BN-1 C 言語開発キット

ユーザーズマニュアル

Version 1.00

株式会社バンダイ
有限会社トラスト･テクノロジー

目次

○このマニュアルについて 1
○このマニュアルの表記法 3
第１章　はじめに 4
第２章　準備 6
2.1　インストールと環境設定 6
2.1.1　動作環境 6
2.1.2　機器設定 7
2.1.3　インストール後のディレクトリ構成図 8
2.1.4　インストール手順 10
2.1.5　環境設定（パス設定） 24
2.1.6　起動と終了 26
2.2　ファームウエアの更新 28
2.2.1　インターフェースボードの接続 28
2.2.2　ファームウエアの更新 30
2.2.3　ファームウエアの戻し方 33
2.3　電源管理について 35
第 3 章　BN-1 実行環境について 36
3.1　BN-1 本体内蔵の CPU とメモリについて 37
3.2　ファームウエアについて 39
3.3　ROM ライターについて 41
3.4　H8GCC クロス環境について 43
3.5　カプリロモニターについて 45
第 4 章　プログラミングと実行 48
4.1　プログラムの記述 48
4.2　コンパイル方法説明 51
4.3　プログラムの転送と実行 57
第５章　Ｈ８ＧＣＣの利用 58
5.1 h8300-hms-gcc 58
5.2 h8300-hms-gdb 61
第６章　チュートリアル 65
6.1 手順 65
6.2 準備 66
6.2.1　起動 66

6.2.2　ディレクトリの作成 67
6.3　パスの設定 70
6.4　ファームウエアの更新 72
6.5　カプリロモニター起動 77
6.6　カプリロモニターでサンプルログラムを BN-1 へ転送 78
6.7　カプリロモニターからサンプルプログラムを実行 78
6.8　エディタでプログラムを記述 80
6.9　Makefile の書き換え 81
6.10　作成したプログラムをコンパイルする 82
6.11　カプリロモニターでプログラムを実行 84
6.12　作成したプログラムの改良 84
付録 A　カプリロモニターのコマンド 95
付録 B　UNIX コマンド 98
付録 C　ｖｉの使い方 101
付録 D　注意事項 104
付録 E　トラブルシューティング 109

○このマニュアルについて

このマニュアルでは、BN-1 のプログラム開発環境について解説します。
本マニュアルは本文 6 章と付録で構成されています。各章の内容を以下に示します。

第 1 章「はじめに」
　　BN-1 C 言語開発キット、CD-ROM の構成について紹介します。

第 2 章「準備」
　　インストールと環境設定、ファームウエア更新方法について解説します。

第 3 章「BN-1 実行環境」
　　BN-1 のプログラム実行環境について解説します。
　　実行環境を実現するのに必要なファームウエア、ライター、
　　H8 クロス環境、カプリロモニターについて解説します。

第 4 章「ユーザプログラムの作成」
　　はじめにサンプルプログラムを用いて BN-1 へ転送、実行する方法を解説します。
　　次にソースプログラムの作成からコンパイルまで、
　　プログラムを開発するのに考慮すべき点を踏まえながら解説します。

第 5 章「H8GCC の利用」
　　gcc（h8300-hms-gcc）と gdb（h8300-hms-gdb) の使用方法を解説します。

第 6 章「チュートリアル」
　　ソースプログラムの作成から、コンパイル、プログラムの転送・実行まで
　　一連の操作手順について解説します。

付録 A「カプリロモニターコマンド」
カプリロモニターで使用するコマンドを解説します。

付録 B「UNIX コマンド」
BN-1 C 言語開発キットを使用する上で最低限必要な UNIX コマンドを解説します。

付録 C「vi の使い方」
UNIX の標準エディタである vi の基本的な使い方を紹介します。

付録 D「注意事項」
BN-1 C 言語開発キットを使用する上で注意して頂きたい事項を列挙します。

付録 E「トラブルシューティング」
トラブルが起きたときの対処法です。

○このマニュアルの表記法

$　　UNIX のシェルプロンプト

~　　ホームディレクトリ

/　　UNIX 表記のディレクトリ区切り

¥　　Windows 表記のディレクトリ区切り

␣　　スペース（空白）

［*Enter*］　　Enter キーを押します

第１章　はじめに

本マニュアルでは BN-1 のプログラム開発環境について解説します。

○BN-1 C 言語開発キットとは

BN-1 C 言語開発キットでは C 言語による開発環境を提供します。
開発は Windows（Cygwin 使用）、Linux、FreeBSD 上で動作する GNU の GCC クロス環境を提供することで C 言語によるプログラミングが可能となっています。

ユーザが作成したプログラムはカプリロモニター（BN-1 C 言語開発キットに付属）を使って BN-1 内蔵の SRAM 領域に転送し、実行することが可能です。

BN-1 のモータ制御やグラフィックアイ制御等につきましては BN-1 を容易に制御出来るよう API として関数化していますので、容易に BN-1 を動かすことが可能です。

プログラミングする上で必要なデバック機能も GNU gdb 等を利用することによりソースレベルデバックが可能です。
（注：但し、実機操作用の API 関数は使用出来ません。）
また、BN-1 C 言語開発キットで用意したトレース関数、ブレーク関数の利用により、BN-1 実機レベルでのデバックが可能となっています。

○内容物の確認

BN-1 C 言語開発キットには以下のものが含まれています。

- CD-ROM　　1 枚
- インターフェースボード　　1 枚
- インターフェースユニット接続シリアルケーブル（D-sub 9 ピン）
- 書類
  - 「はじめにをお読みください」
  - 「セットアップガイド」

【CD-ROM のファイル構成】

BN-1 C 言語開発キット付属の CD-ROM は以下のディレクトリ構成となっています。

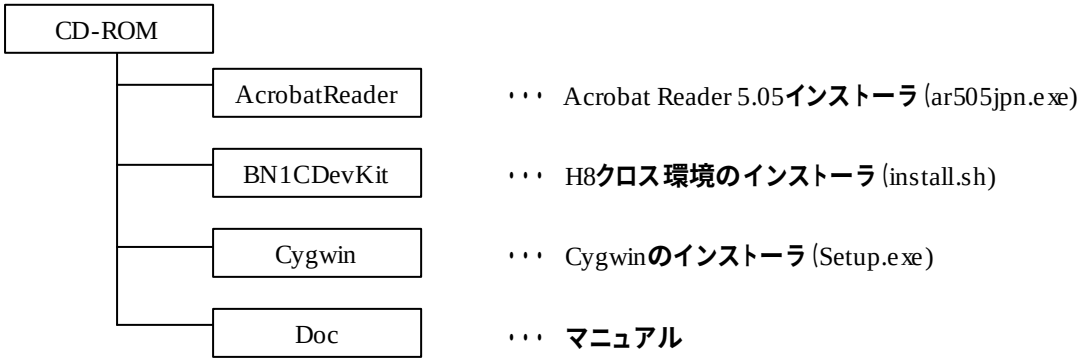


図 1.1　CD-ROM ディレクトリ構成図

（注）CD-ROM 内のマニュアルをご覧になるには、Adobe Acrobat Reader5.01 以上が必要です。
お使いの PC にインストールされていない場合は、BN-1 C 言語開発キット付属の CD-ROM から Acrobat Reader 5.05 をインストールするか下記の URL より Acrobat Reader の最新版をダウンロードして下さい。

http://www.adobe.co.jp/acrobat

第２章　準備

2.1　インストールと環境設定

2.1.1　動作環境

■OS

動作確認済み OS は以下の通りです。

OS	動作確認
Turbolinux Server 6.1	○
Turbolinux Server 6.5	○
FreeBSD 3.4-RELEASE	○
FreeBSD 4.4-RELEASE	○
Windows XP Professional	○
Windows 2000 Professional	○
Windows 98 Second Edition	○
Windows Millennium Edition	○

Windows の場合は、Cygwin をインストールする必要があります。
（Cygwin DLL Version 1.3.4　で動作確認済み）

■CPU

Celeron300 以上を推奨
（Windows の場合、Cygwin を使用しますが Cygwin が少々重い為上記環境を推奨します。
Linux や FreeBSD では、さほど負荷はかかりませんので、これ以下のスペックで可です。）

■メモリ

64M 以上を推奨
（Windows の場合、Cygwin を使用しますが Cygwin が少々重い為上記環境を推奨します）

■ハードディスクの空容量

Windows	Cygwin環境	約300Mバイト(*)
	H8クロス環境	約70Mバイト
Linux, FreeBSD	H8クロス環境	約50Mバイト

(＊)フルインストール時

■その他

CD-ROM ドライブ、D-sub9 ピンシリアルポート

2.1.2 機器設定

○シリアルポートを有効にする（Windows、Linux、FreeBSD）

ご使用のパソコンによってはシリアルポートが有効になっていない場合があります。シリアルポートが有効になっていない場合は BIOS セットアップでシリアルポートを有効にする必要がありますので、ご使用のパソコンのマニュアルに従い設定を行ってください。

○シリアルのデバイスファイルを使用許可に設定する（Linux、FreeBSD）

UNIX では周辺機器をデバイスファイルとして扱いますが、ご利用の UNIX 環境によってはユーザがシリアルのデバイスファイルを使用許可されていないことも考えられます。

シリアルのデバイスファイルが使用出来ない場合には、管理者にご相談し利用出来るようにしてもらうか、root 権限でログインし、chmod コマンドでファイルの許可を変更する必要があります。

（＊）注意！

chmod は危険を伴うコマンドです。

最悪の場合、パソコンが使用出来なくなりますので、正しい知識を持った上で実行して下さい。

【実行例】

所有者、グループ、その他のユーザに読取り書込み権限を与えます。

（１）Linux の場合（但しシリアルポートが COM1（ /dev/ttyS0）の場合）

```
$ chmod␣666␣/dev/ttyS0 [Enter]
```

（２）FreeBSD の場合（但しシリアルポートが COM1（ /dev/cuaa0） の場合）

```
$ chmod␣666␣/dev/cuaa0 [Enter]
```

2.1.3 インストール後のディレクトリ構成図

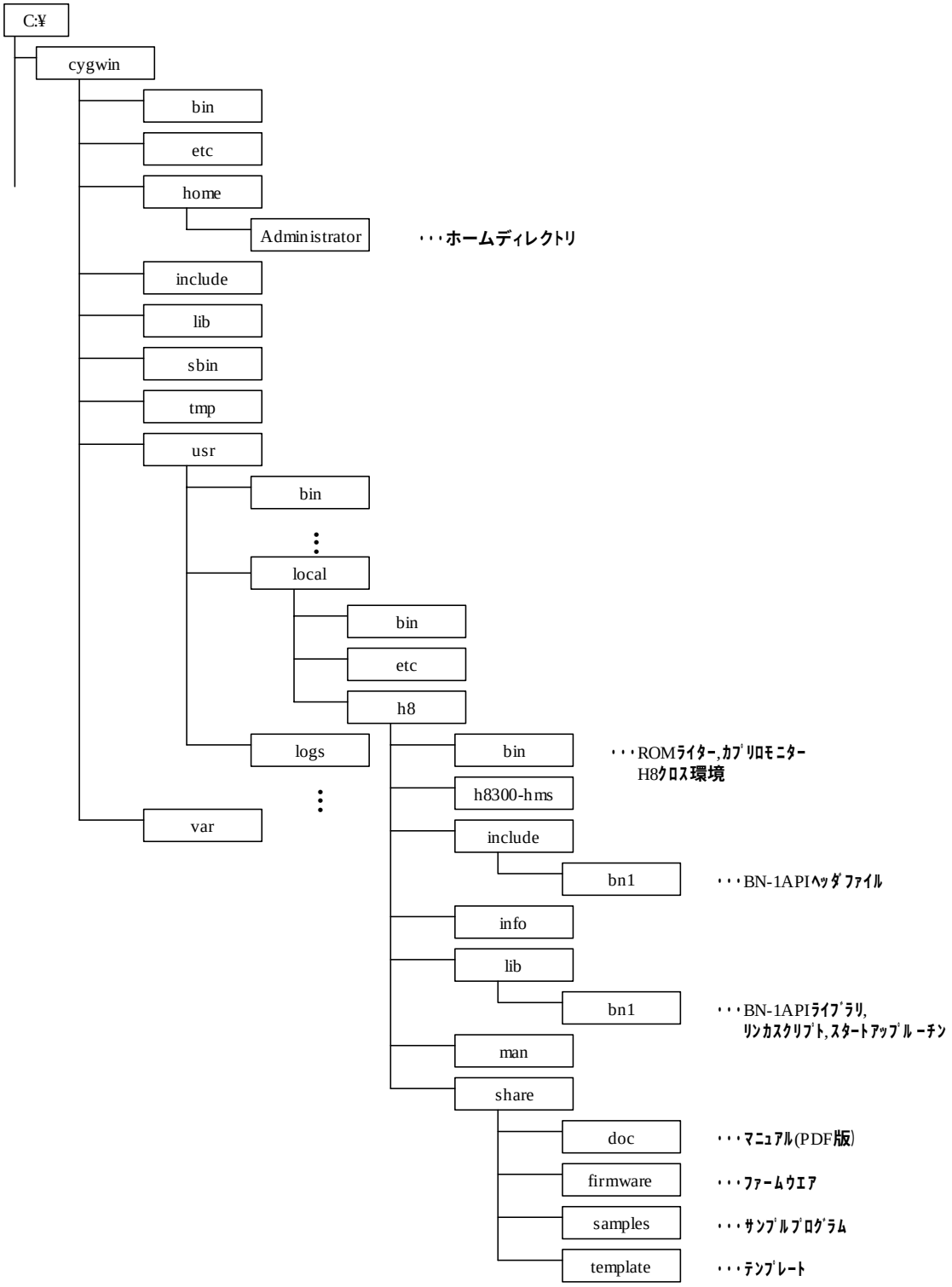


図 2.1　インストール後のディレクトリ構成（Windows）

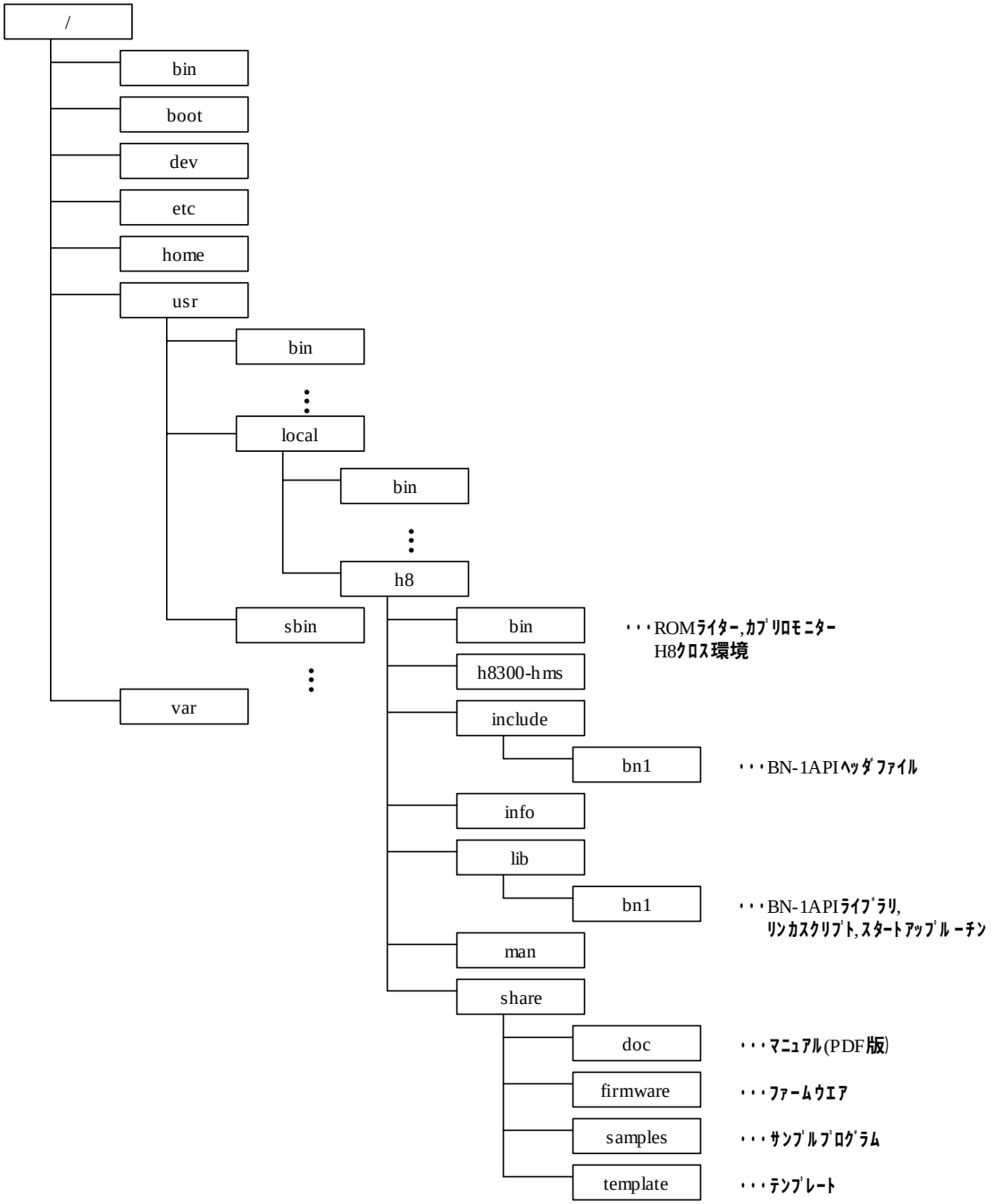


図 2.2　インストール後のディレクトリ構成（FreeBSD / Linux）

2.1.4 インストール手順

2.1.4.1 インストールの前に！

（１）Windows 95/98/Me をご使用の方へ

Cygwin をインストールする際には、ログインするユーザ名に注意する必要があります。Cygwin の環境下ではローマ字表記のユーザでログオンして下さい。
ユーザ名には日本語の使用、スペースの使用をしないで下さい。

（良い例）

taro

（悪い例）

yamada taro　　（スペースが入っている）
山田太郎　　（日本語を使用している）
山田 太郎

もし、現在のユーザ名がローマ字表記でない場合には、
ログオフをしてローマ字表記のユーザでログオンし直してください。

（２）Windows 2000 など Windows NT ベースのＯＳをご使用の方へ

Administrator 権限でインストールを実行して下さい。

（３）アンチウイルスソフトをご使用の方へ

アンチウイルスソフトのリアルタイム検索が有効になっている場合は無効にしてください。インストーラがファイルを展開するときにハングアップする場合があります。

2.1.4.2 Windows 版

○ Cygwin インストール手順

◆ステップ 1

お使いのコンピュータの CD-ROM ドライブに付属の CD-ROM をセットします。

◆ステップ 2

マイ コンピュータから CD-ROM がセットされたドライブを開いてください。(図 2.3)
ここでは CD-ROM ドライブを Q ドライブとします。

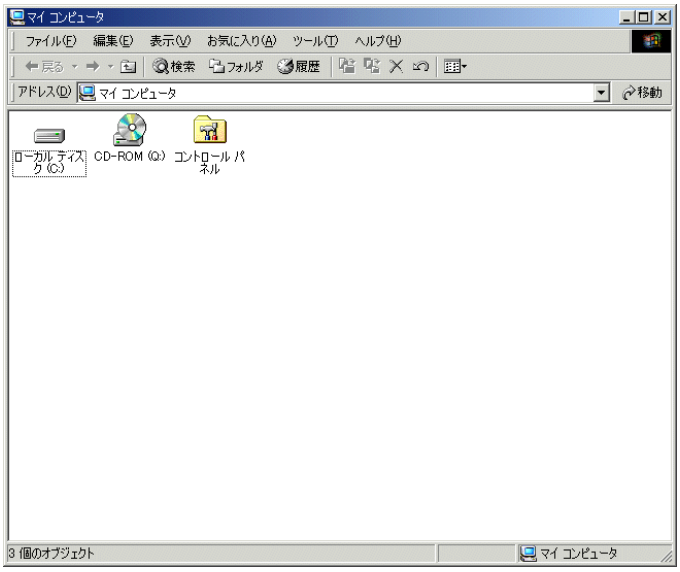


図 2.3

◆ステップ３

CD-ROM ドライブを開くと図 2.4 のようになります。その中にある「Cygwin」フォルダをダブルクリックします。

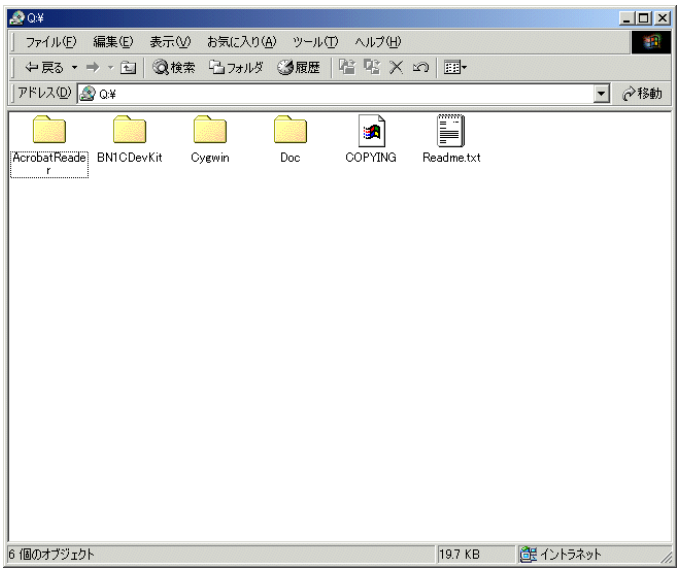


図 2.4

◆ステップ４

「setup.exe」のアイコンをダブルクリックして Cygwin インストーラを起動します。（コンピュータの設定によっては .exe という拡張子が表示されない事があります。）（図 2.5）

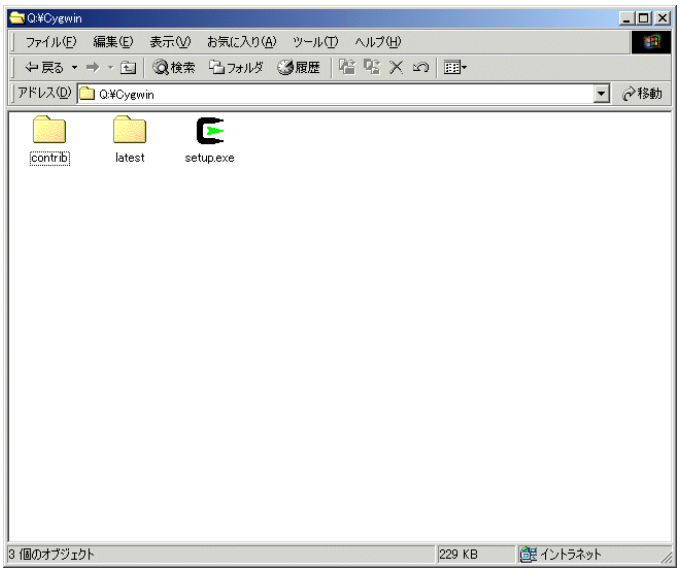


図 2.5

◆ステップ５

Cygwin のインストーラ（setup.exe）を起動すると、最初に図 2.6 のダイアログが表示されますので「Next ‥>」を押して次に進みます。

図 2.6

◆ステップ6

図 2.7 のようなインストーラの行う処理を選択するダイアログが表示されますので、「Install from Local Directory」を選択し「Next ‥>」を押して次に進みます。

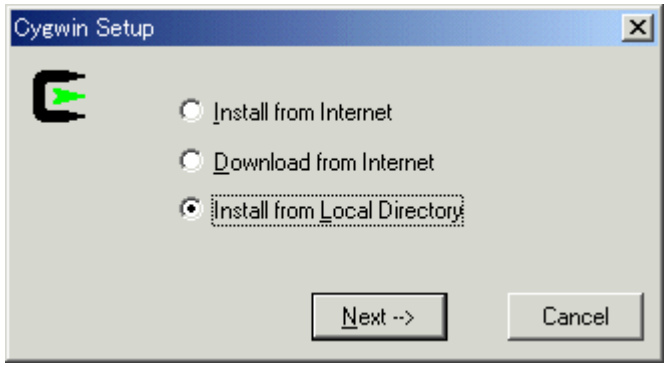


図 2.7

◆ステップ7

図 2.8 のようなインストール元のディレクトリが表示されますので、このまま「Next ‥>」を押して次に進みます。

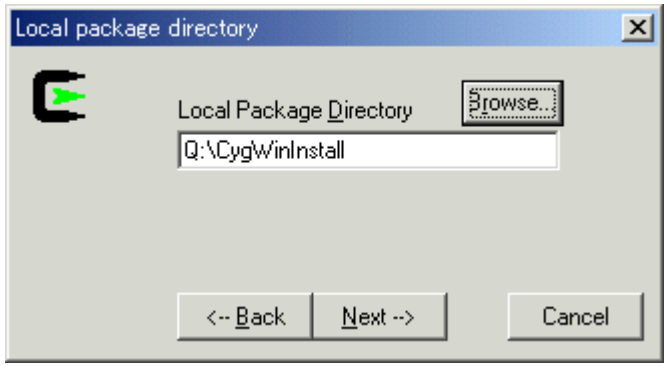


図 2.8

◆ステップ８

図 2.9 のようなインストール先とオプション指定を行うダイアログが表示されますので、このまま「Next ‥>」を押して次に進みます。

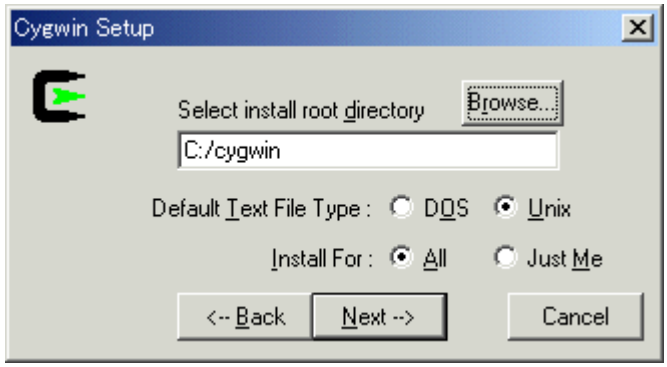


図 2.9

◆ステップ９

図 2.10 のようなパッケージを選択するダイアログが表示されます。

このまま「Next ‥>」を押すと、インストール作業を開始します。

（図 2.10）（図 2.11）

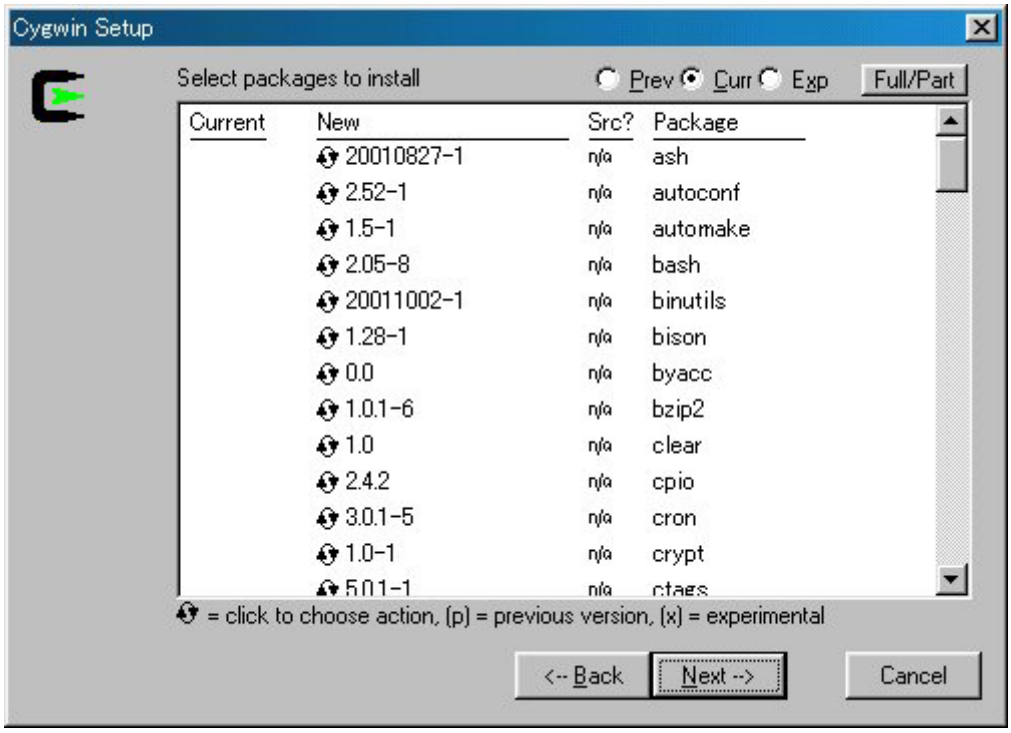


図 2.10

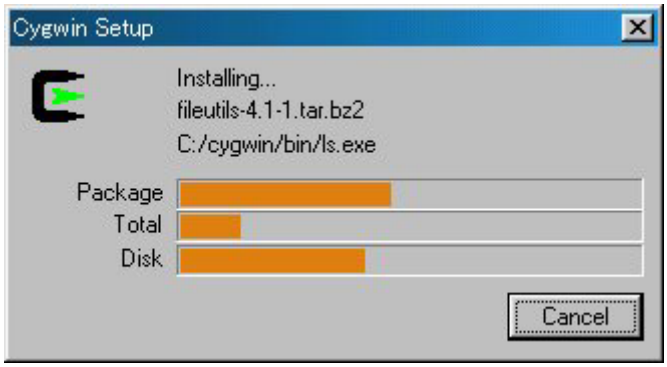


図 2.11

◆ステップ１０

インストールが終わると、図 2.12 のようなスタートメニューの項目とデスクトップアイコンの作成を選択するダイアログが表示されますので、このまま「Next ・・>」を押します。

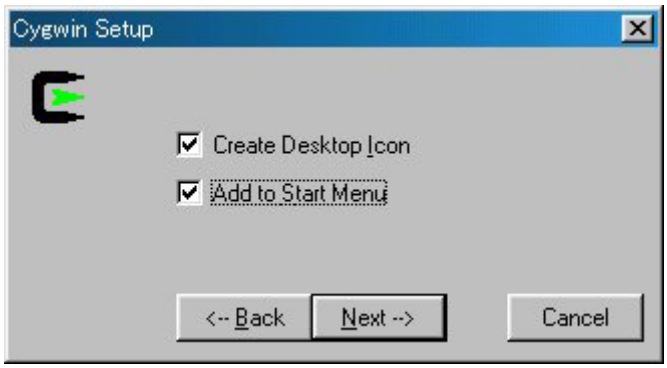


図 2.12

図 2.13 のようなダイアログが表示すれば Cygwin のインストール作業は完了です。「OK」を押してダイアログを閉じて下さい。

図 2.13

試しに Cygwin を起動してみて下さい。デスクトップ上にある Cygwin のアイコンをダブルクリックします。図 2.14 のような画面になりましたか？
Cygwin を終了するには exit コマンドを使用します。
図 2.15 のように exit と打って Enter キーを押してみましょう。Cygwin が終了します。
ここまで上手くいったら、次は H8GCC クロス環境のインストールを行います。

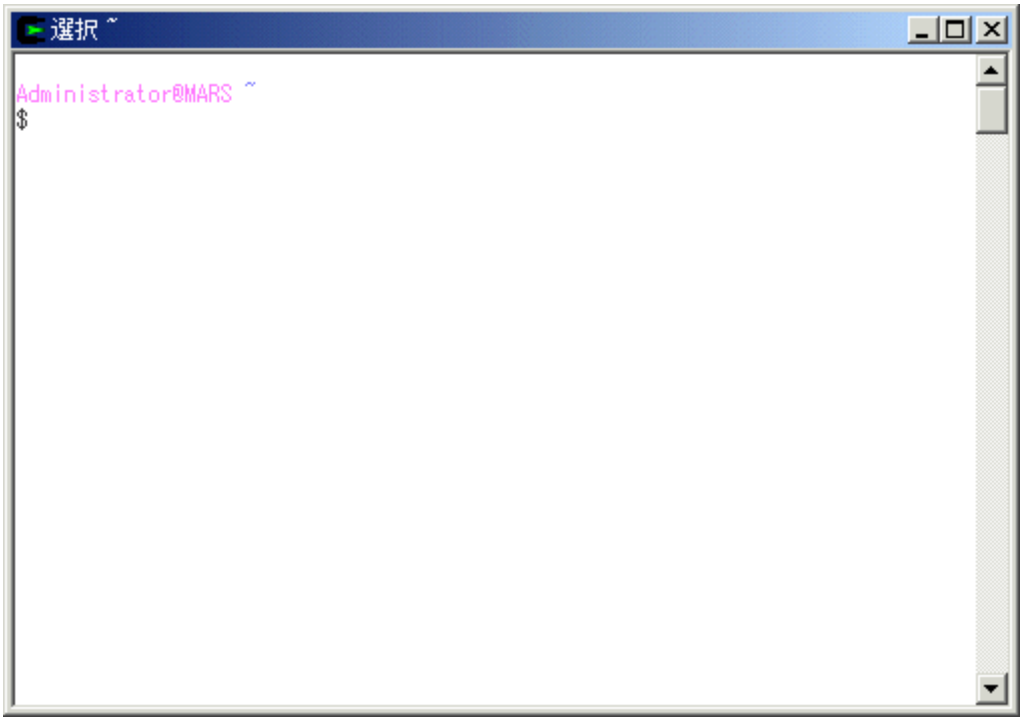


図 2.14

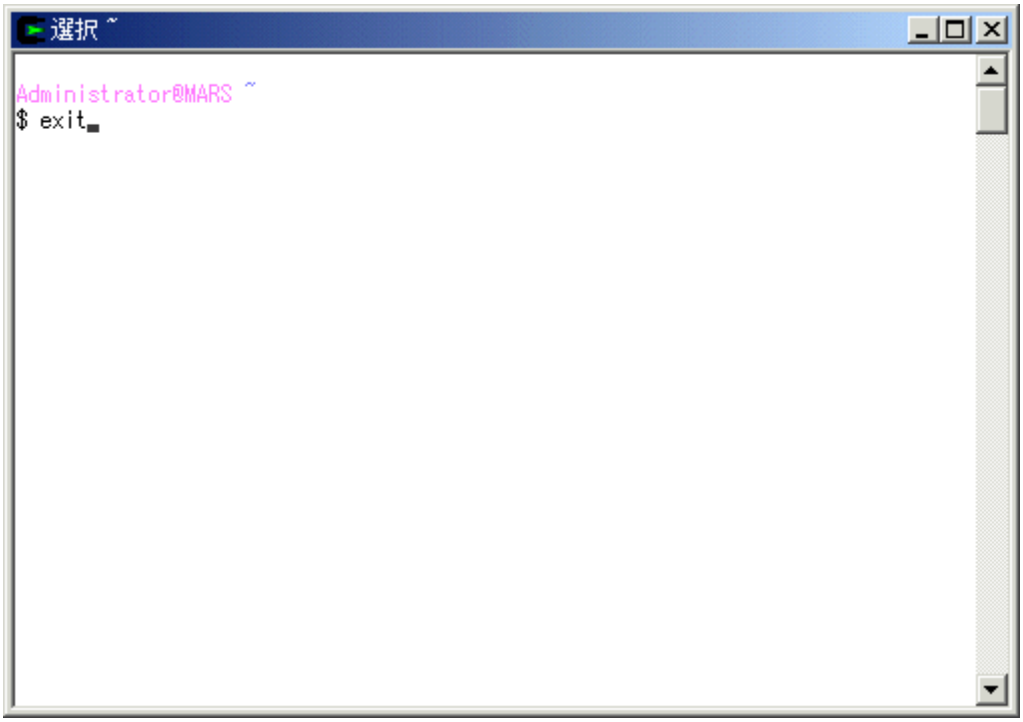


図 2.15

○H8GCC クロス環境のインストール

H8GCC クロスコンパイラ環境のインストールを行います。

（Windows の場合は Cygwin がインストールされている必要があります。）

◆ステップ 1

お使いのコンピュータの CD-ROM ドライブに付属の CD-ROM をセットし、
CD-ROM ドライブへ移動します。

（1）Cygwin の場合

Windows 2000 など Windows NT ベースのＯＳをご使用の場合は、以降の作業も Administrator の権限で行って下さい。

最初に bash を開きます。以降の作業は bash 上で行います。

ex) $ cd␣/cygdrive/q/BN1CDevKit [*Enter*]
(CD-ROM ドライブが Q ドライブの場合)

pwd で移動出来たか確認してみます。

ex) $ pwd [*Enter*]
/cygdrive/q/BN1CDevKit

（2）FreeBSD / Linux の場合

これ以降の作業は root でログインするか su で root 権限となって実行して下さい。

最初に普段お使いのターミナルを開きます。

ex) $ mount␣/mnt/cdrom [*Enter*]
$ cd␣/mnt/cdrom/BN1CDevKit [*Enter*]
（注） CD-ROM のマウントポイントはお使いの環境により異なります。
詳しくは、OS のマニュアルおよび管理者にお問い合わせください。

pwd で移動出来たか確認してみます。

ex) $ pwd [*Enter*]
/mnt/cdrom/BN1CDevKit

◆ステップ２
Ｈ８クロス環境をインストールするには、sh で install.sh を起動します。

```
ex)
$ sh␣install.sh
-----------------------------------------
 Welcome H8 Binary & BN-1 Module Installer
   2001/12/10   (C) Trust Technology, Inc.
-----------------------------------------
Select Install or Uninstall or Serial device setting.  please input No.
    1:     Install
    2:     Uninstall
    3:     Serial device setting
    other No. -> exit
```

上記のようなメニューが出てきます。ここでは、
(1)インストール、(2)アンインストール、(3)シリアルポートの設定
を行うことができます。

インストールを終了または中断したい場合には[CTRL]+[c]もしくは 1～3 以外の番号を入力することで終了することができます。

◆ステップ３

メニューから「Install」を選択し、インストールを行います。

「Install」を選択するには「1」と入力します。

```
ex)
Select Install or Uninstall or Serial device setting.  please input No.
    1:     Install
    2:     Uninstall
    3:     Serial device setting
    other No. -> exit
1                                   <-- 1 を選択する
Select your OS.  please input No.
    1: Cygwin
    2: FreeBSD
    3: Turbolinux
    other No. -> exit
```

◆ステップ４

お使いの OS を選択します。

Cygwin の場合には 1 を選択します。

FreeBSD の場合には 2 を選択します。

Turbolinux の場合には 3 を選択します。

Cygwin を選択したときの例を示します。

```
ex)
Select your OS.  please input No.
    1: Cygwin
    2: FreeBSD
    3: Turbolinux
    other No. -> exit
1                       <-- 1 を選択する
Now start install H8 Binary & BN-1 Modules to /usr/local/h8 sure? (y/[n])
```

y(Yes)か n(No)を聞いてきますので、よろしければ y を入力します。
すると、以下のようにインストールが開始されます。

```
Now start install H8 Binary & BN-1 Modules to /usr/local/h8 sure? (y/[n])y
Install H8 Binary & BN-1 Modules ...
h8/
h8/bin/
h8/bin/capterm.exe
h8/bin/cpp.exe
h8/bin/gcov.exe
h8/bin/h8300-hms-addr2line.exe
        :
        :
        :
h8/share/template/cshrc.default
h8/share/template/Makefile
complete.
```

complete と表示されたら、インストールは完了です。

【シリアルポートの設定】
Linux、FreeBSD でシリアルデバイスを一般ユーザが使用許可にしていない場合には
インストーラーのメニューで 3「Serial device setting」を選択することにより
使用権限を付与できます。 インストーラ内部でシリアルデバイスファイルに
chmod が行われます。事前に手動でシリアルポートのデバイスファイルに chmod を行っている場合や Windows(Cygwin)の場合には、下記操作は不要です。

以下に例を示します。

```
Select Install or Uninstall or Serial device setting.  please input No.
    1:     Install
    2:     Uninstall
    3:     Serial device setting
    other No. -> exit
3                                   <-- 3 を選択する
    Select your OS.  please input No.
        1: FreeBSD
        2: Turbolinux
        other No. -> exit
1                                   <-- 1 を選択する
    Select serial port.  please input No.
        1: COM1
        2: COM2
        3: COM3
        4: COM4
1                                   <-- 1 を選択する
Setting serial device file ...
crw-rw-rw-  1 uucp  dialer   28, 128 Dec 11 07:04 /dev/cuaa0
complete.
```

2.1.5 環境設定（パス設定）

H8GCC クロス環境へのパスを追加します。

○Windows 版（初めて Cygwin を御利用になる方へ）
Cygwin 上で使われるシェルはデフォルトでは bash に設定されていますが、
インストール直後のホームディレクトリには「.bashrc」がありません。
初めて Cygwin を使用する方は /usr/local/h8/share/template にインストールされた「bashrc.default」をホームディレクトリにコピーします。
コピーの際はコピー先のファイル名を「.bashrc」に変更しますので注意して下さい。
コピー例を以下に示します。

ex) $ cp␣/usr/local/h8/share/template/bashrc.default␣~/.bashrc [*Enter*]

ls -a コマンドでホームディレクトリに「.bashrc」があるか確認します。

ex) $ ls␣-a␣~ [*Enter*]

source コマンドでを「.bashrc」を有効にします。

ex) $ source␣~/.bashrc [*Enter*]

○Linux 版、FreeBSD 版、Windows 版（既に Cygwin の環境をお持ちの方へ）
（１）Shell が bash の場合
お使いのエディタで「.bashrc」を開いてください

ex) $ vi␣~/.bashrc [*Enter*]

ホームディレクトリにある「.bashrc」に以下の記述を追加します。
（但し、下記のパスは H8GCC クロス環境のインストール先をデフォルトで行った場合です）

```
if [ -d /usr/local/h8/bin ]; then
      PATH=$PATH:/usr/local/h8/bin
fi
```

ここで行った修正は、次回のログインから有効になりますが、
source コマンドを使用することでログアウトをせずに有効にする事が出来ます。

ex) $ source␣~/.bashrc [*Enter*]

（2）Shell が csh もしくは tcsh の場合
お使いのエディタで「.cshrc」または「.tcshrc」を開いてください

ex) $ vi␣~/.cshrc [*Enter*]

もしくは

$ vi␣~/.tcshrc [*Enter*]

ホームディレクトリにある「.cshrc」または「.tcshrc」に以下の記述を追加します。
（但し、下記のパスは H8 クロス環境のインストール先をデフォルトで行った場合です）

```
if ( -d /usr/local/h8/bin ) then
      setenv  PATH "/usr/local/h8/bin:"$PATH
endif
```

ここで行った修正は、次回のログインから有効になりますが、
source コマンドを使用することでログアウトをせずに有効にする事が出来ます。

ex) $ source␣~/.cshrc [*Enter*]

もしくは

$ source␣~/.tcshrc [*Enter*]

（＊）「.cshrc」のテンプレート
BN-1 C 言語開発キットでは、「bashrc.default」（「.bashrc」のテンプレート）の他に「.cshrc」のテンプレート「cshrc.default」も用意しています。
利用の際はファイル名を「.cshrc」と変更してお使い下さい。

2.1.6 起動と終了

■Windows 版

○Cygwin の起動

パソコンを立ち上げましたら、インストールした Cygwin を立ち上げてください。

Cygwin を起動するには「スタートボタン」→「プログラム」→「Cygnus Solutions」→「Cygwin Bash Shell」

を選択するか、

デスクトップ上に作成された「Cygwin」アイコンをダブルクリックして起動します。

（2.1.4「インストール手順」に従ってインストールされた場合）

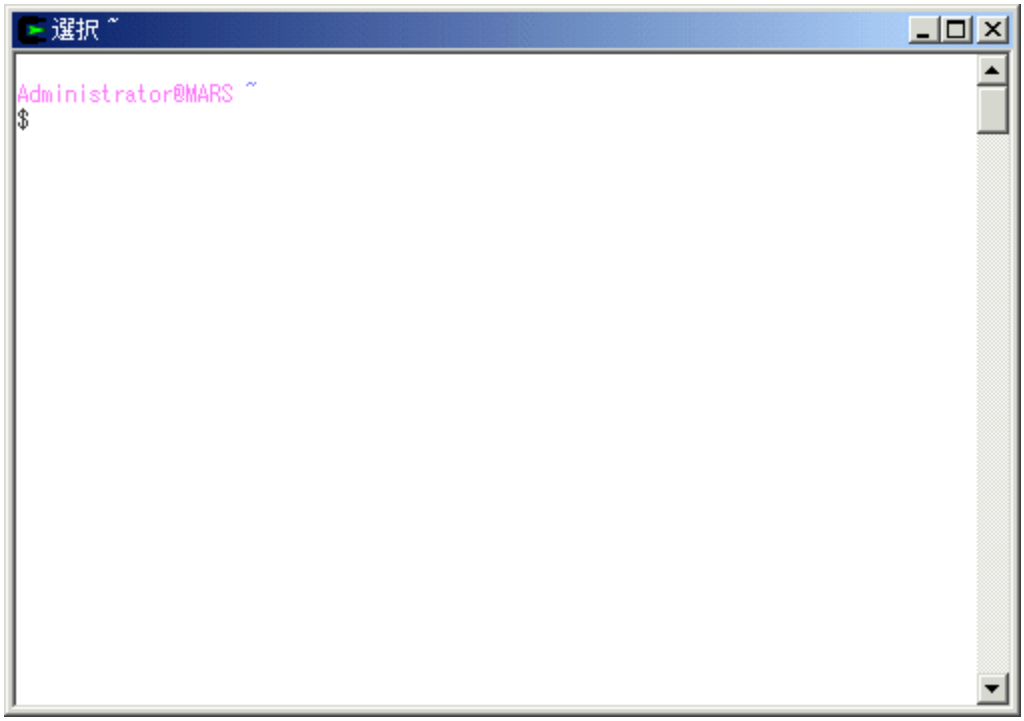


図 2.16 Cygwin 起動時の画面

○Cygwin の終了

Cygwin を終了するには「exit」コマンドを使用します。

ex) $ exit [*Enter*]

○H8 クロス環境へパスが通っているか確認

H8 クロス環境へパスが通っているか確認します。
Cygwin を起動して、echo コマンドで確認する事が出来ます。
引数に環境変数を指定すると、指定した環境変数の値がエコーバックされます。
パスを確認する場合には、引数に PATH を指定します。
以下に例を示します。

```
ex) $ echo␣$PATH [Enter]
        /usr/local/bin:/usr/bin:・・・
                  :
        /usr/local/h8/bin
```

/usr/local/h8/bin にパスが通っている環境では、
上記のように「/usa/local/h8/bin」という記述を確認することが出来ます。
また、echo の代わりに env コマンドで確認することも出来ます。

```
ex) $ env␣|␣grep␣PATH [Enter]
```

■Linux 版、FreeBSD 版

○H8 クロス環境へパスが通っているか確認

H8 クロス環境へパスが通っているか確認します。
ターミナル（kterm 等）を起動して、echo コマンドで確認する事が出来ます。
引数に環境変数を指定すると、指定した環境変数の値がエコーバックされます。
パスを確認する場合には、引数に PATH を指定します。
以下に例を示します。

```
ex) $ echo␣$PATH [Enter]
        /usr/local/bin:/usr/bin:・・・
                  :
        /usr/local/h8/bin
```

上記のように「/usa/local/h8/bin」という記述を確認することが
出来たならば、その環境は /usr/local/h8/bin にパスが通っています。
また、echo の代わりに env コマンドで確認することも出来ます。

```
ex) $ env␣|␣grep␣PATH [Enter]
```

2.2 ファームウエアの更新

2.2.1 インターフェースボードの接続

■インターフェースボード

ファームウエアの書き換え、サンプルプログラム・ユーザプログラムの転送・実行を行うには、パソコンと BN-1 の間を中継するインターフェースボードが必要となります。

【スイッチについて】

インターフェースボードにはスイッチが２つあります。

①電源スイッチ

インターフェースボードに電源を供給するためのスイッチです。

②モード切替スイッチ

「ファームウエア書込みモード」と「プログラム実行モード」のモードを切り替えるためのスイッチです。

「ファームウエア書込みモード」・・・

ファームウエアを更新するときに使用します。

ボード上のスイッチを「ROM WRITE」にセットすると有効になります。

「プログラム実行モード」・・・

ユーザが作成したプログラムを転送、実行するときに使用します。

ボード上のスイッチを「RUN」にセットすると有効になります。

■接続

接続は以下のように行って下さい。

【手順１】

インターフェースボートの電源をＯＦＦにした状態で、付属のシリアルケーブルの一方の端を「インターフェースボードのシリアル端子」に差し込み、もう一方の端を現在ご使用のパソコンの「シリアルコネクタ端子」に差し込んでください。

【手順２】

BN-1 本体の下面（腹部）にはメンテナンスハッチがありますので、マイナスドライバーのような先の平たいもので外してください。

インターフェースボードに接続されているフラットケーブル（6 ピン）の一方の端を「BN-1 本体の下面」に接続します。（図 2.17）

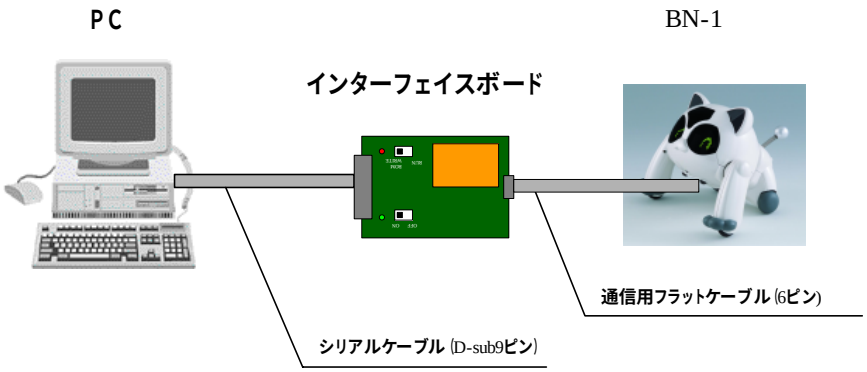


図 2.17　BN-1 と PC の接続

2.2.2 ファームウエアの更新

BN-1 C 言語開発キットをお使いになるにはファームウエアを更新する必要があります。BN-1 本体に内蔵されている H8ROM 領域には、すでにペット用のファームウエアが書き込まれていますが、これを BN-1 C 言語開発キット用のファームウエアに書き換えます。

（＊）注意！
ファームウエアの更新を行いますと、ペットモードで育ててきた今までの経験等が消去されてしまいます。ファームウエアを戻すことで、ペットモードに戻す事が出来ますが、起動時は製品出荷時と同じ状態になりますので御了承下さい。

■更新手順

（１） インターフェースボードが（図 2.17）のように正しく接続されているか確認してください。

（２）インターフェースボードの切替スイッチを「ROM WRITE」にセットします。

（３）インターフェースボードの電源スイッチを ON にします。
インターフェースボードの電源スイッチの横にある緑色の LED と切替スイッチの横にある赤色の LED が点灯することを確認して下さい。

（４）ROM ライターでファームウエアを転送します。
ファームウエア（capmon-v100.mot）は
/usr/local/h8/share/firmware
にインストールされています。

（＊）注意！
ファームウエアの更新には数分かかります。ファームウエア更新の進行状況は画面上に１６進数でダンプ出力されますので、trustwrite から書き込み完了メッセージが出力されるまでしばらくお待ちください。この間、trustwrite を中断させたり、BN-1 本体やインターフェースボード電源スイッチの ON/OFF を行わないでください。

転送方法は以下のとおりです。

①Windows 版

Cygwin を起動し、次のようにコマンドを入力します。

（但し、シリアルポートが COM1 の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣-c␣COM1[Enter]
```

②Linux 版

ターミナル（kterm 等）を起動し、次のようにコマンドを入力します。

（但し、シリアルポートが COM1（/dev/ttyS0）の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣
    -c␣/dev/ttyS0[Enter]
```

②FreeBSD 版

ターミナル（kterm 等）を起動し、次のようにコマンドを入力します。

（但し、シリアルポートが COM1（/dev/cuaa0）の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣
    -c␣/dev/cuaa0[Enter]
```

（＊）注意！

ダンプが中断して更新が止まってしまう場合や trustwrite が停止したままになる場合には、[CTRL] + [c] を押して強制終了させてください。（[CTRL]キーを押しながら[c]キーを押します）

（５）おめでとうございます！ 以上でファームウエアの更新は完了しました。

書き込みが完了すると、trustwrite から書き込み完了メッセージが出力されます。

ここで、インターフェースボード上のスイッチを「RUN」に切り替えますと、

BN-1 のグラフィックアイが表示されますので、確認してください。

この状態で、パソコンからのシリアル通信待ちとなります。

【ROM ライター実行例】

ROM ライター実行例を以下に示します。

パソコンのモニターには以下のような表示がされます。

（Windows 版でシリアルポートが COM1 の場合）

ex)

```
$ trustwrite /usr/local/h8/share/firmware/capmon-v100␣-c␣COM1[Enter]
trustwrite version 1.02 TrustTechnology, Inc.
  interface type:     BN-1 C language development kit
  serial device file: COM1
  serial speed:       19200
H8/3067F(BN-1/AKI) boot mode ready!
Transfer H8 ROM Writer
  writing...   0 %
  writing...  10 %
  writing...  20 %
          :
          :
```

ROM ライターが無事に実行された場合、書き込む ROM イメージの
ダンプが下記のように表示され、最後に書き込み完了の表示がされます。

ex)

```
          :
[14560] 96 00 00 81 ff ff ff ff ff ff ff ff ff ff ff ff
[14570] ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
H8 EEPROM writing complete.
```

trustwrite のダンプが中断する等してファームウエアの書き込みがうまくいかない場合には、もう一度シリアルポートの BIOS 設定と機器の接続を確認してください。それでもうまくいかない場合には –s 9600 オプションを指定して、より低速な転送速度でお試しください。

2.2.3 ファームウエアの戻し方

BN-1 をペットモードで楽しみたい場合には、ファームウエアを戻す必要があります。
ファームウエアを戻す手順を以下に示します。

■ ファームウエア戻し手順

（1） インターフェースボードが（図 2.17）のように正しく接続されているか確認してください。

（2）インターフェースボードの切替スイッチを「ROM WRITE」にセットします。

（3）インターフェースボードの電源スイッチを ON にします。
インターフェースボードの電源スイッチの横にある緑色の LED と切替スイッチの横にある赤色の LED が点灯することを確認して下さい。

（4）ROM ライターでファームウエアを転送します。
ファームウエア（bn1org-v03.mot）は
/usr/local/h8/share/firmware
にインストールされています。

転送方法は以下のとおりです。

①Windows 版
Cygwin を起動し、次のようにコマンドを入力します。
（但し、シリアルポートが COM1 の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/bn1org-v03.mot␣-c␣COM1[Enter]
```

②Linux 版
ターミナル（kterm 等）を起動し、次のようにコマンドを入力します。
（但し、シリアルポートが COM1（/dev/ttyS0）の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/bn1org-v03.mot␣
  -c␣/dev/ttyS0[Enter]
```

②FreeBSD 版

ターミナル（kterm 等）を起動し、次のようにコマンドを入力します。

（但し、シリアルポートが COM1（/dev/cuaa0）の場合）

```
$ trustwrite␣/usr/local/h8/share/firmware/ bn1org-v03.mot␣
  -c␣/dev/cuaa0[Enter]
```

（５）メモリクリアを行います。

メモリクリアの手順は以下のとおりです。

BN-1 本体背中に POWER ボタンがあります。

①POWER ボタンを指で押したままの状態にします。

②ツマヨウジ（または金属製ではない細長い棒状のもの）でリセットスイッチを、奥まで（カチリという手応えがあるまで）押します。

③POWER ボタンを押したままリセットボタンからツマヨウジを離します。

④さらに POWER ボタンをそのまま約 5 秒間押し続けると、

BN-1 本体から「チーン」という音が鳴ります。

⑤POWER ボタンを離し、操作完了です。

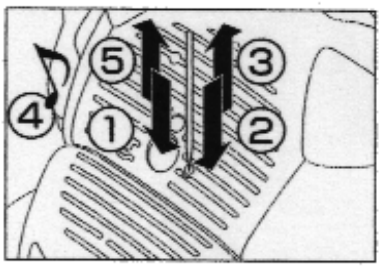


図 2.18　メモリクリア

（６）ファームウエアは元の状態に戻りました。

BN-1 本体背中の POWER ボタンを押しますと、BN-1 は動き出します。

2.3 電源管理について

○BN-1 本体の電源の ON/OFF

BN-1 の背中の POWER ボタンを押すと電源が ON の状態になります。
もう一度、POWER ボタンを押すと、電源が OFF（スリープモード）の状態になります。

BN-1 の電源仕様及び注意事項は以下のようになっています。使用方法を守って正しくお使いください。

（1）充電方法

BN-1 専用バッテリー充電器を BN-1 本体の充電ジャックに差し込むと、自動的にバッテリー充電を開始します。
８時間経つと充電は完了し、「ピピッ」とアラーム音が鳴ります。この時点で充電完了ですので、充電ジャックを外してください。
また、充電は、BN-1 本体の電源をＯＦＦにしてから行ってください。電源を入れた状態のまま充電ジャックに差し込みますと、充電器保護のため自動的にＢＮ－１本体の電源はＯＦＦになります。

（2）充電中の電源 ON について

BN-1 本体の充電ジャックに BN-1 専用バッテリー充電器をつけたまま、BN-1 本体の電源を ON にすると、充電器に多大な負荷がかかりますので、このような使い方はおやめください。

（3）充電器保護機能

充電ジャックに BN-1 専用バッテリー充電器を差し込んでから電源が ON にされた場合、30 分以上カプリロモニターとの通信がない状態が続きますと、充電器保護の為に自動的に BN-1 本体の電源は OFF になります。

第 3 章　BN-1 実行環境について

BN-1 実行環境とはユーザがパソコン上でプログラムを作成し、作成したプログラムを BN-1 内蔵の SRAM 領域に転送して、パソコン上から BN-1 を操作出来るような環境をいいます。

ユーザプログラムを作成し、BN-1 を動かすまでの流れは以下のようになっています。

①エディタでプログラムソースを記述
②H8GCC クロス環境でコンパイル
③カプリロモニターを起動
④カプリロモニターからプログラムを転送
⑤カプリロモニターからプログラムを実行

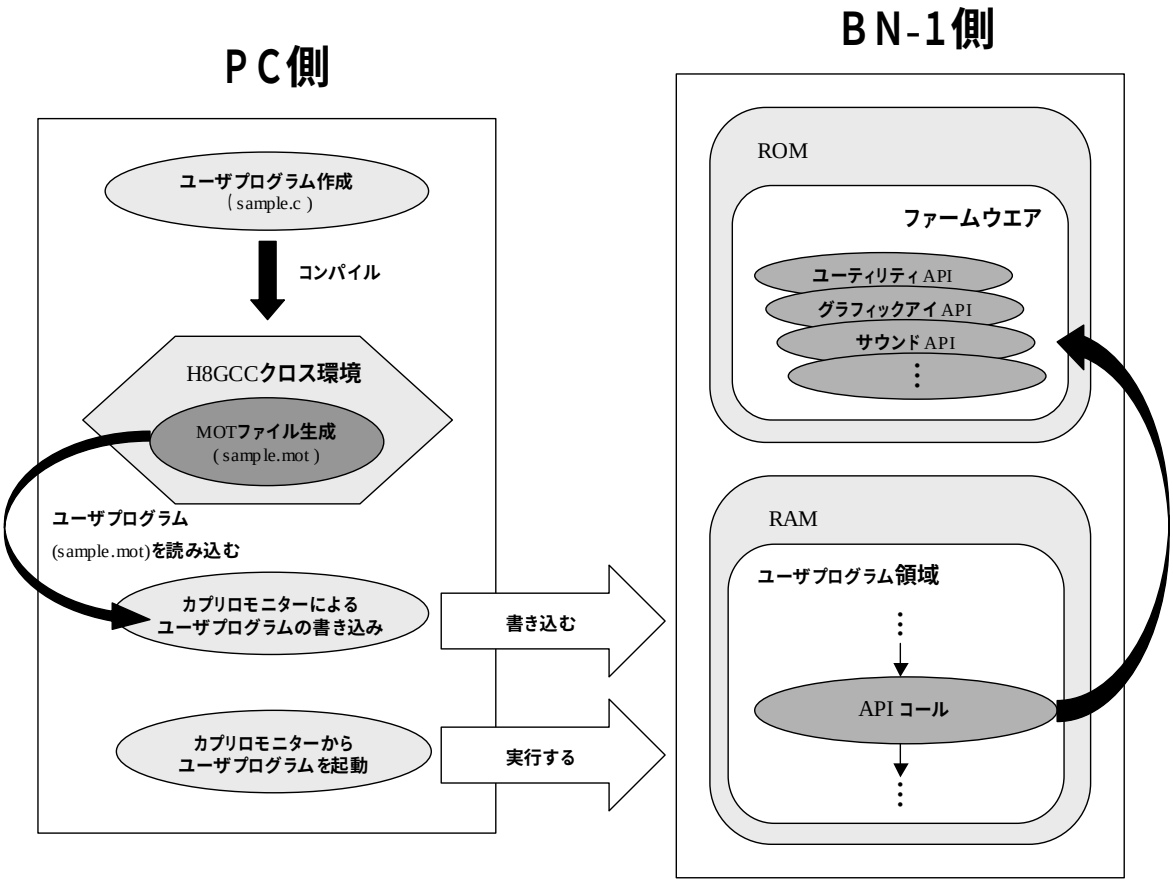


図 3.1　BN-1 実行環境

＊ H8･･･

H8 とは㈱日立製作所が開発したマイクロコンピュータです。
BN-1 の CPU として H8 シリーズである H8/3067F を使用しています。

＊ MOT ファイル･･･

H8 用プログラムのバイナリイメージをモトローラ形式に変換した ASCII テキストファイルです。BN-1 C 言語開発キットでは、そのうちモトローラ S2 形式を使用しています。

＊ BN-1 API･･･

BN-1 API は、ファームウエアとして ROM 領域に配置され、RAM 領域に配置されるユーザプログラムからコールされることにより、BN-1 本体の容易なコントロールを可能にするライブラリです。

3.1 BN-1 本体内蔵の CPU とメモリについて

BN-1 本体には㈱日立製作所製の CPU H8/3067F と H8/3067F に接続された外部 SRAM（64k バイト）が内蔵されています。H8/3067F には 128k バイトの ROM と 4k バイトの内蔵 RAM があります。BN-1 C 言語開発キットの BN-1 実行環境では、H8/3067F の ROM にはファームウエアが書き込まれ、内蔵 RAM にはファームウエア起動時に使われる変数等を格納する領域として使用されます。
ユーザが利用出来る領域は H8/3067F に外付けされた 64k バイトの外部 SRAM となっています。
BN-1 のメモリマップを図 3.2 に示します。
ユーザが利用出来るメモリ容量はインストール時のデフォルトのリンカスクリプトにてプログラム領域 48Ｋバイト、変数領域 16Ｋバイトとして設定されています。
（リンカスクリプト： /usr/local/h8/lib/bn1/bn1ram.x ）

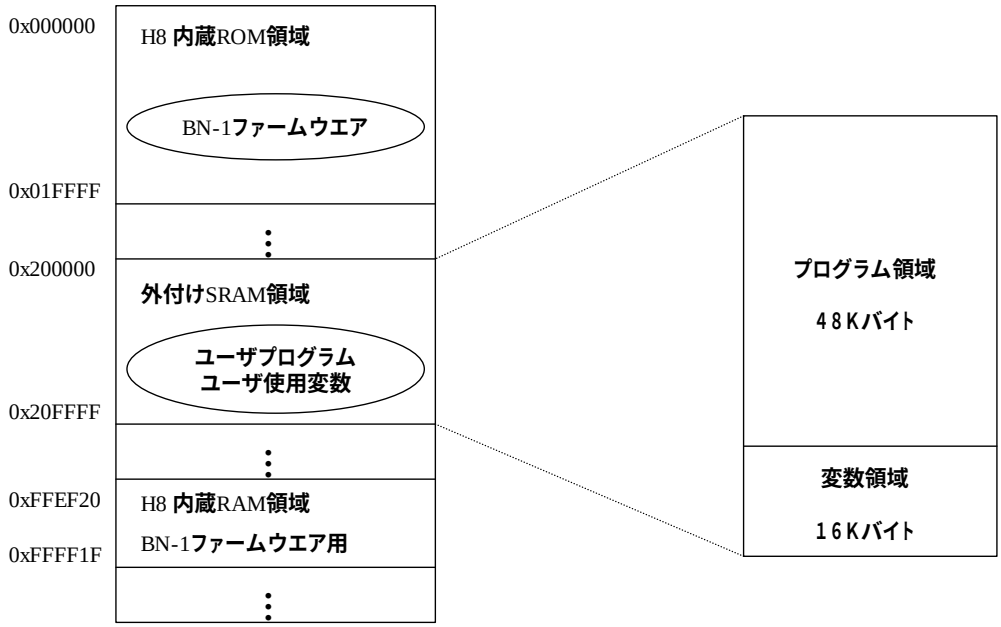


図 3.2 メモリマップ

○H8/3067

H8/3067 シリーズは、日立オリジナルアーキテクチャを採用した H8/300H CPU を核にして、システム構成に必要な周辺機能を集積したシングルチップマイクロコンピュータ（MCU）です。

H8/300H CPU は、内部 32 ビット構成で 16 ビット×16 本の汎用レジスタと高速動作を指向した簡潔で最適化された命令セットを備えており、16M バイトのリニアなアドレス空間を扱うことができます。

システム構成に必要な周辺機能としては、ROM、RAM、16 ビットタイマ、8 ビットタイマ、プログラマブルタイミングパターンコントローラ（TPC）、ウォッチドッグタイマ（WDT）、シリアルコミュニケーションインターフェース（SCI）、A/D 変換器、D/A 変換器、I/O ポート、DMA コントローラ（DMAC）などを内蔵しています。

H8/3067 には、128k バイトの ROM と 4k バイトの RAM が内蔵されています。

H8 のレジスタはリニアなアドレス空間にありますので、ユーザプログラムからレジスタ操作をすることも可能ですが､BN-1 C 言語開発キットではレジスタ操作を BN-1API でのみ行うものとしておりますので、万一、ユーザがユーザプログラムレベルで H8 のレジスタを操作し、BN-1 を動作させる場合につきましては保証外となりますので予め御了承下さい。

3.2　ファームウエアについて

ファームウエアとは機器を動かすために必要なプログラムで、ROM 領域に書き込まれています。普段、BN-1 がペットのように振舞うのは BN-1 の ROM 領域にペットモードのファームウエア(bn1org-v03.mot) が書き込まれているからです。
BN-1 C 言語開発キットで提供する BN-1 プログラム実行環境を構築するには、
ファームウエアを BN-1 C 言語開発キット専用のファームウエア(capmon-v100.mot)に書き換えます。

○BN-1 C 言語開発キット用ファームウエア

BN-1 C 言語開発キットのファームウエア機能を大別すると、以下ような特徴があります。

①BN-1 とパソコン間で通信可能

BN-1 とパソコン間でシリアル通信を行う機能があり、パソコン上からカプリロモニターで通信できます。

②BN-1 API を実装

ユーザがプログラミングする上で BN-1 を容易にコントロール出来るライブラリ（BN-1 API）が、ファームウエアに格納されています。ユーザは、自作のプログラム領域から「BN-1 API」を呼び出すことで、BN-1 本体をコントロールすることができます。

③ユーザプログラムの転送

カプリロモニターを用いてユーザプログラムを転送すると、ユーザプログラムを自動的に BN-1 の SRAM 領域へ転送するような仕組みとなっています。

④ユーザプログラムの転送と実行

カプリロモニター上で SRAM 領域に転送したプログラムを実行する事が出来ます。

⑤カプリロモニターの各種コマンドの利用

カプリロモニターで用意したコマンドを使用することで、BN-1 を制御することが出来ます。

○BN-1 API(Application Programming Interface)

BN-1 API は､RAM 領域に配置されるユーザプログラムからコールされることにより、BN-1 本体の容易なコントロールを可能にするライブラリです。

BN-1 API の制御ロジックはフラッシュ ROM にファームウエアとして格納されており、API インターフェースにより、ユーザプログラム領域を圧迫することなく利用できます。

BN-1 API の機能を大別すると、以下の７種類に分類されます。

1. ユーティリティ関数
2. グラフィックアイ関数
3. サウンド関数
4. モーション関数
5. センサー関数
6. タイマー割込み関数
7. 通信関数

これらの関数の機能・使用方法につきましては別紙の「BN-1 API リファレンス」を参照下さい。

3.3 ROM ライターについて

ROM ライターは、ファームウエアの更新するためのツールです。
ファームウエアは BN-1 に内蔵してある H8 の ROM 領域に書き込まれます。
ROM ライターは trustwrite です。

/usr/local/h8/bin

にインストールされています。

■ 書式

trustwrite -i <Interface> -c <SerialDevice> -s <Speed> MotFile(S2)

【引数】

MotFile(S2) … mot ファイルを指定します

引数なしで trustwrite を実行すると、trustwrite の Version を知ることが出来ます。

【オプション】

-i <Interface> … インターフェースボードを指定します。
通常はこのオプションを指定する必要はありません。

(1)bn1c	C 言語開発キット用インターフェースボード（デフォルト）
(2)bn1b	A.I. TRAINER 付属の赤外線インターフェースユニット

-c <SerialDevice> … シリアルポートを指定します
デフォルトでは以下のシリアルポートを指定します。

(1)Windows	COM1	（シリアルポートが COM1）
(2)Linux	/dev/ttyS0	（シリアルポート COM1（/dev/ttyS0））
(3)FreeBSD	/dev/cuaa0	（シリアルポート COM1（/dev/cuaa0））

-s <Speed> … 通信速度を指定します。
通常はこのオプションを指定する必要はありません。

(1) 4800	4800bps で通信します。
(2) 9600	9600bps で通信します。
(3) 19200	19200bps で通信します。（デフォルト）

-h … ヘルプ（trustwrite の使用方法を表示します。）

■使用方法

○ファームウエア更新方法

（＊）注意！

ファームウエアの更新を行いますと、BN-1 がペットモードで蓄積してきた経験が消去されてしまいます。次回ペットモードで起動した場合には、製品出荷時と同じ状態になりますので御了承下さい。

更新手順は以下のようになります。

（１）インターフェースボードが（図 3.3）のように正しく接続されているか確認してください。

（２）インターフェースボードの切替スイッチを「ROM WRITE」にセットします。

（３）インターフェースボードの電源スイッチを ON にします。
インターフェースボードの電源スイッチの横にある緑色の LED と切替スイッチの横にある赤色の LED が点灯することを確認して下さい。

（４）ROM ライターでファームウエアを転送します。

実行例を以下に示します。

（但し、Windows 版でシリアルポートが COM1 の場合）

```
ex) $ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣-c␣COM1
    [Enter]
```

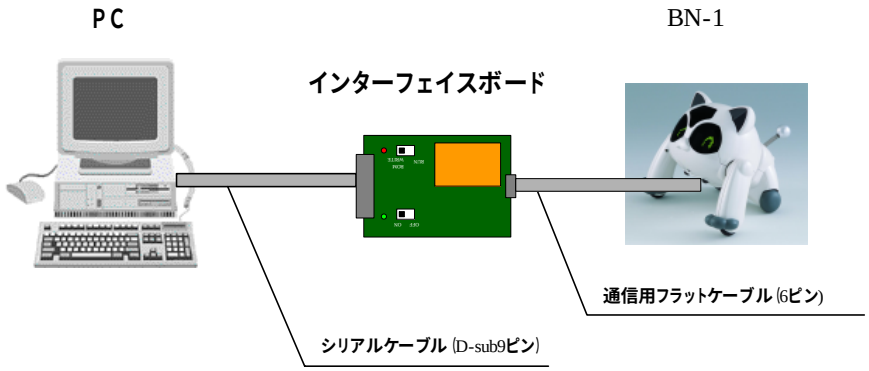


図 3.3　BN-1 と PC の接続

○ROM ライターの Version 確認方法

trustwrite とコマンドを打つだけで ROM ライターの Version を確認できます。
以下に確認方法の例を示します。

```
ex) $ trustwrite [Enter]
    trustwrite version 1.02(C) Trust Technology, Inc.
    Usage: trustwrite -i <Interface> -c <SerialDevice> -s <Speed> MotFile(S2)
           :
           :
```

3.4 H8GCC クロス環境について

○H8GCC クロス環境

H8GCC クロス環境とは、GNU が提供するフリーコンパイラ GCC による H8 用のクロスコンパイラによる開発環境です。
GCC コンパイラは、GCC がその上で使われている CPU と同一タイプの CPU をターゲットとしてコンパイルを行います。
しかし、BN-1 の CPU は H8 ですので、ターゲット CPU を H8 としてコンパイルを行い、コードを生成する必要があります。
このように一つのコンピュータプラットフォームから異なるプラットフォームのコードを生成するコンパイラをクロスコンパイラと呼びます。
BN-1 C 言語開発キットでは H8 用プログラムのバイナリイメージを出力する
GCC クロスコンパイラを用意しました。
クロス環境では x86 環境と同様に、gcc(h8300-hms-gcc)や gdb(h8300-hms-gdb)等の利用が出来ます。

○h8300-hms-gcc

h8300-hms-gcc とは、GNU が提供する H8 用のフリーコンパイラ gcc です。
h8300-hms-gcc を利用することで H8 をターゲットとしたプログラムを生成することが出来ます。

○h8300-hms-gdb

h8300-hms-gdb とは、GNU が提供する H8 用のフリーデバッカ gdb です。
h8300-hms-gdb を利用することでユーザプログラムをソースレベルでデバックすることが可能となります。

○h8300-hms-objcopy

h8300-hms-objcopy とは GNU が提供する H8 用の objcopy です。
objcopy とはオブジェクトファイルのコピーとフォーマット変換を行うツールです。
BN-1 C 言語開発キットのコンパイル時には H8 用プログラムのバイナリイメージを
モトローラ S2 ファイル形式に変換した ASCII テキストファイルで出力します。

■書式

h8300-hms-objcopy［オプション］［入力ファイル］［出力ファイル］

■オプション

－0　srec　　S レコードを生成します。

【用語】

GNU とは…

GNU(GNU is NOT UNIX)は UNIX 互換のソフトウエアシステムで Richard Stallman が創立した FSF(Free Software Foundation)から配布されているフリーソフトウエアです。

URL：http://www.gnu.org

GCC とは…

Gnu Compiler Collection の略
C, C++, FORTLAN などのコンパイラを全て含んだ総称です。

○H8 クロス環境で利用出来るツール一覧

表 3.1　H8 クロス環境で利用出来るツール

h8gcc	gcc	説明
h8300-hms-ar	ar	アーカイブの作成、更新、およびアーカイブからの抽出
h8300-hms-as	as	ポータブルGNUアセンブラ
h8300-hms-c++	c++	C++コンパイラ
h8300-hms-g++	g++	GNUプロジェクトC++コンパイラ
h8300-hms-gasp	gasp	GNUアセンブラプリプロセッサ
h8300-hms-gcc	gcc	GNUプロジェクトCコンパイラ
h8300-hms-gdb	gdb	GNUデバッカ
h8300-hms-ld	ld	GNUリンカ
h8300-hms-nm	nm	オブジェクトファイルからシンボルをリストする
h8300-hms-objcopy	objcopy	オブジェクトファイルのコピーと変換
h8300-hms-objdump	objdump	オブジェクトファイルから情報を表示する
h8300-hms-ranlib	ranlib	アーカイブに対するインデックスの生成
h8300-hms-strings	strings	ファイル内の表示可能な文字列を表示する
h8300-hms-strip	strip	オブジェクトファイルからシンボルの削除

3.5　カプリロモニターについて

カプリロモニターとは BN-1 とプログラム開発者を結びつけるインターフェースです。プログラマーはモニターを使って、プログラムを転送・デバック・実行する事が出来ます。BN-1 C 言語開発キットでは基本的にこれを利用します。

■機能

カプリロモニターには主に以下のような機能があります。

①BN-1 とパソコン間で通信可能

BN-1 とパソコン間でシリアル通信を実現する機能があり、
BN-1 のファームウエア更新が出来ていればパソコン上からカプリロモニターを
起動することで通信が確立されます。

②ユーザプログラムの転送

ユーザプログラムを BN-1 の SRAM 領域に転送します。

③ユーザプログラムの実行

BN-1 に内蔵したプログラムを実行します。

④デバック機能

ユーザプログラムの記述によっては、トレース文でカプリロモニターへの文字列の入力や出力が出来ます。
ブレークポイントを入れてプログラムを BN-1 の動作を一時停止させることが出来、一時停止した状態からプログラムを再開することが出来ます。

⑤スリープ機能

BN-1 をスリープ状態にすることが出来ます。

⑥ウエイクアップ機能

BN-1 をスリープ状態から起動することが出来ます。

⑦!の利用が可能

!!、![数値]、![文字列]といったコマンドヒストリーに残ったコマンドを
利用することが出来ます。

■カプリロモニターの起動

カプリロモニターの起動手順は以下のようになります。

（1）インターフェースボードが正しく接続されているか確認してください。

（2）インターフェースボードのスイッチを「RUN」にセットします。

（3）インターフェースボードの電源スイッチを ON にします。
インターフェースボードの電源スイッチの横にある緑色の LED が点灯し、切替スイッチの横にある赤色の LED が消灯されていることを確認して下さい。

（4）BN-1 の電源を ON にします。
　　BN-1 のグラフィックアイが表示されますので、
　　その状態を確認して下さい。
（5）kterm 等のターミナルから capterm を起動します。
（6）BN-1 のグラフィックアイが表示されれば完了です。

【書式】

capterm -c <SerialDevice>

【オプション】

-c … シリアルポートを指定します。
(1)Windows　COM1　　（シリアルポートが COM1 の場合）
(2)Linux　　/dev/ttyS0　（シリアルポートが COM1（/dev/ttyS0）の場合）
(3)FreeBSD　/dev/cuaa0　（シリアルポートが COM1（/dev/cuaa0）の場合）
-h … ヘルプ（trustwrite の使用方法を表示します。）

【実行例】

以下に実行例を示します。
（シリアルポートが COM1 の場合）

ex) $ capterm␣-c␣COM1 [*Enter*]

モニターと BN-1 間で通信が確立した場合、グラフィックアイが表示され、
パソコンの画面上にモニターの Welcome メッセージが表示されます。

ex) ようこそ BN-1 Monitor へ!
　　BN-1 Monitor:[1]

上記の BN-1 Monitor:[1] は、モニターのプロンプトです。
ここでさまざまなコマンドを入力して BN-1 を操作することができます。

■カプリロモニターの終了

カプリロモニターを終了するには、quit コマンドを使用します。

【実行例】

以下に実行例を示します。

```
ex) BN-1 Monitor:[1] quit [Enter]
    BN-1 Monitor の操作を終了します。
    $
```

■カプリロモニターを強制終了させたい場合

カプリロモニターを強制終了させたい場合には、[CTRL] + [c] を押します。
（[CTRL]キーを押しながら[c]キーを押します）

■文字が消去できない

カプリロモニターで[BS]キーを押しても文字が消去されない場合には[DEL]キーを押してみて下さい。

■カプリロモニターコマンド

表 3.2 カプリロモニターコマンド一覧

コマンド	機能
quit	カプリロモニターを終了します。
load［ファイルパス］	ユーザプログラムをBN-1のSRAM領域に転送します。
run	BN-1のSRAM領域に転送したプログラムを実行します。
setlevel［レベル No］	デバックトレースのレベルNoを設定します。
history	過去のコマンドヒストリーの一覧を表示します。
![文字列]	過去のコマンドヒストリーの中からマッチするコマンドを再実行します。
![数値]	過去のコマンドヒストリーの中から指定した番号のコマンドを再実行します。
!!	直前のコマンドを実行します。
sleep	BN-1をsleep(電源ＯＦＦ)状態にします。
wakeup	BN-1をsleep状態から復帰させます。
version	現在ご使用のカプリロモニターの version を表示します。

(＊) コマンドの使用方法については付録 A の「カプリロモニターのコマンド使用方法について」を参照下さい。

第４章　プログラミングと実行

本章では、ソースプログラムの記述、コンパイル、プログラムの転送・実行を解説します。サンプルプログラムを利用してみたい方は 4.3「プログラムの転送と実行」から読み進めて下さい。

4.1　プログラムの記述

Ｃの記述は ANSI の規格に準拠しています。
プログラムの記述には使い慣れたエディタを使用することが出来ます。
（Windows ならメモ帳等を、Linux または FreeBSD なら vi 等が使用出来ます。）

【注意点】
プログラムを記述する上で注意すべき点を以下に示します。

（１）libbn1api.a をリンクする。
BN-1C 言語開発キットの環境では、BN-1API ライブラリである libbn1api.a をリンクする必要があります。Makefile のテンプレートを利用することでユーザはコンパイル時に libbn1api.a をリンクすることができます。
リンクを行うためのコンパイラへのオプション指定は

```
-L/usr/local/h8/lib/bn1          BN-1 API ライブラリの場所の指定
-lbn1api                         BN-1 API ライブラリのリンク指定
```

となります。(Makefile のテンプレートを参照)

（２）bn1api.h ヘッダファイルをインクルード
BN-1 API を利用するには、bn1api.h をインクルードする必要があります。
プログラムソースを作成する際には必ず以下の例のように bn1api.h をインクルードして下さい。

```
#include <bn1api.h>
```

コンパイラへのオプション指定は

```
-I/usr/local/h8/lib/bn1          BN-1 API ヘッダの場所の指定
```

となります。(Makefile のテンプレートを参照)

（3）ユーザプログラムのサイズ制限

BN-1 のメモリマップは図 4.1 のようになっています。

ユーザプログラムを書き込む事が出来る SRAM 領域は、0x200000 番地から 0x20FFFF 番地までの 64k バイト分となります。これらアドレス空間の定義はリンカスクリプトで指定しています。(/usr/local/h8/lib/bn1/bn1ram.x)

プログラム作成の際には作成した H8 バイナリーイメージが 64k バイト以上の大きさにならないよう注意してください。

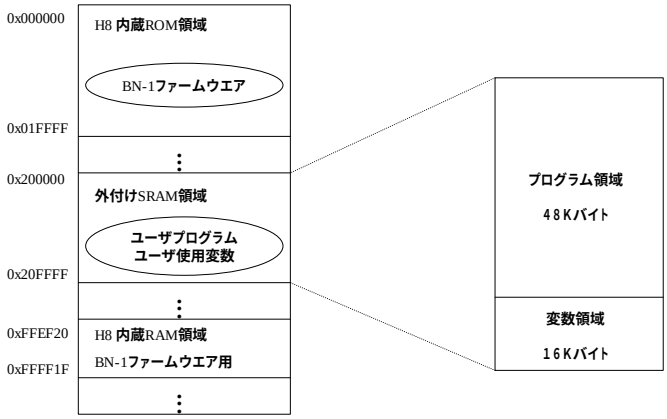


図 4.1　メモリマップ

（3）スタック領域

BN-1 実行環境ではスタック領域（4 バイト）を BN-1 に内蔵の H8RAM 領域に確保しています。H8RAM 領域は 4k バイトとなっており、図 4.2 のように FFFF1C～FFFF1F 番地をスタック領域として確保しています。スタックが増えれば、スタック領域はアドレスの小さい方向へ伸びていきます。ここで注意したいのは、関数を次から次へと呼び出し、スタックがどんどん増え続けた時、スタックが他のデータが使用している記憶領域を壊したりしてしまうことです。このようになりますと、プログラムは暴走してしまいますので、プログラム作成時には注意が必要です。（スタックについての詳細は、付録 D「注意事項」を参照下さい。）

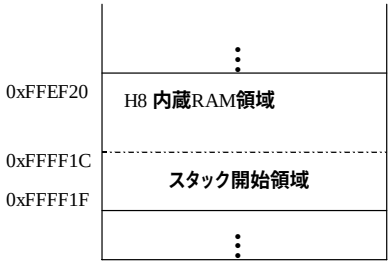


図 4.2　スタック開始領域

（4）h8300-hms-gcc と x86 用 gcc の違いについて

処理系によってコンパイラ仕様が異なりますので、注意する必要があります。

「int 型」と「double 型」のサイズは h8300-hms-gcc と x86 用 gcc とで異なります。また、代入処理を行った時の型変換も h8300-hms-gcc と x86 用 gcc とで異なりますので注意する必要があります。（詳細は、付録 D「注意事項」を参照下さい。）

表 4.1　BN-1 で使用する型とサイズ

型	サイズ（バイト）
char	1
int	2
unsigned int	2
short	2
float	4
long	4
double	4

（5）char 型の負の値について

h8300-hms-gcc では char を 1 バイトとして扱い、上位ビットを切り捨てますので、負の値を扱う場合には注意して下さい。

（詳細は付録 D「注意事項」を参照下さい。）

（6）int と char 変数の参照について

int と char 変数間で値の参照を行ったとき、正しい結果が得られないことがあります。（詳細は付録 D「注意事項」を参照下さい。）

（7）exit 関数の利用不可

main 関数でプログラムを終了させるとき、BN-1 C 言語開発キットでは exit 関数を利用する事が出来ません。プログラムの実行を終了させたい場合は return 文を使用して下さい。

4.2　コンパイル方法説明

H8 用に用意された GCC クロス環境でコンパイルします。
コンパイラには H8 をターゲットとしたクロスコンパイラ h8300-hms-gcc を利用します。クロスコンパイラで生成したバイナリーファイルを MOT ファイルに変換するには H8 をターゲットとしたファイル変換ツール h8300-hms-objcopy を利用します。

＊MOT ファイル･･･ H8 用プログラムのバイナリイメージをモトローラ S2 ファイル形に変換した ASCII テキストファイルです。

コンパイル手順は以下のとおりです。
①個々のソースファイルをオブジェクトファイルにします。
②オブジェクトファイル、スタートアップルーチン、リンカスクリプトから
バイナリーファイルを生成します。
③ファイル形式を変換するツールでバイナリーファイルから S2 フォーマット形式に変換した MOT ファイルを生成します。

コンパイルのイメージは以下のようになります。

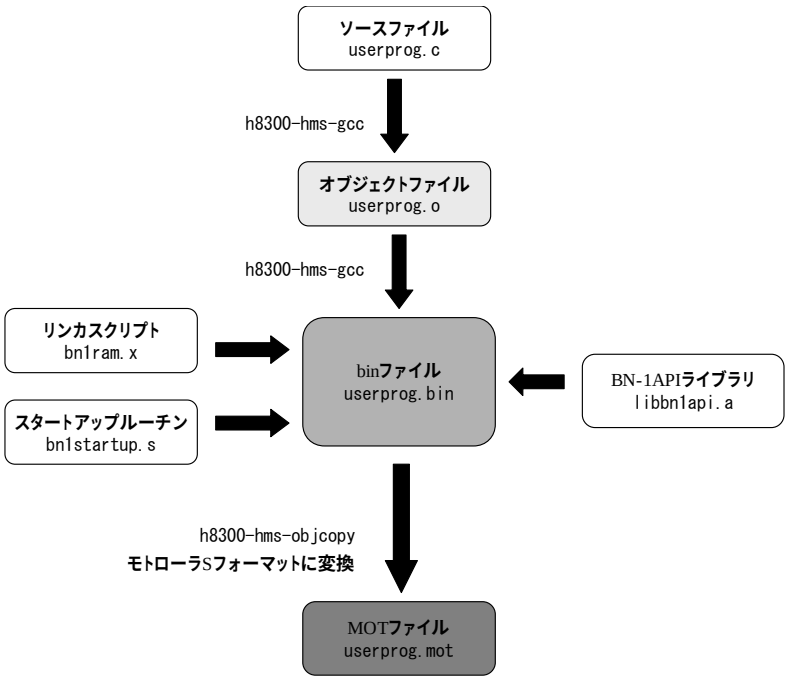


図 4.3　コンパイルイメージ

図 4.3 のようなコンパイル作業を簡単に行えるツールとして make コマンドがあります。
make を使ってコンパイルするには Makefile にコンパイルの命令を記述しなければなりませんが、BN-1 C 言語開発キットでは Makefile のテンプレートを用意しましたので、内容を一部書き換えるだけでコンパイル出来るようにしてあります。

Makefile のテンプレート（ファイル名：Makefile）は
　　/usr/local/h8/share/template
にあります。

【make とは】
make は UNIX のシェルが実行するコマンド列を生成するものです。一般的にソフトウエア開発の一連のファイルをメンテナンスするのに用いられます。
make とコマンドすると、make は Makefile もしくは makefile というファイルを参照します。コンパイルに必要な複数回に渡るコマンド命令を Makefile に記述することで make コマンド１回でコンパイルすることが出来ます。
ソースファイルもしくはオブジェクトファイルが最終的に生成する実行ファイルのタイムスタンプより新しくなっていた場合は、更新されたファイルのみをコンパイルし、再リンクすることで実行ファイルを生成します。

（注！）FreeBSD 版では make は gmake を使用して下さい。
　　gmake は FreeBSD のパッケージからインストールして下さい。

Makefile のテンプレートを修正し、コンパイルする方法を説明します。
コンパイル時には、複数のオブジェクトファイル、スタートアップルーチン、
リンカスクリプトをリンクして、バイナリーファイルを生成します。
以下はバイナリーファイルを生成する為に必要なファイルになります。

（１）bn1startup.s　・・・　スタートアップルーチン
　　（/usr/local/h8/lib/bn1 にあります。）
（２）bn1ram.x　　・・・　リンカスクリプト
　　（/usr/local/h8/lib/bn1 にあります。）
（３）libbn1api.a　・・・　BN-1 API ライブラリ
　　（/usr/local/h8/lib/bn1 にあります。）
（４）bn1api.h　　・・・　BN-1 API を使用する際にインクルードするヘッダファイル
　　（/usr/local/h8/include/bn1 にあります。）

（５）Makefile　　・・・　Makefile のテンプレートを修正したもの
（/usr/local/h8/share/template にあるものを修正し、
ユーザのワークディレクトリに保存します。）
（６）userprog.c　　・・・　ユーザが作成した C ソースプログラム
（作成したプログラムは、ユーザのワークディレクトリに
保存します。）

ユーザは（５）と（６）のファイルを同じディレクトリに置いて下さい。

【Makefile の修正】

Makefile の書き換えは TARGET 以下の記述を変更するだけです。
make を行うと、TARGET 以下に記されたファイルを生成します。
以下に Makefile 書き換えの一例を示します。
(userprog.c をコンパイルして userprog.mot ファイルを生成したい場合)

（Makefile テンプレート）

```
TARGET = **** <- Specify target mot file
```

を

```
TARGET = userprog.mot
```

に書き換えます。

書き換えたファイルは「Makefile」という名前のまま上書き保存します。

【Makefile の補足】

（１）TARGET

TARGET に複数の mot ファイルを記述して途中で改行したい場合には
行末に「¥」（バックスラッシュ）を記述します。

```
ex) TARGET = testeye.mot testsound.mot ¥
             userprog.mot
```

（2）h8300-hms-gcc オプション

Makefile のテンプレートには以下のようなオプション記述があります。
これらは h8300-hms-gcc のオプションで次のような意味があります。

-c file

プログラムソースファイルを file に指定します。ソースファイルのコンパイル、もしくはアセンブルを行いますが、リンクは行いません。

-o file

出力先を file に指定します。このオプションは GCC が実行可能ファイル、オブジェクトファイル、アセンブラファイル、プリプロセス済み C コードなどの、いかなる種類の出力を行なう場合にも適用可能です。-o の指定がない場合は、実行形式ファイルは a.out という名前で出力されます。

-I dir

プログラムヘッダファイルのサーチパスを dir に指定します。

-L dir

ライブラリのサーチパスを dir に指定します。

-l library

リンクするライブラリを library に指定します。ここで指定する library は、ライブラリファイル libXXX.a の先頭文字 lib および拡張子を削除した名称です。BN-1 API では libbn1api.a をリンクしますので、-l bn1api と指定します。

-nostartfiles

デフォルトのスタートアップルーチンを使用しない指定です。Makefile のテンプレートでは-nostartfiles の後に、/usr/local/h8/lib/bn1 に配置された bn1startup.s を指定しています。本システムではこのようにして、デフォルトのスタートアップルーチンの代わりに BN-1 API 専用スタートアップルーチン bn1startup.s を使用します。

-T lscript

メモリマップを決定付けるリンカスクリプトファイルを lscript に指定します。
Makefile のテンプレートでは、/usr/local/h8/lib/bn1 に配置された bn1ram.x をリンカスクリプトとして指定しています。

「H8/300 オプション」

H8/300 用のオプションとして以下のようなものがあります。

-mh

H8/300H 用のコードを生成します。
BN-1 C 言語開発キットではこのオプションは必須です。

-mrelax

可能であれば、リンク時にいくつかのアドレス参照を短くします。

-mint32

int 型のデータをデフォルトで 32 ビットにします。
但し、BN-1 C 言語開発キットを使用の際はこのオプションを指定しないで下さい。（BN-1 C 言語開発キットでは int 型のサイズを 16 ビットと想定している為）

-malign-300

H8/300H 上で、 H8/300 と同一の境界整列規則を使います。
H8/300H のデフォルトでは、 long 型と float 型は 4 バイト境界に境界整列されます。'-malign-300'を指定すると、これらは 2 バイト境界に境界整列されます。
但し、BN-1 C 言語開発キットを使用の際はこのオプションを指定しないで下さい。（BN-1 C 言語開発キットでは long 型、float 型のサイズを 4 バイトと想定している為）

【コンパイルについて】

（1）コンパイルするには

make とコマンドを入力します。

ex）$ make [*Enter*]

（2）再コンパイルするには

make　clean とコマンドを入力して、TARGET 以下に記述した mot ファイルとオブジェクトファイルをすべて削除してから、make コマンドでコンパイルします。

ex）$ make␣clean [*Enter*]
　　$ make [*Enter*]

4.3 プログラムの転送と実行

プログラムの転送と実行にはカプリロモニターを使用します。
プログラムの転送には load コマンドを使用し、実行には run コマンドを使用します。

○プログラムの転送

■コマンド名称　load

■機能　引数で指定したプログラム（MOT ファイル）を BN-1 の SRAM 領域に転送します。

■書式　load [ファイルパス]

■使用例

ex) BN-1 Monitor:[1] load␣../misc/testsound.mot [*Enter*]
..(転送中).....
転送が終了しました。

○プログラムの実行

■コマンド名称　run

■機能　BN-1 の SRAM 領域に転送されたプログラム（MOT ファイル）を実行します。

■書式　run

■使用例

ex) BN-1 Monitor:[2] run [*Enter*]

第５章　Ｈ８ＧＣＣの利用

5.1 h8300-hms-gcc

Ｃの記述は ANSI の規格に準拠しています。
h8300-hms-gcc とは、GNU が提供する H8 用のフリーコンパイラ gcc です。
h8300-hms-gcc を利用することで H8 をターゲットとしたプログラムを生成することが出来ます。
h8300-hms-gcc は、オプションとファイル名を引数として受け付けることが出来ます。

【書式】

h8300-hms-gcc［**オプション**］［**ファイル名**］．．．

コンパイル過程は、四つの段階に分けることができます。
プリプロセス、コンパイル、アセンブル、リンクであり、必ずこの順番で行われます。
前の三つの段階は個々のソースファイルに対して適用され、オブジェクトファイルの生成で終わります。
リンクでは、全てのオブジェクトファイル(新しくコンパイルされ、入力として指定されたもの)を組み合わせて、一つの実行形式ファイルを作ります。

ソースファイル名の拡張子は、その言語が何であるかと、どのような処理が行われるべきかを示します。

.c　C 言語ソースです。プリプロセッサ、コンパイラ、アセンブラにかけられます。
.C　C++言語ソースです。プリプロセッサ、コンパイラ、アセンブラにかけられます。
.cc　C++言語ソースです。プリプロセッサ、コンパイラ、アセンブラにかけられます。
.cxx　C++言語ソースです。プリプロセッサ、コンパイラ、アセンブラにかけられます。
.m　Objective-C 言語ソースです。プリプロセッサ、コンパイラ、アセンブラにかけられます。
.i　プリプロセッサにかけられた C 言語ソースです。コンパイラ、アセンブラにかけられます。
.ii　プリプロセッサにかけられた C++言語ソースです。コンパイラ、アセンブラにかけられます。
.s　アセンブリ言語ソースです。アセンブラにかけられます。
.S　アセンブリ言語ソースです。プリプロセッサ、アセンブラにかけられます。
.h　プリプロセッサファイルです。

【オプション】

-c file

プログラムソースファイルを file に指定します。ソースファイルのコンパイル、もしくはアセンブルを行いますが、リンクは行いません。

-o file

出力先を file に指定します。このオプションは GCC が実行可能ファイル、オブジェクトファイル、アセンブラファイル、プリプロセス済み C コードなどの、いかなる種類の出力を行なう場合にも適用可能です。-o の指定がない場合は、実行形式ファイルは a.out という名前で出力されます。

-I dir

プログラムヘッダファイルのサーチパスを dir に指定します。

-L dir

ライブラリのサーチパスを dir に指定します。

-l library

リンクするライブラリを library に指定します。ここで指定する library は、ライブラリファイル libXXX.a の先頭文字 lib および拡張子を削除した名称です。BN-1 API では libbn1api.a をリンクしますので、-l bn1api と指定します。

-nostartfiles

デフォルトのスタートアップルーチンを使用しない指定です。Makefile のテンプレートでは-nostartfiles の後に、/usr/local/h8/lib/bn1 に配置された bn1startup.s を指定しています。本システムではこのようにして、デフォルトのスタートアップルーチンの代わりに BN-1 API 専用スタートアップルーチン bn1startup.s を使用します。

-T lscript

メモリマップを決定付けるリンカスクリプトファイルを lscript に指定します。Makefile のテンプレートでは、/usr/local/h8/lib/bn1 に配置された bn1ram.x をリンカスクリプトとして指定しています。

-g

GDB が取り扱えるデバック情報を生成します。

【H8/300 オプション】

H8/300 用のオプションとして以下のようなものがあります。

-mh

H8/300H 用のコードを生成します。

BN-1 C 言語開発キットではこのオプションは必須です。

-mrelax

可能であれば、リンク時にいくつかのアドレス参照を短くします。

-mint32

int 型のデータをデフォルトで 32 ビットにします。

但し、BN-1 C 言語開発キットを使用の際はこのオプションを指定しないで下さい。

（BN-1 C 言語開発キットでは int 型のサイズを 16 ビットと想定している為）

-malign-300

H8/300H 上で、 H8/300 と同一の境界整列規則を使います。

H8/300H のデフォルトでは、 long 型と float 型は 4 バイト境界に境界整列されます。`-malign-300'を指定すると、これらは 2 バイト境界に境界整列されます。

但し、BN-1 C 言語開発キットを使用の際はこのオプションを指定しないで下さい。

（BN-1 C 言語開発キットでは long 型、float 型のサイズを 4 バイトと想定している為）

【データ型とサイズ】

h8300-hms-gcc と x86 用 gcc のデータ型とサイズを以下に示します。

表 5.1　h8300-hms-gcc (BN-1)

型	サイズ (バイト)
char	1
int	2
unsigned int	2
short	2
float	4
long	4
double	4

表 5.2　gcc (x86 用)

型	サイズ (バイト)
char	1
int	4
unsigned int	4
short	2
float	4
long	4
double	8

5.2 h8300-hms-gdb

GDB デバッガは、プログラムが実行中もしくはクラッシュした時にそのプログラムの'内部'で何が行なわれているか/行われていたかを調べるのに使用されます。

GDB は、4 つの機能 (加えてこれらをサポートする機能) によって実行中にバグを見つけることを手助けします。

○プ ログラムの動作を詳細に指定してプログラムを実行させる。

○指定した条件でプログラムを停止させる。

○プログラムが止まった時に、何が起こったか調べる。

○バグによる副作用を修正し、別のバグを調べるためプログラムの状態を変更する。

GDB を利用するにはコンパイル時に -g オプションをつけてコンパイルします。

【コンパイル例】

```
$ h8300-hms-gcc␣-o␣hello␣-g␣-mh␣hello.c [Enter]
```

（＊）-mh は H8/300H 用のコードを生成するオプションです。

シミュレータデバッグ時の手順は以下のようになります。

① h8300-hms-gdb を起動します。
② set machine h8300h で H8/300H である事を宣言します。
③ target sim で実行ターゲットとしてシミュレータに接続します。
④ load でターゲットにプログラムをロードします。
⑤ run でプログラムを実行します。

【GDB 起動と終了】

GDB を起動する。

■書式　　h8300-hms-gdb［実行形式ファイル］

gdb を終了する。

■書式　　quit

【GDB の基本コマンド一部紹介】

（１）set machine h8300h

■機能　　ターゲット CPU を H8/300H にします。

■書式　　set␣machine␣h8300h

（２）target sim

■機能　　ユーザプログラムをデバッグする際にハードウェア CPU の代わりに使うことのできる CPU シミュレータを利用します。

■書式　　target␣sim

（３）load

■機能　　実行ファイルをロードします。

■書式　　load

（４）run

■機能　　ロードしたプログラムを実行します。

■書式　　run

【実行例】

以下に h8300-hms-gdb を利用したときの実行例を示します。
テスト用プログラム hello.c を作成することにします。

ex)

```
#include <stdio.h>

int main(void)
{
        printf("Hello,World!¥n");

        return(0);
}
```

デバックが出来るよう、-g オプションを付けてコンパイルします。

ex)

```
$ h8300-hms-gcc␣-o␣hello␣-g␣-mh␣hello.c [Enter]
```

デバッガ(h8300-hms-gdb)を動かします。(Cygwin で起動した場合)

ex)

```
$ h8300-hms-gdb.exe␣hello.exe [Enter]
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=h8300-hms"...
(gdb) set␣machine  h8300h [Enter]
(gdb) target␣sim [Enter]
Connected to the simulator.
(gdb) load [Enter]
Loading section .text, size 0x7f4c vma 0x100
Loading section .data, size 0x8aa vma 0x804c
Loading section .stack, size 0x4 vma 0x3fffc
Start address 0x420
Transfer rate: 278480 bits in <1 sec.
(gdb) run [Enter]
Starting program: /home/Administrator/test/ctest/hello.exe
Hello,World!

Program received signal SIGTRAP, Trace/breakpoint trap.
0x7d6 in _exit (rc=0) at _exit.c:14
14      _exit.c: No such file or directory.
(gdb) quit [Enter]
The program is running.  Exit anyway? (y or n) y [Enter]
```

第６章　チュートリアル

それでは、「BN-1C 言語開発キット」によるプログラミングを始めましょう。
簡単なプログラムを例に作業手順を解説していきます。

○本チュートリアルの表記法

$　UNIX のシェルプロンプト

~　ホームディレクトリ

/　UNIX 表記のディレクトリ区切り

¥　Windows 表記のディレクトリ区切り

␣　スペース（空白）

［*Enter*］　Enter キーを押します

6.1 手順

おおまかな作業手順は以下の通りです。

１．準備
２．パスの設定
３．ファームウエアの更新
４．カプリロモニター起動
５．カプリロモニターでサンプルログラムを BN-1 へ転送
６．カプリロモニターからサンプルプログラムを実行
７．エディタでプログラムを記述
８．Makefile の書き換え
９．作成したプログラムをコンパイルする
１０．カプリロモニターからプログラムを実行
１１．プログラムを改良する

6.2 準備

6.2.1　起動

H8GCC クロス環境のインストールは終了していますか？
インストールされていない場合は本マニュアルの第２章「準備」を参照し、
H8GCC クロス環境の構築を完了させてください。(Windows 版は Cygwin も含みます)

パソコンの電源をオフにした状態で、付属のシリアルケーブルの一方の端を「インターフェースボードのシリアル端子」に差し込み、もう一方の端を現在ご使用のパソコンの「シリアルコネクタ端子」に差し込んでください。

また、「インターフェースボード」に接続されているフラットケーブル(６ピン)の一方の端を BN-1 本体の下面（腹部）に接続してください。（本体の下面にはメンテナンスハッチがありますので、マイナスドライバーのような先の平たいもので外してください。）

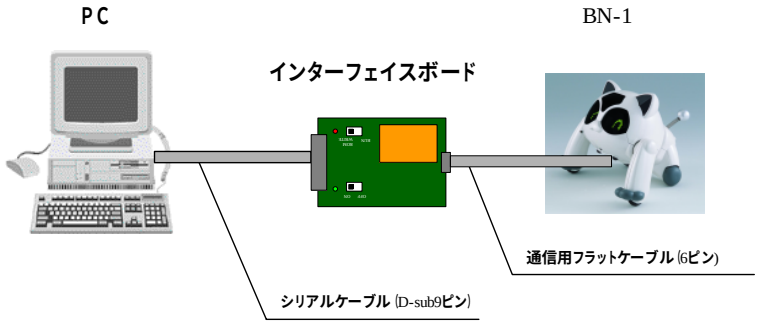


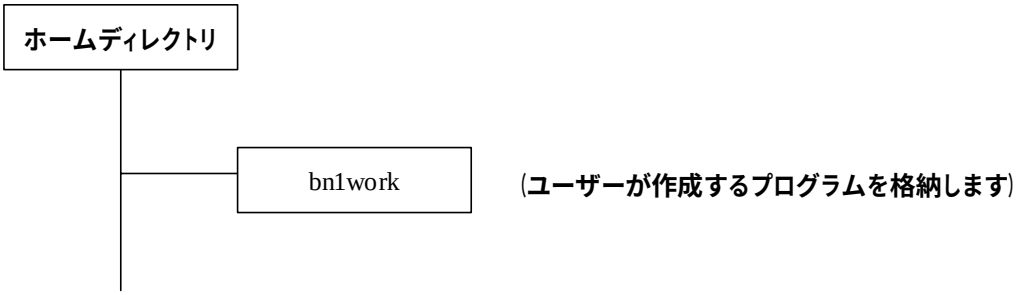
図 6.1　BN-1 と PC の接続

① Windows の場合
パソコンを立ち上げましたら、インストールした Cygwin を立ち上げてください。
Cygwin を起動するには「スタートボタン」→「プログラム」→「Cygnus Solutions」→「Cygwin Bash Shell」
を選択するか、
デスクトップ上に作成された「Cygwin」アイコンをダブルクリックして起動します。（2.1.4「インストール手順」に従ってインストールされた場合）

② Linux、FreeBSD の場合
使い慣れたターミナル（kterm 等）を立ち上げてください。

6.2.2　ディレクトリの作成

これから始めるプログラミングを進めていく上で必要な作業ディレクトリを作成します。本チュートリアルでは下記のようなディレクトリ構成であることを想定し進めていきますので、使用環境に合わせて読み替えて下さい。



◆知っていると便利◆
「ホームディレクトリ」
ホームディレクトリとは、ユーザ毎に用意された作業ディレクトリのことです。
UNIX（Linux / FreeBSD）マシンの場合には、ログインしたときの標準のカレントディレクトリがホームディレクトリとなります。
Cygwin の場合には、bash 起動時のカレントディレクトリであり、
c:¥cygwin¥home¥ログイン名（C ドライブに Cygwin をインストールした場合）が相当します。ホームディレクトリへの移動は、 cd␣~ [*Enter*]　で移動できます。

ホームディレクトリ上に適当なワークディレクトリ「bn1work」を作成します。

```
ex) $ cd␣~ [Enter]
    $ mkdir␣bn1work [Enter]
```

「bn1work」ディレクトリが作成されました。
ディレクトリが作成されたかどうかは ls コマンドで確認する事が出来ます。

```
ex) $ ls [Enter]
    bn1work
```

6.2.3　必要なファイルの確認とファイルのコピー

本チュートリアルを進めていく上で必要となるファイルがあるか確認してみましょう。

○ファームウエア

capmon-v100.mot ・・・

BN-1C 言語開発キットを利用する為のファームウエアです。
ファームウエアは /usr/local/h8/share/firmware にインストールされています。ls コマンドで capmon-v100.mot があるか確認しましょう。

ex) $ ls␣/usr/local/h8/share/firmware [*Enter*]

bn1org-v03.mot ・・・

BN-1 を初期状態に戻す為のファームウエアです。
ファームウエアは /usr/local/h8/share/firmware にインストールされています。ls コマンドで bn1org-v03.mot があるか確認しましょう。

ex) $ ls␣/usr/local/h8/share/firmware [*Enter*]

○ROM ライター

trustwrite ・・・

ROM ライターです。(Windows 版は trustwrite.exe)
カプリロモニターは /usr/local/h8/bin にインストールされています。
ls コマンドで capterm があるか確認しましょう。

ex) $ ls␣/usr/local/h8/bin [*Enter*]

○カプリロモニター

capterm ・・・

カプリロモニターです。(Windows 版は capterm.exe)
カプリロモニターは /usr/local/h8/bin にインストールされています。
ls コマンドで capterm があるか確認しましょう。

ex) $ ls␣/usr/local/h8/bin [*Enter*]

○サンプルプログラム

BN-1 C 言語開発キットではサンプルプログラムを用意しました。
サンプルプログラムは /usr/local/h8/share/samples/misc
にインストールされています。
ls コマンドでサンプルプログラムがあるか確認しましょう。

ex) $ ls␣/usr/local/h8/share/samples/misc [*Enter*]

○ .bashrc（Windows 版のみ）

Cygwin 上で使われるシェルはデフォルトでは bash に設定されていますが、
インストール直後のホームディレクトリには「.bashrc」がありません。
初めて Cygwin を使用する方は /usr/local/h8/share/template にインストールされた
「bashrc.default」をホームディレクトリにコピーします。
コピーの際はコピー先のファイル名を「.bashrc」に変更しますので注意して下さい。
コピー例を以下に示します。

ex) $ cp␣/usr/local/h8/share/template/bashrcdefault␣~/.bashrc [*Enter*]

コピーが終わりましたら、ls -a コマンドで「.bashrc」がホームディレクトリにある
ことを確認してから、source コマンドで「.bashrc」を有効にします。

ex) $ ls␣-a␣~ [*Enter*]
$ source~/.bashrc [*Enter*]

○ Makefile

「Makefile」のテンプレートです。
このファイルの内容を一部修正することで作成したプログラムを簡単にコンパイルする
事が出来ます。
「Makefile」は /usr/local/h8/share/template ディレクトリの下にあります。
「Makefile」を(6.2.2)で作成した「bn1work」ディレクトリの下にコピーして下さい。

ex) $ cp␣/usr/local/h8/share/template/Makefile␣~/bn1work [*Enter*]

6.3 パスの設定

H8GCC クロス環境へのパスを追加します。

① Windows の場合

(6.2) の「準備」で「.bashrc」をコピーされた場合は設定する必要はありません。
(6.4)「ファームウエアの更新」へ進んでください。

（1）Shell が bash の場合

お使いのエディタで「.bashrc」を開いてください

ex) $ vi␣~/.bashrc [*Enter*]

ホームディレクトリにある「.bashrc」に以下の記述を追加します。
（但し、下記のパスは H8 クロス環境のインストール先をデフォルトで行った場合です）

```
if [ -d /usr/local/h8/bin ]; then
        PATH=$PATH:/usr/local/h8/bin
fi
```

ここで行った修正は、次回のログインから有効になりますが、
source コマンドを使用することでログアウトせずに有効にする事が出来ます。

ex) $ source␣~/.bashrc [*Enter*]

② Linux、FreeBSD の場合

（1）Shell が bash の場合

お使いのエディタで「.bashrc」を開いてください

ex) $ vi␣~/.bashrc [*Enter*]

ホームディレクトリにある「.bashrc」に以下の記述を追加します。
（但し、下記のパスは H8 クロス環境のインストール先をデフォルトで行った場合です）

```
if [ -d /usr/local/h8/bin ]; then
```

```
        PATH=$PATH:/usr/local/h8/bin
fi
```

ここで行った修正は、次回のログインから有効になりますが、
source コマンドを使用することでログアウトせずに有効にする事が出来ます。

ex) $ source␣~/.bashrc [*Enter*]

（2）Shell が csh もしくは tcsh の場合
お使いのエディタで「.cshrc」または「.tcshrc」を開いてください

ex) $ vi␣~/.cshrc [*Enter*]

もしくは

$ vi␣~/.tcshrc [*Enter*]

ホームディレクトリにある「.cshrc」または「.tcshrc」に以下の記述を追加します。(但し、下記のパスは H8 クロス環境のインストール先をデフォルトで行った場合です)

```
if ( -d /usr/local/h8/bin ) then
        setenv PATH "/usr/local/h8/bin:"$PATH
endif
```

ここで行った修正は、次回のログインから有効になりますが、
source コマンドを使用することでログアウトせずに有効にする事が出来ます。

ex) $ source␣~/.cshrc [*Enter*]

もしくは

$ source␣~/.tcshrc [*Enter*]

6.4　ファームウエアの更新

ファームウエアの更新を行います。
更新を行う前にパソコン、インターフェースボード、BN-1 本体がケーブルで
つながれているかもう一度確認してみて下さい。

①パソコンの「シリアルコネクタ端子」と「インターフェースボードのシリアル端子」
　はシリアルケーブルでつながれていますか？
②「インターフェースボードのフラットケーブル(6 ピン)」と「BN-1 本体下面（腹部）
　の 6 ピン端子」はつながれていますか？

確認が終わりましたら、インターフェースボードを見てみましょう。
インターフェースボードにモード切替スイッチがあります。
このスイッチを「ROM WRITE」側にセットします。
「ROM WRITE」側にセットしましたら、ボードの電源スイッチを ON にして下さい。
この時、インターフェースボードの電源スイッチの横にある緑色の LED と切替スイッチの横にある赤色の LED が点灯することを確認して下さい。

ファームウエアを更新するにはパソコン側で ROM ライターソフトを起動します。
ROM ライターソフトのインストールは H8GCC クロス環境のインストール時に
行われており、「trustwrite」という ROM ライターソフトが
/usr/local/h8/bin の下にインストールされています。

ファームウエア capmon-v100.mot は /usr/local/h8/share/firmware に
インストールされていますので、
trustwrite の引数に /usr/local/h8/share/firmware/capmon-v100.mot
と指定することで、ファームウエアの更新（BN-1 の ROM 領域への書込み）が行われます。

ライターの実行は以下のようになります。

① Windows (Cygwin)
（シリアルポートが COM1 の場合）

ex) $ trustwrite ␣/usr/local/h8/share/firmware/capmon-v100.mot␣-c␣COM1
[*Enter*]

② Linux
（シリアルポートが COM1（/dev/ttyS0）の場合）

ex) $ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣-c␣/dev/ttyS0
[*Enter*]

③ FreeBSD
（シリアルポートが COM1（/dev/cuaa0）の場合）

ex) $ trustwrite␣/usr/local/h8/share/firmware/capmon-v100.mot␣-c␣/dev/cuaa0
[*Enter*]

◆知っていると便利◆
「ファームウエアの更新が出来ない？」
今まで問題なく行っていたファームウエアの更新が急に出来なくなった場合、
原因として考えられるのは、インターフェースボードのバッテリー切れです。
このような時はバッテリーを新品に交換してみて下さい。

ライターを起動した時の例を示します。
（Windows 版で、シリアルポートが COM1 の場合）

ex)

```
$ trustwrite /usr/local/h8/share/firmware/capmon-v100␣-c␣COM1 [Enter]
trustwrite version 1.02 TrustTechnology, Inc.
  interface type:     BN-1 C language development kit
  serial device file: COM1
  serial speed:       19200
H8/3067F(BN-1/AKI) boot mode ready!
Transfer H8 ROM Writer
  writing...   0 %
  writing...  10 %
  writing...  20 %
          :
          :
```

ROM ライターが無事に実行された場合、書き込む ROM イメージの
ダンプが下記のように表示され、最後に書き込み完了の表示がされます。

ex)　　:

```
[14560] 96 00 00 81 ff ff ff ff ff ff ff ff ff ff ff ff
[14570] ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
H8 EEPROM writing complete.
```

書き込みが完了すると、trustwrite から書き込み完了メッセージが出力されます。
ここで、インターフェースボード上のスイッチを「RUN」に切り替えますと、BN-1
のグラフィックアイが表示されますので、確認してください。
この状態で、パソコンからのシリアル通信待ちとなります。

◆知っていると便利◆

（1）「ROM ライターの Version 確認」

インストール済の ROM ライターを起動してバージョンを確認してみましょう。
パスが効いている環境では trustwrite とコマンドを打つだけで確認できます。

```
ex) $ trustwrite [Enter]
    trustwrite version 1.02 (C) Trust Technology, Inc.
    Usage: trustwrite -i <Interface> -c <SerialDevice> -s <Speed> MotFile(S2)
             :
```

（2）「ROM ライターの引数とオプション」

ROM ライターの引数とオプションは次のようになっています。

```
trustwrite -i <Interface> -c <SerialDevice> -s <Speed> MotFile(S2)
```

【引数】

MotFile(S2)… mot ファイルを指定します

【オプション】

-i <Interface> … インターフェースボードを指定します。
通常はこのオプションを指定する必要はありません。

-c <SerialDevice> … シリアルポートを指定します。

(1)Windows	COM1	（シリアルポートが COM1 の場合）
(2)Linux	/dev/ttyS0	（シリアルポートが COM1（/dev/ttyS0）の場合）
(3)FreeBSD	/dev/cuaa0	（シリアルポートが COM1（/dev/cuaa0）の場合）

-s … 通信速度を指定します

(1) 4800	4800bps で通信します。
(2) 9600	9600bps で通信します。
(3) 19200	19200bps で通信します。（デフォルト）

-h … ヘルプ（trustwrite の使用方法を表示します。）

（3）「mot ファイル」

H8 用プログラムのバイナリイメージをモトローラ S2 形式に
変換した ASCII テキストファイルです。

（4）「BN-1 を元の状態に戻す」

BN-1 を元の状態に戻すには bn1org-v03.mot を BN-1 に書き込みます。
書き込みが終わりましたら、メモリクリアを行います。

○bn1org-v03.mot

BN-1 を初期状態に戻す為のファームウエアです。

○書き込み

BN-1 C 言語開発キット用のファームウエアを更新したときと同じ手順になります。
インターフェースボードのモード切替スイッチを「ROM WRITE」にセットし、
以下のようにライターを実行します。
（Windows 版でシリアルポートが COM の場合）

ex) $ trustwrite␣/usr/local/h8/share/firmware/bn1org-v03.mot␣-c␣COM1[*Enter*]

○メモリクリア

BN-1 本体背中に POWER ボタンがあります。
①POWER ボタンを指で押したままの状態にします。
②ツマヨウジ（または金属製ではない細長い棒状のもの）でリセットスイッチを、
奥まで（カチリという手応えがあるまで）押します。
③POWER ボタンを押したままリセットボタンからツマヨウジを離します。
④さらに POWER ボタンをそのまま約 5 秒間押し続けると、
BN-1 本体から「チーン」という音が鳴ります。
⑤POWER ボタンを離し、操作完了です。

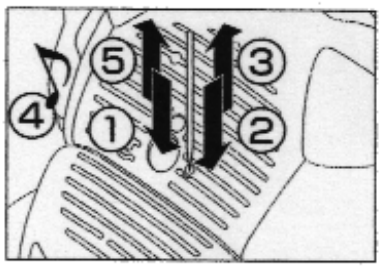


図 6.2　メモリクリア

6.5 カプリロモニター起動

インターフェースボートのモード切替スイッチが「RUN」にセットされて、
電源スイッチがＯＮになっていることを確認して下さい。

◆知っていると便利◆
「モード切替を RUN にする」
インターフェースボートのモード切替スイッチを「RUN」にセットし、
インターフェースボードの電源スイッチを ON にします。
この時、インターフェースボードの電源スイッチの横にある緑色の LED が点灯し、
切替スイッチの横にある赤色の LED が消灯されていることを確認出来ます。

BN-1 本体の電源を ON にして下さい。
BN-1 のグラフィックアイが表示されていますので、その状態を確認して下さい。

BN-1 を実行するにはカプリロモニター(capterm)を使用します。
ターミナル（Windows は Cygwin）上からカプリロモニターを起動して、
BN-1 と通信を確立させます。

①Windows の場合

```
ex) $ capterm␣-c␣COM1 [Enter]
```

（但し、シリアルポートが COM1 の場合）

②Linux の場合

```
ex) $ capterm␣-c␣/dev/ttyS0 [Enter]
```

（但し、シリアルポートが COM1（/dev/ttyS0）の場合）

③FreeBSD の場合

```
ex) $ capterm␣-c␣/dev/cuaa0 [Enter]
```

（但し、シリアルポートが COM1（/dev/cuaa0）の場合）

モニターと BN-1 間で通信が確立した場合、パソコンの画面上のモニターに 「ようこそ」という Welcome メッセージが表示されます。

ex) ようこそ BN-1 Monitor へ!
BN-1 Monitor:[1]

上記の BN-1 Monitor:[1] は、モニターのプロンプトです。
ここでさまざまなコマンドを入力して BN-1 を操作することができます。
コマンドの使用方法については付録 A「カプリロモニターのコマンド」を参照下さい。
もし、カプリロモニターが上手く起動しなかった場合には、[CTRL] + [c] で切断できますので、接続機器の確認とトラブルシューティングを参照して下さい。

6.6 カプリロモニターでサンプルログラムを BN-1 へ転送

カプリロモニターの load コマンドを使用してプログラムを転送します。
例として、BN-1 に組み込まれているサウンドを鳴らすサンプルプログラム testsound.mot を転送してみましょう。

ex) ようこそ BN-1 Monitor へ!
BN-1 Monitor:[1] load␣/usr/local/h8/share/samples/misc/testsound.mot [*Enter*]
..(転送中).....
転送が終了しました。
BN-1 Monitor:[2]

と表示されます。

6.7 カプリロモニターからサンプルプログラムを実行

転送したサンプルプログラムを実行するには、
カプリロモニターの run コマンドを使用してプログラムを実行します。

ex) BN-1 Monitor:[2] run [*Enter*]

BN-1 の目には「GO」と表示されています。
また、プログラム中に取り込んだデバックトレース関数により、
モニターには以下のような表示がされます。

[TraceOut] Test Sound Start!

[TraceOut] Set Sound Volume 22
[Break] - <Continue: Hit Enter Key> -
ここで、[*Enter*]を押すと BN-1 からサウンドが聞こえてきます。
BN-1 の目には「0」と表示されています。
モニターには以下のように表示されます。

[Break] - <Continue: Hit Enter Key> -

続けて、[*Enter*]を押すと BN-1 から別のサウンドが聞こえてきます。
BN-1 の目には「1」と表示されています。
モニターには以下のように表示されます。

[Break] - <Continue: Hit Enter Key> -

同様に最後の曲になるまで、[*Enter*]を押すことができます。

最後の

[Break] - <Continue: Hit Enter Key> -

で[*Enter*]を押すと、BN-1 の目には「EN」と表示されます。
モニターには以下のように表示され、testsound プログラムが終了します。

[TraceOut] Test Sound End!

カプリロモニターを終了させましょう。

ex) BN-1 Monitor:[2] quit [*Enter*]

BN-1 の電源、インターフェースボードの電源を OFF にしてください。

①Windows の場合
Window 版は Cygwin を終了させましょう。

ex) $ exit [*Enter*]

/usr/local/h8/share/samples/misc には多数のサンプルプログラムを用意しています。
いろいろと試してみてください。

6.8 エディタでプログラムを記述

簡単なプログラムを書いてみましょう。
ここで紹介するプログラムは BN-1 の目の前に障害物があった場合、音が「キーン」と鳴って、グラフィックアイには“ ！ ”を表示させてプログラムが終了します。

使い慣れたエディタで下記に記したプログラムを打ち込んでみてください。
（Windows ならメモ帳等を、Linux または FreeBSD なら vi 等が使用出来ます。）

ファイル名は testinfrared.c とし、~/bn1work に保存することにします。
（ここで紹介するプログラム testinfrared.c は/usr/local/h8/share/samples/misc の中に入っています。）

ex) $ cd␣~/bn1work [*Enter*]
$ vi␣testinfrared.c [*Enter*]

（プログラム例）

```
#include <bn1api.h>                /* BN-1 API ヘッダファイル */

int main(void){

        SensorInfo_t     sinfo;            /* センサ情報構造体 */

        /* モニターへ出力 */
        BN1TraceOut(1, "赤外線センサー テストプログラム Start!");
```

```
        BN1PutEye('O'); BN1PutEye('G'); /* グラフィック EYE の表示 */
        BN1BreakOut(2);                 /* プログラムの一時中断 */

        BN1TraceOut(1, "センサー情報を取得します");      /* モニターへ出力 */
        while(1){

                BN1GetSensorInfo(&sinfo);       /* センサー情報を取得する */

                /* 赤外線センサーを取り出し、前に障害物があるか判定 */
                if ( sinfo.infrared & 0x01 )
                        break;                  /* ループを抜ける */
        }

        BN1PlaySound(0x34);                     /* サウンド鳴らす「キーン」 */
        BN1PutEye('!'); BN1PutEye('!');         /* グラフィック EYE の表示 */
        BN1TraceOut(1, "センサーに反応あり!");  /* モニターへ文字列出力 */
        BN1WaitSec(1);                          /* ウエイト(1 秒) */

        BN1PutEye('N'); BN1PutEye('E');         /* グラフィック EYE の表示 */
        BN1BreakOut(2);                         /* プログラムの一時中断 */

        /* モニターへ出力 */
        BN1TraceOut(1, "赤外線センサー テストプログラム End!");

        return(0);
}
```

6.9 Makefile の書き換え

H8 用に用意された GCC クロス環境でコンパイルします。
クロス環境では x86 環境と同様に gcc（h8300-hms-gcc）や gdb（h8300-hms-gdb）等が使用可能です。(但し BN-1 API を用いた実機レベルのデバッグで gdb は使用できません） コンパイルも Makefile を記述することで、make コマンドでコンパイルすることが出来ます。(FreeBSD では gmake をお使いください)

「Makefile」を記述します。
(6.2)の「準備」で「Makefile」のテンプレートを ~/bn1work ディレクトリにコピーしました。
Makefile の内容を一部修正することにします。
お使いのエディタで Makefile を開いて、以下のように修正してください。

ex) $ cd␣~/bn1work [*Enter*]
$ vi␣Makefile [*Enter*]

修正箇所は

```
TARGET = **** <- Specify target mot file
```

を

```
TARGET = testinfrared.mot
```

に書き換えます。
書き換えたファイルは Makefile という名前のまま上書き保存します。

6.10 作成したプログラムをコンパイルする

コンパイルするには make を使用します。
コンパイルの例を以下に示します。

ex) $ make [*Enter*]

```
h8300-hms-gcc -O -mh -I./ -I/usr/local/h8/include/bn1
 -T /usr/local/h8/lib/bn1/bn1ram.x
 -nostartfiles /usr/local/h8/lib/bn1/bn1startup.s
 testinfrared.o -L/usr/local/h8/lib/bn1 -lbn1api -o testinfrared.bin -lc
h8300-hms-objcopy -O srec testinfrared.bin testinfrared.mot
chmod -x testinfrared.mot
rm -f testinfrared.bin
```

TARGET で指定した testinfrared.mot が作成されたか確認してみましょう。
確認は ls コマンドで行います。

ex) $ ls [*Enter*]
    testinfrared.c testinfrared.mot Makefile

◆知っていると便利◆
（1）「make clean」
TARGET で指定した mot ファイルとオブジェクトファイルを削除します。
再コンパイルする際は、make clean を実行してから make しましょう。

ex) $ make␣clean [*Enter*]
    $ make [*Enter*]

（2）「Makefile の記述」
Makefile の中で記述が多くなり、複数行にまたがる場合には行末に「¥」（バックスラッシュ）を入れます。

```
ex) TARGET = testeye.mot testsound.mot testmotion.mot testtimer.mot ¥
             testtrace.mot testsensor.mot
```

（3）「H8/300 オプション」
H8/300 用のオプションとして以下のようなものがあります。

-mh
H8/300H 用のコードを生成します。
BN-1 C 言語開発キットではこのオプションは必須です。

-mrelax
可能であれば、リンク時にいくつかのアドレス参照を短くします。

6.11　カプリロモニターでプログラムを実行

先に説明した(6.5)、(6.6)、(6.7)の手順でカプリロモニターからプログラムを実行します。

ex) $ cd␣~/bn1work [*Enter*]
$ capterm␣-c␣COM1 [*Enter*]

ようこそ BN-1 Monitor へ!
BN-1 Monitor:[1] load␣testinfrared.mot [*Enter*]
..(転送中).....
転送が終了しました。

BN-1 Monitor:[2] run [*Enter*]

BN-1 の目の前に手をかざしてみて下さい。
BN-1 から「キーン」という音が鳴りましたか？
次に、今作成したプログラムをもっと面白く改良してみましょう。

6.12　作成したプログラムの改良

先程作成したプログラム（testinfrared.c）は、BN-1 は動きもせず、
こちらから手をかざすまで何も反応してくれませんでした。
先程作成したプログラム（testinfrared.c）を使って、
障害物を発見したら、障害物を回避するプログラムに改良してみましょう。
（ここで紹介するプログラム avoid.c は/usr/local/h8/share/samples/misc の中に入っています。）

【ステップ１】
メイン関数の while 文にあるセンサー情報を取得している処理と while 文を抜けた後のサウンドを鳴らす処理を関数化してみましょう。

① メイン関数で宣言しているセンサ情報構造体の宣言を削除し、
グローバル変数 EndFlag を追加します。

(例)

```
#include "bn1api.h"                    /* BN-1 API ヘッダファイル */

Bool_t      EndFlag = FALSE;           /* ループ終了のフラグ */


int main(void){
    /* モニターへ出力 */
    BN1TraceOut(1, "赤外線センサー テストプログラム Start!");
      :
      :
```

② センサー情報を取得し、処理するまでの過程を関数化します。
関数名は BN1SensorReaction とします。

(例)

```
void BN1SensorReaction(void){

    SensorInfo_t    sinfo;             /* センサ情報構造体 */

    /* センサー情報を得る */
    BN1TraceOut(1, "センサー情報を取得します");
                                       /* モニターへ文字列出力 */
    BN1GetSensorInfo(&sinfo);          /* センサー情報を取得する */

    /* 前もしくは下に障害物があったときの処理 */
    if ( (sinfo.infrared & 0x01) || (sinfo.infrared & 0x10) ){
        BN1TraceOut(1, "前方に障害物を発見");
                                       /* モニターへ文字列出力 */
        BN1PlaySound(0x34);            /* サウンド鳴らす「キーン」 */
        BN1PutEye('!'); BN1PutEye('!');
                                       /* グラフィック EYE の表示 */
        BN1TraceOut(1, "センサーに反応あり!");
                                       /* モニターへ文字列出力 */
        BN1WaitSec(1);                 /* ウエイト(1 秒) */
```

```
                EndFlag = TRUE;          /* ループ終了のフラグを立てる */
        }
}
```

③メイン関数の while 文の中で BN1SensorReaction 関数を呼ぶようにします。
while 文の後にある「サウンドを鳴らす」、「グラフィックアイ表示」、「文字列出力」、「ウエイト」の処理を削除します。

（例）

```
        EndFlag = FALSE;
        while(!EndFlag){
                /* センサー情報の取得とその後の処理 */
                BN1SensorReaction();
        }

        BN1PutEye('N'); BN1PutEye('E'); /* グラフィック EYE の表示 */
        BN1BreakOut(2);                 /* プログラムの一時中断 */
```

ここまで改良が終わったら、コンパイルして実行してみて下さい。
最初に作成した testinfrared と同じ動作をします。

【ステップ２】
BN-1 に動きをつけましょう。
プログラムが起動したら、BN-1 は前進をし続け、障害物があったら止まるようなプログラムに改良します。

①BN-1 が中速前進をおこなう関数を作成します。
　関数名は BN1InitAction とします。

（例）

```
void BN1InitAction(void)
{
        BN1DoMotion(64);        /* 中速前進 */
}
```

②メイン関数の while 文が始まる前に BN1InitAction() を追加します。

（例）

```
/* BN-1 の初期動作 */
BN1InitAction();

EndFlag = FALSE;
while(!EndFlag){
```

③ センサーが反応したら中速前進が終わるようにしたいので、
　void BN1SensorReaction(void)の if 文の中にある音を鳴らす処理を
　モーション停止に変更しましょう。

（例）

```
if ( (sinfo.infrared & 0x01)==0x01 || (sinfo.infrared & 0x10)==0x10 ){
        BN1TraceOut(1, "前方に障害物を発見");
                                /* モニターへ文字列出力 */
        BN1StopMotion(1);       /* モーション停止（緊張） */
        EndFlag = TRUE;         /* ループ終了のフラグを立てる */
```

```
}
```

ここまで改良が終わったら、コンパイルして実行してみてみます。
フラットケーブル（6 ピン）をインターフェースボードと BN-1 本体に接続したまま実行することは難しいので、カプリロモニターの RUN コマンドでプログラムを実行した後に、BN-1 本体からフラットケーブル（6 ピン）を外すことにします。

転送から実行までの手順は以下のとおりです。

（1）カプリロモニターの load コマンドでプログラムを転送します。

（2）カプリロモニターの setlevel コマンドでプログラムに記述されている BN1TraceOut（カプリロモニターへの文字列表示）と BN1BreakOut（プログラムの一時中断）を実行しないようにします。

ex) BN-1 Monitor:[1] setlevel␣0 [*Enter*]

（3）カプリロモニターの run コマンドでプログラムを実行します。

（4）インターフェースボードの電源を ON にしたまま BN-1 本体からフラットケーブル（6 ピン）を外します。BN-1 前方の赤外線センサーが反応しないよう注意して外して下さい。

BN-1 は前進をし続け、障害物があったら止まることを確認してください。
確認が終わったら、インターフェースボードと BN-1 本体をフラットケーブル（6 ピン）で接続して下さい。（インターフェースボードの電源スイッチは ON のまま接続してください。）

【ステップ３】
障害物発見後の回避行動を追加しましょう。

①センサー反応後の BN-1 の障害物回避行動を関数化します。
　関数名は BN1DoReaction とします。

（例）

```
void BN1DoReaction(void)
{
        /*--- 前に障害物があったときの処理 ---*/

        /* モーションストップ */
        BN1StopMotion(1);                   /* モーション停止（緊張） */
        BN1PlaySound(0x34);                 /* サウンド鳴らす「キーン」 */
        BN1PutEye('!'); BN1PutEye('!'); /* グラフィック EYE の表示 */
        BN1TraceOut(1, "センサーに反応あり!");/* モニターへ文字列出力 */
        BN1WaitSec(1);                      /* ウエイト(1 秒) */

        /* 後に下がる */
        BN1DoMotion(112);                   /* 中速後退 */
        BN1WaitMilliSec(400);               /* ウエイト(0.4 秒) */
                                            /* 0.001×400=0.4s */

        /* 方向転換 */
        BN1DoMotion(4);                     /* 回転姿勢 */
        BN1WaitSec(1);                      /* ウエイト(1 秒) */
        BN1DoMotion(32);                    /* 中速右回転 */
        BN1WaitMilliSec(400);               /* ウエイト(0.4 秒) */
                                            /* 0.001×400=0.4s */

        /* BN-1 普段の動作 */
        BN1NormalAction();
}
```

②void BN1DoReaction(void)の中で出てくるBN1NormalAction関数を作成します。

（例）

```
void BN1NormalAction(void)
{
        BN1DoMotion(64);        /* 中速前進 */
        BN1WaitSec(10);         /* ウエイト(10秒) */
}
```

③センサー情報を取得する関数BN1SensorReactionのif文の中を
下記のように改造します。

（例）

```
if ( (sinfo.infrared & 0x01)==0x01 || (sinfo.infrared & 0x10)==0x10 ){
        BN1TraceOut(1, "前方に障害物を発見");
                                /* モニターへ文字列出力 */
        BN1DoReaction (1);      /* センサー反応後の回避動作 */
        EndFlag = TRUE;         /* ループ終了のフラグを立てる */
}
```

ここまで改良が終わったら、コンパイルして実行してみて下さい。
実行の際は【ステップ２】と同様に、カプリロモニターのRUNコマンドでプログラムを実行し、BN-1本体からフラットケーブル（6ピン）を外して下さい。
BN-1は前進をし続け、障害物があったら後ろに下がり方向転換をします。
その後は障害物があってもBN-1は前進し続けることを確認してください。
(但し、前進してから10秒後にプログラムは終了します。)
確認が終わったら、インターフェースボードとBN-1本体をフラットケーブル（6ピン）で接続して下さい。（インターフェースボードの電源スイッチは ON のまま接続してください。）

【ステップ４】
【ステップ３】の改造では、１回きりしか障害物を避けることが出来ませんでした。
１回ではなく何度でも障害物回避が出来るように改良しましょう。

① センサー情報を取得する関数 BN1SensorReaction を以下のように改良します。
BN1DoReaction()の後の EndFlag=TRUE; の行を削除し、
BN-1 の肉球を同時に触れたらプログラムが終了する処理を追加します。

（例）

```
void BN1SensorReaction(void){

        SensorInfo_t    sinfo;          /* センサ情報構造体 */

        /* センサー情報を得る */
        BN1TraceOut(1, "センサー情報を取得します");
                                        /* モニターへ文字列出力 */
        BN1GetSensorInfo(&sinfo);       /* センサー情報を取得する */

        /* 前もしくは下に障害物があったときの処理 */
        if ( (sinfo.infrared & 0x01) || (sinfo.infrared & 0x10) ){
                BN1TraceOut(1, "前方に障害物を発見");
                                        /* モニターへ文字列出力 */
                BN1DoReaction();        /* センサー反応後の回避動作 */
        }

        /* プログラム終了条件 */
        if ( (sinfo.nikukyu & 0x03)==0x03 ){    /* 両手の肉球が反応 */
                BN1StopMotion(1);       /* モーション停止（緊張） */
                EndFlag = TRUE;         /* ループ終了のフラグを立てる */
        }
}
```

②BN1NormalAction 関数にあるウエイトを削除します。

（例）

```
void BN1NormalAction(void)
{
        BN1DoMotion(64);        /* 中速前進 */
}
```

ここまで改良が終わったら、コンパイルして実行してみて下さい。
実行の際は【ステップ２】と同様に、カプリロモニターの RUN コマンドでプログラムを実行し、BN-1 本体からフラットケーブル（6 ピン）を外して下さい。
BN-1 は１回限りでなく何度も回避行動を行います。
BN-1 の両方の肉球を同時に触れればプログラムは終了します。
確認が終わったら、インターフェースボードと BN-1 本体をフラットケーブル（6 ピン）で接続して下さい。（インターフェースボードの電源スイッチは ON のまま接続してください。）

【ステップ５】
【ステップ４】の改造では、肉球を触れる以外プログラムを終了することが出来ませんでした。プログラム終了条件として、プログラム実行して１分経つとプログラムが終了するように改良しましょう。

①プログラム開始１分後に終了フラグを立てる関数を追加します。
　関数名は InterruptFinishStroll とします。

（例）

```
void InterruptFinishStroll(void){

        /* プログラム終了カウンタが 0 の場合は終了 */
        if( FinishCount <= 0 ){
                BN1StopMotion(1);       /* モーション停止（緊張） */
                EndFlag = TRUE;         /* ループ終了のフラグを立てる */
        }else{
                FinishCount--;
```

```
                    /* プログラム終了カウンタをカウントダウンする */
            }

            BN1SetTimer8(InterruptFinishStroll, 20);/* 166ms * 20 = 約 3.2s */
        }
```

②InterruptFinishStroll 関数内で使用しているプログラム終了カウンタ（FinishCount）をグローバル変数で宣言します。

```
(例)
 #include "bn1api.h"          /* BN-1 API ヘッダファイル */

 Bool_t      EndFlag = FALSE;  /* ループ終了のフラグ */
 int         FinishCount = 20;/* プログラム終了までの秒数をカウントする変数 */
     :
     :
```

③メイン関数の BN1InitAction()の後に BN1SetTimer8 関数を追加します。

```
（例）
        /* BN-1 の初期動作 */
        BN1InitAction();

        /* ユーザレベル 8bit タイマー割り込み関数をセットする */
        BN1SetTimer8(InterruptFinishStroll, 20);/* 166ms * 20 = 約 3.2s */
```

ここまで改良が終わったら、コンパイルして実行してみて下さい。
実行の際は【ステップ２】と同様に、カプリロモニターの RUN コマンドでプログラムを実行し、BN-1 本体からフラットケーブル（6 ピン）を外して下さい。
BN-1 の両方の肉球を同時に触れることがなければ、１分後にプログラムは終了します。
確認が終わったら、インターフェースボードと BN-1 本体をフラットケーブル（6 ピン）で接続して下さい。（インターフェースボードの電源スイッチは ON のまま接続してください。）

【ステップ６（おまけ)】
BN-1 の初期動作（BN1InitAction）を以下のように直して自然な立ち上がりにします。

（例）

```
void BN1InitAction(void)
{
        BN1DoMotion(11);        /* ゆっくりお座り */
        BN1WaitSec(2);          /* ウエイト(2 秒) */
        BN1DoMotion(64);        /* 中速前進 */
}
```

以上でプログラムの改良は終わりです。
この後はご自身で自由にプログラムを変更し、よりロボットらしいプログラムに改良していってください。

==========

付録 A　カプリロモニターのコマンド

==========

【 カプリロモニターのコマンド一覧】

(1) quit

カプリロモニターを終了します。

ex) BN-1 Monitor:[1] quit [*Enter*]
BN-1 Monitor の操作を終了します。
$

(2) load [ファイルパス]

ユーザプログラムを BN-1 の SRAM 領域に転送します。

ex) BN-1 Monitor:[1] load␣../samples/misc/testsound.mot [*Enter*]
..(転送中).....
転送が終了しました。

(3) run

BN-1 の SRAM 領域に転送したプログラムを実行します。

ex) BN-1 Monitor:[2] run [*Enter*]

(4) setlevel [レベル No]

デバッグトレースのレベル No を設定します。
ここで設定した No より大きい値を指定したトレース関数(BN1TraceOut、BN1TraceIn)やブレーク関数(BN1BreakOut)が実行さます。

ex) BN-1 Monitor:[1] setlevel␣3 [*Enter*]
Set Trace Level = 3

(5) hisotry

過去のコマンドヒストリーの一覧を表示します。

ex) BN-1 Monitor:[20] history [*Enter*]
[1] version
[2] load sample.mot
:
[19] run

(6) ![文字列]

過去のコマンドヒストリーの中からマッチするコマンドを再実行します。
(!w とコマンドを打つと、過去の load sample.mot などを容易に再実行できます)

ex) BN-1 Monitor:[1] load␣sample.mot [*Enter*]
..(転送中).....
転送が終了しました。
BN-1 Monitor:[2] run [*Enter*]
:
:
BN-1 Monitor:[3] !w [*Enter*]
..(転送中).....
転送が終了しました。

(7) ![数値]

過去のコマンドヒストリーの中から指定した番号のコマンドを再実行します。
番号はプロンプトに記載されているコマンド番号および history コマンド
で表示された番号になります。

ex) BN-1 Monitor:[10] version [*Enter*]
Firmware Version 0.21
BN-1 Monitor:[11] load␣sample.mot [*Enter*]
..(転送中).....
転送が終了しました。

BN-1 Monitor:[12] run [*Enter*]
BN-1 Monitor:[13] !10 [*Enter*]
Firmware Version 0.21

(8) !!

直前のコマンドを実行します。

ex) BN-1 Monitor:[1] load␣sample.mot [*Enter*]
..(転送中).....
転送が終了しました。
BN-1 Monitor:[2] !! [*Enter*]
..(転送中).....
転送が終了しました。

(9) sleep

BN-1 を sleep（電源 OFF）状態にします

ex) BN-1 Monitor:[1] sleep [*Enter*]
sleep mode に設定しました。

(10)wakeup

BN-1 の電源を ON にします。

ex) BN-1 Monitor:[1] wakeup [*Enter*]

(11) version

現在ご使用のカプリロモニターの version を表示します。

ex) BN-1 Monitor:[1] version [*Enter*]
Firmware Version 0.21

==========

付録 B　UNIX コマンド

==========

UNIX で使われる基本的なコマンドを紹介します。
BN-1 C 言語開発キットを使用する上で最低限必要と思われるものだけを紹介します。
UNIX には便利なコマンドが多数ありますので、ご興味のある方は UNIX 関連の参考書をご覧下さい。

○　ls

■機能　　指定されたパスにあるファイルやディレクトリの一覧を表示します。
　　パスを指定しないと、カレントディレクトリの一覧を表示します。

■書式　ls（**オプション**）（**パス**）

■オプション

-a　　ドット(.)で始まるファイルも含めて表示します。
-l　　ファイルの詳細情報を表示します。
-F　　ファイル名にファイルの種類を表すものを付加する。
　　スラッシュ(/)はディレクトリ
　　アスタリスク(*)は実行可能ファイル
　　アットマーク(@)はシンボリックリンク

ex) $ ls
　　ドットで始まるファイルも含めて、
　　さらに詳細なファイル情報を知るには
　　$ ls -al

○　pwd

■機能　　現在の作業ディレクトリを絶対パスで標準出力に出力します。

ex) $ pwd
　　/home/Administrator

○ cd

■機能　作業ディレクトリを変更します。

■書式　cd［ディレクトリ］

[ディレクトリ]には新しい作業ディレクトリになる絶対パスもしくは相対パス名を表示します。
引数の[ディレクトリ]を省略した場合はホームディレクトリが新しい作業ディレクトリとなります。

（＊）注意！
MS-DOS ではディレクトリの区切りに'¥'を使用しますが、
UNIX および Cygwin 上では'/'(スラッシュ)を使用します。

ex）$ cd␣work

○ mkdir

■機能　ディレクトリを作成します。

■書式　mkdir［ディレクトリ名］

ex）$ mkdir␣samples

○ cp

■機能　ファイルをコピーします。

■書式　cp（オプション）［コピー元ファイル］［コピー先ファイル］

■オプション

-i　すでに存在しているファイルがあった場合、そのファイルへの上書きを行うかどうか確認するように出力する。

-f　すでに存在しているファイルがあった場合、強制的にファイルを上書きする。

-R　[コピー元ファイル]にディレクトリが指定された場合、そのディレクトリ以下の全てのファイルをコピーします。

ex）$ cp␣orgfile.c␣newfile.c

○　rm

■機能　　ファイルを削除します。

■書式　　rm（**オプション**）[**ファイル名**]

■オプション

-i　指定されたファイルを削除する前に確認を求めるようにします。

-r　[ファイル名]にディレクトリが指定された場合､そのディレクトリ以下を削除します。

ex）$ rm␣file.c

○　mv

■機能　　ファイルを指定されたディレクトリに移動します。
　　ファイル名（ディレクトリ名）を変更します。

■書式　　mv（**オプション**）[**ファイル名**][**移動先**]

■オプション

-i　すでに存在しているファイルがあった場合、そのファイルへの上書きを行うかどうか確認するように出力する。

-f　すでに存在しているファイルがあった場合、強制的にファイルを上書きする。

ex）$ mv␣file.c␣work/samples

==========

付録Ｃ　ｖｉの使い方

==========

画面上でテキストを編集する代表的なスクリーンエディタ vi の基本的な使い方
について紹介します。

○コマンドモードとテキスト入力モード

vi にはコマンドモードとテキスト入力モードの２種類があります。
コマンドモードとは、コマンドを入力するモードです。
テキスト入力モードでは、テキスト内容を入力するモードです。

コマンドモードで「Hello」と打ってもテキスト入力することが出来ません。
入力するときは、テキスト入力モードに切り替えてから
「Hello」と入力します。

vi を起動したときはコマンドモードになっています。
テキスト入力モードにするには、" i "コマンド等を使います。
テキスト入力モードからコマンドモードに戻るには [ESC]キーを押します。

○vi の起動

vi の起動は以下のように行います。

```
vi [ファイル名]
```

ファイル名に、
（１） 存在しないファイルを指定した場合
→ 新しいファイルが作成されます。
（２）既存のファイルを指定した場合
→ ファイルの内容を編集することが出来ます。

vi を起動すると、画面の各先頭行には ~ が表示されます。

【実行例】

```
$ vi␣test.c [Enter]
```

○vi の終了

コマンドモードで

ZZ　テキストで書かれた内容をファイルに保存し、vi が終了します。

:q　vi を終了します。（但し、:w でテキストで書かれた内容をファイルに保存してからでないと使えません。）

:q!　vi を強制終了します。

○ファイルの保存

:w　編集中の内容を現在開いているファイルに保存します。

:w!　:w で保存を拒否されたような時でも強制的に保存します。

:w [ファイル名]

[ファイル名]として保存します。

○カーソル移動コマンド

h　カーソルの位置から左へ１文字移動

j　カーソルの位置から下へ１文字移動

k　カーソルの位置から上へ１文字移動

l　カーソルの位置から右へ１文字移動

上記のコマンドの直前に数字を入れると、その数字分カーソル移動が行われます。

ex) 7j　カーソル位置から下へ７文字移動

0　カレント行の先頭に移動

$　カレント行の行末に移動

○画面単位のカーソル移動

[CTRL] + f　次の画面を表示します。

[CTRL] + b　前の画面を表示します。

[CTRL] + d　次の半画面を表示します。

[CTRL] + u　前の半画面を表示します。

○挿入コマンド

i	カーソルの左側からテキストを挿入します。
a	カーソルの右側からテキストを挿入します。
o	カーソル行と次の行の間に新しい行を挿入します。
O	カーソル行と前の行の間に新しい行を挿入します。

コマンドモードに戻るには [ESC]キーを押します。

○削除コマンド

x	カーソルの置かれた文字のみ削除します。
数字 x	数字分の文字を削除します。削除されるのはカーソル位置から右方向の数字分です。
dd	カーソルの置かれたカレント行のみ削除します。
数字 dd	数字分の行を削除します。 削除されるのはカーソル位置から下方向の数字分です。
dw	カーソルの置かれた単語のみ削除します。
数字 dw	数字分の単語を削除します。
d$	カーソルの位置から行末まで削除されます。
d0	カーソルの位置から行頭まで削除されます。

○その他

u	直前に行った編集コマンドを取り消します。
.	直前のコマンドを再度実行します。

==========

付録 D　注意事項

==========

（1）h8300-hms-gcc クロスコンパイラ環境と x86 用 gcc コンパイラ環境の違いについて

処理系によってコンパイラ仕様が異なりますので、注意する必要があります。
「int 型」と「double 型」のサイズは h8300-hms-gcc と x86 用 gcc とで異なります。
h8300-hms-gcc と x86 用 gcc のデータ型とサイズを以下に示します。

表 D.1　h8300-hms-gcc（BN-1）

型	サイズ (バイト)
char	1
int	2
unsigned int	2
short	2
float	4
long	4
double	4

表 D.2　gcc（x86 用）

型	サイズ (バイト)
char	1
int	4
unsigned int	4
short	2
float	4
long	4
double	8

また、代入処理を行った時の型変換も h8300-hms-gcc と x86 用 gcc とで異なりますので注意する必要があります。
「int 型」と「char 型」の型変換について、下記のようなプログラム記述があったとします。

```
char    temp0;

temp0 = -127;

if( temp0 == -127 ){

}
```

h8300-hms-gcc でコンパイルしたとき、if 文の論理値は真になりません。
-127 は ANSI 規格では int 型とみなしますので、16 進表記としたとき、ff81 となります。
一方、変数 temp0 は char で宣言しているので、16 進表記で 81 となります。
上記の if 文は 81 と ff81 を比較するので、論理値は偽となり、正しい結果が得られません。int 型と char 型の検証プログラム例を次に示します。

（プログラム例）

```
#include <stdio.h>

int main(void){
        char    cval=-127;
        short   sval=-127;
        char    temp0;

        printf("char val = %x¥n", cval);
        printf("short val = %x¥n", sval);

        temp0=-127;
        if(temp0 == -127){
                printf("Yes temp0 = %x¥n", temp0);
        }else{
                printf("No temp0 = %x¥n", temp0);
        }
}
```

上記のサンプルプログラムを h8300-hms-gcc と x86 用 gcc の場合に分けて、
コンパイルし gdb を用いて実行したとき、次のような出力結果が得られます。

（h8300-hms-gcc& h8300-hms-gdb の出力結果）

```
char val = 81
short val = ff81
No temp0 = 81
```

（x86 用 gcc 出力結果）

```
char val = ffffff81
short val = ffffff81
Yes temp0 = ffffff81
```

このような問題の解決策としては、

```
if(temp0 == (char)-127){
```

のように定数を char 型にキャストすることで同一のものとして比較出来ます。

（2）SRAM 領域の制限について

BN-1 のメモリマップは以下のようになっています。
ユーザプログラムを書き込む事が出来る SRAM 領域は、0x200000 番地から 0x20FFFF 番地までの 64k バイト分となります。これらアドレス空間の定義はリンカスクリプトで指定しています。(/usr/local/h8/lib/bn1/bn1ram.x)
プログラム作成の際には作成した H8 バイナリーイメージが 64k バイト以上の大きさにならないよう注意してください。

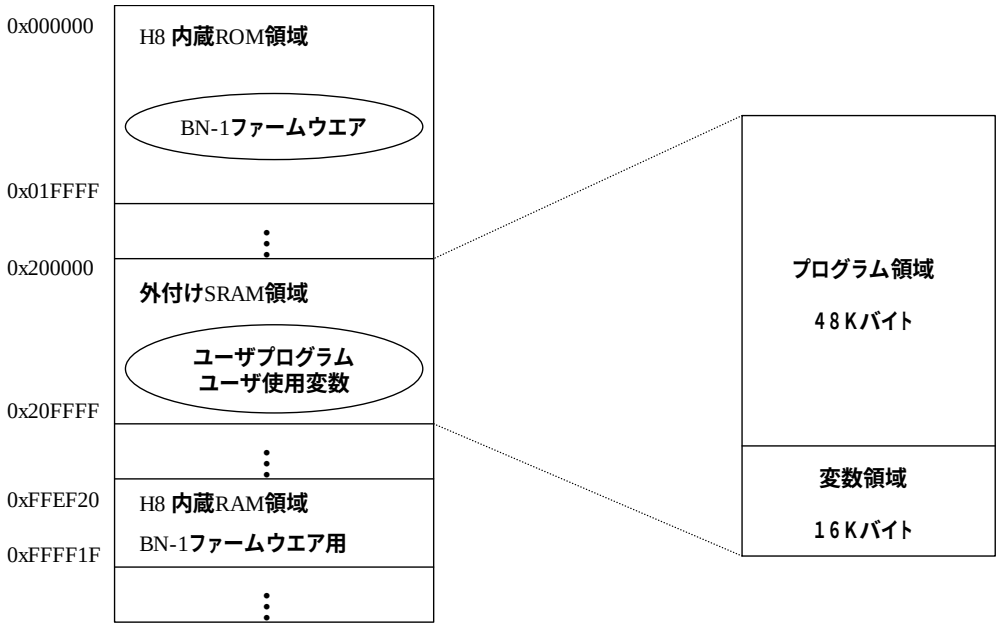


図 D.1 メモリマップ

（3）スタック領域

BN-1 実行環境ではスタック領域（4 バイト)を BN-1 に内蔵の H8RAM 領域に確保しています。H8RAM 領域は 4k バイトとなっており、図のように FFFF1C～FFFF1F 番地をスタック領域として確保しています。

スタックが増えれば、スタック領域はアドレスの小さい方向へ伸びていきます。

ここで注意したいのは、関数を次から次へと呼び出し、スタックがどんどん増え続けた時、スタックが他のデータが使用している記憶領域を壊したりしてしまうことです。

このようになりますと、プログラムは暴走してしまいますので、プログラム作成時には注意が必要です。

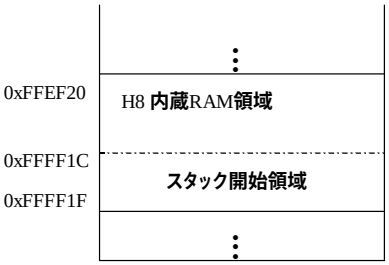


図 D.2　スタック開始領域

○スタックとは

スタックとは、積み重ねとか棚というような意味で、データを積み重ねて一時的に格納させておく領域のことをいいます。

スタックにデータを格納する時は下図にあるように棚の底からデータをどんどん積み上げていき（プッシュ)、データを取り出すときに後から積んだデータから取り出します（ポップ)。このような後入れ先出しの構造を LIFO（Last In First Out）と呼びます。

BN-1 C 言語開発キットでもサブルーチンの処理に使われており、一時的にレジスタの値を退避させたり、サブルーチンへの分岐の戻り番地を入れたりするのに用いています。

一般的に関数の呼び出し時にスタックが積まれると考えて下さい。

つまり、関数の中で自身の関数を呼ぶようなリカーシブコール（再帰呼び出し）を制限なしで多用しますと、暴走します。(程度の問題ですが)

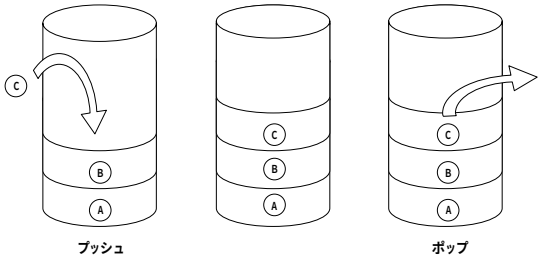


図 D.3　LIFO

（4）H8 レジスタの操作

H8 のレジスタはリニアなアドレス空間にありますので、ユーザプログラムからレジスタ操作をすることも可能ですが、BN-1 C 言語開発キットではレジスタ操作を BN-1API でのみ行うものとしておりますので、万一、ユーザがユーザプログラムレベルで H8 のレジスタを操作し、BN-1 を動作させる場合につきましては保証外となりますので予め御了承下さい。

（5）逆アセンブリによる改造

逆アセンブリによる改造につきましてはサポート外となります。

（6）exit 関数の利用不可

main 関数でプログラムを終了させるとき、BN-1 C 言語開発キットでは exit 関数を利用する事が出来ません。プログラムの実行を終了させたい場合は return 文を使用して下さい。

==========

付録 E　トラブルシューティング

==========

項目	症状	原因･解決策	参照ページ
BN-1本体	BN-1本体の電源スイッチを押しましたが、背中の作動ランプが点灯しません。	本体のバッテリーは正しくセットされていますか？	＊
		本体のバッテリーは充電されていますか？	＊、P.35
	グラフィックアイの表示が普段と比べて暗いです。	BN-1本体のバッテリー切れが考えられます。バッテリーを充電して下さい。	＊、P.35
	BN-1本体の電源スイッチを押してもBN-1はすぐに寝てしまいます。	BN-1本体のバッテリー切れが考えられます。バッテリーを充電して下さい。	＊、P.35
ｲﾝｽﾄｰﾙ	Cygwinのインストール中、ダイアログの「Next ･･>」を選択し続けていたら、インターネットに接続してしまいました。	Cygwin のインストールを進めていくと、Cygwin のインストーラが行う処理を選択するダイアログが表示されます。ここで「Install from Internet」もしくは「Download from Internet」を選択しますとインターネットに接続されてしまいます。ここでは「Install from Local Directory」を選択して下さい。	P.14
	Cygwinのインストールが上手くいきません。	Windows2000をお使いの場合、Administrator権限でインストールして下さい。	P.10
		アンチウイルスソフトのリアルタイム検索を無効にして下さい。	P.10
	H8クロス環境のインストールが上手くいきません。	Cygwinをお使いの場合、CD-ROMドライブ名は正しいですか？（Q:? D:? E:?）	P.19
		Linux、FreeBSDをお使いの場合、インストールはroot権限で行っていますか？	P.19
		選択したOSは正しいですか？	P.21
ﾌｧｰﾑｳｴｱ	ファームウエアの更新が出来ません。	シリアルポートは有効になっていますか？パソコンのマニュアルを参照し、シリアルポートの設定を確認してください。	P.7
		シリアルデバイスの読取りと書込み許可はされていますか？（Linux版、FreeBSD版）	P.7、P.23
		パソコンとインターフェースボードはシリアルケーブルで繋がっていますか？	P.29
		インターフェースボードに接続されているフラットケーブル（6ピン）はBN-1本体の下面に繋がっていますか？	P.29
		BN-1本体のバッテリーは正しくセットされていますか？	＊
		BN-1本体のバッテリーは充電されていますか？	＊、P.35
		インターフェースボードの電池（9V乾電池）は正しくセットされていますか？	―

＊　…　BN-1 の取扱説明書を参照下さい

項目	症状	原因･解決策	参照ページ
ﾌｧｰﾑｳｴｱ	ファームウエアの更新が出来ません。	インターフェースボードは電池切れになっていませんか？ 今まで問題なく行っていたファームウエアの更新が急に出来なくなった場合は、原因としてインターフェイスボードの電池切れが考えがえられます。 このような時は電池を新品に交換してみて下さい。	―
		インターフェイスボードの切替スイッチは「ROM WRITE」にセットされていますか？	P.30、P.42
		インターフェイスボードの電源スイッチは「ON」にセットされていますか？	P.30、P.42
		ファームウエアの更新手順は正しく行いましたか？	P.30、P.42
		ライターの引数に/usr/local/h8/share/firmware/capmon-v100.motを指定していますか？	P.30、P.42
		capmon-v100.mot は/usr/local/h8/share/firmware にインストールされていますか？	P.67
		ライター起動時にシリアルポートを正しく指定していますか？	P.30、31 P.41
		trustwrite は/usr/local/h8/bin にインストールされていますか？	P.68
		パスの設定は正しく行われていますか？	P.24、25
		シリアルポートが使用中ではありませんか？	―
		trustwrite の引数で -s 9600 の指定をして、より低速な転送速度でお試しください。	P.32
	BN-1がペットモードに戻りません。	ライターの引数に/usr/local/h8/share/firmware/bn1org-v03.motを指定していますか？	P.33
		ライターでbn1org-v03.motを書込みした後、メモリクリアを行いましたか？	P.34
		bn1org-v03.mot は/usr/local/h8/share/firmware にインストールされていますか？	P.68
	ファームウエアのVersionがわかりません。	カプリロモニタ(capterm)上でversion[Enter] と打ってください。	P.96
カプリロモニター	カプリロモニターが起動しません。	シリアルポートは有効になっていますか？ パソコンのマニュアルを参照し、シリアルポートの設定を確認してください。	P.7
		シリアルデバイスの読取りと書込み許可はされていますか？（Linux版、FreeBSD版）	P.7、P.23
		パソコンとインターフェースボードはシリアルケーブルで繋がっていますか？	P.29
		インターフェースボードに接続されているフラットケーブル（6ピン）はBN-1本体の下面に繋がっていますか？	P.29
		BN-1本体のバッテリーは正しくセットされていますか？	＊
		BN-1本体のバッテリーは充電されていますか？	＊、P.35
		インターフェースボードの電池（9V乾電池）は正しくセットされていますか？	―

＊ … BN-1 の取扱説明書を参照下さい

項目	症状	原因･解決策	参照ページ
カプリロモニター	カプリロモニターが起動しません。	インターフェースボードの電池切れになっていませんか？	—
		インターフェイスボードの切替スイッチは「RUN」にセットされていますか？	P.42
		インターフェイスボードの電源スイッチは「ON」にセットされていますか？	P.42
		BN-1本体の電源スイッチは「ON」になっていますか？	P.42
		カプリロモニターの起動手順は正しく行いましたか？	P.42
		カプリロモニター起動時にシリアルポートを正しく指定していますか？	P.46
		capterm は/usr/local/h8/bin にインストールされていますか？	P.68
		パスの設定は正しく行われていますか？	P.24、25
		ファームウエアを更新しましたか？	P.28～
	motファイルを転送することが出来ません。	load コマンドで正しくファイル（パスも含む）を指定しましたか？ .c ファイルを指定していませんか？	P.78、95
	モニター上で文字を打ち間違えたのですが、[Back space]キーが使えません。	モニター上で[Back space]キーが使えない時は[Delete]キーを使って下さい。	—
	カプリロモニターの反応がありません。	[CTRL] + [c] でカプリロモニターを強制終了して下さい。	—
	カプリロモニターを起動していて、[ENTER]キーを押し続けていたらモニターが固まってしまいました。	[CTRL] + [c] でカプリロモニターを強制終了して下さい。	—
コンパイル	コンパイル出来ません。	H8GCCクロス環境をインストールしましたか？	P.19～
		/usr/local/h8/bin にパスが通っていますか？	P.24、25
		作成したCソースファイルの拡張子は .c になっていますか？	—
		Makefileにあるターゲットファイルは正しく指定してありますか？	P.53
		MakefileとCソースファイルは同じディレクトリ内に置いてありますか？（Makefileのテンプレートを使用した場合）	P.53
		FreeBSD版ではmakeはgmakeを使用して下さい。gmakeはFreeBSDのパッケージからインストールして下さい。	P.52

＊ … BN-1 の取扱説明書を参照下さい