OpenBoot™ 3.x コマンド・リファレンスマニュアル

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

Part No. 806-2928-10
2000 年 2 月
Revision A

原典	OpenBoot 3.x Command Reference Manual Part No: 806-1377-10 Revision A

目次

はじめに　xi

対象読者　xi

お読みになる前に　xii

本書の構成　xii

関連マニュアル　xiii

書体と記号について　xiv

1. 概要　1

OpenBoot の特長　1

差し込み式デバイスのドライバ　1

FCode インタプリタ　2

デバイスツリー　2

プログラマブルユーザーインタフェース　2

ユーザーインタフェース　2

デバイスツリー　3

デバイスパス名デバイスのパス名、アドレス、引数　4

デバイス別名　6

デバイスツリーの表示　7

ヘルプの表示　11

OpenBoot コマンドの使用上の注意　12

2. システムの起動とテスト　13

システムの起動　13

一般ユーザー向けの起動　15

上級ユーザー向けの起動　15

診断の実行　19

SCSI バスのテスト　20

取り付けられているデバイスのテスト　20

フロッピーディスクドライブのテスト　21

メモリーのテスト　21

クロックのテスト　22

ネットワークコントローラのテスト　22

ネットワークの監視　23

システム情報の表示　23

システムのリセット　24

3. システム変数の設定　25

変数設定の表示と変更　27

セキュリティー変数の設定　29

コマンドセキュリティー　30

フルセキュリティー　31

電源投入時バナーの変更　32

入出力の制御　34

入出力デバイスオプションの選択　34

シリアルポート特性の設定　35

起動オプションの選択　36

電源投入時自己診断テストの制御 (POST)　36

nvramrc の使用方法　37
スクリプトの内容の編集　39
スクリプトファイルの起動　41

4. Forth ツールの使用方法　43

Forth コマンド　43
データ型　45
数値の用法　45
スタック　46
スタックの内容の表示　47
スタックダイアグラム　48
スタックの操作　51
利用者定義の作成　52
演算機能の使用方法　54
単精度整数演算　54
倍精度数演算　56
データ型変換　56
アドレス演算　57
メモリーのアクセス　59
仮想メモリー　59
デバイスレジスタ　63
ワード定義の使用方法　64
辞書の検索　67
データを辞書へコンパイルする　70
数値の表示　71
基数の変更　72
テキスト入出力の制御　73
入出力先の変更　76

コマンド行エディタ　78

条件フラグ　81

制御コマンド　82

if-else-then 構造　82

case 文　84

begin ループ　85

do ループ　86

その他の制御コマンド　89

5. プログラムの読み込みと実行　91

boot の使用方法　93

dl を使ってシリアルポート A から Forth テキストファイルを読み込む　94

load の使用方法　95

dlbin を使って FCode またはバイナリ実行ファイルをシリアルポート A から読み込む　96

dload を使って Ethernet から読み込む　98

Forth プログラム　98

FCode プログラム　99

実行可能バイナリ　99

?go の使用方法　100

6. デバッグ　101

Forth 言語逆コンパイラの使用方法　101

逆アセンブラの使用方法　103

レジスタの表示　103

SPARC レジスタ　104

ブレークポイント　106

Forth ソースレベルデバッガ　107

patch と (patch) の使用方法　109

ftrace の使用方法　112

A. TIP 接続の設定　113

TIP に関する一般的問題　116

B. 起動可能なフロッピーディスクの作成　117

C. 障害追跡ガイド　119

電源投入時の初期設定処理　119

緊急時の手順　120

システムクラッシュ後のデータの保存　121

一般的な障害　122

画面がブランクになる — 出力を表示できない　122

システムが誤ったデバイスから起動される　123

システムが Ethernet から起動しない　124

システムがディスクから起動しない　124

SCSI の問題　125

コンソールを特定のモニターに設定する　126

D. Sun Ultra 5/10 UPA/PCI システム　127

PCI ベースのシステム　127

PCI バスの pcia と pcib　130

E. Sun Ultra 30 UPA/PCI システム　131

PCI ベースのシステム　131

汎用的な名前　134

PCI バスの pcia と pcib　135

F. Sun Ultra 60 UPA/PCI システム 137

PCI ベースのシステム 137

汎用的な名前 140

PCI バスの pcia と pcib 141

G. Sun Ultra 250 UPA/PCI システム 143

Banner コマンドの出力 143

汎用的な名前 143

内蔵 SCSI バス 145

PCI デバイスの .properties 147

.speed コマンド 148

PCI バススロットのプローブ 148

SCSI プローブコマンド 149

H. Sun Ultra 450 UPA/PCI システム 151

Banner コマンドの出力 151

汎用的な名前 151

内蔵 SCSI バス 153

PCI デバイスの .properties 155

.speed コマンド 156

PCI バススロットのプローブ 156

SCSI プローブコマンド 158

I. Forth ワードリファレンス 159

スタック項目の表記 160

デバイスツリー表示コマンド 162

boot コマンドの一般的オプション 163

システム情報表示コマンド 163

システム変数表示/変更用コマンド　164
NVRAMRC エディタコマンド　164
nvedit キー操作コマンド　165
スタック操作コマンド　166
単精度演算機能　168
ビット操作論理演算子　169
倍精度数演算機能　170
32 ビットデータ型変換機能　170
64 ビットデータ型変換機能　171
変換演算子　172
64 ビットアドレス演算機能　173
メモリーアクセスコマンド　173
64 ビットメモリーアクセス機能　175
メモリーマップコマンド　176
ワード定義　177
辞書検索コマンド　178
辞書コンパイルコマンド　179
アセンブリ言語のプログラミング　180
基数値表示　181
基数の変更　181
数値出力ワード用基本式　182
テキスト入力制御　183
テキスト出力表示　184
書式付き出力　184
テキスト文字列の操作　185
入出力先リダイレクトコマンド　186
ASCII 定数　186
コマンド行エディタ用キー操作コマンド　187

コマンド補完キー操作コマンド　188

比較コマンド　188

if-else-then コマンド　189

case 文コマンド　189

begin (条件付き) ループコマンド　190

do (カウント付き) ループコマンド　190

プログラム実行制御コマンド　191

ファイル読み取りコマンド　192

逆コンパイラコマンド　193

ブレークポイントコマンド　193

Forth ソースレベルデバッガコマンド　194

時間ユーティリティー　196

その他の処理　196

マルチプロセッサコマンド　197

メモリー割り当てコマンド　197

メモリー割り当て用基本式　198

キャッシュ操作コマンド　199

Sun-4u マシンの
マシンレジスタ読み取り/書き込み　200

代替アドレス空間アクセスコマンド　200

SPARC レジスタコマンド　201

SPARC V9 レジスタコマンド　202

緊急キーボードコマンド　203

はじめに

『OpenBoot 3.x コマンド・リファレンスマニュアル』では、IEEE Standard 1275-1994『 Standard For Boot Firmware』に準拠するファームウェアを実装した Sun™ のシステムの使用方法について説明します。

本書では、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変数の変更
- プログラムの読み込みと実行
- トラブルシューティング

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、本書ではさらに OpenBoot の Forth インタプリタのコマンドについても説明します。

対象読者

本書は、OpenBoot を使用して SBus および PCI ベースのシステムの構成やデバッグを実施しようとするシステム設計者からシステム管理者、およびエンドユーザーにいたるまで、すべてのユーザーを対象としています。

お読みになる前に

本書に記載されている情報は、バージョン 3.x の OpenBoot を使用するシステムを前提としています。OpenBoot の実装によっては、プロンプトや書式が異なる場合があり、本書で説明するツールと機能の一部をサポートしていない場合もあります。

本書の構成

第 1 章「概要」では、OpenBoot のユーザーインタフェース、およびその他の主要な機能について説明します。

第 2 章「システムの起動とテスト」では、OpenBoot を使用する最も一般的な作業を説明します。

第 3 章「システム変数の設定」では、NVRAM 変数を使ってどのようにシステム管理作業を行うかについて、詳細に説明します。

第 4 章「Forth ツールの使用方法」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「プログラムの読み込みと実行」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「デバッグ」では、逆コンパイラ、Forth のソースレベルデバッガ、ブレークポイントを含む OpenBoot のデバッグ機能について説明します。

付録 A「TIP 接続の設定」では、シリアルポートを使用してシステムをサンの別のシステムに接続する方法について説明します。

付録 B「起動可能なフロッピーディスクの作成」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「障害追跡ガイド」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 D「Sun Ultra 5/10 UPA/PCI システム」では、Sun Ultra 5/10 システムの PCI 関連の情報を示します。

付録 E「Sun Ultra 30 UPA/PCI システム」では、Sun Ultra 30 システムの PCI 関連の情報を示します。

付録 F「Sun Ultra 60 UPA/PCI システム」では、Sun Ultra 60 システムの PCI 関連の情報を示します。

付録 G「Sun Ultra 250 UPA/PCI システム」では、Sun Ultra 250 システムの PCI 関連の情報を示します。

付録 H「Sun Ultra 450 UPA/PCI システム」では、Sun Ultra 450 システムの PCI 関連の情報を示します。

付録 I「Forth ワードリファレンス」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

本書と関連するマニュアルは、次のとおりです。

- 『OpenBoot 3.x の手引き』

OpenBoot FCode の詳細については、次を参照してください。

- 『Writing FCode 2.x Programs』
- 『Writing FCode 3.x Programs』

オープンファームウェアの詳細については、次を参照してください。

IEEE Standard 1275-1994 Standard for Boot (Initialization, Configuration) Firmware, Core Requirements and Practices (IEEE Order Number SH17327. 1-800-678-4333.) `http://playground.sun.com/1275` も参照してください。

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

- 「Forth: The New Model」, Jack Woehr. M & T Books, 1992.
- Forth Interest Group (1-510-89-FORTH)

http://forth.org/fig.html

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 またはゴシック	コマンド行の可変部分。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。 rm ファイル名 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。
ok	OpenBootコマンドプロンプト	ok
%	UNIX C シェルのプロント。	system%

表 P-1　このマニュアルで使用している書体と記号(続き)

字体または記号	意味	例
`$`	UNIX の Bourne シェルと Korn シェルのプロンプト。	`system$`
`#`	スーパーユーザーのプロンプト (シェルの種類を問わない)。	`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、 IEEE Standard 1275-1994,『Standard for Boot Firmware』の定義に従う OpenBoot の概要を示します。OpenBoot ファームウェアは、システムの電源を入れるとただちに実行されます。OpenBoot ファームウェアの主な機能は次のとおりです。

- システムハードウェアをテストし初期化します。
- ハードウェア構成を調べます。
- 大容量記憶装置、またはネットワークのいずれかからオペレーティングシステムを起動します。
- ハードウェアとソフトウェアのテスト用対話式デバッグ機能を提供します。

OpenBoot の特長

OpenBoot アーキテクチャーは、これまでの固有のシステムと比較して、機能と移植性が格段に拡張されています。このアーキテクチャーは、サン・マイクロシステムズ社が SPARC システムの OpenBoot としてはじめて実装したものですが、その設計はプロセッサに依存しません。以下に OpenBoot ファームウェアの特長をいくつか挙げます。

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

差し込み式ドライバは、*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 FCode 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

デバイスノードの特性デバイスツリーは、システムに接続されている (常時取り付けられている差し込み式の) デバイスを記述する、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

ユーザーインタフェースは、対話型コマンドインタプリタをベースにしており、これを介してハードウェアおよびソフトウェア開発、障害の切り分け、デバッグ用の一連の広範な機能を利用できます。さまざまなレベルのユーザーがこれらの機能を利用できます。

ユーザーインタフェースのプロンプトは実装に依存します。

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。

- 「Stop-A」を押します。
- 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ない場合は、システムはこのユーザーインタフェースで停止します。

自動起動を設定している場合でも、ディスプレイコンソールに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する前に、キーボードから「Stop-A」を押すことにより、システムをユーザーインタフェースで停止させることができます。

- 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。)

オペレーティングシステムから OpenBoot に入った後のコマンドの使用方法については 12 ページの「OpenBoot コマンドの使用上の注意」を参照してください。

デバイスツリー

デバイスは、相互に接続されたバスを介してホストコンピュータに接続されています。OpenBoot 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。ホストコンピュータの物理的なアドレスバスを表すノードが、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のものがあります。

- 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
- データ - 上記の方法で使用する私用データの初期値
- 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 - デバイスツリー内のある 1 つの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、それらにつなが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 (デバイスが取り付けら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。デバイスの識別に物理アドレスを使用すると、他のデバイスをシステムに取り付けたり、システムから削除したときに、デバイスアドレスの変更を避けることができます。

子ノードのないノードのことをリーフノードといい、一般にはデバイスを表します。システム提供のファームウェアサービスを表す場合もあります。

デバイスパス名デバイスのパス名、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@1f,0/SUNW,fas@e,8800000/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。各ノード名は次の書式になっています。

driver-name@*unit-address*:*device-arguments*

表 1-1 でこれらの変数を説明します。

表 1-1　デバイスパス名の変数

変数名	説明
driver-name	1～31文字の英字、数字、および記号「, . _ + -」からなる可読文字で、なんらかのニーモニック値をもつ文字列。大文字と小文字の区別があります。場合によっては、この名前にはデバイスのメーカの名前とデバイスの型名がコンマで区切って入っていることがあります。一般には、メーカの名前としては株式略称 (たとえば SUNW,sd) が英文大文字で使用されます。組み込みデバイスの場合は、メーカの名前は通常省略されます (たとえば sbus)。
@	*address* 変数の前に入れます。
unit-address	その親のアドレス空間内のデバイスの物理アドレスを表すテキスト文字列。テキストの書式はバスによって異なります。
:	*arguments* 変数の前に入れなければなりません。
device-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 *unit-address* 部分はその親の物理アドレス空間内の 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。次の例を見てください。

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

- 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、1f,0 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 のカード上のオフセット 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。
- ディスクデバイスは 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 @*unit-address* または *driver-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。一致するノードが複数存在すると、ファームウェアはその中から 1 つ選択します (ユーザーが希望するものと異なることもあります)。

たとえば、`/sbus/esp@0,40000/sd@3,0` の例では、そのシステムにはメインシステムバス上の SBus が 1 つあるものとし、`sbus` を `sbus@1f,0` 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` デバイスが複数存在する可能性もあります。システム上に複数あると、`esp` だけを使用したのではどの esp 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こともあります。

もう 1 つの例として、`/sbus/@2,1/sd@3,0` は通常指定できるのに対して、`/sbus/scsi@2,1/@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と SCSI テープデバイスドライバの両方に 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 `:`*device-arguments* 部分も省略できます。同じ例を示します。

```
/sbus@1f,0/scsi@2,1/sd@3,0:a
```

ディスクデバイスの引数は文字列 `a` です。このデバイス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。

実装によっては、パス名構成要素の省略が可能です。あいまいさが生じなければ省略しても、その実装では目的とするデバイスが選択されます。たとえば、上の例のシステムに `sd` デバイスが 1 つしかなかった場合は、`/sd:a` これにより、前述した省略しない場合と同じデバイスが識別されます。

デバイス別名

デバイスの別名、または単に別名とは、「デバイスパス」の短縮表現のことをいいます。

たとえば、`disk` という別名で完全なデバイスパス名を表すことができます。

```
/sbus@1f,0/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、デバイスパス名を全部入力する必要はほとんどありません。

表 1-2 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 1-2　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 *aliasdevice-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。永久的な別名を作成するには、devalias コマンドを *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 および nvunalias コマンドを使用します。(詳細は、第 3 章「システム変数の設定」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、Solaris™ ディレクトリツリーで現在のディレクトリを変更、表示する Solaris 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 1-3 に示すコマンドを使用して調べます。

表 1-3　デバイスツリー表示コマンド

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

表 1-3　デバイスツリー表示コマンド (続き)

コマンド	説明
device-end	デバイスツリーが選択されない状態にします。
" device-path" find-device	デバイスノードを選択します。dev ノードと似ています。
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。
see *wordname*	指定するワードを逆コンパイルします。
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。
" *device-path*" select-dev	指定されたデバイスノードを選択し、それを有効なノードにします。

.properties は現在のノードのすべての特性の名前と値を表示します。

```
ok dev /zs@1,f0000000
ok .properties
address                 ffee9000
port-b-ignore-cd
port-a-ignore-cd
keyboard
device_type             serial
slave                   00000001
intr                    0000000c  00000000
interrupts              0000000c
reg                     00000001  f0000000  00000008
name                    zs
ok
```

dev は、指定されたノードの内容を表示できるように、現在のノードをそのノードに設定します。たとえば、ACME 社の「ACME,widget」という名前の SBus デバイスを現在のノードにするには、次のように入力します。

```
ok dev /sbus/ACME,widget
```

find-device は基本的には dev と同じです。異なるのは入力パス名の受け渡し方法だけです。

```
ok " /sbus/ACME,widget" find-device
```

注 – dev または find-device でデバイスノードを選択した後には、そのノードの方法は実行できません。それは、dev が現在のインスタンスを設定しないためです。この問題についての詳細は、『Writing FCode 3.x Programs』を参照してください。

---

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
/SUNW,UltraSPARC@0,0
/sbus@1f,0
/counter-timer@1f,3c00
/virtual-memory
/memory@0,0
/aliases
/options
/openprom
/chosen
/packages
/sbus@1f,0/cgsix@1,0
/sbus@1f,0/lebuffer@0,40000
/sbus@1f,0/dma@0,81000
/sbus@1f,0/SUNW,bpp@e,c800000
/sbus@1f,0/SUNW,hme@e,8c00000
/sbus@1f,0/SUNW,fas@e,8800000
/sbus@1f,0/sc@f,1300000
/sbus@1f,0/zs@f,1000000
/sbus@1f,0/zs@f,1100000
/sbus@1f,0/eeprom@f,1200000
/sbus@1f,0/SUNW,fdtwo@f,1400000
/sbus@1f,0/flashprom@f,0
/sbus@1f,0/auxio@f,1900000
/sbus@1f,0/SUNW,CS4231@d,c000000
/sbus@1f,0/SUNW,fas@e,8800000/st
/sbus@1f,0/SUNW,fas@e,8800000/sd
/openprom/client-services
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
ok
```

次に words の使用例を示します。

```
ok dev /zs
ok words
ring-bell       read         remove-abort?  install-abort
close           open         abort?         restore
clear           reset        initkbdmouse   keyboard-addr mouse
1200baud        setbaud      initport       port-addr
```

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 1-4 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 1-4　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
ok
```

上のヘルプメッセージは、まず、dump が Memory access 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。ヘルプが使用できないシステムも存在します。

OpenBoot コマンドの使用上の注意

オペレーティングシステムが実行を開始すると、OpenBoot が正しく動作しない場合があります。たとえば、「Stop-A」または halt を使用した後などです。このような状況になるのは、オペレーティングシステムが、続行中の OpenBoot の操作と一貫性のない方法でシステムの状態を変更する場合に発生します。このような場合は、システムに電源を再投入して正常動作に回復させてください。

たとえば、オペレーティングシステムを起動し、OpenBoot に入り、次に `probe-scsi` コマンド (第 2 章「システムの起動とテスト」で説明) 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かったり、システムの電源を再投入しなければならない場合があります。

オペレーティングシステムが実行されたために失敗した OpenBoot コマンドを実行し直すには、次のようにします。

1. printenv を使用して、NVRAM 設定変数 *auto-boot?* の値を調べます。true の場合は、*SETENV* を使用してこの値を false に設定します。
2. システムをリセットします。
3. ユーザーインタフェースで停止した後、OpenBoot コマンドを実行します。
4. NVRAM 設定 *auto-boot?* の値を復元します。
5. システムをリセットします。

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読み込み、実行するプロセスのことです。起動は、自動的にも、ユーザーインタフェースでコマンドを入力しても開始できます。

起動処理は多数の**システム変数**によって制御されます (システム変数については、第 3 章「システム変数の設定」で詳細に説明します)。起動処理に関係するシステム変数を次に示します。

- `auto-boot?`

この変数は、システムをリセットまたは電源投入後にシステムを自動的に起動させるかどうかを指定します。この変数は一般的に `true` です。

- `boot-command`

この変数は、`auto-boot?` が `true` の場合に実行されるコマンドを指定します。`boot-command` のデフォルト値は、引数を付けない `boot` です。

- diag-switch?

この値が true の場合、診断モードで実行されます。この変数のデフォルト値は false です。

- boot-device

この変数の内容は、OpenBoot が診断モードでないときに使用されるデフォルトの起動デバイスの名前です。

- boot-file

この変数の内容は、OpenBoot が診断モードでないときに使用されるデフォルトの起動引数です。

- diag-device

この変数の内容はデフォルトの診断モード起動デバイスの名前です。

- diag-file

この変数の内容は診断モード時のデフォルトの起動引数です。

上記の各システム変数の値に基づいて、起動処理はさまざまな形態で進められます。次にいくつかの例を示します。

- auto-boot? が true の場合は、OpenBoot が診断モードであるかどうかによって、マシンはデフォルトの起動デバイスか、診断起動デバイスから起動します。
- auto-boot? が false の場合は、マシンはシステムを起動しないで、OpenBoot のユーザーインタフェースで停止します。システムを起動するには、次のいずれかを実行します。
  - 引数をまったく指定しないで boot コマンドを入力します。マシンは、デフォルト起動引数を使用して、デフォルトの起動デバイスから起動します。
  - 起動デバイスを明示的に指定して boot コマンドを入力します。マシンは、指定した引数を使用して、デフォルトの起動デバイスから起動します。
  - 起動引数を明示的に指定して boot コマンドを入力します。マシンは、指定した引数を使用して、デフォルトの起動デバイスから起動します。
  - 起動デバイスと起動引数を明示的に指定して boot コマンドを入力します。マシンは、指定した引数を使用して、指定したデバイスから起動します。

一般ユーザー向けの起動

一般的に、auto-boot? は true、boot-command は boot に設定されており、OpenBoot は診断モードにはなりません。したがって、システムに最初に電源を投入したり、システムをリセットしたときは、システムは、boot-device によって記述されたデバイスから、boot-file によって記述された引数を使用してプログラムを読み込み、実行します。

auto-boot? が false のときにデフォルトプログラムを起動する場合は、ok プロンプトで boot と入力します。

上級ユーザー向けの起動

起動はクライアントプログラムを読み込み実行するプロセスです。クライアントプログラムは、通常はオペレーティングシステムまたはオペレーティングシステムのローダプログラムですが、boot を使用して診断などその他の種類のプログラムを読み込み、実行することもできます。(オペレーティングシステム以外のプログラムの読み込みについての詳細は、第 5 章「プログラムの読み込みと実行」を参照してください。)

起動は、普通は、上記で説明した各システム変数の内容値に基づいて自動的に行われますが、ユーザーインタフェースからも開始できます。

OpenBoot は起動処理で次に示す手順を実行します。

- 最後のリセット以降にクライアントプログラムが実行されている場合は、OpenBoot ファームウェアがマシンをリセットする場合があります。(リセットするかどうかは実装によって決まります。)
- boot コマンド行を読み込んで、使用する起動デバイスと起動引数を知ることによりデバイスが選択されます。起動デバイスや起動引数値は、boot コマンドの形式によってはシステム変数から取り出されます。
- デバイスツリーの /chosen ノードの bootpath、bootargs 特性に選択した値が設定されます。
- 選択されたプログラムが、選択したデバイスのタイプによって決まるプロトコルを使用してメモリーに読み込まれます。たとえば、ディスク起動の場合はディスクの先頭から決まったブロック数が読み込まれるのに対して、テープ起動では特定のテープファイルが読み込まれます。

- 読み込まれたプログラムが実行されます。プログラムの動作は、さらに、(存在する場合) 選択するシステム変数に設定されていた引数文字列、コマンド行で `boot` コマンドに渡される引数文字列によっても制御できます。

多くの場合、起動処理によって読み込み、実行されるプログラムは**二次起動プログラム**であり、それはさらに別のプログラムを読み込みすることを目的とします。この二次起動プログラムは、OpenBoot が使用するものとは異なるプロトコルを使用して二次起動プログラムを読み込むことができます。たとえば、OpenBoot は簡易ファイル転送プロトコル (TFTP) を使用して二次起動プログラムを読み込むこともでき、さらに読み込まれた二次起動プログラムが今度はネットワークファイルシステム (NFS) を使用してオペレーティングシステムを読み込むこともできます。

一般的な二次起動プログラムには次の形式の引数を指定できます。

filename -flags

ここで、*filename* はオペレーティングシステムを格納しているファイルの名前であり、*-flags* は、二次起動プログラム、オペレーティングシステム、またはそれらの両方の起動フェーズの詳細制御用のオプションのリストです。すぐ次の `boot` コマンドのテンプレートに示すように、OpenBoot はそのようなテキストをすべて OpenBoot そのものには特別な意味をもたない、1 つの隠された *arguments* 文字列として取り扱うということ、つまり、引数文字列は変更されないでそのまま指定されたプログラムに渡されるということに注意してください。

`boot` コマンドの形式は次のとおりです。

```
ok boot [device-specifier] [arguments]
```

boot コマンドの省略可能な変数を表 2-1 で説明します。

表 2-1　boot コマンドの省略可能な変数

変数名	説明
[*device-specifier*]	起動デバイスの名前 (フルパス名または devalias)。一般的には次のような値が使用されます。 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) net (Ethernet) tape (SCSI テープ) *device-specifier* を指定しないときに、diagnostic-mode? から false が返された場合は、boot は boot-device システム変数によって指定されるデバイスを使用します。 *device-specifier* を指定しないときに、diagnostic-mode? から true が返された場合は、boot は diag-device システム変数によって指定されるデバイスを使用します。
[*arguments*]	起動するプログラム (たとえば *stand/diag*) の名前と任意のプログラム引数。 *arguments* を指定しないときに、diagnostic-mode? から false が返された場合は、boot は boot-file システム変数によって指定されるファイルを使用します。 *arguments* を指定しないときに、diagnostic-mode? から true が返された場合は、boot は diag-file システム変数によって指定されるファイルを使用します。

注 – デバイス名を必要とする大部分の (boot や test などの) コマンドには、フルデバイス名でもデバイスの別名でも指定できます。本書では、*device-specifier* (デバイス指定子) という用語は、デバイスパス名でもデバイスの別名でも指定できるということを意味しています。

デバイスの別名は構文的には *arguments* と区別できないので、OpenBoot はこのあいまいさを次のように解決します。

- コマンド行上の boot の後の空白文字で区切られた語の最初の文字が / である場合は、その語はデバイスパス、つまり *device-specifier* です。この *device-specifier* の右側のテキストはすべて *arguments* になります。
- 上記とは異なって、空白文字で区切られた語の最初の文字が / ではなくて、かつ既存のデバイスの別名と一致する場合は、その語は *device-specifier* です。この *device-specifier* の右側のテキストはすべて *arguments* になります。

- 上記のどちらとも異なる場合は、該当するデフォルトの起動デバイスが使用され、boot の右側のテキストはすべて *arguments* になります。

したがって、boot コマンド行には次に示すような書式が可能です。

```
ok boot
```

この書式の場合は、boot はデフォルトの起動デバイスからデフォルトの起動引数によって指定されるプログラムを読み込み、実行します。

```
ok boot device-specifier
```

boot の引数が 1 つだけであって、その最初の文字が / であるか、または定義されている devalias の名前である場合は、boot はその引数をデバイス指定子として使用します。つまり、boot は指定されたデバイスからデフォルト起動引数によって指定されるプログラムを読み込み、実行します。

たとえば、明示的にディスクから起動するには、次のように入力します。

```
ok boot disk
```

明示的にネットワークから起動するには、次のように入力します。

```
ok boot net
```

boot の引数が 1 つだけであって、かつその最初の文字が / でもなく、また定義されている devaliasの名前でもない場合は、boot はその後のテキストをすべて boot の引数として使用します。

```
ok boot arguments
```

boot はデフォルトの起動デバイスから引数によって指定されるプログラムを読み込み、実行します。

```
ok boot device-specifier arguments
```

空白文字で区切られた引数が最低 2 つはあって、かつそれらの最初の引数の最初の文字が / であるか、または定義されている devalias の名前である場合は、boot は最初の引数をデバイス指定子として使用し、その後のテキストをすべて boot の引数として使用します。つまり、boot は指定されたデバイスから引数によって指定されるプログラムを読み込み、実行します。

上に示したすべての場合について、boot はそれが使用するデバイスを /chosen ノードの bootpath 特性に記録します。boot はさらに、それが使用する引数も /chosen ノードの bootargs 特性に記録します。

デバイスの別名定義はシステムごとに異なります。第 1 章「概要」で説明している devalias を使用してシステムの別名の定義を調べてください。

診断の実行

いくつかの診断ルーチンがユーザーインタフェースで使用できます。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。

diagnostic-mode? から返される値は、次に示す設定を行います。

- (デバイスとファイルがこれらのコマンドの引数として明示的に指定されなかった場合) boot、load コマンドが使用するデバイスとファイルを選択します。
- 電源投入時に自己診断テストで実行される診断の範囲と (実装によって異なる) 生成される診断メッセージ数を設定します。

OpenBoot は診断モードになり、次の条件のうち少なくとも 1 つが満たされた場合は、diagnostic-mode? コマンドから true が返されます。

- システム変数 diag-switch? が true に設定されている。
- (存在する場合) マシンの診断スイッチが「オン」である。
- 別のシステム依存のインジケータが拡張診断を要求している。

OpenBoot が診断モード (Diagnostic mode) のときは、diag-device の値は boot コマンドの「デフォルトの起動デバイス」として使用され、diag-file の値は「デフォルトの起動引数」として使用されます。

OpenBoot が診断モード (Diagnostic mode) でないときは、boot-device の値は boot コマンドの「デフォルトの起動デバイス」として使用され、boot-file の値は「デフォルトの起動引数」として使用されます。

表 2-2 に診断テスト用コマンドの一覧を示します。これらのテストは OpenBoot で実装していないことがあります。

表 2-2　診断テストコマンド

コマンド	説明
probe-scsi	SCSI バスに接続されているデバイスを確認します。
test *device-specifier*	指定したデバイスの自己診断テスト (selftest) を実行します。例を示します。 test net : ネットワーク接続をテストします。
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

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
```

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。

取り付けられているデバイスのテスト

取り付けられている 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
```

一般的に、メッセージが何も表示されなければテストは成功です。

注 – 多くのデバイスでは、このテストを実行するために、システムの `diag-switch?` パラメタを true にしておく必要があります。

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
```

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

ソフトウェアによる取り出しが可能なシステムのドライブから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
```

メモリーのテスト

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
```

注 – 一部の OpenBoot システムにはこのテストワードはありません。

上の例で、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。

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 秒ごとに 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。

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

ネットワークコントローラのテスト

ネットワーク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 - (結果)
External Loopback test - (結果)
ok
```

システムはテストの結果を示すメッセージを応答します。

注 – 特定のネットワークコントローラとシステムが接続しているネットワークのタイプとによって、各種レベルのテストが可能です。それらのテストによっては、ネットワークインタフェースをネットワークに接続することが必要な場合があります。

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 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

ユーザーインタフェース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。banner は ObenBoot のすべての実装で提供されます。その他のコマンドについては、一部の実装で拡張機能として提供されます。それらのコマンドを 表 2-3 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 ROM 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 ROM 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 2-3　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	取り付けられ、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 の内容をフォーマットされた形式で表示します。
.traps	プロセッサに依存する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。
.speed	プロセッサおよびバスの速度を表示します。

表 1-3 のデバイスツリー表示コマンドも参照してください。

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-all 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-all
```

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、ユーザーインタフェースに入ります。

第3章

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。システム変数は常に注意深く調整する必要があります。

この章で説明する手順は、ユーザーインタフェースに入っているものとしています。ユーザーインタフェースに入る方法については、第 1 章「概要」を参照してください。

表 3-1 に、IEEE Standard 1275-1994 の定義に従う標準的セットの NVRAM システム変数の一覧を示します。

表 3-1　標準システム変数

変数名	設定値	説明
auto-boot?	true	true の場合、電源投入後またはリセット後に自動的に起動します。
boot-command	boot	auto-boot? が true の場合に実行されるコマンド。
boot-device	disk net	起動するデバイス。
boot-file	空白文字	起動するプログラムに渡される引数。
diag-device	net	診断起動ソースデバイス。
diag-file	空白文字	診断モードで起動するプログラムに渡される引数。
diag-switch?	false	true の場合、診断モードで実行します。

表 3-1　標準システム変数 (続き)

変数名	設定値	説明
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。
input-device	keyboard	コンソール入力デバイス (通常 keyboard、ttya、ttyb)
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	コンソール出力デバイス (通常 screen、ttya、ttyb)。
screen-#columns	80	画面上のカラム数 (文字数/行)。
screen-#rows	34	画面上の行数。
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、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)。
use-nvramrc?	false	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)。

上記以外に、IEEE Standard 1275-1994 の SBus 版でシステム変数が定義されています。それらの変数を表 3-2 に示します。

表 3-2　SBus システム変数

変数名	設定値	説明
sbus-probe-list	0123	プローブされる SBus スロットとそれらがプローブされる順番。

注 – OpenBoot の実装が異なると、使用するデフォルトやシステム変数も異なる場合があります。

変数設定の表示と変更

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

表 3-3　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。 `printenv` *variable* は指定する変数の現在値を表示します。
`setenv` *variable value*	*variable* 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。)
`set-default` *variable*	指定する変数 (*variable)* 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。
`password`	security-password を設定します。

以降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。

注 – Sun OS は OpenBoot システム変数を変更するための `eeprom` (1M) ユーティリティーを備えています。

変数の現在の設定の表示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

Variable 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
diag-file
diag-device     net     net
boot-file
boot-device     disk    disk net
auto-boot?      false   true
fcode-debug?    true    false
use-nvramrc?    false   false
nvramrc
screen-#columns 80      80
screen-#rows    34      34
security-mode   none    none
security-password
security-#badlogins     0
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 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。

```
ok setenv variable-name value
```

variable-name 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。

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

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

注 – 多くの場合、変数の値を変更しても、そのままでは *OpenBoot* ファームウェアの動作には無効です。次の電源再投入またはシステムリセットで初めて、ファームウェアはそれらの変数の新しい値を使用します。

`set-default` 変数と `set-defaults` コマンドを使用して、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、次のように入力します。

```
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
```

SPARC システムでは、マシンのパワーアップ処理の間「Stop-N」 を押し下げておくことにより、NVRAM 変数をそれぞれのデフォルト設定に戻すことができます。このコマンドを発行するときは、SPARC システムに電源を投入した直後に「Stop-N」を押し、数秒間またはバナーが表示されるまで (ディスプレイが使用できる場合)、押さえたままにしておきます。これにより、SPARC と互換性のあるマシンの NVRAM の内容をデフォルト設定に戻すことができます。

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、ユーザーがユーザーインタフェース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、セキュリティーの高い順序で示すと次のとおりです。

表 3-4　security-mode 設定用コマンド

モード	コマンド
full	go 以外のコマンドはすべてパスワードを必要とします。
command	boot および go 以外のコマンドはすべてパスワードを必要とします。
none	パスワードを必要としません (デフォルト)。

コマンドセキュリティー

security-mode を設定しているときは、

- boot コマンドだけを入力する場合、パスワードは必要ありません。ただし、引数を付けて boot コマンドを使用すると、パスワードが必要です。
- go コマンドはパスワードを要求しません。
- その他の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。

```
ok boot     (パスワード必要なし)
ok go       (パスワード必要なし)
ok boot filename   (パスワード必要)
Password:     (入力時パスワードは画面表示されない)
ok reset-all     (パスワード必要)
Password:   (入力時パスワードは画面表示されない)
```

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。また、セキュリティーモードを設定する前にセキュリティーパスワードを設定してください。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。購入先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。

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
```

設定するセキュリティーパスワードは 0 ～ 8 つの文字でなければなりません。8 文字目より後の文字は無視されます。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。

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。

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、

- boot コマンドの入力時にはパスワードが必要です。
- go コマンドはパスワードを要求しません。
- その他の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。

```
ok go     (パスワード必要なし)
ok boot    (パスワード必要)
Password:  (入力時パスワードは画面表示されない)
ok boot filename   (パスワード必要)
Password:  (入力時パスワードは画面表示されない)
ok reset-all    (パスワード必要)
Password:   (入力時パスワードは画面表示されない)
```

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。また、セキュリティーモードを設定する前にセキュリティーパスワードを設定してください。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。購入先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。

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
Sun Ultra 1 SBus (UltraSPARC 167 MHz),Keyboard PresentPROM Rev.
3.0, 64MB memory installed, Serial # 289Ethernet address
8:0:20:d:e2:7b, Host ID: 80000121
ok
```

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。

作成カスタマイズしたバナー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、前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、64×64 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。次にこの配列を $setenv で oem-logo にインストールします。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 " oem-logo" $setenv
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。

注 – 一部のシステムは oem-logo 機能をサポートしません。

入出力の制御

コンソールは、OpenBoot とユーザーとの間の第一の対話手段として使用されます。コンソールは、ユーザーから与えられる情報を受け取るために使用される入力デバイスと、ユーザーに情報を送るために使用される出力デバイスからなっています。一般的に、コンソールはテキスト・グラフィックス両用ディスプレイデバイスとキーボードの組み合わせか、シリアルポートに接続された ASCII 端末です。

システム入出力の制御関係の構成用として次に示すシステム変数があります。

- input-device
- output-device
- screen-#columns
- screen-#rows

これらの変数を使用してコンソール用の電源投入時デフォルトを割り当てます。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ファームウェア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。input-device、output-device の値はデバイス指定子でなければなりません。多くの場合、別名 keyboard、screen がこれらの変数の値として使用されます。

システムをリセットすると、指定したデバイスが初めのフォームウェアのコンソール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「Forth ツールの使用方法」で説明する input または output コマンドを使用します。)

設定デフォルト入出力デバイス ttya を電源投入時初期コンソール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、入力は予備のデバイス (通常 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、出力は予備のデバイスに送られます。

デフォルト出力デバイスとして SBus 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/SUNW,leo
ok
```

シリアルポート特性の設定

設定シリアルポート特性シリアルポートの代表的な通信特性の範囲は次のとおりです。

- *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

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、ハンドシェーク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?

auto-boot? が true で OpenBoot が診断モードではない場合は、システムは電源再投入またはシステムのリセット後 (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-file と boot-device に対する変更は、次に起動 (boot) が実行されたときに有効になります。

電源投入時自己診断テストの制御 (POST)

電源投入時自己診断テスト用として次に示す変数があります。

- diag-switch?
- diag-device
- diag-file
- diag-level

diag-switch? を true に設定すると、diagnostic-mode? が true を返します。diagnostic-mode? が true を返すと、システムは次のように動作します。

- 以降の電源投入やシステムリセットプロセスでは、完全な 自己診断テストを実行します。

- 追加ステータスメッセージ (詳細は実装によって異なります) を表示することがあります。
- 異なる起動用システム変数を使用します。(起動処理に対する影響についての詳細は、第 2 章「システムの起動とテスト」を参照してください。)

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
```

注 – 一部のシステムは diagnostic-mode? が true を返す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が設定されているか、または diag-switch? が true に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。

注 – 一部の実装では、電源投入時に実装によって決まるキー入力処理を使用して diag-switch? を true に強制設定できる場合があります。詳細は、システムのマニュアルまたは本書の付録 C「障害追跡ガイド」を参照してください。

---

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
```

診断モードではない場合は、(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実行中は報告せず、診断の一部を実行します。

nvramrc の使用方法

nvramrc システム変数は**スクリプト**と呼ばれる内容を持ち、起動時に実行されるユーザー定義コマンドを格納します。

一般的に、nvramrc は起動時の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ようにASCII で格納されます。

use-nvramrc? システム変数が true の場合は、そのスクリプトが OpenBoot の起動処理で次のように評価されます。

- 電源投入時自己診断テスト (POST)を実行します。
- システムの初期化を実行します。
- (use-nvramrc? が true の場合) スクリプトを評価します。
- probe-all を実行します (FCode を評価します)。
- install-console を実行します。
- banner を実行します。
- 二次診断を実行します。
- (auto-boot? が true の場合) デフォルト起動を実行します。

場合によっては、probe-all install-console banner の処理を変更することが必要な場合があります。たとえば、差し込み式ディスプレイデバイスのプローブが終わってからコンソールデバイスの選択が終わるまでに、差し込み式デバイスの特性を変更するコマンドを実行する必要があることがあります。そのようなコマンドは、probe-all と install-console の間で実行する必要があります。出力をコンソールに表示するコマンドを install-console または banner の後に入れる必要があります。

これは、banner か suppress-banner を内容とするカスタムスクリプトを作成することにより実現できます。それは、そのスクリプトから banner か suppress-banner が実行されれば、probe-all、install-console および banner 処理が実行されないためです。つまり、そのスクリプト内部で probe-all、install-console、banner を場合により他のコマンドと混用する形で使用することができ、スクリプト終了後にそれらのコマンドを再び実行させないようにできます。

次の例外を除く、ほとんどすべてのユーザーインタフェースコマンドがスクリプトで使用できます。

- boot
- go
- nvedit
- password
- reset-all
- setenv security-mode

スクリプトの内容の編集

スクリプトエディタである nvedit では、表 3-5 に示すコマンドを使用して、スクリプトの内容を作成、変更することができます。

表 3-5　NVRAMAC に影響するコマンド

コマンド	説明
nvalias *alias device-path*	スクリプトにコマンド 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。
$nvalias	スタックから引数 *name-string* と *device-string* を使用する以外は、nvalias と同じ機能です。
nvedit	スクリプト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取って、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 に書き込まないで捨てます。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。

表 3-5　NVRAMAC に影響するコマンド (続き)

コマンド	説明
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。
$nvunalias	スタックから引数 *name-string* を使用する以外は、nvunalias と同じ機能です。

表 3-6 にスクリプトエディタで使用できる編集コマンドを示します。

表 3-6　スクリプトエディタキー操作コマンド

キー操作	説明
Control-B	1 文字位置戻ります。
Escape B	1 語戻ります。
Control-F	1 文字位置進みます
Escape F	1 語進みます。
Control-A	行の先頭に戻ります。
Control-E	行の終わりに進みます。
Control-N	スクリプト編集バッファーの次の行に進みます。
Control-P	スクリプト編集バッファーの前の行に戻ります。
Return (Enter)	カーソル位置に改行を挿入し、次の行に進みます。
Control-O	カーソル位置に改行を挿入し、現在の行にとどまっています。
Control-K	カーソル位置から行の終わりまで消去し、消去した文字を保存バッファーに格納します。カーソルが行の終わりにある場合は、現在の行に次の行をつなぎます (つまり、改行を削除します)。
Delete	前の 1 文字を削除します。
Backspace	前の 1 文字を削除します。
Control-H	前の 1 文字を削除します。
Escape H	語の先頭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	語の先頭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	次の 1 文字を消去します。

表 3-6　スクリプトエディタキー操作コマンド (続き)

キー操作	説明
Escape D	カーソルから語の終わりまで消去し、消去した文字を保存バッファーに格納します。
Control-U	1 行全体を消去し、消去した文字を保存バッファーに格納します。
Control-Y	保存バッファーの内容をカーソルの前に挿入します。
Control-Q	次の文字の前に引用符を付けます (つまり、制御文字を挿入できます)。
Control-R	1 行を入力し直します。
Control-L	編集バッファーの内容全体を表示します。
Control-C	エディタを終了し、OpenBoot 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、スクリプトには戻されません。(後で `nvstore` を使用して書いて戻してください。)

スクリプトファイルの起動

次の手順で、スクリプトコマンドファイルを作成し起動してください。

- `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スクリプトの内容を編集します。

- `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。

  変更を保存するために `nvstore` をまだ入力していない場合、`nvrun` と入力して一時編集バッファーの内容を実行できます。

1. `nvstore` と入力して変更結果を保存します。
2. 次のように入力して、スクリプトを有効にします。

```
setenv use-nvramrc? true
```

3. reset-all と入力してシステムをリセットしてからスクリプトの内容を実行するか、次のように入力してスクリプトの内容を直接実行します。

```
nvramrc evaluate
```

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
ok reset-all
…
ok hello
Hello, world.
ok
```

上の例で 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 プログラミング言語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 I「Forth ワードリファレンス」に全コマンドのリストを載せてあります。

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の再投入を行う必要があることがあります。

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
.
0<
+
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で分離する必要があります。他の一部のプログラミング言語で通常「句読文字」として取り扱われる文字は、Forth ワード間の区切りには使用されません。実際には、それらの多くの「句読文字」は Forth ワードそのものなのです。

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 キー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、次の例は、

```
ok testa testb testc
ok
```

次の 3 行と同じです。

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

OpenBoot では、Forth ワード名に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。出力が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。

データ型

表 4-1 に示す用語は Forth が使用するデータ型です。

表 4-1　Forth データ型の定義

表記	説明
byte	8 ビット値。
cell	実装によって定義される固定サイズのセルであり、アドレス単位と対応のビット数で指定されます。データスタック要素、復帰スタック要素、アドレス、実行トークン、フラグ、整数は 1 セル幅です。 OpenBoot システムでは、セルは最低 32 ビットからなり、十分に仮想アドレスを格納できます。セルサイズは実装によって異なることがあります。32 ビット」実装のセルサイズは 4 であり、「64 ビット」実装のセルサイズは 8 です。
doublet	16 ビット値。
octlet	64 ビット値。64 ビット実装にかぎり定義されます。
quadlet	32 ビット値。

数値の用法

数値は、たとえば 55 や -123 など、その値をキーボードで入力します。Forth は整数 (すべての数字) しか受け入れません。分数値 (たとえば 2/3) は解釈しません。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。表記規則では、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。

特に指定がないかぎり、OpenBoot は 1 セルサイズのデータ項目の整数演算を実行して、1 セルサイズの結果を生成します。

OpenBoot の実装ではデフォルトで基数 16 (16 進数) を使用するよう奨励していますが、これは必須ではありません。したがって、正しく演算されるために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。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 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方法を次に示します。

```
ok 10 .d
16
ok
```

上記の画面に表示されている 16 は、16 進で演算が行われることを示しています。10 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。.d は、現在の基数とは無関係に 10 を基数として表示します。

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り、44 はそのすぐ下になる)
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
44 7 8ok noshowstack
ok
```

一番上のスタック項目は、ok プロンプトの直前に、リストの最後の項目として常に表示されます。上の例では、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、noshowstack と入力すれば、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、通常に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。スタックダイアグラムは、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。

-- の左側におかれる項目は、ワードがスタックが取り出し、その操作で使用するスタック項目を表します。これらの項目の最も右側のものがスタックの一番上にあり、それより前の項目は順次にその下にあります。つまり、引数は左から右の順にスタックにプッシュされ、最も新しい項 (ダイアグラムの最も右の項目) がスタックの一番上に残ることになります。

-- の右側におかれる項目は、ワードが実行を終了した後にスタックに残されるスタック項目を表します。この場合もやはり、最も右側の項目がスタックの一番上に置かれ、それより前の項目はその下に入ります。

たとえば、ワード + のスタックダイアグラムは

```
( nu1 nu2 -- sum )
```

です。この場合、+ は 2 つの数値 (nu1 と nu2) をスタックから削除し、それらの和 (sum) をスタックに残します。もう 1 つの例として、ワード . のスタックダイアグラム

```
( nu -- )
```

があります。この場合、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、コマンド行のワードのすぐ後に別のワード、または他のテキストが必要なことがあります。たとえば、ワード 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。本書で使用するスタック項目の省略表記については、表 4-2 を参照してください。

表 4-2 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)。
???	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 (1 セル中の下位バイト)。
char	7 ビット値 (1 セル中の下位バイト)。最上位ビットは不定
cnt	カウント値。
len	長さ。
size	カウント値または長さ。
d*xxx*	倍 (拡張) 精度数。2 セルで、スタックの上部には、最上位のセルが置かれる。quadlet (32 ビット)。

表 4-2　スタック項目の表記法 (続き)

表記	説明
`<eol>`	行末区切り文字。
`false`	0 (false フラグ)。
`n n1 n2 n3`	通常の符号付き 1 セルの値。
`nu nu1`	符号付きまたは符号なしの 1 セルの値。
`<nothing>`	ゼロスタック項目。
`o o1 o2 oct1 oct2`	Octlet (符号付き 64 ビット値)。
`oaddr`	Octlet (64 ビット) 境界のアドレス。
`octlet`	8 バイト数値。
`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` *qxxx*	Quadlet (32 ビット値、1 セル中の下位 4 バイト)
`qaddr`	Quadlet (32 ビット値) 境界のアドレス。
`true`	-1 (true フラグ)。(真)
uxxx	1 セル、符号なし 32 ビット値、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` *wxxx*	Doublet (16 ビット値、1 セル中の下位 2 バイト)。
`x x1`	任意の 1 セルのスタック項目。
`x.lo x.hi`	データ項目の下位／上位ビット
`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 4-3 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 4-3　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
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。
2drop	(x1 x2 --)	スタックから 2 つの項目を削除します。
3drop	(x1 x2 x3 --)	スタックから 3 つの項目を削除します。
dup	(x -- x x)	一番上のスタック項目を複製します。
2dup	(x1 x2 -- x1 x2 x1 x2)	2 つのスタック項目を複製します。
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。
?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。
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。
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。
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)	スタック項目を復帰スタックに転送します。
r>	(-- x) (R: x --)	復帰スタック項目をスタックに転送します。
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。
roll	(xu … x1 x0 u -- xu-1 … x1 x0 xu)	u 個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逆方向に回転します。
-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。
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。

表 4-3　スタック操作コマンド (続き)

コマンド	スタックダイアグラム	説明
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。
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。
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です。

```
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
```

利用者定義の作成

Forth は、既存ワードの処理から新しいコマンドワードを作成するための簡単な手段を提供します。表 4-4 に、利用者定義作成用の Forth ワードを示します。

表 4-4　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、: を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、(+ + + .) の操作を行う add4 の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
```

これらの定義はシステムをリセットすると消去されます。よく使う定義を保存するには、それらをスクリプトにする、つまりテキストファイルにして、ホストシステム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みできます。(ファイルの読み込みについての詳細は、第 5 章「プログラムの読み込みと実行」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
```

複数行の定義に] プロンプトを使用するのは、サン・マイクロシステムズ社の実装の特徴です。

- スタックダイアグラムはワードの正しい使い方を示します。したがって、そのスタック効果がない場合 (--) であっても、作成する定義ごとに必ずスタックダイアグラムを入れてください。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

単精度整数演算

表 4-5 に示すコマンドは、単精度整数演算を行います。

表 4-5 単精度演算機能

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。
*/	(n1 n2 n3 -- quot)	*nu1* * *nu2* / *n3* を計算します。入力、出力、中間値はすべて 1 つのセルに入ります。
/	(n1 n2 -- quot)	*n1* を *n2* で割ります。剰余は捨てられます。
1+	(nu1 -- nu2)	1 を足します。
1-	(nu1 -- nu2)	1 を引きます。
2+	(nu1 -- nu2)	2 を足します。
2-	(nu1 -- nu2)	2 を引きます。
abs	(n -- u)	絶対値。

表 4-5　単精度演算機能 (続き)

コマンド	スタックダイアグラム	説明
bounds	(start len -- len+start start)	do または ?do ループ用に start、len を end、start に変換します。
even	(n -- n | n+1)	最も近い偶数の整数 >= *n* に丸めます。
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。
*/mod	(n1 n2 n3 -- rem quot)	*n1* * *n2* / *n3* の剰余と商。
/mod	(n1 n2 -- rem quot)	*n1* / *n2* の剰余と商。
negate	(n1 -- n2)	*n1* の符号を変更します。
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 1 セル数値を符号なし 1 セル数値で割り、 1 セルの剰余と商を生じます。
<<	(x1 u -- x2)	lshift の同義語。
>>	(x1 u -- x2)	rshift の同義語。
2*	(x1 -- x2)	2 を掛けます。
2/	(x1 -- x2)	2 で割ります。
>>a	(x1 u -- x2)	*x1* を *u* ビット右に算術シフトします。
and	(x1 x2 -- x3)	ビット単位の論理積。
invert	(x1 -- x2)	*x1* の全ビットを反転します。
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。
not	(x1 -- x2)	invert の同義語。
or	(x1 x2 -- x3)	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。
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。
xor	(x1 x2 -- x3)	ビット単位排他的論理和。

倍精度数演算

表 4-6 に示すコマンドは倍精度数の演算を行います。

表 4-6　倍精度数演算機能

コマンド	スタックダイアグラム	説明
d+	(d1 d2 -- d.sum)	d1 を d2 に足し、倍精度数 d.sum を生じます。
d-	(d1 d2 --d.diff)	d1 から d2 を引き、倍精度数 d.diff を生じます。
fm/mod	(d n -- rem quot)	*d* を *n* で割ります。
m*	(n1 n2 -- d)	符号付き乗算を行い、倍精度数の積を生じます。
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。
um*	(u1 u2 -- ud)	符号なし乗算を行い、符号なし倍精度数の積を生じます。
um/mod	(ud u -- urem uprod)	*ud* を *u* で割ります。

データ型変換

表 4-7 に示すコマンドはデータ型の変換を行います。

表 4-7　32 ビットデータ型変換機能

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 quadlet (32 ビット) を作ります。
bwjoin	(b.low b.hi -- word)	2 バイトを結合して doublet (16 ビット) を作ります。
lbflip	(quad1 -- quad2)	quadlet 内の 4 バイトを逆に並べ替えます。
lbsplit	(quad -- b.low b2 b3 b.hi)	quadlet を 4 バイトに分割します。
lwflip	(quad1 -- quad2)	quadlet 内の 2 つの doublet をスワップします。
lwsplit	(quad -- w.low w.hi)	quadlet を 2 つの doublet に分割します。

表 4-7　32 ビットデータ型変換機能 (続き)

コマンド	スタックダイアグラム	説明
wbflip	(word1 -- word2)	doublet 内の 2 バイトをスワップします。
wbsplit	(word -- b.low b.hi)	doublet を 2 バイトに分割します。
wljoin	(w.low w.hi -- quad)	2 つの doublet を結合して quadlet を作ります。

表 4-8 に示すデータ型変換用コマンドは 64 ビットの OpenBoot 実装専用です。

表 4-8　64 ビットデータ型変換機能

コマンド	スタックダイアグラム	説明
bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)	8 バイトを結合して octlet を作ります。
lxjoin	(quad.lo quad.hi -- o)	2 つの quadlet を結合して octlet を作ります。
wxjoin	(w.lo w.2 w.3 w.hi -- o)	4 つの doublet を結合して octlet を作ります。
xbflip	(oct1 -- oct2)	octlet 内の 8 バイトを逆に並べ替えます。
xbsplit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)	octlet を 8 バイトに分割します。
xlflip	(oct1 -- oct2)	octlet 内の 2 つの quadlet をスワップします。各 quadlet 内の 4 バイトは逆に並べ替えられません。
xlsplit	(o -- quad.lo quad.hi)	octlet を 2 つの quadlet に分割します。
xwflip	(oct1 -- oct2)	octlet 内の 4 つの doublet を逆に並べ替えます。各 doublet 内の 2 バイトは並べ替えられません。
xwsplit	(o -- w.lo w.2 w.3 w.hi)	octlet を 4 つの doublet に分割します。

アドレス演算

表 4-9 に示すコマンドはアドレス演算を行います。

表 4-9　アドレス演算機能

コマンド	スタックダイアグラム	説明
aligned	(n1 -- n1 | a-addr)	必要な場合、*n1* 大きくして可変境界アドレスを生じます。
/c	(-- n)	バイトのアドレス単位数 : 1。
/c*	(nu1 -- nu2)	chars の同義語。

表 4-9　アドレス演算機能 (続き)

コマンド	スタックダイアグラム	説明
ca+	(addr1 index -- addr2)	*addr1* を /c の値の *index* 倍増やします。
ca1+	(addr1 -- addr2)	char+ の同義語。
cell+	(addr1 -- addr2)	*addr1* を /n の値だけ増やします。
cells	(nu1 -- nu2)	*nu1* に /n の値を掛けます。
char+	(addr1 -- addr2)	*addr1* を /c の値だけ増やします。
chars	(nu1 -- nu2)	*nu1* に /c の値を掛けます。
/l	(-- n)	quadlet のアドレス単位数 : 通常 4。
/l*	(nu1 -- nu2)	*nu1* に /l の値を掛けます。
la+	(addr1 index -- addr2)	*addr1* を /l の値の *index* 倍増やします
la1+	(addr1 -- addr2)	*addr1* を /l の値だけ増やします。
/n	(-- n)	セルのアドレス単位数。
/n*	(nu1 -- nu2)	cells の同義語。
na+	(addr1 index -- addr2)	*addr1* を/n の値 *index* 倍増やします。
na1+	(addr1 -- addr2)	cell+ の同義語。
/w	(-- n)	doublet のアドレス単位数 : 通常 2。
/w*	(nu1 -- nu2)	*nu1* に /w の値を掛けます。
wa+	(addr1 index -- addr2)	*addr1* を /w の値の *index* 倍増やします。
wa1+	(addr1 -- addr2)	*addr1* を /w の値だけ増やします。

表 4-10 に示すアドレス演算用コマンドは 64 ビットの OpenBoot 実装専用です。

表 4-10　64 ビットアドレス演算機能

コマンド	スタックダイアグラム	説明
/x	(-- n)	octlet のアドレス単位数 : 通常 8。
/x*	(nu1 -- nu2)	nu1 に /x の値を掛けます。
xa+	(addr1 index -- addr2)	addr1 を /x の値の index 倍増やします。
xa1+	(addr1 -- addr2)	addr1 を /x の値だけ増やします。

メモリーのアクセス

仮想メモリー

メモリーアクセスする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割り当て。

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 (システムによっては 64 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 (doublet) の操作を示し、`l` (ロングワード) という接頭辞は 32 ビット (quadlet) の操作を示します。`x` という接頭辞は 64 ビット (octlet) 操作を示します。

`waddr`、`qaddr`、`oaddr` 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 の倍数で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 (システムによっては 64 ビット) を取り出す場合は、@ の代わりにそれぞれ `w@`、`l@`、または `x@` を使用してください。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 4-11 にメモリーアクセス用のコマンドを示します。

表 4-11 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
`!`	(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。
`@`	(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 --)	*addr* 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。
`c@`	(addr -- byte)	*byte* を *addr* から取り出します。
`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* バイトのメモリーを表示します。
`erase`	(addr len --)	*addr* から *len* バイトのメモリーを `0` に設定します。
`fill`	(addr len byte --)	*addr* から *len* バイトのメモリーを値 *byte* に設定します。
`l!`	(q qaddr --)	quadlet *q* を *qaddr* に格納します。
`l@`	(qaddr -- q)	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 をスワップします。

表 4-11 メモリーアクセスコマンド (続き)

コマンド	スタックダイアグラム	説明
lpeek	(qaddr -- false | quad true)	quadlet 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	(q qaddr -- okay?)	quadlet 8 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!	(q addr --)	quadlet *q* を任意の境界に格納します。
unaligned-l@	(addr -- q)	quadlet *q* を任意の境界で取り出します。
unaligned-w!	(w addr --)	doublet *w* を任意の境界に格納します。
unaligned-w@	(addr -- w)	doublet *w* を任意の境界で取り出します。
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。
<w@	(waddr -- n)	doublet *n* を *waddr* から符号拡張して取り出します。
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)	doublet w 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?)	doublet w の数値を *waddr* に格納します。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。

表 4-12 に示すメモリーアクセス用コマンドは 64 ビットの OpenBoot 実装専用です。

表 4-12 64 ビットメモリーアクセス機能

コマンド	スタックダイアグラム	説明
<l@	(qaddr -- n)	quadlet を qaddr から符号を拡張して取り出します。
x@	(oaddr -- o)	octlet を 64 ビット 境界アドレスから取り出します。
x!	(o oaddr --)	octlet を 64 ビット 境界アドレスに格納します。

表 4-12 64 ビットメモリーアクセス機能 (続き)

コマンド	スタックダイアグラム	説明
xbflips	(oaddr len --)	指定された領域の各 octlet 内の 8 バイトを逆に並べ替えます。len が/x の整数倍でない場合は、動作は不定です。
xlflips	(oaddr len --)	指定された領域の各 octlet 内の 2 つの quadlet をスワップします。各 quadlet 内の 4 バイトは逆に並べ替えられません。len が /x の整数倍でない場合は、動作は不定です。
xwflips	(oaddr len --)	指定された領域の各 octlet 内の 4 つの doublet を逆に並べ替えます。各 doublet 内の 2 バイトはスワップされません。len が /x の整数倍でない場合は、動作は不定です。

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。

```
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
```

一部の実装は、メモリーを16、32、64 の各ビット値として表示する、dump のバリエーションをサポートしています。sifting dump (67 ページの「辞書の検索」を参照) を使用すれば、システムにそのようなバリエーションがあるかどうかを確認できます。

(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。

表 4-13 にメモリーマップ操作用のコマンドを示します。

表 4-13 メモリーマップコマンド

コマンド	スタックダイアグラム	説明
alloc-mem	(len -- a-addr)	*len* 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	(a-addr len --)	alloc-mem で割り当てられていたメモリーを開放します。

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 (ef7a48) が表示されます。
- dump が ef7a48 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。
- 最後に、free-mem が、割り当てられた ef7a48 からの 4000 バイトのメモリーを返します。

```
ok
ok 4000 alloc-mem .
ef7a48
ok
ok ef7a48 constant temp
ok temp 20 dump
        0  1  2  3  4  5  6  7  \/  9  a  b  c  d  e  f   01234567v9abcdef
ef7a40  00 00 f5 5f 00 00 40 08  ff ef c4 40 ff ef 03 c8  ..u_..@..oD@.o.H
ef7a50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
ef7a60  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
ef7a40  00 00 f5 5f 00 00 40 08  55 55 55 55 55 55 55 55  ..u_..@.UUUUUUUU
ef7a50  55 55 55 55 55 55 55 55  55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUU
ef7a60  55 55 55 55 55 55 55 55  00 00 00 00 00 00 00 00  UUUUUUUU........
ok
ok temp 4000 free-mem
ok
```

デバイスレジスタ

前項で説明の仮想メモリーアクセス用のオペレータ (コマンド) を使用してデバイスレジスタをアクセスした場合、信頼性が欠けることがあります。デバイスレジスタアクセス用には特別なオペレータ (コマンド) があります。それらのオペレータ (コマンド) の場合は、使用する前にマシンを正しく設定する必要があります。これについての詳細は、『Writing FCode 3.x Programs』を参照してください。

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 ワー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、新しいワードを作成するときのスタック効果を示します。2 番目の (E:) 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 4-14 に新しい Forth ワードを作成するためのワード定義を示します。

作成辞書への登録ある Forth コマンドが既存のコマンドと同じ名前で作成されると、新しいコマンドが正常に作成されます。実装によっては「`new-name isn't unique`」というメッセージが表示されます。そのコマンドの以前の使用方法は、影響を受けません。そのコマンド名をそのあとに使用する場合、最新の定義を使用します。コマンド名の同じものは**すべて**同じ動作をするようにするには、`patch` を使用して修正してください。(109 ページの「patch と (patch)の使用方法」を参照してください。)

表 4-14 ワード定義

コマンド	スタックダイアグラム	説明
`:` *name*	(--) (E: … -- ???)	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*	(--) (E: … -- ???)	*old-name* と同じ操作をする *new-name* を作成します。
`buffer:` *name*	(size --) (E: -- a-addr)	指定されたデータバッファーを作成します。*name* は a-addr を返します。
`constant` *name*	(x --) (E: -- x)	定数 (たとえば、`3 constant bar`) を作成します。
`2constant` *name*	(x1 x2 --) (E: -- x1 x2)	2 つの数値の定数を作成します。
`create` *name*	(--) (E: -- a-addr)	別のコマンドにより設定される動作を行う新しいコマンドを作成します。

表 4-14 ワード定義 (続き)

コマンド	スタックダイアグラム	説明
`$create`	(name-str name-len --)	name-string によって指定される名前を使用して `create` を実行します。
`defer` *name*	(--) (E: … -- ???)	別の動作を行うコマンドを作成します。`to` で変更します。
`does>`	(… -- … a-addr) (E: … -- ???)	`create` で作成されたワードの実行時の動作を指定します。
`field` *name*	(offset size -- offset+size) (E: addr -- addr+offset)	指定された *name* でフィールドオフセットポインタを作成します。
`struct`	(-- 0)	`struct…field` 定義を開始します。
`value` *name*	(x --) (E: -- x)	指定された変数を作成します。`to` で変更します。
`variable` *name*	(--) (E: -- a-addr)	指定された変数を作成します。*name* は a-addr を返します。

`value` では名前に変更可能な任意の数値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 という初期値を `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 `to` 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ワードは、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 1 セル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。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、異なる機能を読み込むスロットが生成され、後で機能を変更できるワードを生成することができます。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 4-15 に辞書検索用のツールを示します。これらのツールの一部は、方法またはコマンドだけを検索処理の対象とするのに対して、その他のツールは (たとえば、変数や値をはじめとする) すべての種類のワードを対象とすることに注意してください。

表 4-15 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* を使用するすべてのコマンドのリストを表示します。
`$find`	(str len -- xt true | str len false)	str len によって指定されるワードを探します。見つかった場合は、xt と true をスタックに残します。見つからなかった場合は、名前の文字列と false をスタックに残します。
`find`	(pstr -- xt n | pstr false)	pstr によって指定されるワードを探します。見つかった場合は、xt と true をスタックに残します。見つからなかった場合は、名前の文字列と false をスタックに残します。 (パックされた文字列を使用しないようにするため、`$find` を使用するよう推奨します)。
`see` *thisword*	(--)	指定されたワードを逆コンパイルします。
`(see)`	(xt --)	実行トークンが xt であるワードを逆コンパイルします。
`$sift`	(text-addr text-len --)	text-string を含むすべてのコマンド名を表示します。
`sifting` *text*	(--)	*text* を含むすべてのコマンド名を表示します。*text* 内には空白文字はありません。
`words`	(--)	次に説明するように辞書内のすべてのワードを表示します。

辞書検索ツールの動作を理解するには、その前に、ワードがどのようにして**見える**ようになるかを理解する必要があります。ワードを定義するときに**アクティブパッケージ**があった場合は、新しいワードはその有効パッケージの方法になり、そのパッケージが有効のときしか見えません。`dev`、`find-device` というコマンドを使用して、有効パッケージを選択したり、変更することができます。コマンド `device-end` は、現在有効なパッケージの選択を解除して、有効パッケージをなくします。

ワードを定義するときにアクティブパッケージがなかった場合は、そのワードは**グローバルに見る**ことができます。(つまり、特定のパッケージ専用ではなく、常に使用可能です)。

辞書検索コマンドは、存在する場合、最初に有効パッケージのワードに対して検索し、次にグローバルに見えるワードに対して検索します。

注 – Forth コマンドの `only` と `also` が、どのワードを見えるようにするかを制御します。

`.calls` を使用して、それぞれの定義に指定されたワードを使用しているすべての Forth コマンドを検索できます。`.calls` はスタックから実行トークンを取り出し、辞書全体を検索して、その実行トークンを使用しているすべての Forth コマンドの名前とアドレスのリストを生成します。次の例を参照してください。

```
ok ' input .calls
    Called from input at 1e248d8
    Called from io at 1e24ac0
    Called from install-console at 1e33598
    Called from install-console at 1e33678
ok
```

`see` は、

`see` *thisword*

の形式で使用すると、*thisword* のソースの「プリティプリント」リストを表示します。次の例を参照してください。

```
ok see see
: see
     ' ['] (see) catch if
     drop
     then
;
ok
```

see の使用上の詳細は、 101 ページの「Forth 言語逆コンパイラの使用方法」を参照してください。

sifting は入力ストリームから文字列を取り出し、辞書の検索順にすべての語を調べ、次の画面に示すように、指定された文字列を含むすべてのコマンド名を見つけます。

```
ok sifting input

     In vocabulary options
(1e333f8) input-device
     In vocabulary forth
(1e2476c) input(1e0a9b4) set-input(1e0a978) restore-input
(1e0a940) save-input(1e0a7f0) more-input?(1e086cc) input-file
ok
```

words は辞書内のすべての見えるワード名を、最も新しい定義から先に表示します。ノードが現在 (たとえば dev で) 選択されている場合は、words によって生成されるリストはその選択されているノードのワードだけに限定されます。

データを辞書へコンパイルする

表 4-16 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 4-16 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。(代わりに postpone を使用するよう推奨します。)
[compile] name	(--)	次の (すぐ次の) ワードをコンパイルします。(代わりに postpone を使用するよう推奨します。)
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。
patch *new-word old-word word-to-patch*	(--)	*old-word* を *word-to-patch* の*new-word* に置き換えます。
(patch)	(new-n old-n xt --)	*old-n* を *xt* によって示されるワードの *new-n* に置き換えます。

表 4-16 辞書コンパイルコマンド (続き)

コマンド	スタックダイアグラム	説明
postpone *name*	(--)	ワード *name* の実行を遅らせます。
recurse	(... -- ???)	コンパイル中のワードへの再帰呼び出しをコンパイルします。
recursive	(--)	辞書内のコンパイル中のコロン定義の名前を見るようにし、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。

表 4-17 に示す辞書コンパイル用コマンドは 64 ビットの OpenBoot 実装専用です。

表 4-17 64 ビット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
x,	(o --)	octlet、o をコンパイルして辞書に入れます (doublet 境界)。

数値の表示

表 4-18 にスタック値表示用の基本コマンドを示します。

表 4-18 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 ロンプトの前でのスタック表示をオフにします。
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

数値を特定の基数で出力することや、表 4-19 に示すコマンドを利用して演算の基数を変更することもできます。

表 4-19 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。基数は変更されません。
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。

d# 、h# 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。

表 4-20 にテキスト入力制御用のコマンドを示します。

表 4-20 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。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。
accept	(addr len1 -- len2)	コンソールの入力デバイスから編集された入力行を獲得し、*addr* に格納します。len1 は許容される最大長です。len2 は実際に受け取られる長さです。
expect	(addr len --)	コンソールから入力行を獲得して表示し、*addr* に格納します。(代わりに accept を使用するよう推奨します。)
key	(-- char)	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。
parse	(char -- str len)	入力バッファーからの、char で区切られたテキストを構文解析します。
parse-word	(-- str len)	入力バッファーからの、空白文字で区切られたテキストを、先行空白文字を読み飛ばして構文解析します。
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。(代わりに parse を使用するよう推奨します。)

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、81 ページの「条件フラグ」の説明を参照してください。

表 4-21 に汎用のテキスト表示用コマンドを示します。

表 4-21 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 --)	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 (space) を表示します。
spaces	(+n --)	+*n* 個の空白文字を表示します。
type	(addr +n --)	*addr* から始まる +*n* 個の文字を表示します。

cr は改行処理を出力デバイスに送ります。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 4-22 にテキスト文字列操作用のコマンドを示します。

表 4-22 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。
." *ccc*"		文字列 *ccc* を表示します。
.(ccc)	(--)	文字列 *ccc* 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)。
pack	(addr len pstr -- pstr)	文字列 addr len をパックされた文字列として *pstr* に格納します。
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-length (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、インタプリンタに遭遇するとただち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
```

コロン定義の外部に " を使用すると、それぞれ最高 80 文字までの解釈済み文字列を 2 つしか同時にアセンブルできません。この制限はコロン定義には当てはまりません。

入出力先の変更

通常、OpenBoot ではコマンド入力にキーボードを、また表示出力にはディスプレイ画面付きのフレームバッファーを使用します。(サーバーシステムはシリアルポートに接続された ASCII 端末を使用します。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、フレームバッファーのデバッグ時に便利です。

表 4-23 に入出力先変更用のコマンドを示します。

表 4-23 入出力先変更用コマンド

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、たとえば ttya、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、たとえば ttya、keyboard、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の再投入を行うと、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「システム変数の設定」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、ttyb または *device-specifier* 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、シリアルポートに出力を送る場合は、次のように入力します。

```
ok ttya output
ok
```

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**、シリアルポートに接続されている端末が ok プロンプトと以降のすべての出力を表示されます。

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。

```
ok ttya io
ok
```

一般的に、input、output、io の引数には *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
```

上の 2 例では、keyboard、screen、ttya、ttyb は定義済みの 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) コマンド行エディタ、いくつかのオプションで使用する拡張機能、および履歴用機能を実装しています。これらのツールを使用して、前のコマンドを入力し直さずに再び実行して、現在のコマンド行を編集して入力エラーを修正し、また前のコマンドを呼び出して変更することができます。

表 4-24 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 4-24 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
Return (Enter)	現在の行の編集を終了し、カーソルの現在の位置に関係なく、表示されている 1 行全部をインタプリタに渡します。
Control-B	1 文字位置戻ります。
Escape B	1 語戻ります。
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。
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	カーソル位置の次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) 再び実行することができます。表 4-25 にコマンド行履歴拡張用のキーを示します。

表 4-25 コマンド行履歴用キー操作コマンド

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	コマンド履歴バッファーすべてを表示します。

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。

コマンド補完拡張用のキーを示します。

表 4-26 コマンド補完用キー操作コマンド

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。

表 4-27 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 4-27 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)。
`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。

表 4-27 比較コマンド (続き)

コマンド	スタックダイアグラム	説明
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、else、then 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 4-28 に条件付き実行フロー制御用のコマンドを示します。

表 4-28 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	(--)	flag 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)
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)
else
     (false 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 コマンドが、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 if…then コマンドよりも読みやすいという利点があります。

表 4-29 に条件付き case コマンドを示します。

表 4-29 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 --)	case…endcase 条件付き構造を終了します。
endof	(--)	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。

次に 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、その値を削除しないでください (. 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 4-30 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 4-30 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。

begin…until 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。フラグが true の場合は、実行はループから抜け出ます。

begin…while…repeat 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」と覚えてください。

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
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、この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 4-31 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 4-31 `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、`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*-1 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`	(--)	`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
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 4-32 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 4-32 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 --)	addr len 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「プログラムの読み込みと実行」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。

第5章

プログラムの読み込みと実行

ユーザーインタフェースではプログラムをマシンに読み込み、実行するいくつかの方法を提供しています。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 ttya を介して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。

これらの方法の多くは、Client プログラムヘッダーを持つファイルを必要とします。詳細については、IEEE 1275.1-1994『Standard for Boot (Initializatin Configuration) Firmware』を参照してください。このヘッダーは多くの UNIX システムで使用される `a.out` ヘッダーに似ています。サンの FCode トークン生成プログラムは、この Client プログラムヘッダーをファイルを生成します。

表 5-1 に、いろいろなソースからファイルを読み込む OpenBoot 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と拡張機能

コマンド	スタックダイアグラム	説明
`boot` *[device-specifier] [arguments]*	(--)	各種のシステム変数、(省略可能な) 引数の値に従って、使用するファイル、デバイスを決定します。マシンをリセットし、指定されたデバイスから、指定されたプログラムを読み込んで実行します。
`byte-load`	(addr xt --)	addr から始まる FCode を解釈します。xt が 1 の場合 (通常の場合) は、`rb@` を使用して FCode を読みます。そうでない場合は、実行トークンが xt であるアクセスルーチンを使用します。
`dl`	(--)	「Control-D」が押されるまで、シリアルラインを通して Forth ソーステキストファイルを読み込み、次に解釈します。たとえば `tip` を使用する場合は次のように入力します。 **`~C cat`** *filename* **`Control-D`**
`dlbin`	(--)	シリアルラインからバイナリファイルを読み込みます。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 を介し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`	(… str len -- ???)	`evaluate` の同義語。
`evaluate`	(… str len -- ???)	指定された文字列からの Forth ソーステキストを解釈します。
`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マシンを準備します。

表 5-1　ファイル読み込み用コマンドと拡張機能 (続き)

コマンド	スタックダイアグラム	説明
load [*device-specifier*] [*arguments*]	(--)	各種のシステム変数、(省略可能な) 引数の値に従って、使用するファイル、デバイスを決定し、指定されたデバイスから指定されたプログラムを読み込みます。
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。
?go	(--)	Forth、FCode、またはバイナリプログラムを実行します。

boot の使用方法

boot は通常はオペレーティングシステムを起動するために使用しますが、任意のクライアントプログラムを読み込み、実行する場合にも使用できます。起動は通常は自動的に行われますが、ユーザーがユーザーインタフェースから開始することもできます。

boot プロセスは次のとおりです。

- 最後のリセット後にクライアントプログラムを実行している場合は、マシンがリセットされることがあります。(リセットが実行されるかどうかは、実装によります。)
- デバイスは、boot コマンド行を読み取って起動デバイスと起動引数に従って選択されます。boot コマンドの形式によっては、起動デバイスや起動引数の値はシステム変数から獲得されます。
- デバイスツリー上の /chosen ノードの bootpath、bootargs 特性には選択された値が設定されます。
- 選択されたプログラムが、選択されたデバイスのタイプによって決まるプロトコルに従ってメモリーに読み込まれます。たとえば、ディスク起動の場合はディスクの先頭から特定の数のブロックが読まれ、テープ起動の場合は特定のテープファイルが読まれることになります。
- 読み込まれたプログラムが実行されます。プログラムの動作は、(存在する場合) 選択されたシステム変数に設定されていたか、コマンド行で boot コマンドに渡された引数文字列によってさらに制御されることがあります。

boot の一般的な形式は次のとおりです。

```
boot [device-specifier] [arguments]
```

ただし、*device-specifier* と *arguments* は省略可能です。boot コマンドの使用に関する詳細については、15 ページの「上級ユーザー向けの起動」を参照してください。

dl を使ってシリアルポート A から Forth テキストファイルを読み込む

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、テストシステムのシリアルポート ttya を、要求があればすぐにファイルを転送できるマシン (つまり、サーバー) に接続し、そのサーバー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、Solaris 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「TIP 接続の設定」を参照してください。

1. テストシステムの ok プロンプトで次のように入力します。

```
ok dl
```

2. サーバーの tip ウィンドウから、次のように入力します。

   ~C

   これによりコマンド行を表示します。このコマンド行を使用して Solaris コマンドをサーバーに発行します。

注 – C は大文字で入力する必要があります。

注 – tip は、~ がコマンド行上の最初の文字の場合にのみ、この文字を tip コマンドとして認識します。tip が ~C を認識できなかった場合は、tip ウィンドウに Enter キーを入力し、もう一度 ~C を入力します。

3. local command プロンプトで cat を使用してファイルを転送します。

```
~C (local command) cat filename
(Away two seconds)
Control-D
```

4. tip が (Away n seconds) の形式のメッセージを表示したら、tip ウィンドウに

   Control-D

   を入力して、ファイルの終わりに達したことを dl に知らせます。

   dl はそこで自動的にファイルを解釈し、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。

load の使用方法

load の構文と動作は、プログラムが読み込まれるだけで、実行はされない点を除いて boot と同じです。load も、boot の場合と異なり、読み込む前にマシンをリセットしません。

load コマンドの一般的な形式は次のとおりです。

```
load [device-specifier] [arguments]
```

load コマンドの変数の解析は、boot の場合と同じシステム変数によって制御され、load の *device-specifier* と *arguments* は同じプロセスで識別されます。(詳細は、15 ページの「上級ユーザー向けの起動」を参照してください。)

device-specifier と *arguments* が識別できたら、読み込みが次のように行われます。

1. *device-specifier* と *arguments* が /chosen ノードのそれぞれ bootpath と bootargs 特性に保存されます。

2. *device-specifier* がシステム変数から得られた場合は、その値はデバイスのリストであることがあります。リストのエントリが 1 つだけであれば、`load` はそのエントリを *device-specifier* として使用します。

注 – リストのエントリが複数の場合は、リスト上の最初のエントリから最後のエントリの 1 つ前までの各デバイスが開こうとします。システムがデバイスを開けると、そのデバイスは閉じられ、`load` がそれを *device-specifier* として使用します。リスト上のどのデバイスも開けなかった場合は、`load` はリストの最後のデバイスを *device-specifier* として使用します。

---

3. `load` が *device-specifier* によって指定されるデバイスを開こうとします。そのデバイスが開けなかった場合は、読み込みは終了されます。

4. デバイスが開けた場合は、その `load` 方法が呼び出されて、指定されたプログラムが指定されたデバイスからシステムのデフォルト読み込みアドレスに読み込まれます。

5. 正常に読み込みができて、読み込まれたイメージの先頭がシステムの有効なクライアントプログラムヘッダーであれば、次の操作が行われます。

   a. そのヘッダーに指定されているアドレスに指摘されたサイズのメモリーが割り当てられます。

   b. 読み込まれたイメージがデフォルト読み込みアドレスから新しく割り当てられたメモリーに転送されます。

   c. `go` コマンドによって読み込まれたプログラムが実行できるようにしてシステムは初期化されます。後に続く `go` コマンドが読み込まれたプログラムの実行を開始するためにシステムが初期化されます。

dlbin を使って FCode またはバイナリ実行ファイルをシリアルポート A から読み込む

`dlbin` で読み込む FCode プログラムまたはバイナリプログラムは、`Client プログラムヘッダー`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 `Client プログラムヘッダー`で示されるエントリポイントに読み込みます。バイナリ

ファイルを 4000 (16 進数) にリンクします。最新バージョンの 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 Client プログラムヘッダーファイルを作成します。

シリアルラインを通してファイルを読み込むには、要求に応じてファイルを転送できるマシン (つまり「サーバー」) にテストシステムのシリアルポート ttya を接続します。サーバーの端末エミュレータを起動し、その端末エミュレータを使用して dlbin を使用するファイルをダウンロードします。

次の例では、Solaris 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「TIP 接続の設定」を参照してください。)

1. テストシステムの ok プロンプトで、次のように入力します。

```
ok dlbin
```

サーバーの tip ウィンドウで次のように入力します。

```
~C
```

これによりコマンド行を表示します。このコマンド行を使用してサーバーに UNIX コマンドを発行できます。

注 – C は大文字で入力する必要があります。

注 – tip は、~ がコマンド行上の最初の文字の場合、この文字を tip コマンドとして認識します。tip が ~C を認識できなかった場合は、tip ウィンドウに Enter を入力し、もう一度 ~C を入力します。

---

1. "local command" プロンプトで、cat を使用してファイルを転送します。

```
~C (local command) cat filename
(Away two seconds)
```

ダウンロードが終了すると、tip が (Away *n* seconds) の形式のメッセージを表示し、テストシステム側の画面に再び 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
```

ダウンロードしたプログラムを実行するには、次のように入力します。

```
ok go
```

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 Client プログラムヘッダー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 は、Client プログラムヘッダーにバイナリプログラムがあることを必要とします。実行可能バイナリは、dload の入力アドレス (4000 など) にリンクされているか、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、Client プログラムヘッダーファイル内のシンボル情報はすべてユーザー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「デバッグ」を参照してください)。

?go の使用方法

プログラムをシステムに読み込むことができたら、?go を使用してプログラムのタイプにかかわりなくそのプログラムを実行できます。

?go は読み込まれたイメージの先頭を調べます。イメージの先頭が文字列 "\" (バックスラッシュと空白文字) である場合は、イメージは Forth テキストとみなされます。インタプリタが呼び出されてイメージを解釈します。

第6章

デバッグ

OpenBoot では、Forth 言語逆コンパイラ、マシン言語逆アセンブラ、レジスタ表示用コマンド、シンボリックデバッガ、ブレークポイント関係コマンド、Forth ソースレベルデバッガ、高水準言語パッチ機能、例外追跡を含むデバッグ用ツールを提供しています。この章では、IEEE Standard 1275-1994 に指定された機能について説明します。

Forth 言語逆コンパイラの使用方法

組み込み Forth 言語逆コンパイラを使用して、どのような定義済み Forth ワードのソースコードでも表示できます。

```
ok see old-name
```

このコマンドは、`old-name` のソースの (ソースコメントなしの) リストを表示します。

`see` は `(see)` と対をなしています。つまり、`(see)` の機能は Forth ワードの逆コンパイルを行い、その実行トークンはスタックから取り出されます。次の例を参照してください。

```
ok ' old-name (see)
```

`(see)` は `see` と同じ書式のリストを生成します。

```
ok see see
: see
   ' ['] (see) catch if
      drop
   then
;
ok see (see)
defer (see) is
: (f0018a44)
   40 rmargin ! dup dup (f00189c4) dup (f0018944) (f0018980) (f0018658)
   ??cr
;
ok f0018a44 (see)
: (f0018a44)
   40 rmargin ! dup dup (f00189c4) dup (f0018944) (f0018980) (f0018658)
   ??cr
;
```

上記のリストは、次のことを示しています。

- see そのものは、fcode-debug? が true に設定されて external または headers としてコンパイルされた Forth ソースワードだけで構成されています。
- (see) は据え置きワードです。(see) 内には、headerless としてコンパイルされた、したがって、括弧で囲まれた 16 進アドレスとして表示されているワードもあります。
- (see) でワードを逆コンパイルすると、see が生成するものと同じリストが生成されます。

Forth アセンブラ言語に実装されているワードについては、see は Forth アセンブラのリストを表示します。たとえば、dup を逆コンパイルすると、次のように表示されます。

```
ok see dup
code dup
f0008c98     sub         %g7, 8, %g7
f0008c9c     stx         %g4, [%g0 + %g7]
f0008ca0     ld          [%g5], %l0
f0008ca4     jmp         %l0, %g2, %g0
f0008ca8     add         %g5, 4, %g5
```

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、対応するアセンブラ言語に翻訳します。

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
- 逆アセンブル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逆アセンブルが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、106 ページの「ブレークポイント」で説明します。) こういった場合には、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これらの値は調べたり、変更することができます。

SPARC レジスタ

表 6-2 に SPARC のレジスタ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
`%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 %y`	(-- value)	指定されたレジスタの値を返します。
`%f0 ～ %f31`	(-- value)	指定された浮動小数点レジスタの値を返します。
`.fregisters`	(--)	`%f0` から `%f31` までの値を表示します。
`.locals`	(--)	`i`、`l`、`o` レジスタの値を表示します。
`.registers`	(--)	プロセッサレジスタの値を表示します。
`.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に格納された値を変更します。*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。

表 6-3　SPARC V9 レジスタコマンド

コマンド	スタックダイアグラム	説明
%fprs	(-- value)	指定されたレジスタの値を返します。
%asi		
%pstate		
%tl-c		
%pil		
%tstate		
%tt		
%tba		
%cwp		
%cansave		
%canrestore		
%otherwin		
%wstate		
%cleanwin		
.pstate	(--)	プロセッサ状態レジスタを特定書式で表示します。
.ver	(--)	バージョンレジスタを特定書式で表示します。
.ccr	(--)	%ccr レジスタを特定書式で表示します。
.trap-registers	(--)	トラップレジスタを表示します。

これらのレジスタの値はすべて保存され、to で変更できます。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。)

SPARC V9 システムでは、N が現在のウィンドウの場合、N-1 は呼び出し元のウィンドウを指定し、N-2 は呼び出し元の呼び出し元を指定します。

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 6-4 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 6-4 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`	(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`	(--)	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 to .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。
`.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。

表 6-4　ブレークポイントコマンド (続き)

コマンド	スタックダイアグラム	説明
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設定することにより、任意のアドレスに移ることができます。
gos	(n --)	go を n 回実行します。
hop	(--)	(step コマンドに似ています。) 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へ読み込みます。
2. 詳細は、第 5 章「プログラムの読み込みと実行」を参照してください。各レジスタの値が自動的に初期化されます。
3. (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。
4. 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。
5. さらに、ブレークポイントを設定し、実行したり (たとえば、*addr* +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。

Forth ソースレベルデバッガ

ソースレベルデバッガでは、Forth プログラムのシングル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 6-5 にこのデバッガのコマンドを示します。

表 6-5　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。
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。
f	下位の Forth インタプリタを起動します。通常はこのインタプリタで Forth コマンドを実行できます。このインタプリタが (resume) で終了されると、制御がデバッガの f コマンドが実行された位置に戻ります。
g	"Go"。デバッガをオフに設定し、プログラムの実行を継続します。
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。
s	"see"。デバッグ中のワードを逆アセンブルします。
$	スタックの一番上の address,len をテキスト文字列として表示します。
h	"Help"。シンボリックデバッガのマニュアルを表示します。
?	"Short Help"。簡略なシンボリックデバッガのマニュアルを表示します。
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	(debug は、入力ストリームから名前を取り出すのでなく、スタックから実行トークンを取り出す点を除いて debug と同じです。
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、制御をデバッガのステップ実行に戻します (この表の f コマンドを参照)。

表 6-5　Forth ソースレベルデバッガコマンド (続き)

コマンド	説明
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 6-5 にあるキー操作のどれかを求められます。

patch と (patch) の使用方法

OpenBoot では、アセンブル済みの Forth ワードの定義を高水準の Forth 言語を使用して変更できます。変更は、通常、該当するソースコードで行われるのに対して、patch 機能はデバッグ時に見つけられなかった誤りを迅速に修正する手段を提供します。

patch は次に示す各情報から入力ストリームを読みます。

- 挿入される新しいコードの名前
- 置き換えられる古いコードの名前
- 古いコードが入っているワードの名前

たとえば、次の例を考えてください。この例では、ワード test が数値 555 に置き換えられます。

```
ok : patch-me  test 0 do  i . cr  loop ;
ok patch 555 test patch-me
ok see patch-me
: patch-me
    h# 555 0 do
    i . cr
    loop
;
```

patch を使用するときは、十分注意して正しいワードを選択し、置き換える必要があります。置き換えようとするワードがターゲットワード内で数回使用され、ターゲットワード内で最初に出てくるものではなく、何番目かに出てくるワードである場合は、特に注意が必要です。そのような場合は、なんらかの回避手段が必要です。

```
ok : patch-me2  dup dup dup ( This third dup should be drop) ;
ok : xx dup ;
ok patch xx dup patch-me2
ok patch xx dup patch-me2
ok patch drop dup patch-me2
ok see patch-me2
: patch-me2
    xx xx drop
;
```

patch のもう 1 つの用途は、パッチするワードに、完全に廃棄する必要がある特定の機能がある場合です。そのような場合は、機能を削除する最初のワードに対してワード exit をパッチします。たとえば、次のような定義をもつワードを考えてみてください。

```
ok : foo good bad unneeded ;
```

この例では、bad の機能は誤っており、unneeded の機能は廃棄する必要があります。

```
ok : right this that exit ;
ok patch right bad foo
```

foo をパッチしようとするとき、最初は、ワード right 内に exit を使用すれば unneeded を実行できなくできると期待して、次のように入力するかもしれません。exit は、それが入っているワード、この場合 right の実行を終了させてしまいます。foo の正しいパッチ方法は次のようになります。

```
ok : right this that ;
ok patch right bad foo
ok patch exit unneeded foo
```

(patch) は、その引数をスタックから得る点を除いて patch と同じです。(patch) のスタックダイアグラムは次のとおりです。

```
( new-n1 num1? old-n2 num2? xt -- )
```

ただし、

- new-n1 と old-n2 は実行トークン、数値のどちらでも差し支えありません。
- num1? と num2? はフラグであり、それぞれ、new-n1 または old-n2 が数値であるかどうかを示します。
- xt はパッチを実行するワードの実行トークンです。

たとえば、次の例を考えてみてください。ここでは、数値 555 を test に置き換えて、最初の patch の例の効果を逆にしています。

```
ok see patch-me
: patch-me
    h# 555 0 do
    i . cr
    loop
;
ok ['] test false 555 true ['] patch-me (patch)
ok see patch-me
: patch-me
    test 0 do
    i . cr
    loop
;
```

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。

最後の数行は、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

TIP 接続の設定

SPARC システムの ttya または ttyb ポートを使用して別のサンのワークステーションに接続することができます。このように 2 つのシステムを接続することにより、サンのワークステーションのシェルウィンドウを SPARC システムの端末として使用できます。(リモートホストへの端末接続についての詳細は、tip(1) のマニュアルページを参照してください。)

この TIP を用いる方法は、起動 PROM を使用する際にウィンドウ機能やオペレーティングシステムの機能を利用できるのでお勧めできます。通信プログラムやその他サン以外のコンピュータも、PROM の tty ポートで使用される出力ボーレートにプログラムを合わせることができれば同様に使用できます。

注 – これ以降、「SPARC システム」とは使用するシステム、「サンのワークステーション」とは使用するシステムに接続するシステムのことを指します。

次の手順に従って、TIP 接続を設定します。

1. シリアル接続ケーブルを使用して、サンのワークステーションの ttyb シリアルポートを SPARC システムの ttya シリアルポートに接続します。3 芯のヌルモデムケーブルを使用し、3-2、2-3、7-7 のワイヤ接続を行ってください。(ヌルモデムケーブルの仕様については、システムインストールマニュアルを参照してください。)

2. サンのワークステーションから、etc/remote ファイルに次に示すような行を追加します。

   2.0 バージョンより前の Solaris 環境を実行する場合は、次のように入力します。

```
hardwire:\
    :dv=/dev/ttyb:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
```

   バージョン 2.x の Solaris 環境を実行する場合は、次のように入力します。

```
hardwire:\
    :dv=/dev/term/b:br#9600:el=^C^S^Q^U^D:ie=%$:oe=^D:
```

3. サンのワークステーションのシェルツールウィンドウから次のように入力します。

```
hostname% tip hardwire
connected
```

   これで、シェルツールウィンドウはサンのワークステーションの ttyb からの TIP ウィンドウになりました。

注 – コマンドツールでなくシェルツールを使用してください。コマンドツールウィンドウでは、一部の TIP コマンドが正しく動作しない場合があります。

---

4. SPARC システムで Forth モニターを起動して、ok プロンプトを表示させます。

注 – SPARC システムにビデオモニターを接続していない場合は、SPARC システムの ttya をサンのワークステーションの ttyb に接続し、SPARC 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、Forth モニターを起動します。システムが完全に動作不可でさえなければ、Forth モニターは使用可能になり、この手順の次の手順に進むことができます。

---

5. 標準入出力先を ttya に変更するには、次のように入力します。

```
ok ttya io
```

画面には応答がエコーされません。

6. サンの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 – SPARC システムの TIP ウィンドウとして使用しているサンの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、ワークステーションのオペレーティングシステムが強制終了されます。(誤って Stop-A と入力した場合は、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。)

---

7. 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 ウィンドウを終了します。

8. 必要な場合は次のように入力して、入出力先をそれぞれキーボードと画面に変更します。

```
ok screen output keyboard input
```

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にするには、まず Return キーを押します。

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)` の 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` ファイルを調べてください。

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、起動可能なフロッピーディスクの作成に必要な手順を簡単に示します。OS のコマンドについては、マニュアル (man) ページを参照してください。また、特定のファイルとファイルシステム内のそれらの位置については、該当する OS リリースを参照してください。

1. フロッピーディスクをフォーマットします。

   fdformat コマンドはフロッピーディスクフォーマット用のユーティリティーです。

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。

   可能な場合は、newfs コマンドを使用して作成できます。

3. フロッピーディスクを一時パーティションにマウントします。

   可能な場合は、mount コマンドを使用してマウントできます。

4. cp コマンドを使用して、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。

   たとえば、boot と ufsboot は第 2 レベルの起動プログラムです。

5. 起動ブロックをフロッピーディスクにインストールします。

   可能な場合は、installboot コマンドを使用してインストールできます。

6. cp コマンドを使用して、起動するファイルをマウントしたフロッピーディスクにコピーします。

7. 可能な場合は、umount を使用してフロッピーディスクをマウント解除します。

8. これで、フロッピーディスクをドライブから取り外せます。

   可能な場合は、eject floppy を使用します。

付録C

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、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、Sun Ultra™ 1 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 true の場合だけ ttya に表示されます。

注 – 実際の OpenBoot 初期設定処理はシステムによって異なります。使用するシステムで表示されるメッセージは異なる場合があります。

コード例 C-1　　OpenBoot の初期化処理

```
...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。)
Probing Memory Bank #0 16 + 16 : 32 Megabytes     (メモリーをプローブする)
Probing Memory Bank #1 0 + 0 : 0 Megabytes
Probing Memory Bank #2 0 + 0 : 0 Megabytes
Probing Memory Bank #3 0 + 0 : 0 Megabytes
                             (use-nvramrc? が true の場合、ファームウェアが NVRAMRC コマンドを実行し、次に Stop-x コマンドの有無を調べ、デバイスをプローブする。そこでキーボードの LED が点滅する。)
Probing UPA Slot at le,0 Nothing there   (デバイスをプローブする)
Probing /sbus@lf,0 at 0,0 cgsix
Probing /sbus@lf,0 at 1,0 Nothing there
Probing /sbus@lf,0 at 2,0 Nothing there
Sun Ultra 1 UPA/SBus (UltraSPARC 167 MHz), Keyboard Present   (バナーを表示する)
OpenBoot 3.0, 32 MB memory installed, Serial #7570016
Ethernet address 8:0:20:73:82:60, Host ID: 80738260.
ok boot disk3
Boot device: /sbus/espdma@e,8400000/esp@e,8800000/sd@3,0  (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後、制御が起動プログラムへ移される。)
FCode UFS Reader 1.8 01 Feb 1995 17:07:00,IEEE 1275 Client Interface.  (起動プログラムが実行を開始する。)
Loading: /platform/sun4u/ufsboot
cpu0: SUNW,UltraSPARC (upaid 0 impl 0x0 ver 0x0 clock 143 MHz)
SunOS Release 5.5 Version quick_gate_build:04/13/95 (UNIX(R) System V Release 4.0)
                        (このメッセージが表示された後、制御がカーネルへ移される。)
Copyright (c) 1983-1995, Sun Microsystems, Inc.  (カーネルが実行を開始する。)
DEBUG enabled      (カーネルメッセージが続く。)
```

緊急時の手順

一部の OpenBoot システムはシステムのキーボード上でキーを組み合わせて押すことによって、OpenBoot のコマンド機能を提供します。

表 C-1 に、SPARC 互換システムが提供するキーボードコー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 C-1　SPARC 互換キーボードコード

コマンド	説明
Stop	POST をします。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	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`」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。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、追加型フレームバッファーが 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。126 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が、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を次に示します。

- NVRAM の変数 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 の変数 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 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。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を次に示します。

- 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障害追跡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
```

   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 2 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 2、3、4 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 ファームウェアは常にコンソールを NVRAM の変数 `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 /sbus/cgsix
ok setenv output-device myscreen
ok reset-all
```

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）
```

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 2013
ok reset-all
```

Sbus フレームバッファー以外のバッファーがインストールされている場合、2 つ目の方法は有効ではありません。

付録 D

Sun Ultra 5/10 UPA/PCI システム

この付録では、この PCI バスをベースにしたシステムとサンの SBus をベースにしたシステムで異なる点について説明します。

PCI ベースのシステム

banner コマンドの出力は、次のように表示され、PCI ベースのシステムであることが示されます (バナーに UPA/PCI と表示されます)。

```
ok banner
Sun Ultra 5/10 UPA/PCI (UltraSPARC-IIi 300MHz), Keyboard Present
OpenBoot 3.11, 32 MB memory installed, Serial #8812498.
Ethernet address 8:0:20:86:77:d2, Host ID: 808677d2.
```

show-devs コマンドの出力には、PCI ベースのノードが表示されます。また、PCI ベースのシステムでは、デバイスの汎用的な名前が使用されます。オンボードのネットワークは "network" という名前で、内蔵ディスクは "diskn" (n はそのディスクの SCSI ターゲット番号) という名前です。

独自の FCodePROM を搭載した追加 PCI カードは、汎用的な名前を使用している場合とそうでない場合があります。汎用的な名前についての詳細は、Open Firmware Working Group のホームページ (http://playground.sun.com/1275)の「Recommended Practices」を参照してください。

```
ok show-devs
/SUNW,UltraSPARC-IIi@0,0
/pci@1f,0
/virtual-memory
/memory@0,0
/aliases
/options
/openprom
/chosen
/packages
/pci@1f,0/pci@1
/pci@1f,0/pci@1,1
/pci@1f,0/pci@1,1/ide@3
/pci@1f,0/pci@1,1/SUNW,m64B@2
/pci@1f,0/pci@1,1/network@1,1
/pci@1f,0/pci@1,1/ebus@1
/pci@1f,0/pci@1,1/ide@3/cdrom
/pci@1f,0/pci@1,1/ide@3/disk
/pci@1f,0/pci@1,1/ebus@1/SUNW,CS4231@14,200000
/pci@1f,0/pci@1,1/ebus@1/flashprom@10,0
/pci@1f,0/pci@1,1/ebus@1/eeprom@14,0
/pci@1f,0/pci@1,1/ebus@1/fdthree@14,3023f0
/pci@1f,0/pci@1,1/ebus@1/ecpp@14,3043bc
/pci@1f,0/pci@1,1/ebus@1/su@14,3062f8
/pci@1f,0/pci@1,1/ebus@1/su@14,3083f8
/pci@1f,0/pci@1,1/ebus@1/se@14,400000
/pci@1f,0/pci@1,1/ebus@1/SUNW,pll@14,504000
/pci@1f,0/pci@1,1/ebus@1/power@14,724000
/pci@1f,0/pci@1,1/ebus@1/auxio@14,726000
/openprom/client-services
/packages/sun-keyboard
/packages/SUNW,builtin-drivers
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
```

Sun Ultra 5/10 UPA/PCI システムでの devalias コマンドの出力は次のとおりです。

```
ok devalias
screen                   /pci@1f,0/pci@1,1/SUNW,m64B@2
net                      /pci@1f,0/pci@1,1/network@1,1
cdrom                    /pci@1f,0/pci@1,1/ide@3/cdrom@2,0:f
disk                     /pci@1f,0/pci@1,1/ide@3/disk@0,0
disk3                    /pci@1f,0/pci@1,1/ide@3/disk@3,0
disk2                    /pci@1f,0/pci@1,1/ide@3/disk@2,0
disk1                    /pci@1f,0/pci@1,1/ide@3/disk@1,0
disk0                    /pci@1f,0/pci@1,1/ide@3/disk@0,0
ide                      /pci@1f,0/pci@1,1/ide@3
floppy                   /pci@1f,0/pci@1,1/ebus@1/fdthree
ttyb                     /pci@1f,0/pci@1,1/ebus@1/se:b
ttya                     /pci@1f,0/pci@1,1/ebus@1/se:a
keyboard!
/pci@1f,0/pci@1,1/ebus@1/su@14,3083f8:forcemode
keyboard                 /pci@1f,0/pci@1,1/ebus@1/su@14,3083f8
mouse                    /pci@1f,0/pci@1,1/ebus@1/su@14,3062f8
name                     aliases
```

.speed コマンドは、システムに接続されたプロセッサおよびバスの速度を表示します。

```
ok .speed
CPU  Speed : 300.00MHz
UPA  Speed : 100.00MHz
PCI  Bus A : 33Mhz
PCI  Bus B : 33Mhz
```

PCI バスの pcia と pcib

Sun Ultra 5/10 UPA/PCI システムには pcia および pcib の 2 つの PCI バスがあります。これらのバスのスロットをプローブする方法は、次の 2 つの NVRAM 設定変数により指定します。

表 D-1　PCI スロット

変数名	デフォルト値	説明
pcia-probe-list	1, 2, 3, 4	pcia の差し込み式デバイスのプローブ順序を制御します。
pcib-probe-list	1, 2, 3	pcib の差し込み式デバイスのプローブ順序を制御します。

付録 E

Sun Ultra 30 UPA/PCI システム

この付録では、この PCI バスベースのシステムとサンの SBus ベースのシステムで異なる点について説明します。

PCI ベースのシステム

banner コマンドの出力は、次のように表示され、PCI ベースのシステムであることが示されます (バナーに UPA/PCI と表示されます)。

```
ok banner
Sun Ultra 30 UPA/PCI (UltraSPARC 200MHz), Keyboard Present
OpenBoot 3.9, 64 MB memory installed, Serial #8431666
Ethernet address 8:0:20:80:a8:32, Host ID: 8080a832
```

show-devs コマンドの出力には、PCI ベースのノードが表示されます。また、PCI ベースのシステムでは、デバイスの汎用的な名前が使用されます。オンボードのネットワークは "network" という名前で、内蔵ディスクは "diskn" (n はそのディスクの SCSI ターゲット番号) という名前です。

独自の FCodePROM を搭載した追加 PCI カードは、汎用的な名前を使用している場合とそうでない場合があります。汎用的な名前についての詳細は、Open Firmware Working Group のホームページ (http://playground.sun.com/1275) の「Recommended Practices」を参照してください。

```
ok show-devs
/SUNW,ffb@1e,0
/SUNW,UltraSPARC@0,0
/counter-timer@1f,1c00
/pci@1f,2000
/pci@1f,4000
/virtual-memory
/memory@0,60000000
/aliases
/options
/openprom
/chosen
/packages
/pci@1f,4000/usb@5
/pci@1f,4000/SUNW,m64B@2
/pci@1f,4000/scsi@3
/pci@1f,4000/network@1,1
/pci@1f,4000/ebus@1
/pci@1f,4000/scsi@3/tape
/pci@1f,4000/scsi@3/disk
/pci@1f,4000/ebus@1/SUNW,CS4231@14,200000
/pci@1f,4000/ebus@1/flashprom@10,0
/pci@1f,4000/ebus@1/eeprom@14,0
/pci@1f,4000/ebus@1/fdthree@14,3023f0
/pci@1f,4000/ebus@1/ecpp@14,3043bc
/pci@1f,4000/ebus@1/su@14,3062f8
/pci@1f,4000/ebus@1/su@14,3083f8
/pci@1f,4000/ebus@1/se@14,400000
/pci@1f,4000/ebus@1/sc@14,500000
/pci@1f,4000/ebus@1/SUNW,pll@14,504000
/pci@1f,4000/ebus@1/power@14,724000
/pci@1f,4000/ebus@1/auxio@14,726000
/openprom/client-services
/packages/sun-keyboard
/packages/SUNW,builtin-drivers
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
```

Sun Ultra 30UPA/PCI システムでの devalias コマンドの出力は次のとおりです。

```
ok devalias

screen                   /SUNW,ffb@1e,0
net                      /pci@1f,4000/network@1,1
disk                     /pci@1f,4000/scsi@3/disk@0,0
cdrom                    /pci@1f,4000/scsi@3/disk@6,0:f
tape                     /pci@1f,4000/scsi@3/tape@4,0
tape1                    /pci@1f,4000/scsi@3/tape@5,0
tape0                    /pci@1f,4000/scsi@3/tape@4,0
disk6                    /pci@1f,4000/scsi@3/disk@6,0
disk5                    /pci@1f,4000/scsi@3/disk@5,0
disk4                    /pci@1f,4000/scsi@3/disk@4,0
disk3                    /pci@1f,4000/scsi@3/disk@3,0
disk2                    /pci@1f,4000/scsi@3/disk@2,0
disk1                    /pci@1f,4000/scsi@3/disk@1,0
disk0                    /pci@1f,4000/scsi@3/disk@0,0
scsi                     /pci@1f,4000/scsi@3
floppy                   /pci@1f,4000/ebus@1/fdthree
ttyb                     /pci@1f,4000/ebus@1/se:b
ttya                     /pci@1f,4000/ebus@1/se:a
keyboard!              /pci@1f,4000/ebus@1/su@14,3083f8:forcemode
keyboard                 /pci@1f,4000/ebus@1/su@14,3083f8
mouse                    /pci@1f,4000/ebus@1/su@14,3062f8
```

PCI デバイスのデバイスノードの属性を見ると、PCI デバイスに固有の属性はほとんどなく、属性のフォーマットも SBus デバイスのフォーマットとほとんど同じです。たとえば、PCI デバイスの .properties の出力は次のようになります。

```
ok cd /pci@1f,4000/scsi@3
ok .properties
interrupts                00000020
assigned-addresses        81001810 00000000 00000400 00000000
00000100
                      82001814 00000000 00010000 00000000 00000100
                      82001818 00000000 00011000 00000000 00001000
device_type               scsi-2
clock-frequency           02625a00
reg                   00001800 00000000 00000000 00000000 00000000
                      01001810 00000000 00000000 00000000 00000100
                      02001814 00000000 00000000 00000000 00000100
                      02001818 00000000 00000000 00000000 00001000
model                     Symbios,53C875
compatible                gl
name                      scsi
devsel-speed              00000001
class-code                00010000
max-latency               00000040
min-grant                 00000011
revision-id               00000003
device-id                 0000000f
vendor-id                 00001000
```

汎用的な名前

次の例は、/pci@1f,4000/scsi@3 の下の汎用的な名前を示しています。

```
ok ls
f00809d8 tape
f007ecdc disk
```

.speed コマンドは、システムに接続されたプロセッサおよびバスの速度を表示します。

```
ok .speed
CPU  Speed : 200.00MHz
UPA  Speed : 100.00MHz
PCI  Bus A : 66Mhz
PCI  Bus B : 33Mhz
```

PCI バスの pcia と pcib

Sun Ultra 30 UPA/PCI システムには pcia および pcib の 2 つの PCI バスがあります。これらのバスのスロットをプローブする方法は、次の 2 つの NVRAM 設定変数により指定します。

表 E-1　PCI スロット

変数名	デフォルト値	説明
pcia-probe-list	1、2	pcia の差し込み式デバイスのプローブ順序を制御します。
pcib-probe-list	3、2、4、5	pcib の差し込み式デバイスのプローブ順序を制御します。

pcia-probe-list は /pci@1f,2000 の下のデバイスに対応し、
pcib-probe-list は /pci@1f,4000 の下のデバイスに対応します。

pcia は 1 つの追加クライアント ("PCI 1, 66" とマークされたスロット 1) をサポートします。pcia は 64 ビット幅で最大 66Mhz で動作するデバイスをサポートします。pcia には値 2 に対応するクライアント/スロットはありませんが、歴史的な理由から pcia-probe-list のデフォルト値に 2 が含まれています。

pcib は 3 つの差し込み式クライアント (それぞれ "PCI 2"、"PCI 3"、"PCI 4" と指定されたスロット 2、4、5) をサポートします。pcib は 64 ビット幅で最大 33Mhz で動作するデバイスをサポートします。

付録 F

Sun Ultra 60 UPA/PCI システム

この付録では、この PCI バスをベースにしたシステムとサンの SBus をベースにしたシステムで異なる点について説明します。

PCI ベースのシステム

banner コマンドの出力は、次のように表示され、PCI ベースのシステムであることが示されます (バナーに UPA/PCI と表示されます)。

```
ok banner
Sun Ultra 60 UPA/PCI (UltraSPARC-II 296MHz), No Keyboard
OpenBoot 3.11, 256 MB memory installed, Serial #9241373.
Ethernet address 8:0:20:8d:3:1d, Host ID: 808d031d.
```

show-devs コマンドの出力には、PCI ベースのノードが表示されます。また、PCI ベースのシステムでは、デバイスの汎用的な名前が使用されます。オンボードのネットワークは "network" という名前で、内蔵ディスクは "diskn" (n はそのディスクの SCSI ターゲット番号) という名前です。

独自の FCodePROM を搭載した追加 PCI カードは、汎用的な名前を使用している場合とそうでない場合があります。汎用的な名前についての詳細は、Open Firmware Working Group のホームページ (http://playground.sun.com/1275) の「Recommended Practices」を参照してください。

```
ok show-devs
/SUNW,UltraSPARC-II@0,0
/counter-timer@1f,1c00
/pci@1f,2000
/pci@1f,4000
/virtual-memory
/memory@0,a0000000
/aliases
/options
/openprom
/chosen
/packages
/pci@1f,4000/scsi@3,1
/pci@1f,4000/scsi@3
/pci@1f,4000/network@1,1
/pci@1f,4000/ebus@1
/pci@1f,4000/scsi@3,1/tape
/pci@1f,4000/scsi@3,1/disk
/pci@1f,4000/scsi@3/tape
/pci@1f,4000/scsi@3/disk
/pci@1f,4000/ebus@1/SUNW,CS4231@14,200000
/pci@1f,4000/ebus@1/flashprom@10,0
/pci@1f,4000/ebus@1/eeprom@14,0
/pci@1f,4000/ebus@1/fdthree@14,3023f0
/pci@1f,4000/ebus@1/ecpp@14,3043bc
/pci@1f,4000/ebus@1/su@14,3062f8
/pci@1f,4000/ebus@1/su@14,3083f8
/pci@1f,4000/ebus@1/se@14,400000
/pci@1f,4000/ebus@1/sc@14,500000
/pci@1f,4000/ebus@1/SUNW,pll@14,504000
/pci@1f,4000/ebus@1/power@14,724000
/pci@1f,4000/ebus@1/auxio@14,726000
/openprom/client-services
/packages/sun-keyboard
/packages/SUNW,builtin-drivers
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
```

Sun Ultra 60 UPA/PCI システムでの devalias コマンドの出力は次のとおりです。

```
ok devalias
screen                   /SUNW,ffb@1e,0
net                      /pci@1f,4000/network@1,1
disk                     /pci@1f,4000/scsi@3/disk@0,0
cdrom                    /pci@1f,4000/scsi@3/disk@6,0:f
tape                     /pci@1f,4000/scsi@3/tape@4,0
tape1                    /pci@1f,4000/scsi@3/tape@5,0
tape0                    /pci@1f,4000/scsi@3/tape@4,0
disk6                    /pci@1f,4000/scsi@3/disk@6,0
disk5                    /pci@1f,4000/scsi@3/disk@5,0
disk4                    /pci@1f,4000/scsi@3/disk@4,0
disk3                    /pci@1f,4000/scsi@3/disk@3,0
disk2                    /pci@1f,4000/scsi@3/disk@2,0
disk1                    /pci@1f,4000/scsi@3/disk@1,0
disk0                    /pci@1f,4000/scsi@3/disk@0,0
scsi                     /pci@1f,4000/scsi@3
floppy                   /pci@1f,4000/ebus@1/fdthree
ttyb                     /pci@1f,4000/ebus@1/se:b
ttya                     /pci@1f,4000/ebus@1/se:a
keyboard!              /pci@1f,4000/ebus@1/su@14,3083f8:forcemode
keyboard                 /pci@1f,4000/ebus@1/su@14,3083f8
mouse                    /pci@1f,4000/ebus@1/su@14,3062f8
name                     aliases
```

PCI デバイスのデバイスノードの属性を見ると、PCI デバイスに固有の属性はほとんどなく、属性の書式も SBus デバイスの書式とほとんど同じです。たとえば、PCI デバイスの .properties の出力は次のようになります。

```
ok cd /pci@1f,4000/scsi@3
ok .properties
assigned-addresses        81001810 00000000 00000400 00000000 00000100
                          82001814 00000000 00010000 00000000 00000100
                          82001818 00000000 00011000 00000000 00001000
device_type               scsi-2
clock-frequency           02625a00
reg                       00001800 00000000 00000000 00000000 00000000
                          01001810 00000000 00000000 00000000 00000100
                          02001814 00000000 00000000 00000000 00000100
                          02001818 00000000 00000000 00000000 00001000
model                     Symbios,53C875
compatible                glm
name                      scsi
devsel-speed              00000001
class-code                00010000
interrupts                00000001
max-latency               00000040
min-grant                 00000011
revision-id               00000014
device-id                 0000000f
vendor-id                 00001000
```

汎用的な名前

次の例は、/pci@1f,4000/scsi@3 の下の汎用的な名前を示しています。

```
ok ls
f007ae2c tape
f00797f4 disk
```

.speed コマンドは、システムに接続されたプロセッサおよびバスの速度を表示します。

```
ok .speed
CPU  Speed : 296.00MHz
UPA  Speed : 098.66MHz
PCI  Bus A : 66Mhz
PCI  Bus B : 33Mhz
```

PCI バスの pcia と pcib

Sun Ultra 60 UPA/PCI システムには pcia および pcib の 2 つの PCI バスがあります。これらのバスのスロットをプローブする方法は、次の 2 つの NVRAM 設定変数により指定します。

表 F-1　PCI スロット

変数名	デフォルト値	説明
pcia-probe-list	1、2	pcia の差し込み式デバイスのプローブ順序を制御します。
pcib-probe-list	3、2、4、5	pcib の差し込み式デバイスのプローブ順序を制御します。

pcia-probe-list は /pci@1f,2000 の下のデバイスに対応し、
pcib-probe-list は /pci@1f,4000 の下のデバイスに対応します。

pcia は 1 つの追加クライアント ("PCI 1, 66" とマークされたスロット 1) をサポートします。pcia は 64 ビット幅で最大 66 MHz で動作するデバイスをサポートします。pcia には値 2 に対応するクライアント/スロットはありませんが、歴史的な理由から pcia-probe-list のデフォルト値に 2 が含まれています。

pcib は 3 つの差し込み式クライアント (それぞれ "PCI 2"、"PCI 3"、"PCI 4" と指定されたスロット 2、4、5) をサポートします。pcib は 64 ビット幅で最大 33 MHz で動作するデバイスをサポートします。

付録 G

Sun Ultra 250 UPA/PCI システム

この付録では、この PCI バスをベースにしたシステムとサンの SBus をベースにしたシステムで異なる点について説明します。

banner コマンドの出力

banner コマンドの出力は次のように表示され、PCI ベースのシステムであることが示されます。

```
ok banner
Sun (TM) Enterprise 250 UPA/PCI (UltraSPARC-II 296MHz), No
Keyboard
OpenBoot 3.7, 128 MB memory installed, Serial #8941639.
Ethernet address 8:0:20:88:70:47, Host ID: 80887047.
```

汎用的な名前

show-devs コマンドの出力には、PCI ベースのノードが表示されます。PCI ベースのシステムでは、デバイスの汎用的な名前が使用されます。オンボードのネットワークは "network" という名前で、内蔵ディスクは "diskn" (n はそのディスクの SCSI ターゲット番号) という名前です (数字のない "disk" は "disk0" を表します)。独自の FCodePROM を搭載した追加 PCI カードは、汎用的な名前を使用している場合とそうでない場合があります。

汎用的な名前についての詳細は、Open Firmware Working Group のホームページ (http://playground.sun.com) の「Recommended Practices」を参照してください。

```
ok show-devs
/SUNW,UltraSPARC-II@0,0
/mc@0,0
/rsc
/pci@1f,2000
/pci@1f,4000
/counter-timer@1f,1c00
/associations
/virtual-memory
/memory@0,0
/aliases
/options
/openprom
/chosen
/packages
/mc@0,0/bank@0,60000000
/mc@0,0/bank@0,40000000
/mc@0,0/bank@0,20000000
/mc@0,0/bank@0,0
/mc@0,0/bank@0,0/dimm@0,3
/mc@0,0/bank@0,0/dimm@0,2
/mc@0,0/bank@0,0/dimm@0,1
/mc@0,0/bank@0,0/dimm@0,0/pci@1f,4000/scsi@3,1
/pci@1f,4000/scsi@3
/pci@1f,4000/network@1,1
/pci@1f,4000/ebus@1
/pci@1f,4000/scsi@3,1/tape
/pci@1f,4000/scsi@3,1/disk
/pci@1f,4000/scsi@3/tape
/pci@1f,4000/scsi@3/disk
/pci@1f,4000/ebus@1/SUNW,envctrltwo@14,600000
/pci@1f,4000/ebus@1/flashprom@10,0
/pci@1f,4000/ebus@1/eeprom@14,0
/pci@1f,4000/ebus@1/fdthree@14,3023f0
/pci@1f,4000/ebus@1/ecpp@14,3043bc
/pci@1f,4000/ebus@1/su@14,3062f8
/pci@1f,4000/ebus@1/su@14,3083f8
/pci@1f,4000/ebus@1/se@14,200000
/pci@1f,4000/ebus@1/se@14,400000
/pci@1f,4000/ebus@1/sc@14,500000
/pci@1f,4000/ebus@1/SUNW,pll@14,504000
/pci@1f,4000/ebus@1/power@14,724000
/pci@1f,4000/ebus@1/auxio@14,726000
```

```
/associations/slot2dev
/associations/slot2disk
/openprom/client-services
/packages/obdiag
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
```

内蔵 SCSI バス

Ultra 250 システムの場合、2 つの内蔵 SCSI バスがあります。デバイス "scsi" は内蔵ディスク用の内蔵 SCSI I/O バスを示します。

Sun Ultra 250 UPA/PCI システムでの devalias コマンドの出力は次のとおりです。

```
ok  devalias
disk5                    /pci@1f,4000/scsi@3/disk@c,0
disk4                    /pci@1f,4000/scsi@3/disk@b,0
disk3                    /pci@1f,4000/scsi@3/disk@a,0
disk2                    /pci@1f,4000/scsi@3/disk@9,0
disk1                    /pci@1f,4000/scsi@3/disk@8,0
disk0                    /pci@1f,4000/scsi@3/disk@0,0
disk                     /pci@1f,4000/scsi@3/disk@0,0
scsi                     /pci@1f,4000/scsi@3
cdrom                    /pci@1f,4000/scsi@3/disk@6,0:f
tape                     /pci@1f,4000/scsi@3/tape@4,0
pcia                     /pci@1f,2000
pcib                     /pci@1f,4000
pci0                     /pci@1f,4000
flash                    /pci@1f,4000/ebus@1/flashprom@10,0
nvram                    /pci@1f,4000/ebus@1/eeprom@14,0
parallel                 /pci@1f,4000/ebus@1/ecpp@14,3043bc
net                      /pci@1f,4000/network@1,1
ebus                     /pci@1f,4000/ebus@1
i2c                      /pci@1f,4000/ebus@1/SUNW,envctrltwo
floppy                   /pci@1f,4000/ebus@1/fdthree
tty                      /pci@1f,4000/ebus@1/se@14,400000
ttya                     /pci@1f,4000/ebus@1/se@14,400000:a
ttyb                     /pci@1f,4000/ebus@1/se@14,400000:b
rscctl                   /pci@1f,4000/ebus@1/se@14,200000:sspctl
rsc                      /pci@1f,4000/ebus@1/se@14,200000:ssp
ttyc                     /pci@1f,4000/ebus@1/se@14,200000:ssp
ttyd                     /pci@1f,4000/ebus@1/se@14,200000:sspctl
keyboard!             /pci@1f,4000/ebus@1/su@14,3083f8:forcemode
keyboard                 /pci@1f,4000/ebus@1/su@14,3083f8
mouse                    /pci@1f,4000/ebus@1/su@14,3062f8
name                     aliases
```

PCI デバイスの .properties

PCI デバイスのデバイスノードの属性を見ると、PCI デバイスに固有の属性はほとんどなく、属性のフォーマットも SBus デバイスのフォーマットとほとんど同じです。たとえば、PCI デバイスについての .properties の出力は次のようになります。

```
ok cd /pci@1f,4000/scsi@3
ok .properties
target6-scsi-options     00 00 05 f8
target5-scsi-options     00 00 05 f8
target4-scsi-options     00 00 05 f8
target3-scsi-options     00 00 05 f8
target2-scsi-options     00 00 05 f8
target1-scsi-options     00 00 05 f8
latency-timer            00000011
assigned-addresses       81001810 00000000 00000400 00000000 00000100
                         82001814 00000000 00010000 00000000 00000100
                         82001818 00000000 00011000 00000000 00001000
device_type              scsi-2
fru                      motherboard
clock-frequency          02625a00
reg                      00001800 00000000 00000000 00000000 00000000
                         01001810 00000000 00000000 00000000 00000100
                         02001814 00000000 00000000 00000000 00000100
                         02001818 00000000 00000000 00000000 00001000
model                    Symbios,53C875
compatible               70 63 69 31 30 30 30 2c 66 00 67 6c 6d 00 70 63
name                     scsi
devsel-speed             00000001
class-code               00010000
interrupts               00000020
max-latency              00000040
min-grant                00000011
revision-id              00000014
device-id                0000000f
vendor-id                00001000
```

次の例は、/pci@1f,4000/scsi@3 の下の汎用的な名前を示しています。

```
ok ls
f008bc60 tape
f007a51c disk
```

.speed コマンド

.speed コマンドは、システムに接続されたプロセッサおよびバスの速度を表示します。

```
ok .speed
CPU  Speed : 296.00MHz
UPA  Speed : 098.66MHz
PCI  Bus A at UPA node 1f: 66Mhz
PCI  Bus B at UPA node 1f: 33Mhz
```

PCI バススロットのプローブ

Sun Ultra 250 UPA/PCI システムでは、単一の PCI バスに配置された、4 つの PCI 追加スロットがあります。これらのバススロットをプローブする方法は、次の 2 つの NVRAM 設定変数により指定します。

表 G-1　NVRAM 設定変数

変数名	デフォルト値	説明
pci0-probe-list	3 、2、4、5	pcio の差し込み式デバイスのプローブ順序を制御します。
pci-slot-skip-list	なし	PCI 差し込み式スロットのスキップを制御します。

pci0-probe-list は、1F PCI コントローラの "B" バス上のデバイスのプローブ順序を指定します。デバイス 3 はマザーボード上の 876 UltraSCSI バス (内蔵ディスク)、デバイス 2、4、5 は差し込み式カード用の 33 MHz、32 ビットの空きスロットです。

pci-slot-skip-list は、プローブしない PCI スロットのリスト (0 から 3) です。"0" から "3" の値は背面パネルの PCI スロットに下から順に対応します。

Ultra 250 システムには、下から上に向かって 0 から 3 の番号が付いた 4 個の PCI 差し込み式スロットがあります (システムの背面からアクセス可能)。

表 G-2　PCI 差し込み式スロット

PCI スロット	PCI バス PCI デバイス	幅		速度
3	pci0	/pci@1f,2000/xxx@1	32 bit	33 MHz
2	pci0	/pci@1f,4000/xxx@2	32 bit	33 MHz
1	pci0	/pci@1f,4000/xxx@4	32 bit	33 MHz
0	pci0	/pci@1f,4000/xxx@5	32 bit	33 MHz

ここで、*xxx* はスロットに挿入された特定の PCI カードに対応します。たとえば、875/glm SCSI コントローラカードをスロット 0 に挿入すると `/pci@1f,4000/scsi@5` が生成され、876 デュアル SCSI カードをスロット 3 に挿入すると、2 つの異なる「デバイス」として `/pci@1f,2000/scsi@1` および `/pci@1f,2000/scsi@1,1` が作成されます。

SCSI プローブコマンド

以下の `probe-scsi` コマンドの出力例は、2 つの内蔵 SCSI バスを示しています。

```
ok probe-scsi
This command may hang the system if a Stop-A or halt command
has been executed.  Please type reset-all to reset the system
before executing this command.
Do you wish to continue? (y/n) y
Target 8
  Unit 0   Disk     SEAGATE ST32171W SUN2.1G8254
```

付録 H

Sun Ultra 450 UPA/PCI システム

この付録では、この PCI バスをベースにしたシステムとサンの SBus をベースにしたシステムで異なる点について説明します。

Banner コマンドの出力

banner コマンドの出力は次のように表示され、PCI ベースのシステムであることが示されます。

```
ok banner
Sun Ultra 450 (3 X UltraSPARC-II 248MHz), Keyboard Present
OpenBoot 3.5,  256 MB memory installed, Serial #8525185
Ethernet address 8:0:20:82:a5:81, Host ID: 80821581
```

汎用的な名前

show-devs コマンドの出力には、PCI ベースのノードが表示されます。PCI ベースのシステムでは、デバイスの汎用的な名前が使用されます。オンボードのネットワークは "network" という名前で、内蔵ディスクは "diskn" (n はそのディスクの SCSI ターゲット番号) という名前です (数字のない "disk" は "disk0" を表します)。独自の FCodePROM を搭載した追加 PCI カードは、汎用的な名前を使用している場合とそうでない場合があります。

汎用的な名前についての詳細は、Open Firmware Working Group のホームページ (http://playground.sun.com/1275) の「Recommended Practices」を参照してください。

```
ok show-devs
/pci@6,2000
/pci@6,4000
/pci@4,2000
/pci@4,4000
/SUNW,ffb@1d,0
/SUNW,UltraSPARC-II@1,0
/mc@0,0
/pci@1f,2000
/pci@1f,4000
/counter-timer@1f,1c00
/associations
/virtual-memory
/memory@0,0
/aliases
/options
/openprom
/chosen
/packages
/pci@6,4000/scsi@4,1
/pci@6,4000/scsi@4
/pci@6,4000/scsi@3,1
/pci@6,4000/scsi@3
/pci@6,4000/scsi@4,1/tape
/pci@6,4000/scsi@4,1/disk
/pci@6,4000/scsi@4/tape
/pci@6,4000/scsi@4/disk
/pci@6,4000/scsi@3,1/tape
/pci@6,4000/scsi@3,1/disk
/pci@6,4000/scsi@3/tape
/pci@6,4000/scsi@3/disk
/mc@0,0/bank@0,c0000000
/mc@0,0/bank@0,80000000
/mc@0,0/bank@0,40000000
/mc@0,0/bank@0,0
/mc@0,0/bank@0,40000000/dimm@0,3
/mc@0,0/bank@0,40000000/dimm@0,2
/mc@0,0/bank@0,40000000/dimm@0,1
/mc@0,0/bank@0,40000000/dimm@0,0
/mc@0,0/bank@0,0/dimm@0,3
/mc@0,0/bank@0,0/dimm@0,2
/mc@0,0/bank@0,0/dimm@0,1
/mc@0,0/bank@0,0/dimm@0,0
```

```
/pci@1f,4000/scsi@2
/pci@1f,4000/scsi@3
/pci@1f,4000/network@1,1
/pci@1f,4000/ebus@1
/pci@1f,4000/scsi@2/tape
/pci@1f,4000/scsi@2/disk
/pci@1f,4000/scsi@3/tape
/pci@1f,4000/scsi@3/disk
/pci@1f,4000/ebus@1/SUNW,CS4231@14,200000
/pci@1f,4000/ebus@1/SUNW,envctrl@14,600000
/pci@1f,4000/ebus@1/flashprom@10,0
/pci@1f,4000/ebus@1/eeprom@14,0
/pci@1f,4000/ebus@1/fdthree@14,3023f0
/pci@1f,4000/ebus@1/ecpp@14,3043bc
/pci@1f,4000/ebus@1/su@14,3062f8
/pci@1f,4000/ebus@1/su@14,3083f8
/pci@1f,4000/ebus@1/se@14,400000
/pci@1f,4000/ebus@1/sc@14,500000
/pci@1f,4000/ebus@1/SUNW,pll@14,504000
/pci@1f,4000/ebus@1/power@14,724000
/pci@1f,4000/ebus@1/auxio@14,726000
/associations/slot2dev
/associations/slot2led
/associations/slot2disk
/openprom/client-services
/packages/obdiag
/packages/disk-label
/packages/obp-tftp
/packages/deblocker
/packages/terminal-emulator
```

内蔵 SCSI バス

Ultra 450 システムの場合、2 つの内蔵 SCSI バスがあります。デバイス "`scsi`" は内蔵ディスク用の内蔵 SCSI I/O バスを示し、デバイス "`scsix`" は着脱式媒体および背面パネルの外部コネクタ用の内蔵 SCSI バスを示します。

Sun Ultra 450 UPA/PCI システムでの devalias コマンドの出力は次のとおりです。

```
ok  devalias
screen                   /SUNW,ffb@1d,0
disk                     /pci@1f,4000/scsi@3/disk@0,0
disk0                    /pci@1f,4000/scsi@3/disk@0,0
disk1                    /pci@1f,4000/scsi@3/disk@1,0
disk2                    /pci@1f,4000/scsi@3/disk@2,0
disk3                    /pci@1f,4000/scsi@3/disk@3,0
scsi                     /pci@1f,4000/scsi@3
diskx0                   /pci@1f,4000/scsi@2/disk@0,
diskx1                   /pci@1f,4000/scsi@2/disk@1,0
diskx2                   /pci@1f,4000/scsi@2/disk@2,0
diskx3                   /pci@1f,4000/scsi@2/disk@3,0
cdrom                    /pci@1f,4000/scsi@2/disk@6,0:f
tape                     /pci@1f,4000/scsi@2/tape@4,0
scsix                    /pci@1f,4000/scsi@2
pci                      /pci@1f,4000
pcia                     /pci@1f,2000
pcib                     /pci@1f,4000
pci0                     /pci@1f,4000
pci1                     /pci@1f,2000
pci2                     /pci@4,4000
pci3                     /pci@4,2000
pci4                     /pci@6,4000
pci5                     /pci@6,2000
flash                    /pci@1f,4000/ebus@1/flashprom@10,0
nvram                    /pci@1f,4000/ebus@1/eeprom@14,0
parallel                 /pci@1f,4000/ebus@1/ecpp@14,3043bc
net                      /pci@1f,4000/network@1,1
ebus                     /pci@1f,4000/ebus@1
i2c                      /pci@1f,4000/ebus@1/SUNW,envctrl
floppy                   /pci@1f,4000/ebus@1/fdthree
tty                      /pci@1f,4000/ebus@1/se
ttyb                     /pci@1f,4000/ebus@1/se:b
ttya                     /pci@1f,4000/ebus@1/se:a
keyboard!       /pci@1f,4000/ebus@1/su@14,3083f8:forcemode
keyboard                 /pci@1f,4000/ebus@1/su@14,3083f8
mouse                    /pci@1f,4000/ebus@1/su@14,3062f8
```

PCI デバイスの .properties

PCI デバイスのデバイスノードの属性を見ると、PCI デバイスに固有の属性はほとんどなく、属性のフォーマットも SBus デバイスのフォーマットとほとんど同じです。たとえば、PCI デバイスについての .properties の出力は次のようになります。

```
ok cd /pci@1f,4000/scsi@3
ok .properties
interrupts            00000020
assigned-addresses  81001810 00000000 00000400 00000000 00000100
                    82001814 00000000 00010000 00000000 00000100
                    82001818 00000000 00011000 00000000 00001000
device_type           scsi-2
clock-frequency       02625a00
reg                 00001800 00000000 00000000 00000000 00000000
                    01001810 00000000 00000000 00000000 00000100
                    02001814 00000000 00000000 00000000 00000100
                    02001818 00000000 00000000 00000000 00001000
model                 Symbios,53C875
compatible            glm
name                  scsi
devsel-speed          00000001
class-code            00010000
max-latency           00000040
min-grant             00000011
revision-id           00000003
device-id             0000000f
vendor-id             00001000
```

次の例は、/pci@1f,4000/scsi@3 の下の汎用的な名前を示しています。

```
ok ls
f00809d8 tape
f007ecdc disk
```

.speed コマンド

.speed コマンドは、システムに接続されたプロセッサおよびバスの速度を表示します。

```
ok .speed
CPU  Speed : 248.00MHz
UPA  Speed : 082.66MHz
PCI  Bus A at UPA node 1f: 66Mhz
PCI  Bus B at UPA node 1f: 33Mhz
PCI  Bus A at UPA node  6: 66Mhz
PCI  Bus B at UPA node  6: 33Mhz
PCI  Bus A at UPA node  4: 66Mhz
PCI  Bus B at UPA node  4: 33Mhz
```

PCI バススロットのプローブ

Sun Ultra 450 UPA/PCI システムでは、6 つの PCI バス (pci0 から pci5) に配置された 10 の PCI 差し込み式スロットがあります。これらのバススロットをプローブする方法は、次の 2 つの NVRAM 設定変数により指定します。

表 H-1　NVRAM 設定変数

変数名	デフォルト値	説明
pcio-probe-list	3 、2、4	*pcio* の差し込み式デバイスのプローブ順序を制御します。
pci-slot-skip-list	なし	PCI 差し込み式スロットのスキップを制御します。

pci0-probe-list は、1F PCI コントローラの "B" バス上のデバイスのプローブ順序を指定します。デバイス 3 はマザーボード上の 875 UltraSCSI バス (内蔵ディスク)、デバイス 2 は着脱式媒体および背面パネルの外部コネクタ用のマザーボード上の 875 (マザーボード上の 875 チップ 2 個)、デバイス 4 は差し込み式カード用の 33 MHz、32 ビットの空きスロットです。

残りの 5 つの PCI バス (pci1 から pci5) は、昇順にデバイススロットをプローブし、その順序は変更できません。

`pci-slot-skip-list` は、プローブしない PCI スロットのリスト (1 から 10) です。"1" から "10" の値は背面パネルの PCI スロットに下から順に対応します。

Ultra 450 システムには、下から上に向かって 1 から 10 の番号が付いた 10 個の PCI 差し込み式スロットがあります (システムの背面からアクセス可能)。10 個の PCI スロットは、次のように 6 つの PCI バスに対応します。

表 H-2

PCI スロット	PCI バス	PCI デバイス	幅	速度
10	`pci0`	`/pci@1f,4000/`*xxx*`@4`	32 Bit	33 MHz
9	`pci2`	`/pci@4,4000/`*xxx*`@2`	32 Bit	33 MHz
8	`pci2`	`/pci@4,4000/`*xxx*`@3`	32 Bit	33 MHz
7	`pci2`	`/pci@4,4000/`*xxx*`@4`	64 Bit	33 MHz
6	`pci3`	`/pci@4,2000/`*xxx*`@1`	64 Bit	66 MHz
5	`pci1`	`/pci@1f,2000/`*xxx*`@1`	64Bit	66 MHz
4	`pci5`	`/pci@6,2000/`*xxx*`@1`	64 Bit	66 MHz
3	`pci4`	`/pci@6,4000/`*xxx*`@2`	64 Bit	33 MHz
2	`pci4`	`/pci@6,4000/`*xxx*`@3`	64 Bit	33 MHz
1	`pci4`	`/pci@6,4000/`*xxx*`@4`	64 Bit	33 MHz

ここで、*xxx* はスロットに挿入された特定の PCI カードに対応します。たとえば、875/glm SCSI コントローラカードをスロット 8 に挿入すると `yield/pci@4,4000/scsi@3` が生成され、876 デュアル SCSI カードをスロット 5 に挿入すると、2 つの異なる「デバイス」として `/pci@1f,2000/scsi@1` および `/pci@1f,2000/scsi@1,1` が作成されます。PCI-PCI カード (PCI バス拡張ボックスや、Sun Swift PCI カードなどのマルチファンクション PCI カードで使用される) をスロット 4 に挿入すると、デバイス名 `/pci@6,2000/pci@1` が作成され、これに接続されるデバイスは、`/pci@6,2000/pci@1/SUNW,hme@0,1` のようにこのノードの「下」に作成されます。

これらのスロットの一部は、特定のグラフィックオプションが挿入されていると使用できません。たとえば、2 つ目の FFB グラフィックカードをインストールすると、PCI スロット 10、9、8 の物理的なスペースが使用できなくなります。その他のグラフィックオプションも PCI スロット 10 から 4 のスペースを使用する場合があります。

SCSI プローブコマンド

以下の `probe-scsi` コマンドの出力例は、2 つの内蔵 SCSI バスを示しています。

```
ok probe-scsi
Primary UltraSCSI bus:
Target 0
  Unit 0   Disk     SEAGATE ST34371W SUN4.2G8254
Target 1
  Unit 0   Disk     SEAGATE ST34371W SUN4.2G8254
Target 2
  Unit 0   Disk     SEAGATE ST34371W SUN4.2G8254
Target 3
  Unit 0   Disk     SEAGATE ST34371W SUN4.2G8254

Removeable-Media/External SCSI bus:
Target 3
  Unit 0   Removable Tape     ARCHIVE VIPER 150  21531-004    SUN-04.00.0
Target 4
  Unit 0   Removable Tape     EXABYTE EXB-8500SMBANXH10458
Target 5
  Unit 0   Removable Tape     EXABYTE EXB-8200        263H
Target 6
  Unit 0   Removable Read Only device    TOSHIBA XM-5401TASUN4XCD3485
```

付録 I

Forth ワードリファレンス

この付録には、OpenBoot 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、本書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。

スタック項目の表記

表 I-1　スタック項目の表記

表記	説明
|	代替スタック結果。空白文字を入れて表示されます。たとえば、(`input -- addr len false | result true`)。
|	代替スタック項目。空白文字を入れないで表示されます。たとえば、(`input -- addr len|0 result`)。
`???`	未知のスタック項目。
...	未知のスタック項目。スタックコメントの両側でされる場合、同じスタック項目が両側に表示されます。
`< > <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)。最上位ビットは不定。
`cnt len size`	カウント値または長さ。
`dxxx`	倍 (拡張) 精度数。スタックの一番上の最上位セルを占める 2 スタック項目。
`<eol>`	行末区切り子。
`false`	0 (false フラグ)。
`ihandle`	パッケージのインスタンス用ポインタ。
`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)
`<nothing>`	ゼロスタック項目。
`phandle`	パッケージへのポインタ。
`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 qxxx`	Quadlet (32 ビット)。

表 I-1　スタック項目の表記 (続き)

表記	説明
qaddr	Quadlet (32 ビット) 境界のアドレス。
{text}	省略可能なテキスト。省略した場合は、デフォルト動作が行われます。
"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。即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。
true	-1 (true フラグ)。
uxxx	符号なしの正の値 (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 wxxx	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	データ項目の下位/上位ビット。
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。実装によって異なります。
(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。

デバイスツリー表示コマンド

表 I-2　デバイスツリー表示コマンド

コマンド	説明
`.properties`	現在のノードの特性の名前と値を表示します。
`dev` *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。
`dev` *node-name*	指定されたノード名を現在のノードの下のサブツリーで捜し、最初に見つかったノードを選択します。
`dev ..`	現在のノードの親にあたるデバイスノードを選択します。
`dev /`	ルートマシンノードを選択します。
`device-end`	デバイスツリーをそのままにします。
" device-path " `find-device`	指定されたデバイスノードを選択します。`dev` と同じ。
`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。
`see` *wordname*	指定されたワードを逆コンパイルします。
`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。
" device-path " `select-dev`	指定されたデバイスを選択し、有効なノードにします。

boot コマンドの一般的オプション

表 I-3　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 (指定している場合) 選択するデバイスとパーティションのルートからのパス名とします。*filename* を指定しないと、起動プログラムは boot-file NVRAM 変数の値 (参照) を使用します。
[*options*]	(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。)

システム情報表示コマンド

表 I-4　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	取り付けられ、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。
.speed	CPU およびバス速度を表示します。
show-devs	取り付けられ、プローブされるすべてのデバイスを表示します。

システム変数表示/変更用コマンド

表 I-5　システム変数表示/変更用コマンド

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*	変数を指定された 10 進値またはテキスト値に設定します。(変更は永久的ですが、通常はリセット後に初めて有効になります。)
`set-default` *parameter*	指定された変数の設定値を工場出荷時のデフォルトに設定します。
`set-defaults`	変数の設定値を工場出荷時のデフォルトに戻します。
`password`	security-password を設定します。

NVRAMRC エディタコマンド

表 I-6　NVRAMRC エディタコマンド

コマンド	説明
`nvalias alias` *device-pat*	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 I-6　NVRAMRC エディタコマンド (続き)

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。

nvedit キー操作コマンド

表 I-7　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。
Escape B	1 語戻ります。
Control-F	1 文字位置進みます。
Escape F	1 語進みます。
Control-A	行の先頭まで戻ります。
Control-E	行の終わりまで進みます。
Control-N	編集バッファーの次の行に進みます。
Control-P	編集バッファーの前の行に戻ります。
Return (Enter)	カーソル位置に改行を挿入し、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-K	カーソル位置から行の終わりまで消去し、消去した文字を保存バッファーに格納します。行の終わりでは、現在行に次の行をつなぎます (つまり、改行を削除します)。
Delete	前の 1 文字を削除します。
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。

表 I-7　nvedit キー操作コマンド (続き)

キー操作	説明
Escape H	語の先頭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	語の先頭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	後の 1 文字を消去します。
Escape D	カーソルから語の終わりまで消去し、消去した文字を保存バッファーに格納します。
Control-U	1 行全体を消去し、消去した文字を保存バッファーに格納します。
Control-Y	保存バッファーの内容をカーソルの前に挿入します。
Control-Q	次の 1 文字の前に引用符を付けます (つまり制御文字を挿入できます)。
Control-R	行を入力し直します。
Control-L	編集バッファー内のすべての行を表示します。
Control-C	スクリプトエディタを終了し、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、スクリプトには戻されません。(後で `nvstore` を使用して一時バッファーをスクリプトに書いて戻してください。)

スタック操作コマンド

表 I-8　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
`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。
`2drop`	(x1 x2 --)	スタックから 2 項目を削除します。
`3drop`	(x1 x2 x3 --)	スタックから 3 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。
`2dup`	(x1 x2 -- x1 x2 x1 x2)	2 スタック項目を複製します。
`3dup`	(x1 x2 x3 -- x1 x2 x3 x1 x2 x3)	3 スタック項目を複製します。
`?dup`	(x -- x x | 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 I-8　スタック操作コマンド (続き)

コマンド	スタックダイアグラム	説明
nip	(x1 x2 -- x2)	2 番目のスタック項目を捨てます。
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。
2over	(x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)	2 番目以降のスタック項目をコピーします。
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)	スタック項目を復帰スタックに転送します。
r>	(-- x) (R: x --)	復帰スタック項目をスタックに転送します。
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。
roll	(xu … x1 x0 u -- xu-1 … x1 x0 xu)	u 個のスタック項目を回転します。(2 roll = rot)。
rot	(x1 x2 x3 -- x2 x3 x1)	3 スタック項目を回転します。
-rot	(x1 x2 x3 -- x3 x1 x2)	3 スタック項目を逆方向に回転します。
2rot	(x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)	2 対のスタック項目を入れ替えます。
swap	(x1 x2 -- x2 x1)	3 対のスタック項目を回転します。
2swap	(x1 x2 x3 x4 -- x3 x4 x1 x2)	一番上の 2 スタック項目を入れ替えます。
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。

単精度演算機能

表 I-9　単精度演算機能

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。
*/	(nu1 nu2 nu3 -- quot)	*n1* * *n2* / *n3* を計算します。
/	(n1 n2 -- quot)	*n1* / *n2* の除算を行います。剰余は捨てられます。
1+	(nu1 -- nu2)	1 を足します。
1-	(nu1 -- nu2)	1 を引きます。
2+	(nu1 -- nu2)	2 を足します。
2-	(nu1 -- nu2)	2 を引きます。
abs	(n -- u)	絶対値
bounds	(n count -- n+count n)	do または ?do ループの引数を準備します。
even	(n -- n | n+1)	>= *n* であって *n* に最も近い偶数の整数に丸めます。
max	(n1 n2 -- n1|n2)	*n1* と *n2* の大きい方の値を返します。
min	(n1 n2 -- n1|n2)	*n1* と *n2* の小さい方の値を返します。
mod	(n1 n2 -- rem)	*n1* / *n2* の剰余を計算します。
*/mod	(n1 n2 n3 -- rem quot)	*n1* * *n2* / *n3* の剰余と商。
/mod	(n1 n2 -- rem quot)	*n1* / *n2* の剰余と商。
negate	(n1 -- n2)	*n1* の符号を変更します。
u*	(u1 u2 -- uprod)	2 つの符号なし数値の乗算を行い、符号なしの積を生じます。
u/mod	(u1 u2 -- urem uquot)	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。

ビット操作論理演算子

表 I-10　ビット操作論理演算子

コマンド	スタックダイアグラム	説明
2*	(x1 -- x2)	2 を掛けます。
2/	(x1 -- x2)	2 で割ります。
>>a	(x1 u -- x2)	*x1* を *u* ビット算術右シフトします。
and	(x1 x2 -- x3)	ビット単位の論理積。
invert	(x1 -- x2)	*x1* のすべてのビットを反転します。
lshift	(x1 u -- x2)	*x1* を *u* ビット分左へシフトします。下位ビットはゼロで埋めます。
or	(x1 x2 -- x3)	ビット単位の論理和。
rshift	(x1 u -- x2)	*x1* を *u* ビット右シフトし、上位ビットはゼロで埋めます。
u2/	(x1 -- x2)	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。
xor	(x1 x2 -- x3)	ビット単位の排他的論理和。

倍精度数演算機能

表 I-11　倍精度数演算機能

コマンド	スタックダイアグラム	説明
d+	(d1 d2 -- d.sum)	d1 を d2 に足して、倍精度数 d.sum を生じます。
d-	(d1 d2 -- d.diff)	d2 から d1 を引いて、倍精度数 d.diff を生じます。
fm/mod	(d n -- rem quot)	*d* を *n* で割ります。
m*	(n1 n2 -- d)	符号付き乗算を行い、倍精度数の積を生じます。
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。
um*	(u1 u2 -- ud)	符号なし乗算を行って、符号なし倍精度数の積を生じます。
um/mod	(ud u -- urem uprod)	*ud* を *u* で割ります。

32 ビットデータ型変換機能

表 I-12　32 ビットデータ型変換機能

コマンド	スタックダイアグラム	説明
bljoin	(b.low b2 b3 b.hi -- quad)	2 バイトを結合して quadlet を作ります。
bwjoin	(b.low b.hi -- word)	2 バイトを結合して doublet を作ります。
lbflip	(quad1 -- quad2)	quadlet 内の 4 バイトを逆に並べ替えます。
lbsplit	(quad -- b.low b2 b3 b.hi)	quadlet を 2 つの 16 ビットワードに分割します。
lwflip	(quad1 -- quad2)	quadlet 内の 2 つの doublet をスワップします。
lwsplit	(quad -- w.low w.hi)	n3 を n1 と n2 の最大値とします。
wbflip	(word1 -- word2)	doublet 内の 2 バイトをスワップします。
wbsplit	(word -- b.low b.hi)	doublet を 2 バイトに分割します。
wljoin	(w.low w.hi -- quad)	2 つの doublet を結合して、quadlet を作ります。

64 ビットデータ型変換機能

表 I-13　64 ビットデータ型変換機能

コマンド	スタックダイアグラム	説明
bxjoin	(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)	8 バイトを結合して octlet を作ります。
lxjoin	(quad.lo quad.hi -- o)	2 つの quadlet を結合して octlet を作ります。
wxjoin	(w.lo w.2 w.3 w.hi -- o)	4 つの doublet を結合して cotlet を作ります。
xbflip	(oct1 -- oct2)	octlet 内の 4 つの doublet を逆に並べ替えます。
xbflips	(oaddr len --)	指定された領域の各 octlet 内の 8 バイトを逆に並べ替えます。len が /x の整数倍でない場合は、動作は不定です。
xbsplit	(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)	octlet を 8 バイトに分割します。
xlflip	(oct1 -- oct2)	octlet 内の 2 つの quadlet をスワップします。各 quadlet 内の 4 バイトは逆に並べ替えられません。
xlflips	(oaddr len --)	指定された領域の各 octlet 内の 2 つの quadlet をスワップします。各 quadlet 内の 4 バイトは逆に並べ替えられません。len が /x の整数倍でない場合は、動作は不定です。
xlsplit	(o -- quad.lo quad.hi)	octlet を 2 つの quadlet に分割します。
xwflip	(oct1 -- oct2)	octlet 内の 4 つの doublet を逆に並べ替えます。各 doublet 内の 2 バイトはスワップされません。
xwflips	(oaddr len --)	指定された領域の各 octlet 内の 4 つの doublet を逆に並べ替えます。各 doublet 内の 2 バイトはスワップされません。len が /x の整数倍でない場合は、動作は不定です。
xwsplit	(o -- w.lo w.2 w.3 w.hi)	octlet を 4 つの doublet に分割します。

変換演算子

表 I-14　変換演算子

コマンド	スタックダイアグラム	説明
aligned	(n1 -- n1 | a-addr)	必要な場合、*n1* を大きくして可変境界アドレスを生じます。
/c	(-- n)	1 バイトのバイト数 = 1。
/c*	(nu1 -- nu2)	chars の同義語。
ca+	(addr1 index -- addr2)	*addr1* を *index* の /c 倍増分します。
ca1+	(addr1 -- addr2)	char+ の同義語。
char+	(addr1 -- addr2)	*addr1* を /c の値だけ増やします。
cell+	(addr1 -- addr2)	*addr1* を /n の値だけ増やします。
chars	(nu1 -- nu2)	*nu1* に /c の値を掛けます。
cells	(nu1 -- nu2)	*nu1* に /n の値を掛けます。
/l	(-- n)	quadlet のアドレス単位数。通常 4。
/l*	(nu1 -- nu2)	nu1 に /l を掛けます。
la+	(addr1 index -- addr2)	*addr1* を index の /l 倍増分します。
la1+	(addr1 -- addr2)	*addr1* を/l 増分します。
/n	(-- n)	セルのアドレス単位数。
/n*	(nu1 -- nu2)	cells の同義語。
na+	(addr1 index -- addr2)	*addr1* を index の /n 倍増分します。
na1+	(addr1 -- addr2)	cell+ の同義語。
/w	(-- n)	16 ビットワードのバイト数 = 2。
/w*	(nu1 -- nu2)	*nu1* に /w を掛けます。
wa+	(addr1 index -- addr2)	*addr1* を index の /w 倍増分します。
wa1+	(addr1 -- addr2)	*addr1* を /w だけ増分します。

64 ビットアドレス演算機能

表 I-15　64 ビットアドレス演算機能

コマンド	スタックダイアグラム	説明
/x	(-- n)	octlet のアドレス単位数。通常 8。
/x*	(nu1 -- nu2)	nu1 に /x の値を掛けます。
xa+	(addr1 index -- addr2)	addr1 を /x の値の index 倍増やします。
xa1+	(addr1 -- addr2)	addr1 を /x の値だけ増やします。

メモリーアクセスコマンド

表 I-16　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
!	(x a-addr --)	数値を *a-addr* に格納します。
+!	(nu a-addr --)	*a-addr* に格納されている数値に *nu* を加算します。
@	(a-addr -- x)	数値を *a-addr* から 取り出します。
2!	(x1 x2 a-addr --)	2 数値を *a-addr* (*x2* を下位アドレス) に格納します。
2@	(a-addr -- x1 x2)	2 数値を *a-addr* (*x2* を下位アドレス) から取り出します。
blank	(addr len --)	*addr* で始まる *len* バイトのメモリーを空白文字 (10 進の 32) に設定します。
c!	(byte addr --)	*byte* を *addr* に格納します。
c@	(addr -- byte)	1 バイト を *addr* から取り出します。
cpeek	(addr -- false | byte true)	1 バイトを *addr* 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 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。

表 I-16　メモリーアクセスコマンド (続き)

コマンド	スタックダイアグラム	説明
comp	(addr1 addr2 len -- diff?)	2 つのバイト配列を比較します。両配列が一致する場合 *diff?* = 0、最初の異なるバイトが配列 *addr* 1 側より小さい場合 *diff?* = -1、それ以外の場合は *diff?* = 1 になります。
dump	(addr len --)	*addr* から始まる *len* バイトを 0 に設定します。
erase	(addr len --)	*addr* から始まる *len* バイトを 0 に設定します。
fill	(addr len byte --)	*addr* から始まる *len* バイトを *byte* に設定します。
l!	(quad qaddr --)	quadlet *q* を *qaddr* に格納します。
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 4 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 ビットの数を *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 --)	false を *a-addr* に格納します。
on	(a-addr --)	true を *a-addr* に格納します。
unaligned-l!	(quad addr --)	quadlet *q* を格納します。境界は任意です。
unaligned-l@	(addr -- quad)	quadlet *q* を取り出します。境界は任意です。
unaligned-w!	(w addr --)	doublet *w* を格納します。境界は任意です。
unaligned-w@	(addr -- w)	doublet *w* を取り出します。境界は任意です。
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。
<w@	(waddr -- n)	符号付き doublet *w* を *waddr* から取り出します。

表 I-16　メモリーアクセスコマンド (続き)

コマンド	スタックダイアグラム	説明
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。

64 ビットメモリーアクセス機能

表 I-17　64 ビットメモリーアクセス機能

コマンド	スタックダイアグラム	説明
<l@	(qaddr -- n)	quadr から quadlet を符号拡張して取り出します。
x,	(o --)	octlet, o をコンパイルして辞書に入れます (doublet 境界)。
x@	(oaddr -- o)	octlet 境界のアドレスから octlet を取り出します。
x!	(o oaddr --)	octlet 境界アドレスに octlet を格納します。
xbflips	(oaddr len --)	指定された領域の各 octlet 内の 8 バイトを逆に並べ替えます。len が /x の整数倍でない場合は、動作は不定です。
xlflips	(oaddr len --)	指定された領域の各 octlet 内の 2 つの quadlet を並べ変えます。各 quadlet 内の 4 バイトは逆に並べ替えられません。len が /x の整数倍でない場合は、動作は不定です。
xwflips	(oaddr len --)	指定された領域の各 octlet 内の 4 つの doublet を逆に並べ替えます。各 doublet 内の 2 バイトは並べ変えられません。len が /x の整数倍でない場合は、動作は不定です。

メモリーマップコマンド

表 I-18　メモリーマップコマンド

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 によりマップを解除します。
free-mem	(virt size --)	alloc-mem で割り当てられていたメモリーを開放します。
map?	(virt --)	仮想アドレスのメモリーマップ情報を表示します。

ワード定義

表 I-19　ワード定義

コマンド	スタックダイアグラム	説明
`:` *new-name*	(--) (E: … -- ???)	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*	(--) (E: … -- ???)	*old-name* と同じ動作をする *new-name* を作成します。
`buffer:` *name*	(size --) (E: -- 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 --) (E: -- n)	定数 (たとえば、`3 constant bar`) を定義します。
`2constant` *name*	(n1 n2 --) (E: -- n1 n2)	2 数値の定数を定義します。
`create` *name*	(--) (E: -- a-addr)	汎用定義ワード。
`defer` *name*	(--) (E: … -- ???)	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>`	(… -- … a-addr) (E: … -- ???)	ワード定義の実行節を開始します。
`field` *name*	(offset size -- offset+size) (E: 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 --) (E: -- n)	指定された、変更可能な数を作成します。
`variable` *name*	(--) (E: -- a-addr)	変数を定義します。

辞書検索コマンド

表 I-20　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)	ワードを辞書から検索します。実行トークンを返します。定義外で使用してください。
['] *name*	(-- xt)	定義内、外のどちらでも使用できる点以外は、' と同じです。
.calls	(xt --)	実行トークンが *xt* であるワードを呼び出すすべてのワードリストを表示します。
$find	(str len -- str len false | xt true)	str、len で指定するワードを検索します。見つかった場合、*xt* と true をスタックに残します。見つからなかった場合、文字列と false をスタックに残します。
find	(pstr -- pstr false | xt n)	pstr で指定するワードを検索します。見つかった場合、*xt* と true をスタックに残します。見つからなかった場合、文字列と false をスタックに残します。(パックされた文字列を使用しないようにするため、$find の使用を推奨します。)
see *thisword*	(--)	指定されたコマンドを逆コンパイルします。
(see)	(xt --)	実行トークンによって示されるワードを逆コンパイルします。
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。

辞書コンパイルコマンド

表 I-21　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,	(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、コンパイルを開始します。
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。(postpone の使用を推奨)
[compile] *name*	(--)	次の (即値) ワードをコンパイルします。(postpone の使用を推奨)
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	(-- 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。
patch *new-word old-word word-to-patch*	(--)	*old-word* を *word-to-patch* の *new-word* に置き換えます。
(patch)	(new-n old-n xt --)	*old-n* を *xt* によって示されるワードの *new-n* に置き換えます。

表 I-21　辞書コンパイルコマンド (続き)

コマンド	スタックダイアグラム	説明
`postpone` *name*	(--)	ワード *name* の実行を遅延させます。
`recursive`	(--)	辞書内のコンパイル中のコロン定義を表示可能にし、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)	コンパイル状態でゼロ以外の変数。

アセンブリ言語のプログラミング

表 I-22　アセンブリ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*	(-- code-sys) (E: ... -- ???)	*name* と呼ばれるアセンブリ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;`	(code-sys --)	アセンブリ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リ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。
`label` *name*	(-- code-sys) (E: -- a-addr)	*name* と呼ばれるアセンブリ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`	(code-sys --)	`label` で開始されたアセンブリ言語のパッチを終了します。

基数値表示

表 I-23　基数値表示

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。
noshowstack	(--)	各 ok プロンプトの前のスタックの自動表示をオフに設定します。
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。

基数の変更

表 I-24　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 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。

数値出力ワード用基本式

表 I-25　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-- addr +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.)	(-- addr 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	(addr len -- true | n false)	文字列を数値に変換します。
#s	(l -- 0)	残り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。

テキスト入力制御

表 I-26　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。
accept	(addr len1 -- len2)	コンソール入力デバイスから編集済み入力行を得て *addr* に格納します。len 1 は許容される最大の長さ、len 2 は実際に受け取られた長さです。
expect	(addr len --)	コンソールから入力行を得て表示し、*addr* に格納します。(accept の使用を推奨します。)
key	(-- char)	コンソール入力デバイスから 1 文字を読みます。
key?	(-- flag)	コンソール入力デバイスでキーが押された場合 true。
parse	(char -- str len)	入力バッファーからの char で区切られたテキストを構文解析します。
parse-word	(-- str len)	入力バッファーからの char で区切られたテキストを、先行空白文字を読み飛ばして構文解析します。
safe-parse-word	(-- str len)	入力が NULL 文字列のときエラーを示す場合を使用目的としている点を除いて、parse-word と同じです。
word	(char -- pstr)	入力バッファーから *char* で区切られている文字列をまとめ、メモリ位置 *pstr* にパックされた文字列として入れます。(parse の使用を推奨します。)

テキスト出力表示

表 I-27　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 (space) 文字を表示します。
spaces	(+n --)	*+n* 個の空白文字を表示します。
type	(addr +n --)	*n* 個の文字を表示します。

書式付き出力

表 I-28　書式付き出力

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

テキスト文字列の操作

表 I-29　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
`",`	(addr len --)	*addr* 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"`	(-- addr 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`	(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)	*char* の文字列を分割します (*char* は捨てられます)。
`pack`	(addr len pstr -- pstr)	*addr 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。
`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。

入出力先リダイレクトコマンド

表 I-30　入出力先リダイレクトコマンド

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（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（screen、または *device-specifier*) を選択します。

ASCII 定数

表 I-31　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。

コマンド行エディタ用キー操作コマンド

表 I-32　コマンド行エディタ用キー操作コマンド

キー操作	説明
Control-B	1 文字位置戻ります。
Escape B	1 語戻ります。
Control-F	1 文字位置進みます。
Escape F	1 語進みます。
Control-A	行の始めに戻ります。
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	1 つ前の文字を消去します。
Escape H	語の先頭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	語の先頭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	現在位置の文字を消去します。
Escape D	語の現在位置から終わりまで消去し、消去した文字を保存バッファーに格納します。
Control-K	カーソルから行の終わりまで消去し、消去した文字を保存バッファーに格納します。
Control-U	1 行全体を消去し、消去した文字を保存バッファーに格納します。
Control-R	行を再表示します。
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-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-P	履歴から、1 行前の行を選択、表示します。
Control-N	履歴から、1 行後の行を選択、表示します。
Control-L	履歴から、編集バッファーの全内容を表示します。

コマンド補完キー操作コマンド

表 I-33　コマンド補完キー操作コマンド

キー操作	説明
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。

比較コマンド

表 I-34　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* が *n2* に等しくない場合 true。
=	(n1 n2 -- flag)	*n1* = *n2* の場合 true。
>	(n1 n2 -- flag)	*n1* > *n2* の場合 true。
>=	(n1 n2 -- flag)	*n1* >= *n2* の場合 true。
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)。
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 の値 = 0。
true	(-- -1)	TRUE 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。

表 I-34　比較コマンド (続き)

コマンド	スタックダイアグラム	説明
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。

if-else-then コマンド

表 I-35　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、次のコードを実行します。
else	(--)	*flag* が false の場合、次のコードを実行します。
then	(--)	if…else…then を終了します。

case 文コマンド

表 I-36　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。

begin (条件付き) ループコマンド

表 I-37 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。

do (カウント付き) ループコマンド

表 I-38 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を加算し、do に戻ります (*n* < 0 の場合は、インデックスは *start* から *end* まで変わります。)
?do	(end start --)	?do…loop の 0 回またはそれ以上の実行を開始します。インデックスは *start* から *end*-1 まで変わります。*end* = *start* の場合はループは実行されません。
?leave	(flag --)	*flag* がゼロ以外の場合、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*-1 まで変わります。 例: 10 0 do i . loop (0 1 2…d e f と出力します。)
i	(-- n)	ループインデックス。

表 I-38　do (カウント付き) ループコマンド (続き)

コマンド	スタックダイアグラム	説明
j	(-- n)	1 つ外側のループのループインデックス。
leave	(--)	do…loop から抜けます。
loop	(--)	do…loop を終了します。

プログラム実行制御コマンド

表 I-39　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、メッセージを表示します。
eval	(… str len -- ???)	evaluate の同義語。
evaluate	(… str len -- ???)	指定する文字列から 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。

ファイル読み取りコマンド

表 I-40　ファイル読み取りコマンド

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`	(--)	`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*	(addr --)	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。
`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`	(--)	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`	(-- addr)	`load` がデバイスから読んだデータを読み込むアドレス。

逆コンパイラコマンド

表 I-41　逆コンパイラコマンド

コマンド	スタックダイアグラム	説明
+dis	(--)	最後に逆コンパイルを中断したところから逆コンパイルを継続します。
dis	(addr --)	指定されたアドレスから逆コンパイルを開始します。

ブレークポイントコマンド

表 I-42　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	(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。
.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。

表 I-42　ブレークポイントコマンド (続き)

コマンド	スタックダイアグラム	説明
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設定することにより、任意のアドレスに移ることができます。
gos	(n --)	go を *n* 回実行します。
hop	(--)	(step コマンドと同じです。) サブルーチン呼び出しを 1 つの命令と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	(addr --)	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。

Forth ソースレベルデバッガコマンド

表 I-43　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。
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。

表 I-43　Forth ソースレベルデバッガコマンド (続き)

コマンド	説明
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。
g	"Go"。デバッガをオフに設定し、実行を継続します。
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。
s	"see"。デバッグされているワードを逆コンパイルします。
$	スタックの一番上の address,len をテキスト文字列として表示します。
h	"Help"。シンボリックデバッガマニュアルを表示します。
?	"Short Help"。簡略シンボリックデバッガマニュアルを表示します。
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。("dup" などの基本 Forth ワードはデバッグしないでください。)
(debug	入力ストリームから名前ではなく、スタックから実行トークンを取り出す点を除いて、debug と同じです。
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追跡し、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。

時間ユーティリティー

表 I-44　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。

その他の処理

表 I-45　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 サン OS を呼び出します。
catch	(… xt -- ??? error-code | ??? false)	*xt* 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、0x00030001 = ファームウェアバージョン 3.1) 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。

表 I-45　その他の処理 (続き)

コマンド	スタックダイアグラム	説明
reset-all	(--)	システム全体をリセットします (電源再投入と同じ)。
sync	(--)	オペレーティングシステムを呼び出してすべての保留情報をハードディスクに書き出します。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。

マルチプロセッサコマンド

表 I-46　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
switch-cpu	(cpu# --)	指定された CPU に切り替えます。

メモリー割り当てコマンド

表 I-47　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
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。

メモリー割り当て用基本式

表 I-48　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
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。
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。
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。
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。
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。
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 *size* に割り当てます。
map-region	(region# virt --)	1 つの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。
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 *size* を返します。
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。

表 I-48　メモリー割り当て用基本式 (続き)

コマンド	スタックダイアグラム	説明
segmentsize	(-- size)	セグメントの *size* を返します。
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返します。
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。

キャッシュ操作コマンド

表 I-49　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。
ecdata!	(data offset --)	データ をキャッシュオフセット に格納します。
ecdata@	(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。
ectag!	(value offset --)	タグ値 をキャッシュオフセット に格納します。
ectag@	(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。

Sun-4u マシンの
マシンレジスタ読み取り/書き込み

表 I-50　Sun-4u 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
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出します。

代替アドレス空間アクセスコマンド

表 I-51　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 addr asi --)	*asi* の 1 バイト を *addr* に格納します。
spacec?	(addr asi --)	*addr* アドレスの *asi* の 1 バイト を表示します。
spacec@	(addr asi -- byte)	*addr* にある *asi* から 1 バイト を取り出します。
spaced!	(quad1 quad2 addr asi --)	*asi* の 2 つの quadlet をアドレス *addr* に格納します。数値の順序は実装によります。
spaced?	(addr asi --)	*addr* の *asi* にある 2 つの quadlet を表示します。数値の順序は実装によります。
spaced@	(addr asi -- quad1 quad2)	*asi* の *addr* から 2 つ 4 バイトワードを取り出します。
spacel!	(quad addr asi --)	*asi* の *addr* に 4 バイトワードを格納します。
spacel?	(addr asi --)	*asi* の *addr* にある 4 バイトワードを表示します。
spacel@	(addr asi -- quad)	*asi* の *addr* から 4 バイトワードを取り出します。
spacew!	(w addr asi --)	*asi* の *addr* に 2 バイトワードを格納します。
spacew?	(addr asi --)	*asi* の *addr* にある 2 バイトワードを表示します。
spacew@	(addr asi -- w)	*asi* の *addr* から 2 バイトワードを取り出します。

表 I-51　代替アドレス空間アクセスコマンド (続き)

コマンド	スタックダイアグラム	説明
spacex!	(x addr asi --)	*asi* の *addr* に数値を格納します。
spacex?	(addr asi --)	*asi* の *addr* にあるワードを表示します。
spacex@	(addr asi -- x)	*asi* の *addr* からワードを取り出します。

SPARC レジスタコマンド

表 I-52　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
%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 %y	(-- value)	指定されたレジスタの値を返します。
%f0 ～ %f31	(-- value)	指定された浮動小数点レジスタの値を返します。
.fregisters	(--)	%f0 から %f31 までの値を表示します。
.locals	(--)	i、l、o レジスタの値を表示します。
.registers	(--)	プロセッサレジスタの値を表示します。
.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 を *new-value* に、%npc を (*new-value* + 4) に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、%ix、%lx、または %ox 表示向けに設定します。

SPARC V9 レジスタコマンド

表 I-53　SPARC V9 レジスタコマンド

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

緊急キーボードコマンド

表 I-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	NVRAM の内容をデフォルト設定に戻します。

索引

記号

!, 60
<=, 81
<>, 81
<l@, 61
<w@, 61
", 75
', 67
",, 75
$create, 65
$find, 67
$nvalias, 39
$nvunalias, 40
$sift, 67
%npc, 105
%pc, 105
(, 54, 73
(cr, 74
(debug, 108
(patch), 70
(see), 67, 101
), 73
*/mod, 55
+, 47
+bp, 106, 107
+dis, 103
+loop, 87
--bp, 106
-bp, 106
-rot, 51
-trailing, 75
,, 70
., 48, 71
.", 74, 75
.(, 75
.bp, 106
.breakpoint, 106
.calls, 67
.d, 46, 72
.fregisters, 104
.h, 72
.idprom, 24
.instruction, 106
.locals, 104
.properties, 7, 8
.r, 71
.registers, 104
.s, 71
.step, 106
.traps, 24
.version, 24
.window, 104

/c, 57
/c*, 57
/etc/remote ファイル, 114
/l, 58
/l*, 58
/mod, 55
/n, 58
/n*, 58
/w, 58
/w*, 58
/x, 58
/x*, 58
:, 52, 53, 64
;, 52, 53, 64
=, 81
>, 81, 82
> =, 81
>body, 70
>r, 51
?do, 87
?dup, 51
?go, 93
?leave, 87
@, 59, 65
[compile], 70
['], 67

数字

0=, 81, 82
0>, 81
0>=, 81
0<=, 81
0<>, 81
2constant, 64
2drop, 51
2dup, 51
2over, 51
2rot, 51
2swap, 52
3drop, 51
3dup, 51

A

abort, 89
abort", 89
accept, 73
again, 85
alias, 64
aligned, 57
alloc-mem, 62
allot, 70
ascii, 73
auto-boot?, 15, 25, 28, 36

B

banner, 32, 38
base, 72
begin, 85
begin ループ, 85
between, 81
bl, 75
bljoin, 56
body>, 70
boot, 39, 92, 93, 94
boot-command, 13, 15, 25
boot-device, 14, 15, 20, 25
boot-file, 14, 15, 20, 25
bounds, 55
bpoff, 106
buffer:, 64
bwjoin, 56
bxjoin, 57
byte-load, 92

C

c,, 70
ca+, 58
ca1+, 58
call オペコード, 103
case, 84
cell+, 58
cells, 58
char+, 58
chars, 58
clear, 51
comp, 60
compile, 70
constant, 64
count, 75, 76
cpeek, 60
cpoke, 60
cr, 74
create, 64
ctrace, 104

D

d-, 56
d#, 72
d+, 56
debug, 108
debug-off, 108
decimal, 46, 72
defer, 65, 66
depth, 51
dev, 7
devalias, 7
device-end, 8
diag-device, 14, 19, 25, 36
diag-file, 14, 19, 25, 36
diagnostic-mode?, 19
diag-switch?, 14, 19, 25, 36
dis, 103
dl, 92
dlbin, 92
dload, 92
do, 87
does>, 65
do ループ, 86
drop, 51
dump, 44, 60, 62
dup, 51, 52

E

else, 82
emit, 74
endcase, 84
endof, 84
erase, 60
Ethernet
　アドレスの表示, 24
eval, 89, 92
evaluate, 92
execute, 89
exit, 89
exit?, 74
expect, 73

F

false, 81
fcode-debug?, 26
FCode インタプリタ, 2
FCode プログラム, 99
field, 65
fill, 60
find, 67
find-device, 8
finish-loop, 106

fm/mod, 56
Forth モニター, 2
Forth
　コマンドの書式, 43
　ソースレベルデバッガ, 108
　プログラム, 94, 98
　モニター, 2
Forth コードのコマンド, 74
free-mem, 62
ftrace, 112
full セキュリティーモード, 31

G

go, 39, 92, 105, 107
gos, 107

H

h#, 72
help, 11
here, 70
hex, 46, 72
hop, 107
hops, 107

I

i, 87
if, 82
immediate, 70
init-program, 92
input, 76
input-device, 26, 34
input-device, 77
install-console, 38
invert, 55
io, 76, 78

J

j, 87

K

key, 73
key?, 73, 74, 116

L

l,, 70
l@, 59
la+, 58
la1+, 58
lbflip, 56
lbflips, 60
lbsplit, 56
lcc, 75
leave, 87
left-parse-string, 75
literal, 70
load, 93
loop, 87
lpeek, 61
lpoke, 61
ls, 8
lwflips, 60
lwsplit, 56
lxjoin, 57

M

m*, 56
max, 55
min, 55
mod, 55
move, 61

N

na+, 58
na1+, 58
negate, 55
nip, 51
noshowstack, 47, 71
not, 55
nvalias, 39
nvedit
　キーボードコマンドの概要, 40
nvedit, 39, 42
nvquit, 39
NVRAM, 25
NVRAMRC
　nvramrc コマンド, 26
nvrecover, 39
nvrun, 39
nvstore, 40
nvunalias, 40

O

oem-banner, 26, 32
oem-banner?, 26, 32
oem-logo, 26, 32
oem-logo?, 26, 32
of, 84
off, 61
on, 61
origin, 70
output, 76
output-device, 26, 34
output-device, 77
over, 51

P

pack, 75
parse, 73
parse-word, 73
password, 39
patch, 70
pick, 51
postpone, 71
printenv, 28
probe-all, 38
probe-scsi, 12, 20
pwd, 8

Q

quit, 89

R

r>, 51
r@, 51
recurse, 71
recursive, 71
repeat, 85
reset-all, 24, 39
resume, 108
return, 107
returnl, 107
roll, 51
rot, 51
rshift, 55

S

s>d, 56
sbus-probe-list, 26
screen-#columns, 26, 34
screen-#rows, 26, 34
SCSI デバイス
　確認, 20

security-#badlogins, 26, 29
security-mode, 26, 29, 30
security-password, 26, 29
see, 8, 67, 101 ~ 102
set-default, 27, 29
set-defaults, 27, 29
setenv, 28
setenv security-mode, 39
set-pc, 104, 105
show-devs, 8
show-sbus, 24
showstack, 47, 71
sifting, 67
skip, 107
sm/rem, 56
space, 74
space-bar, 109
spaces, 74
SPARC レジスタ
  %f0 - %f31, 104
  %i0 - %i7, 104
  %npc, 104, 105
  %o0 - %o7, 104
  %pc, 104, 105
state, 71
step, 107
stepping, 109
steps, 107
Stop-A, 77
struct, 65
suppress-banner, 38

T

test, 20
then, 82
till, 107
TIP ウィンドウ, 113, 114
TIP に関する問題, 116
to, 70, 104
tracing, 109
true, 81
ttya, 77
ttyb, 77
type, 74

U

u., 71
u.r, 71
u/mod, 55
u>, 82
u>=, 82
u2/, 55
um*, 56
um/mod, 56
until, 85
upc, 75
use-nvramrc?, 26
u<=, 81

V

value, 64, 65
variable, 65

W

w, 104
w,, 70
w@, 59
wa+, 58
wa1+, 58
watch-clock, 22
watch-net, 23
wbflip, 56, 57

wbflips, 61
wbsplit, 57
while, 85
within, 82
wljoin, 57
word, 73
words, 8, 44, 67
wpeek, 61
wpoke, 61
wxjoin, 57

X

x!, 61
x,, 71
x@, 59, 61
xa+, 58
xa1+, 58
xbflip, 57
xbflips, 62
xlflip, 57
xlflips, 62
xlsplit, 57
xor, 55
xwflip, 57
xwflips, 62
xwsplit, 57

え

演算機能
  アドレス演算, 57
  アドレス演算、64 ビット, 58
  単精度, 54
  データ型の変換、64 ビット, 56
  倍精度, 56

か

拡張診断、実行, 36
仮想アドレス, 59
括弧, 73

き

キーボードコード, 29
基数の変更, 72
逆アセンブラコマンド, 103
緊急キーボードコード, 29

け

現在の変数設定の表示, 28

こ

コマンド行エディタ, 80
  オプションのコマンド補完用コマンド, 80
  オプションの履歴用コマンド, 80
  必須コマンド, 79
コマンド辞書, 64
コマンドセキュリティーモード, 30
コロン定義, 52

さ

作成
  新しいコマンド, 52
  新しいロゴ, 33
  カスタムバナー, 33
  辞書項目, 64
差し込み式デバイスのドライバ, 1

し

辞書の検索, 67

辞書へのデータのコンパイル, 70
  64 ビット, 71
システム変数
  Sbus
    sbus-probe-list, 26
  設定, 27, 28
  表示, 27
  標準
    boot-device, 20
    boot-file, 20
    diag-device, 19
    auto-boot?, 15, 25, 36
    boot-command, 13, 15, 25
    boot-device, 14, 15, 25
    boot-file, 14, 15, 25
    diag-device, 14, 25, 36
    diag-file, 14, 25, 36
    diag-switch?, 14, 19, 25, 36
    fcode-debug?, 26
    input-device, 26, 34
    nvramrc, 26
    oem-banner, 26, 32
    oem-banner?, 26, 32
    oem-logo, 26, 32
    oem-logo?, 26, 32
    output-device, 26, 34
    screen-#columns, 26, 34
    screen-#rows, 26, 34
    security-#badlogins, 26, 29
    security-mode, 26, 29
    security-password, 26, 29
    use-nvramrc?, 26
実行可能バイナリプログラム, 96, 99
シリアルポート, 76
診断
  ルーチン, 19
シンボルテーブル, 103

す

数値表示, 71
スクリプト, 37
  エディタコマンド, 39
  使用できないコマンド, 38
スタック
  説明, 46
  操作コマンド, 51
  ダイアグラム, 48
スタックコメント
  表記法, 49

せ

セキュリティー
  コマンド, 30
  フル, 31
設定
  シリアルポート特性, 35
  デフォルト入出力デバイス, 35
  ファームウェアセキュリティ, 29
設定を戻す
  システム変数をデフォルト設定に, 29

た

端末, 76

て

テキスト出力コマンド, 74
テキスト入力コマンド, 73
テキスト文字列操作, 75
テスト
  クロック, 22
  ネットワーク接続, 20, 22
  フロッピーディスクドライブ, 21
  メモリー, 21
デバイス
  ツリーの表示/走査, 7
  ノードの性質, 3
  パス名, 4
  別名, 6
デバイス指定子, 17

デバッガコマンド
$, 108
?, 108
c, 108
d, 108
(debug, 108
debug, 108
debug-off, 108
f, 108
g, 108
h, 108
q, 108
resume, 108
s, 108
space-bar, 109
stepping, 109
tracing, 109
u, 108
電源投入時
バナー, 24
電源の再投入, 43, 77

に

二次起動プログラム, 16
入出力先変更, 76

ぬ

ヌルモデムケーブル, 113

ひ

比較コマンド, 81
表記法
スタックコメント, 49

ふ

ファイルの読み込みと実行
Ethernet から, 98
load の使用, 95
シリアルポート A から FCode またはバイナリファイルを, 96
シリアルポートから Forth テキストファイルを, 94
boot の使用, 93, 94
ファイル読み込み用コマンド, 92
物理アドレス, 59
フラグ, 81
ブレークポイントコマンド, 106
go, 105
フレームバッファー, 76
プログラムカウンタ, 105
プログラム実行制御コマンド, 89
プロンプト, 53, 83

へ

変数, 33

め

メモリー
アクセス, 59, 60
アクセス、64 ビット, 61

も

文字列、操作, 75

ゆ

ユーザーインタフェース
コマンド行エディタ, 78, 80
オプションのコマンド補完用コマンド, 80
オプションの履歴用コマンド, 80
必須コマンド, 79

り

リセット
　システム, 24
履歴用機能, 78

る

ループ
　カウント付き, 86
　条件付き, 85

れ

レジスタの表示, 103
レジスタの読み込みと書き出し
　SPARC マシン, 104

わ

ワード定義, 64