

### *Procedure for the Pre-Solaris 2.0 Operating Environment*

Use the following procedure if you are using a pre-Solaris 2.0 version of the operating system.

#### **1. Format the diskette.**

```
hostname# fdformat
```

#### **2. Create the diskette's file systems.**

```
hostname# /usr/etc/newfs /dev/rfd0a
```

**3. Mount the diskette.**

```
hostname# mount /dev/fd0a /mnt
```

**4. Copy the second-level disk booter to the diskette.**

```
hostname# cp /boot /mnt
```

**5. Install a boot block on the floppy.**

```
hostname# /usr/mdec/installboot /mnt/boot /usr/mdec/bootfd /dev/rfd0a
```

**6. Copy the file you want to boot to /mnt.**

**7. Unmount the diskette and remove it from the drive.**

```
hostname# umount /mnt  
hostname# eject floppy
```

## *Procedure for the Solaris 2.0 or 2.1 Operating Environment*

Use the following procedure if you are using the Solaris 2.0 or 2.1 operating environment.

**1. Format the diskette.**

```
hostname# fdformat
```

**2. Create the diskette's file systems.**

```
hostname# /usr/sbin/newfs /dev/rdiskette
```

---

**3. Mount the diskette.**

```
hostname# mount /dev/diskette /mnt
```

**4. Copy the second-level disk booter to the diskette.**

```
hostname# cp /ufsboot /mnt
```

**5. Install a boot block on the floppy.**

```
hostname# /usr/sbin/installboot /usr/lib/fs/ufs/bootblk /dev/rdiskette
```

**6. Copy the file you want to boot to /mnt.**

**7. Unmount the diskette and remove it from the drive.**

```
hostname# umount /mnt  
hostname# eject floppy
```



## Unsupported Commands



Some features of the OpenBoot firmware may not be available in early systems. If you want to use a documented command that is not available in your system, refer to this appendix for a possible workaround.

Table C-1 Workarounds for Unsupported Commands

Command	Availability	Workaround
" <i>embedded bytes</i>	Not supported in earlier systems.	Use other array-creation mechanisms, such as <code>alloc-mem</code> and <code>c, .</code>
<code>.attributes</code>	Not supported until OpenBoot 2.0.	A loadable <code>showdevs</code> utility, which provides some of this functionality, is available from the Sun SBus Support Group.
<code>alloc-mem</code>	See workaround.	Pre-2.0, size is restricted to total remaining FORTH dictionary space. Using more than several hundred bytes is dangerous. Use <code>dma-alloc ( size -- virt )</code> instead.
<code>boot-device</code> <code>boot-file</code>	Not supported until OpenBoot 2.0.	Use <code>boot-from</code> to indicate boot device and boot file.
<code>cd</code>	Not supported until OpenBoot 2.0.	A loadable <code>showdevs</code> utility, which provides some of this functionality, is available from the Sun SBus Support Group.
Command completion	Not supported in early systems.	Type the entire command name.
<code>cpeek</code> <code>cpoke</code>	Not supported in early systems.	<code>probe</code> words exist in early systems to provide a similar functionality, as: <code>cprobe ( adr -- ok? )</code> Test for data exception using <code>c@</code> .
<code>d!</code> <code>d?</code> <code>d@</code>	Not supported in early systems.	Use combinations of 32-bit accesses.



Table C-1 Workarounds for Unsupported Commands (Continued)

Command	Availability	Workaround
diag-device diag-file	Not supported until OpenBoot 2.0.	Use <code>boot-from-diag</code> to indicate diagnostic boot device and boot file.
lpeek lpoke	Not supported in early systems.	<code>probe</code> words exist in early systems to provide a similar functionality, as: <code>lprobe ( adr32 -- ok? )</code> Test for data exception using <code>l@</code> .
ls	Not supported until OpenBoot 2.0.	A loadable <code>showdevs</code> utility, which provides some of this functionality, is available from the Sun SBus Support Group.
NVRAMRC	Not supported until OpenBoot 2.0.	No workaround. A different version exists in OpenBoot 1.6; do not use this version.
nvalias nvunalias	Not supported until OpenBoot 2.6.	Manually edit NVRAMRC.
nodefult-bytes	Not supported until OpenBoot 2.0.	No workaround.
patch	See workaround.	Pre-2.6, <code>patch</code> would patch words but not numbers within definitions. To patch numbers, use : <code>npatch word-to-patch ( new-n old-n -- )</code> .
probe-scsi-all	Not supported until OpenBoot 2.6.	No workaround.
pwd	Not supported until OpenBoot 2.0.	A loadable <code>showdevs</code> utility, which provides some of this functionality, is available from the Sun SBus Support Group.
show-devs	Not supported until OpenBoot 2.0.	A loadable <code>showdevs</code> utility, which provides some of this functionality, is available from the Sun SBus Support Group.
show-sbus	Not supported until OpenBoot 2.3.	Use: <code>ok cd /sbus</code> <code>ok ls</code> (Similar information is presented, but in a different format.)
showstack	Does not toggle (turn off) until OpenBoot 2.6.	To turn off <code>showstack</code> , either reset the system or type: <code>['] noop</code> is status
spaced?	Not supported until OpenBoot 2.6.	Use <code>spaced@</code> and <code>“ . ”</code>
Stop-F Stop-D Stop-N	Not supported until OpenBoot 2.0.	No workaround.



*Table C-1 Workarounds for Unsupported Commands (Continued)*

<b>Command</b>	<b>Availability</b>	<b>Workaround</b>
<code>test xxx</code>	Not supported until OpenBoot 2.0.	It is possible to test certain devices on OpenBoot 1.x systems with: <code>test-memory (--)</code> (similar to: <code>test /memory</code> ). Some plug-in devices can also be tested by directly entering the appropriate test name (on OpenBoot 1.x only).
User-added device aliases	Not supported until OpenBoot 2.0.	No workaround.
<code>watch-net</code>	Not supported in OpenBoot 1.3 through 2.2.	No workaround.
<code>wpeek</code> <code>wpoke</code>	Not supported in early systems.	<code>probe</code> words exist in early systems to provide a similar functionality, as: <code>wprobe ( adr16 -- ok? )</code> Test for data exception using <code>w@</code> .



## *Troubleshooting Guide*

---



What do you do if your system fails to boot properly? This appendix discusses some common failures and ways to alleviate them.

### *Power-on Initialization Sequence*

Familiarize yourself with the system power-on initialization messages. You can then identify problems more accurately because these messages show you the types of functions the system performs at various stages of system start-up. They also show the transfer of control from POST to the OpenBoot firmware to the Booter to the kernel.

The example that follows shows the OpenBoot initialization sequence in a SPARCstation 10 system. The messages before the banner appear on TTYA only if the `diag-switch?` parameter is true.

---

**Note** – The displayed kernel messages may vary depending on the version of the operating system you are using.

---

```
...

ttya initialized                               (At this point, POST has finished execution
                                              and transferred control to the OpenBoot firmware)

Cpu #0 TI,TMS390Z50                            (Probe CPU module)
Cpu #1 Nothing there
Cpu #2 Nothing there
Cpu #3 Nothing there
Probing Memory Bank #0 16 Megabytes of DRAM    (Probe memory)
Probing Memory Bank #1 Nothing there
Probing Memory Bank #2 Nothing there
Probing Memory Bank #3 Nothing there
Probing Memory Bank #4 Nothing there
Probing Memory Bank #5 Nothing there
Probing Memory Bank #6 Nothing there
Probing Memory Bank #7 Nothing there

                                              (Before probing the devices, the firmware executes
                                              NVRAMRC commands - if use-nvramrc? is true -
                                              and checks for Stop-x commands
                                              Keyboard LEDs flash)

Probing /iommuf,e0000000/sbus@f,e0001000 at f,0 (Probe devices)
  espdma esp sd st ledma le SUNW,bpp SUNW,DBRIa
Probing /iommuf,e0000000/sbus@f,e0001000 at 0,0
  Nothing there
Probing /iommuf,e0000000/sbus@f,e0001000 at 1,0
  Nothing there
Probing /iommuf,e0000000/sbus@f,e0001000 at 2,0
  Nothing there
Probing /iommuf,e0000000/sbus@f,e0001000 at 3,0
  Nothing there

SPARCstation 10 (1 X 390Z50), Keyboard Present (Display banner)
ROM Rev. 2.10, 16 MB memory installed, Serial #4194577.
Ethernet address 8:0:20:10:61:b5, Host ID: 72400111.

Boot device: /iommuf,sbus/espdma@f,400000/esp@f,800000/ (The firmware is TFTP-ing in the boot program)
sd@3,0 File and args: (Control is transferred to Booter after
this message is displayed)

root on /iommuf,e0000000/sbus@f,e0001000/espdma@ (Booter starts executing)
f,400000/esp@f,800000/sd@3,0:a fstype 4.2

Boot: vmunix
Size: 1425408+436752+176288 bytes (Control is passed to the Kernel after
this message is displayed)

Viking/NE: PAC ENABLED (Kernel starts to execute)
... (More kernel messages)
```

## Emergency Procedures

Table D-1 describes commands that are useful in some failure situations. When issuing any of these commands, hold down the keys immediately after turning on the power to your system, until the keyboard LEDs flash.

Table D-1 Emergency Keyboard Commands

Command	Description
Stop	Bypass POST. This command does not depend on security-mode. (Note: some systems bypass POST as a default; in such cases, use <code>Stop-D</code> to start POST.)
Stop-A	Abort.
Stop-D	Enter diagnostic mode (set <code>diag-switch?</code> to <code>true</code> ).
Stop-F	Enter FORTH on TTYA instead of probing. Use <code>fexit</code> to continue with the initialization sequence. Useful if hardware is broken.
Stop-N	Reset NVRAM contents to default values.

**Note** – These commands are disabled if the PROM security is on. Also, if your system has `full` security enabled, you cannot apply any of the suggested commands unless you have the password to get to the `ok` prompt.

## 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 operating system has crashed, or has been interrupted without preserving all data first.

`sync` actually returns control to the operating system, which then performs the data saving operations. After the disk data has been synchronized, the operating system begins to save a core image of itself. If you do not need this core dump, you can interrupt the operation with the `Stop-A` key sequence.

## Common Failures

This section describes some common failures and how you can fix them.

## *Blank Screen - No Output*

**Problem:** Your system screen is blank and does not show any output.

Here are possible causes for this problem:

- Hardware has failed.  
Refer to your system documentation.
- Keyboard is not attached.  
If the keyboard is not plugged in, the output goes to TTYA instead. To fix this problem, power the system down, plug the keyboard in, and power on again.
- Monitor is not turned on or plugged in.  
Check the power cable on the monitor. Make sure the monitor cable is plugged into the system frame buffer; then turn the monitor on.
- `output-device` is set to TTYA or TTYB.  
This means the NVRAM parameter `output-device` is set to `ttya` or `ttyb` instead of being set to `screen`. You can do one of the following:
  - Power the system down. Then turn it on, and immediately press `Stop-N`. This sets all NVRAM parameters to their default values. As a result, the `output-device` parameter is set to `screen`. Be warned that all previous non-default settings are reset to their default values as well. You must restore them as needed.
  - Connect a terminal to TTYA and reset the system. After getting to the `ok` prompt on the terminal, type: `screen output` to send output to the frame buffer. Use `setenv` to change the default display device, if needed.
- System has multiple frame buffers.  
If your system has several plugged-in frame buffers, or it has one built-in frame buffer and one or more plugged in, then it is possible that the wrong frame buffer is being used as the console device. See “Setting the Console to a Specific Monitor” on page 115.

## *System Boots From the Wrong Device*

**Problem:** Your system is supposed to boot from the disk; instead, it boots from the net.

There are two possible causes for this:

- The `diag-switch?` NVRAM parameter is inadvertently set to `true`. Interrupt the booting process with `Stop-A`. Type the following commands at the `ok` prompt:

```
ok setenv diag-switch? false
ok boot
```

The system should now start booting from the disk.

- The `boot-device` NVRAM parameter is set to `net` instead of `disk`. Interrupt the booting process with `Stop-A`. Type the following commands at the `ok` prompt:

```
ok setenv boot-device disk
ok boot
```

Note that the preceding commands cause the system to boot from the disk defined as `disk` (target 3) in the device aliases list. If you want to boot from `disk1` (target 1), `disk2` (target 2), or `disk3` (target 3), set `boot-device` accordingly.

**Problem:** Your system is booting from a disk instead of from the net.

- `boot-device` is not set to `net`. Interrupt the booting process with `Stop-A`. Type the following commands at the `ok` prompt:

```
ok setenv boot-device net
ok boot
```

**Problem:** Your system is booting from the wrong disk. (For example, you have more than one disk in your system. You want the system to boot from `disk2`, but the system is booting from `disk1` instead.)

- `boot-device` is not set to the correct disk.  
Interrupt the booting process with `Stop-A`. Type the following commands at the `ok` prompt:

```
ok setenv boot-device disk2
ok boot
```

### *System Will Not Boot From Ethernet*

**Problem:** Your system fails to boot from the net.

The problem could be one of the following:

- NIS maps are out of date.  
Report the problem to your system administrator.
- Ethernet cable is not plugged in.  
Plug in the ethernet cable. The system should continue with the booting process.
- Server is not responding: no `carrier` messages.  
Report the problem to your system administrator.
- `tpe-link-test` is disabled.  
Refer to the troubleshooting information in your system documentation.  
(Note: systems that do not have Twisted Pair Ethernet will not have the `tpe-link-test` parameter.)

### *System Will Not Boot From Disk*

**Problem:** You are booting from a disk and the system fails with the message:  
`The file just loaded does not appear to be executable.`

- The boot block is missing or corrupted.  
Install a new boot block.

**Problem:** You are booting from a disk and the system fails with the message:  
`Can't open boot device.`

- The disk may be powered down (especially if it is an external disk).  
Turn on power to the disk, and make sure the SCSI cable is connected to the disk and the system.



## *SCSI Problems*

**Problem:** Your system has more than one disk installed, and you get SCSI-related errors.

- Your system might have duplicate SCSI target number settings. Try the following procedure:
  - a. **Unplug all but one of the disks.**
  - b. **At the `ok` prompt, type:**

```
ok probe-scsi-all
```

Note the target number and its corresponding unit number.

- c. **Plug in another disk and perform step b again.**
- d. **If you get an error, change the target number of this disk to be one of the unused target numbers.**
- e. **Repeat steps b, c, and d until all the disks are plugged back in.**

## *Setting the Console to a Specific Monitor*

**Problem:** You have more than one monitor attached to the system, and the console is not set to an intended monitor.

- If you have more than one monitor attached to the system, the OpenBoot firmware always assigns the console to the frame buffer specified by the `output-device` NVRAM parameter. The default value of `output-device` is `screen`, which is an alias for the first frame buffer that the firmware finds in the system.

A common way to change this default is to change `output-device` to the appropriate frame buffer:

```
ok nvalias myscreen /obio/cgfourteen
ok setenv output-device myscreen
ok reset
```

Another way of setting the console to a specific monitor is to change the `sbus-probe-list` NVRAM parameter.

```
ok show sbus-probe-list (Display the current and default values)
sbus-probe-list f0123 f0123 (Your system may have a different number of SBus slots)
ok
```

If the frame buffer that you are choosing as the console is in slot 2, change `sbus-probe-list` to probe slot 2 first:

```
ok setenv sbus-probe-list 23f01
ok reset
```

## Forth Word Reference



This appendix contains the Forth commands supported by the OpenBoot firmware.

For the most part, the commands are listed in the order in which they were introduced in the chapters. Some of the tables in this appendix show commands not listed elsewhere in this manual. These additional commands (such as memory mapping or output display primitives, or machine-specific register commands) are also part of the set of words in the OpenBoot implementation of Forth; they are included with relevant groups of commands.

Table E-1 Stack Item Notation

Notation	Description
	Alternate stack results, for example: ( input -- adr len false   result true ).
?	Unknown stack items (changed from ???).
???	Unknown stack items.
acf	Code field address.
adr	Memory address (generally a virtual address).
adr16	Memory address, must be 16-bit aligned.
adr32	Memory address, must be 32-bit aligned.
adr64	Memory address, must be 64-bit aligned.
byte bxxx	8-bit value (smallest byte in a 32-bit word).

Table E-1 Stack Item Notation (Continued)

Notation	Description
char	7-bit value (smallest byte), high bit unspecified.
cnt len size	Count or length.
flag xxx?	0 = false; any other value = true (usually -1).
long Lxxx	32-bit value.
n n1 n2 n3	Normal signed values (32-bit).
+n u	Unsigned, positive values (32-bit).
n[64] (n.low n.hi)	Extended-precision (64-bit) numbers (2 stack items).
phys	Physical address (actual hardware address).
pstr	Packed string (adr len means unpacked string).
virt	Virtual address (address used by software).
word wxxx	16-bit value (smallest two bytes in a 32-bit word).

Table E-2 Restricted Monitor Commands

Command	Description
b [ <i>specifiers</i> ]	Boot the operating system (same as boot at the ok prompt).
c	Resume the execution of a halted program (same as go at ok prompt).
n	Enter the Forth Monitor.

Table E-3 Examining and Creating Device Aliases

Command	Description
devalias	Display all current device aliases.
devalias <i>alias</i>	Display the device path name corresponding to <i>alias</i> .
devalias <i>alias device-path</i>	Define an alias representing <i>device-path</i> . If an alias with the same name exists, the new value supercedes the old.

Table E-4 Commands for Browsing the Device Tree

Command	Description
<code>.attributes</code>	Display the names and values of the current node's properties.
<code>cd <i>device-path</i></code>	Select the indicated device node, making it the current node.
<code>cd <i>node-name</i></code>	Search for a node with the given name in the subtree below the current node, and select the first such node found.
<code>cd ..</code>	Select the device node that is the parent of the current node.
<code>cd /</code>	Select the root machine node.
<code>device-end</code>	De-select the current device node, leaving no node selected.
<code>ls</code>	Display the names of the current node's children.
<code>pwd</code>	Display the device path name that names the current node.
<code>show-devs [<i>device-path</i>]</code>	Display all the devices known to the system directly beneath a given level in the device hierarchy. <code>show-devs</code> used by itself shows the entire device tree.
<code>words</code>	Display the names of the current node's methods.

Table E-5 Help Commands

Command	Description
<code>help</code>	List main help categories.
<code>help <i>category</i></code>	Show help for all commands in the category. Use only the first word of the category description.
<code>help <i>command</i></code>	Show help for individual command (where available).

Table E-6 Common Options for the boot Command

Parameter	Description
boot [ <i>device-specifier</i> ] [ <i>filename</i> ] [ <i>options</i> ]	
[ <i>device-specifier</i> ]	The name (full path name or alias) of the boot device. Typical values include: cdrom (CD-ROM drive) disk (hard disk) floppy (3-1/2" diskette drive) net (Ethernet) tape (SCSI tape)
[ <i>filename</i> ]	The name of the program to be booted (for example, <code>stand/diag</code> ). <i>filename</i> is relative to the root of the selected device and partition (if specified). If <i>filename</i> is not specified, the boot program uses the value of the <code>boot-file</code> NVRAM parameter (see Chapter 3).
[ <i>options</i> ]	-a - Prompt interactively for the device and name of the boot file. -h - Halt after loading the program. (These options are specific to the operating system, and may differ from system to system.)

Table E-7 Diagnostic Test Commands

Command	Description
probe-scsi	Identify devices attached to the built-in SCSI bus.
probe-scsi-all [ <i>device-path</i> ]	Perform <code>probe-scsi</code> on all SCSI buses installed in the system below the specified device tree node. (If <i>device-path</i> is absent, the root node is used.)
test <i>device-specifier</i>	Execute the specified device's self-test method. For example: test floppy - test the floppy drive, if installed test /memory - test number of megabytes specified in the <code>selftest-#megs</code> NVRAM parameter; or test all of memory if <code>diag-switch?</code> is true test net - test the network connection
test-all [ <i>device-specifier</i> ]	Test all devices (that have a built-in self-test method) below the specified device tree node. (If <i>device-specifier</i> is absent, the root node is used.)
watch-clock	Test the clock function.
watch-net	Monitor the network connection.

Table E-8 System Information Display Commands

Command	Description
banner	Display power-on banner.
show-sbus	Display list of installed and probed SBus devices.
.enet-addr	Display current Ethernet address.
.idprom	Display ID PROM contents, formatted.
.traps	Display a list of SPARC trap types.
.version	Display version and date of the boot PROM.

Table E-9 NVRAM Configuration Parameters

Parameter	Typical Default	Description
auto-boot?	true	If true, boot automatically after power-on or reset.
boot-device	disk	Device from which to boot.
boot-file	empty string	File to boot (an empty string lets secondary booter choose default).
boot-from	vmunix	Boot device and file (1.x only).
boot-from-diag	le()vmunix	Diagnostic boot device and file (1.x only).
diag-device	net	Diagnostic boot source device.
diag-file	empty string	File from which to boot in diagnostic mode.
diag-switch?	false	If true, run in diagnostic mode.
fcode-debug?	false	If true, include name fields for plug-in device FCodes.
hardware-revision	no default	System version information.
input-device	keyboard	Power-on input device (usually keyboard, ttya, or ttyb).
keyboard-click?	false	If true, enable keyboard click.
keymap	no default	Keymap for custom keyboard.
last-hardware-update	no default	System update information.
local-mac-address?	false	If true, network drivers use their own MAC address, not system's.
mfg-switch?	false	If true, repeat system self-tests until interrupted with Stop-A.
nvrामrc	empty	Contents of NVRAMRC .
oem-banner	empty string	Custom OEM banner (enabled by oem-banner? true).



Table E-9 NVRAM Configuration Parameters (Continued)

Parameter	Typical Default	Description
oem-banner?	false	If true, use custom OEM banner.
oem-logo	no default	Byte array custom OEM logo (enabled by oem-logo? true). Displayed in hexadecimal.
oem-logo?	false	If true, use custom OEM logo (else, use Sun logo).
output-device	screen	Power-on output device (usually screen, ttya, or ttyb).
sbus-probe-list	0123	Which SBus slots are probed and in what order.
screen-#columns	80	Number of on-screen columns (characters/line).
screen-#rows	34	Number of on-screen rows (lines).
scsi-initiator-id	7	SCSI bus address of host adapter, range 0-7.
sd-targets	31204567	Map SCSI disk units (1.x only).
security-#badlogins	no default	Number of incorrect security password attempts.
security-mode	none	Firmware security level (options: none, command, or full).
security-password	no default	Firmware security password (never displayed). <i>Do not set this directly.</i>
selftest-#megs	1	Megabytes of RAM to test. Ignored if diag-switch? is true.
skip-vme-loopback?	false	If true, POST does not do VMEbus loopback tests.
st-targets	45670123	Map SCSI tape units (1.x only).
sunmon-compat?	false	If true, display Restricted Monitor prompt (>).
testarea	0	One-byte scratch field, available for read/write test.
tpe-link-test?	true	Enable 10baseT link test for built-in twisted pair Ethernet.
ttya-mode	9600,8,n,1,-	TTYA (baud rate, #bits, parity, #stop, handshake).
ttyb-mode	9600,8,n,1,-	TTYB (baud rate, #bits, parity, #stop, handshake).
ttya-ignore-cd	true	If true, operating system ignores carrier-detect on TTYA.
ttyb-ignore-cd	true	If true, operating system ignores carrier-detect on TTYB.
ttya-rts-dtr-off	false	If true, operating system does not assert DTR and RTS on TTYA.
ttyb-rts-dtr-off	false	If true, operating system does not assert DTR and RTS on TTYB.
use-nvramrc?	false	If true, execute commands in NVRAMRC during system start-up.
version2?	true	If true, hybrid (1.x/2.x) PROM comes up in version 2.x.
watchdog-reboot?	false	If true, reboot after watchdog reset.



Table E-10 Viewing/Changing Configuration Parameters

Command	Description
printenv	Display all current parameters and current default values. (Numbers are usually shown as decimal values.) printenv <i>parameter</i> shows the current value of the named parameter.
setenv <i>parameter value</i>	Set <i>parameter</i> to the given decimal or text <i>value</i> . (Changes are permanent, but usually only take effect after a reset.)
set-default <i>parameter</i>	Reset the value of the named parameter to the factory default.
set-defaults	Reset parameter values to the factory defaults.

Table E-11 Configuration Parameter Command Primitives

Command	Stack Diagram	Description
nodefault-bytes <i>parameter</i>	( len -- ) Usage: ( -- adr len )	Create custom NVRAM parameter. Use this command in NVRAMRC to make the parameter permanent.
parameter	( -- ??? )	Return the (current) field value (data type is parameter-dependent).
show <i>parameter</i>	( -- )	Display the (current) field value (numbers shown in decimal).

Table E-12 NVRAMRC Editor Commands

Command	Description
nvalias <i>alias device-path</i>	Store the command "devalias <i>alias device-path</i> " in NVRAMRC. The alias persists until the nvunalias or set-defaults commands are executed.
nvedit	Enter the NVRAMRC editor. If data remains in the temporary buffer from a previous nvedit session, resume editing those previous contents. If not, read the contents of NVRAMRC into the temporary buffer and begin editing it.
nvquit	Discard the contents of the temporary buffer, without writing it to NVRAMRC. Prompt for confirmation.
nvrecover	Recover the contents of NVRAMRC if they have been lost as a result of the execution of set-defaults; then enter the editor as with nvedit. nvrecover fails if nvedit is executed between the time that the NVRAMRC contents were lost and the time that nvrecover is executed.

Table E-12 NVRAMRC Editor Commands

Command	Description
nvr <code>run</code>	Execute the contents of the temporary buffer.
nvs <code>to</code> re	Copy the contents of the temporary buffer to NVRAMRC; discard the contents of the temporary buffer.
nvun <code>al</code> ias <i>alias</i>	Delete the corresponding alias from NVRAMRC.

Table E-13 nvedit Keystroke Commands

Keystroke	Description
Control-B	Move backward one character.
Control-C	Exit the NVRAMRC editor and return to the OpenBoot command interpreter. The temporary buffer is preserved but is not written back to NVRAMRC. (Use <code>nvs<code>to</code>re</code> afterwards to write back the temporary buffer.)
Control-F	Move forward one character.
Control-K	If at the end of a line, join the next line to the current line (that is, delete the new line).
Control-L	List all lines.
Control-N	Move to the next line of the NVRAMRC editing buffer.
Control-O	Insert a new line at the cursor position and stay on the current line.
Control-P	Move to the previous line of the NVRAMRC editing buffer.
Delete	Delete the previous character.
Return	Insert a new line at the cursor position and advance to the next line.

Table E-14 Stack Manipulation Commands

Command	Stack Diagram	Description
-rot	( n1 n2 n3 -- n3 n1 n2 )	Inversely rotate 3 stack items.
>r	( n -- )	Move a stack item to the return stack. (Use with caution.)
?dup	( n -- n n   0 )	Duplicate the top stack item if it is non-zero.
2drop	( n1 n2 -- )	Remove 2 items from the stack.
2dup	( n1 n2 -- n1 n2 n1 n2 )	Duplicate 2 stack items.
2over	( n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2 )	Copy second 2 stack items.
2rot	( n1 n2 n3 n4 n5 n6 -- n3 n4 n5 n6 n1 n2 )	Rotate 3 pairs of stack items.

Table E-14 Stack Manipulation Commands (Continued)

Command	Stack Diagram	Description
2swap	( n1 n2 n3 n4 -- n3 n4 n1 n2 )	Exchange 2 pairs of stack items.
3drop	( n1 n2 n3 -- )	Remove 3 items from the stack.
3dup	( n1 n2 n3 -- n1 n2 n3 n1 n2 n3 )	Duplicate 3 stack items.
clear	( ??? -- )	Empty the stack.
depth	( ??? -- ??? +n )	Return the number of items on the stack.
drop	( n -- )	Remove top item from the stack.
dup	( n -- n n )	Duplicate the top stack item.
nip	( n1 n2 -- n2 )	Discard the second stack item.
over	( n1 n2 -- n1 n2 n1 )	Copy second stack item to top of stack.
pick	( ??? +n -- ??? n2 )	Copy +n-th stack item (1 pick = over).
r>	( -- n )	Move a return stack item to the stack. (Use with caution.)
r@	( -- n )	Copy the top of the return stack to the stack.
roll	( ??? +n -- ? )	Rotate +n stack items (2 roll = rot).
rot	( n1 n2 n3 -- n2 n3 n1 )	Rotate 3 stack items.
swap	( n1 n2 -- n2 n1 )	Exchange the top 2 stack items.
tuck	( n1 n2 -- n2 n1 n2 )	Copy top stack item below second item.

Table E-15 Colon Definition Words

Command	Stack Diagram	Description
: name	( -- )	Start creating a new definition.
;	( -- )	Finish creating a new definition.

Table E-16 Arithmetic Functions

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; remainder is discarded.

Table E-16 Arithmetic Functions (Continued)

Command	Stack Diagram	Description
/mod	( n1 n2 -- rem quot )	Remainder, quotient of n1 / n2.
<<	( n1 +n -- n2 )	Left-shift n1 by +n bits.
>>	( n1 +n -- n2 )	Right-shift n1 by +n bits.
>>a	( n1 +n -- n2 )	Arithmetic right-shift n1 by +n bits.
*/	( n1 n2 n3 -- n4 )	n1 * n2 / n3.
*/mod	( n1 n2 n3 -- rem quot )	Remainder, quotient of n1 * n2 / n3.
1+	( n1 -- n2 )	Add 1.
1-	( n1 -- n2 )	Subtract 1.
2*	( n1 -- n2 )	Multiply by 2.
2+	( n1 -- n2 )	Add 2.
2-	( n1 -- n2 )	Subtract 2.
2/	( n1 -- n2 )	Divide by 2.
abs	( n -- u )	Absolute value.
aligned	( n1 -- n2 )	Round n1 up to the next multiple of 4.
and	( n1 n2 -- n3 )	Bitwise logical AND.
bounds	( startadr len -- endadr startadr )	Convert startadr len to endadr startadr for do loop.
bljoin	( b.low b2 b3 b.hi -- long )	Join four bytes to form a 32-bit longword.
bwjoin	( b.low b.hi -- word )	Join two bytes to form a 16-bit word.
flip	( word1 -- word2 )	Swap the bytes within a 16-bit word.
lbsplit	( long -- b.low b2 b3 b.hi )	Split a 32-bit longword into four bytes.
lwsplit	( long -- w.low w.hi )	Split a 32-bit longword into two 16-bit words.
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.
negate	( n1 -- n2 )	Change the sign of n1.
not	( n1 -- n2 )	Bitwise ones complement.
or	( n1 n2 -- n3 )	Bitwise logical OR.
u*x	( u1 u2 -- product[64] )	Multiply 2 unsigned 32-bit numbers; yield unsigned 64-bit product.
u/mod	( u1 u2 -- un.rem un.quot )	Divide unsigned 32-bit number by an unsigned 32-bit number; yield 32-bit remainder and quotient.
u2/	( u1 -- u2 )	Logical right shift 1 bit; zero shifted into vacated sign bit.

Table E-16 Arithmetic Functions (Continued)

Command	Stack Diagram	Description
wbsplit	( word -- b.low b.hi )	Split 16-bit word into two bytes.
wflip	( long1 -- long2 )	Swap halves of 32-bit longword.
wljoin	( w.low w.hi -- long )	Join two words to form a longword.
x+	( n1[64] n2[64] -- n3[64] )	Add two 64-bit numbers.
x-	( n1[64] n2[64] -- n3[64] )	Subtract two 64-bit numbers.
xor	( n1 n2 -- n3 )	Bitwise exclusive OR.
xu/mod	( u1[64] u2 -- rem quot )	Divide unsigned 64-bit number by unsigned 32-bit number; yield 32-bit remainder and quotient.

Table E-17 Conversion Operators

Command	Stack Diagram	Description
/c	( -- n )	The number of bytes in a byte; 1.
/c*	( n1 -- n2 )	Multiply n1 by /c.
ca+	( adr1 index -- adr2 )	Increment adr1 by index times /c.
ca1+	( adr1 -- adr2 )	Increment adr1 by /c.
/L	( -- n )	Number of bytes in a longword; 4.
/L*	( n1 -- n2 )	Multiply n1 by /L.
La+	( adr1 index -- adr2 )	Increment adr1 by index times /L.
La1+	( adr1 -- adr2 )	Increment adr1 by /L.
/n	( -- n )	Number of bytes in a normal; 4.
/n*	( n1 -- n2 )	Multiply n1 by /n.
na+	( adr1 index -- adr2 )	Increment adr1 by index times /n.
na1+	( adr1 -- adr2 )	Increment adr1 by /n.
/w	( -- n )	Number of bytes in a 16-bit word; 2.
/w*	( n1 -- n2 )	Multiply n1 by /w.
wa+	( adr1 index -- adr2 )	Increment adr1 by index times /w.
wa1+	( adr1 -- adr2 )	Increment adr1 by /w.

Table E-18 Memory Access Commands

Command	Stack Diagram	Description
!	( n adr16 -- )	Store a 32-bit number at adr16, must be 16-bit aligned.
+!	( n adr16 -- )	Add n to the 32-bit number stored at adr16, must be 16-bit aligned.
<w@	( adr16 -- n )	Fetch signed 16-bit word at adr16, must be 16-bit aligned.
?	( adr16 -- )	Display the 32-bit number at adr16, must be 16-bit aligned.
@	( adr16 -- n )	Fetch a 32-bit number from adr16, must be 16-bit aligned.
2!	( n1 n2 adr16 -- )	Store 2 numbers at adr16, n2 at lower address, must be 16-bit aligned.
2@	( adr16 -- n1 n2 )	Fetch 2 numbers from adr16, n2 from lower address, must be 16-bit aligned.
blank	( adr u -- )	Set u bytes of memory to space (decimal 32).
c!	( n adr -- )	Store low byte of n at adr.
c?	( adr -- )	Display the byte at adr.
c@	( adr -- byte )	Fetch a byte from adr.
cmove	( adr1 adr2 u -- )	Copy u bytes from adr1 to adr2, starting at low byte.
cmove>	( adr1 adr2 u -- )	Copy u bytes from adr1 to adr2, starting at high byte.
cpeek	( adr -- false   byte true )	Fetch the byte at adr. Return the data and true if the access was successful. Return false if a read access error occurred.
cpoke	( byte adr -- okay? )	Store the byte to adr. Return true if the access was successful. Return false if a write access error occurred.
comp	( adr1 adr2 len -- n )	Compare two byte arrays, n = 0 if arrays are identical, n = 1 if first byte that is different is greater in array#1, n = -1 otherwise.
d!	( n1 n2 adr64 -- )	Store two 32-bit numbers at adr64, must be 64-bit aligned. Order is implementation-dependent.
d?	( adr64 -- )	Display the two 32-bit numbers at adr64, must be 64-bit aligned. Order is implementation-dependent.
d@	( adr64-- n1 n2 )	Fetch two 32-bit numbers from adr64, must be 64-bit aligned. Order is implementation-dependent.
dump	( adr len -- )	Display len bytes of memory starting at adr.
erase	( adr u -- )	Set u bytes of memory to 0.
fill	( adr size byte -- )	Set size bytes of memory to byte.
L!	( n adr32 -- )	Store a 32-bit number at adr32, must be 32-bit aligned.

Table E-18 Memory Access Commands (Continued)

Command	Stack Diagram	Description
L?	( adr32 -- )	Display the 32-bit number at adr32, must be 32-bit aligned.
L@	( adr32 -- long )	Fetch a 32-bit number from adr32, must be 32-bit aligned.
lflips	( adr len -- )	Exchange 16-bit words within 32-bit longwords in specified region.
lpeek	( adr32 -- false   long true )	Fetch the 32-bit quantity at adr32. Return the data and true if the access was successful. Return false if a read access error occurred.
lpoke	( long adr32 -- okay? )	Store the 32-bit quantity at adr32. Return true if the access was successful. Return false if a write access error occurred.
move	( adr1 adr2 u -- )	Copy u bytes from adr1 to adr2, handle overlap properly.
off	( adr16 -- )	Store false (32-bit 0) at adr16.
on	( adr16 -- )	Store true (32-bit -1) at adr16.
unaligned-L!	( long adr -- )	Store a 32-bit number, any alignment
unaligned-L@	( adr -- long )	Fetch a 32-bit number, any alignment.
unaligned-w!	( word adr -- )	Store a 16-bit number, any alignment.
unaligned-w@	( adr -- word )	Fetch a 16-bit number, any alignment.
w!	( n adr16 -- )	Store a 16-bit number at adr16, must be 16-bit aligned.
w?	( adr16 -- )	Display the 16-bit number at adr16, must be 16-bit aligned.
w@	( adr16 -- word )	Fetch a 16-bit number from adr16, must be 16-bit aligned.
wflips	( adr len -- )	Exchange bytes within 16-bit words in specified region.
wpeek	( adr16 -- false   word true )	Fetch the 16-bit quantity at adr16. Return the data and true if the access was successful. Return false if a read access error occurred.
wpoke	( word adr16 -- okay? )	Store the 16-bit quantity to adr16. Return true if the access was successful. Return false if a write access error occurred.

Table E-19 Memory Mapping Commands

Command	Stack Diagram	Description
alloc-mem	( size -- virt )	Allocate and map size bytes of available memory; return the virtual address. Unmap with free-mem.
free-mem	( virt size -- )	Free memory allocated by alloc-mem.
free-virtual	( virt size -- )	Undo mappings created with memmap.
map?	( virt -- )	Display memory map information for the virtual address.

Table E-19 Memory Mapping Commands (Continued)

Command	Stack Diagram	Description
memmap	( phys space size -- virt )	Map a region of physical addresses; return the allocated virtual address. Unmap with <code>free-virtual</code> .
obio	( -- space )	Specify the device address space for mapping.
obmem	( -- space )	Specify the onboard memory address space for mapping.
sbus	( -- space )	Specify the SBus address space for mapping.

Table E-20 Memory Mapping Primitives

Command	Stack Diagram	Description
cacheable	( space -- cache-space )	Modify the address space so that the subsequent address mapping is made cacheable.
iomap?	( virt -- )	Display IOMMU page map entry for the virtual address. The stack diagram shown applies to Sun-4m machines.
iomap-page	( phys space virt -- )	Map physical page given by <code>phys</code> and <code>space</code> to the virtual address. The stack diagram shown applies to Sun-4m machines.
iomap-pages	( phys space virt size -- )	Perform consecutive <code>iomap-pages</code> to map a region of memory given by <code>size</code> . The stack diagram shown applies to Sun-4m machines.
iopgmap@	( virt -- pte   0 )	Return IOMMU page map entry for the virtual address. The stack diagram shown applies to Sun-4m machines.
iopgmap!	( pte virt -- )	Store a new page map entry for the virtual address. The stack diagram shown applies to Sun-4m machines.
map-page	( phys space virt -- )	Map one page of memory starting at address <code>phys</code> on to virtual address <code>virt</code> in the given address space. All addresses are truncated to lie on a page boundary.
map-pages	( phys space virt size -- )	Perform consecutive <code>map-pages</code> to map a region of memory to the given size.
map-region	( region# virt -- )	Map a region.
map-regions	( region# virt size -- )	Map successive regions.
map-segments	( smentry virt len -- )	Perform consecutive <code>smap!s</code> to map a region of memory.



Table E-20 Memory Mapping Primitives (Continued)

Command	Stack Diagram	Description
pgmap!	( pmentry virt -- )	Store a new page map entry for the virtual address.
pgmap?	( virt -- )	Display the page map entry (decoded and in English) corresponding to the virtual address.
pgmap@	( virt -- pmentry )	Return the page map entry for the virtual address.
pagesize	( -- size )	Return the size of a page, often 4K (hex 1000).
rmap!	( rmentry virt -- )	Store a new region map entry for the virtual address.
rmap@	( virt -- rmentry )	Return the region map entry for the virtual address.
segmentsize	( -- size )	Return the size of a segment, often 256K (hex 40000).
smap!	( smentry virt -- )	Store a new segment map entry for the virtual address.
smap?	( virt -- )	Formatted display of the segment map entry for the virtual address.
smap@	( virt -- smentry )	Return the segment map entry for the virtual address.

Table E-21 Cache Manipulation Commands

Command	Stack Diagram	Description
clear-cache	( -- )	Invalidate all cache entries.
cache-off	( -- )	Disable the cache.
cache-on	( -- )	Enable the cache.
cdata!	( data offset -- )	Store the 32-bit data at the cache offset.
cdata@	( offset -- data )	Fetch (return) data from the cache offset.
ctag!	( value offset -- )	Store the tag value at the cache offset.
ctag@	( offset -- value )	Return the tag value at the cache offset.
flush-cache	( -- )	Write back any pending data from the cache.

*Table E-22 Reading/Writing Machine Registers in Sun-4D Machines*

<b>Command</b>	<b>Stack Diagram</b>	<b>Description</b>
<b>SuperSPARC™ Module Register Access</b>		
cxr!	( data -- )	Write MMU context register.
mcr!	( data -- )	Write module control register.
cxr@	( -- data )	Read MMU context register.
mcr@	( -- data )	Read MMU control register.
sfsr@	( -- data )	Read synchronous fault status register.
sfar@	( -- data )	Read synchronous fault address register.
afsr@	( -- data )	Read asynchronous fault status register.
afar@	( -- data )	Read asynchronous fault address register.
.mcr	( -- )	Display module control register.
.sfsr	( -- )	Display synchronous fault status register.
<b>MXCC Interrupt Register Access</b>		
interrupt- enable!	( data -- )	Write interrupt mask register.
interrupt- enable@	( -- data )	Read interrupt mask register.
interrupt- pending@	( -- data )	Read interrupt pending register.
interrupt- clear!	( data -- )	Write interrupt clear register.
<b>BootBus Register Access</b>		
control!	( datat -- )	Write BootBus control register.
control@	( -- datat )	Read BootBus control register.
status1@	( -- datat )	Read BootBus status1 register.
status2@	( -- datat )	Read BootBus status2 register.

Table E-23 Reading/Writing Machine Registers in Sun-4M Machines

Command	Stack Diagram	Description
.mcr	( -- )	Display module control register.
.mfsr	( -- )	Display memory controller fault status register.
.sfsr	( -- )	Display synchronous fault status register.
.sipr	( -- )	Display system interrupt pending register.
aux!	( data -- )	Write auxiliary register.
aux@	( -- data )	Read auxiliary register.
cxr!	( data -- )	Write MMU context register.
cxr@	( -- data )	Read MMU context register.
interrupt- enable!	( data -- )	Write system interrupt target mask register.
interrupt- enable@	( -- data )	Read system interrupt target mask register.
iommu-ctl!	( data -- )	Write IOMMU control register.
iommu-ctl@	( -- data )	Read IOMMU control register.
mcr!	( data -- )	Write module control register.
mcr@	( -- data )	Read module control register.
mfsr!	( data -- )	Write memory controller fault status register.
mfsr@	( -- data )	Read memory controller fault status register.
msafar@	( -- data )	Read MBus-to-SBus asynchronous fault address register.
msafsr!	( data -- )	Write MBus-to-SBus asynchronous fault status register.
msafsr@	( -- data )	Read MBus-to-SBus asynchronous fault status register.
sfsr!	( data -- )	Write synchronous fault status register.
sfsr@	( -- data )	Read synchronous fault status register.
sfar!	( data -- )	Write synchronous fault address register.
sfar@	( -- data )	Read synchronous fault address register.

*Table E-24 Reading/Writing Machine Registers in Sun-4C Machines*

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

*Table E-25 Alternate Address Space Access Commands*

<b>Command</b>	<b>Stack Diagram</b>	<b>Description</b>
spacec!	( byte adr asi -- )	Store the byte at asi and address.
spacec?	( adr asi -- )	Display the byte at asi and address.
spacec@	( adr asi -- byte )	Fetch the byte from asi and address.
spaced!	( n1 n2 adr asi -- )	Store the two 32-bit words at asi and address. Order is implementation-dependent.
spaced?	( adr asi -- )	Display the two 32-bit words at asi and address. Order is implementation-dependent.
spaced@	( adr asi -- n1 n2 )	Fetch the two 32-bit words from asi and address. Order is implementation-dependent.

Table E-25 Alternate Address Space Access Commands

Command	Stack Diagram	Description
spaceL!	( long adr asi -- )	Store the 32-bit word at asi and address.
spaceL?	( adr asi -- )	Display the 32-bit word at asi and address.
spaceL@	( adr asi -- long )	Fetch the 32-bit word from asi and address.
spacew!	( word adr asi -- )	Store the 16-bit word at asi and address.
spacew?	( adr asi -- )	Display the 16-bit word at asi and address.
spacew@	( adr asi -- word )	Fetch the 16-bit word from asi and address.

Table E-26 Defining Words

Command	Stack Diagram	Description
: <i>name</i>	( -- ) Usage: ( ??? -- ? )	Start creating a new colon definition.
;	( -- )	Finish creating a new colon definition.
alias <i>new-name</i> <i>old-name</i>	( -- ) Usage: ( ??? -- ? )	Create <i>new-name</i> with the same responses as <i>old-name</i> .
buffer: <i>name</i>	( size -- ) Usage: ( -- adr64 )	Create a named array in temporary storage.
constant <i>name</i>	( n -- ) Usage: ( -- n )	Define a constant (for example, 3 constant bar).
2constant <i>name</i>	( n1 n2 -- ) Usage: ( -- n1 n2 )	Define a 2-number constant.
create <i>name</i>	( -- ) Usage: ( -- adr16 )	Generic defining word.
defer <i>name</i>	( -- ) Usage: ( ??? -- ? )	Define a word for forward references or execution vectors using code field address.
does>	( -- adr16 )	Start the run-time clause for defining words.
field <i>name</i>	( offset size -- offset+size ) Usage: ( adr -- adr+offset )	Create a named offset pointer.
struct	( -- 0 )	Initialize for field creation.
value <i>name</i>	( n -- ) Usage: ( -- n )	Create a changeable, named 32-bit quantity.
variable <i>name</i>	( -- ) Usage: ( -- adr16 )	Define a variable.

Table E-27 Dictionary Searching Commands

Command	Stack Diagram	Description
' <i>name</i>	( -- acf )	Find the named word in the dictionary. Returns the code field address. Use outside definitions.
[ ' ] <i>name</i>	( -- acf )	Similar to ' but is used either inside or outside definitions.
.calls	( acf -- )	Display a list of all words that call the word whose compilation address is <i>acf</i> .
\$find	( adr len -- adr len false   acf n )	Find a word. <i>n</i> = 0 if not found, <i>n</i> = 1 if immediate, <i>n</i> = -1 otherwise.
find	( pstr -- pstr false   acf n )	Search for a word in the dictionary. The word to be found is indicated by <i>pstr</i> . <i>n</i> = 0 if not found, <i>n</i> = 1 if immediate, <i>n</i> = -1 otherwise.
see <i>thisword</i>	( -- )	Decompile the named command.
(see)	( acf -- )	Decompile the word indicated by the code field address.
sift	( pstr -- )	Display names of all dictionary entries containing the string pointed to by <i>pstr</i> .
sifting <i>ccc</i>	( -- )	Display names of all dictionary entries containing the sequence of characters. <i>ccc</i> contains no spaces.
words	( -- )	Display all visible words in the dictionary.

Table E-28 Dictionary Compilation Commands

Command	Stack Diagram	Description
,	( n -- )	Place a number in the dictionary.
c,	( byte -- )	Place a byte in the dictionary.
w,	( word -- )	Place a 16-bit number in the dictionary.
L,	( long -- )	Place a 32-bit number in the dictionary.
[	( -- )	Begin interpreting.
]	( -- )	End interpreting, resume compilation.
allot	( n -- )	Allocate <i>n</i> bytes in the dictionary.
>body	( acf -- apf )	Find parameter field address from compilation address.

Table E-28 Dictionary Compilation Commands (Continued)

Command	Stack Diagram	Description
body>	( apf -- acf )	Find compilation address from parameter field address.
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 definition as immediate.
is <i>name</i>	( n -- )	Install a new action in a defer word or value.
literal	( n -- )	Compile a number.
origin	( -- adr )	Return the address of the start of the Forth system.
patch <i>new-word old-word word-to-patch</i>	( -- )	Replace <i>old-word</i> with <i>new-word</i> in <i>word-to-patch</i> .
(patch	( new-n old-n acf -- )	Replace <i>old-n</i> with <i>new-n</i> in word indicated by <i>acf</i> .
recursive	( -- )	Make the name of the colon definition being compiled visible in the dictionary, and thus allow the name of the word to be used recursively in its own definition.
state	( -- adr )	Variable that is non-zero in compile state.

Table E-29 Assembly Language Programming

Command	Stack Diagram	Description
code <i>name</i>	( -- ) Usage: ( ??? -- ? )	Begin the creation of an assembly language routine called <i>name</i> . Commands that follow are interpreted as assembler mnemonics. Note that if the assembler is not installed, <i>code</i> is still present, except that machine code must be entered numerically (for example, in hex) with “,”.

Table E-29 Assembly Language Programming

Command	Stack Diagram	Description
c;	( -- )	End the creation of an assembly language routine. Automatically assemble the Forth interpreter <code>next</code> function so that the created assembly-code word, when executed, returns control to the calling routine as usual.
label name	( -- ) Usage: ( -- adr16 )	Begin the creation of an assembly language routine called <i>name</i> . Words created with <code>label</code> leave the address of the code on the stack when executed. The commands that follow are interpreted as assembler mnemonics. As with <code>code</code> , <code>label</code> is present even if the assembler is not installed.
end-code	( -- )	End the assembly language patch started with <code>label</code> .

Table E-30 Basic Number Display

Command	Stack Diagram	Description
.	( n -- )	Display a number in the current base.
.r	( n size -- )	Display a number in a fixed width field.
.s	( -- )	Display contents of data stack.
showstack	( -- )	Execute <code>.s</code> automatically before each <code>ok</code> prompt.
u.	( u -- )	Display an unsigned number.
u.r	( u size -- )	Display an unsigned number in a fixed width field.

Table E-31 Changing the Number Base

Command	Stack Diagram	Description
base	( -- adr )	Variable containing number base.
binary	( -- )	Set the number base to 2.
decimal	( -- )	Set the number base to 10.
d# number	( -- n )	Interpret the next number in decimal; base is unchanged.
hex	( -- )	Set the number base to 16.
h# number	( -- n )	Interpret the next number in hex; base is unchanged.
.d	( n -- )	Display <i>n</i> in decimal without changing base.
.h	( n -- )	Display <i>n</i> in hex without changing base.



Table E-32 Numeric Output Word Primitives

Command	Stack Diagram	Description
#	( +L1 -- +L2 )	Convert a digit in pictured numeric output.
#>	( L -- adr +n )	End pictured numeric output.
<#	( -- )	Initialize pictured numeric output.
(.)	( n -- )	Convert a number to a string.
(u.)	( -- adr len )	Convert unsigned to string.
digit	( char base -- digit true   char false )	Convert a character to a digit.
hold	( char -- )	Insert the char in the pictured numeric output string.
\$number	( adr len -- true   n false )	Convert a string to a number.
#s	( L -- 0 )	Convert the rest of the digits in pictured numeric output.
sign	( n -- )	Set sign of pictured output.

Table E-33 Controlling Text Input

Command	Stack Diagram	Description
( ccc )	( -- )	Begin a comment.
\ <i>rest-of-line</i>	( -- )	Skip the rest of the line.
ascii ccc	( -- char )	Get numerical value of first ASCII character of next word.
expect	( adr +n -- )	Get a line of edited input from the assigned input device's keyboard; store at adr.
key	( -- char )	Read a character from the assigned input device's keyboard.
key?	( -- flag )	True if a key has been typed on the input device's keyboard.
span	( -- adr16 )	Variable containing the number of characters read by expect.
word	( char -- pstr )	Collect a string delimited by char from input string and place in memory at pstr.

Table E-34 Displaying Text Output

Command	Stack Diagram	Description
. " ccc"	( -- )	Compile a string for later display.
(cr	( -- )	Move the output cursor back to the beginning of the current line.
cr	( -- )	Terminate a line on the display and go to the next line.
emit	( char -- )	Display the character.
exit?	( -- flag )	Enable the scrolling control prompt: More [<space>, <cr>, q] ? The return flag is true if the user wants the output to be terminated.
space	( -- )	Display a space character.
spaces	( +n -- )	Display +n spaces.
type	( adr +n -- )	Display n characters.

Table E-35 Formatted Output

Command	Stack Diagram	Description
#line	( -- adr16 )	Variable holding the line number on the output device.
#out	( -- adr16 )	Variable holding the column number on the output device.

Table E-36 Manipulating Text Strings

Command	Stack Diagram	Description
",	( adr len -- )	Compile an array of bytes from adr of length len, at the top of the dictionary as a packed string.
" ccc"	( -- adr len )	Collect an input stream string, either interpreted or compiled. Within the string, "(00, ff...)" can be used to include arbitrary byte values.
. ( ccc)	( -- )	Display a string immediately.
-trailing	( adr +n1 -- adr +n2 )	Remove trailing spaces.
bl	( -- char )	ASCII code for the space character; decimal 32.
count	( pstr -- adr +n )	Unpack a packed string.
lcc	( char -- lowercase-char )	Convert a character to lowercase.

Table E-36 Manipulating Text Strings (Continued)

Command	Stack Diagram	Description
left-parse-string	( adr len char -- adrR lenR adrL lenL )	Split a string at the given delimiter (which is discarded).
pack	( adr len pstr -- pstr )	Make a packed string from adr len; place it at pstr.
p" ccc"	( -- pstr )	Collect a string from the input stream; store as a packed string.
upc	( char -- uppercase-char )	Convert a character to uppercase.

Table E-37 I/O Redirection Commands

Command	Stack Diagram	Description
input	( device -- )	Select device (ttya, ttyb, keyboard, or " device-specifier") for subsequent input.
io	( device -- )	Select device for subsequent input and output.
output	( device -- )	Select device (ttya, ttyb, screen, or " device-specifier") for subsequent output.

Table E-38 ASCII Constants

Command	Stack Diagram	Description
bell	( -- n )	ASCII code for the bell character; decimal 7.
bs	( -- n )	ASCII code for the backspace character; decimal 8.

Table E-39 Line Editor Commands

Command	Function
Control-A	Go to start of line.
Control-B	Go backward one character.
Control-D	Erase this character.
Control-E	Go to end of line.

*Table E-39* Line Editor Commands (Continued)

<b>Command</b>	<b>Function</b>
Control-F	Go forward one character.
Control-H	Erase previous character (also Delete or Back Space keys).
Control-K	Erase forward, from here to end of line.
Control-L	Show command history list, then re-type line.
Control-N	Recall subsequent command line.
Control-P	Recall previous command line.
Control-Q	Quote next character (to type a control character).
Control-R	Re-type line.
Control-U	Erase entire line.
Control-W	Erase previous word.
Control-Y	Insert save buffer contents before the cursor.
Control-space	Complete the current command.
Control-/	Show all possible matches/completions.
Control-?	Show all possible matches/completions.
Control-}	Show all possible matches/completions.
Esc-B	Go backward one word.
Esc-D	Erase this portion of word, from here to end of word.
Esc-F	Go forward one word.
Esc-H	Erase previous portion of word (also Control-W).

*Table E-40* Comparison Commands

<b>Command</b>	<b>Stack Diagram</b>	<b>Description</b>
<	( 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.
0<	( n -- flag )	True if n < 0.

Table E-40 Comparison Commands (Continued)

Command	Stack Diagram	Description
0<=	( n -- flag )	True if $n \leq 0$ .
0<>	( n -- flag )	True if $n \neq 0$ .
0=	( n -- flag )	True if $n = 0$ (also inverts any flag).
0>	( n -- flag )	True if $n > 0$ .
0>=	( n -- flag )	True if $n \geq 0$ .
between	( n min max -- flag )	True if $\text{min} \leq n \leq \text{max}$ .
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 $u1 \leq u2$ , unsigned.
u>	( u1 u2 -- flag )	True if $u1 > u2$ , unsigned.
u>=	( u1 u2 -- flag )	True if $u1 \geq u2$ , unsigned.
within	( n min max -- flag )	True if $\text{min} \leq n < \text{max}$ .

Table E-41 if-then-else Commands

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

Table E-42 case Statement Commands

Command	Stack Diagram	Description
case	( selector -- selector )	Begin a case...endcase conditional.
endcase	( selector   {empty} -- )	Terminate a case...endcase conditional.
endof	( -- )	Terminate an of...endof clause within a case...endcase
of	( selector test-value -- selector   {empty} )	Begin an of...endof clause within a case conditional.

Table E-43 begin (Conditional) Loop Commands

Command	Stack Diagram	Description
again	( -- )	End a begin...again infinite loop.
begin	( -- )	Begin a begin...while...repeat, begin...until, or begin...again loop.
repeat	( -- )	End a begin...while...repeat loop.
until	( flag -- )	Continue executing a begin...until loop until flag is true.
while	( flag -- )	Continue executing a begin...while...repeat loop while flag is true.

Table E-44 do (Counted) Loop Commands

Command	Stack Diagram	Description
+loop	( n -- )	End a do...+loop construct; add n to loop index and return to do (if n < 0, index goes from start to end inclusive).
?do	( end start -- )	Begin ?do...loop to be executed 0 or more times. Index goes from start to end-1 inclusive. If end = start, loop is not executed.
?leave	( flag -- )	Exit from a do...loop if flag is non-zero.
do	( end start -- )	Begin a do...loop. Index goes from start to end-1 inclusive. Example: 10 0 do i . loop (prints 0 1 2...d e f).
i	( -- n )	Loop index.
j	( -- n )	Loop index for next enclosing loop.
leave	( -- )	Exit from do...loop.
loop	( -- )	End of do...loop.

Table E-45 Program Execution Control Commands

Command	Stack Diagram	Description
abort	( -- )	Abort current execution and interpret keyboard commands.
abort" ccc"	( abort? -- )	If flag is true, abort and display message.
eval	( adr len -- )	Interpret Forth source from an array.

Table E-45 Program Execution Control Commands

Command	Stack Diagram	Description
execute	( acf -- )	Execute the word whose code field address is on the stack.
exit	( -- )	Return from the current word. (Cannot be used in counted loops.)
quit	( -- )	Same as <code>abort</code> , but leave stack intact.

Table E-46 File Loading Commands

Command	Stack Diagram	Description
?go	( -- )	Execute Forth, FCode, or binary programs.
boot [ <i>specifiers</i> ] -h	( -- )	Load file from specified source.
byte-load	( adr span -- )	Interpret loaded FCode binary file. <i>span</i> is usually 1.
dl	( -- )	Load a Forth file over a serial line with <code>TIP</code> and interpret. Type: ~C <i>cat filename</i> ^D
dlbin	( -- )	Load a binary file over a serial line with <code>TIP</code> . Type: ~C <i>cat filename</i>
dload <i>filename</i>	( adr -- )	Load the specified file over Ethernet at the given address.
eval	( adr len -- )	Interpret loaded Forth text file.
go	( -- )	Begin executing a previously-loaded binary program, or resume executing an interrupted program.
init-program	( -- )	Initialize to execute a binary file.
load <i>device-specifier argument</i>	( -- )	Load data from specified device into memory at the address given by <code>load-base</code> .
load-base	( -- adr )	Address at which <code>load</code> places the data it reads from a device.

Table E-47 Disassembler Commands

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

Table E-48 SPARC Register Commands

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

Table E-49 Breakpoint Commands

Command	Stack Diagram	Description
+bp	( adr -- )	Add a breakpoint at the given address.
-bp	( adr -- )	Remove the breakpoint at the given address.
--bp	( -- )	Remove the most-recently-set breakpoint.
.bp	( -- )	Display all currently set breakpoints.



Table E-49 Breakpoint Commands (Continued)

Command	Stack Diagram	Description
.breakpoint	( -- )	Perform a specified action when a breakpoint occurs. This word can be altered to perform any desired action. For example, to display registers at every breakpoint, type: ['] .registers is .breakpoint. The default action is .instruction. To perform multiple actions, create a single definition which calls all desired actions, then load that word into .breakpoint.
.instruction	( -- )	Display the address, opcode for the last-encountered breakpoint.
.step	( -- )	Perform a specified action when a single step occurs (see .breakpoint).
bpcoff	( -- )	Remove all breakpoints.
finish-loop	( -- )	Execute until the end of this loop.
go	( -- )	Continue from a breakpoint. This can be used to go to an arbitrary address by setting up the processor's program counter before issuing go.
gos	( n -- )	Execute go n times.
hop	( -- )	(Like the step command.) Treat a subroutine call as a single instruction.
hops	( n -- )	Execute hop n times.
return	( -- )	Execute until the end of this subroutine.
returnL	( -- )	Execute until the end of this leaf subroutine.
skip	( -- )	Skip (do not execute) the current instruction.
step	( -- )	Single-step one instruction.
steps	( n -- )	Execute step n times.
till	( adr -- )	Execute until the given address is encountered. Equivalent to +bp go.

Table E-50 Forth Source-level Debugger Commands

Command	Description
C	"Continue". Switch from stepping to tracing, thus tracing the remainder of the execution of the word being debugged.
D	"Down a level". Mark for debugging the word whose name was just displayed, then execute it.
F	Start a subordinate Forth interpreter. When that interpreter exits (with resume), control returns to the debugger at the place where the F command was executed.
Q	"Quit". Abort the execution of the word being debugged and all its callers and return to the command interpreter.
U	"Up a level". Un-mark the word being debugged, mark its caller for debugging, and finish executing the word that was previously being debugged.

Table E-50 Forth Source-level Debugger Commands (Continued)

Command	Description
<code>debug name</code>	Mark the named Forth word for debugging. Enter the Forth Source-level Debugger with any subsequent attempts to execute that word. After executing <code>debug</code> , the execution speed of the system may decrease until debugging is turned off with <code>debug-off</code> . (Do not debug basic Forth words such as “.”.)
<code>debug-off</code>	Turn off the Forth Source-level Debugger so that no word is being debugged.
<code>resume</code>	Exit from a subordinate interpreter, and go back to the stepper (see the <code>F</code> command in this table).
<code>stepping</code>	Set <code>step mode</code> for the Forth Source-level Debugger, allowing the interactive, step-by-step execution of the word being debugged. Step mode is the default.
<code>tracing</code>	Set <code>trace mode</code> for the Forth Source-level Debugger. This traces the execution of the word being debugged, while showing the name and stack contents for each word called by that word.
<code>Space</code>	Execute the word just displayed and proceed to the next word.

Table E-51 Time Utilities

Command	Stack Diagram	Description
<code>get-msecs</code>	( -- ms )	Return the approximate current time in milliseconds.
<code>ms</code>	( n -- )	Delay for n milliseconds. Resolution is 1 millisecond.

Table E-52 Miscellaneous Operations

Command	Stack Diagram	Description
<code>callback string</code>	( value -- )	Call SunOS™ with the given value and string.
<code>catch</code>	( ??? acf -- ? error-code )	Execute <code>acf</code> ; return <code>throw</code> error code or 0 if <code>throw</code> is not called.
<code>eject-floppy</code>	( -- )	Eject the diskette from the floppy drive.
<code>firmware-version</code>	(-- n)	Return major/minor CPU firmware version (that is, 0x00020001 = firmware version 2.1).
<code>forth</code>	( -- )	Restore main Forth vocabulary to top of search order.
<code>ftrace</code>	( -- )	Show calling sequence when exception occurred.
<code>noop</code>	( -- )	Do nothing.
<code>old-mode</code>	( -- )	Go to Restricted Monitor.
<code>reset</code>	( -- )	Reset the entire system (similar to a power-cycle).

Table E-52 Miscellaneous Operations (Continued)

Command	Stack Diagram	Description
ramforth	( -- )	Copy Forth dictionary to RAM. (Speeds up interpretation in some systems and enables system word patching.)
romforth	( -- )	Turn off ramforth.
sync	( -- )	Call the operating system to write any pending information to the hard disk. Also boot after sync-ing file systems.
throw	( error-code -- )	Return given error code to catch.

Table E-53 Multiprocessor Commands

Command	Stack Diagram	Description
module-info	( -- )	Display type and speed of all CPU modules.
switch-cpu	( cpu# -- )	Switch to indicated CPU.

Table E-54 Emergency Keyboard Commands

Command	Description
Stop	Bypass POST. This command does not depend on security-mode. (Note: some systems bypass POST as a default; in such cases, use Stop-D to start POST.)
Stop-A	Abort.
Stop-D	Enter diagnostic mode (set diag-switch? to true).
Stop-F	Enter Forth on TTYA instead of probing. Use fexit to continue with the initialization sequence. Useful if hardware is broken.
Stop-N	Reset NVRAM contents to default values.



# Index

---

## Symbols

!, 53, 61, 128  
" ccc", 68, 140  
",, 68, 140  
#, 139  
#>, 139  
#line, 140  
#out, 140  
#s, 139  
\$find, 62, 136  
\$number, 139  
%f0, 90, 146  
%fsr, 90, 146  
%g0, 90, 146  
%i0, 90, 146  
%L0, 90, 146  
%npc, 90, 91, 146  
%o0, 90, 146  
%pc, 90, 91, 92, 146  
%psr, 90, 146  
%tbr, 90, 146  
%wim, 90, 146  
%y, 90, 146  
(, 50, 66  
( ccc ), 66, 139  
(.), 139  
(cr, 67, 140  
(patch, 64, 137  
(see), 62, 136  
(u.), 139  
) , 50, 66  
\*, 51, 125  
\*/ , 51, 126  
\*/mod, 51, 126  
+, 45, 51, 125  
+!, 53, 128  
+bp, 91, 93, 146  
+dis, 89, 145  
+loop, 79, 144  
+n, 118  
., 63, 136  
., 45, 64, 138  
.", 61, 67, 140  
.(, 68, 140  
.attributes, 8, 9, 105  
.bp, 91, 146  
.breakpoint, 92, 147  
.calls, 61, 136  
.d, 43, 61, 65, 138  
.enet-addr, 21, 121  
.fregisters, 90, 146

---

.h, 61, 65, 138  
.idprom, 21, 121  
.instruction, 92, 147  
.locals, 90, 146  
.mcr, 133  
.mfsr, 133  
.psr, 90, 146  
.r, 64, 138  
.registers, 90, 146  
.s, 64, 138  
.sfsr, 133  
.sipr, 133  
.step, 92, 147  
.traps, 21, 121  
.version, 21, 121  
.window, 90, 91, 146  
/, 51, 125  
/c, 127  
/c\*, 127  
/L, 127  
/L\*, 127  
/mod, 51, 126  
/n, 127  
/n\*, 127  
/w, 127  
/w\*, 127  
:, 49, 50, 59, 125, 135  
;, 49, 59, 125, 135  
<, 74, 142  
<#, 139  
<<, 52, 126  
<=, 74, 142  
<>, 74, 142  
<w@, 53, 128  
=, 74, 142  
>, 74, 142  
>=, 74, 142  
>>, 51, 126  
>>a, 51, 126  
>body, 63, 136

>r, 48, 124  
?, 117, 128  
???, 117  
?do, 79, 144  
?dup, 48, 124  
?go, 83, 145  
?leave, 79, 144  
@, 53, 54, 60, 61, 128  
[, 63, 136  
[], 61, 136  
[compile], 63, 137  
\, 66, 139  
], 63, 136  
|, 117  
~., 99  
“, 105  
, 61, 136

## Numerics

0<, 74, 142  
0<=, 74, 142  
0<>, 74, 142  
0=, 74, 75, 143  
0>, 74, 143  
0>=, 74, 143  
1-, 51, 126  
1+, 51, 126  
2-, 51, 126  
2!, 54, 128  
2\*, 51, 126  
2+, 51, 126  
2/, 51, 126  
2@, 54, 128  
2constant, 59, 135  
2drop, 48, 124  
2dup, 48, 124  
2over, 48, 124  
2rot, 48, 124  
2swap, 48, 125  
3drop, 48, 125

---

3dup, 48, 125  
n, 118

## A

abort, 81, 144  
abort", 81, 144  
abs, 51, 126  
acf, 117  
adr, 117  
adr16, 117  
adr32, 117  
adr64, 117  
aerr!, 134  
aerr@, 134  
again, 77, 144  
alias, 59, 135  
aligned, 51, 126  
alloc-mem, 56, 105, 129  
allot, 63, 136  
alternate address space commands, 134  
and, 51, 126  
arithmetic functions, 51, 125  
ascii, 66, 139  
ASCII constants, 141  
assembly language commands, 137  
auto-boot?, 23, 35, 121  
aux!, 133, 134  
aux@, 133, 134  
averr!, 134  
averr@, 134

## B

b (boot), 29, 30  
banner, 21, 31, 36, 121  
base, 65, 138  
baud rate, 25, 34  
begin, 77, 144  
begin loops, 77  
bell, 141

between, 74, 143  
binary, 138  
binary executable programs, 85, 87, 88  
bl, 68, 140  
blank, 54, 128  
bljoin, 51, 126  
body>, 63, 137  
boot, 37, 83, 145  
boot command options, 14, 120  
boot -h, 92  
boot-device, 23, 35, 105, 121  
boot-file, 23, 35, 105, 121  
boot-from, 23, 121  
boot-from-diag, 23, 121  
booting failures, 111 to 115  
bounds, 51, 126  
-bp, 91, 146  
--bp, 91, 146  
bpoft, 92, 147  
breakpoint commands, 91, 146  
bs, 141  
buffer:, 59, 135  
building bootable floppy disks, 101  
bwjoin, 51, 126  
byte b, 117  
byte-load, 83, 145

## C

c (continue), 29, 30  
c!, 54, 55, 128  
c., 63, 136  
c;, 138  
c?, 128  
c@, 54, 79, 128  
ca+, 127  
ca1+, 127  
cache manipulation commands, 131  
cacheable, 130  
cache-off, 131  
cache-on, 131

---

call opcode, 89  
callback, 148  
carriage-return, 67  
case, 76, 143  
catch, 148  
cd, 8, 105, 119  
cdata!, 131  
cdata@, 131  
changing the number base, 138  
char, 118  
clear, 48, 125  
clear\_colormap, 21  
clear-cache, 131  
cmove, 54, 128  
cmove>, 54, 128  
cnt, 118  
code, 137  
colon definitions, 49  
command completion, 105  
command line editor, 71 to 73  
command security mode, 28  
comments in Forth code, 66  
comp, 54, 128  
comparison commands, 142  
compile, 63, 137  
compiling data into the dictionary, 136  
configuration parameter primitives, 123  
configuration parameters  
  displaying, 25  
  resetting to defaults, 25  
  setting, 25, 27  
constant, 59, 60, 135  
context!, 134  
context@, 134  
conversion operators, 127  
count, 68, 140  
cpeek, 54, 105, 128  
cpoke, 54, 105, 128  
CPU data register, 90  
cr, 67, 140

create, 59, 135  
creating  
  custom banner, 31  
  dictionary entries, 59  
  new commands, 49  
  new logo, 32  
ctag!, 131  
ctag@, 131  
ctrace, 90, 146  
cxr!, 133  
cxr@, 133

## D

d-, 51, 52  
d!, 105, 128  
d#, 65  
d+, 51  
d?, 105, 128  
d@, 105, 128  
dcontext@, 134  
debug, 93, 148  
debug-off, 93, 148  
decimal, 43, 65, 138  
default values, 25  
defer, 59, 61, 135  
defining words, 59, 135  
depth, 48, 125  
determining SCSI devices, 16, 120  
devalias, 7, 118  
device  
  aliases, 7, 15, 107  
  node characteristics, 4  
  path names, 5  
  tree display/traversal, 8, 119  
device-end, 8, 22, 119  
device-specifier, 14, 16  
diag-device, 35, 106  
diag-file, 24, 35, 106, 121  
diagnostic  
  boot from device, 35  
  boot from file, 35



---

routines, 16  
 switch setting, 35  
 diagnostic test commands, 16, 120  
 diag-switch?, 24, 35, 121  
 dictionary of commands, 59  
 digit, 139  
 dis, 89, 145  
 disassembler commands, 89, 145  
 displaying current parameter settings, 26  
 displaying registers, 90  
 dl, 83, 145  
 dllbin, 83, 145  
 dload, 83, 92, 145  
 do, 79, 144  
 do loops, 79  
 does>, 59, 135  
 drop, 48, 125  
 dump, 42, 54, 55, 128  
 dup, 48, 49, 125

**E**

editing NVRAMRC contents, 37  
 eeprom utility, 28, 32  
 eject-floppy, 18, 148  
 else, 75, 143  
 emergency keyboard commands, 111, 149  
 emit, 67, 140  
 enable!, 134  
 enable@, 134  
 endcase, 76, 143  
 end-code, 138  
 endof, 76, 143  
 erase, 128  
 Ethernet  
   displaying the address, 21  
   testing the controller, 19  
 eval, 81, 83, 144, 145  
 execute, 81, 145  
 exit, 81, 145  
 exit?, 67, 140

expect, 66, 139

**F**

fakeboot, 86  
 false, 74, 143  
 FCode interpreter, 2  
 FCode programs, 84, 86, 88  
 fcode-debug?, 24, 121  
 field, 59, 135  
 file loading commands, 83, 145  
 fill, 128  
 find, 62, 136  
 finish-loop, 92, 147  
 firmware-version, 148  
 flag, 73, 118  
 flip, 52, 126  
 flush-cache, 131  
 forget, 63, 137  
 formatted output commands, 140  
 Forth  
   command format, 41  
   programs, 84, 86, 87  
   reference material, xix  
   Source-level Debugger, 93, 147  
 forth, 148  
 Forth Monitor, 3  
 frame buffer, 69  
 free-mem, 56, 129  
 free-virtual, 56, 129  
 ftrace, 94, 148  
 full security mode, 30

**G**

get-msecs, 148  
 go, 37, 83, 91, 92, 93, 145, 147  
 gos, 92, 147

**H**

h#, 65, 138  
 hardware-revision, 24, 121

---

help, 10, 119  
here, 63, 137  
hex, 43, 65, 138  
history mechanism, 71  
hold, 139  
hop, 92, 147  
hops, 92, 147

## I

i, 79, 80, 144  
ID PROM, 21  
if, 75, 143  
immediate, 63, 137  
init-program, 84, 145  
input, 69, 141  
input devices, 33  
input-device, 24, 32, 69, 121  
interrupt-enable!, 133, 134  
interrupt-enable@, 133, 134  
io, 69, 70, 141  
iomap?, 130  
iomap-page, 130  
iomap-pages, 130  
iommu-ctl!, 133  
iommu-ctl@, 133  
iopgmap!, 130  
iopgmap@, 130  
is, 137

## J

j, 79, 144  
jmp opcode, 89

## K

key, 66, 139  
key?, 66, 67, 79, 100, 139  
keyboard, 33, 70  
keyboard-click?, 24, 121  
keymap, 24, 121

## L

L!, 128  
ll, 54  
L., 136  
l., 63  
L?, 129  
L@, 129  
l@, 53, 54  
La+, 127  
La1+, 127  
label, 138  
last-hardware-update, 24, 121  
lbsplit, 52, 126  
lcc, 68, 140  
leave, 79, 144  
left-parse-string, 68, 141  
len, 118  
lflips, 54, 129  
line editor commands, 71, 141  
literal, 63, 137  
load, 84, 145  
load-base, 84, 145  
loading/executing files  
    FCode/Binary over serial port A, 88  
    Forth over serial port A, 87  
    over Ethernet, 84  
    over hard disk/floppy/Ethernet, 85  
local-mac-address?, 24, 121  
long L, 118  
loop, 79, 144  
loops  
    conditional, 77  
    counted, 79  
lpeek, 54, 106, 129  
lpoke, 54, 106, 129  
ls, 8, 106, 119  
lwsplit, 52, 126

## M

manipulating text strings, 140

---

map?, 129  
map-page, 130  
map-pages, 130  
map-region, 130  
map-regions, 130  
map-segments, 130  
max, 52, 126  
mcr!, 133  
mcr@, 133  
memmap, 130  
memory  
    accessing, 53, 128  
    mapping primitives, 130  
    testing, 35  
mfg-switch?, 24, 35, 121  
mfsr!, 133  
mfsr@, 133  
min, 52, 126  
miscellaneous operations, 148  
mod, 52, 126  
module-info, 149  
move, 54, 129  
ms, 148  
msafar@, 133  
msafsrl, 133  
msafsrl@, 133  
multiprocessor commands, 149

## N

n, 118  
n (enter Forth Monitor), 29, 30  
na+, 127  
na1+, 127  
negate, 52, 126  
nip, 48, 125  
nodefault-bytes, 106, 123  
noop, 148  
noshowstack, 44, 64  
not, 52, 126  
null modem cable, 98

number display, 138  
numeric output primitives, 139  
nvalias, 37, 106, 123  
nvedit, 37, 39, 123  
nvedit keystroke commands, 38, 124  
nvquit, 37, 123  
NVRAM, 23  
NVRAMRC  
    availability, 106  
    editor commands, 37, 123  
    nvramrc command, 24, 36, 121  
nvrecover, 37, 123  
nvrund, 37, 124  
nvstore, 37, 124  
nvunalias, 37, 106, 124

## O

o#, 65  
obio, 130  
obmem, 130  
octal, 43, 65  
oem-banner, 24, 30, 121  
oem-banner?, 24, 30, 32, 121  
oem-logo, 24, 30, 31, 122  
oem-logo?, 24, 30, 32, 122  
of, 76, 143  
off, 54, 129  
old-mode, 4, 148  
on, 54, 129  
or, 52, 126  
origin, 63, 137  
output, 69, 141  
output devices, 33  
output-device, 24, 32, 69, 122  
over, 48, 125

## P

p", 68, 141  
pack, 68, 141  
pagesize, 131

---

parentheses, 66, 139  
password, 29, 37  
patch, 63, 106, 137  
pgmap!, 131  
pgmap?, 131  
pgmap@, 131  
phys, 118  
physical address, 53  
pick, 48, 125  
plug-in device drivers, 1  
power cycle, 41, 69  
power-on banner, 21, 31  
power-on initialization sequence, 109  
printenv, 25, 26, 123  
probe-scsi, 11, 16, 17, 120  
probe-scsi-all, 16, 17, 106, 120  
program counter, 91  
program execution control  
  commands, 144  
PROM version and date, 21  
prompt, 76  
pstr, 118  
pwd, 8, 106, 119

## Q

quit, 81, 145

## R

r>, 48, 125  
r@, 48, 125  
ramforth, 149  
reading/writing registers  
  Sun-4C machines, 134  
  Sun-4D machines, 132  
  Sun-4M machines, 133  
recursive, 64, 137  
redirecting input/output, 141  
repeat, 78, 144  
reset, 11, 21, 37, 148  
resetting

  parameter defaults, 27  
  the system, 21  
restoring color tables, 21  
Restricted Monitor commands, 3, 118  
resume, 93, 148  
return, 92, 147  
returnL, 92, 147  
rmap!, 131  
rmap@, 131  
roll, 48, 125  
romforth, 149  
-rot, 48, 124  
rot, 48, 125  
running extended diagnostics, 35

## S

saving data after a system crash, 111  
sbus, 130  
sbus-probe-list, 24, 122  
screen, 33, 70  
screen-#columns, 24, 32, 122  
screen-#rows, 24, 32, 122  
scsi-initiator-id, 24, 122  
sd-targets, 24, 122  
searching the dictionary, 136  
security  
  command, 28  
  full, 30  
  none, 28  
  password, 29  
security-#badlogins, 24, 28, 122  
security-mode, 24, 27, 122  
security-password, 24, 28, 122  
see, 46, 62, 136  
segmentsize, 131  
selftest-#megs, 24, 35, 122  
serial ports, 33, 34, 69  
serr!, 134  
serr@, 134  
set-default, 25, 27, 123

---

set-defaults, 25, 27, 123  
 setenv, 25, 27, 123  
 setenv security-mode exception, 37  
 set-pc, 90, 91, 146  
 setting  
     default input/output devices, 33  
     firmware security, 27  
     security password, 29  
 sfar!, 133  
 sfar@, 133  
 sfsr!, 133  
 sfsr@, 133  
 show, 123  
 show-devs, 8, 9, 106, 119  
 show-sbus, 21, 106, 121  
 showstack, 44, 64, 106, 138  
 sift, 62, 136  
 sifting, 62, 136  
 sign, 139  
 size, 118  
 skip, 92, 147  
 skip-vme-loopback?, 24, 122  
 smap!, 131  
 smap?, 131  
 smap@, 131  
 Space, 93, 148  
 space, 67, 140  
 spacec!, 134  
 spacec?, 134  
 spacec@, 134  
 spaced!, 134  
 spaced?, 106, 134  
 spaced@, 134  
 spaceL!, 135  
 spaceL?, 135  
 spaceL@, 135  
 spaces, 67, 140  
 spacew!, 135  
 spacew?, 135  
 spacew@, 135  
 span, 66, 139  
 SPARC register commands, 90, 146  
 specifying auto-booting from Ethernet, 35  
 stack  
     description, 44  
     diagram, 45  
     item notation, 117  
     manipulation commands, 124  
 state, 64, 137  
 step, 92, 147  
 stepping, 93, 148  
 steps, 92, 147  
 Stop, 111, 149  
 Stop-A, 70, 90, 111, 149  
 Stop-D, 36, 106, 111, 149  
 Stop-F, 106, 111, 149  
 Stop-N, 106, 111, 149  
 strings, manipulating, 140  
 struct, 59, 135  
 st-targets, 24, 122  
 sunmon-compat?, 24, 122  
 sverr!, 134  
 sverr@, 134  
 swap, 48, 125  
 switch-cpu, 149  
 symbol table, 90  
 sync, 11, 111, 149  
 system configuration parameters, *See*  
     configuration parameters  
 system information display  
     commands, 121

**T**

terminal, 69  
 test, 16, 107, 120  
 test-all, 16, 120  
 testarea, 24, 122  
 testing  
     clock, 16, 20, 120  
     diskette drive, 16, 18, 120  
     memory, 16, 18, 120

---

network connection, 16, 19, 120  
SBus devices, 16, 120  
text input commands, 66, 139  
text output commands, 67, 140  
then, 75, 143  
throw, 149  
till, 92, 147  
time utilities, 148  
TIP problems, 99  
TIP window, 97  
to, 32, 63, 90, 146  
Tokenizer, 86  
tpe-link-test?, 24, 122  
tracing, 93, 148  
-trailing, 68, 140  
true, 74, 143  
ttya, 33, 70  
ttya-ignore-cd, 25, 122  
ttya-mode, 25, 32, 34, 122  
ttya-rts-dtr-off, 25, 122  
ttyb, 33, 70  
ttyb-ignore-cd, 25, 122  
ttyb-mode, 25, 32, 34, 122  
ttyb-rts-dtr-off, 25, 122  
type, 67, 140

## U

u\*x, 126  
u., 64, 138  
u.r, 64, 138  
u/mod, 52, 126  
u<, 74, 143  
u<=, 74, 143  
u>, 74, 143  
u>=, 74, 143  
u2/, 52, 126  
um\*, 52  
unaligned-L!, 54, 129  
unaligned-L@, 55, 129  
unaligned-w!, 55, 129

unaligned-w@, 55, 129  
until, 78, 144  
upc, 68, 141  
use-nvramrc?, 25, 36, 122  
User Interface  
    command line editor, 71 to 73

## V

value, 59, 60, 135  
variable, 59, 61, 135  
version2?, 25, 122  
virt, 118  
virtual address, 53

## W

w, 90, 91, 146  
w!, 55, 129  
w., 63, 136  
w?, 129  
w@, 53, 55, 129  
wa+, 127  
wal+, 127  
watch-clock, 16, 20, 120  
watchdog-reboot?, 25, 122  
watch-net, 16, 20, 107, 120  
ways to enter Forth Monitor, 2  
wbflip, 52  
wbsplit, 52, 127  
wflip, 127  
wflips, 55, 129  
while, 78, 144  
within, 74, 143  
wljoin, 52, 127  
word, 66, 118, 139  
words, 8, 10, 42, 62, 119, 136  
wpeek, 55, 107, 129  
wpoke, 55, 107, 129

---

## **X**

x-, 127

x+, 127

xor, 52, 127

xu/mod, 127





### Reader Comment Card

Your comments and suggestions are important to us. Please let us know what you think about the *OpenBoot 2.x Command Reference*, part number 802-3241-10.

1. Were the procedures well documented? Yes  No

Please explain: \_\_\_\_\_

\_\_\_\_\_

2. Were the tasks easy to follow? Yes  No

Please explain: \_\_\_\_\_

\_\_\_\_\_

4. Was the information complete and easy to find? Yes  No

Please explain: \_\_\_\_\_

\_\_\_\_\_

5. Do you have additional comments about the OpenBoot 2.x Command Reference?

You can send detailed comments via email to [smcc-docs@sun.com](mailto:smcc-docs@sun.com), or send a fax to *SMCC Doc Feedback* at (415) 786-6443.

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Your Name: \_\_\_\_\_

Title: \_\_\_\_\_

Company Name: \_\_\_\_\_

Address: \_\_\_\_\_

City: \_\_\_\_\_ State/Province: \_\_\_\_\_

Country: \_\_\_\_\_ Zip/Postal Code: \_\_\_\_\_

Email Address: \_\_\_\_\_

Telephone: \_\_\_\_\_

**AIR MAIL  
PAR AVION**

IBRS/CCRI No. 808



**INTERNATIONAL BUSINESS REPLY MAIL/REPOSE PAYEE**

PERMIT NO. 808 MOUNTAIN VIEW CA

POSTAGE WILL BE PAID BY ADDRESSEE

**INFORMATION PRODUCTS  
M/S MPK14-108  
SUN MICROSYSTEMS INC  
2550 GARCIA AVE  
MOUNTAIN VIEW CA 94043-9551  
UNITED STATES OF AMERICA**

NE PAS AFFRANCHIR

**NO POSTAGE  
NECESSARY  
IF MAILED  
TO THE  
UNITED STATES**



Copyright 1995 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100 U.S.A.

Tous droits réservés. Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduits sous aucune forme, par quelque moyen que ce soit sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Des parties de ce produit pourront être dérivées du système UNIX® et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays, et licenciée exclusivement par X/Open Company Ltd. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, et Solaris sont des marques déposées ou enregistrées par Sun Microsystems, Inc. aux Etats-Unis et dans certains d'autres pays. Toutes les marques SPARC, utilisées sous license, sont des marques déposées ou enregistrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Les utilisateurs d'interfaces graphiques OPEN LOOK® et Sun™ ont été développés par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique, cette licence couvrant aussi les licenciés de Sun qui mettent en place OPEN LOOK GUIs et qui en outre se conforment aux licences écrites de Sun.

Le système X Window est un produit du X Consortium, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A REpondre A UNE UTILISATION PARTICULIERE OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS



