

GNUPRO™ TOOLKIT

GNUPro TOOLKITリファレンスガイド
eCos 松下 MN10300

eCos1.1
1998年10月

CYGNUS

本書に関する全ての著作権は **CYGNUS SOLUTIONS, Inc.** が所有しています。
CYGNUS SOLUTIONS, Inc.の書面による事前の承諾無しに、いかなる形式、いかなる手段にかかわらず、本書のいかなる部分の複製を禁じます。
CYGNUS SOLUTIONS, Inc.の書面による事前の承諾無く、本書のいかなる部分の変更、修正も禁じます。
GNUPro™, GNUPro™ のロゴ, そして Cygnus Solutions のロゴは **CYGNUS SOLUTIONS, Inc.** の登録商標であり、その他のブランド名及び商品名は、その著作権所有者の商標または登録商標です。

シグナスソリューションズ 本社

CYGNUS SOLUTIONS

1325 Chesapeake Terrace

Sunnyvale, CA 94089 USA

TEL: +1 408 542 9600 (代表)

サポート:アメリカ及びカナダ国内: +1 800 CYGNUS1

その他の地域: +1 408 542 9601

FAX: +1 408 542 9699

E-mail: info@cygnus.com

URL: <http://www.cygnus.com/>

東海岸支社

CYGNUS SOLUTIONS

955 Massachusetts Avenue

Cambridge, MA 02139-3180

日本支社

日本シグナスソリューションズ

〒102-0094

東京都千代田区紀尾井町4-13 マードレ松田ビル

TEL: +81-3-3234-3896

FAX: +81-3-3239-3300

E-mail: info@cygnus.co.jp

URL: <http://www.cygnus.co.jp/>

英国支社

CYGNUS SOLUTIONS

35-36 Cambridge Place

Cambridge CB2 1NS

United Kingdom

Part #: 300-400-1010048-1.

目次

はじめに.....	1
ツールの命名規約.....	1
ツールキットの特徴.....	2
プロセッサのバージョン.....	2
サポートするターゲット.....	2
サポートするホスト.....	3
オブジェクトファイルフォーマット.....	3
WINDOWS NT上でのGNUPRO.....	4
Windows NTでの環境設定.....	4
大小文字の区別.....	4
REDHAT LINUX上での GNUPRO.....	6
参考資料.....	7
コンパイラ.....	7
MN10300 固有のコマンドラインオプション.....	7
プリプロセッサのシンボル.....	7
MN10300固有の属性.....	7
コンパイラとリンカの新機能.....	8
初期設定の優先順位付け.....	8
選択的リンク.....	9
EABIのまとめ.....	10
データタイプのサイズとアラインメント.....	10
レジスタの割り当て.....	10
レジスタの使用.....	11
スイッチ.....	11
スタックフレーム.....	12
引数の渡し方.....	14
関数のリターン値.....	14
アセンブラ.....	15
アセンブラのコマンドラインオプション.....	15
シンタックス.....	15
特殊文字.....	15
レジスタ名.....	15
アドレッシングモード.....	16
浮動小数点.....	17
命令オペレーションコード.....	17
合成命令.....	17
リンカ.....	18
リンカのコマンドラインオプション.....	18
デバッグ.....	19
MN10300固有のコマンドラインオプション.....	20

マルチスレッドプログラムのデバッグ.....	20
シミュレータ	24
特徴.....	24
シミュレータ固有のコマンドラインオプション	24
シミュレータの使い方.....	26
GDBのシミュレータ例外	27
CygMon、シミュレータ、スレッド認識デバッグ	28
付録 A CYGMON (CYGNUS ROM MONITOR)	31
CYGMONのインストールと作成.....	32
CygMonソースのインストール.....	32
CygMonの作成	34
CYGMONコマンド一覧	35
<i>baud</i>	35
<i>break</i>	35
<i>disassemble</i>	36
<i>dump</i>	36
<i>go</i>	36
<i>help</i>	37
<i>load</i>	37
<i>memory</i>	38
<i>port</i>	38
<i>register</i>	39
<i>step</i>	39
<i>terminal</i>	39
<i>transfer</i>	40
<i>unbreak</i>	40
<i>usage</i>	40
<i>version</i>	40
CYGMONコマンドの編集.....	41
CYGMONのデバッグ.....	42
付表 B: MN10300固有のアセンブラエラーメッセージ.....	43
付表 C: 参考文献.....	45

はじめに

GNUPro Toolkitは、松下MN10300におけるCおよびC++言語開発ツールで、コンパイラ、インタラクティブデバッガ、ユーティリティライブラリなどを含んでいます。さらに、松下MN10300シリーズ評価ボードに対するデバッガ及びリンクのサポートも含まれています。このマニュアルと合わせて、『*Getting started with eCos*』もお読み下さい。

ツールの命名規約

GNUPro Toolkit におけるクロス開発ツールは、ターゲットプロセッサ及び出力されるオブジェクトファイルフォーマットに対応する名前がついています（例：ELF）。このネーミングは、同じバイナリディレクトリに2種類以上のツール、例えばネイティブツールとクロスツールを共にインストールすることが可能です。

完全なツールネームは3つの部分から構成され、それぞれがハイフンで連結された文字列となっています。1番目の部分は特定のプロセッサか、たとえば‘mn10300’のようなプロセッサシリーズを意味します。2番目の文字部分はツールによるたとえば‘elf’のようなファイルフォーマットアウトプットを意味します。3番目の文字部分はたとえば‘gcc’のように一般的なツール名を意味します。たとえば、松下MN10300用のGCCコンパイラは‘mn10300-elf-gcc’となります。

MN10300パッケージには以下のツールが含まれます。

ツールの種類	ツール名
Cコンパイラ	mn10300-elf-gcc
C++コンパイラ	mn10300-elf-c++
アセンブラ	mn10300-elf-as
リンカ	mn10300-elf-ld
スタンドアローンシミュレータ	mn10300-elf-run
バイナリユーティリティ	mn10300-elf-ar mn10300-elf-nm mn10300-elf-objcopy mn10300-elf-objdump mn10300-elf-ranlib mn10300-elf-size mn10300-elf-strings mn10300-elf-strip
デバッガ	mn10300-elf-gdb

Windows NTがホストとなるツールチェインのバイナリは、'.exe'の拡張子がついています。ただし、この'.exe'は実行形式ファイルを実行する際にタイプする必要はありません。

ツールキットの特徴

次にMN10300シリーズ用GNUPro Toolkitの特徴を説明します。

プロセッサのバージョン

松下MN10300シリーズには、MN10300とMN103002があります。これらは、PanasonicかPana-Xのブランド名でも市販されています。このマニュアルで松下か10300の名称が使用されている場合、PanasonicやPana-Xは同一製品であると見なしてください。

サポートするターゲット

GNUPro Instruction Set Simulator

MN10300シリーズ評価ボード

どちらのターゲットもリトルエンディアンモードで実行されます。

サポートするホスト

CPU	オペレーティングシステム	メーカー
x86	Windows NT 4.0	Microsoft
x86	Redhat Linux 5.x	Redhat

オブジェクトファイルフォーマット

MN10300ツールは、ELFオブジェクトファイルフォーマットをサポートしています。『*System V Application Binary Interface*』(Prentice Hall出版、1990年発行)の第4章を参照してください。また、Sレコードを出力する際には、‘ld’ (『*GNUPro Utilities*』の「Using LD」を参照)あるいは、‘objcopy’ (『*GNUPro Utilities*』の「The GNU Binary Utilities」を参照)を使用します。

Windows NT上でのGNUPro

Windows NTでの環境設定

Windows NTホストにおけるツールチェインは、正しく機能させるために以下のような環境設定を行う必要があります。

まず、以下の場所に本リリースがインストールされているものと仮定します。

```
C:¥cygnus¥gnupro¥i386-cygwin32¥mn10300-elf¥ecos-98r1p3
```

その後、以下のように環境設定を行ないます。

```
SET PROOT=C:¥cygnus¥gnupro¥i386-cygwin32¥mn10300-elf¥ecos-98r1p3
SET PATH=%PROOT%¥H-i386-cygwin32¥bin;%PATH%
SET INFOPATH=%PROOT%¥info
REM Set TMPDIR to point to a ramdisk if you have one
SET TMPDIR=C:¥TEMP
```

eCosのソースウェアのリリースについては、eCosにおいてサポートされていないツールもインストールされていますのでご注意ください。これらが以下に示されたデフォルトとなる場所にインストールされていると仮定します。

```
C:¥cygnus¥gnupro¥i386-cygwin32¥i386-cygwin32¥unsupported-98r1p2
```

その後、以下の環境設定を加えます。

```
SET PATH=C:¥cygnus¥gnupro¥i386-cygwin32¥i386-
cygwin32¥unsupported-98r1p2¥H-i386-cygwin32¥bin;%PATH%
```

また、eCos Developer's Kit CD リリース (ソースウェアのリリースには含まれません。)を使用している場合は、以下のようなショートカットをWindowsNTのスタートメニューよりセレクトする事により、環境設定を行なう事ができます。

```
Programs->Cygnus eCos->eCos Development Environment.
```

これは "bash" を実行しているウィンドウを表示し、その結果Windows 環境は自動的にセットアップされます。

大小文字の区別

以下の文字列は Windows NT上で実行する場合、大文字 と小文字の区別が必要です。

- コマンドラインオプション

-
- アセンブララベル
 - リンカスクリプトコマンド
 - セクションの名前

以下の文字列は WindowsNT上で実行する場合、大文字 と小文字の区別が必要ではありません:

- gdb コマンド
- アセンブラ命令,レジスタの名前

WindowsNT 上における大文字と小文字の区別の必要性は、各システム のそれぞれのコンフィギュレーションにより異なります。ただしデフォルトとして、ファイルネームは区別の必要がありません。

Redhat Linux上での GNUPro

Redhat Linux上において使用されるGNUProツールは、ZIPフォーマットのファイルになっています。インストールの詳細は eCos のソースウェアのウェブサイトをご覧ください。

`http://sourceware.cygnum.com/ecos/`

ツールのソースが以下の場所にインストールされていると仮定します。

`/usr/cygnus/ecosSWtools-981021/src`

その後、インストールガイドの指示に従うことにより、以下の場所にツールのインストールが行われます。

`/usr/cygnus/ecosSWtools-981021/H-i386-pc-linux-gnu/bin`

また、シェルの startup スクリプトに以下の設定を行います。

Bourneシェル互換シェルにおいては:

```
PROOT=/usr/cygnus/ecosSWtools-981021
PATH=$PROOT/H-i386-pc-linux-gnu/bin:$PATH
INFOPATH=$PROOT/info:${INFOPATH-}/usr/local/info:/usr/info}
MANPATH=$PROOT/man:${MANPATH-}/usr/local/man:/usr/man}
export PATH INFOPATH MANPATH
```

Cシェル互換シェルにおいては:

```
setenv PROOT /usr/cygnus/ecosSWtools-981021

if ( "$?MANPATH" == "0" ) then
    setenv MANPATH "/usr/local/man:/usr/man"
endif

if ( "$?INFOPATH" == "0" ) then
    setenv INFOPATH "/usr/local/info:/usr/info"
endif

setenv MANPATH $PROOT/man:$MANPATH
setenv INFOPATH $PROOT/info:$INFOPATH
set path = ( $PROOT/H-i386-pc-linux-gnu/bin $path )
```

参考資料

コンパイラ

このセクションでは、GNUProコンパイラにおいてMN10300固有の事項について説明します。

MN10300 固有のコマンドラインオプション

一般的なコンパイラオプションのリストにつきましては、『*GNUPro Compiler Tools*』の「*Using GNU CC*」の“GNU CC Command Options”を参照してください。さらに、以下のMN10300仕様のコマンドラインオプションがサポートされています。

`-mmult-bug`

mn10300 multiply instructionでバグをワークアラウンドするコードを作成します。これがデフォルトです。

`-mno-mult-bug`

mn10300 multiply instructionでバグをワークアラウンドするコードを作成しません。

プリプロセッサのシンボル

デフォルトで、コンパイラによって‘`__MN10300__`’と‘`__mn10300__`’というプリプロセッサシンボルが定義されます。

MN10300固有の属性

MN10300固有の属性はありません。詳細は、『*GNUPro Compiler Tools*』の「*Using GNU CC*」の“Extensions to the C Language Family”の“Declaring Attributes of Functions”と“Specifying Attributes of Variables”の部分を参照してください。

コンパイラとリンカの新機能

GNUProコンパイラとリンカを改良して組み込みターゲット用に開発を行う際に有効な追加機能として、スタートアップと選択的リンク時の初期設定の保証順位が提供されています。

初期設定の優先順位付け

C++においては、静的（スタティック）及びグローバルオブジェクトをコンストラクタにより定義する事も可能であり、また静的及びグローバル変数を初期化する事も可能です。これはコンストラクタがプログラムの他の部分を実行される前に実行されるという事を意味します。しかし、これらのオブジェクトが複数のファイルに点在している場合、C++のスタンダードとしてはこれらのファイルが初期化される順番を指定しないためランダムな順番でこれらが初期化されるという事態が発生します。組み込みシステムとしてはこれは問題であり、そのため静的スレッドがシステムがスタートする前にスタティックスケジューラーオブジェクトが初期化されている事、あるいはデバイスが使用される前に初期化されている事等が必要でした。GNUProはこの問題を防ぐため、静的あるいはグローバル変数が定義された時点でそのプライオリティを定義する事が可能です。以下に例を示します：

```
static object_t myobj __attribute__((init_priority (30000) ));
```

シンタックスはオブジェクトがコンストラクタに引数を渡した場合は、やや異なってきます。

```
static object_t myobj __attribute__((init_priority (30000) )) =  
object_t(arg1, arg2);
```

プライオリティの番号は1から65535の中で選び、その際 1が最も高いプライオリティ、65535が最も低いプライオリティとなります。ちなみにこのような属性を持たないオブジェクトのデフォルト・プライオリティは65535です。最も高いプライオリティを持つコンストラクタはそれよりも低いプライオリティを持つコンストラクタより先に実行されます。

どのような場合でも、引数 '-finit-priority'をコンパイラのコマンド・ラインに渡さない限り、C++のソース・ファイルをコンパイルする際この属性は認識されません。

eCosを使用する場合、eCosは初期設定優先順位をインターナルに行いますのでご注意ください。自分のコンストラクタで参照する前に他のeCosサブ

システムが初期設定しているように正しい優先レベルを選択してください。

選択的リンク

C及びC++のコーディングをする際、ソース・ファイルに複数の関数が含まれる場合がよくあります。たとえば、C++ではある特定クラスの方法を全て同じC++ソース・ファイルに入れることがよくあります。しかし問題としては、今までは一般的にその関数のうち 1つしか使わない場合もそのファイルに含まれている関数の全てが最終的な実行形式ファイルのイメージに含まれてしまうという事があったのですが、組み込みシステムとして、これでは最終的にROMサイズが必要以上に大きくなってしまい、デバッグの際にもそれが全部RAMにロードされる事になってしまいます。

GNUProのC及びC++のコンパイラでは、このような不必要な関数を最終イメージから取り除く事が可能です。そのために、削除される関数でのみ参照される共有グローバルデータを削除いたします。具体的には、コマンドラインに `'-ffunction-sections'` または `'-fdata-sections'` というオプションをC及びC++のコンパイラを呼び出す際に付け加える事によって可能であり、`'-ffunction-sections'` オプションは不必要な関数を取り除き、`'-fdata-sections'` オプションは 不必要なデータを取り除きます。

さらにC++の場合でクラスが仮想メソッドを定義している場合、使われていないメソッドを最終イメージから取り除く事は `'-fvtable-gc'` オプションをコマンドラインにおいてC++コンパイラに渡す事により可能です。

以上、これはすべてのケースに当てはまりますが、リンクする際は必ずコマンドライン・オプションを指示する必要があります。もしリンク ldを直接呼び出す場合は `'--gc-sections'` をコマンドラインに指示する必要があります。また別の方法としては、実行形式ファイルをリンクする場合に、`'gcc -o <program name> <file1>.o <file2>.o'`、というフォームをまず使用し、次に `'-Wl,--gc-sections'` オプションをコマンドライン・オプションに指示する必要があります。以下はその例です。

```
gcc -o prog f1.o f2.o -Wl,--gc-sections
```

EABIのまとめ

このセクションではMN10300アプリケーションバイナリインターフェースについて説明します。

データタイプのサイズとアラインメント

以下の表はすべてのデータタイプのサイズ及びそのアラインメントを示しています：

タイプ	サイズ	アラインメント
char	1 バイト	1 バイト
short	2 バイト	2 バイト
int	4 バイト	4 バイト
long	4 バイト	4 バイト
long long	8 バイト	8 バイト
float	4 バイト	4 バイト
double	8 バイト	8 バイト
long double	8 バイト	8 バイト
pointer	4 バイト	4 バイト

- スタックは4バイトでアラインされます。
- 構造体と共用体はその最も厳密にアラインされたコンポーネントと同じアラインメントを持ちます。

レジスタの割り当て

コンパイラによってレジスタは次の順序で割当てられます。
'd0', 'd1', 'a0', 'a1', 'd2', 'd3', 'a2', 'a3'

レジスタの使用

次の表ではレジスタの使用法を説明します。

タイプ	レジスタ	注記
揮発性	'd0', 'd1', 'a0', 'a1'	
保存	'd2', 'd3', 'a2', 'a3'	
特殊用途	'sp', 'ccr', 'mdr', 'lar', 'lir'	(1) (2)
フレームポインタ	'a3' (必要な場合)	(3)

注記

1. コンパイラでは'ccr'、'lar'、'lir'のいずれかを使用するコードを生成しません。
2. 'mdr'は整数割算と剰余演算のみに使用します。
3. 'a3'はフレームポインタを必要とする関数の中のフレームポインタです。それ以外の場合は、割り当て可能なレジスタです。

スイッチ

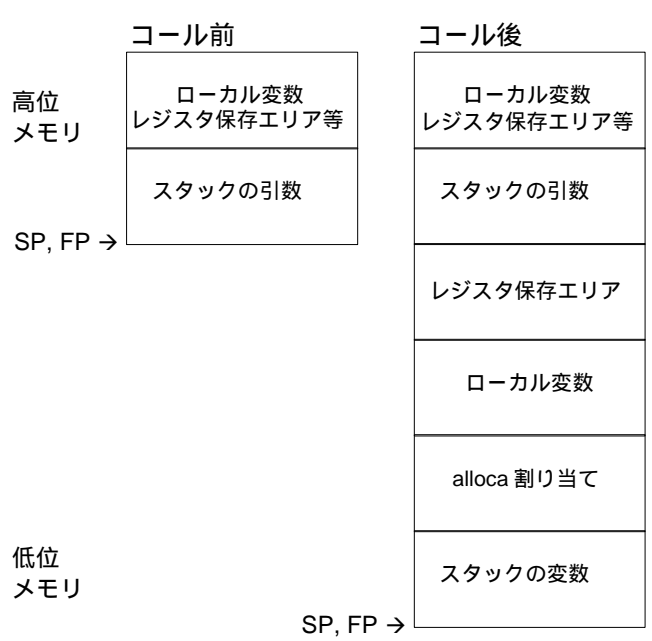
ABIや呼び出し規則に影響を与えるスイッチはありません。コード生成の特定の局面を制御する2つのスイッチがあります。7ページの“MN10300 固有のコマンドラインオプション”を参照してください。

スタックフレーム

ここではMN10300のスタックフレームについて説明します。

- スタックは高位アドレスから低位アドレスへと降順に増えます。
- 葉関数は必要ない場合はスタックフレームを割当てする必要がありません。
- フレームポインタを割当てする必要はありません。
- スタックポインタは必ず4バイト境界に合わせます。

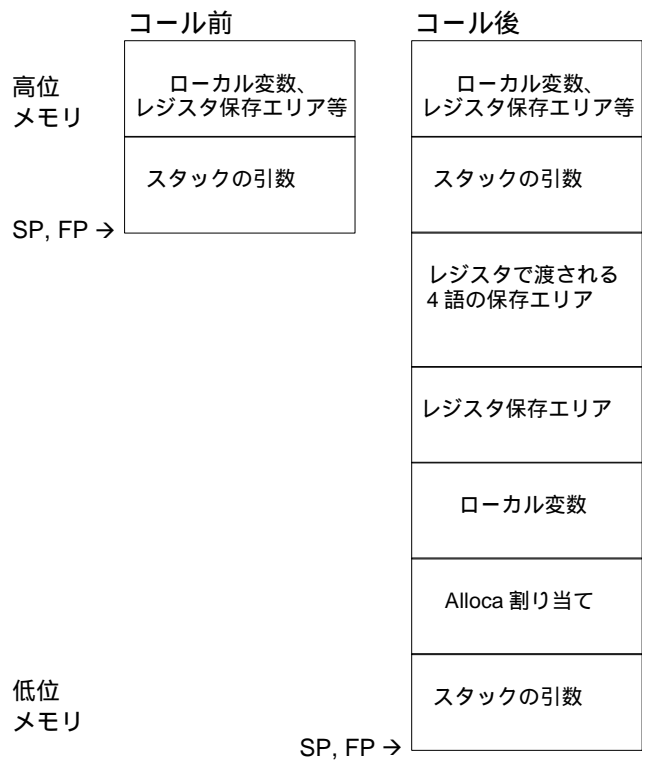
固定数の引数を用いる関数のスタックフレームは次のようになります。



注記:

FPはSPと同じ位置を指します。

不特定数の引数を用いる関数のスタックフレームは次のようになります。



引数の渡し方

‘a0’と‘d1’は最初の2ワードを渡すために使用されます。後に追加される引数のワードはスタック上で渡されます。

8バイトより大きい引数は、見えない参照によって渡されます。呼び出された方が引数を変更する場合は、呼び出されたほうが引数をコピーする責任があります。

関数のリターン値

a0は、ポインタ値を返すために使用されます。

d0とd1は、8バイト以下の長さの他のスカラや構造体を返すために使用されます。

関数が、8バイトより長い構造体を返す場合、呼び出し元は、呼び出された方がそのリターン値を保存するためのロケーションを指定するポインタを呼び出された方に渡す責任があります。このポインタは、関数の宣言されたパラメータの前に、引数の最初のワードとして渡されます。

アセンブラ

MN10300用GNUProアセンブラには、以下のような特別な機能があります。

アセンブラのコマンドラインオプション

一般に使用可能なアセンブラオプションの一覧については『*GNUPro Utilities*』の「*Using AS*」に記載された“Command-Line Options”を参照してください。MN10300固有のアセンブラのコマンドラインオプションはありません。

シンタックス

MN10300シンタックスは、松下の『*MN10300 Architecture Manual*』に記載されたシンタックスに基づいています。

本アセンブラは、「ユーザ定義の命令」や合成命令(複数の機械語命令に相当する擬似命令)をサポートしません。

特殊文字

MN10300アセンブラは、';' (セミコロン)と'#' (パウンド)をサポートしています。カラムゼロで使用するとコメント行文字になります。セミコロンは、行の任意の位置からコメントを書き始めるのにも使用できます。

レジスタ名

d0, d1, d2, d3, a0, a1, a2, a3, sp, mdr, ccr, lir, lar のレジスタ名が、MN10300用にサポートされています。

アドレッシングモード

MN10300用アセンブラのアドレッシングモードは、次のように設定されます。

レジスタ直接

Dm/Dn

Am/An

即値

imm8/regs

imm16

imm32

imm40

imm48

レジスタ間接

(Am)/(An)

レジスタ相対間接

(d8,Am)/(d8,An)

d8は符号拡張

(d16,Am)/(d16,An)

d16は符号拡張

(d32,Am)/(d32,An)

(d8,pc)

d8は符号拡張

(d16,pc)

d16は符号拡張

(d32,pc)

(d8,sp)

d8は符号拡張

(d16,sp)

d16は符号拡張

(d32,sp)

絶対:

(abs16)

abs16はゼロ拡張

(abs32)

インデックス修飾レジスタ間接

(Di,Am)/(Di,An)

‘m’、‘n’、‘i’の各添字はそれぞれ、ソース、デスティネーション、インデックスを意味します。‘m’、‘n’、‘i’の値は0から3までです。

詳細については『*MN10300 Series Instruction Manual*』を参照してください。

浮動小数点

MN10300にはハードウェアによる浮動小数点機構はありませんが、‘.float’と‘.double’ディレクティブで他の開発ツールと互換性を持つIEEE形式の浮動小数点値を生成することができます。

命令オペレーションコード

MN10300機械語命令セットの詳細については『*MN10300 Series Instruction Manual*』を参照してください。GNUアセンブラ(GAS)は標準的なMN10300命令オペレーションコードをすべて実装しています。

合成命令

本アセンブラは、「ユーザ定義命令」や合成命令(複数の機械語命令に相当する擬似命令)をサポートしません。

リンカ

eCosでは、選択したeCosコンフィギュレーションに適したリンクスクリプトを生成します。このリンクスクリプトの使用法は『*Getting Started with eCos*』を参照してください。

リンカのコマンドラインオプション

一般に使用可能なリンカオプションの一覧については『*GNUPro Utilities*』の「*Using LD*」に記載されている「Command Language」を参照してください。MN10300固有のコマンドラインオプションはありません。

デバッガ

このセクションではMN10300固有のGNUProデバッガの特長について説明します。

デバッグの例については、『*Getting Started with eCos*』の「Run an eCos test case」の章を参照してください。

GDBがMN10300ターゲットと通信するには、2つの方法があります。各ターゲットは、ターゲット固有のリンクスクリプトを用いて、プログラムをコンパイルする必要があります。

1. シミュレータ

シミュレータ用のeCosを作成するには、ROMスタートアップで標準評価ボード用か、RAMスタートアップでminimalシミュレータ用のいずれかに構築するように構成します。eCosの‘libtarget.a’ライブラリとプログラムをリンクし‘target.ld’というリンクスクリプトを使って、シミュレータにロードするための最終の実行形式ファイルを作成します。

注記

RAMスタートアップの標準評価ボード用に構築されたバイナリをシミュレータにロードしても作動しません。

GDBでシミュレータを動作させるには、このマニュアルの後半のシミュレータのセクションの手順に従います。

2. リモートターゲットボード

プログラムを標準評価ボードにロードするには、RAMスタートアップのハードウェア用にeCosを構築します。ボードにCygMon ROMか、GDBロードスタブROMが取り付けられている場合、‘target remote <devicename>’(<devicename>に、Windows NT用には‘com2’、Linux用には‘/dev/ttyS1’のようなシリアル装置を代入)というコマンドを使ってGDBをターゲットボードに接続できます。次に‘load’と入力してコードをターゲットボードにロードします。ダウンロード終了後、プログラムは実行可能になります。

注記

リモートターゲットを使用するときには、GDBは‘run’コマンドを受入れません。ただし、プログラムをダウンロードするとPCを開始アドレスに設定するという副次効果があり、‘continue’(‘c’が‘continue’コマンドのショートカット)と入力してプログラムを起動できます。 .

MN10300固有のコマンドラインオプション

一般的に使用可能なデバッガオプションは『*GNUPro Debugging Tools*』の「*Debugging with GDB*」を参照してください。MN10300固有のデバッガコマンドラインオプションはありません。

マルチスレッドプログラムのデバッグ

マルチスレッドプログラムは GDB のグラフィック・ユーザー・インターフェース、GDBTk、または GDB コマンドライン・インターフェースを使用してデバッグが可能です。以下にマルチスレッドを GDB コマンドラインを使用してデバッグする方法を説明しています。

eCos のようなオペレーティングシステムにおいて、1 つのプログラムに実行スレッドが複数ある場合があります。オペレーティングシステムによりスレッドのセマンティクスは異なりますが、一般的に一つのプログラムにおけるスレッドは、一つアドレス空間を共有している点（つまり、同じ変数をどこからでも変更したり参照したりできる）を除いては、マルチプロセスと似ています。逆に、スレッドはそれぞれ自分のレジスタと実行スタックを持っています。自分自身のプライベート・メモリまでも持っている場合もあります。

マルチスレッドプログラムのデバッグを行なうため、GDB には以下の機能が用意されています：

- ‘thread threadno’ スレッド間でスイッチを行うコマンド
- ‘info threads’ 既存のスレッドについて問い合わせるコマンド
- ‘thread apply [threadno][all] args’ スレッドリストの全てに対し同一コマンドを指示するコマンド
- それぞれのスレッド固有のブレイクポイント

GDB スレッド・デバッグ機能を使うと、動作中しているプログラムがの全スレッドを観察できます。ただし、GDB のコントロール下においては特定のスレッドに焦点が定められ、このスレッドをカレント・スレッドと呼びます。デバッグ・コマンドは、このカレント・スレッドから見たプログラム情報を表示します。

また、デバッグのために GDB は自己のスレッド・ナンバーを（常に 1 つの整数）プログラム内のそれぞれのスレッドに関連させます。

```
info threads
```

現在プログラム内に存在するすべてのスレッドの要約を表示します。GDB はそれぞれのスレッドを表示します（順番は以下の通りです）：

1. GDBにより与えられたスレッド番号

2. ターゲットシステムのスレッド

3. スレッドのカレントスタックフレームの概要

アスタリスク(*)が左についたスレッド番号はそのスレッドがカレント・スレッドである事を示します。以下に例を示します。

```
(gdb) info threads
* 2 thread 2 breakme ()
  at /eCos/packages/kernel/v1_1/tests/thread_gdb.c:91
  Name: controller, State: running, Priority: 0, More: <none>
1 thread 1 Cyg_HardwareThread::thread_entry (thread=0x1111aaa2)
  at /eCos/packages/kernel/v1_1/src/common/thread.cxx:68
  Name: Idle Thread, State: running, Priority: 31, More: <none>
```

thread <threadno>

スレッド番号 '<threadno>' をカレント・スレッドにします。コマンド引数の '<threadno>' は、内部GDBスレッド番号であり、'info threads' ディスプレイの最初のフィールドに示されています。GDBは選択されたスレッドにシステム識別子とその時点でのスタックフレームの状態を表示します。以下はその例です。

```
(gdb) thread 2
[Switching to thread 2]
#0 change_state (id=0, newstate=0 '¥000')
  at /eCos/kernel/current/tests/bin_sem2.cxx:93
93     if (PHILO_LOOPS == state_changes++)
Current language: auto; currently c++
```

thread apply [<threadno>][<all>] <args>

thread apply コマンドは複数のスレッドに対しコマンドを指示する場合に使用します。コマンド引数 '<threadno>' を使ってコマンドを指示するスレッド番号を定義します。 '<threadno>' は内部的なGDBのスレッド番号であり、'info threads' ディスプレイの最初のフィールドに示されています。コマンドをすべてのスレッドへ指示したい場合は 'thread apply all args' を使ってください。

GDBがブレークポイントあるいはシグナルによりプログラムを中止するたびに、GDBはそのブレークポイントあるいはシグナルが発生したスレッドを選択します。

プログラムが複数のスレッドを持つ場合、ブレークポイントを全てのスレッドにセットするか、あるいはある特定のスレッドにセットするかを選択できます。

```
break <linespec> thread <threadno>
```

‘<linespec>’ がソース・ラインを指示する場合、指示方法がいくつかあります。修飾子 ‘thread <threadno>’ をブレークポイントコマンドと共に使用することにより、ある特定のスレッドがこのブレークポイントに到達した時点で、プログラムを停止するように指定します。‘<threadno>’ はGDBにより与えられた番号スレッド識別子の1つであり、‘info threads’ディスプレイの最初のコラムに出て来ます。

ブレークポイントをセットした際に ‘thread <threadno>’ を指定しない場合、そのブレークポイントは全てのプログラムに適用されます。

スレッド修飾子は条件付きブレークポイントでも使用できます。この場合は ‘thread <threadno>’ をブレークポイントの条件の前に指示しなくてはなりません。以下はその例です。

```
(gdb) break frik.c:13 thread 28 if bartab > lim
```

GDBにおいてプログラムが停止した場合、すべてのスレッドの実行は停止します。これはカレント・スレッドに限りません。これにより、スレッド間でスイッチするなどの際に、プログラム全体への影響などを心配すること無くユーザーがプログラム全体の状態を調べることができます。

逆に、プログラムを再起動した時点で、すべてのスレッドが実行を開始します。これは‘step’や‘next’などのコマンドを使いシングル・ステップを実行する際も含まれます。特にここで注意しておきたいのは、GDBはすべてのスレッドをロックステップ状態でシングル・ステップを実行することはできないということです。スレッド・スケジューリングはユーザーのデバッグ・ターゲット・オペレーティングシステム次第なので（GDBからはコントロールできません）、その他のスレッドは複数のステートメントを実行する間にカレント・スレッドはシングル・ステップしか終了できない場合もあります。一般的に、プログラムが停止された場合、ステートメントの途中で止まるのが普通で、ステートメントが終了した、きりのよい部分で終了することはほとんどありません。

プログラムは継続後かシングルステップ後にも他のスレッドで停止することがあり、最初のスレッドがリクエストした部分を終了する以前に、そ

他のスレッドがブレークポイント、シグナル、あるいは例外等により停止する場合があります。

シミュレータ

GNUProシミュレータはサポートされているホストコンピュータにおいて、MN10300ターゲットCPU用にコンパイルされたプログラムの実行を行いません。ターゲットCPU命令セットとメモリのシミュレータモジュールが入っている他、シリアルI/Oやタイマをシミュレートする機能が入っている場合もあります。これら機能により、CPUの搭載された実際のボードなしでもプログラマーがMN10300プログラムのテストが可能となります。

MN10300シミュレータはターゲットボードとの間でのタイミングが一致するようにはデザインされていません。たとえば、CPUモジュールは単一のクロック・サイクルを全ての命令に使用するのに加え、メモリもずっと速く、またシミュレートされているシリアルI/Oも比喩にならないほど速く動作します。さらに付加デバイスをシミュレートする際、あまり使用されない機能は省かれています。シミュレータはeCosプログラムを実行する際に必要な複雑さと正確さは持っております。

特徴

MN10300シミュレータは次のレジスタをサポートします。

揮発性レジスタ: d0, d1, a0, a1

保存レジスタ: d2, d3, a2, a3

特殊用途レジスタ: sp, pc, ccr, mdr, lar, lir

メモリは‘0x4800000’の位置で始まり4メガバイトです。スタックは最高位メモリアドレスから始まり降順に続きます。ヒープはテキスト、データ、bssの後に最低位から始まります。

シミュレータ固有のコマンドラインオプション

以下の一般オプションはすべてシミュレータによりサポートされています。

`--board=BOARD`

指定のハードウェア・ボードをモデルにシミュレートするオプションです。mn10300において、‘--board=stdevall’オプションは、mn103002オンボード割込みコントローラタイマー、シリアルI/Oモジュールをも含む、評価ボードの周辺装置のサポートを提供するものであり、ボード上の実際のROMおよびRAMメモリレイアウトに適合します。

`--profile [on|off]`

このオプションでは、プロファイル情報の入った‘gmon.out’というファイルを作成します。またこのファイルは GNU プロファイラである‘gprof’へのインプットとしても使用できます。

`--sockser-addr=HOSTNAME:PORT`

通常使用されるシリアルI/O周辺装置のシミュレーションに代わり TCP/IPソケットを介してシミュレータ・コンソールとデータやり取りを行なうように指示します。TCP/IP ソケット・リスニング・アドレスは引数により定義します。HOSTNAMEはホストのIPアドレスを指示し、PORT は現在 使用されていないポート番号で、1024から65535の間より選択します。telnet、kermit、あるいはsocketなどのプログラムを使用してTCP/IPにアクセスすることも可能であり、また直接シミュレートされているプログラムへアクセスする事も可能です。またシミュレータがgdbスタブを持つ場合は、gdb’s ‘target remote HOSTNAME:PORT コマンドを使用しアクセスする事も可能です。

`--trace=[on|off]`

これはトレース情報が書かれた‘trace.din’ というファイルを作成し、以下で説明される ‘--tracefile’スイッチ を用いて出力ファイル名が変更可能です。

```
C:¥> mn10300-elf-run --trace hello
Hello, world!
3 + 4 = 7
```

以下は上記を実行して作成されたファイルの最初の10行です。

```
2 a0040004 ; width 4 ; load instruction
2 a0040008 ; width 4 ; load instruction
2 a004000c ; width 4 ; load instruction
2 a0040010 ; width 4 ; load instruction
2 a0040014 ; width 4 ; load instruction
2 a0040018 ; width 4 ; load instruction
2 a004001c ; width 4 ; load instruction
2 a0040020 ; width 4 ; load instruction
2 a0040024 ; width 4 ; load instruction
2 a0040028 ; width 4 ; load instruction
```

`--tracefile <file>`

これはトレース情報が書かれるファイルの名前を変更します。

```
C:¥> mn10300-elf-run --trace --tracefile=trace.out hello
Placing trace information into file "trace.out"
Hello, world!
3 + 4 = 7
```

シミュレータの使い方

eCosプログラムはある特定のダウンロード方法をもとに構築されており、最終的にターゲットハードウェアにロードされるプログラムによります。ここで掲げるいろいろな方法により作成されるプログラムも考慮を加えることで動作可能です。その場合、eCos's Hardware Abstraction Layer (HAL)パッケージのコンフィギュレーション・オプションは、いろいろなダウンロード方法をサポートしています。

以下の表は、いろいろな種類のダウンロード方法によって必要となるeCosイメージの処理方法をまとめています。

いろいろなダウンロード方法とHALのコンフィギュレーション:

ダウンロードの方法	HALコンフィギュレーション
ハードウェアのROMに直接焼き付け	ROMスタートアップ
ROMエミュレータへダウンロード	ROMスタートアップ
CygMonがロードされたボードへダウンロード	RAMスタートアップ
CygMonがロードされていないシミュレータにダウンロード	ROMスタートアップ
CygMonがロードされたシミュレータへのダウンロード	RAMスタートアップ
デバイスを無視するシミュレータへのダウンロード	SIMコンフィギュレーション

注意

RAMスタートアップ用にコンフィギュレートされたアプリケーションはシミュレータから直接動かすことはできません。スタートアップの際にクラッシュします。以下で説明するように、シミュレータがCygMonを動かしている場合にのみシミュレータへダウンロードすることができます。

シミュレータ仕様のコンフィギュレーションはシミュレータがエミュレートする必要のあるデバイスドライバやその他監視のためのデバイスを含まないで、ご注意ください。

ほとんどの場合、付加デバイスを無視してシミュレータへダウンロードすることは避けた方が賢明です。そしてこのダウンロード方法を使った場合、コンフィギュレーションによって作成されたバイナリはターゲット・ボード上では動きません;シミュレータ上でのみ起動することができるのです。逆に、ターゲット・ボード用に作成されたバイナリはシミュレータ上で動きます。

また、シミュレータやデバイス・ドライバに問題がある場合、シミュレー

タ仕様のコンフィギュレーションを使ってその問題をうまく避けてと
おるような工夫も可能です。

アクティブなスレッドがないのに、シミュレータの時計やタイマーが非常
にゆっくり動いているように感じられる事があります。何秒かの遅れのは
ずが何分も遅れたりする事もあります。これらは eCos カーネル・アイド
ル・スレッドが非常に忙しいのが原因であり、シミュレータはそれを忠実
にエミュレートしているのです。SIMコンフィギュレーションの際は、
eCosカーネルもこれがシミュレーション環境だという事を認識し、それ
に応じてクロックを合わせる事ができます。

GDBのシミュレータ例外

‘target sim’ コマンドを使いGDBからシミュレータを起動する場合で、シグ
ナルと例外を処理する際に曖昧な状態が現れることがあります。シミュレ
ータから例外が伝えられた場合に、GDBはシミュレーションプログラムが
この例外を処理するのか、あるいはGDBが自分自身でこれを処理するのか
判断が付きません。

たとえば、GDBよりシミュレータを起動してROMモニタのデバッグをしてい
る際に（GDB1ととりあえず呼んでおきます）新たなGDBセッションからア
プリケーションをダウンロードしたとします（これはGDB2と呼びましょ
う）。2回目のGDBセッションのGDB2から見るとシミュレートされたター
ゲットはただのリモート・ターゲットに見えます。ここでこのGDB2がプロ
グラムにブレークポイントをセットしたとします。ブレークポイントは物
理的にはGDB1にセットされるので、このブレークポイントを処理した時点
ではROMモニタからは本物のターゲットのように扱われ、このブレークポ
イントは、GDB1からみるとユーザーからROMモニタコードにブレークポ
イントをセットするように指示されたかのように見えます。このような結果
が好ましくない場合は、GDB1にブレークポイントは自分自身で処理せずに
シミュレートされているターゲットにこれをやらせるよう知らせる事
により解決します。

それには、以下のコマンドを使います：

```
handle SIGxxxx pass nostop noprint
```

‘SIGxxxx’ とはGDBコンソール・プロンプトに‘info handle’ コマンド を入力
した際にGDBがリストするシグナルの 1つです。

たとえば、‘handle SIGTRAP pass nostop noprint’コマンドはGDBに ブレークポ
イントに到達してもシミュレーションを中止しないように指示し、そのか
わりに、その情報をプログラムへ渡すように知らせます。このコマンドは、

ちょっと変更することによって、その他のシグナルや例外にも使えます。

注意:ここで説明しているコマンドを使用する場合、ROMモニタコードにおいてブレークポイントをセットする事は不可能となります(ここにあげた例において)。条件付ブレークポイントを使う事によって、このような問題があっても工夫してそれを解決する事はできます。条件付ブレークポイントについては、GDBマニュアルを参考にしてください。

ここで説明したような曖昧な状態というものは、スタンドアローンのシミュレータをご使用の場合は関係のないものです。この場合ではこのスタンドアローン・シミュレータのみが例外をハンドルできるターゲット・プログラムです。

CygMon、シミュレータ、スレッド認識デバッグ

シミュレータはスレッド認識デバッグをサポートしていません。ただし、CygMonはそれをサポートしているので、シミュレータにその機能を与えるためにはシミュレータの下でCygMonを起動し、そしてCygMon に対して実際ハードウェア上で動いているかのように扱うことです。

この方法は新しい機能をサポートする結果になるのですが、残念ながら欠点もいくつかあります。シミュレータは真のハードウェアに比べるとやはりコードの実行も遅い上に、ROMモニタによるシミュレーションは、コードのダウンロードのスピードをほぼシリアルポートのレベルまで落としてしまいます。加えてシミュレータはCygMonの実行という負荷も抱えているので、GDBのパフォーマンスにも影響してきます。両方のプログラムをメモリにロードしそこから動かすだけのメモリ容量がない場合、メモリ・ページングの影響でスピードが遅くなることもあります。

また、このテクニックはTCP/IPをシミュレータとGDB間のコミュニケーションに使っているので、TCP/IPのプロトコル・スタックがちゃんとコンピュータへインストールされている事を確認する必要があります。

CygMonをシミュレータ上で走らせるには、コマンドラインに以下のコマンドを入力します。

```
mn10300-elf-run --board=stdevall --sockser-addr=localhost:XXXX cygmon.rom
```

‘XXXX’,は現在コンピュータにより使われていないTCPポート番号です。1234という値は使える場合が多いですが、エラーメッセージが出ない以上は1024から65535の間どの番号でも大丈夫です。実行形式ファイルの‘cygmon.rom’ は‘loaders/mn10300-stdevall’ディレクトリのeCosインストールの中を探せば見つかります。さて、これでCygMonを動かすシミュレータ

が起動しました。また別のGDBセッションを走らせて これに繋ぐことにより使えます。

GDBをコマンドライン・モードで動かす場合、次のコマンドを使ってシミュレートされているCygMonを繋げることができます。

```
(gdb) target remote localhost:XXXX
```

ここで使われている ‘XXXX’はシミュレータのコールに答えて与えられたポート番号です。GDBはシミュレータに接続し、実際ハードウェア上で動いているCygMonと話しているかのようにこのCygMonを扱います。

GDBTk を走らせている場合は Target Settings ダイアログに出て来る “ Remote/TCP を“Target”edit フィールドの中で選んでください。 Host edit フィールドへ ‘localhost’と打ち込み、選択したポート番号 の値をポートエントリ へ入れてください。

RAMセットアップがされているターゲットボードにおいて実行形式ファイルが動くようにコンフィギュレーションしてください。ここでシミュレータへコンフィギュレーションしないようにご注意ください。

両プログラムを動かすにはあまりにも負荷が大きすぎる場合は、2台のコンピュータを使う事も可能です。その場合、1台をシミュレータ用に、もう1台をGDB用としてください。この2台はTCP/IPネットワークにおいてつながれている事が必要です。シミュレータを1台で動かして、‘--sockseraddr’オプションの‘localhost’のところにコンピュータの名前を入れてください。GDBをもう1台において動かす際は、シミュレータを動かしているコンピュータの名前を‘localhost’のところにに入れてください。

付録 A

CygMon (Cygnus ROM Monitor)

CygMonはいくつかの組み込みシステムの間で使用出来るようにデザインされた ROM モニタです。従来の GDB プロトコルとも完全な互換性があり、GNU ツールとスタンダードな ROM モニタをいろいろなタイプの組み込みシステムと共にご使用頂けます。

CygMon はデバッグコマンド及び基本的なプログラム機能を持ち、プログラムはメモリにロードして走らせます。またメモリの内容の参照も可能です。コンパイルの際に、逆アセンブラなどのより高度なオプションを付け加えることも可能です。(もちろんコードサイズはそれに依りて大きくなります。)

CygMon には GDB リモートスタブが含まれているため、デバッグは GDB を走らせているホストおよび、CygMon を走らせているターゲットにて、実行可能です。CygMon が GDB とターゲットの通信を検知しスタブモードに切り替え、CygMon モニタモードと GDB スタブモードの切り替えはユーザーが把握できるようになっています。GDB とターゲット間の通信が終了した時点で、CygMon モニタモードに切り替わるようにスタブへ終了パケットを送ります。

コマンドパーサーは CygMon 用に特別に書かれたもので、限られたスペースで必要な機能を提供するようにつくられたものです。すべてのコマンドはコマンド名の後にスペースで区切られた後に引数が続きます。コマンドの略称も使用できます。コマンド名にユニークなサブセットはすべてそのコマンドだとみなされ、たとえば 'du' は 'dump' コマンドとなります。曖昧な場合に解決を求めるプロンプトが表示されます。一般的に、コマンドで必要な引数が全てあるいは一部抜けているものはデフォルトの引数を使用するか、あるいはコマンドによりコントロールされたオペレーションのステータスを返します。

CygMon には API が含まれており、これを利用すれば、その下で実行されるユーザプログラムが様々な機能を持ったシステムコールを使用することができます。シリアルポートやオンボードタイマ機能がある場合は、使用可能なシステムコールによってそれらにアクセスすることが可能になります。

CygMonのインストールと作成

このセクションは eCos Development Kitをお持ちの方にCygMonソースコードのインストール方法、そしてWindows NT オペレーティングシステムにおいてのCygMon ROM モニタ プログラム の作成方法を説明します。eCosのソース・ウェア・リリースをお使いの方は、

<http://sourceware.cygnum.com/ecos/>

をご覧ください。

CygMonソースのインストール

デフォルトとして、GNUPro ツールチェイン ・インストーラーはCygMon ソースコードをインストールしますが、そのインストールをキャンセル等なさないよう強くお勧めします。またインストールされたソース・コードは以下の場所にインストールされます。

```
C:\cygnum\gnupro\i386-cygwin32\mn10300-elf\ecos-98r1p3\cygmon-src
```

ソース・コードをその他の場所にインストールしたい場合は、‘cygmon-src’ディレクトリはそのロケーションから見た ‘ecos-98r1p3\cygmon-src’ にあります。またここでは、コンフィギュレーションや使われた作成・ツールのため、ソース・コードがインストールされるディレクトリの名前には文字、数字、ダッシュ、アンダースコア以外は使用出来ませんのでご注意ください。

ここにあげた例では、全てのインストールは‘c:’ドライブにされる事を前提としています。もしCygMon ソースが自動的にインストールされなかった場合、CDからGNUPro コンパイラ ツール ・インストーラー を走らせることにより後からインストールが行なえます。(‘d:’ は、ご使用されているCD-ROMドライブのドライブレターに書き換える必要があります。)

```
D:\tools\comp\mn10300\setup.exe
```

どこにインストールするか聞かれた場合、場所を選択して下さい。この時、ディレクトリの名前には文字、番号、ダッシュ、あるいはアンダースコア以外は使用しないようにご注意ください。たとえば、

```
C:\ecos-cygmon
```

などが使用出来ます、次に、“CygMon Source Code” のみにチェックがあるよう確認してください。チェックボックスが自動的にいくつかチェック

されていた場合は、“Tools”, “Documenatation” そして “Source Code” などからチェックを取り除いてください。

インストールが終わった時点で、ソースは以下の所にインストールされているはずで

C:\¥ecos-cygmon¥ecos-98r1p3¥cygmon-src

CygMonの作成

スタートメニューから、以下のように選択してください：

Programs->Cygnus eCos->eCos Development Environment

これによって、“bash”を走らせているウィンドウが出て来るはずですが。

ここでソースのロケーションがまだマウントされていない場合は、マウントしてください。

```
mount C: /c
```

さらに、‘bin’:などのサポートされていないツールのインストール・ディレクトリもマウントする必要があります。

```
mount C:/cygnus/gnupro/i386-cygwin32/i386-cygwin32/  
unsupported-98r1p2/H-i386-cygwin32/bin /bin
```

ここでCygMonの作成とコンフィギュレーションができます。

```
mkdir //c/ecos-cygmon/ecos-98r1p3/objdir  
cd //c/ecos-cygmon/ecos-98r1p3/objdir  
../cygmon-src/configure --host=i386-cygwin32 --target=mn10300-elf  
make -w
```

完了した時点で CygMon イメージが作成されています。

```
mn10300-elf/cygmon/eval
```

イメージは：

cygmon.exe

CygMonのRAM常駐バージョンのELF実行形式ファイルです。

cygmon.img

CygMonのバイナリフォーマットのROM常駐バージョンです。

cygmon.rom

CygMonのROM常駐バージョンのELF executableです。スレッド認識デバッグのスタンドアロンシミュレータに使用されます。

cygmon.sre

CygMonのSレコードフォーマットのROM常駐バージョンです。

CygMonコマンド一覧

これはCygMonのコマンドリストの一覧です。引数で[括弧]に入っているものはオプションであり、括弧に入っていないものは必要なものです。また、全てのコマンドはコマンド名を呼び出すのに必要とされる部分(最初から数えて)を打ち込んだ時点でそのコマンドを呼び出す事が出来ます。コマンドの中にはエイリアスがあるものもあり、それらは各コマンドの先頭のアルファベットが特殊なものでない場合に限りそのコマンドのみに使える特殊なアルファベットを指定してあるというものです。コマンドのエイリアスは ヘルプスクリーンには全て表示してあります。

baud

使い方 : baud speed

baudコマンドは使用中のシリアルポートの スピードを指定します。引数は 1つで十分であり、ポートのスピードを指定します。

例 : baud 9600

シリアルポートのスピードを9600baudに指定。

break

使い方 : break [location]

breakコマンドはメモリのブレークポイントを表示および指定します。引数は 1つあるいは 0です。引数が 0の場合、現在指定されているブレークポイントを全て表示します。引数が 1つの場合、ある特定の場所にブレークポイントをセットします。

例 : break 4ff5

メモリロケーション '4ff5' にブレークポイントセット。

disassemble

使い方 : disassemble [location]

disassembleコマンドはメモリの内容を逆アセンブルします。ブレークポイントのハンドリングの方法により命令は全て見えるように、またはブレークポイントは全て見えないようになります。逆アセンブルコマンドの引数は 1つあるいは 0です。引数 0の場合、現在使用中のユーザープログラム pcより逆アセンブルを始めます。アドレスと共に呼ばれた場合、そのアドレスから逆アセンブルを始めます。また、引数無しで前回のコールの後で呼ばれた場合は前回逆アセンブリされた部分の次のアドレスから始めます。逆アセンブリコマンドは著作権問題のため、使えないようにする場合もあります。

例 : disassemble 45667000

メモリロケーション '45667000'から逆アセンブリしたコードを表示します。

dump

使い方 : dump location

dumpコマンドはある特定アドレス前後の16バイトのエリアを16バイトごとに表示します。すなわち 'dump 65' は '60' から '6f'までの 16バイトを表示します。

例 : dump 44f5

メモリロケーション '44f0'から '44ff'までの16 バイトを表示します。

go

使い方 : go [location]

goコマンドはユーザープログラムの実行を開始します。引数は 1つあるいは 0です。引数は与えられない場合、今現在の pcより実行が開始されます。引数を与えられている場合は特定のアドレスより実行が始まります。

例 : go 40020000

'pc' を '40020000'にセットし、プログラムの実行を開始。

help

使い方 : help [command]

helpコマンドは引数が与えられなかった場合、今現在使用出来る全てのコマンドを短い説明も加えてディスプレイします。コマンド名がそのまま引数になっている場合、そのコマンドの使い方及びそのコマンドの説明がされます。使い方はそのコマンドの名前としてディスプレイされ、引数及び拡張子の名前がその後に表示されます。

[括弧]に入った引数は絶対必要なわけではありませんが、テキストとしての引数は必要です。ところで、全てのコマンドはそのコマンドの名前を最初から必要なだけタイプしそのコマンド固有だということが判明した時点でそのコマンドが呼び出されます。エイリアスを持つコマンドもあり、それはそのコマンドを短くしたアルファベット1文字で、そのコマンド固有のもので、エイリアスはヘルプスクリーンに表示してあり、以下のようなフォーマットになっています。

コマンドの名前: (もしあればエイリアス) コマンドの内容の説明

例 : help foo

fooコマンドのヘルプスクリーンを表示します。

load

使い方 : load

loadコマンドはモニターを、全入力をs-recordsとして扱い、メモリへ格納するという状態に切り替えます。 終端コードに達した場合、あるいはエラー(例えば使用出来ないs-record)が生じた場合にモニターはこの状態から抜け出します。

memory

使い方 : `memory[.size] location [value]`

memoryコマンドはメモリの中のある特定の場所を見たりそれに変更を加えたりします。サイズの拡張子も使用可能で、コマンド名あるいはスペースで区切らないでコマンド名の後に続き、ピリオド、そしてビット数という順番で指示します。ビット数としては8、16、32、64が可能であり、またサイズ拡張子が無い場合はデフォルトとして8ビット毎に変更あるいは表示します。

memoryコマンドの引数は1つあるいは2つで、サイズ拡張子が定義されているかどうかとは無関係です。引数が1つの場合はある特定のアドレスの内容を表示し、引数が2つの場合はその特定のアドレス内容を与えられた値に変更します。

例 : `memory.8 45f6b2 57`

メモリロケーション '45f6b2' に8ビットの57という値をセットする。

port

使い方 : `port [port number]`

portコマンドはモニターにより使われているシリアルポートのコントロールをします。引数は1つあるいは0です。引数がない場合、今現在使われているポートを表示します。引数を与えられた場合、現在モニターにより使われているポートを定義したものへ変更します。その後、ポートが変更されたのを確認した上で、新しいポートからメッセージを表示します。

例 : `port 1`

現在モニターにより使われているポートをポート 1に変更。

register

使い方: register [register name] [value]

register コマンドはレジスタの内容を見たり操作したりします。引数は 0、1つ、2つのいずれかです。引数 0でコールされた場合、レジスタコマンドはすべてのレジスタの値を表示します。レジスタの関数の名前のみで呼ばれた場合はその特定のレジスタの内容を表示し、レジスタの名前及びその値の両方でコールされた場合は、その値をその特定のレジスタにセットします。

例: register g1 1f

‘1f’ を ‘g1’ レジスタにセット。

step

使い方: step [location]

stepコマンドはユーザープログラムによりある命令を実行した後にコントロールをモニターへ戻します。引数は 1つあるいは 0です。引数が定義されなかった場合、stepは現在の pcでその命令を実行します。アドレスが定義されている場合はそのアドレスで命令が 1つ実行されます。

例: step

現在の ‘pc’ である 1つの命令を実行。

terminal

使い方: terminal type

terminal コマンドは現在使用中のターミナルのタイプをタイプ引数により定義されたように定義します。ターミナルタイプには 2種あり、vt100とdumbです。この関数はラインエディタがターミナルのディスプレイをアップデートするのに使用します。

例: terminal dumb

現在使用中のターミナルタイプをdumbに変更します。

transfer

使い方 : transfer

transferコマンドや\$ function transfersコマンドはgdbスタブを制御します。この命令はユーザーがコールする必要はなく、ボードをgdbに接続する際に自動的に行なわれます。引数は必要ありません。\$ コマンドはリターンを押す必要はなく、すぐに実行されます。ただしtelnetセットアップがインラインモードである場合でユーザーにより実行された場合に限り、リターンを押す必要があります。(リターンを入力しないとホストが GDB に文字を渡さない) またこの際、gdbがボードからディスコネクトした場合は、自動的にモニターにコントロールが戻ります。

unbreak

使い方 : unbreak location

unbreak コマンドはメモリからブレークポイントを取り除きます。ブレークポイントを取り除きたい場所の引数 1つが必要です。

例 : unbreak 4ff5

メモリロケーション'4ff5'におけるブレークポイントを取り除きます。

usage

使い方 : usage

モニターにより使用されているメモリをカテゴリー別に表示します。コマンド名はまぎらわしいですが、コマンドの使用法を示すものではありません。

version

使い方 : version

version コマンドはモニターのバージョン番号を表示します。

CygMonコマンドの編集

CygMonにはライン編集機能が用意されています。誤ったタイプ入力をした場合、修正が可能です。カーソルを後ろへと動かし、タイプミスした部分を削除します。ユーザーはこれを用いて以前に使用したコマンドの再利用および、再実行が可能です。

以下に示すものは“EMACS” および “X” text editing escape sequences にて、よく使われるものです。キーストロークは以下のようになります。

キー入力	コマンド
CR \backslash n'	現在表示されているコマンドを実行
ctl-a	行の最初へ移動
ctl-e	行の最後へ移動
ctl-f	次のキャラクタへ移動
ctl-b	前のキャラクタへ移動
ctl-k	行の最後まで削除
ctl-y	キャラクタをコピー(消去しない)
ctl-d	キャラクタを削除
ctl-p	前のコマンドを編集
ctl-u	行を削除
ctl-n	次のコマンドを編集

CygMonのデバッグ

CygMonディレクトリにおいて2種類のバイナリを作成する事が出来ます。1つはROMのイメージであり、もう1つはダウンロードプログラムで、これはモニターの監視のもとに走らせるものです。またダウンロードされたCygMonにはシリアルコネクションが必要であり、GDBを用いて接続します。ダウンロードされたCygMonが例外ハンドルのインストールするレベルまで初期化された時点で元のモニターでのデバッグは不可能となりますが、CygMonを使用してその他のプログラムをダウンロードを行ない、走らせる事が可能です。ここでメモリのレイアウトを慎重に行なう事が重要となってきます。CygMonを開発及びデバッグする際、CygMon自体をダウンロード可能なアプリケーションとして取り扱う事が便利な事もあります。つまりプログラマーがプログラムを一時的に変更してしまうのです。たとえば、CygMonが例外ハンドルを扱えないようにコンフィグレーションを変更するか、あるいはテストプログラムのブレークポイントを明示的にコールさせる方法があります。この方法はCygMonの機能を十分発揮出来なくすることになりますが、基本的な機能はすべて使用可能であり、デバッグも可能です。ブレークポイントの管理、メモリアクセスのテスト、レジスタのフェッチ、シリアルドライバをデバッグするなどの際に有効な手段です。しかし、ブレークポイントに到達後の実行の再開やシングルステップの再開、またはそれらのテストは不可能となります。CygMonにコンテキストセーブ及びコンテキスト復帰させる事は可能なので、まず基本的な部分をすべてデバッグし、作動を確認してから例外パッチ機能を復帰させるのが最適でしょう。

付表 B:

MN10300固有のアセンブラエラーメッセージ

Error: Unrecognized opcode

この命令にスペルエラーがあるか、どこかにシンタックスエラーがあります。

Warning: operand out of range

指定された即値が命令に対して過大です。

付表 C: 参考文献

MN10300 シリーズ使用マニュアル (バージョン 1.03, Matsushita Electronics Corporation 著, 1996年11月発行)

MN103000 LSI ユーザーマニュアル (第一版, Matsushita Electronics Corporation 著, 1996年11月発行)

32-bit Microcontroller MN103002A LSI Manual (バージョン 3.0, Matsushita Electronics Corporation 著)

Getting Started with eCos
(Sunnyvale: Cygnus Solutions 著, 1998年)

eCos User Guides
(Sunnyvale: Cygnus Solutions 著, 1998年)

eCos Reference Manual
(Sunnyvale: Cygnus Solutions 著, 1998年)

Getting Started with GNUPro Toolkit
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Compiler Tools
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Debugging Tools
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Libraries
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Utilities
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Advanced Topics
(Sunnyvale: Cygnus Solutions 著, 1998年)

GNUPro Tools for Embedded Systems
(Sunnyvale: Cygnus Solutions 著, 1998年)

System V Application Binary Interface (Prentice Hall 出版, 1990年)

