OpenBoot 2.x コマンド・リファレンスマニュアル

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

Part Number 805-5646
1998年11月(Revision A)

原典 ：OpenBoot 2.x Command Reference

Part No: 805–4434

Revision A

目次

はじめに

『OpenBoot(TM) 2.x コマンド・リファレンスマニュアル』では、Sun™ システムの起動 PROM の一部である OpenBoot 2.x ファームウェアについて説明します。

対象読者

OpenBoot ファームウェアの機能は、システム管理者や開発者だけでなく、エンドユーザーも使用できます。このマニュアルは、OpenBoot 2.x ファームウェアを使用してシステムを構成したりデバッグするすべてのユーザーを対象にしています。

このマニュアルの内容

このマニュアルでは、OpenBoot ファームウェアを使用して次の作業を行う方法について説明します。

- オペレーティングシステムの起動
- 診断の実行
- システムを起動するためのシステム変数の変更
- プログラムの読み込みと実行
- 障害追跡

Forth プログラムを作成したり、このファームウェアの (デバッグ機能などの) より高度な機能を使用する読者のために、このマニュアルではさらに OpenBoot の Forth インタプリタのコマンドについても説明します。

お読みになる前に

このマニュアルでは、読者がバージョン 2.*x* の OpenBoot PROM を搭載する SPARC® システムを使用しているものとしています。このマニュアルで説明するツールや機能には、2.*x* より前の SPARC システムの PROM にはないものがあります。SPARCstation™ 1、SPARCstation IPC™ または、2. *x* より前のバージョンの PROM を使用しているシステムの場合、このマニュアルの前のバージョンである『Open Boot PROM Toolkit User's Guide』を参照してください。なお、このマニュアルの付録 C には、サポートされていないコマンドの一覧が載せてあります。

このマニュアルの構成

このマニュアルの構成は次のとおりです。

第 1 章では、OpenBoot ファームウェアのユーザーインタフェースなど、主要な機能について説明します。

第 2 章では、OpenBoot ファームウェアを使用する最も一般的な作業を説明します。

第 3 章では、NVRAM 変数を使ってどのようにシステム管理作業を実行するかについて、詳細に説明します。

第 4 章では、OpenBoot の Forth 言語の基本、および高度な機能について説明します。

第 5 章では、(Ethernet、ディスク、シリアルポートなどの) 様々なソースからプログラムを読み込み実行する方法について説明します。

第 6 章では、逆アセンブラ、Forth のソースレベルデバッガ、ブレークポイントを含む OpenBoot ファームウェアのデバッグ機能について説明します。

付録 Aでは、どのようにシリアルポートを使用してシステムを別のサン™ システムに接続するかについて説明します。

付録 Bでは、プログラムやファイルを読み込める起動可能なフロッピーディスクを作成する方法について説明します。

付録 Cでは、以前の OpenBoot システムでは利用できなかったコマンド、それらコマンドの機能を実現するための対処方法の一覧を示します。

付録 Dでは、オペレーティングシステムを起動できない代表的な状況に対する解決方法について検討します。

付録 Eには、現在サポートされているすべての OpenBoot の Forth コマンドを示します。

関連マニュアル

- このマニュアルと関連するマニュアル :
- 『OpenBoot 2.x の手引き』

この手引きは、よく使用される OpenBoot Forth コマンドの要約一覧です。

- FCode、つまり SBus カードを使用するための OpenBoot 2.x ファームウェアに実装された Forth についての詳細は、次のサンマニュアルを参照してください。
- 『Writing FCode 2.x Programs』

Forth 言語についての詳細は、次の刊行書をお読みください。

- 「ANSI X3.215-1994, American National Standard for Information Systems-Programming Languages-FORTH.」
- 『Starting FORTH』- 原書 (初版および第 2 版)

  Leo Brody. FORTH, Inc., second edition, 1987.

  Prentice-Hall Software Series

  Eaglewood Cliffs, New Jersey 07632
- 『FORTH 入門』- 翻語版 (初版のみ)

  レオ・ブロディ著

  原道宏訳

  工学社出版

『*Starting Forth*』の第 2 版 (未訳) では、現在の Forth の標準仕様である Forth 83 について説明しています。

注 - 上記の刊行書で説明している Forth のバージョンと、このマニュアルで説明しているバージョンにはいくつかの相違があります。特に起動 PROM Forth モニターは、16 ビットでなく、32 ビット版を使用しています。さらに、この刊行書で説明しているテキストエディタは、Forth モニターのエディタと異なります。

書体と記号について

このマニュアルで使用する書体と記号について説明します。

表 P–1 このマニュアルで使用している書体と記号

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system% You have mail.`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表します。	`system%` **`su`** `password:`
AaBbCc123	コマン行の可変部分。実際の名前または実際の値と置き換えてください。	`rm` *filename* と入力します。
『 』	参照する書名を示します。	『SPARCstorage Array ユーザーマニュアル』
「 」	参照する章、節、または強調する単語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
`%`	UNIX の C シェルのプロント。	`system%`
`$`	UNIX の Bourne シェルとKorn シェルのプロンプト。	`system$`

表 P–1 このマニュアルで使用している書体と記号 続く

字体または記号	意味	例
#	スーパーユーザーのプロンプト (シェルの種類を問わな問わない)。	`system#`
\	枠で囲まれたコーディング例で、テキストがページ行幅をこえる場合、バックスラッシュは継続を示します。	**`grep ‘^#define \ XV_VERSION_STRING’`**

第 1 章

概要

この章では、サンシステムの標準ファームウェアである OpenBoot ファームウェアを紹介します。

OpenBoot バージョン 1 は、サンの SPARCstation 1 で導入したファームウェアで、SPARCstation 1+、SPARCstation IPC、SPARCstation SLC™ の各システムでも使用されています。このマニュアルでは、バージョン 2 ファームウェアについて説明します。

このファームウェアは、SPARCstation 2 システムから導入されています。

OpenBoot ファームウェアは、システムの起動 PROM (プログラマブル読み取り専用メモリー) に格納されているため、システムの電源を入れるとただちに実行されます。OpenBoot ファームウェアの最初の動作は、大容量記憶装置またはネットワークからオペレーティングシステムを起動することです。このファームウェアは、その他にハードウェアとソフトウェアを対話的にテストするための豊富な機能も備えています。

OpenBoot の特長

OpenBoot のアーキテクチャーは、以前のサンシステムでの起動 PROM よりも、大きく拡張された機能を提供します。このアーキテクチャーは、SPARC システムで、はじめて導入したものですが、その設計はプロセッサに依存しません。次に、OpenBoot ファームウェアの主な機能をいくつか示します。

- 差し込み式デバイスのドライバ — 差し込み式デバイスのドライバは、通常、SBus カードなどの差し込み式デバイスから読み込まれます。差し込み式デバイスのドライバを使用して、そのデバイスからオペレーティングシステムを起動したり、オペレーティングシステムがそれ自身のドライバを起動する前にそのデバイスにテキストを表示できます。この機能によって、システム PROM を変更しないで、特定のシステムがサポートする入出力デバイスの機能を拡張できます。
- FCode インタプリタ — 追加ドライバは、*FCode*というマシンに依存しないインタプリタ言語で書かれています。各 OpenBoot システム PROM には FCoce インタプリタが含まれています。したがって、異なる CPU 命令セットを使用しているマシンに対して、同じデバイスとドライバを使用できます。
- デバイスツリー — デバイスツリーは、システムに接続されている (常時インストールされている差し込み式の) デバイスを記述する、OpenBoot のデータ構造です。ユーザーもオペレーティングシステムも、デバイスツリーを調べることによりシステムの構成を知ることができます。
- プログラマブルユーザーインタフェース — OpenBoot のユーザーインタフェースは、対話型プログラミング言語である *Forth*をベースにしています。ユーザーコマンドの処理を組み合わせて完全なプログラムを作り上げることができます。その結果、ハードウェアとソフトウェアのデバッグを行う強力な機能を提供します。

ユーザーインタフェース

次の方法で OpenBoot 環境に入ります。

- オペレーティングシステムを停止します。
- キーボードから `Stop-A` キー処理を使用します。(この方法では、オペレーティングシステムの実行がただちに停止するので、使用には注意が必要です。)
- システムに電源を再投入 (パワーサイクル) します。(システムが自動的に起動するようになっている場合は、ディスプレイコンソールバナーが表示された後、システムがオペレーティングシステムの起動を開始しないうちに `Stop-A` を押して OpenBoot 環境にしてください。自動起動するようになっていない場合は、システムはオペレーティングシステムを起動しないで、自動的に OpenBoot 環境になります。)
- システムのハードウェアが回復不可能なエラーを検出したとき。(これはウォッチドッグリセットとして知られています。)

OpenBoot ファームウェアには、次の 3 つの外部インタフェースがあります。

- オペレーティングシステムまたはその他のスタンドアロンプログラム用のインタフェース
- 拡張バス差し込み式ボード (たとえば SBus) 用のインタフェース
- システムコンソールから使用するユーザー用のコマンド行インタフェース

このマニュアルでは 3 つ目のインタフェースである、システムコンソールから使用するコマンド行インタフェースについて説明します。

コマンド行インタフェースには次の 2 つのモードがあります。

- 制限付きモニター
- Forth モニター

制限付きモニター

制限付きモニターでは、簡単なコマンドセットを使用して、システムの起動開始、システムの実行再開、または、Forth モニターに入ることができます。また、制限付きモニターはシステムのセキュリティーを設定するためにも使用します。(システムセキュリティーについては、第 3 章を参照してください。)

制限付きモニターのプロンプトは > です。制限付きモニターに入ると次の画面が表示されます。

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

制限付きモニターのコマンドの例を次の表に示します。

表 1–1 制限付きモニターコマンド

コマンド	説明
b [*specifiers*]	オペレーティングシステムを起動します。
c	停止しているプログラムの実行を再開します。
n	Forth モニタに入ります。

Forth モニター

制限付きモニターの機能 b (システムの起動) と c (停止しているプログラムの実行再開)は、Forth モニターではそれぞれ boot (第 2 章を参照) と go (第 5 章を参照) コマンドとして提供されています。

Forth モニターは対話型コマンドインタプリタで、これによりハードウェアおよびソフトウェア開発、障害の切り分け、デバッグ用の広範な機能にアクセスできます。エンドユーザーから、システム管理者、システム開発者にいたるまで、さまざまなシステムユーザーがこれらの機能を利用できます。

Forth モニターのプロンプトは ok です。Forth モニターに入ると次のメッセージが表示されます。

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

デフォルトモード

初期の OpenBoot システムでは、デフォルトモードは制限付きモニターでした。これは、OpenBoot より前のシステムに近いデフォルトのルック & フィールを提供するのが主な目的でした。

The SPARCserver™ 690 システムは、Forth モニターをデフォルトモードとして備える最初のシステムでした。それ以降に発表されたシステムはすべて、デフォルトでこのモードになります。これらのシステムにおいては、制限付きモニターの実用的な機能はシステムセキュリテイのサポートだけです。(システムセキュリティーについては、第 3 章で説明します。)

Forth モニターを終了して制限付きモニターに入るには、次のように入力します。

```
ok old-mode
```

デバイスツリー

デバイスは、相互に接続されたバス上で SPARC ベースのシステムに接続されています。OpenBoot ファームウェアは、接続されたバスとそれらのバスに接続されたデバイスをノードのツリーという形で表します。そのようなツリーをデバイスツリーと呼びます。マシン全体を表すノードが、ツリーのルートノードになります。

各デバイスノードには次があります。

- 特性 — ノードとそれに関連付けられているデバイスを記述するデータ構造
- 方法 —デバイスにアクセスするためのソフトウェア手続き
- 子 — あるノードに「接続され」ていて、デバイスツリーにおいてそのノードのすぐ下にある他のデバイスノード
- 親 — デバイスツリーにおいて、あるノードのすぐ上にあるノード

子があるノードは通常、複数のバスと、それらのバスに接続されているコントローラを表します。そのようなノードはそれぞれ、それらに接続されているデバイス相互間を区別する物理的なアドレス空間を定義します。そのノードの子には、親のアドレス空間内の物理アドレスがそれぞれ割り当てられます。

物理アドレスは、一般に (デバイスがインストールされているバスアドレスまたはスロット番号などの) デバイス固有の物理的性質を表します。これにより、他のデバイスをシステムにインストールしたときに、デバイスアドレスの変更を避けることができます。

デバイスのパス名、アドレス、引数

OpenBoot ファームウェアはシステム内のハードウェアデバイスを直接取り扱います。各デバイスには、デバイスの種類、システムアドレス構造内のそのデバイスの位置を表す固有の名前があります。次の例でデバイスのフルパス名を示します。

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

デバイスのフルパス名は、スラッシュ (/)で区切られた一連のノード名です。ツリーのルートは明示的には示されない、先頭のスラッシュ (/) で示されるマシンノードです。各ノード名は次の形式になっています。

name@address:arguments

表 1–2でこれらの変数を説明します。

表 1–2 デバイスパス名の変数

変数名	説明
name	理想的には、なんらかのニーモニック値をもつテキスト文字列。(たとえば、 `sd` は SCSI ディスクを表します。多くの名前、特に差し込み式モジュールの名前は、デバイスのメーカの名前または株式略称を含みます (たとえば、`SUNW`)。
`@`	*address* 変数の前に入れます。
address	アドレスを表すテキスト文字列。通常は、*hex_number*、*hex_number* の形式。(数値は 16 進形式で指定します。)
`:`	*arguments* 変数の前に入れなければなりません。
arguments	形式が特定のデバイスによって決まるテキスト文字列。これを使用してデバイスのソフトウェアに追加情報を渡すことができます。

フルデバイスパス名は、システムが使用するハードウェアアドレス指定をまねて、異なるデバイスを区別します。したがって、特定のデバイスをあいまいさなしに指定できます。

一般的に、ノード名の *address* 部分はその親のアドレス空間内の 1 つのアドレスを表します。特定のアドレスの正確な意味は、そのアドレスのデバイスが接続されているバスによって決まります。同じ例をもう一度示します。

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

- `1,f8000000` SBus インタフェースはメインシステムバスに直接接続されているので、1,f8000000 はメインシステムバス上のアドレスを表します。
- `esp`デバイスは SBus のスロット `0`、オフセット `40000`にあるので、0,40000 は SBus のスロット番号 (0) とそのスロット内のオフセット (40000) です。(明確にわかる名前ではありませんが、この例では、デバイスは SCSI のホストアダプタです。)
- `sd`デバイスは SCSI バスのターゲット `3`、論理ユニット `0`に接続されているので、3, 0 は SCSI のターゲットと論理ユニット番号です。

パス名を指定するときは、ノード名の *@address*または *name* 部分は省略できます。省略すると、ファームウェアは指定した名前に最もよく一致するデバイスを選択

しようとします。同じ程度に一致するデバイスが複数存在すると、ファームウェアはそれらから選択します (ユーザーが希望するものと異なるかも知れません)。

たとえば、/sbus/esp@0,40000/sd@3,0 を使用するとすると、そのシステムにはメインシステムバス上の sbus インタフェースが 1 つあるものとし、sbus を sbus@1,f8000000と表すのと同じように明確にします。しかし同じシステムでも、/sbus/esp/sd@3,0と表すと、あいまいな場合と、そうでない場合があります。SBus には差し込み式カードが装着できるので、同じ SBus 上に esp だけを使用したのではどのデバイスかを指定できず、ファームウェアはユーザーが意図するesp デバイスを選択しないかも知れません。

もう 1 つの例として、/sbus/@0,40000/sd@3,0 は通常指定できるのに対して、/sbus/esp@0,40000/@3,0 は通常では指定できません。それは、SCSI ディスクデバイスドライバ (sd) と SCSI テープデバイスドライバ (st) の両方に SCSI のターゲット、論理ユニットアドレス 3,0を使用できるためです。

ノード名の *:arguments* 部分も省略できます。同じ例を示します。

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

sd デバイスの引数は文字列 a です。sdのソフトウェアドライバはその引数をディスクパーティションとして解釈するため、デバイスパス名はそのディスク上のパーティション a を参照します。

デバイスの別名

デバイスの名前には次の 2 種類があります。

- デバイスのフルパス名 (前節で説明) : /sbus@1,f8000000/esp@0,40000/sd@3,0:a
- デバイスの別名 : disk

デバイスの別名、あるいは、単に別名 (alias) とは、デバイスのパス名を表す方法の 1 つです。別名はデバイスのパス名の構成要素ではなく、デバイスのパス名全体を表します。たとえば、disk という別名は次のようにデバイスのパス名を表します。

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

システムは、よく使用されるデバイスのほとんどに対して、デバイスの別名をあらかじめ定義しているので、デバイスのパス名を全部入力することはまれにしかありません。

表 1–3で、別名の確認、作成、変更を行う devalias コマンドを説明します。

表 1–3 デバイス別名の確認と作成

コマンド	説明
`devalias`	現在のすべてのデバイス別名を表示します。
`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。

ユーザーが定義する別名は、システムのリセット後、または電源の再投入後に失われます。永続的な別名を作成するには、`devalias` コマンドの出力を NVRAMRC と呼ばれる不揮発性 RAM (NVRAM) の一部に手作業で格納するか `nvalias` コマンド、`nvunalias` コマンドを使用します。(詳細は、第 3 章を参照してください。)

デバイスツリーの表示

デバイスツリーを表示し、ブラウズして、個々のデバイスツリーノードを調べ、変更することができます。デバイスツリーの表示用コマンドは、UNIX® ディレクトリツリーで作業ディレクトリを変更する UNIX コマンドと同じです。デバイスノードを選択すると、それが現在のノードになります。

デバイスツリーは表 1–4に示すコマンドを使用して調べます。

表 1–4 デバイスツリー表示コマンド

コマンド	説明
`.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。

表 1–4 デバイスツリー表示コマンド 続く

コマンド	説明
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。

デバイスツリーを表示していて、システムをリセットする場合は、次のように入力します。

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

次の例で .attributesの使用例を示します。

```
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
```

(続く)

```
ok
```

show-devs は次の例で示すように、OpenBoot デバイスツリー上のすべてのデバイスのリストを表示します。

```
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
```

次に wordsの使用例を示します。

```
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
```

ヘルプの表示

ディスプレイ上に ok が表示されているときは、表 1–5に示すヘルプコマンドを入力して、ヘルプを表示できます。

表 1–5 ヘルプコマンド

コマンド	説明
`help`	ヘルプの主なカテゴリを表示します。
`help` *category*	カテゴリ内の全コマンドのヘルプを表示します。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)。

help のみを入力すると、ヘルプシステムの使用方法についての説明と、提供されているヘルプのカテゴリが表示されます。コマンドの数は非常に多いため、ヘルプはよく使用されるコマンドだけに用意されています。

選択したカテゴリ内のすべてのコマンドのヘルプメッセージ、あるいは、サブカテゴリのリストを表示するには、次のように入力します。

```
ok help category
```

特定のコマンドのヘルプが必要な場合は、次のように入力します。

```
ok help  command
```

たとえば、dump コマンドのヘルプは次のように表示されます。

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

(続く)

```
ok
```

上記のヘルプメッセージは、まず、dump がメモリーアクセスカテゴリのコマンドであることを示し、さらに、dump コマンドの構文も示します。

注 - 一部の新しいシステムでは、help コマンドでマシン固有の追加コマンドの説明を表示できます。

一部の OpenBoot コマンドの使用上の注意

オペレーティングシステムを起動し、Stop-A または halt コマンドでオペレーティングシステムを終了してから、特定の OpenBoot コマンドを使用した場合、それらのコマンドは期待どおりに動作しないことがあります。

たとえば、オペレーティングシステムを起動し、 Stop-A で終了し、次に probe-scsi コマンドを実行したものとします。probe-scsi の実行が失敗し、オペレーティングシステムの実行が再開できないことがあります。このような場合、次のコマンドを入力します。

```
ok sync
ok boot
```

オペレーティングシステムが停止したために正常に実行できなかった OpenBoot コマンドを実行し直すには、次のようにシステムをリセットしてから、そのコマンドを起動します。

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

第 2 章

システムの起動とテスト

この章では、OpenBoot ファームウェアを使用して行う最も一般的な作業について説明します。次のような作業があります。

- システムの起動
- 診断の実行
- システム情報の表示
- システムのリセット

システムの起動

OpenBoot ファームウェアの最も重要な機能はシステムを起動することです。起動とは、オペレーティングシステムなどのスタンドアロンプログラムをロードし、実行するプロセスのことです。電源を入れるとシステムは、通常、ユーザーの操作なしに自動的に起動します。必要な場合、ユーザーは OpenBoot コマンドインタプリタから明示的に起動処理を開始できます。自動起動では、不揮発性 RAM (NVRAM) で指定されるデフォルトの起動デバイスが使用されます。ユーザーが開始する起動では、デフォルトの起動デバイスか、ユーザーが指定する起動デバイスが使用されます。

システムをデフォルト起動デバイスから起動するには、Forth モニターのプロンプトで次のコマンドを入力します。

```
ok boot
```

制限付きモニターのプロンプトでシステムを起動するには、次のように入力します。

```
> b
```

boot コマンドの構文は次のとおりです。

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

boot コマンドのオプションの変数を表 2–1で説明します。

表 2–1 bootコマンドの一般的オプション

変数名	説明
[*device-specifier*]	起動デバイス名 (フルパス名または別名)。例を示します。 cdrom (CD-ROM ドライブ) disk (ハードディスク) floppy (3.5 インチフロッピーディスクドライブ) 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 NVRAM 変数の値 (第 3 章参照) 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 - プログラムを読み込み後、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。)

注 - デバイス名の指定を必要とする (boot や testなどの) 多くのコマンドには、フルデバイスパス名、デバイスの別名のどちらを指定してもかまいません。このマニュアルでは、どちらの場合に対しても、*device-specifier* という表記を使用します。

明示的に内部ディスクから起動するには (ディスクフルシステムの場合)、次のように入力します。

```
ok boot disk
```

明示的に Ethernet から起動するには、次のように入力します。

```
ok boot net
```

制限付きモニタープロンプトで起動デバイスを指定するには、次の例に示すように、起動デバイスの名前を指定して bコマンドを使用します。

```
> b disk （ディスクフルシステム用の内部ディスクから明示的に起動する場合）
> b net （Ethernet から明示的に起動する場合）
```

デバイスの別名定義はシステムによって異なります。システムの別名の定義を知るには、第 1 章で説明したdevaliasコマンドを使用します。表 2–2は、デバイスの別名と、SPARCstation 2 および SPARCstation IPX システムでのデバイス別名の定義の例です。見出しの「旧パス」は、対応する SBus デバイスに対する OpenBoot バージョン 1.*x* での指定形式を示します。

表 2–2 代表的デバイスの別名

別名	起動パス	旧パス	説明
disk	/sbus/esp/sd@3,0	sd(0,0,0)	デフォルトディスク (第 1 内部ディスク)
disk0	/sbus/esp/sd@3,0	sd(0,0,0)	第 1 内部ディスクsd0
disk1	/sbus/esp/sd@1,0	sd(0,1,0)	第 2 内部ディスク sd1
disk2	/sbus/esp/sd@2,0	sd(0,2,0)	外部ディスク sd2
disk3	/sbus/esp/sd@0,0	sd(0,3,0)	外部ディスク sd3
tape	/sbus/esp/st@4,0	st(0,0,0)	第 1 テープドライブ st0.
tape0	/sbus/esp/st@4,0	st(0,0,0)	第 1 テープドライブ st0
tape1	/sbus/esp/st@5,0	st(0,1,0)	第 2 テープドライブ st1

表 2–2 代表的デバイスの別名 続く

別名	起動パス	旧パス	説明
cdrom	/sbus/esp/sd@6,0:c	sd(0,6,2)	CD-ROM パーティションc
cdroma	/sbus/esp/sd@6,0:a	sd(0,6,0)	CD-ROM パーティション a
net	/sbus/le	le(0,0,0)	Ethernet
floppy	/fd	fd(0,0,0)	フロッピードライブ

この表の sd0、sd1 などは、これらのデバイスを記述するために Solaris® 1.*x* オペレーティング環境で使用されていた用語です。The Solaris 2.*x* では、表 2–3に示すように異なっています。

表 2–3 Solaris オペレーティング環境の別名

別名	**Solaris 1.*x*** での名前	**Solaris 2.x** での名前
disk と disk0	sd0	c0t3d0s0
disk1	sd1	c0t1d0s0
disk2	sd2	c0t2d0s0
disk3	sd3	c0t0d0s0

診断の実行

いくつかの診断ルーチンが Forth モニターで使用できます。これらのオンボードテストでは、ネットワークコントローラ、フロッピーディスクシステム、メモリー、装着されている SBus カード、SCSI デバイス、システムクロックなどのデバイスの機能を確認できます。ユーザーがインストールするデバイスは、そのファームウェアに自己診断機能があればテストできます。

表 2–4に診断テスト用コマンドの一覧を示します。*device-specifier* は、デバイスパス名またはデバイスの別名を意味します。

表 2–4 診断テストコマンド

コマンド	説明
`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、指定したデバイスツリー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* を指定しないと、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 /memory` : NVRAM 変数 `selftest-#megs` で指定される M バイト数をテストします。 または `diag-switch?` が `true` の場合は全メモリをテストします。 `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 (組み込みセルフテスト方法を備える) 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。)
`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。

SCSI バスのテスト

組み込み SCSI バスに接続されているデバイスの機能を確認するには、次のように入力します。

```
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
```

システムに接続されているすべての SCSI バスをテストするには、次のように入力します。

```
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
```

応答は SCSI バスに接続されているデバイスによって異なります。

インストールされているデバイスのテスト

インストールされている 1 つのデバイスをテストするには、次のように入力します。

```
ok test device-specifier
```

これは、指定したデバイスノードの (selftest という名前の) デバイステスト方法を実行します。応答はデバイスノードの自己診断テストの方法によって異なります。

インストールされているデバイスのグループをテストするには、次のように入力します。

```
ok test-all
```

デバイスツリーのルートノードの下のすべてのデバイスがテストされます。応答は、各デバイスの自己診断テストの方法によって異なります。`test-all` コマンドに *device-specifier* オプションを使用すると、指定したデバイスツリーノードの下のすべてのデバイスがテストされます。

フロッピーディスクドライブのテスト

フロッピーディスクドライブのテストは、フロッピーディスクドライブが正しく機能するかどうかを調べます。このテストを実行するには、フロッピーディスクドライブにフォーマット済みの高密度 (HD) ディスクをセットしておかなければなりません。

フロッピーディスクドライブをテストするには、次のように入力します。

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

このテストが失敗した場合は、エラーメッセージを参照してください。

フロッピーディスクを取り出すには、次のように入力します。

```
ok eject-floppy
ok
```

このコマンドが失敗する場合は、紙クリップを伸ばしてディスクスロットの近くの小さな穴に差し込んで、フロッピーディスクを取り出せます。

メモリーのテスト

メモリーテストルーチンを使用すると、システムは NVRAM のシステム変数 `selftest-#megs` で指定した M バイト数のメモリーをテストします。(NVRAM のシステム変数についての詳細は、第 3 章を参照してください。) デフォルトでは 1 M バイトのメモリーがテストされます。ハードウェア診断スイッチ (システムにある場合) 、NVRAM のシステム変数 `diag-switch?` のどちらかが有効にしてある場合は、全メモリーがテストされます。

メモリーをテストするには、次のように入力します。

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

上記の例で、最初の数値 (4000000) はテストの基底アドレスであり、その次の数値 (11) はテストされる M バイト数です。

PROM がシステムをテストするにはしばらく時間がかかります。システムがこのテストに失敗した場合は、エラーメッセージが表示されます。

Ethernet コントローラのテスト

ボード上の Ethernet コントローラをテストするには、次のように入力します。

```
ok test net
Internal Loopback test - (result)
External Loopback test - (result)
ok
```

システムはテストの結果を示すメッセージを表示して応答します。

注 - システムが Ethernet に接続されていない場合は、このテストの外部ループバック部分が失敗します。

クロックのテスト

クロック機能をテストするには、次のように入力します。

```
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
```

数値が 1 秒毎に 1 つずつ増えていきます。テストを停止するには任意のキーを押します。

ネットワークの監視

ネットワーク接続を監視するには、次のように入力します。

```
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
```

システムはネットワークトラフィックを監視し、エラーのないパケットを受け取るたびに " . " を、また、ネットワークハードウェアインタフェースによって検出できるエラーがあるパケットを受け取るたびに x をそれぞれ表示します。

注 - 一部の OpenBoot 2.*x* システムにはこのテストワードがありません。

システム情報の表示

Forth モニターはシステム情報を表示するコマンドをいくつか備えています。それらのコマンドを表 2–5に示します。これらのコマンドでは、システムバナー、Ethernet コントローラの Ethernet アドレス、ID PROM の内容、OpenBoot ファームウェアのバージョン番号を表示できます。(ID PROM 内容は、シリアル番号、製造年月日、マシンに割り当てられている Ethernet アドレスを含む各マシン固有の情報です。)

表 2–5 システム情報表示コマンド

コマンド	説明
`banner`	電源投入時のバナーを表示します。
`show-sbus`	インストールされ、プローブされる SBus デバイスのリストを表示します。
`.enet-addr`	現在の Ethernet アドレスを表示します。
`.idprom`	ID PROM の内容を書式付きで表示します。
`.traps`	SPARC のトラップタイプのリストを表示します。
`.version`	起動 PROM のバージョンと日付を表示します。

デバイスツリー表示コマンドも参照してください。

注 - オペレーティングシステムを停止し `banner` と入力し、次にシステムを再起動すれば、カラーテーブルが変更されてしまうことがあります。Solaris 2.0 以前の OS でこのカラーテーブルを復元するためには、`clear_colormap` を実行し、次に `Utilities` メニューから `Refresh` を選択します。Solaris 2.0 または 2.1 のオペレーティング環境に復元するには、`Properties...` メニューから `ColorChooser` を選択します。

システムのリセット

場合により、システムをリセットすることが必要になることがあります。reset コマンドはシステム全体をリセットします。これは、電源再投入 (パワーサイクル) を行うのと同じです。

システムをリセットするには、次のように入力します。

```
ok reset
```

リセット時にパワーオン自己診断テスト (POST) および初期化手続きを実行するようにシステムを設定してある場合は、このコマンドを起動すると、それらの手続きの実行が開始されます。(システムによっては、電源投入後に POST だけが実行されます。) POSTが終了すると、電源再投入後と同様に、システムは自動的に起動するか、Forth モニターに入ります。

注 - デバイスツリーをブラウズしていた場合は、システムをリセットする前に device-end コマンドの実行が必要なことがあります。

システム変数の設定

この章では、不揮発性 RAM (NVRAM) のシステム変数にアクセスし、変更する方法について説明します。

システム変数はシステム NVRAM に格納されます。それらの変数は、起動時のマシン構成と関連する通信特性を設定します。システム変数のデフォルト値は変更することができ、行った変更は電源再投入後も有効です。システム変数はつねに注意深く調整する必要があります。これらの変数を正しく使用すれば、システムのハードウェアを柔軟に取り扱えます。

この章で説明する手順は、画面には ok プロンプトが表示されるものとしています。Forth モニターに入る方法については 第 1 章を参照してください。

表 3–1 に現在の NVRAM のシステム変数の一覧を示します。

表 3–1 NVRAM システム変数

変数名	設定値	説明
`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`	空白文字	起動するファイル (空白文字の場合、第 2 ブーター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)。

表 3–1 NVRAM システム変数 続く

変数名	設定値	説明
`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	空白文字
`diag-switch?`	false	true の場合、診断プログラムを実行します。
`fcode-debug?`	false	true の場合、差し込み式デバイス FCode の名前フィールドを取り入れます。
`hardware-revision`	デフォルトなし	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`	デフォルトなし	カスタムキーボードのキーマップ。
`last-hardware-update`	デフォルトなし	システム更新情報。
`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テストを繰り返します。
`nvramrc`	空白文字	NVRAMRC の内容。
`oem-banner`	空白文字	カスタム OEM バナー (`oem-banner?` が `true`で使用可能になります)。

表 3–1 NVRAM システム変数 続く

変数名	設定値	説明
oem-banner?	false	true の場合、カスタム OEM バナーを使用します。
oem-logo	デフォルトなし	バイト配列カスタム OEM ロゴ (oem-logo? が true で使用可能になります)。 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 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 screen、ttya、ttyb)。
sbus-probe-list	0123	プローブされる SBus スロット、それらのスロットがプローブされる順番。
screen-#columns	80	画面上のカラム数 (文字数 / 行)。
screen-#rows	34	画面上の行数。
scsi-initiator-id	7	ホストアダプタの SCSI バスアドレス。範囲は 0 - 7。
sd-targets	31204567	
security-#badlogins	デフォルトなし	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	デフォルトなし	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)。これを直接設定してはなりません。
selftest-#megs	1	テストするメモリーの M バイト数。 diag-switch? が true の場合は無視されます。

表 3–1 NVRAM システム変数 続く

変数名	設定値	説明
skip-vme-loopback?	false	rue の場合、POST は VMEbus のループバックテストを行いません。
st-targets	45670123	SCSI テープユニットを割り当てます(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	1 バイトのスクラッチフィールド。読み取り／書き込みテストに使用されます。
tpe-link-test?	true	組み込みより対線 Ethernet 向け 10baseT リンクテストを有効にします。
ttya-mode	9600,8,n,1,-	ttya (ボーレート、データビット数、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、データビット数、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、オペレーティングシステムは ttya のキャリヤ検出を無視します。
ttyb-ignore-cd	true	true の場合、オペレーティングシステムは ttyb のキャリヤ検出を無視します。
ttya-rts-dtr-off	false	true の場合、オペレーティングシステムは ttya の DTR、RTS をアクティブにしません。
ttyb-rts-dtr-off	false	true の場合、オペレーティングシステムは ttyb の DTR、RTS をアクティブにしません。

表 3–1　NVRAM システム変数　続く

変数名	設定値	説明
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。
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*で起動します。
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。

注 - OpenBoot システムによっては一部の変数をサポートしていません。デフォルトは、システムのタイプや PROM のバージョンによって変わることがあります。

変数設定の表示と変更

NVRAM のシステム変数は、表 3–2に示すコマンドを使用して表示、変更できます。

表 3–2　システム変数の表示と変更

コマンド	説明
printenv	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 進で示されます。) printenv *parameter* は指定する変数の現在値を表示します。
setenv *parameter value*	*parameter* (変数)を 10 進またはテキスト値 *value* に設定します。 (変更は永続的ですが、通常はリセット後に初めて有効になります。)

表 3–2 システム変数の表示と変更 続く

コマンド	説明
`set-default parameter`	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。

以降のページでこれらのコマンドをどのように使用できるかを示します。

システムの現在の変数設定のリストを表示するには、次のように入力します。

```
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
```

(続く)

```
testarea              0                             0
mfg-switch?           false                         false
diag-switch?          true                          false
ok
```

現在の設定の書式付きリストでは、数値変数は特に注記がない限り 10 進数で示されます。

変数設定を変更するには、次のように入力します。

```
setenv parameter value
```

parameter は変数の名前であり、*value* は変数に該当する数値またはテキスト文字列です。数値のデータ型は 0x を前に付けなければ 10 進になります。0xは 16 進数の修飾子です。(大部分の変数変更は、次の電源再投入またはシステムリセットによって有効になります。)

たとえば、auto-boot? 変数の設定を true から false に変更するには、次のように入力します。

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

set-default 変数と set-defaults コマンドを使用して、NVRAM のシステム変数の特定の 1 つまたは大部分をもとのデフォルト設定に戻すことができます。

たとえば、auto-boot? 変数をそのもとのデフォルト設定 (true) に戻すには、次のようの入力します。

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

大部分の変数をそれぞれのもとのデフォルト設定に戻すには、次のように入力します。

```
ok set-defaults
ok
```

セキュリティー変数の設定

NVRAM のシステムセキュリティー用として次に示す変数があります。

- security-mode
- security-password
- security-#badlogins

security-mode は、許可のないユーザーが Forth モニターから実行できる一連の処理を制限できます。3 つのセキュリティーモードを、セキュリティーの低い順序で示すと次のとおりです。

- none
- command
- full

command モードおよび fullモードを使用するには、制限付きモニターを使用します。セキュリティーを command モードまたは fullモードに設定すると、OpenBootファームウェアは制限付きモニターとして起動します。none セキュリティーモードでは、どちらがデフォルトかによって、OpenBoot ファームウェアは Forth モニターか、制限付きモニターとして起動します。

none セキュリティーモードでは、どのコマンドも制限付きモードで入力でき、パスワードは必要ありません。command および full セキュリティーモードでは、特定のコマンドを実行するのにパスワードが必要です。たとえば、Forth モードにする

にはパスワードが必要です。Forth モニターに入った後は、パスワードは必要なくなります。

security-mode はオペレーティングシステムの eeprom ユーティリティーで変更できます。

コマンドセキュリティー

security-mode に設定しているときは、システムは制限付きモニターとして起動します。このモニターモードでは、パスワードの必要性は次のとおりです。

- b コマンドを入力する場合、変数を指定しなければ、パスワードは必要ありません。
- c コマンドはパスワードを要求しません。
- n コマンドを実行するにはパスワードが必要です。

次に画面で例を示します。

```
> b
(パスワード必要なし)
> c       (パスワード必要なし)
> b filename      (パスワード必要)
PROM Password:        (入力時パスワードは画面表示されない)
> n       (パスワード必要)
PROM Password:        (入力時パスワードは画面表示されない)
```

セキュリティーパスワードと command セキュリティーモードを設定するには、ok プロンプトで次のように入力します。

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

注 - この例でも機能しますが、通常はオペレーティングシステムからの eeprom コマンドで 2 つの security 変数を設定します。

設定するセキュリティーパスワードは root パスワードと同じ規則に従います。つまり、6 ～ 8 の英字および数字の組み合わせです。セキュリティーパスワードは root パスワードと同じでも、異なっても差し支えありません。システムをリセットする必要はありません。セキュリティー機能はコマンドを入力した直後に有効になります。

注意 - セキュリティーパスワードを絶対に忘れないようにしてください。このパスワードを忘れると、システムが使用できなくなります。販売代理店に連絡してマシンを再び起動可能にする必要があります。

誤ったセキュリティーパスワードを入力した場合は、約 10 秒の遅延があってから次の起動プロンプトが現れます。誤ったセキュリティーパスワードを入力した回数は `security-#badlogins` 変数に格納されます。この変数は 32 ビットの符号付きの数値です。

フルセキュリティー

full セキュリティーモードは最も制限の多いモードです。`security-mode`を full に設定した場合は、システムは制限付きモニターとして起動します。このモードでは、パスワードの必要性は次のとおりです。

- `b` コマンドの入力時にはパスワードが必要です。
- `c` コマンドはパスワードを要求しません。
- `n` コマンドを実行するにはパスワードが必要です。

次に例を示します。

```
> c       (パスワード必要なし)
> b       (パスワード必要)
PROM Password:        (入力時パスワードは画面表示されない)
> b filename      (パスワード必要)
PROM Password:        (入力時パスワードは画面表示されない)
> n       (パスワード必要)
PROM Password:        (入力時パスワードは画面表示されない)
```

セキュリティーパスワードと full セキュリティーを設定するには、`ok`プロンプトで次のように入力します。

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

電源投入時バナーの変更

バナー構成用として次の変数があります。

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

電源投入時バナーを表示するには、次のように入力します。

```
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
```

PROM がシステムバナーを表示します。これは SPARCstation 2 のバナーの例です。SPARC システムによりバナーはこれとは異なることがあります。

バナーは、テキストフィールドとロゴの 2 つの部分からなっています (シリアルポートを介す場合は、テキストフィールドしか表示されません)。oem-banner と oem-banner? システム変数を使用して、既存のテキストフィールドを、カスタマイズしたテキストメッセージに置き換えることができます。

電源投入時バナーにカスタマイズしたテキストフィールドを挿入するには、次のように入力します。

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

             Hello Mom and Dad

ok
```

システムは、この画面に示すように、新しいメッセージ付きのバナーを表示します。

図形ロゴは多少異なる方法で取り扱わなければなりません。oem-logoは、64 x 64 に配列された合計 4096 ビットからなる 512 バイトの配列です。各ビットはそれぞれ 1 ピクセルに相当します。最初のバイトの最上位ビット (MSB) が左上コーナのピクセルを制御します。次のビットはその右のピクセルを制御し、以下同様に各ビットは順次にピクセルに対応します。

新しいロゴを作成するには、まず、正しいデータを収容した Forth 配列を作成し、次にこの配列を oem-logo にコピーします。次の例では、Forth のコマンドを使用して配列を作成しています。(ロゴは、オペレーティングシステムのもとでも eeprom コマンドを使用して作成できます。) 作成した配列を次に to コマンドを使用してコピーしています。次の例では、oem-logo の上側の半分に昇順パターンを書き込んでいます。

```
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
```

初期設定のサンの電源投入時バナーを復元するには、oem-logo? および oem-banner?変数を falseに設定します。

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

oem-logo 配列は非常に大きいので、printenv は最初のほぼ 8 バイト (16 進) しか表示しません。配列全体を表示するには、oem-logo dump コマンドを使用します。oem-logo 配列は、データの復元が難しいことがあるので、set-defaults によって消去されません。しかし、set-defaults を実行すると、oem-logo? が false に設定され、したがってカスタマイズしたロゴはそれ以降表示されなくなります。

入出力の制御

システム入出力の制御関係の構成用として次に示す変数があります。

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

これらの変数を使用して入出力用の電源投入時デフォルトを割り当て、ttya と ttyb のシリアルポートの通信特性を調整できます。ttya-mode と ttyb-mode の結果を除いて、これらの値は次の電源再投入またはシステムリセットまで有効になりません。

入出力デバイスオプションの選択

input-device および output-device 変数は、電源投入リセット後の入出力デバイスの選択を制御します。input-deviceのデフォルト値は keyboard であり、output-device のデフォルト値は screen です。入出力は 表 3–3 の値に設定できます。

表 3–3 入出力デバイス変数

オプション	説明
device-specifier	そのデバイスのパス名または別名によって識別されるデバイス。
keyboard	(入力専用) デフォルトシステムキーボード。
screen	(出力専用) デフォルトグラフィックスディスプレイ。
ttya	シリアルポート A
ttyb	シリアルポート B

システムをリセットすると、指定したデバイスがデフォルトの入力または出力デバイスになります。(入力または出力デバイスを一時的に変更する場合は、第 4 章で説明する input または output コマンドを使用します。)

ttyaを電源投入時デフォルト入力デバイスとして設定するには、次のように入力します。

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

input-device として keyboard を選択したが、このデバイスが接続されていない場合は、次の電源再投入またはシステムリセット後は、入力は ttya から受け入れられます。output-device として screen を選択したが、フレームバッファーが存在しない場合は、次の電源再投入またはシステムリセット後は、出力はttya に送られます。

デフォルト出力デバイスとして bwtwo フレームバッファーを指定するには (特にシステムに複数のフレームバッファーが存在する場合)、次のように入力します。

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

シリアルポート特性の設定

大部分のサンシステムで、ttya と ttyb の両方に対して次にデフォルト設定が使用されます。

9600 ボー、8 データビット、パリティーなし、1 ストップビット、ハンドシェークなし

2 つのシリアルポート ttya および ttyb の通信特性は、`ttya-mode` および `ttyb-mode` の各変数に次に値を設定します。

- `baud` = 110、300、1200、2400、4800、9600、19200、または 38400 ビット /秒
- `#bits` = 5、6、7、または 8 (データビット)
- `parity` = n(なし)、e (偶数)、または o (奇数)、パリティービット
- `#stop` = 1 (1)、. (1.5)、または 2 (2) ストップビット
- `handshake` = - (なし)、h (ハードウェア(rts/cts))、または s (ソフトウェア (xon/xoff)).

たとえば、ttya を 1200 ボー、7 データビット、偶数パリティー、1 ストップビット、ハンドシェークなしに設定するには、次のように入力します。

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

これらの変数値の変更はただちに有効になります。

注 - システムによっては、rts/cts および xon/xoff ハンドシェークは実装されていません。選択したプロトコルが実装されていないときは、*handshake* 変数は受け入れられますが、無視されます。メッセージは何も表示されません。

起動オプションの選択

次にシステム変数を使用して、電源再投入またはシステムリセット後にシステムを自動的に起動させるかどうかを設定できます。

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

auto-boot? が true の場合は、システムは (boot-device と boot-file の値を使用して) 自動的に起動します。

手動起動時にも、これらの変数を使用して起動デバイスと起動するプログラムを選択することができます。たとえば、Ethernet からの自動起動を指定するには、次のように入力します。

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

指定した起動は通常、すぐに開始されます。

注 - boot-device と boot-file の指定方法は、diag-switch? を true に設定しているときには異なります。詳細は、次の節を参照してください。

電源投入時の自己診断テストの制御

電源投入時テスト用として次に示す変数があります。

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

大部分のシステムでは、diag-switch? 変数の工場出荷時のデフォルトは falseです。diag-switch? を true に設定するには、次のように入力します。

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

diag-switch? を有効にすると、システムはそれ以降の電源投入時に常により厳密な自己診断テストを実行します。diag-switch? を有効にすると、追加ステータスメッセージが (一部は ttya に、一部は指定した出力デバイスに) 送出され、全メモリーがテストされ、さまざまなデフォルト起動オプションが使用されます。起動 PROM は diag-file 変数によって指定されたプログラムを diag-device によって指定されるデバイスから起動しようとします。

注 - 一部の SPARC システムはハードウェアの診断スイッチを備えています。そのハードウェアスイッチまたは diag-switch? が設定されている場合は、システムは電源投入時にフルテストを実行します。

電源投入時に Stop-D キー処理を使用しても、diag-switch? を true に強制設定できます。

diag-switch? を false に設定するには、次のように入力します。

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

diag-switch? が false のときは、診断テストは実行されるので、(テストでエラーが発生しなければ) システムは診断テストを呼び出さず、診断の一部を実行します。

NVRAMRC の使用方法

NVRAM の一部に nvramrc と呼ばれる部分があります。そのサイズはそれぞれの SPARC システムによって異なります。これは、起動時に実行されるユーザー定義のコマンドの格納用として予約されています。

一般的に、NVRAMRC は起動時システム変数を保存したり、デバイスドライバコードをパッチしたり、インストール先固有のデバイス構成とデバイスの別名を定義するためにデバイスドライバが使用します。また、バグパッチまたはユーザーインス

トールの拡張用にも使用できます。コマンドは、ユーザーがコンソールから入力するとおりの ASCII で格納されます。

NVRAMRC 関係のシステム変数には次の 2 つがあります。

- nvramrc
- use-nvramrc?

NVRAMRC 内のコマンドは、use-nvramrc? を true に設定している場合に、システム起動時に実行されます。次の例外を除く、ほとんどすべての Forth モニターコマンドがここで使用できます。

- banner (使用に際しては注意が必要)
- boot
- go
- nvedit
- password
- reset
- setenv security-mode

NVRAMRC 内容の編集

NVRAMRC のエディタである nvedit では、表 3–4に示すコマンドを使用して、NVRAMRC の内容を作成、変更することができます。

表 3–4 NVRAMRC エディタコマンド

コマンド	説明
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。
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。捨てる前に、確認を求めます。

表 3–4 NVRAMRC エディタコマンド 続く

コマンド	説明
`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。
`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。

注 - 一部の OpenBoot 2.*x* には `nvalias` および `nvunalias` コマンドがありません。

表 3–5 に NVRAM で使用できる編集コマンドを示します。

表 3–5 `nvedit`キー操作コマンド

キー操作	説明
`Control-B`	1 文字位置戻ります。
`Control-C`	エディタを終了し、OpenBook コマンドインタプリタに戻ります。一時バッファーは保存されていますが、 NVRAMRC には戻されません。 (後で `nvstore` を使用して一時バッファーを NVRAMRC に書いて戻してください。)
`Control-F`	1 文字位置戻ります。
`Control-K`	行の終わりでは、現在行に次の行をつなぎます (つまり、2 つの行を 1 つにします)。
`Control-L`	すべての行を表示します。
`Control-N`	NVRAMRC 編集バッファの次の行に進みます。
`Control-O`	カーソル位置に new line を挿入し、現在行にとどまっています。

表 3–5　nveditキー操作コマンド　続く

キー操作	説明
Control-P	NVRAMRC 編集バッファーの前の行に戻ります。
Delete	前の 1 文字を削除します。
Return	カーソル位置に改行を挿入し、次の行に進みます。

そのほかの標準的エディタ用コマンドについては、第 4 章を参照してください。

NVRAMRC ファイルの起動

次の手順で、NVRAMRC コマンドファイルを起動してください。

1. ok プロンプトで、**nvedit**と入力します。
   エディタのコマンドを使用して NVRAMRC の内容を編集します。

2. **Control-C** を入力してエディタを終了し、再び ok プロンプトを表示させます。

3. **nvstore** と入力して変更結果を保存します。

4. **setenv use-nvramrc? true** と入力して、**NVRAMRC** を有効にします。

5. **reset** と入力してシステムをリセットしてから **NVRAM** の内容を実行するか、**nvramrc eval** と入力して **nvramrc** の内容を直接実行します。まだ **nvstore** と入力して変更結果を保存してなかった場合は、**nvrun** と入力して一時編集バッファーの内容を実行してください。

次の例で、NVRAMRC内に単純なコロン定義を作成する方法を示します。

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

(続く)

```
ok reset
....
ok hello
Hello, world.
ok
```

上記の例で、nveditの行番号のプロンプト (0:、1:、2:、3:) に注意してください。これらのプロンプトはシステムによって異なることがあります。

第 4 章

Forth ツールの使用方法

この章では、OpenBoot に実装されている Forth の概要を説明します。Forth プログラミング言語に詳しい読者も、この章の例を確認してください。これらの例には、OpenBoot に関連する特有の情報が含まれています。

OpenBoot に含まれる Forth のバージョンは、ANS Forth に準拠しています。付録 Eに全コマンドのリストが載せてあります。SBus デバイス用 OpenBoot FCode プログラムを書くための専用のワードについては、『*Writing FCode 2.x Programs*』マニュアルに説明されています。

注 - この章では、読者はユーザーインタフェースの起動、終了手順を知っているものとしています。ok プロンプトで入力したコマンドのためにシステムがハングアップし、キー入力操作で回復できない場合は、正常動作に復帰させるために電源再投入を行う必要があるかも知れません。

Forth コマンド

Forth のコマンド構造は非常に単純です。Forth のコマンドは、Forth ワードとも呼ばれますが、印刷可能な文字、たとえば、英字、数字、句読記号 - の任意の組み合わせです。正しいワードの例を次に示します。

```
@
dump
.
```

(続く)

```
0<
+
probe-scsi
```

probe-scsi コマンドとして認識されるためには、Forth ワードはそれぞれの間を 1 つまたはそれ以上の空白文字 (ブランク) で分離する必要があります。どのコマンド行の終わりで Return キーを押しても、そこまで入力したコマンドが実行されます。(この章に示すすべての例で、行の終わりでは Return が押されるものとしています。)

1 コマンド行に複数のワードを入力できます。1 行上の複数のワードは、左から右に向かって、つまり入力順に 1 つ 1 つ実行されます。たとえば、次の例

```
ok testa testb testc
ok
```

は次の 3 行と同じです。

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

OpenBoot では、大文字と小文字の区別はありません。したがって、testa、TESTA、TesTaはすべて同じコマンドを起動します。しかし、習慣によりコマンドは小文字で書きます。

コマンドによっては (たとえば、dump または words) 、大量の出力を生成するものがあります。そのようなコマンドは、q以外の任意のキーを押して中断できます。(q を押した場合は、出力は一時停止でなく強制終了されてしまいます。) コマンドを中断すると、出力は一時的に停止され、次のメッセージが表示されます。

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

これに対して、スペースバー (<space>) を押して出力を再開するか、Return (<cr>) キーを押して 1 行出力し、再び休止するか、または q を入力してコマンドを強制終了します。出力を複数ページ生成する場合は、システムは自動的に各ページの終わりに上に示したプロンプトを表示します。

数値の用法

数値は、たとえば 55 とか -123 など、その値をキーボードで入力します。Forth は整数しか受け入れません。分数値 (たとえば、2/3) は入力できません。数値の終わりにピリオドを入力すると、それが倍精度であることを意味します。数値のなかにピリオド、コンマを埋め込んでも無視されます。したがって、5.77 は 577 と解釈されます。表記の規約では、区切り記号は通常 4 桁おきに使用します。数値は、1 つまたはそれ以上の空白文字を使用してワードや別の数値と区切ってください。

OpenBoot は 32 ビット整数の数値演算を行います。別に指定がないかぎり、数値はすべて 32 ビットです。

OpenBoot では 16 進の (変換) 基数を使用するよう奨励されていますが、必ずそうしなければならないわけではありません。したがって、正しく演算されるためにはコードが特定の基数に依存する場合は、そのような基数を設定する必要があります。

演算する数値の基数は octal、decimal 、hex といったコマンドを使用して変更できます。これらのコマンドは、以降の数値の入出力をそれぞれ 8、10、16 を基数として行わせます。

たとえば、10 進で演算するには、次のように入力します。

```
ok decimal
ok
```

16 進に変更するには、次のように入力します。

```
ok hex
ok
```

現在使用されている基数を知る 2 つの単純な方法を次に示します。

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

上記の画面に表示されている 16 と f は、16 進で演算が行われることを示しています。10 と 9 が表示される場合は、10 を基数としていることを意味します。8 と 7 であれば、8 進を示します。

スタック

Forth のスタックは、数値情報の一時的保持用の後入れ先出し型 (LIFO) バッファーです。これを積み重ねられた本と考えてみてください。その場合、最後に置いた、つまり本の積み重ねの一番上に乗せた本から先に取ることになります。Forth を使用するには、スタックを理解することが不可欠です。

スタックに数値を入れる (一番上に乗せる) には、単にその値を入力します。

```
ok 44  (値 44 がスタックの一番上に乗る)
ok 7     (値 7 がスタックの一番上に乗る)
ok
```

スタックの内容の表示

スタックの内容は通常は表示されません。しかし、希望する結果を得るためには、現在のスタックの内容を確認する必要があります。ok プロンプトが現れるごとにスタックの内容を表示することができますが、それには、次のように入力します。

```
ok showstack
44 7 ok 8
47 7 8 ok showstack
ok
```

一番上のスタック項目は、ok プロンプトのすぐ前に、リストの最後の項目としてつねに表示されます。上記の例では、一番上のスタック項目は 8 です。

前に showstack を実行している場合は、もう一度 noshowstack と入力すると、各プロンプトの前のスタック表示が削除されます。

注 - この章のいくつかの例では showstack を有効にしています。それらの例では、各 ok プロンプトのすぐ前にそのときのスタックの内容が表示されています。それらの例は、スタックの内容が表示される点を除けば、showstack を有効にしてない場合と同じです。

数値変数を必要とするほとんどすべてのワードは、それらの変数をスタックの一番上から取り出します。また、返されるどの値も、通常、スタックの一番上に残され、別のコマンドで表示したり、「消費」する (つまり演算などに使ってスタックから削除する) ことができます。たとえば、+ という Forth ワードは、スタックから 2 つの数値を削除し、それらを加算し、結果をスタックに残します。次の例では、演算はすべて 16 進で行われます。

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

2 つの値が加算されると、結果がスタックの一番上に乗せられます。Forth ワードの . は一番上のスタック項目を削除し、その値を画面に表示します。次の例を参照してください。

```
53 ok 12
53 12 ok .
12
53 ok .
53
ok   (ここではスタックは空)
ok 3 5 + .
8
```

```
ok   (ここではスタックは空)
ok .
Stack Underflow
ok
```

スタックダイアグラム

規則に従うコーディング形式では、わかりやすいように Forth ワードの定義ごとに、それぞれの最初の定義行に (--) の形式のスタックダイアグラムを表記する必要があります。スタックダイアブラムは、ワードを実行するとスタックがどうなるかを指定するものです。

-- の左側に置かれる項目は、 スタックから消費 (つまり削除) され、そのワードの演算に使用されるスタック項目を示します。-- の右側に置かれる項目は、ワードの実行の終了後にスタックに残されるスタック項目を示します。たとえば、ワード + のスタックダイアグラムは (nu1 nu2 -- sum) であり、ワード . のスタックダイアグラムは (nu --) です。したがって、+ は 2 つの数値 (nu1 と nu2)を削除し、次にそれらの和 (sum) をスタックに残します。ワード . はスタックの一番上の数値 (nu) を削除し、それを表示します。

スタックの内容に影響しないワード (showstack や decimal など)のスタックダイアグラムは (--) になります。

場合によっては、ワードのすぐ後に別のワード、またはほかのテキストが必要なことがあります。たとえば、seeは、 see *thisword* (–)という形式で使用されます。

スタック項目は、正しい使い方がわかりやすいように、一般的に (意味を表すような) 説明的名前を使用して書きます。このマニュアルで使用するスタック項目の省略表記については、表 4–1を参照してください。

表 4–1 スタック項目の表記法

表記	説明
|	前後に空白文字を入れて表示される代替スタック結果。たとえば、(input – addr len false | result true)。
|	前後に空白文字なしで表示される代替スタック項目。たとえば、(input – addr len | 0 result)。
`???`	未知のスタック項目 (1 つまたは複数)。
`...`	未知のスタック項目 (1 つまたは複数)。スタックコメントの両側に使用した場合は、両側に同じスタック項目があることを意味します。
`< > <space>`	空白区切り文字。先行空白文字は無視されます。
`a-addr`	可変境界アドレス。
`addr`	メモリーアドレス (一般的に仮想アドレス)。
`addr len`	メモリー領域のアドレスと長さ。
`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)。上位ビットは不定。
`cnt len size`	カウント値または長さ。
d*xxx*	倍 (拡張) 精度数。2 スタック項目 - スタックの一番上の上位 quadlet (32 ビット)。
`<eol>`	行末区切り文字。
`false`	0 (false フラグ)。
`ihandle`	パッケージのインスタンスのポインタ。
`n n1 n2 n3`	通常の符号付き値 (32 ビット)
`nu nu1`	符号付きまたは符号なしの値 (32 ビット)

表 4–1 スタック項目の表記法 続く

表記	説明
`<nothing>`	ゼロスタック項目。
`phandle`	パッケージのポインタ。
`phys`	物理アドレス (実際のハードウェアアドレス)。
`phys.lo phys.hi`	物理アドレスの下位/上位セル。
`pstr`	パックされた文字列。
`quad q`*xxx*	quadlet (32 ビット値)。
`qaddr`	quadlet (32 ビット値) 境界のアドレス。
`{text}`	省略可能なテキスト。省略した場合は、デフォルト操作が行われます。
`"text<`*delim*`>"`	入力バッファーテキスト。コマンドの実行時に構文解析されます。テキスト区切り文字を <> で囲みます。
`[text<`*delim*`>]`	同じ行上のコマンドのすぐ後のテキスト。ただちに構文解析されます。テキスト区切り文字は <> で囲みます。
`true`	-1 (true フラグ)。(真)
u*xxx*	符号なしの値、正の値 (32 ビット)。
`virt`	仮想アドレス (ソフトウェアが使用するアドレス)。
`waddr`	doublet (16 ビット) 境界のアドレス。
`word w`*xxx*	doublet (16 ビット値、 - 32 ビットワードの下位 2 バイト)。
`x x1`	任意のスタック項目。
`x.lo x.hi`	データ項目の最下位/最上位ビット

表 4–1 スタック項目の表記法 続く

表記	説明
`xt`	実行トークン。
`xxx?`	フラグ。名前は用途を示します (たとえば、 `done? ok? error?`)。
`xyz-str xyz-len`	パックされない文字列のアドレスと長さ。
`xyz-sys`	制御フロー用スタック項目。実装に依存します。
`( C: -- )`	コンパイルスタックダイアグラム。
`( -- )( E: -- )`	実行スタックダイアグラム。
`( R: -- )`	復帰スタックダイアグラム。

スタックの操作

スタック操作用のコマンド (表 4–2で説明) では、スタック上の項目の追加、削除、並べ替えができます。

表 4–2 スタック操作コマンド

コマンド	スタックダイアグラム	説明
`-rot`	(x1 x2 x3 – x3 x1 x2)	3 つのスタック項目を逆方向に回転します。
`>r`	(x –)(R: – x)	スタック項目を復帰スタックに転送します。 (使用には注意が必要です。)
`?dup`	(x – x x 0)	一番上のスタック項目がゼロ以外の場合、複製します
`2drop`	(x1 x2 –)	スタックから 2 つの項目を削除します。

表 4–2 スタック操作コマンド 続く

コマンド	スタックダイアグラム	説明
`2dup`	(x1 x2 – x1 x2 x1 x2)	2 つのスタック項目を複製します。
`2over`	(x1 x2 x3 x4 – x1 x2 x3 x4 x1 x2)	初めから 2 つのスタック項目をコピーします。
`2rot`	(x1 x2 x3 x4 x5 x6 – x3 x4 x5 x6 x1 x2)	3 対のスタック項目を回転します。
`2swap`	(x1 x2 x3 x4 – x3 x4 x1 x2)	2 対のスタック項目を入れ換えます。
`3drop`	(x1 x2 x3 –)	スタックから 3 つの項目を削除します。
`3dup`	(x1 x2 x3 – x1 x2 x3 x1 x2 x3)	3 つのスタック項目を複製します。
`clear`	(??? –)	スタックを空にします。
`depth`	(– u)	スタック上の項目数を返します。
`drop`	(x –)	一番上のスタック項目を削除します。
`dup`	(x – x x)	一番上のスタック項目を複製します。
`nip`	(x1 x2 – x2)	2 番目のスタック項目を削除します。
`over`	(x1 x2 – x1 x2 x1)	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`	(xu … x1 x0 u – xu … x1 x0 xu)	*u* 番目のスタック項目をコピーします (`1 pick = over`)。
`r>`	(– x)(R: x –)	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@`	(– x)(R: x – x)	復帰スタックの一番上をスタックにコピーします。

表 4–2 スタック操作コマンド　続く

コマンド	スタックダイアグラム	説明
`roll`	(xu … x1 x0 u – xu-1 … x1 x0 xu)	u 箇のスタック項目を回転します (`2 roll = rot`)。
`rot`	(x1 x2 x3 – x2 x3 x1)	3 つのスタック項目を回転します。
`swap`	(x1 x2 – x2 x1)	一番上の 2 つのスタック項目を入れ換えます。
`tuck`	(x1 x2 – x2 x1 x2)	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。

代表的なスタック操作の用途は、次の例に示すように、すべてのスタック項目を保持しておきながら、一番上のスタック項目を表示することかも知れません。

```
5 77 ok dup      (一番上のスタック項目を複製)
5 77 77 ok .     (一番上のスタック項目を削除し、表示)
77
5 77 ok
```

カスタム定義の作成

Forth は、新しいコマンドワードの利用者定義を作成するための簡単な手段を提供します。表 4–3に、利用者定義作成用の Forth ワードを示します。

表 4–3 コロン定義ワード

コマンド	スタックダイアグラム	説明
: *new-name*	(–)	ワード *new-name* の新しいコロン定義を開始します。
;	(–)	コロン定義を終了します。

新しいコマンドの定義は、: 用いて定義することから、コロン定義と呼ばれます。たとえば、任意の 4 つの数値を加算し、結果を表示する新しいワード add4 を作成するとします。定義は、たとえば次のように作成できます。

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

; (セミコロン) は、(+ + + .)の操作を行わせるように add4 を定義する定義の終わりを示します。3 つの加算演算子 (+) は 4 つのスタック項目を 1 つの和に変えてスタックに残します。次に . はその結果を削除し、表示します。次に例を示します。.

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

これらの定義はローカルメモリーに格納されます。つまり、システムをリセットすると消去されるということです。よく使う定義を保存するには、(オペレーティングシステムのもとでテキストエディタを使用して、または NVRAMRC エディタを使用して) テキストファイルにそれらの定義を保存します。このテキストファイルは、以降、必要に応じて読み込めます。(ファイルの読み込みについての詳細は、第 5 章を参照してください。)

ユーザーインタフェースから定義を入力すると、: (コロン) を入力してから ; (セミコロン) を入力するまで、ok プロンプトが] (右角括弧) プロンプトになります。たとえば、add4 の定義は次のように入力できます。

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

(テキストファイル内に) 作成するすべての定義には、そのスタック効果がない場合 (–) であっても、それぞれに、定義が表すスタック効果のダイアグラムが必要です。スタックダイアグラムは、ワードの正しい使い方を示すのできわめて重要です。さ

らに、複雑な定義内にはわかりやすいスタックコメントを使用してください。それによって、実行のフローを容易に追跡できます。たとえば、add4 を作成するには、次のように定義できます。

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

または、次のようにも定義できます。

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

注 - 「(」(左側括弧) は、それ以降「)」 (右側括弧) までのテキストを無視することを意味します。ほかのすべての Forth ワードと同様に、左側括弧の右側には 1 つまたはそれ以上の空白文字が必要です。

演算機能の使用方法

表 4–4 に示すコマンドは、データスタック上の項目に対する基本演算コマンドです。

表 4–4 演算機能

コマンド	スタックダイアグラム	説明
+	(nu1 nu2 – sum)	*nu1* + *nu2*の加算を行います。
-	(nu1 nu2 – diff)	*nu1* - *nu2*の減算を行います。
*	(nu1 nu2 – prod)	*nu1* * *nu2*の乗算を行います。

表 4–4 演算機能 続く

コマンド	スタックダイアグラム	説明
/	(n1 n2 – quot)	*n1* を*n2* で割ります。剰余は捨てられます。
/mod	(n1 n2 – rem quot)	*n1* / *n2*の剰余と商。
<<	(x1 u – x2)	lshiftの同義語。
>>	(x1 u – x2)	rshiftの同義語。
>>a	(x1 u – x2)	*x1* を *u* ビット 右に算術シフトします。
*/	(n1 n2 n3 – quot)	*n1* * *n2* / *n3*。
*/mod	(n1 n2 n3 – rem quot)	*n1* * *n2* / *n3*の剰余と商。
1+	(nu1 – nu2)	1を足します。
1-	(nu1 – nu2)	1を引きます。
2*	(nu1 – nu2)	2 を掛けます。
2+	(nu1 – nu2)	2 を足します。
2-	(nu1 – nu2)	2 を引きます。
2/	(nu1 – nu2)	2 で割ります。
abs	(n – u)	絶対値。
aligned	(n1 – n1 a-addr)	*n1* を次の 4 の整数倍に切り上げます。
and	(n1 n2 – n3)	ビット単位の論理積。
bounds	(startaddr len – endaddr startaddr)	*do* ループ用に *startaddr len* を*endaddr* startaddr に変換します。

表 4–4 演算機能 続く

コマンド	スタックダイアグラム	説明
`bljoin`	(b.low b2 b3 b.hi – quad)	4 バイトを結合して 32 ビットの 4 バイトワードを作ります。
`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。
`d+`	(d1 d2 – d.sum)	2 つの 64 ビット数値の加算を行います。
`d-`	(d1 d2 –d.diff)	2 つの 64 ビット数値の減算を行います。
`even`	(n – n n+1)	最も近い偶数の整数 >= *n* に丸めます。
`fm/mod`	(d n – rem quot)	*d* を n で割ります。
`invert`	(x1 – x2)	*x1*の全ビットを反転します。
`lbflip`	(quad1 – quad2)	32 ビットの 4 バイトワード内のバイトをスワップします。
`lbsplit`	(quad – b.low b2 b3 b.hi)	32 ビットの 4 バイトワードを 4 バイトに分割します。
`lwflip`	(quad1 – quad2)	32 ビットの 4 バイトワードの半分をスワップします。
`lwsplit`	(quad – w.low w.hi)	32 ビットの 4 バイトワードを 2 つの 16 ビットワードに分割します。
`lshift`	(x1 u – x2)	*x1* を *u* ビット左シフトし、下位ビットにゼロを埋め込みます。

表 4–4 演算機能 続く

コマンド	スタックダイアグラム	説明
`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* の小さいほうの値を *n3* とします。
`mod`	(n1 n2 – rem)	*n1* / *n2* の剰余を計算します。
`negate`	(n1 – n2)	*n1* の符号を変更します。
`not`	(x1 – x2)	`invert` の同義語。
`or`	(n1 n2 – n3)	ビット単位の論理和。
`rshift`	(x1 u – x2)	*x1* を *u* ビット右シフトし、上位ビットにゼロを埋め込みます。
`s>d`	(n1 – d1)	数値を倍精度数に変換します。
`sm/rem`	(d n – rem quot)	d を *n* で割ります。対称除算。
`u2/`	(x1 – x2)	1 ビット右へ論理シフトし、上位ビットにゼロをシフトします。
`u*`	(u1 u2 – uprod)	符号なしの 2 つの数値を乗算し、符号なしの積を生じます。
`u/mod`	(u1 u2 – urem uquot)	符号なし 32 ビット数値を符号なし 32 ビット数値で割り、32 ビットの剰余と商を生じます。
`um*`	(u1 u2 – ud)	符号なしの 2 つの 32 ビット数値を乗算し、符号なし倍精度数の積を生じます。

表 4–4 演算機能 続く

コマンド	スタックダイアグラム	説明
`um/mod`	(ud u – urem uprod)	ud を *u* で割ります。
`wbflip`	(word1 – word2)	16 ビットワード内のバイトをスワップします。
`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 – quad)	2 ワードを結合して 4 バイトワードを作ります。
`xorxor`	(x1 x2 –x3)	ビット単位排他的の論理和。

メモリーのアクセス

ユーザーインタフェースはメモリー内容の確認および設定用のコマンドを備えています。次の操作はユーザーインタフェースを使用して行います。

- 任意の仮想アドレスの読み取り、書き込み。
- 仮想アドレスの物理アドレスへの対応付け。

メモリー演算子を使用すると、任意のメモリー位置からの読み取り、任意のメモリー位置への書き込みが行えます。以降の例に示すメモリーアドレスはすべて仮想アドレスです。

8 ビット、16 ビット、32 ビットのさまざまな操作ができます。一般的に、`c` (文字) という接頭辞は 8 ビット (1 バイト) の操作を示し、`w` (ワード) という接頭辞は 16 ビット (2 バイト) の操作を示し、`l` (quadlet) という接頭辞は 32 ビット (4 バイト) の操作を示します。

注 - `l` は、場合によっては、数字の 1 との混同を避けるために大文字で示すことがあります。

waddr 、qaddr、addr64 は境界の制約をもつアドレスを示します。たとえば、qaddr は 32 ビット (4 バイト) 境界を示し、したがってそのアドレス値は次の例に示すように 4 で割り切れなければなりません。

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

OpenBoot に実装されている Forth インタプリタはできるだけ ANS の Forth 標準に準拠しています。明示的に 16 ビットまたは 32 ビットを取り出す場合は、@ の代わりにそれぞれ w@ または L@を使用してください。そのほかのメモリーやデバイスレジスタへのアクセスコマンドもこの規則に従います。

表 4–5 にメモリーアクセス用のコマンドを示します。

表 4–5 メモリーアクセスコマンド

コマンド	スタックダイアブラム	説明
!	(x a-addr –)	数値を *a-addr* に格納します。
+!	(nu a-addr –)	*nu* を *a-addr* に格納されている数値に加算します。
<w@<w@	(waddr – n)	doublet *w* を*waddr* から符号拡張して取り出します。
@	(a-addr –x)	数値を *a-addr*から取り出します。
2!	(x1 x2 a-addr –)	2 つの数値を *a-addr* に格納します。*x2* が下位アドレス。
2@	(a-addr – x1 x2)	2 つの数値を *a-addr* から取り出します。 *x2* が下位アドレス。
blank	(addr len –)	*addrlen* からの *len* バイトを空白文字 (10 進の 32) に設定します。
c!	(byte addr –)	*byte* を*addr* に格納します。

表 4–5 メモリーアクセスコマンド 続く

コマンド	スタックダイアグラム	説明
`c@`	(addr – byte)	*byte* を *addr* から取り出します。
`cmove`	(addr1 addr2 u –)	*addr1* から *addr2* に、下位バイトから先に、*u* バイトをコピーします。
`cmove>`	(addr1 addr2 u –)	*addr1* から *addr2* に、上位バイトから先に、 *u* バイトをコピーします。
`cpeek`	(addr – false byte true)	*addr* の 1 バイトを取り出します。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。
`cpoke`	(byte addr – okay?)	*byte* を *addr* 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。
`comp`	(addr1 addr2 len – diff?)	2 つのバイト配列を比較します。 両配列が等しい場合は *diff?* = 0、異なる最初のバイトにおいて、 *addr1* の文字列の方が小さい場合は *diff?*-1、 それ以外の場合は *diff?* = 1 になります。
`dump`	(addr len –)	*addr* から *len* バイトのメモリを表示します。
`erase`	(addr len –)	*addr* から *len* バイトのメモリを `0` に設定します。
`fill`	(addr len byte –)	*addr* から *len* バイトのメモリを値 *byte* に設定します。
`l!`	(n qaddr –)	quadlet *q* を*qaddr* に格納します。

表 4–5 メモリーアクセスコマンド 続く

コマンド	スタックダイアブラム	説明
`l@`	(qaddr – quad)	quadlet *q* を *qaddr* から取り出します。
`lbflips`	(qaddr len –)	指定された領域の各 quadlet 内のバイトを逆に並べ替えます。
`lwflips`	(qaddr len –)	指定された領域の各 quadlet 内の doublet をスワップします。
`lpeek`	(qaddr – false quad true)	32 ビット数値を *qaddr* 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。
`lpoke`	(quad qaddr – okay?)	32 ビットの数値を *qaddr*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。
`move`	(src-addr dest-addr len –)	*src-addr* から *dest-addr* に *len*バイトをコピーします。
`off`	(a-addr –)	*a-addr* に `false` を格納します。
`on`	(a-addr –)	*a-addr* に `true` を格納します。
`unaligned-l!`	(quad addr –)	quadlet *q*を任意の境界に格納します。(境界は問われません。)
`unaligned-l@`	(addr – quad)	quadlet *q* を任意の境界で取り出します。(境界は問われません。)
`unaligned-w!`	(w addr –)	doublet *w* を任意の境界に格納します。(境界は問われません。)
`unaligned-w@`	(addr – w)	doublet *w* を任意の境界で取り出します。(境界は問われません。)

表 4–5 メモリーアクセスコマンド 続く

コマンド	スタックダイアグラム	説明
w!	(w waddr –)	doublet *w* を *waddr* に格納します。
w@	(waddr – w)	doublet *w* を *waddr* から取り出します。
wbflips	(waddr len –)	指定された領域の各 doublet 内のバイトをスワップします。
wpeek	(waddr – false w true)	16 ビットの数値を *waddr* 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。
wpoke	(w waddr – okay?)	16 ビットの数値を *waddr* 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。

dump コマンドは特に便利です。このコマンドは、メモリーの領域をバイト値、ASCII 値の両方で表示します。次の例は、仮想アドレス 10000 からの 20 バイトを表示します。さらに、特定のメモリー位置に読み書きする方法も示しています。

```
ok 10000 20 dump   (仮想アドレス 10000 からの 20 バイトを表示)
      \/ 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!   (メモリー位置 10004 の 8 ビットバイトを 22 に変更)
ok
```

(たとえば、@ を使用して) 無効なメモリー位置をアクセスしようとした場合は、処理はただちに終了し、PROM が Data Access Exception または Bus Error などのエラーメッセージを表示します。(注 : 数字は 16 進数で記述されています。)

表 4–6にメモリー割り当て用のコマンドを示します。

表 4–6　メモリー割り当てコマンド

コマンド	スタックダイアグラム	説明
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	(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。

次の画面は alloc-mem と free-mem の使用例です。

- alloc-mem が 4000 バイトのメモリーを割り当てます。その予約領域の開始アドレス (ffef7a48) が表示されます。
- dump が ffef7a48 から始まるメモリー 20 バイトの内容を表示します。
- 次に、このメモリー領域を値 55でみたします。(注 : ここで数字は 16 進数で記述されています。)
- 最後に、free-mem が割り当てられた ffef7a48 からの 4000 バイトのメモリーを返します。

```
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
```

memmap の使用例を次に示します。

```
ok 200.0000 sbus 1000 memmap ( virt )
ok
```

SBus デバイスのマップ

ここでは、システムに依存するデバイスアドレスを知る必要なしに、ok プロンプトから SBus デバイスを割り当てる一般的な方法を示します。この方法は、SBus デバイス上に有効な FCode PROM があるかどうかには依存しません。2.0 またはそれ以降のバージョンのすべての OpenBoot システムで有効です。

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

たとえば、システムのスロット番号 3 のデバイスの FCode PROM の内容を調べるには、次のように入力します。

```
ok '' /sbus''  select-dev
ok 0 3 1000 map-in  .s
ffed3000
ok dup 20 dump
( FCode PROM の内容が最初の 20 バイト表示されます)
ok
```

この方法は、多少変更することができます。

1. システムによっては、システムの SBus のパス名が異なることがあります。たとえば、" /iommu/sbus" (Sun4m の場合)や " /io-unit/sbi" (Sun4d の場合)です。ok プロンプトで show-devs コマンド (すべてのシステムデバイスをリストします) を使用するのが、正しいパスを知る 1 つの方法です。
2. スタックに (オフセットサイズを) 直接入れる方法は、将来のシステムの一般的ケースでは有効である場合とそうでない場合があります。問題が生じた場合は、さらに一般的な手段として、次の方法を試みてください。

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

ワード定義の使用方法

辞書には用意されているすべての Forth コマンドが含まれています。ワード定義を使って新しい Forth コマンドを作成します。

ワード定義は 2 つのスタックダイアグラムを必要とします。最初のダイアグラムでは、新しいコマンドを作成するときのスタック効果を示します。第 2番目の (Usage:) ダイアグラムはそのコマンドが後で実行されるときのスタック効果を示します。

表 4–7 に辞書エントリを作成するためのワード定義を示します。

表 4–7 ワード定義

コマンド	スタックダイアグラム	説明
: *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。
`alias` *new-name old-name*	(–) Usage: (??? – ?)	*old-name* と同じ操作をする *new-name* を作成します。
`buffer:` *name*	(size –) Usage: (– a-addr)	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。
`2constant` *name*	(n1 n2 –) Usage: (– n1 n2)	2 つの数値の定数を定義します。

表 4–7 ワード定義 続く

コマンド	スタックダイアグラム	説明
`create` *name*	(–) Usage: (– waddr)	汎用ワード定義。
`defer` *name*	(–) Usage: (??? – ?)	フォワードリファレンス、または実行トークンを使用する実行ベクトル用のワードを定義します。
`does>`	(– waddr)	ワード定義のための実行時の節を開始します。
`field` *name*	(offset size – offset+size) Usage: (addr – addr+offset)	指定されたオフセットポインタを作成します。
`struct`	(– 0)	`field` の作成に備えて初期化します。
`value` *name*	(n –) Usage: (– n)	指定された、変更可能な 32 ビットの数値を作成します。
`variable` *name*	(–) Usage: (– waddr)	変数を定義します。

ワード定義 `constant` を使用して、値が変わらない名前を作成できます。単純なコロン定義 `: foo 22 ;` でも同じ結果になります。

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

`value` では任意の数値に名前を付けることができます。その名前を後で実行すると、その代入値がスタックに残されます。次の例は 22 という値を `foo` という名前のワードに代入し、次に `foo` を呼び出してその代入値を演算に使用します。

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

値は辞書コンパイルワード is で変更できます。たとえば、次の例を参照してください。

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

value を使用して作成したコマンドは、数値が必要な場合、@ を使用しないで済むので便利です。

ワード定義 variable は 32 ビットのメモリー領域に名前を割り当てます。この領域は必要に応じて値の保存用として使用できます。後でその名前を実行すると、領域のメモリーアドレスがスタックに残されます。一般的に、そのアドレスの読み書きには @ と ! が使用されます。次の例を参照してください。

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

ワード定義 defer により、必要に応じて異なる機能を読み込むスロットが生成されるので、後から機能を変更できるコマンドを生成できます。次の例を参照してください。

```
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
```

辞書の検索

「辞書」 にはシステムが備えているすべての Forth コマンドが含まれています。表 4–8 に辞書検索用のツールを示します。

表 4–8 辞書検索コマンド

コマンド	スタックダイアグラム	説明
' *name*	(– xt)	指定され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。外部定義を使用します。
['] *name*	(– xt)	内部、外部のどちらの定義でも使用される点以外は、'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のリストを表示します。
$find	(addr len – addr len false xt n)	ワードを検索します。検索できなかった場合 *n* = 0 、検索できた場合 *n* = 1、それ以外の場合 *n* = -1 。
find	(pstr – pstr false xt n)	ワードを辞書から検索します。検索するワードは *pstr* です。検索できなかった場合 *n* = 0、検索できた場合 *n* = 1、それ以外の場合 *n* = -1 。

表 4–8 辞書検索コマンド 続く

コマンド	スタックダイアグラム	説明
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。
sift	(pstr –)	*pstr* によって示される文字列を含むすべての辞書エントリの名前を表示します。
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	(–)	辞書内のすべての表示可能なワードを表示します。

see *thisword* の形式で使用した場合、see は指定されたコマンドを逆コンパイルします (つまり、*thisword* を作成するための定義を表示します)。メモリースペースを節減するために内部の名前が PROM のシンボルテーブルから省略されていることがあるため、逆コンパイル結果の定義はときとして不明確なものになる場合があります。

次の画面は sifting の使用例です。

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

words は、辞書内のすべてのワード (コマンド) 名を最新の定義から先に表示します。

データを辞書へコンパイルする

表 4–9 に、データを辞書へコンパイルするためのコマンドを示します。

表 4–9 辞書コンパイルコマンド

コマンド	スタックダイアグラム	説明
,	(n –)	数値を辞書に入れます。
c,	(byte –)	1 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,	(quad –)	32 ビット数値を辞書に入れます。
[	(–)	解釈を開始します。
]	(–)	解釈を終了し、コンパイルを再開します。
allot	(n –)	辞書に *n* バイトを割り当てます。
>body	(xt – a-addr)	実行トークンからデータフィールドアドレスを見つけます。
body>	(a-addr – xt)	データフィールドアドレスから実行トークンを見つけます。
compile	(–)	次のワードを実行時にコンパイルします。
[compile] *name*	(–)	次の (すぐ次の) ワードをコンパイルします。
forget *namep*	(–)	指定されたワードとそれ以降の全ワードを辞書から削除します。
here	(– addr)	辞書の先頭アドレス。
immediate	(–)	最後の定義を即値としてマークします。
to *name*	(n –)	defer ワードまたは value に新しい処理を実装します。
literal	(n –)	数値をコンパイルします。
origin	(– addr)	Forth システムの開始アドレスを返します。

表 4–9 辞書コンパイルコマンド 続く

コマンド	スタックダイアグラム	説明
`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 –)	Replace *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。
`recursive`	(–)	辞書内のコンパイル中のコロン定義の名前を表示可能にし、したがって、そのワードの名前をそれ自身の定義内で再帰的に使用可能にします。
`state`	(– addr)	コンパイル状態のゼロ以外の変数。

数値の表示

表 4–10 にスタック値表示用の基本コマンドを示します。

表 4–10 基本数値表示

コマンド	スタックダイアグラム	説明
`.`	(n –)	数値を現在の基数で表示します。
`.r`	(n size –)	数値を固定幅フィールドで表示します。
`.s`	(–)	データスタックの内容を表示します。
`showstack`	(??? – ???)	各 ok プロンプトの前で自動的に `.s` を実行します。
`noshowstack`	(??? – ???)	各 ok プロンプトの前でのスタック表示をオフにします。

表 4–10 基本数値表示 続く

コマンド	スタックダイアグラム	説明
`u.`	(u –)	符号なし数値を表示します。
`u.r`	(u size –)	数値を固定幅フィールドで表示します。

`.s` コマンドはスタックの内容全体をそのまま表示します。このコマンドはいつでもデバッグ目的に使用して安全です。(これは、`showstack` が自動的に実行する機能です。)

基数の変更

表 4–11のコマンドを使用して、現在使用している数値の基数を変更できます。

表 4–11 基数の変更

コマンド	スタックダイアグラム	説明
`.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。
`base`	(– addr)	基数を格納している変数。
`decimal`	(–)	基数を 10 進に設定します。
`d#` *number*	(– n)	*number* を 10 進に変換します。基数は変わりません。
`hex`	(–)	基数を 16 進に設定します。
`h#` *number*	(– n)	*number* を 16 進に変換します。基数は変更されません。

表 4–11 基数の変更 続く

コマンド	スタックダイアグラム	説明
octal	(–)	基数を 8 進に設定します。
o# *number*	(– n)	*number* を 8 進に変換します。基数は変更されません。

d# 、h# 、o# の各コマンドは、現在の基数を明示的に変更しないで、特定の数値を別の基数で入力するときに便利です。

```
ok decimal          (基数を 10 進に変更)
ok 4 h# ff 17 2
4 255 17 2 ok
```

.d および .h コマンドの機能は、現在の基数設定にかかわりなく、値をそれぞれ 10 進または 16 進で表示する点を除いて、「.」と同じです。次の例を参照してください。

```
ok hex
ok ff .  ff .d
ff 255
```

テキスト入出力の制御

この節ではテキストの入出力用コマンドについて説明します。これらのコマンドは文字列や文字配列を制御し、ユーザーからのコメント入力およびキーボードの走査制御を可能にします。

表 4–12 にテキスト入力制御用のコマンドを示します。

表 4–12 テキスト入力制御

コマンド	スタックダイアグラム	説明
(*ccc*)	(–)	コメントを作成します。習慣上スタックダイアグラム用に使用されます。
\ *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 文字の数値を取り出します。
`expect`	(addr +n –)	割り当てられた入力デバイスのキーボードから編集結果の 1 行を受け取り、*addr* に格納します。
`key`	(– char)	割り当てられた入力デバイスのキーボードから 1 文字を読みます。
`key?`	(– flag)	入力デバイスのキーボードでキーが押された場合 true。
`span`	(– waddr)	`expect`で読み出された文字数を格納する変数。
`word`	(char – pstr)	入力文字列から *char* で区切られる文字列を集め、メモリー位置 *pstr* に入れます。

コメントは、コードの機能を記述するために、(一般的にテキストファイル内の) Forth ソースコードに使用します。((左側括弧) がコメントを開始する Forth ワードです。右側括弧)の前までの文字はすべて、Forth インタプリタが無視します。スタックダイアグラムは (を使用するコメントとして取り扱われます。

注 - (の後に空白文字を入れることを忘れないでください。それによって、(は Forth ワードとして認識されます。

\(バックスラッシュ) はテキスト行末でコメントが終わりになることを示します。

`key` はキーが押されるまで待ち、押されると、そのキーの ASCII 値をスタックに返します。

`ascii` は、`ascii` *x* の形式で使用され、文字 *x* の数字コードをスタックに返します。

key? はキーボードを走査して、ユーザーが新たになんらかのキーを押したかどうかを調べ、フラグをスタックに返します。つまり、キーが押されていた場合は true を、押されていない場合は false を返します。フラグの使い方については、102ページの「条件フラグ」 の説明を参照してください。

表 4–13 に汎用のテキスト表示用コマンドを示します。

表 4–13 テキスト出力表示

コマンド	スタックダイアグラム	説明
.″ ccc"	(–)	後の表示に備えて、文字列をコンパイルします。
(cr	(–)	出力カーソルを現在行の先頭に戻します。
cr	(–)	ディスプレイ上の行を終了し、次の行に進みます。
emit	(char –)	現在位置の文字を表示します。
exit?	(– flag)	スクロール制御プロンプト More [<space>,<cr>,q] ? を有効にします。 リターンフラグは、ユーザーが出力を終了する場合 true です。
space	(–)	空白文字 を表示します。
spaces	(+n –)	*+n* 箇の空白文字を表示します。
type	(addr +n –)	*addr* から始まる *+n* 箇の文字を表示します。

cr はキャリッジリターン文字を出力に送ります。次の例を参照してください。

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

emit は ASCII 値がスタックにある英字を表示します。

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

表 4–14 にテキスト文字列操作用のコマンドを示します。

表 4–14 テキスト文字列操作

コマンド	スタックダイアグラム	説明
",	(addr len –)	*addr* から始まり、長さが *len* のバイトの配列をパックされた文字列としてコンパイルし、辞書の先頭に入れます。
″ ccc"	(– addr len)	翻訳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。文字列内に"(00,ff) を使用して任意のバイト値を含めることができます。
.(*ccc*)	(–)	文字列を即時に表示します。
-trailing	(addr +n1 – addr +n2)	後続空白文字を削除します。
bl	(– char)	空白文字の ASCII コード。10 進の 32。
count	(pstr – addr +n)	パックされている文字列をアンパックします。
lcc	(char – lowercase-char)	文字を小文字に変換します。
left-parse-string	(addr len char – addrR lenR addrL lenL)	文字列を *char* で分割します (char は捨てられます)。

表 4–14 テキスト文字列操作 続く

コマンド	スタックダイアグラム	説明
pack	(addr len pstr – pstr)	*addr len* からパックされた文字列を作り、 *pstr* に入れます。
"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。
upc	(char – uppercase-char)	文字を大文字に変換します。

一部の文字列操作コマンドは、アドレス (それらの文字があるメモリー内の位置) と長さ (文字列の文字数) を指定します。その他のコマンドは、パックされた文字列、または長さを表すバイトを格納するメモリー位置である pstr とその後の一連の文字を使用します。コマンドのスタックダイアグラムは、どの形式が使用されるかを示します。たとえば、count はパックされた文字列を addr-len (アドレスと長さの組み合わせ) 文字列に変換します。

コマンド ." は ." *string*" の形式で使用します。このコマンドは必要なときにテキストを出力します。"(二重引用符) はテキスト文字列の終わりを示します。次の例を参照してください。

```
ok  : testing 34 .  ." This is a test"  55 . ;
ok
ok testing
34 This is a test55
ok
```

入出力先の変更

通常、システムはすべてのユーザー入力にキーボードを、また大部分の表示出力にディスプレイ画面付きのフレームバッファーをそれぞれ使用します。(サーバーシステムはシステムのシリアルポートに接続された ASCII 端末を使用できます。システ

ム本体への端末の接続についての詳細は、システムのインストールマニュアルを参照してください。) 入力先、出力先、それらの両方をシステムのいずれかのシリアルポートに変更できます。これは、たとえばフレームバッファーのデバッグ時に便利です。

表 4–15 に入出力先変更用のコマンドを示します。

表 4–15 入出力先の変更用コマンド

コマンド	スタックダイアグラム	説明
`input`	(device –)	入力用のデバイス (`keyboard` または *device-specifier*) を選択します。
`io`	(device –)	入出力用のデバイスを選択します。
`output`	(device –)	出力用のデバイス (`screen` または *device-specifier*) を選択します。

`input` および `output` コマンドは、それぞれ、現在の入力および出力用デバイスを一時的に変更します。変更はコマンドの入力時に行われます。システムをリセットする必要はありません。システムリセットまたは電源再投入を行うと、入出力デバイスは NVRAM システム変数 `input-device` と `output-device` に指定されているデフォルト設定に戻ります。これらの変数は、必要に応じて変更できます (デフォルトの変更についての詳細は、第 3 章を参照してください)。

`input` の前には、`keyboard`、`ttya`、`ttyb` または *device-specifier* テキスト文字列のうちのどれか 1 つを入れます。たとえば、入力が現在キーボードから受け入れられていて、入力がシリアルポート ttya に接続されている端末から受け入れられるように変更する場合、次のように入力します。

```
ok ttya input
ok
```

この時点で、キーボードは (`Stop-A` 以外は) 機能しなくなりますが、ttya に接続されている端末から入力されるテキストはすべて入力として処理されるようになります。すべてのコマンドが通常どおりに実行されます。

キーボードを再び入力デバイスとして使用するには、端末のキーボードを使用して次のように入力します。

```
ok keyboard input
ok
```

同様に、output の前にも screen、ttya、または ttyb のどれか 1 つを入れます。たとえば、通常のディスプレイ画面でなく、ttya に出力を送りたい場合は、次のように入力します。

```
ok ttya output
```

通常のディスプレイ画面は応答の ok プロンプトを表示せず、ok プロンプトも以降のすべての出力も ttya に接続されている端末に表示されます。

io も、入出力の両方を指定した場所に変更する点以外、同じ方法で使用されます。

一般的に、input、output、io には *device-specifier* を指定する必要があります。*device-specifier* はデバイスパス名、デバイスの別名のどちらでもかまいません。次の 2 つの例に示すように、デバイスは、二重引用符 (") を使用して Forth の文字列として次のように指定する必要があります。

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

または、次のように指定します。

```
ok " screen" output
```

上記の 2 つの例では 、ttya、screen、 keyboard は Forth のワードであり、いずれも、それぞれの対応のデバイス別名文字列をスタックに入れます。

コマンド行エディタ

OpenBoot では、ユーザーインタフェース用として (一般的なテキストエディタである EMACS のような) 使用するコマンド行エディタ、一部のオプション拡張、オプションの履歴機を指定しています。これらの強力なツールを使用して、前のコマンドを入力し直さないで再び実行して、現在のコマンド行を編集して入力エラーを修正し、前のコマンドを呼び出すことができます。

表 4–16に、ok プロンプト時に使用できる行編集用のコマンドを示します。

表 4–16 コマンド行エディタ用必須キー操作コマンド

操作キー	説明
Delete	1 つ前の文字を消去します。
Backspace	1 つ前の文字を消去します。
Control-U	現在の行を消去します。
Return (Enter)	現在の行の編集を終了し、表示されている 1 行全部をインタプリタに渡します。

OpenBoot 標準でも、これらの機能の 3 つの拡張グループを記述しています。表 4–17に、コマンド行編集用の拡張グループを示します。

表 4–17 コマンド行エディタ用オプションキー操作コマンド

操作キー	説明
Control-B	1 文字位置戻ります。
Escape B	1 語後戻ります。
Control-F	1 文字位置進みます。
Escape F	1 語進みます。
Control-A	行の始めまで戻ります。

表 4–17 コマンド行エディタ用オプションキー操作コマンド 続く

操作キー	説明
Control-E	行の終わりに進みます。
Delete	前の 1 文字を消去します。
Backspace	前の 1 文字を消去します。
Control-H	前の 1 文字を消去します。
Escape H	語の初めからカーソルの直前まで消去し、消去した文字を保存バッファーに格納します。
Control-W	語の初めからカーソルの直前まで消去し、消去した文字を保存バッファーに格納します。
Control-D	カーソル位置の文字を消去します。
Escape D	カーソル位置から語の終りまで消去し、消去した文字を保存バッファーに格納します。
Control-K	カーソル位置から行の終りまで消去し、消去した文字を保存バッファーに格納します。
Control-U	1 行を全部消去し、消去した文字を保存バッファーに格納します。
Control-R	1 行を表示しなおします。
Control-Q	次の文字の前に引用符を付けます (制御文字を挿入できます)。
Control-Y	保存バッファーの内容をカーソル位置の前に挿入します。

コマンド行履歴の拡張機能として、前に入力したコマンドを EMACS のようなコマンド履歴バッファーに保存できます。このバッファーは 8 エントリ以上入れることができます。保存したコマンドは、バッファー内を前後に移動することにより、再び呼び出すことができます。呼び出したコマンドは、編集したり、(Return キーを押して) 再びインタプリタに渡すことができます。表 4–18 にコマンド行履歴拡張用のキーを示します。

表 4–18 コマンド行履歴用オプションキー操作コマンド

操作キー	説明
Control-P	コマンド履歴バッファー内の 1 行前のコマンド行を表示します。
Control-N	コマンド履歴バッファー内の次のコマンド行を表示します。
Control-L	Dコマンド履歴バッファー全てを表示します。

コマンド名補完機能では、ワードのすでに入力した部分に基づいて辞書から 1 つまたはそれ以上の一致するワードを検索して、長い Forth のワード名を補完します。ワードの一部を入力した後にコマンド補完キー操作として `Control-Space` を押すと、システムは次のように応答します。

- システムが一致ワードを 1 つだけ検索した場合は、ワードの未入力部分が自動的に表示されます。
- システムは、一致候補を複数検索した場合は、すべての候補のすべての共通文字を表示します。
- システムは、入力した文字に一致する文字を検索できなかった場合は、残りの文字との一致が 1 つ以上現れるまで、右側から文字を削除します。
- システムは、正しい候補を判定できない場合は、警報音を鳴らします。

表 4–19にコマンド補完拡張用のキーを示します。

表 4–19 コマンド補完用オプションキー操作コマンド

操作キー	説明
Control-Space	現在のワードの名前を補完します。
Control-?	現在のワードのすべての一致候補を表示します。
Control-/	現在のワードのすべての一致候補を表示します。

条件フラグ

Forth の条件付き制御コマンドはフラグを使用して真/偽の値を示します。フラグは、テスト基準に基づいて、いくつかの方法で生成できます。生成できたら、ワード "." でスタックから表示したり、条件付き制御コマンドの入力として使用できます。条件付き制御コマンドは、フラグが真 (true) の場合と偽 (false) の場合にそれぞれ異なる応答を表示します。

表示値が `0` の場合は、フラグの値が `false`であることを示します。`-1` またはその他のゼロ以外の任意の数値は、フラグが `true`であることを示します。(16 進では、値 `-1` は `ffffffff` として表示されます。)

表 4–20 に、比較テストを実行し、`true`または `false`フラグの結果をスタックに残すコマンドを示します。

表 4–20 比較コマンド

コマンド	スタックダイアグラム	説明
`<`	(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。
`0<`	(n – flag)	*n* < 0 の場合 true。
`0<=`	(n – flag)	*n* <= 0 の場合 true。
`0<>`	(n – flag)	*n* <> 0 の場合 true。
`0=`	(n – flag)	*n* = 0 の場合 true (さらにフラグを反転します)。

表 4–20 比較コマンド 続く

コマンド	スタックダイアグラム	説明
0>	(n – flag)	*n* > 0 の場合 true。
0>=	(n – flag)	*n* >= 0 の場合 true。
between	(n min max – flag)	*min* <= *n* <= *max* の場合 true。
false	(– 0)	FALSE (偽) の値 0。
true	(– -1)	TRUE (真) の値 -1。
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。u1、u2 とも符号なし。
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。u1、u2 とも符号なし。
within	(n min max – flag)	*min* <= *n* < *max* の場合 true。

> はスタックから 2 つの数値を取り出し、最初の数値が 2 番目の数値より大きかった場合は true (-1) をスタックに返し、そうでなかった場合は false (0)を返します。次に例を示します。

```
ok 3 6 > .
0                (3 は 6 より大きくない)
ok
```

0= はスタックから 1 項目を取り出し、その項目が 0 であった場合は true を返し、そうでなかった場合は false を返します。このワードはどちらのフラグもその反対の値に反転します。

制御コマンド

以降の各項では、Forth プログラム内の実行フローの制御用のワードについて説明します。

if...else...then 構造

if、then、else の各コマンドは組み合わされて単純な制御構造を作ります。

表 4–21に条件付き実行フロー制御用のコマンドを示します。

表 4–21 if...else...then コマンド

コマンド	スタックダイアグラム	説明
if	(flag –)	flag が true の場合、このコマンドの後のコードを実行します。
else	(–)	if が false の場合、このコマンドの後のコードを実行します。
then	(–)	if...else...then を終了します。

これらのコマンドの書式は次のとおりです。

```
flag if
  (true の場合これを実行)
else
  (false の場合これを実行)
then
  (通常どおりに実行を継続)
```

または

```
flag if
  (true の場合これを実行)
then
  (通常どおりに実行を継続)
```

if コマンドはスタックからフラグを 1 つ「消費」します。そのフラグが true (ゼロ以外) であれば、if の後のコマンドが実行されます。true でなければ、(存在する場合) else の後のコマンドが実行されます。

```
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
```

注 -] プロンプトは、それが現れる間は、新しいコロン定義の作成の途中であることをユーザーに示します。このプロンプトはセミコロンを入力して定義を終了すると ok に戻ります。

---

case 文

高水準の case コマンドであり、複数の候補のなかから代替実行フローを選択するために用意されています。このコマンドの方が、深く入れ子になった ifthen コマンドよりも読みやすいという利点があります。

表 4–22に条件付き case コマンドを示します。

表 **4–22** case文コマンド

コマンド	スタックダイアグラム	説明
case	(selector – selector)	caseendcase 条件付き構造を開始します。
endcase	(selector {empty} –)	caseendcase 条件付き構造を終了します。
endof	(–)	caseendcase 条件付き構造内の ofendof 句を終了します。
of	(selector test-value – selector {empty})	case 条件付き構造内の ofendof 句を開始します。

case コマンドの使用例を示します。

```
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
```

注 - (省略可能な) default 句はまだスタックにあるテスト値を使用できますが、その値を削除してはなりません (. でなく dup .を使用してください)。of 句が正常に実行されれば、テスト値はスタックから自動的に削除されます。

begin ループ

begin ループは、特定の条件が満たされるまで、同じコマンドの実行を繰り返します。そのようなループのことを条件付きループといいます。

表 4–23に条件付きループの実行制御用のコマンドを示します。

表 4–23 begin(条件付き) ループコマンド

コマンド	スタックダイアグラム	説明
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 である間、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。

次に 2 つの一般的な形式を示します。

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

および

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

上記の両方の場合とも、所定のフラグ値によってループが終了させられるまで、ループ内のコマンドが繰り返し実行されます。ループが終了すると、通常、実行はループを閉じているワード (until または repeat) の後のコマンドに継続されます。

beginuntil の場合は、until がスタックの一番上からフラグを削除してそれを調べます。フラグが false の場合は、実行は beginのすぐ後に引き継がれて、ループが繰り返されます。フラグの trueの場合は、実行はループから抜け出ます。

beginwhilerepeat の場合は、while がスタックの一番上からフラグを削除して調べます。フラグが true の場合は、whileのすぐ後のコマンドが実行されてループが繰り返されます。repeatコマンドは制御を自動的に begin に戻してループを継続させます。while が現れたときにフラグが false であった場合は、実行はただちにループから抜け出し、制御がループを閉じている repeat の後の最初のコマンドに移ります。

これらのループのいずれについても、「true ならば通り過ぎる」と覚えると忘れないでしょう。

次に簡単な例を示します。

```
ok begin 4000 c@ .  key? until (任意のキーが押されるまで繰り返す)
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43
ok
```

この例では、ループはまずメモリー位置 4000 から 1 バイトを取り出して表示します。次に、key? コマンドが呼び出され、これが、ユーザーがそれまでにどれかのキーを押していれば true をスタックに残し、そうでない場合は false を残します。このフラグはuntil によって「消費」され、その値が false であった場合は、ループが繰り返されます。キーを押せば、次に呼び出されたとき、key? は true を返し、したがってループは終了します。

Forth の多くのバージョンとは異なり、ユーザーインタフェースの場合は、ループや条件付き制御構造を対話的に使用できます。つまり、まず最初に定義を作成する必要がありません。

do ループ

do ループ (カウント付きループとも呼ばれます) は、ループの繰り返し回数があらかじめ計算できるときに使用します。do ループは、通常、指定した終了値に達する直前に終了します。

表 4–24に カウント付きループの実行制御用コマンドを示します。

表 4–24 do (カウント付き)ループコマンド

コマンド	スタックダイアグラム	説明
+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* まで使用されます)。
?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 まで使用されます。 *end* = *start* の場合はループは実行されません。
?leave	(flag –)	*flag* がゼロ以外の場合、実行を do...loop から抜けます。
do	(end start –)	do...loop を開始します。インデックスは *start* から *end*まで使用されます。 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をスタックに残します。
j	(– n)	1 つ外側のループのループインデックスをスタックに残します。
leave	(–)	実行を do...loop から抜けます。
loop	(–)	do...loop を終了します。

次の画面で、ループの使用方法をいくつか示します。

```
ok 10 5 do  i .  loop
5 6 7 8 9 a b c d e f
ok
```

(続く)

```
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
       (5000 ～ 6000 のメモリー領域を走査して指定のバイト値に一致しないバイトを調べる)
]    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
     (メモリー領域にステップパターンを埋め込む)
ok
ok 500 value testloc
ok : test16 ( -- ) 1.0000 0 ( do 0-ffff )
     (指定位置に異なる 16 ビット値を書き込む)
]     do i testloc w! testloc w@ i <> ( error? )
     (さらに位置をチェック)
]       if ." Error - wrote " i . ." read " testloc w@ . cr
]        leave ( exit after first error found )
     (この行は省略可能)
]       then
]     loop
]  ;
ok test16
ok 6000 to testloc
ok test16
Error - wrote 200 read 300
ok
```

その他の制御コマンド

表 4–25に、前記以外のプログラム実行制御用のコマンドについて説明します。

表 4–25 プログラム実行制御コマンド

コマンド	スタックダイアグラム	説明
`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? –)	*abort?* が true の場合は、実行を終了させてメッセージを表示します。
`eval`	(addr 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`	(–)	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。

`abort` はプログラムの実行を即時に終了させ、制御をキーボードに返します。`abort"` は 2 点を除いて `abort` と同じです。第 1 点は、フラグが `true` の場合にスタックからフラグを削除し、その後は何もしないで強制終了させることです。もう 1 点は、強制終了が行われたとき、なんらかのメッセージを表示することです。

`eval` は (アドレスと長さにより指定された) 文字列をスタックから取り出します。次に、キーボードから入力される場合と同様に、その文字列の文字が解釈されます。Forth のテキストファイルをメモリーに読み込んでいる (第 5 章を参照) 場合は、`eval` を使用してそのファイル内の定義をコンパイルできます。

プログラムの読み込みと実行

ユーザーインタフェースにはプログラムを読み込み、実行するいくつかの方法があります。これらの方法は、それぞれ、Ethernet、ハードディスク、フロッピーディスク、シリアルポート A からファイルを読み込むためのもので、Forth、FCode、実行可能バイナリプログラムをサポートします。

表 5–1 に、いろいろなソースからファイルを読み込むコマンドを示します。

表 5–1 ファイル読み込み用コマンド

コマンド	スタックダイアグラム	説明
`?go`	(–)	Forth、FCode、またはバイナリプログラムを実行します。
`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`	(addr 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`	(–)	端末エミュレータを使用してシリアルラインから Forth ファイルを読み込み、解釈します。例として、`tip` を使用する場合は次のように入力します。 **`~C cat`** *filename* **`^-D`**

表 5–1 ファイル読み込み用コマンド 続く

コマンド	スタックダイアグラム	説明
`dlbin`	(–)	端末エミュレータを使用してシリアルラインからバイナリファイルを読み込み、解釈します。例として、 `tip` を使用する場合は次のように入力します。 **`~C cat`** *filename*
`dload` *filename*	(addr –)	Ethernet を通じて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。
`eval`	(addr len –)	読み込まれた Forth テキストファイルを解釈します。
`go`	(–)	前に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` *device-specifier argument*	(–)	指定されたデバイスから `load-base` によって指定されるメモリーアドレスにデータを読み込みます。
`load-base`	(– addr)	`load` コマンドによりデバイスから読み込んだデータを読み込むアドレス。

`dload` を使って **Ethernet** から読み込む

`dload` は、次に示すように、Ethernet を通じて指定されたアドレスに読み込みます。

```
ok 4000 dload filename
```

上記の例で、*filename* はサーバーのルートからの相対パス名でなければなりません。dload 入力のアドレスとして **4000** (16 進) を使用します。dload は簡易ファイル転送プロトコル (TFTP) を使用するので、このコマンド行が正しく動作するためには、サーバーのアクセス権の調整が必要なことがあります。

Forth プログラム

dload で読み込む Forth プログラムは、最初の 2 文字が「\ 」(バックスラッシュと空白文字) である ASCII ファイルでなければなりません。読み込んだ Forth プログラムを実行するには、次のように入力します。

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

上記の例で、*file-size* には読み込んだイメージのサイズを設定します。

FCode プログラム

dload で読み込む FCode プログラムは a.out ファイルでなければなりません。読み込んだ FCode プログラムを実行するには、次のように入力します。

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

byte-loadは、SBus などの拡張ボード上での FCode プログラムの解釈用として OpenBoot が使用します。例の中にある 1 は、一般の場合の FCode 間の区切り指定用の変数の特定値です。dloadはシステムメモリーに読み込まれるので、1 は正しい区切りになります。

実行可能バイナリ

dload で読み込む実行可能なバイナリプログラムは、a.outファイルであり、dloadの入力アドレス (4000) を実行するようにリンクされていなければなりません。または、位置に依存しないようになっていなければなりません。バイナリプログラムを実行するには、次のように入力します。

```
ok go
```

上記のプログラムをもう一度実行するには、次のように入力します。

```
ok init-program go
```

dload は、(起動コマンドの場合とは異なり) 中間起動プログラムを使用しません。したがって、a.out ファイル内のシンボル情報はすべてユーザインタフェースのシンボリックデバッグ機能で処理できます。(シンボリックデバッグについての詳細は、第 6 章 を参照してください)。

boot を使ってディスク、フロッピーディスク、または Ethernet から読み込む

通常はオペレーティングシステムを読み込むのに使用される boot でも、プログラムを読み込み、実行することができます。boot の形式は次のとおりです。

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

device-specifier はデバイスのフルパス名かデバイスの別名です。(デバイスのパス名および別名についての詳細は、第 1 章を参照してください。)

ハードディスクまたはフロッピーディスクのパーティションからの読み込みの場合は、*filename* は該当するファイルシステムからのファイルパスです。(起動可能フロッピーディスクの作成については、付録 Bを参照してください。) Ethernet からの読み込みの場合は、*filename* はそのルートサーバー上のシステムのルートパーティションからのファイルパスです。どちらの場合も、先行の /をファイルパスでは省略する必要があります。

-h フラグは、プログラムを読み込むが、実行しないことを示します。

boot は、その処理を行うために、中間起動プログラムを使用します。ハードディスクまたはフロッピーディスクから読み込むときには、OpenBoot はまずディスクの起動ブロックを読み込み、次にこのブロックが第 2 レベルの起動プログラムを読

み込みます。Ethernet から読み込むときには、OpenBoot ファームウェアは TFTP を使用して第 2 レベルの起動プログラムを読み込みます。*filename* と -h はそれらの中間起動プログラムに渡されます。

Forth プログラム

Forth のプログラムは ASCII ファイルであり、これは、2 次起動プログラムが必要とするファイル形式に変換されなければなりません。この変換用として、サン の SBus サポートグループから fakeboot という名前のユーティリティーが提供されています。ファイルは、メモリーに読み込んだ後は、eval コマンドを使用して実行できます。

たとえば、ファイルをアドレス 4010 (16 進) に読み込んでいて、その934 バイト分について実行する場合は、次のように入力します。

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

FCode プログラム

トークン生成プログラム (FCode 生成プログラム) によって生成された FCode プログラムは、2 次起動プログラムのファイル形式への変換が必要なことがあります。この処理には fakeboot が便利です。ファイルがメモリーに用意できたら、byte-load コマンドで実行します。

たとえば、ファイルがアドレス 4030 (16 進) に読み込みしてあるものとすれば、次のように入力します。

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

実行可能バイナリプログラム

オペレーティングシステム以外のバイナリプログラムも、次のように入力して読み込み、実行できます。

```
ok go
```

boot コマンドは -h を使用しているので、go が必要です。

dl を使ってシリアルポートから Forth を読み込む

dl で読み込む Forth のプログラムは ASCII ファイルでなければなりません。

シリアルラインからファイルを読み込むには、被試験システムのシリアルポートを、要求があればすぐにファイルを転送できるマシンに接続し、そのシステムで端末エミュレータを起動します。次に、端末エミュレータを使用して dl でファイルをダウンロードします。

次の例では、UNIX の端末エミュレータ tip を使用するものとします。(この手順についての詳細は、付録 A を参照してください。

1. ok プロンプトで次のように入力します。

```
ok dl
```

2. もう一方のシステムの tip ウィンドウから、ファイルを転送し、その後 Control-D を押してファイルの終わりを知らせます。

```
~C (local command) cat filename
(2 秒待つ)
^-D
```

ファイルは、読み込まれた後に自動的に解釈され、ファイルが読み込まれたシステムの画面に再び ok プロンプトが現れます。

dlbin を使ってシリアルポートから FCode またはバイナリを読み込む

dlbin で読み込む FCode プログラムおよびバイナリプログラムは a.out ファイルでなければなりません。dlbin は、それらのファイルを a.out のヘッダーで示されるエントリポイントに読み込みます。最近のバージョンの FCode トークン生成プログラムは、エントリポイント を 4000 として a.out ファイルを作成します。

シリアルラインからファイルを読み込むには、システムのシリアルポート A を、要求があればすぐにファイルを転送できるマシンに接続します。次の例では、tip ウィンドウのセットアップを前提としています。(この手順については、付録 A を参照してください。)

1. ok プロンプトで、次のように入力します。

```
ok dlbin
```

2. もう一方のシステムの **TIP** ウィンドウからファイルを転送します。

```
~C (local command) cat filename
(2 秒待つ)
```

ファイルが読み込まれる側のシステムの画面に ok プロンプトが現れます。

FCode プログラムを実行するには、次のように入力します。

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

バイナリプログラムを実行するには、次のように入力します。

```
ok go
```

デバッグ

OpenBoot は、逆アセンブラ、レジスタ表示用コマンド、ブレークポイント関連コマンドなどのデバッグツールを提供します。

逆アセンブラの使用方法

組み込み逆アセンブラはメモリーの内容を、対応する SPARC アセンブル言語に翻訳します。

表 6–1 に、メモリーの内容を対応するオペコードに逆アセンブルするコマンドを示します。

表 6–1 逆アセンブラコマンド

コマンド	スタックダイアグラム	説明
`+dis`	(–)	最後に逆アセンブルを終了したところから逆アセンブルを継続します。
`dis`	(addr –)	指定されたアドレスから逆アセンブルを開始します。

`dis` は指定する任意のアドレスから、メモリーの内容の逆アセンブルを開始します。システムは次の場合に中断します。

- 逆アセンブルの進行中に任意のキーを押したとき

- 逆アセンブラの出力が画面一杯になったとき
- call または jump オペコードが現れたとき

中断したら、逆アセンブルを停止することも、+dis コマンドを使用して最後に停止したところから逆アセンブルを継続することもできます。

メモリーアドレスは通常は 16 進で示されますが、シンボルテーブルがある場合は、メモリーアドレスは可能なかぎりシンボルで表示されます。

レジスタの表示

プログラムがクラッシュしたり、ユーザーが Stop-A で中止したり、あるいはブレークポイントに遭遇した結果、プログラムの実行途中でユーザーインタフェースに入ってしまうことがあります。(ブレークポイントについては、124ページの「ブレークポイント」 で説明します。) 上記の場合には、ユーザーインタフェースは自動的にすべての CPU データレジスタの値をバッファー領域に保存します。デバッグの目的のためにそれらの値を調べたり、変更することができます。

表 6–2 に SPARC のレジスタ操作コマンドを示します。

表 6–2 SPARC レジスタコマンド

コマンド	スタックダイアグラム	説明
%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)	浮動小数点ステータスレジスタの値を返します。
%g0 ～ %g7	(– value)	指定されたグローバルレジスタの値を返します。
%i0 ～ %i7	(– value)	指定された入力レジスタの値を返します。
%l0 ～ %l7	(– value)	指定されたローカルレジスタの値を返します。
%o0 ～ %o7	(– value)	指定された出力レジスタの値を返します。
%pc %npc %psr %y %wim %tbr	(– 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。

表 6–2　SPARC レジスタコマンド　続く

コマンド	スタックダイアグラム	説明
.locals	(–)	i、l 、 o レジスタの値を表示します。
.psr	(–)	プログラムステータスレジスタの書式付きで表示します。
.registers .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# –)	w .localsと同じ。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	(new-value –)	%pc に *new-value* を、%npc に (*new-value*+4)をそれぞれ設定します。
to *regname*	(new-value –)	上記のうちの任意のレジスタの格納値を変更します。*new-value* to *regname* の形式で使用してください。
w	(window# –)	現在のウィンドウを、%i*x*、%L*x*、または %o*x* を表示するために設定します。

値の確認や変更が終わったら、go コマンドを使用してプログラムの実行を継続できます。保存したレジスタの値 (変更したものを含めて) は、(コピーして) CPU に戻され、保存されたプログラムカウンタによって指定された位置から実行が再開されます。

to を使用して %pc を変更する場合は、%npc も同時に変更する必要があります。(set-pc の方が両レジスタを自動的に変更するので簡単です。)

w および .window　コマンドについては、ウィンドウ値 0 は通常、現在のウィンドウを指定します。つまり、プログラムが中断されたときのサブルーチンのアクティブウィンドウです。ウィンドウ値が 1 の場合は、そのサブルーチンの呼び出し元のウィンドウであり、2 の場合はその呼び出し元の呼び出し元を指定します。以下同様に、有効なスタックフレーム数までこの関係が繰り返されます。デフォルトの開始値は 0 です。

ブレークポイント

ユーザーインタフェースは、スタンドアロンプログラムの開発とデバッグの支援用として、ブレークポイント機能を備えています。(オペレーティングシステムのもとで実行されるプログラムは、一般的にこの機能は使用しないで、オペレーティングシステムのもとで動作するほかのデバッガを使用します。) ブレークポイント機能では、テストプログラムを停止させたい場所で停止することができます。プログラムの実行が停止した後は、レジスタまたはメモリーを調べたり、変更できるほか、ブレークポイントを新たに設定またはクリアすることができます。プログラムの実行は go コマンドで再開できます。

表 6–3 に、プログラム実行の制御、監視用のブレークポイントコマンドを示します。

表 6–3 ブレークポイントコマンド

コマンド	スタックダイアグラム	説明
+bp=bp	(addr –)	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	(addr –)	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	(–)	最後に設定されたブレークポイントを削除します。
.bp.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。このワードは、実行させたい任意の処理を実行するように変更できます。たとえば、ブレークポイントごとにレジスタを表示するには、['] .registers is .breakpoint と入力します。デフォルト処理は .instruction です。複数の処理を実行させるには、実行させたいすべての処理を呼び出す 1 つの定義を作成し、次にそのワードを .breakpoint に読み込みます。
.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。

表 6–3　ブレークポイントコマンド 続く

コマンド	スタックダイアグラム	説明
.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	(–)	このループの終わりまで実行します。
go	(–)	ブレークポイントから処理を継続します。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 コマンドに似ています。) サブルーチン呼び出しを 1 つの命令として取り扱います。
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	(–)	現在の命令をスキップします (実行しません)。
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	(addr –)	指定されたアドレスが現れるまで実行します。 +bp go と等価です。

ブレークポイントを使用してプログラムをデバッグするには、次の手順に従います。

1. テストプログラムをメモリーアドレス **4000 (16 進)** へ読み込みます。

   詳細は、第 5 章を参照してください。一般的に dload を使用するのが最善の方法です。理由は、プログラムのシンボルテーブルが保存されるためです。

Ethernet を介してプログラムを読み込めない場合は、 `boot -h` も同じ機能を果たします。

`%pc` およびそのほかのすべてのレジスタの値が自動的に初期設定されます。

2. **(省略可能)** ダウンロードされたプログラムを逆アセンブルして、ファイルが正しく読み込まれているかどうかを確認します。

3. `step` コマンドを使用してテストプログラムを **1** 命令ずつ実行します。

さらに、ブレークポイントを設定し、実行したり (たとえば、`4020+bp` および `go`コマンドを実行します)、ほかの方法で実行することもできます。

ソースレベルデバッガ

Forth ソースレベルデバッガでは、Forth プログラムのステップ実行およびトレースが可能です。各実行ステップが 1 つの Forth ワードに対応します。

表 6–4 にこのデバッガのコマンドを示します。

表 6–4 Forth ソースレベルデバッガコマンド

コマンド	説明
`c`	“Continue (継続)”。シングルステップ実行からトレースに切り替え、デバッグ中のワードの実行の残り部分をトレースします。
`d`	“Down a level (1 レベルダウン)”。今表示された名前のワードをデバッグ対象としてマークし、次にそのワードを実行します。
`f`	下位の Forth インタプリタを起動します。そのインタプリタを (`resume`で) 終了させると、 `F` コマンドが実行されたところで制御がデバッガに戻ります。
`q`	“Quit (終了)”。デバッグ中のワードとそのすべての呼び出し元の実行を強制終了させ、制御をコマンドインタプリタに戻します。
`u`	“Up a level (1 レベルアップ)”。デバッグ中のワードからデバッグ対象のマークを取り消します。その呼び出し元をデバッグ対象としてマークし、それまでデバッグされていたワードの実行を終了します。

表 6–4 Forth ソースレベルデバッガコマンド 続く

コマンド	説明
`debug` *name*	指定された Forth ワードをデバッグ対象としてマークします。以降は、*name* を実行しようとするたびに、必ず Forth ソースレベルデバッガを起動します。 `debug` の実行後は、 `debug-off` でデバッグがオフされるまではシステムの実行速度が落ちることがあります。(“.”などの基本 Forth ワードはデバッグしないでください。)
`debug-off`	Forth ソースレベルデバッガをオフにします。以降、ワードのデバッグは行われません。
`resume`	下位インタプリタを終了し、制御をデバッガのシングルステップ実行に戻します (この表の `F` コマンドを参照)。
`stepping`	Forth ソースレベルデバッガを "シングルステップ (実行) モード" に設定し、デバッグ中のワードを 1 ステップずつ対話的に実行できるようにします。シングルステップモードはデフォルトです。
`tracing`	Forth ソースレベルデバッガを "トレースモード" に設定します。このモードは、デバッグ中のワードの実行をトレースし、その間そのワードが呼び出す各ワードの名前とスタックの内容を表示します。
`<space-bar>`	今表示されたワードを実行し、次のワードのデバッグに移ります。

すべての Forth ワードはそれぞれに、「コンポーネント」ワードと呼べる 1 つまたは複数の一連のワードとして定義されています。指定されたワードをデバッグしている間に、デバッガは、そのワードの各「コンポーネント」ワードを実行中にスタックの内容に関する情報を表示します。各コンポーネントワードを実行する直前に、デバッガはスタックの内容と、実行されようとしているコンポーネントワードの名前を表示します。

トレースモードでは、そのコンポーネントワードがそこで実行され、プロセスは次のコンポーネントワードに引き継がれます。

ステップモード (デフォルト) では、ユーザーがデバッガの実行動作を制御します。各コンポーネントワードの実行前に、ユーザーはプロンプトで 表 6–4にある 1 つのキー操作のどれかを求められます。

ftrace の使用方法

ftrace コマンドは、最後の例外割り込み時に実行されていた Forth ワード処理を表示します。次に ftrace の例を示します。

```
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
```

上記の例では、test2 が test1 を呼び出し、test1 は境界に合わないアドレスに値を格納しようとします。その結果、Memory address not aligned という例外が発生します。

ftrace の出力の 1 行目は、例外を発生させた最後のコマンドを示しています。2 行目以降は、その後のコマンドが呼び出されようとしていたメモリーアドレスを示しています。

最後の 13 行は、通常どの ftrace の出力とも同じですが、これは、それが Forth インタプリタが入力ストリームからワードを解釈するときに有効な呼び出し処理であるからです。

付録 **A**

端末エミュレータを使うテスト

テストしようとするシステムの 1 つまたは複数のシリアルポートから、ファイルサーバーとして動作させる第 2 のシステムに接続することができます。このファイルサーバーは、次の条件が満たされれば、システムタイプが同じであってもなくても差し支えありません。

- ファイルサーバーのシリアルポートの性能がテストされるシステムと互換性がある。
- ファイルサーバーに、その出力ボーレートをテストされるシステムのそれと一致するように設定できる端末エミュレータを備えている。

2 つのシステムをこのように接続することにより、ファイルサーバー上の端末エミュレータを、テストしようとしているシステムへの端末として使用できます。(UNIX システムについては、リモートホストへ接続する端末に関する詳細について、オンラインの `tip` マニュアルページを参照してください。Windows システムについては、アクセサリのターミナルの説明を参照してください。Macintosh® システムについては、MacTerminal® のマニュアルを参照してください。)

この端末エミュレーション方式は、起動 ROM を使って作業を行うときに通常のエディタとオペレーティングシステムの機能を使用できるので、(単にダム端末に接続するよりも) お薦めします。

注 - 以降では、「システム」とはテストしようとするシステム、つまりテストされるシステムのことで、「サーバー」とはテストされるシステムに接続するファイルサーバーのことです。

この章に示す手順は、UNIX の tip 端末エミュレータを使用するものとしています。他の端末エミュレータの場合も、手順は似ています。

1. **3** 芯の「ヌルモデム」ケーブル **(**つまり、ピン **3** をピン **2** に、ピン **2** をピン **3** に、ピン **7** をピン **7** に接続するケーブル**)** を使用して、サーバーのシリアルポートをシステムのシリアルポートに接続します。以降の例では、システムのポート **A** とサーバーのポート **B** を使用するものとします。

2. サーバーで tip セッションを設定するため、次のように入力します。

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

注 - サンワークステーションでは、コマンドツールウィンドウでなく、シェルツールウィンドウを使用してください。一部の tip コマンドがコマンドツールウィンドウでは正しく機能しないことがあります。

---

3. システムを起動して、ok プロンプトを表示させます。

   システムにビディオモニタを接続してない場合は、システムの ttya をサーバーの ttyb に接続し、システムに電源を投入します。数秒待ってから、Stop-A を押して電源投入処理を中断し、n を入力して ok プロンプトを表示させます。システムが完全に動作不可でさえなければ、ユーザーインタフェースが使用可能になり、ok プロンプトに対しコマンド入力が可能となり、この手順の次の手順に進むことができます。

4. 標準入出力先を **ttya** にリダイレクトするには、次のように入力します。

```
ok ttya io
```

   画面には応答はありません。

5. サンワークステーションのキーボードで **Return** キーを押します。ok プロンプトが **tip** ウィンドウに表示されます。

   tip ウィンドウに ~# と入力するのは、SPARC システムで Stop-A と入力するのと同じです。

注 - テストされるシステムのサーバーとして使用しているサンワークステーションからは Stop-A を入力しないでください。入力すると、サーバーのオペレーティングシステムが強制終了されてしまいます。(誤って Stop-A と入力した場合は、ただちに > プロンプトで c を入力するか、ok プロンプトで go を入力して、正常状態を回復してください。再表示コマンドにより画面がもとの状態にもどります。)

---

6. tip ウィンドウの使用が終わったら、tip セッションを終了して tip ウィンドウを閉じます。

   a. 必要な場合は、入出力先をそれぞれキーボードと画面にリダイレクトします。

   b. tip ウィンドウで次のように入力します。

```
ok ~.

hostname%
```

注 - tipウィンドウに~ (チルド) コマンドを入力するときは、~ はその行の最初の入力文字でなければなりません。文字の入力位置を確実に新しい行の先頭するには、まず Return キーを押すことです。

---

tip に関する一般的問題

この節では、2.0 より前の Solaris オペレーティング環境で発生する tip の障害の解決方法について説明します。

tip にかかわる障害は、次のような場合に発生することがあります。

- ロックディレクトリがなくなっているか、誤っている。

/usr/spool/uucp という名前のディレクトリが必要です。所有者は uucp で、モードは drwxr-sr-x です。

- ttyb がログイン用に有効になっている。

  /etc/ttytab 内の ttyb (または使用しているシリアルポート) のステータスフィールドを off に設定してなければなりません。このエントリを変更する必要がある場合は、必ず root になって kill -HUP 1を実行してください (init(8) マニュアルページを参照)。

- /dev/ttyb がアクセスできない。

  ときどき、プログラムが /dev/ttyb (使用するシリアルポート) のモードを変更してしまい、アクセスできなくなることがあります。/dev/ttyb のモードが crw-rw-rw- に設定されていることを確認してください。

- シリアルラインがタンデムモードになっている。

  tip 接続がタンデムモードの場合は、オペレーティングシステムはときどき (特に他のウィンドウのプログラムが大量に出力しているときに) XON (^S)文字を送出することがあります。XON 文字は Forth の key? ワードによって検出され、混乱を生じることがあります。この解決方法は、 ~s !tandem tip コマンドでタンデムモードをオフにすることです。

- .cshrcファイルがテキストを生成する。

  tip が、cat を実行するためにサブシェルを開くため、読み込まれたたファイルの先頭にテキストを付け加えてしまいます。dl を使用して予期されない出力を調べる場合は、.cshrc ファイルをチェックしてください。

付録 **B**

起動可能なフロッピーディスクの作成

この付録では、プログラムを起動できるフロッピーディスクの作成手順を説明します。高密度フロッピーディスク (DD でなく HD) を使用してください。手順は次の 2 つです。

- 最初の手順は、133ページの「2.0 より前の Solaris オペレーティング環境の手順」を使用するシステム用です。
- 第 2 の手順は、134ページの「Solaris 2.0 または 2.1 オペレーティング環境の手順」 を使用するシステム用です。

2.0 より前の Solaris オペレーティング環境の手順

2.0 より前のバージョンの Solaris オペレーティングシステムを使用するには、次の手順に従ってください。

1. フロッピーディスクをフォーマットします。

```
hostname# fdformat
```

2. フロッピーディスクのファイルシステムを作成します。

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

3. フロッピーディスクをマウントします。

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

4. 第 2 レベルのディスク起動プログラムをフロッピーディスクにコピーします。

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

5. フロッピーに起動ブロックをインストールします。

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

6. 起動するファイルを /mntにコピーします。

7. フロッピーディスクをアンマウントし、ドライブから取り外します。

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

Solaris 2.0 または 2.1 オペレーティング環境の手順

Solaris 2.0 または 2.1 オペレーティングシステムを使用するには、次の手順に従ってください。

1. フロッピーディスクをフォーマットします。

```
hostname# fdformat
```

2. フロッピーディスクのファイルシステムを作成します。

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

3. フロッピーディスクをマウントします。

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

4. 第 2 レベルのディスク起動プログラムをフロッピーディスクにコピーします。

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

5. フロッピーに起動ブロックをインストールします。

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

6. 起動するファイルを /mntにコピーします。

7. フロッピーディスクをアンマウントし、ドライブから取り外します。

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

付録 **C**

サポートされていないコマンド

OpenBoot ファームウェアの一部の機能は、以前のシステムでは利用できないことがあります。マニュアルに記載されているにもかかわらず、使用しているシステムで利用できないコマンドについては、この付録を参照してください。

表 C–1 サポートされていないコマンドに対する対処方法

コマンド	説明	対処方法
`" 埋め込みバイト`	以前のシステムではサポートされていません。	`alloc-mem` や `c,` など、ほかの配列作成方法を使用します。
`.attributes`	OpenBoot 2.0 までサポートされていません。	読み込み可能な`showdevs` ユーティリティーにこの機能の一部が含まれます。
`alloc-mem`	対処方法を参照。	2.0 より前では、サイズが Forth 辞書の残り合計空間に制限されます。数百バイトを超えて使用すると危険です。代わりに `dma-alloc` (size – virt) を使用します。
`boot-device` `boot-file`	OpenBoot 2.0 までサポートされていません。	`boot-from` を使用して起動デバイスと起動ファイルを指示します。
`cd`	OpenBoot 2.0 までサポートされていません。	読み込み可能な `showdevs` ユーティリティーにこの機能の一部が含まれます。

表 C–1 サポートされていないコマンドに対する対処方法 続く

コマンド	説明	対処方法
コマンド名補完	以前のシステムではサポートされていません。	コマンド名を完全に入力してください。
`cpeek` `cpoke`	以前のシステムではサポートされていません。	以前のシステムには `probe` ワードがあり、 `cprobe` (adr – ok?) と同じ機能を提供します。データ例外の有無は `c@` を使用してテストします。

`d!`
`d?`
`d@`
	以前のシステムではサポートされていません。	32 ビットアクセスを組み合わせて使用します。

`diag-device diag-file`	OpenBoot 2.0 までサポートされていません。	`boot-from-diag` を使用して診断起動デバイスと同起動ファイルを指示します。
`lpeek` `lpoke`	以前のシステムではサポートされていません。	以前のシステムには `probe` ワードがあり、 `lprobe` (adr32 – ok?) と同じ機能を提供します。データ例外の有無は `l@` を使用してテストします。
`ls`	OpenBoot 2.0 までサポートされていません。	読み込み可能な `showdevs` ユーティリティーにこの機能の一部が含まれます。
`NVRAMRC`	OpenBoot 2.0 までサポートされていません。	対処方法はありません。OpenBoot 1.6 にあるバージョンは別のものであり、このバージョンは使用しないでください。
`nvalias` `nvunalias`	OpenBoot 2.6 までサポートされていません。	NVRAMRC を手作業で編集します。
`nodefault-bytes`	OpenBoot 2.0 までサポートされていません。	対処方法はありません。

表 C–1 サポートされていないコマンドに対する対処方法 続く

コマンド	説明	対処方法
`patch`	対処方法を参照。	2.6 より前では、 `patch` で定義内のワードはパッチできますが、数値はできません。数値をパッチするには、 `npatch` *word-to-patch* (new-n old-n –) のように使用します。
`probe-scsi-all`	OpenBoot 2.6 までサポートされていません。	対処方法はありません。
`pwd`	OpenBoot 2.0 までサポートされていません。	読み込み可能な `showdevs` ユーティリティーにこの機能の一部が含まれます。
`show-devs`	OpenBoot 2.0 までサポートされていません。	読み込み可能な `showdevs` ユーティリティーにこの機能の一部が含まれます。
`show-sbus`	OpenBoot 2.3 までサポートされていません。	次のワードを使用します。 `ok` **`cd /sbus`** `ok` **`ls`** (同様な情報が表示されますが、書式が異なります。)
`showstack`	OpenBoot 2.6 までは (オフに) 切り替えてはなりません。	showstack をオフにするには、システムをリセットするか、 `[`ｲ`] noop is status`と入力します。
`spaced?`	OpenBoot 2.6 までサポートされていません。	`spaced@` と“.”を使用してください。

`Stop-F`
`Stop-D`
`Stop-N`
	OpenBoot 2.0 までサポートされていません。	対処方法はありません。

表 C–1 サポートされていないコマンドに対する対処方法 続く

コマンド	説明	対処方法
`test` *xxx*	OpenBoot 2.0 までサポートされていません。	OpenBoot 1.*x* システムでは、`test-memory` (–)(`test /memory` と同じ)を使用して特定のデバイスをテストできます。 一部の差し込み式デバイスも、正しいテスト名を直接入力してテストできます (OpenBoot 1.*x* に限られます)。
ユーザが追加した デバイス別名	OpenBoot 2.0 までサポートされていません。	対処方法はありません。
`watch-net`	OpenBoot 1.3 ～ 2.2 ではサポートされていません。	対処方法はありません。
`wpeek` `wpoke`	以前のシステムではサポートされていません。	以前のシステムには `probe` ワードがあり、 `wprobe` (adr16 – ok?) と同じ機能を提供します。データ例外の有無は `w@` を使用してテストします。

付録 **D**

障害追跡ガイド

この付録では、システムが正常に起動できない場合、いくつかの一般的な障害とそれらを軽減する方法について説明します。

電源投入時の初期設定処理

システム電源投入時の初期設定メッセージについてよく理解してください。これらのメッセージは、システム起動時のさまざまな段階でシステムが実行する機能を示すため、問題をより正確に判断できます。さらに、POST から OpenBoot ファームウェア、起動プログラム、カーネルへの制御の移動も示します。

次の例では、SPARCstation 10 システムでの OpenBoot 初期設定処理を示します。バナーの前のメッセージは、`diag-switch?` 変数が真の場合だけ ttya に表示されます。

注 - 表示されるカーネルメッセージは、使用するオペレーティングシステムのバージョンによって変わることがあります。

```
ttya initialized
     (ここで POST は実行を終了し、OpenBootファームウェアに制御を移す。)
Cpu #0 TI,TMS390Z50
     (CPU モジュールをプローブする)
Cpu #1 Nothing there
Cpu #2 Nothing there
Cpu #3 Nothing there
```

(続く)

```
Probing Memory Bank #0 16 Megabytes of DRAM
     (メモリーをプローブする)
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
     (デバイスをプローブする前に、use-nvramrc? が真なら
      NVRAMRC コマンドを実行し、Stop コマンドの
      キーボード LED フラッシュをチェックする。
Probing /iommu@f,e0000000/sbus@f,e0001000 at f,0
     (デバイスをプローブする)
       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
     (バナーを表示する)
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/
     (ファームウェアが起動プログラムで tftp を実行する。)
 sd@3,0 File and args:
     (このメッセージが表示されたあと、 制御が 起動プログラムへ移される。)
root on /iommu@f,e0000000/sbus@f,e0001000/espdma@
     (起動プログラムが実行を開始する。)
 f,400000/esp@f,800000/sd@3,0:a fstype 4.2

Boot: vmunix
Size: 1425408+436752+176288 bytes
     (このメッセージが表示されたあと、制御が カーネルへ移される。)
Viking/NE: PAC ENABLED          (カーネルが実行を開始する。)
  ...     (カーネルメッセージが続く。)
```

緊急の手順

表 D–1 に、一部の障害の解決に有効なコマンドについて説明します。これらのコマンドのどれを実行するときも、システムに電源を投入した直後に対応のキーを、キーボードの LED が点灯するまで押し続けてください。

表 D–1 緊急キーボードコマンド

コマンド	説明
`Stop`	POST を省略します。このコマンドはセキュリティ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設定に戻します。

注 - これらのコマンドは、PROM セキュリティーがオンの場合は使用不可になります。また、システムが `full` セキュリティーを有効にしている場合も、`ok` プロンプトを表示できるパスワードがなければ、上記のコマンドはどれも使えません。

システムクラッシュ後のデータの保存

`sync` コマンドは、処理中のどのような情報でもただちに強制的にハードディスクに書き出します。これは、オペレーティングシステムがクラッシュしたり、すべてのデータを保存できないうちに中断されてしまった場合に効果があります。

`sync` は実際には制御をオペレーティングシステムに戻し、データ保存処理が行われます。ディスクデータが書き込まれると、オペレーティングシステムは自身の

コアイメージの保存を開始します。このコアダンプが必要でない場合は、Stop-A キー処理でこの保存処理を中断できます。

一般的な障害

この節では、一部の一般的障害とそれらの障害の解決方法について説明します。

画面がブランクになる - 出力を表示できない

障害 : システムの画面がブランクになり、出力をまったく表示しない。

この問題の考えられる原因を次に示します。

- ハードウェアに障害がある。

  システムのマニュアルを参照してください。

- キーボードが接続されていない。

  キーボードを接続してない場合は、出力は代わりに TTYA に送られます。この問題を解決するには、システムの電源を落とし、キーボードを接続し、再び電源を投入します。

- モニターに電源が入らないか、接続されていない。

  モニターの電源ケーブルを点検してください。モニターケーブルがシステムフレームバッファーに接続されていることを確認します。その後で、モニターに電源を入れます。

- output-device が TTYA または TTYB に設定されている。

  これは、NVRAM 変数 output-device が、screen にでなく、ttya または ttyb に設定されているということです。次のいずれかの処置を行います。

- システムの電源を落とします。次に再び電源を投入し、ただちに Stop-N を押します。これで、すべての NVRAM 変数がそれぞれのデフォルト値に設定されます。その結果、output-device 変数が screenに設定されます。以前のほかのデフォルト以外の設定もすべてデフォルト値に戻されるので注意が必要です。それらは必要に応じて復元する必要があります。

- 端末を TTYA に接続し、システムをリセットします。端末に ok プロンプトが表示されたら、出力をフレームバッファーに送るように **screen output** と入力し

ます。必要な場合は、setenv を使用してデフォルトのディスプレイデバイスを変更してください。

- システムに複数のフレームバッファーがある。

  システムに複数の追加型フレームバッファーがある場合、または組み込みフレームバッファーが 1 つと、追加型が 1 つまたはそれ以上ある場合は、誤ったフレームバッファーがコンソルデバイスとして使用されることもあり得ます。148ページの「コンソールを特定のモニターに設定する」 を参照してください。

システムが誤ったデバイスから起動される

障害 : システムが、ディスクから起動されることになっているのに、ネットワークから起動される。

この問題の考えられる原因は次の 2 つです。

- NVRAM 変数 diag-switch? が誤って true に設定されている。

  Stop-Aを使用して起動処理を中断してください。ok プロンプトで次のコマンドを入力します。

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

  システムはこれでディスクから起動を開始します。

- NVRAM 変数 boot-device が disk でなく net に設定されている。

  Stop-A を使用して起動処理を中断してください。ok プロンプトで次のコマンドを入力します。

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

  上記のコマンドは、デバイス別名リストに disk (ターゲット 3)として定義されているディスクからシステムを起動させるので注意してください。disk1 (ターゲット 1)、disk2 (ターゲット 2)、または disk3 (ターゲット 3)から起動する場合は、boot-device を適切に設定します。

障害 : システムがネットワークからでなくディスクから起動する。

- boot-device が net に設定されていない。

  Stop-Aを使用して起動処理を中断してください。ok プロンプトで次のコマンドを入力します。

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

障害 : システムが誤ったディスクから起動する。(たとえば、システムにディスクが複数あって、システムを disk2 から起動したいのに、disk1 から起動する。)

- boot-device が正しいディスクに設定されていない。

  Stop-Aを使用して起動処理を中断してください。ok プロンプトで次のコマンドを入力します。

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

システムが Ethernet から起動しない

障害 : システムがネットワークから起動しない。

この問題の考えられる原因は次の 2 つです。

- NIS マップが古くなっている。

  システム管理者に報告してください。

- Ethernet ケーブルが接続されていない。

  Ethernet ケーブルを接続してください。システムは起動処理を開始します。

- サーバーが応答せず、「no carrier」メッセージが表示される。

  システム管理者に報告してください。

- tpe-link-test が使用不可になっている。

  システムマニュアルのトラブルシューティングに関する説明を参照してください。(注 : より対線 Ethernet がないシステムには tpe-link-test 変数がないので注意してください。)

  (test net で Ethernet の動作を確認することができます)

システムがディスクから起動しない

障害 : ディスクからシステムを起動しようとすると、失敗し、次のようなメッセージが表示される。

```
The file just loaded does not appear to be executable.
```

- 起動ブロックがなくなっているか、壊れている。

  新しい起動ブロックをインストールしてください。

障害 : ディスクからシステムを起動しようとすると、失敗し、次のようなメッセージが表示される。

```
Can't open boot device.
```

- (特に外部ディスクの場合) ディスクの電源が落ちていることがある。

  ディスクに電源を投入し、ディスクとシステムにSCSI ケーブルが接続されていることを確認してください。

SCSI の問題

障害 : システムにディスクが複数インストールされていて、SCSI 関係のエラーメッセージが表示される。

- SCSI ターゲット番号設定が重複している可能性があります。

次の手順を行ってみてください。

1. **1** つだけ残して、すべてのディスクの接続を外します。
2. ok プロンプトで次のように入力します。

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

   ターゲット番号とその対応ユニット番号を書き留めてください。
3. 別のディスクを接続して手順 **b** をもう一度実行します。
4. エラーが発生したら、そのディスクのターゲット番号を使用されていない別のターゲット番号に変更します。
5. すべてのディスクが再び接続されるまで、手順 **b**、**c**、**d** を繰り返します。

コンソールを特定のモニターに設定する

障害 : システムに複数のモニターが接続されていて、コンソールが意図するモニターに設定されていない。

- システムに複数のモニターが接続されている場合は、OpenBoot ファームウェアはつねにコンソールを NVRAM 変数 output-device によって指定されるフレームバッファーに設定します。output-device のデフォルト値は screen であり、これは、ファームウェアがシステム内で最初に見つけるフレームバッファーの別名です。

  このデフォルトを変更する一般的な方法は、たとえば次のように、output-device を該当するフレームバッファーに変更することです。

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

  コンソールを特定のモニターに設定するもう 1 つの方法は、NVRAM 変数 sbus-probe-list を変更することです。

```
ok show sbus-probe-list      (現在値とデフォルト値を表示)
sbus-probe-list f0123 f0123      (システムの SBus スロット数はシステムにより異なる。)
ok
```

  コンソールとして選定するフレームバッファーがスロット 2 にある場合は、最初にスロット 2 をプローブするように sbus-probe-list を変更します。

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

付録 **E**

Forth ワードリファレンス

この付録には、OpenBoot ファームウェアがサポートする Forth のコマンドを一覧表で示します。

大部分のコマンドは、各章での説明順に並んでいます。ただし一部の表では、このマニュアルには記載されていないコマンドを示しています。これらの追加コマンド (メモリーマップまたは出力表示用の基本式、マシン固有のレジスタ操作用コマンド) も、Forth の OpenBoot 実装のワードセットの一部です。したがって、これらのコマンドはそれぞれ該当するグループのコマンドと一緒に示してあります。

表 E–1 スタック項目の表記

表記	説明
|	代替スタック結果。たとえば、 (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。
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。

表 E–1 スタック項目の表記 続く

表記	説明
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 bxxx	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 = false。そのほかのすべての値 = true (通常 -1)。
long Lxxx	32 ビット値。
n n1 n2 n3	通常の符号付きの値 (32 ビット)。
+n u	符号なしの正の値 (32 ビット)。
n[64] (n.low n.hi)	拡張精度 (64 ビット) 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 wxxx	16 ビット値 (32 ビットワードの下位 2 バイト)。

表 E–2 制限付きモニターコマンド

コマンド	説明
`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 `boot` を入力するのと同じ)。
`c`	停止されているプログラムの実行を再開します(`ok` プロンプトで `go` を入力するのと同じ)。
`n`	Forth モニタ-に入ります。

表 E–3 デバイス別名の確認と作成

コマンド	説明
`devalias`	現在のすべてのデバイス別名を表示します。
`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。

表 E–4 デバイスツリー表示コマンド

コマンド	説明
`.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捜し、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。

表 E–4 デバイスツリー表示コマンド 続く

コマンド	説明
`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。

表 E–5 ヘルプコマンド

コマンド	説明
`help`	ヘルプの主なカテゴリを表示します。
`help` *category*	カテゴリ内の全コマンドのヘルプを表示します。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)。

表 E–6 boot コマンドの一般的オプション

変数	説明
boot [*device-specifier] [filename] [options]*	
[*device-specifier*]	起動デバイス名 (フルパス名または別名)。例を示します。 cdrom (CD-ROM ドライブ) disk (ハードディスク) floppy (3.5 インチフロッピーディスクドライブ) 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 NVRAM 変数の値 (第 3 章参照) 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 - プログラムを読み込んだ後、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。)

表 E–7 診断テストコマンド

コマンド	説明
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、指定したデバイスツリー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* を指定しないと、ルートノードが使用されます。)

表 E–7 診断テストコマンド 続く

コマンド	説明
`test` *device-specifier*	指定したデバイスのセルフテスト方法を実行します。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 /memory` : NVRAM 変数 `selftest-#megs` で指定される M バイト数をテストします。または `diag-switch?` が `true` の場合は全メモリをテストします。 `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 (組み込みセルフテスト方法を備える) 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。)
`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。

表 E–8 システム情報表示コマンド

コマンド	説明
`banner`	電源投入時のバナーを表示します。
`show-sbus`	インストールされ、プローブされる SBus デバイスのリストを表示します。
`.enet-addr`	現在の Ethernet アドレスを表示します。
`.idprom`	ID PROM の内容を書式付きで表示します。
`.traps`	SPARC のトラップタイプのリストを表示します。
`.version`	起動 PROM のバージョンと日付を表示します。

表 **E–9** NVRAM システム変数

変数名	設定値	説明
`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`	空白文字	起動するファイル (空白の場合、第 2 ブーター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、診断プログラムを実行します。
`fcode-debug?`	false	true の場合、差し込み式デバイス FCode の名前フィールドを取り入れます。
`hardware-revision`	デフォルトなし	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`	デフォルトなし	カスタムキーボードのキーマップ。
`last-hardware-update`	デフォルトなし	システム更新情報。

表 E–9 NVRAM システム変数 続く

変数名	設定値	説明
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セルフテストを繰り返します。
nvramrc	空白文字	NVRAMRC の内容。
oem-banner	空白文字	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	デフォルトなし	バイト配列カスタム OEM ロゴ (oem-logo? が true で使用可能になります)。 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 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 screenttya、または ttyb)。
sbus-probe-list	0123	プローブされる SBus スロット、それらのスロットがプローブされる順番。
screen-#columns	80	画面上のカラム数 (文字数/行)。
screen-#rows	34	画面上の行数。
scsi-initiator-id	7	ホストアダプタの SCSI バスアドレス。範囲は 0 - 7。

表 E–9 NVRAM システム変数 続く

変数名	設定値	説明
sd-targets	31204567	SCSI ディスクユニットを割り当てます (1.*x* のみ)。
security-#badlogins	デフォルトなし	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	デフォルトなし	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)。これを直接設定してはなりません。
selftest-#megs	1	テストするメモリーの M バイト数。 diag-switch? が true の場合は無視。
skip-vme-loopback?	false	true の場合、POST は VMEbus のループバックテストを行いません。
st-targets	45670123	SCSI テープユニットを割り当てます (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	1 バイトのスクラッチフィールド。読み取り/書き込みテストに使用されます。
tpe-link-test?	true	組み込みより対線 Ethernet 向け 10baseT リンクテストを有効にします。
ttya-mode	9600,8,n,1,-	ttya (ボーレート、データビット数、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、データビット数、パリティー、ストップビット数、ハンドシェーク)。

表 E–9 NVRAM システム変数 続く

変数名	設定値	説明
`ttya-ignore-cd`	true	true の場合、オペレーティングシステムは ttya のキャリア検出を無視します。
`ttyb-ignore-cd`	true	true の場合、オペレーティングシステムは ttyb のキャリア検出を無視します。
`ttya-rts-dtr-off`	false	true の場合、オペレーティングシステムは ttya の DTR、RTS を有効にしません。
`ttyb-rts-dtr-off`	false	true の場合、オペレーティングシステムは ttyb の DTR、RTS を有効にしません。
`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。
`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*で起動します。
`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 E–10 システム変数の表示/変更

コマンド	説明
`printenv`	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 進で示されます。) `printenv` *parameter* は指定する変数の現在値を表示します。
`setenv` *parameter value*	*parameter* を 10 進またはテキスト値 value 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` *parameter*	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。

表 E–11 システム変数用コマンド基本式

コマンド	スタックダイアグラム	説明
`nodefault-bytes` *parameter*	(len –)Usage: (– adr len)	(カスタム) NVRAM 変数を作成します。このコマンドは変数を永続的にするために NVRAMRC で使用します。
変数	(– ???)	(現在の) フィールド値を返します (データ型は変数によります)。
`show` *parameter*	(–)	(現在の) フィールド値を表示します (数値は 10 進表示)。

表 E–12 NVRAMRC エディタコマンド

コマンド	説明
`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。
`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。捨てる前に、確認を求めます。
`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、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。
`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。

表 E–13 nvedit キー操作コマンド

キー操作	説明
Control-B	1 文字位置戻ります。
Control-C	エディタを終了し、OpenBook コマンドインタプリタに戻ります。一時バッファーは保存されていますが、NVRAMRC には戻されません。(後で nvstore を使用して一時バッファーを NVRAMRC に書いて戻してください。)
Control-F	文字位置進みます。
Control-K	行の終わりでは、現在行に次の行をつなぎます (つまり、この行を 1 つにします)。
Control-L	すべての行を表示します。
Control-N	NVRAMRC 編集バッファーの次の行に進みます。
Control-O	カーソル位置に new line を挿入し、現在行にとどまっています。
Control-P	NVRAMRC 編集バッファーの前の行に戻ります。
Delete	前の 1 文字を削除します。
Return	カーソル位置に改行を挿入し、次の行に進みます。

表 E–14 スタック操作コマンド

コマンド	スタックダイアグラム	説明
-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。(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。

表 E–14 スタック操作コマンド 続く

コマンド	スタックダイアグラム	説明
`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 スタック項目を複製します。
`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。
`2rot`	(n1 n2 n3 n4 n5 n6 – n3 n4 n5 n6 n1 n2)	3 対のスタック項目を回転します。
`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。
`3drop`	(n1 n2 n3 –)	3 スタック項目を複製します。
`3dup`	(n1 n2 n3 – n1 n2 n3 n1 n2 n3)	3 スタック項目を複製します。
`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コピーします。
`pick`	(??? +n – ??? n2)	+n 番目のスタック項目をコピーします (1 `pick` = `over`)。

表 E–14 スタック操作コマンド 続く

コマンド	スタックダイアグラム	説明
`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 箇のスタック項目を回転します。 (2 `roll` = `rot`)。
`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。

表 E–15 コロン定義ワード

コマンド	スタックダイアグラム	説明
: *name*	(–)	新しいワード定義の作成を開始します。
;	(–)	新しいワード定義の作成を終了します。

表 E–16 演算機能

コマンド	スタックダイアグラム	説明
*	(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。
/mod	(n1 n2 – rem quot)	n1 / n2の剰余と商。
<<	(n1 +n – n2)	n1 を +n ビット左シフトします。
>>	(n1 +n – n2)	n1 を +n ビット右シフトします。
>>a	(n1 +n – n2)	n1 を +n ビット算術右シフトします。
*/	(n1 n2 n3 – n4)	n1 * n2 / n3。
*/mod	(n1 n2 n3 – rem quot)	n1 * n2 / n3の剰余と商。
1+	(n1 – n2)	1 を足します。
1-	(n1 – n2)	1 を引きます。
2*	(n1 – n2)	2 を掛けます。
2+	(n1 – n2)	2 を足します。
2-	(n1 – n2)	2 を引きます。
2/	(n1 – n2)	2 で割ります。
abs	(n – u)	絶対値。
aligned	(n1 – n2)	n1 を次の 4 の整数倍に切り上げます。

表 E–16 演算機能 続く

コマンド	スタックダイアグラム	説明
`and`	(n1 n2 – n3)	ビット単位の論理積。
`bounds`	(startadr len – endadr startadr)	`startadr len` を `do` ループ用に`endadrstartadr` 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。
`flip`	(word1 – word2)	16 ビットワード内の 2 バイトをスワップします。
`lbsplit`	(long – b.low b2 b3 b.hi)	32 ビットロングワードを 4 バイトに分割します。
`lwsplit`	(long – w.low w.hi)	32 ビットロングワードを 2 つの 16 ビットワードに分割します。
`max`	(n1 n2 – n3)	`n1` と`n2` の大きいほうの値を `n3` とします。
`min`	(n1 n2 – n3)	`n1` と `n2` の小さいほうの値を `n3` とします
`mod`	(n1 n2 – rem)	`n1 / n2` の剰余を計算します。
`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。
`u*x`	(u1 u2 – product[64])	2 つの符号なし 32 ビット数値の乗算を行い、符号なしの 64 ビットの積を生じます。

表 E–16 演算機能 続く

コマンド	スタックダイアグラム	説明
u/mod	(u1 u2 – un.rem un.quot)	2 つの符号なし 32 ビット数値の除算を行い、32 ビットの剰余と商を生じます。
u2/	(u1 – u2)	1 ビット論理右シフトし、空になった符号ビットにゼロをシフトします。
wbsplit	(word – b.low b.hi)	16 ビットワードを 2 バイトに分割します。
wflip	(long1 – long2)	32 ビットロングワードの半分をスワップします。
wljoin	(w.low w.hi – long)	2 ワードを結合してロングワードを作ります。
x+	(n1[64] n2[64] – n3[64])	2 つの 64 ビット数値の加算を行います。
x-	(n1[64] n2[64] – n3[64])	2 つの 64 ビット数値の減算を行います。
xor	(n1 n2 – n3)	ビット単位の排他的論理和。
xu/mod	(u1[64] u2 – rem quot)	符号なしの 64 ビット数値を符号なしの 32 ビット数値で割り、32 ビットの剰余と商を生じます。

表 E–17 変換演算子

コマンド	スタックダイアグラム	説明
/c	(– n)	1 バイトのバイト数 = 1。
/c*	(n1 – n2)	n1 に/cを掛けます。
ca+	(adr1 index – adr2)	adr1 を index の /c 倍増分します。

表 E–17 変換演算子　続く

コマンド	スタックダイアグラム	説明
ca1+	(adr1 – adr2)	adr1 を /c だけ増分します。
/L	(– n)	ロングワードのバイト数 = 4。
/L*	(n1 – n2)	n1 に /L を掛けます。
La+	(adr1 index – adr2)	adr1 を index の /L 倍増分します。
La1+	(adr1 – adr2)	adr1 を /L 増分します。
/n	(– n)	通常のバイト数 - 4。
/n*	(n1 – n2)	n1 に /n を掛けます。
na+	(adr1 index – adr2)	adr1 を index の /n 倍増分します
na1+	(adr1 – adr2)	adr1 を /n だけ増分します。
/w	(– n)	16 ビットワードのバイト数 = 2。
/w*	(n1 – n2)	n1 に /w を掛けます。
wa+	(adr1 index – adr2)	adr1 をindex の /w 倍増分します
wa1+	(adr1 – adr2)	adr1 を /w だけ増分します。

表 E–18 メモリーアクセスコマンド

コマンド	スタックダイアグラム	説明
`!`	(n adr16 –)	32 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。
`+!`	(n adr16 –)	`adr16` に格納されている 32 ビット数値に `n` を加算します。16 ビット境界でなければなりません。
`<w@`	(adr16 – n)	符号付き 16 ビットワードを `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。
`?`	(adr16 –)	`adr16` の 32 ビット数値を表示します。16 ビット境界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 ビット境界でなければなりません。
`2!`	(n1 n2 adr16 –)	2 数値を`adr16` (`n2` を下位アドレス)に格納します。16 ビット境界でなければなりません。
`2@`	(adr16 – n1 n2)	2 数値を `adr16` (`n2` を下位アドレス)から取り出します。16 ビット境界でなければなりません。
`blank`	(adr u –)	メモリーの `u` バイトを 空白文字 (10 進の 32) に設定します。
`c!`	(n adr –)	`n` の下位バイトを `adr` に格納します。
`c?`	(adr –)	`adr` の 1 バイトを表示します。
`c@`	(adr – byte)	1 バイトを `adr` から取り出します。
`cmove`	(adr1 adr2 u –)	`adr1` から `adr2` に、下位バイトから先に `u` バイトをコピーします。

表 E–18 メモリーアクセスコマンド 続く

コマンド	スタックダイアグラム	説明
cmove>	(adr1 adr2 u –)	adr1 から adr2 に、上位バイトから先に u バイトをコピーします。
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。
cpoke	(byte adr – okay?)	1 バイトを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。
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が配列 #1 側より小さい場合 n = 1、それ以外の場合は n = -1 になります。
d!	(n1 n2 adr64 –)	2 つの 32 ビット数値を adr64 に格納します。64 ビット境界でなければなりません。順序は実装により異なります。
d?	(adr64 –)	adr64の 2 つの 32 ビット数値を表示します。64 ビット境界でなければなりません。順序は実装により異なります。
d@	(adr64– n1 n2)	2 つの 32 ビット数値をadr64 から取り出します。64 ビット境界でなければなりません。順序は実装により異なります。
dump	(adr len –)	adr から始まる len メモリーバイトを表示します。
erase	(adr u –)	u メモリーバイトを 0 に設定します。
fill	(adr size byte –)	size メモリーバイトを byte に設定します。

表 E–18 メモリーアクセスコマンド 続く

コマンド	スタックダイアグラム	説明
`L!`	(n adr32 –)	32 ビット数値を `adr32` に格納します。32 ビット境界でなければなりません。
`L?`	(adr32 –)	`adr32` の 32 ビット数値を表示します。32 ビット境界でなければなりません。
`L@`	(adr32 – long)	32 ビット数値を `adr32` から取り出します。32 ビット境界でなければなりません。
`lflips`	(adr len –)	指定された領域の 32 ビットロングワード内の 2 つの 16 ビットワードを入れ替えます。
`lpeek`	(adr32 – false long true)	32 ビットの数を `adr32` 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`	(long adr32 – okay?)	32 ビットの数を `adr32` 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。
`move`	(adr1 adr2 u –)	`adr1` から `adr2` に `u` バイトをコピーし、オーバラップを正しく処理します。
`off`	(adr16 –)	`false` (32 ビットの 0) を `adr16` に格納します。
`on`	(adr16 –)	`true` (32 ビットの -1) を `adr16` に格納します。
`unaligned-L!`	(long adr –)	32 ビット数値を格納します。境界は任意です。
`unaligned-L@`	(adr – long)	32 ビット数値を取り出します。境界は任意です。

表 E–18 メモリーアクセスコマンド 続く

コマンド	スタックダイアグラム	説明
`unaligned-w!`	(word adr –)	16 ビット数値を格納します。境界は任意です。
`unaligned-w@`	(adr – word)	16 ビット数値を取り出します。境界は任意です。
`w!`	(n adr16 –)	16 ビット数値を `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 –)	`adr16` の 16 ビット数値を表示します。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。
`wflips`	(adr len –)	指定された領域の16 ビットワード内の 2 バイトを入れ替えます。
`wpeek`	(adr16 – false word true)	16 ビットの数を `adr16` 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。
`wpoke`	(word adr16 – okay?)	16 ビット数量値を `adr16` 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。

表 E–19 メモリー割り当てコマンド

コマンド	スタックダイアグラム	説明
`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` により割り当てを解除します。
`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` 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)	物理アドレス領域を割り当て、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)	デバイスアドレス空間を割り当て対象として指定します。
`obmem`	(– space)	オンボードのメモリーアドレス空間を割り当て対象として指定します。
`sbus`	(– space)	SBus アドレス空間を割り当て対象として指定します。

表 E–20 メモリー割り当て用基本式

コマンド	スタックダイアグラム	説明
`cacheable`	(space – cache-space)	以降のアドレス割り当てをキャッシュ可能にするようにアドレス空間を変更します。
`iomap?`	(virt –)	仮想アドレスの IOMMU ページ割り当てエントリを表示します。左のスタックダイアグラムは Sun-4m マシン用です。

表 E–20 メモリー割り当て用基本式 続く

コマンド	スタックダイアグラム	説明
`iomap-page`	(phys space virt –)	`phys` と`space` によって指定される物理ページを仮想アドレスに割り当てます。左のスタックダイアグラムは Sun-4m マシン用です。
`iomap-pages`	(phys space virt size –)	`iomap-page`を連続して実行して`size` によって指定されるメモリー領域を割り当てます。左のスタックダイアグラムは Sun-4m マシン用です。
`iopgmap@`	(virt – pte 0)	仮想アドレスの IOMMU ページ割り当てエントリを返します。左のスタックダイアグラムは Sun-4m マシン用です。
`iopgmap!`	(pte virt –)	仮想アドレスの新しいページ割り当てエントリを格納します。左のスタックダイアグラムは Sun-4m マシン用です。
`map-page`	(phys space virt –)	アドレス `phys` から始まる 1 メモリーページを指定されたアドレス空間内の仮想アドレス `virt` に割り当てます。アドレスはすべてページ境界に揃うように、切り捨てが行われます。
`map-pages`	(phys space virt size –)	map-page を連続して実行してメモリー領域を指定されたサイズに割り当てます。
`map-region`	(region# virt –)	1 つの領域を割り当てます。
`map-regions`	(region# virt size –)	連続領域を割り当てます。
`map-segments`	(smentry virt len –)	`smap!` を連続して実行してメモリー領域を割り当てます。
`pgmap!`	(pmentry virt –)	仮想アドレスの新しいページ割り当てエントリを格納します。

表 E–20 メモリー割り当て用基本式 続く

コマンド	スタックダイアグラム	説明
`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)	仮想アドレスのページ割り当てエントリを返します。
`pagesize`	(– size)	ページのサイズを返します。通常は 4K (16 進の 1000)。
`rmap!`	(rmentry virt –)	仮想アドレスの新しい領域割り当てエントリを格納します。
`rmap@`	(virt – rmentry)	仮想アドレスの領域割り当てエントリを返します。
`segmentsize`	(– size)	セグメントのサイズを返します。通常 256K (16 進の 4000)
`smap!`	(smentry virt –)	仮想アドレスの新しいセグメント割り当てエントリを格納します。
`smap?`	(virt –)	仮想アドレスのセグメント割り当てエントリを書式付きで表示します。
`smap@`	(virt – smentry)	仮想アドレスのセグメント割り当てエントリを返します。

表 E–21 キャッシュ操作コマンド

コマンド	スタックダイアグラム	説明
`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。

表 E–21 キャッシュ操作コマンド 続く

コマンド	スタックダイアグラム	説明
`cdata!`	(data offset –)	32ビットのデータをキャッシュオフセットに格納します。
`cdata@`	(offset – data)	データをキャッシュオフセットから取り出し (返し) ます。
`ctag!`	(value offset –)	タグ値をキャッシュオフセットに格納します。
`ctag@`	(offset – value)	キャッシュオフセットのタグ値を返します。
`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。

表 E–22 Sun-4D マシンのマシンレジスタ読み取り/書き込み

コマンド	スタックダイアグラム	説明
SuperSPARC™ モジュールレジスタアクセス用		
`cxr!`	(data –)	MMU コンテキストレジスタに書き込みます。
`mcr!`	(data –)	モジュール制御レジスタに書き込みます。
`cxr@`	(– data)	MMU コンテキストレジスタから読み取ります。
`mcr@`	(– data)	MMU 制御レジスタから読み取ります。
`sfsr@`	(– data)	同期障害ステータスレジスタから読み取ります。
`sfar@`	(– data)	同期障害アドレスレジスタから読み取ります。

表 E–22 Sun-4D マシンのマシンレジスタ読み取り/書き込み 続く

コマンド	スタックダイアグラム	説明
`afsr@`	(– data)	非同期障害アドレスレジスタから読み取ります。
`afar@`	(– data)	非同期障害アドレスレジスタから読み取ります。
`.mcr`	(–)	モジュール制御レジスタを表示します。
`.sfsr`	(–)	同期障害ステータスレジスタを表示します。
MXCC 割り込みレジスタアクセス用		
`interrupt-enable!`	(data –)	割り込みマスクレジスタに書き込みます。
`interrupt-enable@`	(– data)	割り込みマスクレジスタから読み取ります。
`interrupt-pending@`	(– data)	割り込み保留レジスタから読み取ります。
`interrupt-clear!`	(data –)	割り込みクリアレジスタに書き込みます。
BootBus レジスタアクセス用		
`control!`	(datat –)	BootBus 制御レジスタに書き込みます。
`control@`	(– datat)	BootBus 制御レジスタから読み取ります。
`status1@`	(– datat)	BootBus status1 レジスタから読み取ります。
`status2@`	(– datat)	BootBus status2 レジスタから読み取ります。

表 **E–23** Sun-4M マシンのマシンレジスタ読み取り/書き込み

コマンド	スタックダイアグラム	説明
`.mcr`	(–)	モジュール制御レジスタを表示します。
`.mfsr`	(–)	メモリーコントローラ障害ステータスレジスタを表示します。
`.sfsr`	(–)	同期障害ステータスレジスタを表示します。
`.sipr`	(–)	システム割り込み保留レジスタを表示します。
`aux!`	(data –)	補助レジスタに書き込みます。
`aux@`	(– data)	補助レジスタから読み取ります。
`cxr!`	(data –)	MMU コンテキストレジスタに書き込みます。
`cxr@`	(– data)	MMU コンテキストレジスタから読み取ります。
`interrupt-enable!`	(data –)	システム割り込みターゲットマスクレジスタに書き込みます。
`interrupt-enable@`	(– data)	システム割り込みターゲットマスクレジスタから読み取ります。
`iommu-ctl!`	(data –)	IOMMU 制御レジスタに書き込みます。
`iommu-ctl@`	(– data)	IOMMU 制御レジスタから読み取ります。
`mcr!`	(data –)	モジュール制御レジスタに書き込みます。
`mcr@`	(– data)	モジュール制御レジスタから読み取ります。
`mfsr!`	(data –)	メモリーコントローラ障害ステータスレジスタに書き込みます。
`mfsr@`	(– data)	メモリーコントローラ障害ステータスレジスタから読み取ります。

表 **E–23** Sun-4M マシンのマシンレジスタ読み取り/書き込み 続く

コマンド	スタックダイアグラム	説明
msafar@	(– data)	MBus-to-SBus 非同期障害アドレスレジスタの内容を読み取ります。
msafsr!	(data –)	MBus-to-SBus 非同期障害ステータスレジスタに書き込みます。
msafsr@	(– data)	MBus-to-SBus 非同期障害ステータスレジスタの内容を読み取ります。
sfsr!	(data –)	同期障害ステータスレジスタに書き込みます。
sfsr@	(– data)	同期障害ステータスレジスタの内容を読み取ります。
sfar!	(data –)	同期障害アドレスレジスタに書き込みます。
sfar@	(– data)	同期障害アドレスレジスタの内容を読み取ります。

表 **E–24** Sun-4C マシンのマシンレジスタ読み取り/書き込み

コマンド	スタックダイアグラム	説明
aerr!	(data –)	非同期エラーレジスタに書き込みます。
aerr@	(– data)	非同期エラーレジスタから読み取ります。
averr!	(data –)	非同期エラー仮想アドレスレジスタに書き込みます。
averr@	(– data)	非同期エラー仮想アドレスレジスタから読み取ります。
aux!	(data –)	補助レジスタに書き込みます。
aux@	(– data)	補助レジスタの内容を表示します。

表 **E–24** Sun-4C マシンのマシンレジスタ読み取り/書き込み 続く

コマンド	スタックダイアグラム	説明
`context!`	(data –)	コンテキストレジスタに書き込みます。
`context@`	(– data)	コンテキストレジスタ (MMU コンテキスト) から読み取ります。
`dcontext@`	(– data)	コンテキストレジスタ (キャッシュコンテキスト) から読み取ります。
`enable!`	(data –)	システム有効化レジスタに書き込みます。
`enable@`	(– data)	システム有効化レジスタから読み取ります。
`interrupt-enable!`	(data –)	割り込み許可レジスタに書き込みます。
`interrupt-enable@`	(– data)	割り込み許可レジスタから読み取ります。
`serr!`	(data –)	同期エラーレジスタに書き込みます。
`serr@`	(– data)	同期エラーレジスタから読み取ります。
`sverr!`	(data –)	同期エラー仮想アドレスレジスタに書き込みます。
`sverr@`	(– data)	同期エラー仮想アドレスレジスタから読み取ります。

表 **E–25** 代替アドレス空間アクセスコマンド

コマンド	スタックダイアグラム	説明
`spacec!`	(byte adr asi –)	1 バイトを asi とアドレスに格納します。
`spacec?`	(adr asi –)	asi とアドレスの 1 バイトを表示します。
`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 32 ビット数値を asi とアドレスに格納します。 数値の順序は実装によります。
`spaced?`	(adr asi –)	asi とアドレスの 2 つの 32 ビット数値を表示します。 数値の順序は実装によります。
`spaced@`	(adr asi – n1 n2)	2 つの 32 ビット数値を asi とアドレスから取り出します。 数値の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。
`spaceL?`	(adr asi –)	asi とアドレスの 32 ビットロングワードを表示します。
`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 –)	asi とアドレスの 16 ビットワードを表示します。
`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。

表 E–26 ワード定義

コマンド	スタックダイアグラム	説明
`:` *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。
`alias` *new-name old-name*	(–)Usage: (??? – ?)	*old-name* と同じ操作をする *new-name*を作成します。
`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 (たとえば、`3 constant bar`)を定義します。
`2constant` *name*	(n1 n2 –)Usage: (– n1 n2)	2 数値の定数を定義します。
`create` *name*	(–)Usage: (– adr16)	汎用定義ワード
`defer` *name*	(–)Usage: (??? – ?)	フォワードリファレンス、またはコードフィールドアドレスを使用する実行ベクトルのワードを定義します。
`does>`	(– adr16)	ワード定義の実行節を開始します。
`field` *name*	(offset size – offset+size)Usage: (adr – adr+offset)	指定されたオフセットポインタを作成します。
`struct`	(– 0)	`field` の作成に備えて初期化します。

表 E–26 ワード定義 続く

コマンド	スタックダイアグラム	説明
`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。

表 E–27 辞書検索コマンド

コマンド	スタックダイアグラム	説明
' *name*	(– acf)	ワードを辞書から検索します。コードフィールドアドレスを返します。定義外で使用してください。
['] *name*	(– acf)	定義内、外のどちらでも使用できる点以外は、 'と同じです。
`.calls`	(acf –)	コンパイルアドレスが `acf` である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。見つからなかった場合 n = 0 、ワードが即値の場合 n = 1 それ以外の場合は n = -1 になります。
`find`	(pstr – pstr false acf n)	辞書からワードを検索します。検索するワードは `pstr` で示されます。見つからなかった場合 n = 0、ワードが即値の場合 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)`	(acf –)	コードフィールドアドレスによって示されるワードを逆コンパイルします。

表 E–27 辞書検索コマンド 続く

コマンド	スタックダイアグラム	説明
`sift`	(pstr –)	`pstr` によって示される文字列を含むすべての辞書エントリの名前を表示します。
`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`	(–)	辞書内のすべての表示可能なワードを表示します。

表 E–28 辞書コンパイルコマンド

コマンド	スタックダイアグラム	説明
`,`	(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。
`[`	(–)	解釈を開始します。
`]`	(–)	解釈を終了し、コンパイルを開始します。
`allot`	(n –)	辞書に `n` バイトを割り当てます。

表 **E–28** 辞書コンパイルコマンド 続く

コマンド	スタックダイアグラム	説明
`>body`	(acf – apf)	変数フィールドアドレスからコンパイルアドレスを見つけます。
`body>`	(apf – acf)	変数フィールドアドレスからコンパイルアドレスを見つけます。
`compile`	(–)	次のワードを実行時にコンパイルします。
`[compile]` *name*	(–)	次の (即値) ワードをコンパイルします。
`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。
`immediate`	(–)	最後の定義を即値としてマークします。
`is` name	(n –)	`defer` ワードまたは `value` に新しい処理を実装します。
`literal`	(n –)	数値をコンパイルします。
`origin`	(– adr)	Forth システムの開始アドレスを返します。
`patch` *new-word old-word word-to-patch*	(–)	*old-word* を *word-to-patch* の *new-word* に置き換えます。

表 E–28 辞書コンパイルコマンド 続く

コマンド	スタックダイアグラム	説明
(patch	(new-n old-n acf –)	old-n を acf によって示されるワードの new-n に置き換えます。
recursive	(–)	辞書内のコンパイル中のコロン定義を表示可能にし、したがって、そのワードの名前をそれ自身の定義内で再帰的に使用可能にします。
state	(– adr)	コンパイル状態でゼロ以外の変数。

表 E–29 アセンブル言語のプログラミング

コマンド	スタックダイアグラム	説明
code *name*	(–) Usage: (??? – ?)	*name* と呼ばれるアセンブル言語ルーチンの作成を開始します。code の後のコマンドはアセンブラニーモニックとして解釈されます。 アセンブラがインストールされていなくても、 "," を使用してマシンコードを数値 (たとえば、16 進) で入力しなければならなくなる点は別として、 code は依然存在することに注意してください。
c;	(–)	アセンブル言語ルーチンの作成を終了します。自動的に Forth インタプリタの next 機能をアセンブルし、その結果それが実行されたとき、next から作成されたアセンブルコードワードが制御を通常どおり呼び出し元に戻すようにします。

表 E–29 アセンブル言語のプログラミング 続く

コマンド	スタックダイアグラム	説明
`label` *name*	(–) Usage: (– adr16)	*name* と呼ばれるアセンブル言語ルーチンの作成を開始します。`label` で作成されたワードは、実行されたとき、そのコードのアドレスをスタックに残します。`label`の後のコマンドはアセンブラニーモニックとして解釈されます。`code` の場合と同様に、label はアセンブラがインストールされていなくても存在します。
`end-code`	(–)	`label`で開始されたアセンブル言語のパッチを終了します。

表 E–30 基本数値表示

コマンド	スタックダイアグラム	説明
`.`	(n –)	数値を現在の基数で表示します。
`.r`	(n size –)	数値を固定幅フィールドで表示します。
`.s`	(–)	データスタックの内容を表示します。
`showstack`	(–)	自動的に各 `ok` プロンプトの前で `.s` を実行します。
`u.`	(u –)	符号なしの数値を表示します。
`u.r`	(u size –)	符号なしの数値を固定幅フィールドで表示します。

表 E–31 基数の変更

コマンド	スタックダイアグラム	説明
`base`	(– adr)	基数を格納している変数。
`binary`	(–)	基数を 2 に設定します。

表 E–31 基数の変更 続く

コマンド	スタックダイアグラム	説明
`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。

表 E–32 数値出力ワード用基本式

コマンド	スタックダイアグラム	説明
`#`	(+L1 – +L2)	数字をピクチャ数値出力に変換します。
`#>`	(L – adr +n)	ピクチャ数値出力を終了します。
`<#`	(–)	ピクチャ数値出力を初期化します。
`(.)`	(n –)	数値を文字列に変換します。
`(u.)`	(– adr len)	符号なし数値を文字列に変換します。
`digit`	(char base – digit true char false)	文字を数字に変換します。
`hold`	(char –)	ピクチャ数値出力文字列内に char を挿入します。
`$number`	(adr len – true n false)	文字列を数値に変換します。

表 E–32 数値出力ワード用基本式 続く

コマンド	スタックダイアグラム	説明
#s	(L – 0)	#s の後の数字をピクチャ数値出力に変換します。
sign	(n –)	ピクチャ出力の符号を設定します。

表 E–33 テキスト入力制御

コマンド	スタックダイアグラム	説明
(*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。
expect	(adr +n –)	割り当てられた入力デバイスのキーボードから編集済みの入力を得、adr に格納します。
key	(– char)	割り当てられた入力デバイスのキーボードから 1 文字を読みます。
key?	(– flag)	入力デバイスのキーボードでキーが押された場合 true。
span	(– adr16)	expect が読む文字数を格納している変数。
word	(char – pstr)	入力文字列から char で区切られている文字列をまとめ、メモリー位置 pstr に入れます。

表 E–34 テキスト出力表示

コマンド	スタックダイアグラム	説明
." *ccc*"	(–)	後の表示に備えて、文字列をコンパイルします。
(cr	(–)	出力カーソルを現在行の先頭に戻します。
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。
exit?	(– flag)	スクロール制御プロンプト More [<space>,<cr>,q] ? を有効にします。 復帰フラグは、ユーザーが出力を終了する場合 true です。
space	(–)	空白文字 を表示します。
spaces	(+n –)	+n 箇の空白文字を表示します。
type	(adr +n –)	n 箇の文字を表示します。

表 E–35 書式付き出力

コマンド	スタックダイアグラム	説明
#line	(– adr16)	出力デバイス上の行番号を保持する変数。
#out	(– adr16)	出力デバイス上のカラム番号を保持する変数。

表 E–36 テキスト文字列の操作

コマンド	スタックダイアグラム	説明
",	(adr len –)	adr から始まり、長さが len バイトである配列をパックされた文字列としてコンパイルし、辞書の一番上に入れます。
" *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*)	(–)	文字列を即時に表示します。
-trailing	(adr +n1 – adr +n2)	後続空白文字を削除します。
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。
lcc	(char – lowercase-char)	文字を小文字に変換します。
left-parse-string	(adr len char – adrR lenR adrL lenL)	文字列を指定された区切り文字で分割します (区切り文字は捨てられます)。
pack	(adr len pstr – pstr)	adr len からパックされた文字列を作り、メモリー位置 pstr に入れます。

表 E–36 テキスト文字列の操作 続く

コマンド	スタックダイアグラム	説明
`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。
`upc`	(char – uppercase-char)	文字を大文字に変換します。

表 E–37 入出力先の変更コマンド

コマンド	スタックダイアグラム	説明
`input`	(device –)	以降の入力に使用されるデバイス (`ttya`、`ttyb`、`keyboard`、または *"device-specifier "*)を選択します。
`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 (`ttya`、`ttyb`、`screen`、または *"device-specifier"*)を選択します。

表 E–38 ASCII 定数

コマンド	スタックダイアグラム	説明
`bell`	(– n)	ベル文字の ASCII コード。10 進の 7。
`bs`	(– n)	バックスペース文字の ASCII コード。10 進の 8。

表 **E–39** 行エディタコマンド

コマンド	機能
`Control-A`	行の始めに戻ります。
`Control-B`	1 文字位置戻ります。
`Control-D`	現在位置の文字を消去します。
`Control-E`	行の終わりに進みます。
`Control-F`	1 文字位置進みます。
`Control-H`	1 つ前の文字を消去します (Delete または Backspace キーと同じです)。
`Control-K`	現在位置から行の終わりまで、消去します。
`Control-L`	コマンド履歴リストを表示します。
`Control-N`	1 行後のコマンド行を呼び出します。
`Control-P`	1 行前のコマンド行を呼び出します。
`Control-Q`	(制御文字を入力するために) 次の制御文字をそのまま入力可能にします。
`Control-R`	行を再表示します。
`Control-U`	1 行全体を消去します。
`Control-W`	1 つ前の語を消去します。
`Control-Y`	カーソルの前に保存バッファーの内容を挿入します。
`Control-space`	現在のコマンドを補完します。
`Control-/`	すべての一致/コマンド補完の候補を表示します。
`Control-?`	すべての一致/コマンド補完の候補を表示します。

表 E–39 行エディタコマンド 続く

コマンド	機能
Control-}	すべての一致/コマンド補完の候補を表示します。
Esc-B	1 語戻ります。
Esc-D	語の現在位置から終わりまで消去します。
Esc-F	1 語進みます。
Esc-H	語の現在位置からはじめまで消去します (Control-Wと同じです)。

表 E–40 比較コマンド

コマンド	スタックダイアグラム	説明
<	(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。
0<	(n – flag)	n < 0 の場合 true。
0<=	(n – flag)	n <= 0 の場合 true。
0<>	(n – flag)	n <> 0 の場合 true。

表 E–40 比較コマンド　続く

コマンド	スタックダイアグラム	説明
0=	(n – flag)	n = 0 の場合 true (さらにフラグを反転します)。
0>	(n – flag)	n > 0 の場合 true。
0>=	(n – flag)	n >= 0 の場合 true。
between	(n min max – flag)	min <= n <= max の場合 true。
false	(– 0)	FALSE(偽) の値 = 0。
true	(– -1)	TRUE(真) の値 = -1。
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。
u>=	(u1 u2 – flag)	u1 >= u2 の場合 true、符号なし。
within	(n min max – flag)	min <= n < maxの場合 true。

表 E–41 if...then...else コマンド

コマンド	スタックダイアグラム	説明
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。

表 E–42 case 文コマンド

コマンド	スタックダイアグラム	説明
case	(selector – selector)	case...endcase 条件付き構造を開始します。
endcase	(selector {empty} –)	case...endcase 条件付き構造を終了します。
endof	(–)	case...endcase 条件付き構造内のof...endof 句を終了します。
of	(selector test-value – selector {empty})	case 条件付き構造内の of...endof 句を開始します。

表 E–43 begin (条件付き) ループコマンド

コマンド	スタックダイアグラム	説明
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 **E–44** do(カウント付き) ループコマンド

コマンド	スタックダイアグラム	説明
`+loop`	(n –)	`do...+loop` 構造を終了します。ループインデックスに n 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。
`?leave`	(flag –)	`flag` がゼロ以外の場合になった場合、`do...loop` から抜け出させます。
`do`	(end start –)	`do...loop` を開始します。インデックスは `start` から `end` 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` から抜け出させます。
`loop`	(–)	`do...loop` の終わり。

表 **E–45** プログラム実行制御コマンド

コマンド	スタックダイアグラム	説明
`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。

表 E–45 プログラム実行制御コマンド 続く

コマンド	スタックダイアグラム	説明
execute	(acf –)	コードフィールドアドレス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	(–)	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。

表 E–46 ファイル読み込みコマンド

コマンド	スタックダイアグラム	説明
?go	(–)	Forth、FCode、またはバイナリプログラムを実行します。
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。

表 E–46 ファイル読み込みコマンド 続く

コマンド	スタックダイアグラム	説明
`eval`	(adr len –)	読み込まれた Forth テキストファイルを解釈します。
`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` *device-specifier argument*	(–)	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。
`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。

表 E–47 逆アセンブラコマンド

コマンド	スタックダイアグラム	説明
`+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。

表 **E–48** SPARC レジスタコマンド

コマンド	スタックダイアグラム	説明
`%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。
`.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` データ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# –)	`w .locals` と同じ。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リターンスタックを表示します。

表 E–48 SPARC レジスタコマンド 続く

コマンド	スタックダイアグラム	説明
`set-pc`	(value –)	`%pc %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# –)	現在のウィンドウを、`%i`*x*、`%L`*x*、または `%o`*x* 表示向けに設定します。

表 E–49 ブレークポイントコマンド

コマンド	スタックダイアグラム	説明
`+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。 このワードは、実行させたい任意の処理を実行するように変更できます。たとえば、ブレークポイントごとにレジスタを表示するには `['] .registers is .breakpoint` と入力します。デフォルト処理は `.instruction` です。複数の処理を実行させるには、実行させたいすべての処理を呼び出す 1 つの定義を作成し、次にそのワードを `.breakpoint` に読み込みます。
`.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。

表 E–49 ブレークポイントコマンド 続く

コマンド	スタックダイアグラム	説明
.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)。
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。

表 E–50 Forth ソースレベルデバッガコマンド

コマンド	説明
`C`	"Continue (継続)"。シングルステップ実行から追跡に切り替え、デバッグ中のワードの実行の残り部分を追跡します。
`D`	"Down a level (1 レベルダウン)"。今表示された名前のワードをデバッグ対象として指定し、次にそのワードを実行します。
`F`	下位の Forth インタプリタを起動します。そのインタプリタを (`resume`で) 終了させると、 `F` コマンドが実行されたところで制御がデバッガに戻ります。
`Q`	"Quit (終了)"。デバッグ中のワードとそのすべての呼び出し元の実行を強制終了させ、制御をコマンドインタプリタに戻します。
`U`	"Up a level (1 レベルアップ)"。デバッグ中のワードからデバッグ対象の指定を取り消します。その呼び出し元をデバッグ対象として指定し、それまでデバッグされていたワードの実行を終了します。
`debug` *name*	指定された Forth ワードをデバッグ対象として指定します。以降は、name を実行しようとするたびに、必ず Forth ソースレベルデバッガを起動します。`debug` の実行後は、 `debug-off` でデバッグがオフされるまではシステムの実行速度が落ちることがあります。 (. などの基本 Forth ワードはデバッグしないでください。)
`debug-off`	Forth ソースレベルデバッガをオフにします。以降、ワードのデバッグは行われません。
`resume`	下位インタプリタを終了し、制御をデバッガのシングルステップ実行に戻します (この表の `F` コマンドを参照)。
`stepping`	Forth ソースレベルデバッガを " シングルステップ (実行) モード " に設定し、デバッグ中のワードを 1 ステップずつ対話的に実行できるようにします。シングルステップモードはデフォルトです。
`tracing`	Forth ソースレベルデバッガを追跡モードに設定します。このモードは、デバッグ中のワードの実行をトレースし、その間そのワードが呼び出す各ワードの名前とスタックの内容を表示します。
`Space`	今表示されたワードを実行し、次のワードのデバッグに移ります。

表 E–51 時間ユーティリティー

コマンド	スタックダイアグラム	説明
`get-msecs`	(– ms)	現在の ミリ秒 (ms) 単位の概略時刻を返します。
`ms`	(n –)	n ミリ秒 (ms) 遅延させます。分解能は 1 ミリ秒 (ms) です。

表 E–52 その他の処理

コマンド	スタックダイアグラム	説明
`callback string`	(value –)	指定された値と文字列を使用して SunOS™ を呼び出します。
`catch`	(??? acf – ? error-code)	acf を実行し、 `throw` が呼び出されない場合は `throw` エラーコードまたは 0 を返します。
`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、 0x00020001 = ファームウェアバージョン 2.1)を返します。
`forth`	(–)	Forth の主要ワードを検索順の一番上に復元します。
`ftrace`	(–)	例外発生時の呼び出し順序を表示します。
`noop`	(–)	何もしません。
`old-mode`	(–)	制限付きモニターにします。

表 E–52 その他の処理 続く

コマンド	スタックダイアグラム	説明
`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)。
`ramforth`	(–)	Forth の辞書を RAM にコピーします。(システムによっては解釈の速度を上げ、システムワードのパッチを有効にします。)
`romforth`	(–)	`ramforth` をオフにします。
`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。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。
`throw`	(error-code –)	与えられたエラーコードを `catch` に返します。

表 E–53 マルチプロセッサコマンド

コマンド	スタックダイアグラム	説明
`module-info`	(–)	すべての CPU モジュールのタイプと速度を表示します。
`switch-cpu`	(cpu# –)	指定された CPU に切り替えます。(マルチプロセッサの場合有効)

表 E–54 緊急キーボードコマンド

コマンド	説明
`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`	NVRAMC の内容をデフォルト設定に戻します。

索引

記号

数字

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U