OpenBoot 2.x Command Reference Manual

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto,CA 94303-4900
U.S.A. 650-960-1300

Part No. 806-2906-10
February 2000, Revision A

Send comments about this document to: docfeedback@sun.com

Please
Recycle

Contents

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*

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*
- 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.

Sun Welcomes Your Comments

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

- Email: `docfeedback@sun.com`

CHAPTER 1

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, 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 the following table.

TABLE 1-1 Restricted Monitor Commands

Command	Description
`b` [*specifiers*]	Boot the operating system.
`c`	Resume the execution of a halted program.
`n`	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) and `go` (see Chapter 5) 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, 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

The following table describes each of these parameters.

TABLE 1-2 Device Path Name Parameters

Path Name Parameter	Description
name	A text string that, ideally, has some mnemonic value. (For example, `sd` represents "SCSI disk".) Many names, especially names of plug-in modules, include the name or stock symbol of the device's manufacturer (for example, `SUNW,esp`).
`@`	Must precede the *address* parameter.
address	A text string representing an address, usually of the form *hex_number*,*hex_number*. (Numbers are given in hexadecimal format.)
`:`	Must precede the *arguments* 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.

The following table describes the `devalias` command, which is used to examine, create, and change aliases.

TABLE 1-3 Examining and Creating Device Aliases

Command	Description
`devalias`	Display all current device aliases.
`devalias` *alias*	Display the device path name corresponding to *alias*.
`devalias` *alias device-path*	Define an alias representing *device path*. 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, 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 the following table.

TABLE 1-4 Commands for Browsing the Device Tree

Command	Description
`.attributes`	Display the names and values of the current node's properties.
`cd` *device-path*	Select the indicated device node, making it the current node.
`cd` *node-name*	Search for a node with the given name in the subtree below the current node, and select the first such node found.
`cd ..`	Select the device node that is the parent of the current node.
`cd /`	Select the root machine node.
`device-end`	De-select the current device node, leaving no node selected.

TABLE 1-4 Commands for Browsing the Device Tree

Command	Description
`ls`	Display the names of the current node's children.
`pwd`	Display the device path name that names the current node.
`show-devs` [*device-path*]	Display all the devices known to the system directly beneath a given level in the device hierarchy. `show-devs` used by itself shows the entire device tree.
`words`	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
/eeprom@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 the following table.

TABLE 1-5 Help Commands

Command	Description
`help`	List main help categories.
`help` *category*	Show help for all commands in the category. Use only the first word of the category description.
`help` *command*	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. 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
```

CHAPTER 2

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 the following table.

TABLE 2-1 Common Options for the `boot` Command

Parameter	Description
[*device-specifier*]	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)
[*filename*]	The name of the program to be booted (for example, `stand/diag`). *filename* is relative to the root of the selected device and partition (if specified). If *filename* is not specified, the boot program uses the value of the `boot-file` NVRAM parameter (see Chapter 3).
[*options*]	`-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, for definitions of your system's aliases the following table 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 2-2 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`.
`cdroma`	`/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 the following table 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 below.

TABLE 2-3 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.

The following table lists diagnostic test commands. Remember: *device-specifier* refers to either a device path name or a device alias.

TABLE 2-4 Diagnostic Test Commands

Command	Description
`probe-scsi`	Identify devices attached to the built-in SCSI bus.
`probe-scsi-all` [*device-path*]	Perform `probe-scsi` on all SCSI buses installed in the system below the specified device tree node. (If *device-path* is absent, the root node is used.)
`test` *device-specifier*	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 `selftest-#megs` NVRAM parameter; or test all of memory if `diag-switch?` is `true` `test net` - test the network connection

TABLE 2-4 Diagnostic Test Commands *(Continued)*

Command	Description
`test-all` [*device-specifier*]	Test all devices (that have a built-in self-test method) below the specified device tree node. (If *device-specifier* is absent, the root node is used.)
`watch-clock`	Test the clock function.
`watch-net`	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, 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 the following table, 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 2-5 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.

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 before you reset the system.

CHAPTER 3

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, for information about entering the Forth Monitor.

TABLE 3-1 lists current NVRAM configuration parameters.

TABLE 3-1 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.

TABLE 3-1 NVRAM Configuration Parameters *(Continued)*

Parameter	Typical Default	Description
`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`.
`nvramrc`	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`).

TABLE 3-1 NVRAM Configuration Parameters *(Continued)*

Parameter	Typical Default	Description
`security-password`	no default	Firmware security password (never displayed). *Do not set this directly*.
`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.

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 3-2.

TABLE 3-2 Viewing/Changing Configuration Parameters

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

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

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

             Hello Mom and Dad

ok
```

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

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 `eeprom` 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 3-3.

TABLE 3-3 I/O Device Parameters

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

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:

- `nvramrc`
- `use-nvramrc?`

Commands in NVRAMRC are executed during system start-up if `use-nvramrc?` 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 3-4.

TABLE 3-4 NVRAMRC Editor Commands

Command	Description
`nvalias` *alias device-path*	Store the command "`devalias` *alias device-path*" 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 them.
`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.
`nvrun`	Execute the contents of the temporary buffer.
`nvstore`	Copy the contents of the temporary buffer to NVRAMRC; discard the contents of the temporary buffer.
`nvunalias` *alias*	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 3-5 are used within the NVRAM

TABLE 3-5 `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 `nvstore` 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.

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 `nvrun` 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.

CHAPTER 4

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 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 4-1 for stack item abbreviations used in this manual.

TABLE 4-1 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 b`*xxx*	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.
`d`*xxx*	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).
`<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.

TABLE 4-1 Stack Item Notation *(Continued)*

Notation	Description
`quad q`*xxx*	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).
`u`*xxx*	Unsigned value, positive values (32-bit).
`virt`	Virtual address (address used by software).
`waddr`	Doublet (16-bit) aligned address
`word w`*xxx*	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 4-2) allow you to add, delete, and reorder items on the stack.

TABLE 4-2 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`).

TABLE 4-2 Stack Manipulation Commands *(Continued)*

Command	Stack Diagram	Description
`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 4-3 shows the Forth words used to create custom definitions.

TABLE 4-3 Color Definition Words

Command	Stack Diagram	Description
: *new-name*	(--)	Start a new colon definition of the word *new-name*.
;	(--)	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, 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 4-4 perform basic arithmetic with items on the data stack.

TABLE 4-4 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 `lshift`.
>>	(x1 u -- x2)	Synonym for `rshift`.
>>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.

TABLE 4-4 Arithmetic Functions *(Continued)*

Command	Stack Diagram	Description
`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 *startaddr len* to *endaddr startaddr* for `do` 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 >= *n*.
`fm/mod`	(d n -- rem quot)	Divide *d* by n.
`invert`	(x1 -- x2)	Invert all bits of *x1*.
`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 *x1* by *u* bits. Zero-fill low bits.
`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`	(x1 -- x2)	Synonym for `invert`.
`or`	(n1 n2 -- n3)	Bitwise logical OR.
`rshift`	(x1 u -- x2)	Right-shift *x1* by *u* bits. Zero-fill high bits.
`s>d`	(n1 -- d1)	Convert a number to a double number.

TABLE 4-4 Arithmetic Functions *(Continued)*

Command	Stack Diagram	Description
`sm/rem`	(d n -- rem quot)	Divide d by *n*, 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 ud by *u*.
`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 4-5 lists the commands used to access memory.

TABLE 4-5 Memory Access Commands

Command	Stack Diagram	Description
`!`	(x a-addr --)	Store a number at *a-addr*.
`+!`	(nu a-addr --)	Add *nu* to the number stored at *a-addr*.
`<w@`	(waddr -- n)	Fetch doublet *w* from *waddr*, sign-extended.
`@`	(a-addr --x)	Fetch a number from *a-addr*.
`2!`	(x1 x2 a-addr --)	Store 2 numbers at *a-addr*, *x2* at lower address.
`2@`	(a-addr -- x1 x2)	Fetch 2 numbers from *a-addr*, *x2* from lower address.
`blank`	(addr len --)	Set *len* bytes of memory beginning at *addr* to `space` (decimal 32).
`c!`	(byte addr --)	Store *byte* at *addr*.
`c@`	(addr -- byte)	Fetch a *byte* from *addr*.
`cmove`	(addr1 addr2 u --)	Copy *u* bytes from *addr1* to *addr2*, starting at low byte.
`cmove>`	(addr1 addr2 u --)	Copy *u* bytes from *addr1* to *addr2*, starting at high byte.

TABLE 4-5 Memory Access Commands *(Continued)*

Command	Stack Diagram	Description
`cpeek`	(addr -- false | byte true)	Fetch the byte at *addr*. Return the data and `true` if the access was successful. Return `false` if a read access error occurred.
`cpoke`	(byte addr -- okay?)	Store the *byte* to *addr*. Return `true` if the access was successful. Return `false` if a write access error occurred.
`comp`	(addr1 addr2 len -- diff?)	Compare two byte arrays. *diff?* = 0 if arrays are identical, *diff?* = -1 if first byte that is different is lesser in string at *addr1*, *diff?* = 1 otherwise.
`dump`	(addr len --)	Display *len* bytes of memory starting at *addr*.
`erase`	(addr len --)	Set *len* bytes of memory beginning at *addr* to `0`.
`fill`	(addr len byte --)	Set *len* bytes of memory beginning at *addr* to the value *byte*.
`l!`	(n qaddr --)	Store a quadlet *q* at *qaddr*.
`l@`	(qaddr -- quad)	Fetch a quadlet *q* from *qaddr*.
`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 *qaddr*. Return the data and `true` if the access was successful. Return `false` if a read access error occurred.
`lpoke`	(quad qaddr -- okay?)	Store the 32-bit quantity at *qaddr*. Return `true` if the access was successful. Return `false` if a a write access error occurred.
`move`	(src-addr dest-addr len --)	Copy *len* bytes from *src-addr* to *dest-addr*.
`off`	(a-addr --)	Store `false` at *a-addr*.
`on`	(a-addr --)	Store `true` at *a-addr*.
`unaligned-l!`	(quad addr --)	Store a quadlet *q*, any alignment

TABLE 4-5 Memory Access Commands *(Continued)*

Command	Stack Diagram	Description
`unaligned-l@`	(addr -- quad)	Fetch a quadlet *q*, any alignment.
`unaligned-w!`	(w addr --)	Store a doublet *w*, any alignment.
`unaligned-w@`	(addr -- w)	Fetch a doublet *w*, any alignment.
`w!`	(w waddr --)	Store a doublet *w* at *waddr*.
`w@`	(waddr -- w)	Fetch a doublet *w* from *waddr*.
`wbflips`	(waddr len --)	Swap the bytes within each doublet in the specified region.
`wpeek`	(waddr -- false | w true)	Fetch the 16-bit quantity at *waddr*. Return the data and `true` if the access was successful. Return `false` if a read access error occurred.
`wpoke`	(w waddr -- okay?)	Store the 16-bit quantity to *waddr*. Return `true` if the access was successful. Return `false` 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 4-6 lists memory mapping commands.

TABLE 4-6 Memory Mapping Commands

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

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 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 4-7 lists the defining words that you can use to create dictionary entries.

TABLE 4-7 Defining Words

Command	Stack Diagram	Description
: *name*	(--) Usage: (??? -- ?)	Start creating a new colon definition.
;	(--)	Finish creating a new colon definition.
`alias` *new-name old-name*	(--) Usage: (??? -- ?)	Create *new-name* with the same behavior as *old-name*.
`buffer:` *name*	(size --) Usage: (-- a-addr)	Create a named array in temporary storage.
`constant` *name*	(n --) Usage: (-- n)	Define a constant (for example, `3 constant bar`).
`2constant` *name*	(n1 n2 --) Usage: (-- n1 n2)	Define a 2-number constant.
`create` *name*	(--) Usage: (-- waddr)	Generic defining word.
`defer` *name*	(--) Usage: (??? -- ?)	Define a word for forward references or execution vectors using execution token.
`does>`	(-- waddr)	Start the run-time clause for defining words.
`field` *name*	(offset size -- offset+size) Usage: (addr -- addr+offset)	Create a named offset pointer.

TABLE 4-7 Defining Words *(Continued)*

Command	Stack Diagram	Description
`struct`	(-- 0)	Initialize for `field` creation.
`value` *name*	(n --) Usage: (-- n)	Create a changeable, named 32-bit quantity.
`variable` *name*	(--) 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 4-8 lists tools you can use to search the dictionary.

TABLE 4-8 Dictionary Searching Commands

Command	Stack Diagram	Description
`'` *name*	(-- xt)	Find the named word in the dictionary. Returns the execution token. Use outside definitions.
`[']` *name*	(-- xt)	Similar to `'` but is used either inside or outside definitions.
`.calls`	(xt --)	Display a list of all words that call the word whose execution token is *xt*.
`$find`	(addr len -- addr len false | xt n)	Find a word. *n* = 0 if not found, *n* = 1 if immediate, *n* = -1 otherwise.
`find`	(pstr -- pstr false | xt n)	Search for a word in the dictionary. The word to be found is indicated by *pstr*. *n* = 0 if not found, *n* = 1 if immediate, *n* = -1 otherwise.
`see` *thisword*	(--)	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 *pstr*.
`sifting` *ccc*	(--)	Display names of all dictionary entries containing the sequence of characters. *ccc* 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 4-9 control the compilation of data into the dictionary.

TABLE 4-9 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 *n* 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]` *name*	(--)	Compile the next (immediate) word.
`forget` *namep*	(--)	Remove word from dictionary and all subsequent words.
`here`	(-- addr)	Address of top of dictionary.
`immediate`	(--)	Mark the last definition as immediate.

TABLE 4-9 Dictionary Compilation Commands *(Continued)*

Command	Stack Diagram	Description
`to` *name*	(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` *new-word old-word word-to-patch*	(--)	Replace *old-word* with *new-word* in *word-to-patch*.
`(patch`	(new-n old-n xt --)	Replace *old-n* with *new-n* in word indicated by *xt*.
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 4-10 shows basic commands to display stack values.

TABLE 4-10 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 4-11.

TABLE 4-11 Changing the Number Base

Command	Stack Diagram	Description
`.d`	(n --)	Display *n* in decimal without changing base.
`.h`	(n --)	Display *n* in hex without changing base.
`base`	(-- addr)	Variable containing number base.
`decimal`	(--)	Set the number base to 10.
`d#` *number*	(-- n)	Interpret *number* in decimal; base is unchanged.
`hex`	(--)	Set the number base to 16.
`h#` *number*	(-- n)	Interpret *number* in hex; base is unchanged.
`octal`	(--)	Set the number base to 16.
`o#` *number*	(-- n)	Interpret *number* 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 4-12 lists commands to control text input.

TABLE 4-12 Controlling Text Input

Command	Stack Diagram	Description
`(` *ccc* `)`	(--)	Begin a comment. Conventionally used for stack diagrams.
`\` *rest-of-line*	(--)	Treat the rest of the line as a comment.
`ascii` *ccc*	(-- 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 *addr*.
`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 `expect`.
`word`	(char -- pstr)	Collect a string delimited by *char* from input string and place in memory at *pstr*.

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 4-13 lists general-purpose text display commands.

TABLE 4-13 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`	(addr +n --)	Display the +*n* characters beginning at *addr*.

`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 4-14 shows commands used to manipulate text strings.

TABLE 4-14 Manipulating Text Strings

Command	Stack Diagram	Description
",	(addr len --)	Compile an array of bytes from *addr* of length *len*, 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 *char* (which is discarded).
`pack`	(addr len pstr -- pstr)	Make a packed string from *addr 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.

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 4-15 lists commands you can use to redirect input and output.

TABLE 4-15 I/O Redirection Commands

Command	Stack Diagram	Description
`input`	(device --)	Select device (`keyboard`, or *device-specifier*) for input.
`io`	(device --)	Select device for input and output.
`output`	(device --)	Select device (`screen`, or *device-specifier*) 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 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 4-16 lists the required line-editing commands available at the `ok` prompt.

TABLE 4-16 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 4-17 lists the command line editing extension group.

TABLE 4-17 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.
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.

TABLE 4-17 Optional Command Line Editor Keystroke Commands *(Continued)*

Keystroke	Description
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 4-18 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 4-19 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 `ffffffff`.)

TABLE 4-20 lists commands that perform relational tests, and leave a `true` or `false` flag result on the stack.

TABLE 4-20 Comparison Commands

Command	Stack Diagram	Description
`<`	(n1 n2 -- flag)	True if *n1* < *n2*.
`<=`	(n1 n2 -- flag)	True if *n1* <= *n2*.
`<>`	(n1 n2 -- flag)	True if *n1* is not equal to *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.
`0<=`	(n -- flag)	True if *n* <= 0.
`0<>`	(n -- flag)	True if *n* <> 0.
`0=`	(n -- flag)	True if *n* = 0 (also inverts any flag).
`0>`	(n -- flag)	True if *n* > 0.
`0>=`	(n -- flag)	True if *n* >= 0.
`between`	(n min max -- flag)	True if *min* <= *n* <= *max*.
`false`	(-- 0)	The value `FALSE`, which is 0.
`true`	(-- -1)	The value `TRUE`, which is -1.
`u<`	(u1 u2 -- flag)	True if *u1* < *u2*, unsigned.

TABLE 4-20 Comparison Commands *(Continued)*

Command	Stack Diagram	Description
`u<=`	(u1 u2 -- flag)	True if *u1* <= *u2*, unsigned.
`u>`	(u1 u2 -- flag)	True if *u1* > *u2*, unsigned.
`u>=`	(u1 u2 -- flag)	True if *u1* >= *u2*, unsigned.
`within`	(n min max -- flag)	True if *min* <= *n* < *max*.

`>` 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 4-21 control the flow of conditional execution.

TABLE 4-21 `if..else..then` Commands

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

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 4-22 lists the conditional case commands.

TABLE 4-22 `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.

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 4-23 lists commands to control the execution of conditional loops.

TABLE 4-23 `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.

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 4-24 lists commands to control the execution of counted loops.

TABLE 4-24 `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)	Leaves the loop index on the stack.

TABLE 4-24 do (Counted) Loop Commands

Command	Stack Diagram	Description
j	(-- n)	Leaves the loop index of the next outer enclosing loop on the stack.
leave	(--)	Exit from do…loop.
loop	(--)	End of do…loop.

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 4-1 contains descriptions of additional program execution control commands.

TABLE 4-25 Program Execution Control Commands

Command	Stack Diagram	Description
`abort`	(--)	Abort current execution and interpret keyboard commands.
`abort"` *ccc*"	(abort? --)	If *abort?* is true, abort and display message.
`eval`	(addr len --)	Interpret Forth source from an array.
`execute`	(xt --)	Execute the word whose execution token is on the stack.
`exit`	(--)	Return from the current word. (Cannot be used in counted loops.)
`quit`	(--)	Same as `abort`, 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, then `eval` can be used to compile the definitions contained in the file.

CHAPTER 5

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 5-1 lists commands for loading files from various sources.

TABLE 5-1 File Loading Commands

Command	Stack Diagram	Description
`?go`	(--)	Execute Forth, FCode or binary programs.
`boot` *[specifiers]* `-h`	(--)	Load file from specified source.
`byte-load`	(addr span --)	Interpret loaded FCode binary file. *span* is usually 1.
`dl`	(--)	Load a Forth file over a serial line with a terminal emulator and interpret. Using `tip` as an example, type:: **`~C cat`** *filename* **`^-D`**
`dlbin`	(--)	Load a binary file over a serial line with a terminal emulator. Using `tip` as an example, type: **`~C cat`** *filename*
`dload` *filename*	(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 5-1 File Loading Commands *(Continued)*

Command	Stack Diagram	Description
`init-program`	(--)	Initialize to execute a binary file.
`load` *device-specifier argument*	(--)	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" 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 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", 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 dl to Load Forth Over a Serial Port

Forth programs loaded with dl 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 dl.

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

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

```
ok dl
```

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 re-appears 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", 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
```

CHAPTER 6

Debugging

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 6-1 lists commands that disassemble memory into equivalent op codes.

TABLE 6-1 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 "Breakpoints" on page 89.) 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 6-2 lists the SPARC register commands.

TABLE 6-2 SPARC Register Commands

Command	Stack Diagram	Description
`%f0` through `%f31`	(-- value)	Return the value in the specified floating point register.
`%fsr`	(-- value)	Return the value in the floating point status register.
`%g0` through `%g7`	(-- value)	Return the value in the specified global register.
`%i0` through `%i7`	(-- value)	Return the value in the specified input register.
`%l0` through `%l7`	(-- value)	Return the value in the specified local register.
`%o0` through `%o7`	(-- value)	Return the value in the specified output register.
`%pc %npc %psr %y %wim %tbr`	(-- value)	Return the value in the specified register.
`.fregisters`	(--)	Display the values in `%f0` through `%f31`.
`.locals`	(--)	Display the values in the `i`, `l` and `o` registers.
`.psr`	(--)	Formatted display of the program status register.
`.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.

TABLE 6-2 SPARC Register Commands *(Continued)*

Command	Stack Diagram	Description
`set-pc`	(new-value --)	Set `%pc` to *new-value*, and set `%npc` to (*new-value*+4).
`to` *regname*	(new-value --)	Change the value stored in any of the registers above. Use in the form: *new-value* `to` *regname*.
`w`	(window# --)	Set the current window for displaying `%i`*x*, `%L`*x*, or `%o`*x*.

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 6-3 lists the breakpoint commands that control and monitor program execution.

TABLE 6-3 Breakpoint Commands

Command	Stack Diagram	Description
`+bp`	(addr --)	Add a breakpoint at the specified address.
`-bp`	(addr --)	Remove the breakpoint at the specified address.
`--bp`	(--)	Remove the most-recently-set breakpoint.
`.bp`	(--)	Display all currently set breakpoints.
`.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 behavior is `.instruction`. To perform multiple behaviors, create a single definition which calls all desired behaviors, 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`).
`bpoff`	(--)	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`	(addr --)	Execute until the given address is encountered. Equivalent to `+bp go`.

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

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

   See Chapter 5 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 6-4.

TABLE 6-4 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.
`debug` *name*	Mark the specified Forth word for debugging. Enter the Forth Source-level Debugger on all subsequent attempts to execute *name*. After executing `debug`, the execution speed of the system may decrease until debugging is turned off with `debug-off`. (Do not debug basic Forth words such as ".".)

TABLE 6-4 Forth Source-level Debugger Commands *(Continued)*

Command	Description
`debug-off`	Turn off the Forth Source-level Debugger so that no word is being debugged.
`resume`	Exit from a subordinate interpreter, and go back to the stepper (see the `F` command in this table).
`stepping`	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.
`tracing`	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.
`<space-bar>`	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 6-4.

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
(ffe8b574) Called from (interpret at ffe8b6f8
execute Called from catch at ffe8a8ba
    ffefeff0
    0
    ffefebdc
catch    Called from (fload) at ffe8ced8
    0
(fload) Called from interact at ffe8cf74
execute Called from catch at ffe8a8ba
    ffefefd4
    0
    ffefebdc
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.

APPENDIX **A**

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.

APPENDIX **B**

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:

- "Procedure for the Pre-Solaris 2.0 Operating Environment" on page 99 describes the procedure for systems using pre-Solaris 2.0 operating environments.
- "Procedure for the Solaris 2.0 or 2.1 Operating Environment" on page 100 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
```

APPENDIX **C**

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
`"` *embedded bytes*	Not supported in earlier systems.	Use other array-creation mechanisms, such as `alloc-mem` and `c, .`
`.attributes`	Not supported until OpenBoot 2.0.	A loadable `showdevs` utility, which provides some of this functionality, is available from the Sun SBus Support Group.
`alloc-mem`	See workaround.	Pre-2.0, size is restricted to total remaining FORTH dictionary space. Using more than several hundred bytes is dangerous. Use `dma-alloc` (size -- virt) instead.
`boot-device` `boot-file`	Not supported until OpenBoot 2.0.	Use `boot-from` to indicate boot device and boot file.
`cd`	Not supported until OpenBoot 2.0.	A loadable `showdevs` 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.

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

Command	Availability	Workaround
`cpeek` `cpoke`	Not supported in early systems.	`probe` words exist in early systems to provide a similar functionality, as: `cprobe` (adr -- ok?) Test for data exception using `c@`.

`d!`
`d?`
`d@`
	Not supported in early systems.	Use combinations of 32-bit accesses.

`diag-device` `diag-file`	Not supported until OpenBoot 2.0.	Use `boot-from-diag` to indicate diagnostic boot device and boot file.
`lpeek` `lpoke`	Not supported in early systems.	`probe` words exist in early systems to provide a similar functionality, as: `lprobe` (adr32 -- ok?) Test for data exception using `l@`.
`ls`	Not supported until OpenBoot 2.0.	A loadable `showdevs` 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.
`nodefault-bytes`	Not supported until OpenBoot 2.0.	No workaround.
`patch`	See workaround.	Pre-2.6, `patch` would patch words but not numbers within definitions. To patch numbers, use : `npatch` *word-to-patch* (new-n old-n --).
`probe-scsi-all`	Not supported until OpenBoot 2.6.	No workaround.
`pwd`	Not supported until OpenBoot 2.0.	A loadable `showdevs` 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 `showdevs` utility, which provides some of this functionality, is available from the Sun SBus Support Group.

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

Command	Availability	Workaround
`show-sbus`	Not supported until OpenBoot 2.3.	Use: `ok` **`cd /sbus`** `ok` **`ls`** (Similar information is presented, but in a different format.)
`showstack`	Does not toggle (turn off) until OpenBoot 2.6.	To turn off showstack, either reset the system or type: `[‘] noop is status`
`spaced?`	Not supported until OpenBoot 2.6.	Use `spaced@` and “`.`”

`Stop-F`
`Stop-D`
`Stop-N`
	Not supported until OpenBoot 2.0.	No workaround.

`test` *xxx*	Not supported until OpenBoot 2.0.	It is possible to test certain devices on OpenBoot 1.*x* systems with: `test-memory ( -- )` (similar to: `test /memory`). 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.
`watch-net`	Not supported in OpenBoot 1.3 through 2.2.	No workaround.
`wpeek` `wpoke`	Not supported in early systems.	`probe` words exist in early systems to provide a similar functionality, as: `wprobe ( adr16 -- ok? )` Test for data exception using `w@`.

APPENDIX **D**

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

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 113.

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:

1. **Unplug all but one of the disks.**
2. **At the `ok` prompt, type:**

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

Note the target number and its corresponding unit number.

3. **Plug in another disk and perform step b again.**
4. **If you get an error, change the target number of this disk to be one of the unused target numbers.**
5. **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
```

APPENDIX **E**

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).
`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.

TABLE E-1 Stack Item Notation *(Continued)*

Notation	Description
`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` [*specifiers*]	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` *alias*	Display the device path name corresponding to alias.
`devalias` *alias device-path*	Define an alias representing *device-path*. 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
`.attributes`	Display the names and values of the current node's properties.
`cd` *device-path*	Select the indicated device node, making it the current node.
`cd` *node-name*	Search for a node with the given name in the subtree below the current node, and select the first such node found.

TABLE E-4 Commands for Browsing the Device Tree

Command	Description
`cd ..`	Select the device node that is the parent of the current node.
`cd /`	Select the root machine node.
`device-end`	De-select the current device node, leaving no node selected.
`ls`	Display the names of the current node's children.
`pwd`	Display the device path name that names the current node.
`show-devs` [*device-path*]	Display all the devices known to the system directly beneath a given level in the device hierarchy. `show-devs` used by itself shows the entire device tree.
`words`	Display the names of the current node's methods.

TABLE E-5 Help Commands

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

TABLE E-6 Common Options for the `boot` Command

Parameter	Description
`boot` [*device-specifier] [filename] [options]*	

TABLE E-6 Common Options for the `boot` Command

Parameter	Description
[*device-specifier*]	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)
[*filename*]	The name of the program to be booted (for example, `stand/diag`). *filename* is relative to the root of the selected device and partition (if specified). If *filename* is not specified, the boot program uses the value of the `boot-file` NVRAM parameter (see Chapter 3).
[*options*]	`-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` [*device-path*]	Perform `probe-scsi` on all SCSI buses installed in the system below the specified device tree node. (If *device-path* is absent, the root node is used.)
`test` *device-specifier*	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 `selftest-#megs` NVRAM parameter; or test all of memory if `diag-switch?` is `true` `test net` - test the network connection
`test-all` [*device-specifier*]	Test all devices (that have a built-in self-test method) below the specified device tree node. (If *device-specifier* 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`.
`nvramrc`	empty	Contents of NVRAMRC.

TABLE E-9 NVRAM Configuration Parameters *(Continued)*

Parameter	Typical Default	Description
`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). *Do not set this directly.*
`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).

TABLE E-9 NVRAM Configuration Parameters *(Continued)*

Parameter	Typical Default	Description
`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` *parameter* shows the current value of the named parameter.
`setenv` *parameter value*	Set *parameter* to the given decimal or text value. (Changes are permanent, but usually only take effect after a reset.)
`set-default` *parameter*	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` *parameter*	(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` *parameter*	(--)	Display the (current) field value (numbers shown in decimal).

TABLE E-12 NVRAMRC Editor Commands

Command	Description
`nvalias` *alias device-path*	Store the command "`devalias` *alias device-path*" 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.
`nvrun`	Execute the contents of the temporary buffer.
`nvstore`	Copy the contents of the temporary buffer to NVRAMRC; discard the contents of the temporary buffer.
`nvunalias` *alias*	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 `nvstore` 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.
`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.

TABLE E-14 Stack Manipulation Commands *(Continued)*

Command	Stack Diagram	Description
`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.
/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.

TABLE E-16 Arithmetic Functions *(Continued)*

Command	Stack Diagram	Description
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.
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.

TABLE E-17 Conversion Operators *(Continued)*

Command	Stack Diagram	Description
/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.

TABLE E-18 Memory Access Commands *(Continued)*

Command	Stack Diagram	Description
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.
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.

TABLE E-18 Memory Access Commands *(Continued)*

Command	Stack Diagram	Description
lpoke	(long adr32 -- okay?)	Store the 32-bit quantity at adr32. Return true if the access was successful. Return false if a 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.

TABLE E-19 Memory Mapping Commands *(Continued)*

Command	Stack Diagram	Description
`map?`	(virt --)	Display memory map information for the virtual address.
`memmap`	(phys space size -- virt)	Map a region of physical addresses; return the allocated virtual address. Unmap with `free-virtual`.
`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 `phys` and `space` to the virtual address. The stack diagram shown applies to Sun-4m machines.
`iomap-pages`	(phys space virt size --)	Perform consecutive `iomap-page`s to map a region of memory given by `size`. 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 `phys` on to virtual address `virt` in the given address space. All addresses are truncated to lie on a page boundary.
`map-pages`	(phys space virt size --)	Perform consecutive map-pages to map a region of memory to the given size.
`map-region`	(region# virt --)	Map a region.

TABLE E-20 Memory Mapping Primitives *(Continued)*

Command	Stack Diagram	Description
`map-regions`	(region# virt size --)	Map successive regions.
`map-segments`	(smentry virt len --)	Perform consecutive `smap!`s to map a region of memory.
`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

Command	Stack Diagram	Description
SuperSPARC™ Module Register Access		
`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.
MXCC Interrupt Register Access		
`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.
BootBus Register Access		
`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

Command	Stack Diagram	Description
`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

Command	Stack Diagram	Description
`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
`:` *name*	(--) Usage: (??? -- ?)	Start creating a new colon definition.
`;`	(--)	Finish creating a new colon definition.
`alias` *new-name old-name*	(--) Usage: (??? -- ?)	Create *new-name* with the same responses as *old-name*.
`buffer:` *name*	(size --) Usage: (-- adr64)	Create a named array in temporary storage.
constant *name*	(n --) Usage: (-- n)	Define a constant (for example, `3 constant bar`).
`2constant` *name*	(n1 n2 --) Usage: (-- n1 n2)	Define a 2-number constant.
`create` *name*	(--) Usage: (-- adr16)	Generic defining word.
`defer` *name*	(--) 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` *name*	(offset size -- offset+size) Usage: (adr -- adr+offset)	Create a named offset pointer.
`struct`	(-- 0)	Initialize for `field` creation.
`value` *name*	(n --) Usage: (-- n)	Create a changeable, named 32-bit quantity.
`variable` *name*	(--) Usage: (-- adr16)	Define a variable.

TABLE E-27 Dictionary Searching Commands

Command	Stack Diagram	Description
' *name*	(-- acf)	Find the named word in the dictionary. Returns the code field address. Use outside definitions.
['] *name*	(-- 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 acf.
$find	(adr len -- adr len false | acf n)	Find a word. n = 0 if not found, n = 1 if immediate, n = -1 otherwise.
find	(pstr -- pstr false | acf n)	Search for a word in the dictionary. The word to be found is indicated by pstr. n = 0 if not found, n = 1 if immediate, n = -1 otherwise.
see *thisword*	(--)	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 pstr.
sifting *ccc*	(--)	Display names of all dictionary entries containing the sequence of characters. *ccc* 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 n bytes in the dictionary.
>body	(acf -- apf)	Find parameter field address from compilation address.
body>	(apf -- acf)	Find compilation address from parameter field address.
compile	(--)	Compile next word at run time.
[compile] *name*	(--)	Compile the next (immediate) word.
forget *name*	(--)	Remove word from dictionary and all subsequent words.
here	(-- adr)	Address of top of dictionary.
immediate	(--)	Mark the last definition as immediate.
is name	(n --)	Install a new action in a defer word or value.

TABLE E-28 Dictionary Compilation Commands *(Continued)*

Command	Stack Diagram	Description
`literal`	(n --)	Compile a number.
`origin`	(-- adr)	Return the address of the start of the Forth system.
`patch` *new-word old-word word-to-patch*	(--)	Replace *old-word* with *new-word* in *word-to-patch*.
`(patch`	(new-n old-n acf --)	Replace `old-n` with `new-n` in word indicated by `acf`.
`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` *name*	(--) Usage: (??? -- ?)	Begin the creation of an assembly language routine called *name*. Commands that follow are interpreted as assembler mnemonics. Note that if the assembler is not installed, `code` is still present, except that machine code must be entered numerically (for example, in hex) with ",".
`c;`	(--)	End the creation of an assembly language routine. Automatically assemble the Forth interpreter `next` 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 *name*. Words created with `label` leave the address of the code on the stack when executed. The commands that follow are interpreted as assembler mnemonics. As with `code`, `label` is present even if the assembler is not installed.
`end-code`	(--)	End the assembly language patch started with `label`.

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 `.s` automatically before each `ok` 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.

TABLE E-31 Changing the Number Base *(Continued)*

Command	Stack Diagram	Description
`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 n in decimal without changing base.
`.h`	(n --)	Display n 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.
`\` *rest-of-line*	(--)	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`.

TABLE E-33 Controlling Text Input *(Continued)*

Command	Stack Diagram	Description
`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.
`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.
`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.

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

Command	Function
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

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

TABLE E-40 Comparison Commands *(Continued)*

Command	Stack Diagram	Description
u<=	(u1 u2 -- flag)	True if u1 <= u2, unsigned.
u>	(u1 u2 -- flag)	True if u1 > u2, unsigned.
u>=	(u1 u2 -- flag)	True if u1 >= u2, unsigned.
within	(n min max -- flag)	True if min <= n < max.

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.

TABLE E-43 `begin` (Conditional) Loop Commands

Command	Stack Diagram	Description
`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 `abort`, but leave stack intact.

TABLE E-46 File Loading Commands

Command	Stack Diagram	Description
`?go`	(--)	Execute Forth, FCode, or binary programs.
`boot` [*specifiers*] `-h`	(--)	Load file from specified source.
`byte-load`	(adr span --)	Interpret loaded FCode binary file. `span` is usually 1.
`dl`	(--)	Load a Forth file over a serial line with `TIP` and interpret. Type: **`~C cat`** *filename* **`^-D`**
`dlbin`	(--)	Load a binary file over a serial line with `TIP`. Type: **`~C cat`** *filename*
`dload` *filename*	(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` *device-specifier argument*	(--)	Load data from specified device into memory at the address given by `load-base`.
`load-base`	(-- adr)	Address at which `load` 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 regname`	(value --)	Change the value stored in any of the above registers. Use in the form: *value* `to` *regname*.
`w`	(window# --)	Set the current window for displaying `%i`*x*, `%L`*x*, or `%o`*x*.

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.
.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).
bpoff	(--)	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.
`debug` *name*	Mark the named Forth word for debugging. Enter the Forth Source-level Debugger with any subsequent attempts to execute that word. After executing `debug`, the execution speed of the system may decrease until debugging is turned off with `debug-off`. (Do not debug basic Forth words such as ".".)
`debug-off`	Turn off the Forth Source-level Debugger so that no word is being debugged.
`resume`	Exit from a subordinate interpreter, and go back to the stepper (see the `F` command in this table).
`stepping`	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.
`tracing`	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.
`Space`	Execute the word just displayed and proceed to the next word.

TABLE E-51 Time Utilities

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

TABLE E-52 Miscellaneous Operations

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

NUMERICS

A

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X