OpenBoot 2.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 805-5647
1998年11月(Revision A)

Copyright 1998 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

本製品およびそれに関連する文書は著作権法により保護されており、その使用、複製、頒布および逆コンパイルを制限するライセンスのもとにおいて頒布されます。日本サン・マイクロシステムズ株式会社の書面による事前の許可なく、本製品および関連する文書のいかなる部分も、いかなる方法によっても複製することが禁じられます。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company Limited が独占的にライセンスしている米国ならびに他の国における登録商標です。本製品のフォント技術を含む第三者のソフトウェアは、著作権法により保護されており、提供者からライセンスを受けているものです。

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

本製品は、株式会社モリサワからライセンス供与されたリュウミン L-KL (Ryumin-Light) および中ゴシック BBB (GothicBBB-Medium) のフォント・データを含んでいます。

本製品に含まれる HG 明朝 L と HG ゴシック B は、株式会社リコーがリョービイマジクス株式会社からライセンス供与されたタイプフェースマスタをもとに作成されたものです。平成明朝体 W3 は、株式会社リコーが財団法人 日本規格協会 文字フォント開発・普及センターからライセンス供与されたタイプフェースマスタをもとに作成されたものです。また、HG 明朝 L と HG ゴシック B の補助漢字部分は、平成明朝体W3 の補助漢字を使用しています。なお、フォントとして無断複製することは禁止されています。

Sun、Sun Microsystems、SunSoft、SunDocs、SunExpress、SunOS、OpenWindows、SunPics、DeskSet、ONC、ONC+、Power Management、TurboGX、TurboGX Plus、S24、SunFastEthernet、SunSwift、NFS、JumpStart、AnswerBook、Magnify Help、Sun Workstation、SunCD、SunCD Plus、SunCD 2Plus、SunInstall、SunView、SunLink、SunPHIGS、XGL、SLC、ELC、IPC、IPX、SunVideo、SUNprint、NeWSprinter、NeWSprinter CL+、NeWSprint、CDmanager、SunDiag、X11/NeWS、ToolTalk 、NeWS、SunNet Manager、Sun-3、Sun-4、Solstice、SuperCache、SunVTS、Sun RSM 、Ultra、UltraComputing、UltraServer、Ultra Enterprise 、PDB、SyMON、Netra、Netra NFS SmartServe、WebNFS、PC-NFSpro、Sun Enterprise Network Array、SunSwitch は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

サンのロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャーに基づくものです。

OPENLOOK、OpenBoot、JLE は、日本サン・マイクロシステムズ株式会社の登録商標です。

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフトウェアです。

Wnn6 は、オムロン株式会社で開発されたソフトウェアです。(Copyright OMRON Co., Ltd. 1997 All Rights Reserved.)

ATOK は、株式会社ジャストシステムの登録商標です。

ATOK7 は株式会社ジャストシステムの著作物であり、ATOK7 にかかる著作権その他の権利は、すべて株式会社ジャストシステムに帰属します。

ATOK8 は株式会社ジャストシステムの著作物であり、ATOK8 にかかる著作権その他の権利は、すべて株式会社ジャストシステムに帰属します。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問い合わせください。

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザーおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたは グラフィカル・ユーザーインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、 このライセンスは米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されない、明示的であるか黙示的であるかを問わない、なんらの保証も行われないものとします。

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物または役務) に該当する場合、本製品を輸出または日本国外へ持ち出す際には、日本サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典 : OpenBoot 2.x Quick Reference

Part No: 805–4435

Revision A

1998 by Sun Microsystems, Inc.

Adobe PostScript

目次

第 1 章

OpenBoot 2.x の手引き

構文

コマンドを ok プロンプトの後に入力して改行キーを押すと、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 つ以上のスペースで区切ってください。

ヘルプコマンド

表 1–1 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)

制限付きモニターコマンド

表 1–2　制限付きモニターコマンド

`b` [*specifiers*]	オペレーティングシステムを起動します (`ok` プロンプトでの `boot`と同じです)。
`c`	停止しているプログラムの実行を再開します (`ok` プロンプトでの `go` と同じです)。
`n`	Forth モニターに入ります。

デバイス別名の検査と作成

表 1–3　デバイス別名の検査と作成

`devalias`	現在のデバイス別名をすべて表示します。
`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。

デバイスツリー表示コマンド

表 1–4 デバイスツリー表示コマンド

`.attributes`	現在のノードの特性の名前と値を表示します。
`cd` *device-path*	指定されたデバイスノードを選択し、それを現在のノードにします。
`cd` *node-name*	指定されたノード名を現在のノードの下の サブツリーで検索し、最初に見つかったノードを選択します。
`cd ..`	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。
`cd /`	ルートマシンノードを選択します。
`device-end`	現在のデバイスノードを選択解除し、ノードが選択されていない状態にします。
`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*]	デバイス階層内の指定されたレベルのすぐ下の、システムに認識されているすべてのデバイスを表示します。(`show-devs` だけを使用すると、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 コマンドの共通オプション

表 1–5　boot コマンドの共通オプション

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) `floppy` (3.5 インチディスクドライブ) `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` 変数の値を使用します。
[*options*]	`-a`: デバイスと起動ファイル名を聞いてきます。 `-h` : プログラムを読み取り後、停止します。 (これらは OS 固有のオプションで、システムによって異なります)

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。
`probe-scsi-all` [*device-path*]	システムの、指定したノードの下にインストールされているすべての SCSI バスに対して `probe-scsi` を実行します。 (*device-path* を指定しない場合、ルートノードが使用されます)
`test` *device-specifier*	指定したデバイスの自己診断テストを 実行します。 例 `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 /memory` : `selftest-#megs` で指定されているメガバイト数をテストします。 `diag-switch?` が `true` の場合、すべてのメモリーをテストします。 `test net` : ネットワークの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 (組み込み自己診断テストを備える) 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)
`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。

システム情報表示コマンド

表 1–7 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。

緊急時キーボードコマンド

表 1–8 緊急時キーボードコマンド

電源投入処理中に次のキーを押してください。

`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`	プローブを行なわず、ttya で Forth に入ります。`fexit` を使用して、初期設定処理を続けます。(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。

ファイル読み取りコマンド

表 1–9 ファイル読み取りコマンド

`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 span --)**	読み込まれた FCode バイナリファイルを解釈します。`span` 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。次のように入力してください。 `~C cat` *filename*

表 1–9　ファイル読み取りコマンド　続く

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から読み込みます。
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のメモリーにデータを読み込みます。(boot の書式を参照してください)
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

%f0 ～ %f31	**(-- value)**	指定された浮動小数点レジスタの値を返します。
%fsr	**(-- value)**	指定された浮動小数点レジスタの値を返します。
%g0 ～ %g7	**(-- value)**	指定されたレジスタの値を返します。
%i0 ～ %i7	**(-- value)**	指定されたレジスタの値を返します。
%L0 ～ %L7	**(-- value)**	指定されたレジスタの値を返します。
%o0 ～ %o7	**(-- value)**	指定されたレジスタの値を返します。
%pc %npc %psr	**(-- value)**	指定されたレジスタの値を返します。
%y %wim %tbr	**(-- value)**	指定されたレジスタの値を返します。

表 1–10　SPARC レジスタコマンド　続く

.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。
.psr	**(--)**	%psr data を書式付きで表示します。
.registers	**(--)**	%g0 から %g7 までのほかに、 %pc、%npc、%psr、%y、%wim、%tbr の値を表示します。
.window	**(window# --)**	希望するウィンドウを表示します。
ctrace	**(--)**	C サブルーチンを示すリターンスタックを表示します。
set-pc	**(value --)**	%pc を指定された値に、%npc を (指定された値 +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# --)**	現在のウィンドウに %ix %Lx または %ox を表示するようにします。

ブレークポイントコマンド

表 1–11　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の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。

表 1–11 ブレークポイントコマンド 続く

.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 は .breakpoint です)
.instruction	(--)	最後に発生したブレイクポイントのアドレス、opcode 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。 (.breakpoint を参照。)
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設定することにより、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。) サブルーチン呼出しを 1 つの命令として扱います。
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	(--)	現在の命令を省略します (実行しません)。
step	(--)	1 ステップ 1 命令です。
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あたるまで実行します。+bp go と同じです。

逆アセンブラコマンド

表 1–12 逆アセンブラコマンド

+dis	**(--)**	最後に逆アセンブルを中断したところから、逆アセンブルを継続します。
dis	**(adr --)**	指定されたアドレスから、逆アセンブルを開始します。

その他の処理

表 1–13 その他の処理

eject-floppy	**(--)**	フロッピーディスクをドライブから取り出します。
firmware-version	**(-- n)**	メジャー/マイナー CPU ファームウェアバージョンを表示します。(0x00020001 = ファームウェアバージョン 2.1)
ftrace	**(--)**	例外発生時の呼出し順序を表示します。
get-msecs	**(-- ms)**	現在のミリ秒 (ms) 単位の概略時間を返します。
ms	**(n --)**	n ミリ秒 (ms) 遅延させます。分解能は 1 ミリ b 秒 (ms) です。
reset	**(--)**	システム全体をリセットします (電源再投入と似ています) 。
sync	**(--)**	オペレーティングシステムを呼出して、保留情報をハードディスクに書き出します。さらに、ファイルシステム間の同期が取れたら起動します。

NVRAM 設定変数

表 1–14　NVRAM 設定変数

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-device	**disk**	起動するデバイス。
boot-file	**empty string**	起動するファイル。(空白の場合、第二起動プログラムがデフォルトを選択します)
boot-from	**vmunix**	起動デバイスとファイルを指定します。 (1.*x* のみ)
boot-from-diag	**le()vmunix**	診断起動デバイスとファイル (1.*x* のみ)
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ル。
diag-switch?	**false**	true の場合、診断モードで実行します。
fcode-debug?	**false**	true の場合、追加デバイスの FCodes の名前フィールドを取り入れます。
hardware-revision	**no default**	システムバージョン情報。
input-device	**keyboard**	電源投入時の入力デバイス。(通常 keyboard、ttya、または ttyb)
keyboard-click?	**false**	true の場合、キーボードクリックを使用可能にします。
keymap	**no default**	キーボードのカスタマイズ用キー割り当て。
last-hardware-update	**no default**	システム更新情報。

表 1–14　NVRAM 設定変数　続く

local-mac-address?	**false**	true の場合、ネットワークドライバはシステムのアドレスではなく、自分自身の MAC アドレスを使用します。
mfg-switch?	**false**	true の場合、 Stop-A で中断されるまでシステムの自己診断を繰り返します。
nvramrc	**empty**	NVRAMRC の内容。
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 の場合、カスタム OEM バナーを使用します。
oem-logo	**no default**	バイト配列カスタム OEM ロゴ。(oem-logo? が true なら使用可能です) 16 進で表示。
oem-logo?	**false**	true の場合、カスタム OEM ロゴを使用します。(true でない場合は、サンのロゴを使用します)
output-device	**screen**	電源投入時の出力デバイス。(通常は screen、ttya、または ttyb)
sbus-probe-list	**0123**	プローブする SBus スロットがどのような順番でプローブされたかを表します。
screen-#columns	**80**	画面上のカラム数。(1 行あたりの文字数)
screen-#rows	**34**	画面上の列数。(行数)
scsi-initiator-id	**7**	ホストアダプタの SCSI バスアドレス。 範囲 0-7。
sd-targets	**31204567**	SCSI ディスクユニットの割り当て。(1.*x* のみ)

表 1–14 NVRAM 設定変数 続く

security-#badlogins	**no default**	誤ったセキュリティーパスワードの試行回数。
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) 設定を直接行わないでください。
selftest-#megs	**1**	テストする RAM のメガバイト数。 diag-switch? が true の場合、無視されます。
skip-vme-loopback?	**false**	true の場合、 POST は VMEbus ループバックテストを行いません。
st-targets	**45670123**	SCSI テープユニットの割り当て (1.*x* のみ)
sunmon-compat?	**false**	true の場合、制限付きモニタープロンプト(>) を表示します。
testarea	**0**	NVRAM テストのための 1 バイトスクラッチフィールド。
tpe-link-test?	**true**	組み込み 10BASE-T Ethernet の接続テストを有効にします。
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、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、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。

表 1–14　NVRAM 設定変数　続く

ttya-rts-dtr-off	**false**	true の場合、OS は ttya 上で DTR、RTS を使用しません。
ttyb-rts-dtr-off	**false**	true の場合、OS は ttyb 上で DTR、RTS を使用しません。
use-nvramrc?	**false**	true の場合、システムの初期設定中に NVRAMRC にあるコマンドを実行します。
version2?	**true**	true の場合、ハイブリッド (1.*x*/2.*x*) PROM が バージョン 2.*x* で使われます。
watchdog-reboot?	**false**	true の場合、ウォッチドッグリセット後に再起動します。

表示、変更設定パラメタ

表 1–15　表示、変更設定パラメタ

printenv	現在の変数とデフォルトの値をすべて表示します (数字は通常 10 進で表されます) 。 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＞	変数に 10 進、またはテキストの値を設定します。 (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＞	指定する変数の値を工場出荷時のデフォルトに設定します。
set-defaults	変数設定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 "`devalias` *alias device-path*" コマンドを格納します。(別名は `nvunalias` または `set-defaults` が実行されるまで有効です)
`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に残っている場合、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、それらの編集から開始します。
`nvquit`	一時バッファーの内容を NVRAMRC に書き込まないで捨てます。捨てる前に、確認のためのプロンプトが表示されます。
`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ます。それから `nvedit` でエディタを起動します。`nvrecover` は、 `nvedit` が NVRAMRC の内容が失われた時間と `nvrecover` が実行された時間との間に実行された場合には、失敗します。
`nvrun`	一時バッファーの内容を実行します。
`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、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**				
すべての一致を表示				**^/** または **^?}**				

esc = エスケープキーを最初に押して離す

^ = コントロールキーを押したままにする

NVRAMRC エディタの使用

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)

数値の用法とスタックコメント

- 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。
- 現在どちらの進法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。

各コマンドの後ろにある (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 1–18 数値の用法とスタックコメント

|	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。(複数)
`acf`	コードフィールドアドレス。
`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 ビットの値 (32 ビットワードの下位バイト)

表 1–18　数値の用法とスタックコメント　続く

char	7 ビットの値 (下位バイト)、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 そのほかのすべての場合、true (通常 -1)
long L*xxx*	32 ビット値。
n n1 n2 n3	符号付きの値。(32 ビット)
+n u	符号なしの正の値。(32 ビット)
n[64] or (n.low n.hi)	拡張精度 (64 ビット) の数。(2 スタック項目)
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 はパックされない文字列のアドレスと長さ。)
virt	仮想アドレス。(ソフトウェアが使用するアドレス)
word w*xxx*	16ビットの値。(32 ビットワードの下位 2 バイト)

基数の変更

表 1–19　基数の変更

decimal	(--)	基数を 2 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 進に設定します。
h# *number*	(-- n)	次の数を 16 進で解釈します。基数は変わりません。

表 1–19　基数の変更　続く

.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。

スタック操作コマンド

表 1–21　スタック操作コマンド

-rot	**(n1 n2 n3 -- n3 n1 n2)**	3 つのスタック項目を逆回転させます。
>r	**(n --)**	スタック項目を復帰スタックに移動します。(使用には注意が必要です。)
?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 --)**	スタックから 2 項目を削除します。
2dup	**(n1 n2 -- n1 n2 n1 n2)**	2 スタック項目を複写します。

表 1–21　スタック操作コマンド　続く

2over	**(n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2)**	2 番目以降のスタック項目をコピーします。
2swap	**(n1 n2 n3 n4 -- n3 n4 n1 n2)**	2 対の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。
nip	**(n1 n2 -- n2)**	2 番目のスタック項目を捨てます。
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複製します。
pick	**(??? +n -- ??? n2)**	+n 番目の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。(使用には注意が必要です。)
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	**(??? +n -- ?)**	+n 個のスタック項目を回転させます。(2 roll = rot)。
rot	**(n1 n2 n3 -- n2 n3 n1)**	3 スタック項目を回転させ 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。

単精度演算機能

表 1–22　単精度演算機能

*	**(n1 n2 -- n3)**	乗算。 n1 * n2。
+	**(n1 n2 -- n3)**	加算。n1 + n2。
-	**(n1 n2 -- n3)**	減算。n1 - n2。
/	**(n1 n2 -- quot)**	除算。n1 / n2 。剰余は破棄されます。
<<	**(n1 +n -- n2)**	n1 を +n ビット、左にシフトします。
>>	**(n1 +n -- n2)**	n1 を +n ビット、右にシフトします。
>>a	**(n1 +n -- n2)**	n1 を +n ビット、右に算術シフトします。
abs	**(n -- u)**	絶対値。
and	**(n1 n2 -- n3)**	ビット単位の 論理積。
bounds	**(startadr len -- endadr startadr)**	do ループ用に、 startadr len を endadr startadrに変換します。
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作ります。
bwjoin	**(b.low b.hi -- word)**	2 バイトを結合して、16 ビットのワードを作ります。
lbsplit	**(long -- b.low b2 b3 b.hi)**	32 ビットのロングワードを分割して、4 バイトにします。
lwsplit	**(long -- w.low w.hi)**	32 ビットのロングワードを分割して、16 ビットのワード 2 つにします。
max	**(n1 n2 -- n3)**	n1 と n2 の大きいほうの値を n3 とします。
min	**(n1 n2 -- n3)**	n1 と n2 の大きいほうの値を n3 とします。

表 1–22　単精度演算機能　続く

mod	**(n1 n2 -- rem)**	n1 / n2 の剰余。
negate	**(n1 -- n2)**	n1 の符号を変更します。
not	**(n1 -- n2)**	ビット単位の 1 の補数。
or	**(n1 n2 -- n3)**	ビット単位の 論理和。
wbsplit	**(word -- b.low b.hi)**	16ビットのロングワードを分割して、2 バイトにします。
wljoin	**(w.low w.hi -- long)**	2 ワードを結合して、ロングワードにします。
xor	**(n1 n2 -- n3)**	ビット単位の 排他的論理和。

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 adr16 --)**	32 ビットの数を adr16 に格納します。16 ビットに境界でなければなりません。
+!	**(n adr16 --)**	adr16 に保存されている 32 ビットの数値に n を加算します。16 ビットでなければなりません。
@	**(adr16 -- n)**	32 ビット数値を adr16 から取り出します。16 ビットに境界でなければなりません。
c!	**(n adr --)**	nの下位バイトを adr に保存します。
c@	**(adr -- byte)**	1 バイトを adr から 取り出します。

表 1–23　メモリーアクセスコマンド　続く

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、データと true を返します。読み取りエラーが発生した場合は、false を返します。(lpeek、wpeek も同様)
cpoke	**(byte adr -- okay?)**	1 バイトを adr にバイトを格納します。アクセスが成功した場合、true を返します。書き込みエラーが発生した場合は、false を返します。 (lpoke、wpoke も同様)
comp	**(adr1 adr2 len -- n)**	2 つのバイト配列を比較します。両配列が等しい場合は n = 0、最初の異なるバイトが配列 #1 側より小さい場合は n = 1、それ以外の場合は n = -1 になります。
dump	**(adr len --)**	adr から始まる len バイト表示します。
fill	**(adr size byte --)**	size メモリーバイトを byte に設定します。
L!	**(n adr32 --)**	adr32 に 32 ビットの数値を格納します。
L@	**(adr32 -- long)**	32 ビットの数値を adr32 から取り出します。
move	**(adr1 adr2 u --)**	adr1 から adr2 へ u バイト分、コピーします。オーバーラップは適切に処理されます。
w!	**(n adr16 --)**	16 ビットの数値を adr16 に格納します。16 ビット境界でなければなりません。
w@	**(adr16 -- word)**	16 ビットの数値を adr16 から取り出します。16 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)**	使用可能なメモリーを size バイト分割り当てます。仮想アドレスを返します。free-mem を取り消します。
cacheable	**(space -- cache-space)**	次のアドレス割り当てがキャッシュ可能になるように、アドレス空間を変更します。
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 で作成されてい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。割り当てた仮想アドレスを返します。 free-virtual で割り当てを解除します。
obio	**(-- space)**	割り当てるデバイスアドレス空間を指定します。
obmem	**(-- space)**	割り当てるボード上のメモリーアドレス空間を指定します。
pgmap!	**(pmentry virt --)**	仮想アドレスに対する新しいページ割り当てエントリを返します。
pgmap?	**(virt --)**	仮想アドレスに対応するページ割り当てエントリ (復号化された英語)を表示します。
pgmap@	**(virt -- pmentry)**	仮想アドレスに対応するページ割り当てエントリを返します。
pagesize	**(-- size)**	ページサイズを返します。(通常 4K)
sbus	**(-- space)**	割り当てる SBus のアドレス空間を指定します。

ワードの定義

表 1–25　ワードの定義

: 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: (-- adr64)**	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)**	定数を定義します。(例えば 3 constant bar)
create name	**(--)Usage: (-- adr16)**	汎用定義ワード。
defer name	**(--)Usage: (??? -- ?)**	前方参照、または実行ベクトルを定義します。
does>	**(-- adr16)**	ワード定義用の実行節を開始します。
value name	**(n --)Usage: (-- n)**	指定された、変更可能な 32 ビット数を作成します。
variable name	**(--)Usage: (-- adr16)**	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	**(-- acf)**	指定されたワードを辞書から検索します。(コードフィールドのアドレスを返します。外部定義を使用してください)
['] name	**(-- acf)**	'と似ていますが、内部定義にも外部定義にも使用されます。
.calls	**(acf --)**	コンパイルアドレスが acf のワードを呼び出すすべてのワードのリストを表示します。
$find	**(adr len -- adr len false acf n)**	ワードを検索します。検索できなかった場合 n = 0、直接の場合 n = 1、その他は n = -1。
see thisword	**(--)**	指定されたコマンドを逆コンパイルします。
(see)	**(acf --)**	コードフィールドアドレスで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	**(--)**	辞書内の表示可能なワードをすべて表示します。

辞書編集コマンド

表 1–27　辞書編集コマンド

,	**(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。

表 1–27　辞書編集コマンド　続く

allot	**(n --)**	辞書に n バイトを割り当てます。
forget name	**(--)**	辞書から指定されたワードとそれ以降の全ワードを削除します。
here	**(-- adr)**	辞書の先頭アドレス。
is name	**(n --)**	defer ワード、または value の新しい処理を実装します。
patch *new-word old-word word-to-patch*	**(--)**	*word-to-patch* 内の *old-word* を *new-word* に置き換えます。
(patch	**(new-n old-n acf --)**	acf で示されるワード内の old-n を new-n に置き換えます。

テキスト入力の制御

表 1–28　テキスト入力の制御

(*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)**	割り当てられた入力デバイスのキーボードから 1 文字を読み取ります。
key?	**(-- flag)**	入力デバイスのキーボードからキー入力された場合、true。

テキスト出力の表示

表 1–29 テキスト出力の表示

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 文字を表示します。

テキスト文字列の操作

表 1–30 テキスト文字列の操作

" *ccc*"	**(-- adr len)**	解釈結果またはコンパイル結果の入力ストリーム文字列をまとめます。文字列内では、"(00,ff...) を使用して任意のバイト値を取り入れることができます。
." *ccc*"	**(--)**	後の表示に備えて、文字列をコンパイルします。
bl	**(-- char)**	空白文字の ASCII コード。10 進で 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の文字列をまとめ、パックされた文字列として格納します。

入出力先の変更

表 1–31　入出力先の変更

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

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 <> n2 の場合、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 (符号なし) の場合、true。

表 1–32　比較コマンド　続く

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

if...then...else コマンド

表 1–33　if...then...else コマンド

else	**(--)**	比較が成立しなかった場合、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 あるいは 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 になるまで、 begin...until ループの実行を継続します。
while	**(flag --)**	flag が true の間、 begin...while...repeat ループの実行を継続します。

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 を加算して、do に戻ります。(n < 0 の場合、インデックスは start から end まで変わります)
?do	**(end start --)**	0 回、またはそれ以上の ?do...loop 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、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。次に例を示します。 10 0 do i . loop (prints 0 1 2...d e f)
i	**(-- n)**	ループインデックス。
j	**(-- n)**	一つ外側のループのインデックス。
leave	**(--)**	do...loop から抜けます。
loop	**(--)**	do...loop を終了します。

case 文

(value)

```
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
```

キャッシュ操作コマンド

表 1–36 キャッシュ操作コマンド

`clear-cache`	(--)	すべてのキャッシュ入力を無効にします。
`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。

アドレス空間アクセス代替コマンド

表 1–37　アドレス空間アクセス代替コマンド

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 --)**	asi と アドレスに 2 つの 32 ビットの ワードを格納します。順序は実装に依存 します。
spaced@	**(adr asi -- n1 n2)**	asi と アドレスから 2 つの 32 ビットの ワードを取り出します。順序は実装に依存します。
spaceL!	**(long adr asi --)**	32 ビットのロングワードを asi とアドレスに格納します。
spaceL@	**(adr asi -- long)**	32 ビットのロングワードを 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)**	asi と アドレスの 16 ビットワードを取り出します。

マルチプロセッサコマンド

表 1–38　マルチプロセッサコマンド

module-info	**(--)**	すべての CPU モジュールの型と速度を表示します。
switch-cpu	**(cpu# --)**	指示された CPU に切り替えます。

プログラム実行制御コマンド

表 1–39　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 ture 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 ソースを解釈します。
execute	**(acf --)**	コードフィールドアドレスがスタック上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内では使用できません。)
quit	**(--)**	スタック内容をまったく変えない点を除いて、abort と同じです。