OpenBoot 3.x Quick Reference

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

Part No. 806-2908-10
February 2000, Revision A

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

Please
Recycle

Contents

1

OpenBoot 3.x Quick Reference

Syntax

Enter commands at the `ok` prompt. They are executed left-to-right after a carriage-return. Separate all commands by one or more spaces.

Numeric Usage and Stack Comments

- Numeric I/O defaults to hexadecimal.
- Switch to decimal with `decimal`, switch to hexadecimal with `hex`.
- Use `10 .d` to see which base is currently active.

A numeric stack is used for all numeric parameters. Typing any integer puts that value on top of the stack. (Previous values are pushed down.) The right-hand item in a set always indicates the topmost stack item.

- The command "." removes and displays the top stack value.
- The command `.s` non-destructively shows the entire stack contents.

A stack comment such as (n1 n2 -- n3) or (adr len --) or (--) listed after each command name shows the effect on the stack of executing that command. Items *before* the -- are used by the command and removed from the stack. These items *must*

be present on the stack `before` the command can properly execute. Items after the -- are left on the stack after the command completes execution, and are available for use by subsequent commands.

TABLE 1-1 Numeric Usage and Stack Comments

`|`	Alternate stack results. Example: (input -- adr len false | result true).
`?`	Unknown stack items (changed from ???).
`???`	Unknown stack items.
`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 b`*xxx*	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 l`*xxx*	32-bit value.
`n n1 n2 n3`	Normal signed values.
`+n u`	Unsigned, positive values.
`phys`	Physical address (actual hardware address).
`pstr`	Packed string (`adr len` means unpacked string).
`virt`	Virtual address (address used by software).
`word w`*xxx*	16-bit value.
`xt`	Execution token.

Help Commands

TABLE 1-2 Help Commands

`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).

Device Tree Browsing Commands

TABLE 1-3 Device Tree Browsing Commands

`.properties`	Display the names and values of the current node's properties.
`dev` *node-name*	Search for a node with the given name in the subtree below the current node, and choose the first such node found.
`dev ..`	Choose the device node that is the parent of the current node.
`dev /`	Choose the root machine node.
`device-end`	Leave the device tree.
`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 directly under the specified device in the device tree; without *device-path* it shows the entire device tree.
`words`	Display the names of the current node's methods.

Common Options for the boot Command

TABLE 1-4 Common Options for the boot Command

`boot` [*device-specifier*] [*filename*] [*options*]	
[*device-specifier*]	The name (full path name or alias) of a device. Examples: `cdrom` (CD-ROM drive) `disk` (hard disk) `net` (Ethernet) `tape` (SCSI tape)
[*filename*]	The name of the program to be booted (for example, `stand/diag`). *If specified, filename* is relative to the root of the selected device and partition. If not, the boot program uses the value of the `boot-file` or `diag-file` based on `diag-switch?` parameter.
[*options*]	`-a` - Prompt interactively for the device and name of the boot file. `-h` - Halt after loading the program. (*OS-specific options may differ from system to system.*)

Emergency Keyboard Commands

TABLE 1-5 Emergency Keyboard Commands

Hold down keys during power-on sequence.	
`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.

Diagnostic Test Commands

TABLE 1-6 Diagnostic Test Commands

`probe-scsi`	Identify devices attached to the built-in SCSI bus.
`test` *device-specifier*	Execute the specified device's self-test method. For example: `test floppy` - test the floppy drive, if installed `test net` - test the network connection
`test-all` [*device-specifier*]	Test all devices (that have a built-in self-test method) below the specified node. (If *device-specifier* is absent, the root node is used.)
`watch-clock`	Test the clock function.
`watch-net`	Monitor the network connection.

Examining and Creating Device Aliases

TABLE 1-7 Examining and Creating Device Aliases

`devalias`	Display all current device aliases.
`devalias` *alias*	Display the device path name corresponding to alias.
`devalias` *alias device-path*	Define an alias representing the device path. If an alias with the same name already exists, the new value supersedes the old.

System Information Display Commands

TABLE 1-8 System Information Display Commands

`banner`	Display the power-on banner.
`.version`	Display the version and date of the boot PROM.
`.speed`	Display CPU and bus speeds.

File Load & Run Commands

TABLE 1-9 File Load & Run Commands

`boot` [*specifiers*] `-h`	**(--)**	Load file from specified source.
`byte-load`	**(adr xt--)**	Interpret a loaded FCode binary file. xt 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 specified file over Ethernet to given address.
`go`	**(--)**	Begin executing a previously-loaded binary program, or resume executing an interrupted program.
`init-program`	**(--)**	Initialize to execute a binary file.
`load` [*specifiers*]	**(--)**	Load data from specified device into memory at the address given by `load-base`. (See `boot` format.)
`load-base`	**(-- adr)**	Address at which `load` places the data it reads from a device.

SPARC™ Register Commands

TABLE 1-10 SPARC Register Commands

`%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`	**(-- value)**	Return the value in the given register.

TABLE 1-10 SPARC Register Commands *(Continued)*

`.fregisters`	**`( -- )`**	Display values in %f0 through %f31.
`.locals`	**`( -- )`**	Display the values in the i, l and o registers.
`.registers`	**`( -- )`**	Display values in %g0 through %g7, plus some processor registers.
`.window`	**`( window# -- )`**	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 registers.

SPARC V9 Register Commands

TABLE 1-11 SPARC V9 Register Commands

%fprs %asi %pstate %tl-c %pil %tstate %tt %tba %cwp %cansave %canrestore %otherwin %wstate %cleanwin	**(-- value)**	Return the value in the specified register.
.pstate	**(--)**	Formatted display of the processor state register.
.ver	**(--)**	Formatted display of the version register.
.ccr	**(--)**	Formatted display of the ccr register.
.trap-registers	**(--)**	Display trap-related registers.

Breakpoint Commands

TABLE 1-12 Breakpoint Commands

+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 (Example,). `['] .registers to .breakpoint`

TABLE 1-12 Breakpoint Commands *(Continued)*

`.instruction`	**(--)**	Display the address, opcode for the last-encountered breakpoint.
`.step`	**(--)**	Perform a specified action when a single step occurs.
`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.) Treats 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`.

Miscellaneous Operations

TABLE 1-13 Miscellaneous Operations

`eject-floppy`	**`( -- )`**	Eject the diskette from the drive.
`firmware-version`	**`( -- n )`**	Return major/minor CPU firmware version (that is, 0x00030009 = firmware version 3.9).
`ftrace`	**`( -- )`**	Show calling sequence when exception occurred.
`get-msecs`	**`( -- ms )`**	Return the approximate current time in milliseconds.
`ms`	**`( n -- )`**	Delay for n milliseconds. Resolution is 1 millisecond.
`reset-all`	**`( -- )`**	Reset the entire system (similar to a power cycle).
`sync`	**`( -- )`**	Call the operating system to write any pending information to the hard disk.

NVRAM Configuration Parameters

TABLE 1-14 NVRAM Configuration Parameters

Parameter Name	**`Default`**	Description
`auto-boot?`	**`true`**	If true, boot automatically after power-on or reset.
`boot-command`	**`boot`**	Executed when `auto-boot?` is true.
`boot-device`	**`disk net`**	Device from which to boot.
`boot-file`	**`empty string`**	File to boot (an empty string lets secondary booter choose default).
`diag-device`	**`net`**	Diagnostic boot source device.
`diag-file`	**`empty string`**	File from which to boot in diagnostic mode.
`diag-level`	**`min`**	Level of diagnostics to run (`min` or `max`).
`diag-switch?`	**`false`**	If true, run in diagnostic mode.

TABLE 1-14 NVRAM Configuration Parameters *(Continued)*

fcode-debug?	**false**	If true, include name fields for plug-in device FCodes.
input-device	**keyboard**	Power-on input device (usually keyboard, ttya, or ttyb).
keymap	**no default**	Keymap for custom keyboard.
nvramrc	**empty string**	NVRAM Startup script.
oem-banner	**empty string**	Custom OEM banner (enabled by oem-banner? true).
oem-banner?	**false**	If true, use custom OEM banner.
output-device	**screen**	Power-on output device (usually screen, ttya, or ttyb).
sbus-probe-list	**01**	Which SBus slots are probed and in what order.
scsi-initiator-id	**7**	SCSI bus address of host adapter, range 0-f.
security-mode	**none**	Firmware security level (none, command, or full).
security-password	**no default**	Firmware security password (never displayed).
ttya-mode	**9600,8,n,1,-**	TTYA (baud, #bits, parity, #stop, handshake).
ttyb-mode	**9600,8,n,1,-**	TTYB (baud, #bits, parity, #stop, handshake).
ttya-ignore-cd	**true**	If true, OS ignores TTYA carrier-detect.
ttyb-ignore-cd	**true**	If true, OS ignores TTYB carrier-detect.
ttya-rts-dtr-off	**false**	If true, OS does not assert DTR and RTS on TTYA.
ttyb-rts-dtr-off	**false**	If true, OS does not assert DTR and RTS on TTYB.
use-nvramrc?	**false**	If true, execute commands in NVRAMRC during system start-up.
watchdog-reboot?	**false**	If true, reboot after watchdog reset.

Viewing and Changing Configuration Parameters

TABLE 1-15 Viewing and Changing Configuration Parameters

`password`	Set `security-password`.
`printenv` *[parameter]*	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 the 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.

Commands Affecting NVRAMRC

TABLE 1-16 Commands Affecting NVRAMRC

`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.) Turns on `use-nvramrc?`
`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.

TABLE 1-16 Commands Affecting NVRAMRC *(Continued)*

`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.
`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.

Editor Commands (for Command Lines and NVRAMRC)

TABLE 1-17 Editor Commands (for Command Lines and NVRAMRC)

	Previous Line	Begin Line	Previous Word	Prev. Char	Next Character	Next Word	End Line	Next Line
Move	**^P**	**^A**	**escB**	**^B**	**^F**	**escF**	**^E**	**^N**
Delete		**^U**	**^ W**	**Del**	**^D**	**escD**	**^K**	

Re-type line: **^R**

Show all lines: **^L**

Paste after: **^K ^Y**

Complete command: **^ space**

Show all matches: **^/** or **^?}**

esc = Press and release Escape key first;
^ = Press and hold Control key

Using the NVRAMRC Editor

TABLE 1-18 Using the NVRAMRC Editor

ok **nvedit**	
:	
(use editor commands)	
:	
^c	get back to ok prompt
ok **nvstore**	save changes
ok **setenv use-nvramrc? true**	enable NVRAMRC

Stack Manipulation Commands

TABLE 1-19 Stack Manipulation Commands

-rot	**(n1 n2 n3 -- n3 n1 n2)**	Inversely rotate three stack items.
>r	**(n --)**	Move a stack item to the return stack.
?dup	**(n -- n n | 0)**	Duplicate the top stack item if non-zero.
2drop	**(n1 n2 --)**	Remove top two items from the stack.
2dup	**(n1 n2 -- n1 n2 n1 n2)**	Duplicate top two stack items.
2over	**(n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)**	Copy second two stack items.
2swap	**(n1 n2 n3 n4 -- n3 n4 n1 n2)**	Exchange top two pairs of stack items.
clear	**(??? --)**	Empty the stack.
depth	**(??? -- ??? +n)**	Return the number of items on the stack.

TABLE 1-19 Stack Manipulation Commands *(Continued)*

drop	**(n --)**	Remove the top item from the stack.
dup	**(n -- n n)**	Duplicate the top stack item.
over	**(n1 n2 -- n1 n2 n1)**	Copy the second stack item to the top of the stack.
pick	**(nu ... n1 n0 u -- nu ... n1 n0 nu)**	Copy u-th stack item (1 pick = over).
r>	**(-- n)**	Move a return stack item to the stack.
r@	**(-- n)**	Copy the top of the return stack to the stack.
roll	**(nu ... n1 n0 u -- nu-1 ... n1 n0 nu)**	Rotate u stack items (2 roll = rot).
rot	**(n1 n2 n3 -- n2 n3 n1)**	Rotate three stack items.
swap	**(n1 n2 -- n2 n1)**	Exchange the top two stack items.
tuck	**(n1 n2 -- n2 n1 n2)**	Copy the top stack item below the second item.

Changing the Number Base

TABLE 1-20 Changing the Number Base

`decimal`	**(--)**	Set the number base to 10.
`d#` *number*	**(-- n)**	Interpret the next number in decimal; base is unchanged.
`hex`	**(--)**	Set the number base to 16.
`h#` *number*	**(-- n)**	Interpret the next number in hex; base is unchanged.
`.d`	**(n --)**	Display n in decimal without changing base.
`.h`	**(n --)**	Display n in hex without changing base.

Basic Number Display

TABLE 1-21 Basic Number Display

`.`	**(n --)**	Display a number in the current base.
`.s`	**(--)**	Display contents of data stack.
`showstack`	**(--)**	Execute `.s` automatically before each `ok` prompt.

Arithmetic Functions

TABLE 1-22 Arithmetic Functions

*	**(n1 n2 -- n3)**	Multiply n1 * n2.
+	**(n1 n2 -- n3)**	Add n1 + n2.
-	**(n1 n2 -- n3)**	Subtract n1 - n2
/	**(n1 n2 -- quot)**	Divide n1 / n2; remainder is discarded.

TABLE 1-22 Arithmetic Functions *(Continued)*

lshift	**(n1 +n -- n2)**	Left-shift n1 by +n bits.
rshift	**(n1 +n -- n2)**	Right-shift n1 by +n bits.
>>a	**(n1 +n -- n2)**	Arithmetic right-shift n1 by +n bits.
abs	**(n -- u)**	Absolute value.
and	**(n1 n2 -- n3)**	Bitwise logical AND.
bounds	**(n cnt -- n+cnt n)**	Prepare arguments for do or ?do loop.
bljoin	**(b.low b2 b3 b.hi -- long)**	Join four bytes to form a 32-bit value.
bwjoin	**(b.low b.hi -- word)**	Join two bytes to form a 16-bit value.
lbsplit	**(long -- b.low b2 b3 b.hi)**	Split a 32-bit value into four bytes.
lwsplit	**(long -- w.low w.hi)**	Split a 32-bit value into two 16-bit words.
max	**(n1 n2 -- n3)**	n3 is maximum of n1 and n2.
min	**(n1 n2 -- n3)**	n3 is minimum of n1 and n2.
mod	**(n1 n2 -- rem)**	Remainder of n1 / n2.
negate	**(n1 -- n2)**	Change the sign of n1.
invert	**(n1 -- n2)**	Bitwise ones complement.
or	**(n1 n2 -- n3)**	Bitwise logical OR.
wbsplit	**(word -- b.low b.hi)**	Split 16-bit value into two bytes.
wljoin	**(w.low w.hi -- long)**	Join two 16-bit values to form a 32-bit value.
xor	**(n1 n2 -- n3)**	Bitwise exclusive OR.

Disassembler Commands

TABLE 1-23 Disassembler Commands

`+dis`	**`( -- )`**	Continue disassembling where the last disassembly left off.
`dis`	**`( adr -- )`**	Begin disassembling at the given address.

Memory Access Commands

TABLE 1-24 Memory Access Commands

`!`	**`( n adr -- )`**	Store a number at `adr`.
`+!`	**`( n adr -- )`**	Add `n` to the number stored at `ad`.
`@`	**`( adr -- n )`**	Fetch a number from `adr`.
`c!`	**`( n adr -- )`**	Store low byte of `n` at `adr`.
`c@`	**`( adr -- byte )`**	Fetch a byte from `adr`.
`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. (Also `lpeek`, `wpeek`.)
`cpoke`	**`( byte adr -- okay? )`**	Store the byte to adr. Return true if the access was successful. Return false if a write access error occurred. (Also `lpoke`, `wpoke`.)
`comp`	**`( adr1 adr2 len -- n )`**	Compare two byte arrays, n = 0 if arrays are identical, n = 1 if first byte that is different is greater in array#1, n = -1 otherwise.
`dump`	**`( adr len -- )`**	Display len bytes of memory starting at adr.
`fill`	**`( adr size byte -- )`**	Set size bytes of memory to byte.
`l!`	**`( n adr32 -- )`**	Store a 32-bit number at adr32.

TABLE 1-24 Memory Access Commands *(Continued)*

l@	**(adr32 -- long)**	Fetch a 32-bit number from adr32.
move	**(src dst u --)**	Copy u bytes from src to dst, handle overlap properly.
w!	**(n adr16 --)**	Store a 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.
x!	**(o oaddr --)**	Store a 64-bit number at oaddr, must be 64-bit aligned.
x@	**(oaddr -- o)**	Fetch a 64-bit number from oaddr, must be 64-bit aligned.

Memory Mapping Commands

TABLE 1-25 Memory Mapping Commands

`alloc-mem`	**`( size -- virt )`**	Allocate and map size bytes of available memory; return the virtual address. Unmap with `free-mem`.
`free-mem`	**`( virt size -- )`**	Free memory allocated by `alloc-mem`.
`free-virtual`	**`( virt size -- )`**	Undo mappings created with `memmap`.
`map?`	**`( virt -- )`**	Display memory map information for the virtual address.
`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.
`pgmap!`	**`( pmentry virt -- )`**	Store a new page map entry for the virtual address.
`pgmap?`	**`( virt -- )`**	Display the decoded page map entry 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 8K).
`sbus`	**`( -- space )`**	Specify the SBus address space for mapping.

Defining Words

TABLE 1-26 Defining Words

: *name*	**(--)** **Usage: (??? -- ?)**	Start creating a new colon definition.
;	**(--)**	Finish creating a new colon definition.
buffer: *name*	**(size --)** **Usage: (-- adr)**	Create a named array in temporary storage.
constant *name*	**(n --)** **Usage: (-- n)**	Define a constant (for example, `3 constant bar`).
create *name*	**(--)** **Usage: (-- adr)**	Generic defining word.
defer *name*	**(--)** **Usage: (??? -- ?)**	Define forward reference or execution vector.
value *name*	**(n --)** **Usage: (-- n)**	Create a changeable, named quantity.
variable *name*	**(--)** **Usage: (-- adr)**	Define a variable.

Dictionary Searching Commands

TABLE 1-27 Dictionary Searching Commands

' name	**(-- xt)**	Find the named word in the dictionary. (Returns the execution token. Use outside definitions.)
['] name	**(-- xt)**	Similar to ' but is used inside definitions.
.calls	**(xt --)**	Display a list of all words that call the word whose execution token is xt.
$find	**(adr len -- adr len false | xt n)**	Find a word. n = 0 if not found, n = 1 if immediate, n = -1 otherwise.
see thisword	**(--)**	Decompile the named command.

TABLE 1-27 Dictionary Searching Commands *(Continued)*

(see)	**(xt --)**	Decompile the word indicated by the execution token.
sifting ccc	**(--)**	Display names of all dictionary entries containing the sequence of characters. ccc contains no spaces.
words	**(--)**	Display visible words in the dictionary.

Manipulating Text Strings

TABLE 1-28 Manipulating Text Strings

" C*ccc*"	**(-- adr len)**	Collect an input stream string.
." *ccc*"	**(--)**	Compile a string for later display.
bl	**(-- char)**	ASCII code for the space character; decimal 32.
count	**(pstr -- adr +n)**	Unpack a packed string.
p" *ccc*"	**(-- pstr)**	Collect a string from the input stream; store as a packed string.

Dictionary Compilation Commands

TABLE 1-29 Dictionary Compilation Commands

`,`	**`( 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.
`allot`	**`( n -- )`**	Allocate n bytes in the dictionary.
`forget name`	**`( -- )`**	Remove word from dictionary and all subsequent words.
`here`	**`( -- adr )`**	Address of top of dictionary.
`to` *name*	**`( n -- )`**	Install a new action in a `defer` word or `value`.
`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.

Controlling Text Input

TABLE 1-30 Controlling Text Input

`(` *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.
`key`	**`( -- char )`**	Read a character from the assigned input device.
`key?`	**`( -- flag )`**	True if a character has been entered from the input device..

Displaying Text Output

TABLE 1-31 Displaying Text Output

`cr`	**`( -- )`**	Terminate a line on the display and go to the next line.
`emit`	**`( char -- )`**	Display the character.
`type`	**`( adr +n -- )`**	Display n characters.

Redirecting I/O

TABLE 1-32 Redirecting I/O

`input`	**`( dev-spec -- )`**	Select device (`ttya`, `ttyb`, `keyboard`, or *" dev-speci*") for subsequent input.
`io`	**`( dev-spec -- )`**	Select device for subsequent input and output.
`output`	**`( dev-spec -- )`**	Select device (`ttya`, `ttyb`, `screen`, or " *dev-spec*") for subsequent output.

Comparison Commands

TABLE 1-33 Comparison Commands

`<`	**`( n1 n2 -- flag )`**	True if n1 < n2.
`<=`	**`( n1 n2 -- flag )`**	True if n1 <= n2.
`<>`	**`( n1 n2 -- flag )`**	True if n1 <> n2.
`=`	**`( n1 n2 -- flag )`**	True if n1 = n2.
`>`	**`( n1 n2 -- flag )`**	True if n1 > n2.
`>=`	**`( n1 n2 -- flag )`**	True if n1 >= n2.
`between`	**`( n min max -- flag )`**	True if min <= n <= max.

TABLE 1-33 Comparison Commands *(Continued)*

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

if-else-then Commands

TABLE 1-34 if-else-then Commands

else	**(--)**	Execute the following code if `if` failed.
if	**(flag --)**	Execute the following code if flag is true.
then	**(--)**	Terminate `if`...`else`...`then`.

`begin` (Conditional) Loop Commands

TABLE 1-35 begin (Conditional) Loop Commands

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.

do (Counted) Loop Commands

TABLE 1-36 do (Counted) Loop Commands

+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.
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.

case Statement

(value)

case

2 of **." it was two"** endof

0 of **." it was zero"** endof

." it was " dup . *(optional default clause)*

endcase

Program Execution Control Commands

TABLE 1-37 Program Execution Control Commands

`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.
`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.

Alternate Address Space Access Commands

TABLE 1-38 Alternate Address Space Access Commands

`spacec!`	**`( byte adr asi -- )`**	Store 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 values at asi and address. Order is implementation-dependent.
`spaced@`	**`( adr asi -- n1 n2 )`**	Fetch the two values from asi and address. Order is implementation-dependent.
`spacel!`	**`( long adr asi -- )`**	Store 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 -- word )`**	Fetch the 16-bit word from asi and address.
`spacex!`	**`( x adr asi -- )`**	Store the 64-bit word at asi and address.
`spacex@`	**`( adr asi -- x )`**	Fetch the 64-bit word from asi and address.

Cache Manipulation Commands

TABLE 1-39 Cache Manipulation Commands

`clear-cache`	**`( -- )`**	Invalidate all cache entries.
`cache-off`	**`( -- )`**	Disable the cache.
`cache-on`	**`( -- )`**	Enable the cache.
`flush-cache`	**`( -- )`**	Write back any pending data from the cache.

Multiprocessor Command

TABLE 1-40 Multiprocessor Command

`switch-cpu`	**`( cpu# -- )`**	Switch to indicated CPU.

Program Execution Control Commands

TABLE 1-41 Program Execution Control Commands

`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.
`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.