OpenBoot 3.x の手引き

A Sun Microsystems, Inc. Business
901 San Antonio Road
Palo Alto, , CA 94303-4900
USA 650 960-1300fax 650 969-9131

Part Number 805-5645
1998年11月(Revision A)

原典 : OpenBoot 3.x Quick Reference

Part No: 805–4437

Revision A

目次

OpenBoot 3.x の手引き

構文

ok プロンプトでコマンドを入力し Return キーを押すと、左のコマンドから順番に実行されます。コマンドとコマンドの間は 1 つ以上の空白文字で区切ってください。

数値の使用方法とスタックのコメント

- 数値の入出力のデフォルトは 16 進です。
- decimalで 10 進に切り替わり、hexで 16 進に切り替わります。
- 現在どちらの進方が有効なのかを調べるには 10 .d を使用してください。

すべての数値パラメタは数値スタックを使用します。整数を入力すると、その値はスタックの一番上に置かれます。(以前の値はプッシュされます) 一連の入力で右側の項目が常にスタックの一番上の項目になります。

- "." コマンドはスタックの一番上の値を削除して表示します。
- .s コマンドはスタックの内容を壊さずに、すべて表示します。

各コマンドの後にある (n1 n2 – n3)、(adr len –)、または (–) のようなスタックコメントは、そのコマンドを実行したときのスタックの結果を示しています。– の前にある項目はそのコマンドで使用され、スタックから削除されます。これらの項目はコマンドが実行される前にスタックに存在していなければなりません。– の後

にある項目はコマンドの実行を終了した後にスタックに残り、続きのコマンドで使用することができます。

表 1–1 数値の使用方法とスタックのコメント

|	代替スタック結果。 例: (input – adr len false | result true)
?	未知のスタック項目 (???から変更)。
???	未知のスタック項目。
adr	メモリーアドレス (一般的に仮想アドレス)。
adr16	メモリーアドレス。16 ビット境界でなければなりません。
adr32	メモリーアドレス。32 ビット境界でなければなりません。
adr64	メモリーアドレス。64 ビット境界でなければなりません。
byte b*xxx*	8 ビット値 (32 ビットワードの下位バイト)。
char	7 ビット値 (下位バイト)。最上位ビットは不定。
cnt/len/size	カウント値または長さ。
flag xxx?	0 の場合 false。それ以外の場合、true (通常は -1)。
long l*xxx*	32 ビット値。
n n1 n2 n3	通常の符号付きの値。
+n u	符号なし、正の値 。
phys	物理アドレス (実際のハードウェアアドレス)。
pstr	パックされた文字列。(adr len はアンパックされた文字列)
virt	仮想アドレス (ソフトウェアが使用するアドレス)。

表 1–1　数値の使用方法とスタックのコメント　続く

word w*xxx*	16 ビット値。
xt	実行トークン。

ヘルプコマンド

表 1–2　ヘルプコマンド

help	ヘルプの主なカテゴリを表示します。
help *category*	カテゴリ内のコマンドのヘルプをすべて表示します。 カテゴリ記述の最初の単語だけを使用します。
help *command*	各コマンドのヘルプを表示します (ただし、ヘルプが提供されている場合)。

デバイスツリー表示コマンド

表 1–3　デバイスツリー表示コマンド

.properties	現在のノードの特性の名前と値を表示します。
dev *node-name*	指定されたノード名を現在のノードの下のサブツリーで捜し、最初に見つかったノードを選択します。
dev ..	現在のノードの親にあたるデバイスノードを選択します。
dev /	ルートマシンノードを選択します。
device-end	デバイスツリーを解除します。

表 1–3　デバイスツリー表示コマンド　続く

`ls`	現在のノードの子の名前を表示します。
`pwd`	現在のノードを示すデバイスパス名を表示します。
`show-devs` [*device-path*]	デバイスツリー内の指定されたデバイスのすぐ下のすべてのデバイスを表示します。*device-path* を指定しないと、デバイスツリー全体を表示します。
`words`	現在のノードの方式名を表示します。

boot コマンドの一般的オプション

表 1–4　boot コマンドの一般的オプション

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

[*device-specifier*]	起動デバイス名 (フルパス名または別名)。 例: `cdrom` (CD-ROM ドライブ) `disk` (ハードディスク) `net` (Ethernet) `tape` (SCSI テープ)
[*filename*]	起動するプログラムの名前 (たとえば `stand/diag`)。 filename は (指定している場合) 選択するデバイスとパーティションのルートからのパス名とします。filename を指定しないと、起動プログラムは `boot-file` の値や `diag-switch?` 変数に基づいた `diag-file` の値を使用します。
[*options*]	`-a` - デバイスと起動ファイルの名前を対話式に入力します。 `-h` - プログラムを読み込んでから停止します。 (これらは OS に固有のオプションで、システムによって異なります。)

緊急キーボードコマンド

表 1–5 緊急キーボードコマンド

電源投入処理時に次のキーを押したままにしてください。	
`Stop`	POST を省略します。このコマンドはセキュリティーモードには依存しません。(注 : 一部のシステムはデフォルトで POST を省略します。そのような場合は、 `Stop-D` を使用して POST を起動してください。)
`Stop-A`	強制終了させます。
`Stop-D`	診断モードに入ります (`diag-switch?` を `true` に設定します)。
`Stop-F`	プローブを行わず、ttya で FORTH に入ります。`fexit` を使用して初期設定省略を続けます。ハードウェアが壊れている場合に効果があります。
`Stop-N`	NVRAM の内容をデフォルトに設定します。

診断テストコマンド

表 1–6 診断テストコマンド

`probe-scsi`	組み込み SCSI バスに接続されているデバイスを確認します。
`test` *device-specifier*	指定したデバイスの自己診断方法を実行します。例を示します。 `test floppy` : フロッピードライブが接続されている場合、テストします。 `test net` : ネットワーク接続をテストします。
`test-all` [*device-specifier*]	指定したデバイスツリーノードの下の (組み込み自己診断方法を備える) すべてのデバイスをテストします (*device-specifier* を指定しないと、ルートノードが使用されます。)

表 1–6　診断テストコマンド　続く

`watch-clock`	時計機能をテストします。
`watch-net`	ネットワークの接続を監視します。

デバイス別名の確認と作成

表 1–7　デバイス別名の確認と作成

`devalias`	デバイス別名の確認と作成
`devalias` *alias*	*alias* に対応するデバイスパス名を表示します。
`devalias` *alias device-path*	*device-path* を表す別名を定義します。同じ名前の別名がすでに存在すると、新しい名前に更新します。

システム情報表示コマンド

表 1–8　システム情報表示コマンド

`banner`	電源投入時のバナーを表示します。
`.version`	起動 PROM のバージョンと日付を表示します。
`.speed`	CPU とバスの速度を表示します。

ファイル読み込みおよび実行コマンド

表 1–9 ファイル読み込みおよび実行コマンド

boot [*specifiers*] -h	**(--)**	指定されたソースからファイルを読み込みます。
byte-load	**(adr xt--)**	読み込まれた FCode バイナリファイルを解釈します。xt は通常 1 です。
dl	**(--)**	tip を使用してシリアルライン経由で Forth ファイルを読み込み、解釈します。次のように入力します。 ~C cat *filename* ^-D
dlbin	**(--)**	tip を使用してシリアルライン経由でバイナリファイルを読み込みます。次のように入力します。 ~C cat *filename*
dload *filename*	**(adr --)**	Ethernet 経由で指定されたファイルを指定されたアドレスに読み込みます。
go	**(--)**	あらかじめ読み込まれていたバイナリプログラムの実行を開始します。または、中断されたプログラムの実行を再開します。
init-program	**(--)**	バイナリファイルの実行に備えて初期化します。
load [*specifiers*]	**(--)**	指定されたデバイスから load-baseによって指定されるアドレスにデータを読み込みます。(boot の形式を参照)
load-base	**(-- adr)**	load がデバイスから読んだデータを読み込むアドレス。

SPARC™ レジスタコマンド

表 1–10 SPARC レジスタコマンド

%g0 through %g7	**(-- value)**	指定されたレジスタの値を返します。
%i0 through %i7	**(-- value)**	指定されたレジスタの値を返します。
%l0 through %l7	**(-- value)**	指定されたレジスタの値を返します。
%o0 through %o7	**(-- value)**	指定されたレジスタの値を返します。
%pc %npc	**(-- value)**	指定されたレジスタの値を返します。
.fregisters	**(--)**	%f0 から %f31 までの値を表示します。
.locals	**(--)**	i、l、o レジスタの値を表示します。
.registers	**(--)**	%g0 から %g7 までのほかに、いくつかのプロセッサレジスタの値を表示します。
.window	**(window# --)**	指定されたウィンドウを表示します。
ctrace	**(--)**	C サブルーチンを示す復帰スタックを表示します。
set-pc	**(value --)**	%pc を value に、 %npc を (value+4)にそれぞれ設定します。
to *regname*	**(value --)**	上記のうちの任意のレジスタの格納値を変更します。*value* to *regname* の形式で使用してください。
w	**(window# --)**	レジスタを表示する現在のウィンドウを設定します。

SPARC V9 レジスタコマンド

表 1–11 SPARC V9 レジスタコマンド

%fprs	**(-- value)**	指定されたレジスタの値を返します。
%asi		
%pstate		
%tl-c		
%pil		
%tstate		
%tt		
%tba		
%cwp		
%cansave		
%canrestore		
%otherwin		
%wstate		
%cleanwin		
.pstate	**(--)**	プロセッサ状態レジスタの書式付き表示。
.ver	**(--)**	バージョンレジスタの書式付き表示。
.ccr	**(--)**	ccr レジスタの書式付き表示。
.trap-registers	**(--)**	トラップレジスタを表示します。

ブレークポイントコマンド

表 1–12 ブレークポイントコマンド

+bp	**(adr --)**	指定されたアドレスにブレークポイントを追加します。
-bp	**(adr --)**	指定されたアドレスからブレークポイントを削除します。
--bp	**(--)**	最新に設定されたブレークポイントを削除します。
.bp	**(--)**	現在設定されているすべてのブレークポイントを表示します。
.breakpoint	**(--)**	ブレークポイントが発生したときに指定された処理を実行します。例: ['] .registers to .breakpoint と入力。
.instruction	**(--)**	最後に現れたブレークポイントのアドレスとオペコードを表示します。
.step	**(--)**	シングルステップで実行になったときに指定された処理を実行します
bpoff	**(--)**	すべてのブレークポイントを削除します。
finish-loop	**(--)**	このループの終わりまで実行します。
go	**(--)**	ブレークポイントから実行を継続します。これを利用して、 go を発行する前にプロセッサのプログラムカウンタをセットアップすることにより、任意のアドレスに移ることができます。
gos	**(n --)**	go を n 回実行します。
hop	**(--)**	(step コマンドと同じです。)サブルーチン呼び出しを 1 つの命令として使用して扱ってください。
hops	**(n --)**	hop を n 回実行します。
return	**(--)**	このサブルーチンの終わりまで実行します。
returnl	**(--)**	このリーフサブルーチンの終わりまで実行します。
skip	**(--)**	現在の命令をスキップします (実行しません)。

表 1–12　ブレークポイントコマンド　続く

step	**(--)**	1 命令を 1 つずつ実行します。
steps	**(n --)**	step を n 回実行します。
till	**(adr --)**	指定されたアドレスに行き当たるまで実行します。+bp goと等価。

その他の処理

表 1–13　その他の処理

eject-floppy	**(--)**	フロッピードライブからフロッピーディスクをイジェクトします。
firmware-version	**(-- n)**	メジャー / マイナー CPU ファームウェアバージョン (つまり、0x00030009 = ファームウェアバージョン 3.9) を返します。
ftrace	**(--)**	例外発生時の呼び出し順序を表示します。
get-msecs	**(-- ms)**	ミリ秒単位で、現在のおおよその時間を返します。
ms	**(n --)**	n ミリ秒の遅延。1 ミリ秒単位。
reset-all	**(--)**	システム全体をリセットします (電源再投入と同じ)。
sync	**(--)**	オペレーティングシステムを呼び出してすべての保留情報をハードディスクに書き出します。

重要な NVRAM システム変数

表 1–14　重要な NVRAM システム変数

auto-boot?	**true**	true の場合、電源投入またはリセット後に自動的に起動します。
boot-command	**boot**	auto-boot が true の場合に実行されるコマンド。
boot-device	**disk net**	起動するデバイ
boot-file	**empty string**	起動するファイル (空白の場合、第 2 起動プログラムがデフォルトを選択します)。
diag-device	**net**	診断起動ソースデバイス。
diag-file	**empty string**	診断モードで起動するファイ。
diag-level	**min**	実行される診断レベル (min または max)。
diag-switch?	**false**	true の場合、追加デバイス FCode の名前フィールドを取り入れます。
fcode-debug?	**false**	true の場合、診断プログラムを実行します。
input-device	**keyboard**	電源投入時の入力デバイス (通常 keyboard、ttya、またはttyb)。
keymap	**no default**	キーボードカスタマイズ用キーマップ。
nvramrc	**empty string**	NVRAM 起動スクリプト。
oem-banner	**empty string**	カスタム OEM バナー (oem-banner が true で使用可能になります)。
oem-banner?	**false**	true の場、カスタム OEM バナーを使用します。
output-device	**screen**	電源投入時の出力デバイス (通常 screen、ttya、またはttyb)。
sbus-probe-list	**01**	プローブする SBus スロットと、順序を指定します。

表 1–14 重要な NVRAM システム変数 続く

scsi-initiator-id	**7**	ホストアダプタの SCSI バスアドレス。範囲は 0 から f 。
security-mode	**none**	ファームウェアセキュリティーレベル。(none、commandまたはfull)。
security-password	**no default**	ファームウェアセキュリティーパスワード (表示されません)。
ttya-mode	**9600,8,n,1,-**	ttya (ボーレート、ビット数、パリティー、ストップビット数、ハンドシェーク)
ttyb-mode	**9600,8,n,1,-**	ttyb (ボーレート、ビット数、パリティー、ストップビット数、ハンドシェーク)
ttya-mode	**9600,8,n,1,-**	ttya (ボーレート、ビット数、パリティー、ストップビット数、ハンドシェーク)
ttya-ignore-cd	**true**	true の場合、OS は ttya キャリア検出を無視します。
ttyb-ignore-cd	**true**	true の場合、OS は ttyb キャリア検出を無視します。
ttya-rts-dtr-off	**false**	true の場合、ttya 上で DTR と RTS の表明を行いません。
ttyb-rts-dtr-off	**false**	true の場合、ttyb 上で DTR と RTS の表明を行いません。
use-nvramrc?	**false**	true の場合、システム起動時に nvramrc のコマンドを実行します。
watchdog-reboot?	**false**	true の場、ウォッチドッグリセットの後再起動します。

システム変数表示/変更用コマンド

表 1–15　システム変数表示/変更用コマンド

password	security-password を設定します。
printenv ＜システム変数＞	すべての現在の変数とデフォルト値を表示します。(数値は通常 10 進値で表示されます。) printenv ＜システム変数＞ は指定された変数の現在値を表示します。
setenv ＜システム変数＞＜値＞	変数を指定された 10 進値またはテキスト値に設定します。 (変更は永久的ですが、通常はリセット後に初めて有効になります。)
set-default ＜システム変数＞	指定された変数の設定値を工場出荷時のデフォルトに戻します。
set-defaults	変数の設定値を工場出荷時のデフォルトに戻します。

NVRAMRC エディタコマンド

表 1–16　NVRAMRC エディタコマンド

nvalias *alias device-path*	NVRAMRC にコマンド "devalias *alias device-path*" を格納します。この別名は、nvunalias または set-defaults コマンドが実行されるまで有効です。 use-nvramrc?は オンになります。
nvedit	NVRAMRC エディタを起動します。前の nvedit セッションからのデータが一時バッファー内に残っている場合は、以前の内容の編集を再開します。残っていない場合は、NVRAMRC の内容を一時バッファーに読み込んで、それらの編集を開始します。
nvquit	一時バッファーの内容を、NVRAMRC に書かないで捨てます。

表 1–16　NVRAMRC エディタコマンド　続く

`nvrecover`	NVRAMRC の内容が `set-defaults` の実行結果として失われている場合、それらの内容を回復し、次に `nvedit` の場合と同様にこのエディタを起動します。NVRAMRC の内容が失われたときから `nvrecover` が実行されるまでの間に `nvedit` を実行した場合は、`nvrecover` は失敗します。
`nvstore`	一時バッファーの内容を NVRAMRC にコピーします。一時バッファーの内容は捨てます。
`nvunalias` *alias*	対応する別名を NVRAMRC から削除します。

エディタコマンド (コマンド行と NVRAMRC用)

表 1–17　エディタコマンド (コマンド行、NVRAMRC 用)

	前の行	行の先頭	前の単語	前の文字	次の文字	次の単語	行の終り	次の行
移動	**^P**	**^A**	**esc B**	**^B**	**^F**	**esc F**	**^E**	**^N**
削除		**^U**	**^W**	**Del**	**^D**	**esc D**	**^K**	
行の上書き				**^R**				
すべての行を表示				**^L**				
^-K の後、ペースト				**^Y**				
コマンド完了				**^-space**				
すべての一致するものを表示				**^/** または **^?}**				

esc = Escape キーを最初に押して離します。 ^ = Control キーを押したままにする。

NVRAMRC エディタの使用方法

ok **nvedit**

:

(エディタコマンドを使用します)

:

^-C　(ok プロンプトに戻ります)

ok **nvstore**　(変更を保存します)

ok **setenv use-nvramrc? true**　(NVRAMRC を使用可能にします)

スタック操作コマンド

表 1–18　スタック操作コマンド

-rot	**(n1 n2 n3 -- n3 n1 n2)**	3 スタック項目を逆方向に回転します。
>r	**(n --)**	スタック項目を復帰スタックに転送します。
?dup	**(n -- n n | 0)**	ゼロ以外の場合、一番上のスタック項目を複製します。
2drop	**(n1 n2 --)**	一番上の 2 スタック項目を削除します。
2dup	**(n1 n2 -- n1 n2 n1 n2)**	一番上の 2 スタック項目を複製します。

表 1–18 スタック操作コマンド　続く

2over	**(n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)**	2 番目以降の 2 スタック項目をコピーします。
2swap	**(n1 n2 n3 n4 -- n3 n4 n1 n2)**	一番上の 2 スタック項目を入れ替えます。
clear	**(??? --)**	スタックを空にします。
depth	**(??? -- ??? +n)**	スタック上の項目数を返します。
drop	**(n --)**	一番上のスタック項目を削除します。
dup	**(n -- n n)**	一番上のスタック項目を複製します。
over	**(n1 n2 -- n1 n2 n1)**	2 番目のスタック項目をスタックの一番上にコピーします。
pick	**(nu ... n1 n0 u -- nu ... n1 n0 nu)**	u 番目のスタック項目をコピーします (1 pick = over)。
r>	**(-- n)**	復帰スタック項目をスタックに転送します。
r@	**(-- n)**	復帰スタックの一番上をスタックにコピーします。
roll	**(nu ... n1 n0 u -- nu-1 ... n1 n0 nu)**	u 箇のスタック項目を回転します。(2 roll = rot)。

表 1–18　スタック操作コマンド　続く

rot	**(n1 n2 n3 -- n2 n3 n1)**	3 スタック項目を回転します。
swap	**(n1 n2 -- n2 n1)**	一番上の 2 スタック項目を入れ替えます。
tuck	**(n1 n2 -- n2 n1 n2)**	一番上のスタック項目を 2 番目の項目の下にコピーします。

基数の変更

表 1–19　基数の変更

decimal	**(--)**	基数を 10 に設定します。
d# *number*	**(-- n)**	次の数値を 10 進で解釈します。基数は変わりません。
hex	**(--)**	基数を 16 に設定します。
h# *number*	**(-- n)**	次の数値を 16 進で解釈します。基数は変わりません。
.d	**(n --)**	基数を変更しないで n を 10 進で表示します。
.h	**(n --)**	基数を変更しないで n を 16 進で表示します。

基数値表示

表 1–20 基数値表示

.	**(n --)**	数値を現在の基数で表示します。
.s	**(--)**	データスタックの内容を表示します。
showstack	**(--)**	各 ok プロンプトの前で.s を自動的に実行します。

単精度演算機能

表 1–21 単精度演算機能

*	**(n1 n2 -- n3)**	n1 * n2 の乗算を行います。
+	**(n1 n2 -- n3)**	n1 + n2 の加算を行います。
-	**(n1 n2 -- n3)**	n1 - n2 の減算を行います。
/	**(n1 n2 -- quot)**	n1 / n2 の除算を行います。剰余は捨てられます。
lshift	**(n1 +n -- n2)**	n1 を+n ビット左へシフトします。
rshift	**(n1 +n -- n2)**	n1 を +n ビット右へシフトします。
>>a	**(n1 +n -- n2)**	n1 を +n ビット算術右シフトします。
abs	**(n -- u)**	絶対値。
and	**(n1 n2 -- n3)**	ビット単位の論理積。AND
bounds	**(n cnt -- n+cnt n)**	do または ?do ループ用に引数を用意します。
bljoin	**(b.low b2 b3 b.hi -- long)**	4 バイトを結合して、32 ビットのロングワードを生じます。

表 1–21　単精度演算機能　続く

bwjoin	**(b.low b.hi -- word)**	2 バイトを結合して、16 ビットのワードを生じます。
lbsplit	**(long -- b.low b2 b3 b.hi)**	32 ビットのロング語を 4 バイトに分割します。
lwsplit	**(long -- w.low w.hi)**	32 ビットのロングワードを 2 つの 16 ビットワードに分割します。
max	**(n1 n2 -- n3)**	n1 とn2 の大きい方の値がn3に入ります。
min	**(n1 n2 -- n3)**	n1 と n2 の小さい方の値が n3に入ります。
mod	**(n1 n2 -- rem)**	n1/n2の剰余。
negate	**(n1 -- n2)**	n1の符号を変更します。
invert	**(n1 -- n2)**	n1 のすべてのビットを反転します。
or	**(n1 n2 -- n3)**	ビット単位の論理和。 OR
wbsplit	**(word -- b.low b.hi)**	16 ビットワードを 2 バイトに分割します。
wljoin	**(w.low w.hi -- long)**	2 つの 16 ビットのロングワードを結合して、1 つの 32 ビットロングワードを生じます。
xor	**(n1 n2 -- n3)**	ビット単位の排他的論理和。 OR

逆コンパイラコマンド

表 1–22 逆コンパイラコマンド

+dis	**(--)**	最後に逆コンパイルを中断したところから逆コンパイルを継続します。
dis	**(adr --)**	指定されたアドレスから逆コンパイルを開始します。

メモリーアクセスコマンド

表 1–23 メモリーアクセスコマンド

!	**(n adr --)**	数値を adr に格納します。
+!	**(n adr --)**	adに格納されている数値に n を加算します。
@	**(adr -- n)**	数値を adr から 取り出します。
c!	**(n adr --)**	n の下位バイトをadr に格納します。
c@	**(adr -- byte)**	1 バイトを adr から取り出します。
cpeek	**(adr -- false | byte true)**	1 バイトを adr から取り出します。アクセスが成功した場合はそのデータと true を返し、読み取りエラーが発生した場合は false を返します。 (lpeek、wpeekも同じ)
cpoke	**(byte adr -- okay?)**	byte を adr に格納します。アクセスが成功した場合は true を返し、書き込みエラーが発生した場合は false を返します。 (lpoke、wpokeも同じ)
comp	**(adr1 adr2 len -- n)**	2 つのバイト配列を比較します。両配列が一致する場合 n = 0、最初の異なるバイトの値が配列 1 側より大きい場合 n = 1、それ以外の場合は n = -1 になります。

表 1–23　メモリーアクセスコマンド　続く

dump	**(adr len --)**	adr から始まる lens メモリーバイトを表示します。
fill	**(adr size byte --)**	メモリーのバイトを byte に設定します。
l!	**(n adr32 --)**	32 ビット数を adr32 に格納します。
l@	**(adr32 -- long)**	32 ビット数を adr32 から取り出します。
move	**(src dst u --)**	src から dst に u バイトをコピーします。
w!	**(n adr16 --)**	16 ビット数を adr16 に格納します。adr16 は 16 ビット境界でなければなりません。
w@	**(adr16 -- word)**	16 ビット数を adr16 から取り出します。adr16 は 16 ビット境界でなければなりません。
x!	**(o oaddr --)**	64 ビット数を oaddr に格納します。 oaddr は 64 ビット境界でなければなりません。
x@	**(oaddr -- o)**	64 ビット数を oaddr から取り出します。 oaddr は 64 ビット境界でなければなりません。

メモリー割り当てコマンド

表 1–24 メモリー割り当てコマンド

alloc-mem	**(size -- virt)**	size バイトの空きメモリーを割り当てます。割り当てた仮想アドレスを返します。free-mem によりマップを解除します。
free-mem	**(virt size --)**	alloc-mem で割り当てられていたメモリーを開放します。
free-virtual	**(virt size --)**	memmap により作成した割り当てを取り消します。
map?	**(virt --)**	仮想アドレスのメモリー割り当て情報を表示します。
memmap	**(phys space size -- virt)**	物理アドレスの領域を割り当てます。割り当てられた仮想アドレスを返します。free-virtual により割り当てを解除します。
obio	**(-- space)**	割り当て用にデバイスアドレス空間を指定します。
obmem	**(-- space)**	割り当て用にオンボードメモリアドレス空間を指定します。
pgmap!	**(pmentry virt --)**	仮想アドレス用に新しいページ割り当てエントリを格納します。
pgmap?	**(virt --)**	仮想アドレスに対応する復号化されたページ割り当てエントリを表示します。
pgmap@	**(virt -- pmentry)**	仮想アドレス用に新しいページ割り当てエントリを返します。
pagesize	**(-- size)**	ページのサイズ (8K の場合が多い)、を返します。
sbus	**(-- space)**	割り当て用に SBus アドレス空間を指定します。

ワード定義

表 1–25　ワード定義

: *name*	**(--)Usage: (??? -- ?)**	新しいコロン定義の作成を開始します。
;	**(--)**	新しいコロン定義の作成を終了します。
buffer: *name*	**(size --)Usage: (-- adr)**	指定された配列を一時記憶領域に作成します。
constant *name*	**(n --)Usage: (-- n)**	定数 (たとえば、3 constant bar) を定義します。
create *name*	**(--)Usage: (-- adr)**	汎用定義ワード
defer *name*	**(--)Usage: (??? -- ?)**	前方参照、または実行ベクトルのワードを定義します。
value *name*	**(n --)Usage: (-- n)**	指定された、変更可能な数値を作成します。
variable *name*	**(--)Usage: (-- adr)**	変数を定義します。

辞書検索コマンド

表 1–26　辞書検索コマンド

' name	**(-- xt)**	指定したワードを辞書から検索します。実行トークンを返します。定義外で使用してください。
['] name	**(-- xt)**	定義内で使用できる点以外は、' と同じです。
.calls	**(xt --)**	実行トークンが xt であるワードを呼び出すすべてのワードリストを表示します。

表 1–26　辞書検索コマンド　続く

$find	**(adr len -- adr len false | xt n)**	ワードを検索します。見つからなかった場合 n=0、見つかった場合 n=1、それ以外の場合は n=-1 になります。
see thisword	**(--)**	指定されたコマンドを逆コンパイルします。
(see)	**(xt --)**	実行トークンによって示されるワードを逆コンパイルします。
sifting ccc	**(--)**	指定された文字処理を含むすべての辞書エントリの名前を表示します。ccc 内には空白文字は含まれません。
words	**(--)**	指定された文字処理を含むすべての辞書エントリの名前を表示します。ccc 内には空白文字は含まれません。

テキスト文字列の操作

表 1–27　テキスト文字列の操作

" *ccc*"	**(-- adr len)**	入力ストリーム文字列をまとめます。
." *ccc*"	**(--)**	文字列を表示のためにコンパイルします。
bl	**(-- char)**	空白文字の ASCII コード。10 進の 32。
count	**(pstr -- adr +n)**	パックされている文字列をアンパックします。
p" *ccc*"	**(-- pstr)**	入力ストリームから文字列をまとめ、パックされた文字列として格納します。

辞書コンパイルコマンド

表 1–28　辞書コンパイルコマンド

,	**(n --)**	数値を辞書に入れます。
c,	**(byte --)**	バイトを辞書に入れます。
w,	**(word --)**	16 ビット数値を辞書に入れます。
l,	**(long --)**	32 ビット数値を辞書に入れます。
allot	**(n --)**	辞書に n バイトを割り当てます。
forget name	**(--)**	辞書からワードと、そのあとのすべてのワードを削除します。
here	**(-- adr)**	辞書の先頭アドレス。
to *name*	**(n --)**	defer ワードまたは value に新しい処理を実装します。
patch *new-word old-word word-to-patch*	**(--)**	old-word を word-to-patch の new-word に置き換えます。
(patch)	**(new-n old-n xt --)**	old-n を xt によって示されるワードの new-n に置き換えます。

テキスト入力の制御

表 1–29　テキスト入力制御

(*ccc*)	**(--)**	コメントを開始します。
\ rest-of-line	**(--)**	行の残りの部分をコメントとして扱います。
ascii *ccc*	**(-- char)**	次のワードの最初の ASCII 文字の数値を得ます。
key	**(-- char)**	コンソール入力デバイスから 1 文字を読みます。
key?	**(-- flag)**	コンソール入力デバイスでキーが押された場合 true。

テキスト出力の表示

表 1–30　テキスト出力表示

cr	**(--)**	ディスプレイ上の 1 行を終了し、次の行に進みます。
emit	**(char --)**	文字を表示します。
type	**(adr +n --)**	n 箇の文字を表示します。

入出力先の変更

表 1–31　入出力先の変更

input	**(dev-spec --)**	以降の入力に使用されるデバイス (ttya、ttyb、keyboard、または " *dev-speci*") を選択します。
io	**(dev-spec --)**	以降の入出力に使用されるデバイスを選択します。
output	**(dev-spec --)**	以降の出力に使用されるデバイス (ttya、ttyb、screen、または" *dev-spec*") を選択します。

比較コマンド

表 1–32　比較コマンド

<	**(n1 n2 -- flag)**	n1 < n2 の場合 true。
<=	**(n1 n2 -- flag)**	n1 <= n2 の場合 true。
<>	**(n1 n2 -- flag)**	n1 が n2 に等しくない場合 true。
=	**(n1 n2 -- flag)**	n1 = n2 の場合 true。
>	**(n1 n2 -- flag)**	n1 > n2 の場合 true。
>=	**(n1 n2 -- flag)**	n1 >= n2 の場合 true。
between	**(n min max -- flag)**	min <= n <= max の場合 true。
u<	**(u1 u2 -- flag)**	u1 < u2 の場合 true。u1、u2 とも符号なし。
u<=	**(u1 u2 -- flag)**	u1 <= u2 の場合 true、符号なし。
u>	**(u1 u2 -- flag)**	u1 > u2 の場合 true、符号なし。

表 1–32　比較コマンド　続く

u>=	**(u1 u2 -- flag)**	u1 >= u2 の場合 true 。符合なし。
within	**(n min max -- flag)**	min <= n < maxの場合 true。

if...then...else コマンド

表 1–33　if...then...else コマンド

else	**(--)**	if の条件が成り立たない場合、次のコードを実行します。
if	**(flag --)**	flag が true の場合、次のコードを実行します。
then	**(--)**	if...then...elseを終了します。

begin (条件付き) ループコマンド

表 1–34　begin (条件付き) ループコマンド

again	**(--)**	begin...again 無限ループを終了します。
begin	**(--)**	begin...while...repeat、begin...until、またはbegin...againループを開始します。
repeat	**(--)**	begin...while...repeat ループを終了します。
until	**(flag --)**	flag が true の間、 begin...until ループの実行を続けます。
while	**(flag --)**	flag が true の間、 begin...while...repeat ループの実行を続けます。

表 1–34　begin (条件付き) ループコマンド　続く

do (カウント付き) ループコマンド

表 1–35　do (カウント付き) ループコマンド

+loop	**(n --)**	do...+loop 構造を終了します。ループインデックスに n を加算し、 do に戻ります (n < 0 の場合は、インデックスは start から end まで変わります)。
?do	**(end start --)**	?do...loop の 0 回またはそれ以上の実行を開始します。インデックスは start から end-1 まで変わります。end = start の場合はループは実行されません。
do	**(end start --)**	do...loopを開始します。インデックスは start から end-1 まで変わります。例： 10 0 do i . loop (0 1 2...d e f と出力します)。
i	**(-- n)**	ループインデックス。
j	**(-- n)**	1 つ外側のループのループインデックス。
leave	**(--)**	do...loop から抜けます。
loop	**(--)**	do...loop の終わり。

case 文

```
( 値 )
case
2  of  ." it was two" endof
0  of  ." it was zero" endof
." it was " dup .  (省略可能なデフォルト節)
endcase
```

プログラム実行制御コマンド

表 1–36 プログラム実行制御コマンド

abort	**(--)**	現在の実行を終了させ、キーボードコマンドを解釈します。
abort" ccc"	**(abort? --)**	flag が true の場合は、実行を終了させ、メッセージを表示します。
eval	**(adr len --)**	配列から Forth のソースを解釈します
execute	**(xt --)**	実行トークンがスタックにあるワードを実行します。
exit	**(--)**	現在のワードから復帰します。(カウント付きループでは使用できません。)
quit	**(--)**	abortと同じ。スタックをそのままにします。

代替アドレス空間アクセスコマンド

表 1–37 代替アドレス空間アクセスコマンド

spacec!	**(byte adr asi --)**	1 バイトを asi とアドレスに格納します。
spacec@	**(adr asi -- byte)**	1 バイトを asi とアドレスから取り出します。
spaced!	**(n1 n2 adr asi --)**	2 つの 値を asi とアドレスに格納します。順序は実装によります。
spaced@	**(adr asi -- n1 n2)**	2 つの 値を asi とアドレスから取り出します。順序は実装によります。
spacel!	**(long adr asi --)**	32 ビットロングワードを asi とアドレスに格納します。

表 1–37　代替アドレス空間アクセスコマンド　続く

spacel@	**(adr asi -- long)**	32 ビットロングワードを asi とアドレスから取り出します。
spacew!	**(word adr asi --)**	16 ビットワードを asi とアドレスに格納します。
spacew@	**(adr asi -- word)**	16 ビットワードを asi とアドレスから取り出します。
spacex!	**(x adr asi --)**	64 ビットワードを asi とアドレスに格納します。
spacex@	**(adr asi -- x)**	64 ビットワードを asi とアドレスから取り出します。

キャッシュ操作コマンド

表 1–38　キャッシュ操作コマンド

clear-cache	**(--)**	すべてのキャッシュエントリを無効にします。
cache-off	**(--)**	キャッシュを使用不可にします。
cache-on	**(--)**	キャッシュを使用可能にします。
flush-cache	**(--)**	保留状態のデータをキャッシュから書いて戻します。

マルチプロセッサコマンド

表 1–39　マルチプロセッサコマンド

switch-cpu	**(cpu# --)**	指定された CPU に切り替えます。

表 1–39　マルチプロセッサコマンド　続く

プログラム実行制御コマンド

表 1–40　プログラム実行制御コマンド

abort	**(--)**	現在の実行を終了させ、キーボードコマンドを解釈します。
abort" ccc"	**(abort? --)**	flag が true の場合は、実行を終了させ、メッセージを表示します。
eval	**(adr len --)**	配列から Forth のソースを解釈します。
execute	**(xt --)**	実行トークンがスタックにあるワードを実行します。
exit	**(--)**	現在のワーから復帰します。(カウント付きループでは使用できません。)
quit	**(--)**	abort と同じ。スタックをそのままにします。