2003.07.25

情報工学ゼミナール研究

μITRON 仕様に基づく組込みリアルタイム OS の研究

情報工学科　藤川雅男（r00h0390）
多賀創（r00h0211）

目次

研究テーマ

μITRON 仕様に基づいた組込みリアルタイム OS の開発

開発目標

前期：リアルタイム OS の感覚に慣れるため、簡単なロボットを製作し、既存のμITRON 仕様 OS を使って制御を行う。

後期：μITRON 仕様 OS そのものを作成してみる。

メンバの役割分担

マイクロマウス設計・組立　　：藤川

TOPPERS 開発方法研究　　：多賀

進行状況

4 月　　リアルタイム OS についての基礎研究、マイクロマウス設計

5 月　　マイクロマウス製作、OS を使わないプログラミングの練習、OS と開発環境の決定

6 月～　OS を使ったプログラミングについての研究（開発環境の整備）

研究の概要

１．組込みシステムとリアルタイム OS

１．１．組込みシステム

組込みシステムとは、PC 等の汎用的なコンピュータ用途とは異なり、特化された用途のために製品に組み込まれたコンピュータシステムのことである。家電製品・通信機器・車・人工衛星など身の回りのあらゆる製品に組み込まれている。

特化用途の製品（たとえばカメラ・車など）はもともとコンピュータとは関係なく存在したが、組込みシステムと結びつくことによってさらに付加価値を増大し、現在では組込みシステムなしには成立し得ないほどになっている。

１．２．組込みシステムのシステム的要求

組込みシステムは多種多様な製品に対応しているので、システム的要求もまた一様ではない。たとえば戦闘機のシステムは人命にかかわるため対故障性を重視するが、炊飯器のシステムは対故障性よりも安価に大量生産できることを重視する。

１．３．リアルタイム処理とマルチプログラミング

大半の組込みシステムは一定時間に応答のあることを要求する。システムに応答を要求する事態をイベントと呼ぶ。何らかのイベントが発生すると、それに対して一定時間内に何らかの反応を返すことを要求するシステムをリアルタイムシステムと呼ぶ。制限時間の比較的穏やかなリアルタイムシステムをソフトリアルタイムシステム、厳格なものをハードリアルタイムシステムと呼ぶ。リアルタイム性を保証する仕組みをもった OS をリアルタイム OS と呼ぶ。

割り込みによって MPU はイベントの発生を受け付ける。割り込みが発生した時点でプログラムの流れを変え、緊急度の高い処理を優先的に実行するのがリアルタイム処理である。このようなプログラムの制御を可能にするには、プログラムはイベントに対応して切り替えが可能になるように作成する必要がある。そのようなプログラム制御を可能にする方式をマルチプログラミングと呼ぶ。

リアルタイム処理はマルチプログラミングを前提にしているが、マルチプログラミングであるからといってリアルタイム処理が可能というわけではない。

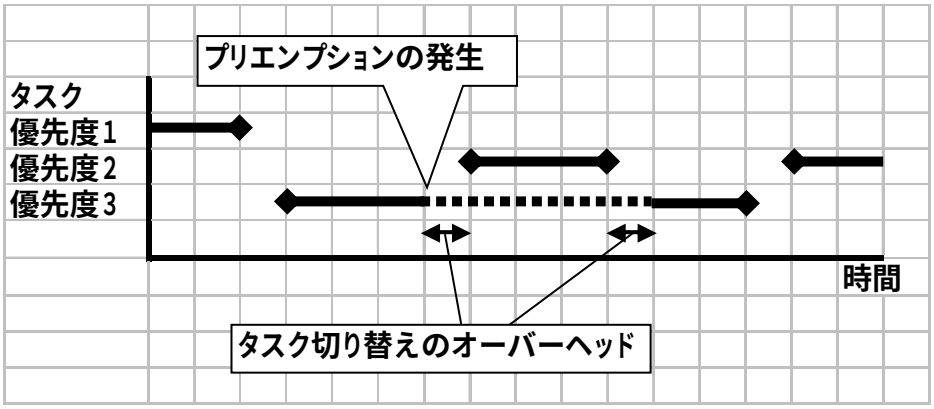
１．４．マルチプログラミングとは

逐次処理を前提に作成された複数のプログラムを、何らかのトリガによって切り替えながら実行させる方式である。

１．５．タスク

マルチプログラミング環境で実行される逐次処理の単位を、タスクと呼ぶ。MPU を交互使用するためにあるタスクが別のタスクに中断されても、中断されたタスクはそのことを知ることはないし、知る必要もない。

一般に逐次処理の流れをコンテキストと呼び、コンテキストが切り替わることをコンテキストスイッチングと呼ぶ。コンテキストスイッチングでは逐次処理の流れを保証するものを退避し、再開時に復旧する。具体的にはレジスタをスタックに退避する。あるタスクが別タスクによって中断されている状態をプリエンプションされているという。



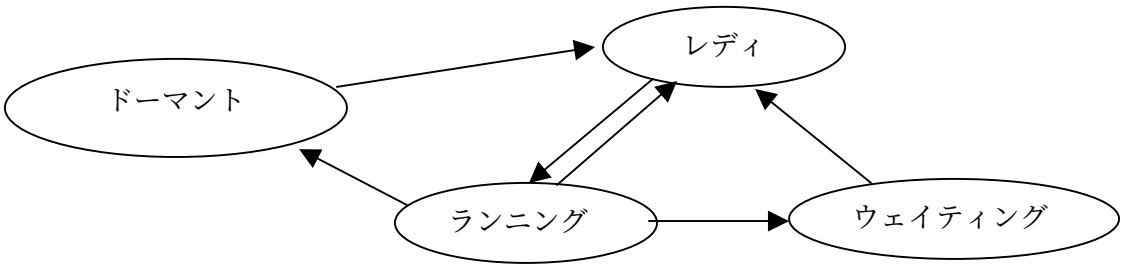
１．６．スケジューリング

割り込み処理は割り込まれたタスクに戻らずに、次に実行するタスクをどれにするか決定する処理を実行することができる。次に実行するタスクを決定することをスケジューリングと呼び、それを実行するプログラムをスケジューラと呼ぶ。スケジューリングは OS の主要な機能である。

１．７．リアルタイム OS の基本的な機能

リアルタイム OS は MPU の管理（スケジューリング）、メモリの管理、割り込みの管理、時間管理といったことを行う。

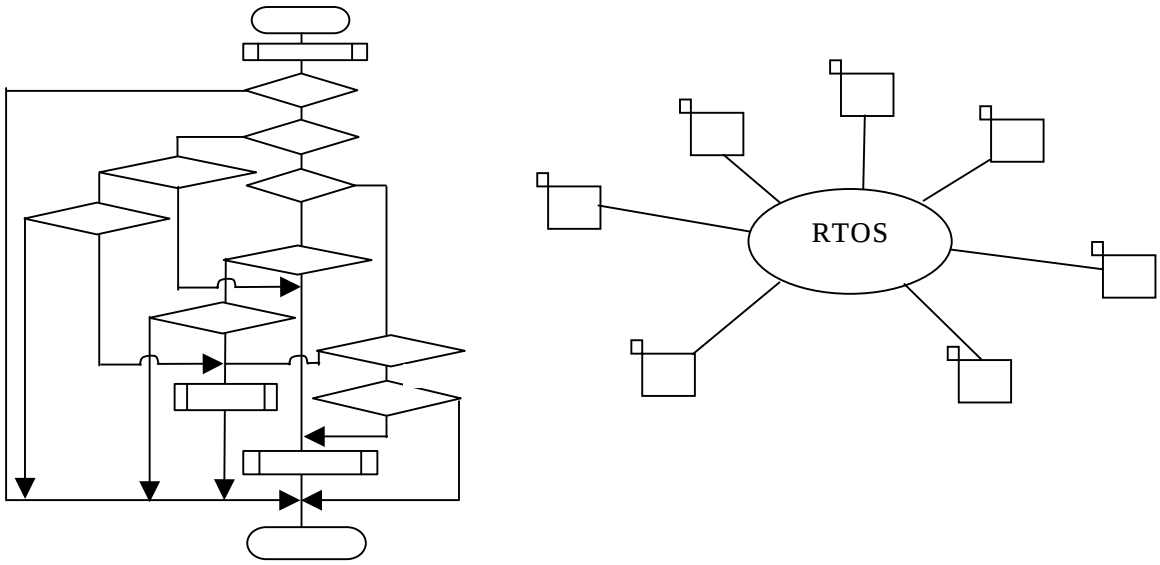
タスクはレディ（実行可能状態）・ウェイティング（実行待ち状態）・ランニング（実行状態）・ドーマント（休止状態）などの状態を持つ。OS は各タスクの優先順位に基づいてタスクの状態遷移を行い、MPU の割り当てを行う。割り込みが入った時「最も優先順位が高く」「最もはやく実行要求を出した」タスクがランニングになり、MPU が割り当てられる。その時優先順位の低いタスクがランニングの状態にあった場合、そのタスクは待ち状態になる。



１．８．OS を使うことによって得られるメリット

① OS を使用したときのほうが一般に見通しの良いプログラムを書くことができる。たとえばセンサが一つ増えたらタスクが一つ増えるといった形で対応可能であり、プログラムの拡張性・保守性が良くなる。

② 要求があるとすぐにその要求に対応した処理を実行することができる。実行までの時間は数マイクロ秒のオーダーで済む。



（左）通常のプログラムのイメージ（右）OS を用いたプログラムのイメージ

１．９．組込み OS の具体例

ITRON 仕様 OS と Linux が現在流行している。

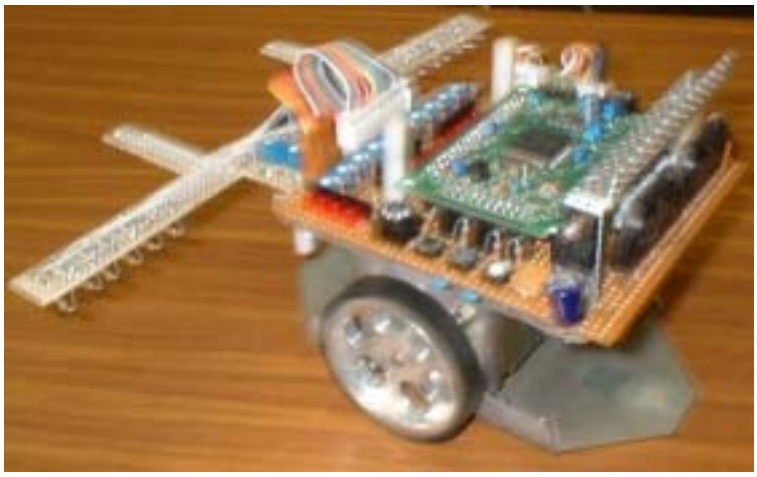
１．９．１．ITRON（Industrial TRON）

現在μITRON4.0 が最新の仕様として公開されている。日本ではこれが最も広く普及している。当初 ITRON はすべての機能を搭載しなければならない規格だったが、サイズが大きくなるため、内容を選択的に用いることができるμITRON 仕様と分離した。μITRON3.0 以降からはμITRON に統一された。

１．９．２．RTLinux（Real Time Linux）

Linux とは、UNIX 仕様の API を提供するフリーの OS である。これをリアルタイムシステムをサポートするように直したものが RTLinux である。本来 Linux は汎用システム用の OS であり、コードサイズは巨大なものとなっており、利用するためには数メガのメモリが必要となる。そのためこれまではあまり一般的には使われていなかったが、最近は SuperH のように大きなメモリを積んだマイコンが安価で提供されるようになったため、流行の兆しがある。

２．マイクロマウスの設計開発



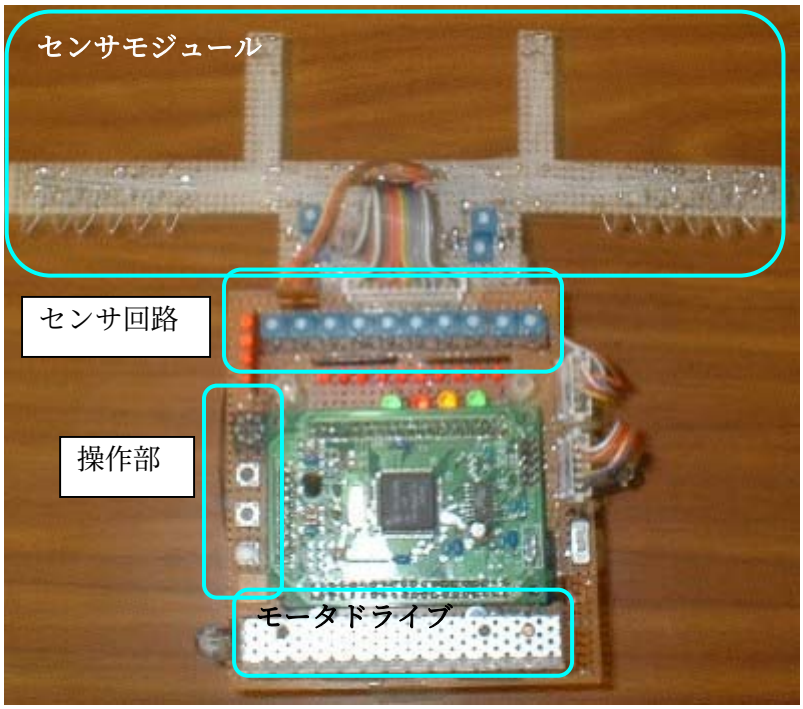
マイクロマウスの例（2002 年度藤川作成）

２．１．マイクロマウスとは

マイクロマウスとは、自立式の迷路探索ロボットである。

主な機能

- モーターを回転させることによる移動
- センサ入力による姿勢制御
- 迷路探索
- 最短距離探索



２．２．なぜマイクロマウスを作成するか

まず RTOS というものに慣れるため、すでに完成している μITRON 仕様 OS を使い、簡単なロボットを制御するプログラム開発の方法について学ぼうと考えた。

マイクロマウスは、センサやアクチュエータを使用して移動しつつ迷路探索の計算を行うロボットである。ロボットとしてわかりやすい能力を持つ上、マルチタスクにするときのタスク分割も容易である。また、OS を使わないときのデータの蓄積があり、新たに覚えねばならないことも少ない。そのため、OS 使用の練習用として適していると考える。

２．３．マイクロマウスに求められる能力

- ステッピングモータを駆動して移動・方向転換する能力
- センサを使って壁にぶつからないように軌道修正する能力
- センサからの入力と記憶データをもとに地図の作成を行い、現在の自分の位置からめ色探索を行う能力
- 記憶データをもとに最短経路を計算する能力
- 以上の能力を統括し、自動的に適切な走行を行う能力

２．４．μITRON を使用することによりどれだけのリソースを消費するか？

２．４．１．タスク切り替えに伴うオーバーヘッド

タスクの切り替えにオーバーヘッドが生じるが、それがどの程度のものなのかはまだよくわからない。数マイクロ秒程度と言われる。

２．４．２．メモリ

C 言語のプログラムをコンパイルすると、メモリには以下のように振り分けが行われる。

ROM：	P 領域：プログラム
	C 領域：定数
	D 領域：初期値のある変数（１）
RAM：	D 領域：初期値のある変数（２）
	B 領域：変数
	残りの領域：スタック

プログラムのスタートアップルーチンには ROM の D 領域のデータを RAM の D 領域にコピーする動作が書かれる。

μITRON そのものは、Windows のような OS と違ってユーザのコードと一緒にコンパイルを行うため、どの機能を使うかによって大きさが異なる。そのため、実際にコンパイルをして見てみないとよくわからない。しかし、H8-3048F の内蔵 ROM は 128 キロバイトあるので、相当大きなプログラムになったとしても十分な容量がある。そのため、これに関してはあまり容量の心配をする必要はないと思われる。

ただ、TRON を使用する場合、タスクごとに独自のスタック領域を使用するので、スタック領域はたくさん取る必要がある。昨年度のマイクロマウス走行用プログラムは、H8-3048F 内蔵の RAM（4 キロバイト）だけで動かすことが可能であった。単純計算で、1 タスクにつき 4 キロバイト使うとして 20 タスク作成したとすると、

４×２０＝８０

より、80 キロバイトの RAM を使うことになる。これほど多くなるかどうかは疑問だが、用心のため RAM は多めに増設する方が良いと考えられる。

２．５．各部品の選定

以上の要件および過去の経験、世間での状況などを加味して、下記のとおり各部品を選定した。

２．５．１．CPU の選定

CPU には、H8-3048F を用いる。理由は、以下のとおりである。

・ 安価であり、手軽に手に入れることができる。
・ 世間で広く使われており、資料が得やすい。
・ メモリ増設が可能である。
・ I/O ポートが豊富なので、増設の手間が省ける。
・ 小さいのでロボットに積みやすい。
・ この CPU 用のμITRON が無償で提供されている。

２．５．２．モータの選定

モータは KH39FM2-801 を用いる。これは小さいうえに十分なパワーがあり、マイクロマウス大会で使われた実績も多く、入手も容易なモータだからである。これを回すための方法はソフト的な方法と専用素子を使ったハード的な方法があるが、プログラムを単純にして I/O ポートを節約するために専用素子を用いる。

CPU のタイマ割り込みによりトグル出力を行い、このパルスを TA8415 でモータ回転用の 4 つのパルス列に変化させる。これを電流チョッパ型モータドライブ素子 SLA7033 に与えてモータを回転させる。プログラム側としては、タイマの割り込み発生の間隔を決めるレジスタの値を変えるだけでスピードの変化を行うことができる。

２．５．３．センサの選定

センサには横向きのものと上から見るものとがあるが、ここでは原理が単純な上から見る形のものを用いる。センサ用素子には反射型フォトインタラプタ EE-SY121 を用い、パルスにより点灯させる。この素子を選んだ理由は、値段が安いこと（１個 20 円)、これまでの実績もあることである。また、パルス点灯にした理由は、2002 年度マイクロマウス大会で常時点灯方式を用いたところ、過電流のため三端子レギュレータが破損した経験から、電力を節約する必要があると考えたからである。

２．５．４．車体

車体は、森永英一郎氏の「ベーシックマウス」を手本として作成したものを流用する。
参考：http://www8.big.or.jp/~morinaga/basicmouse/frame_basicmouse.htm

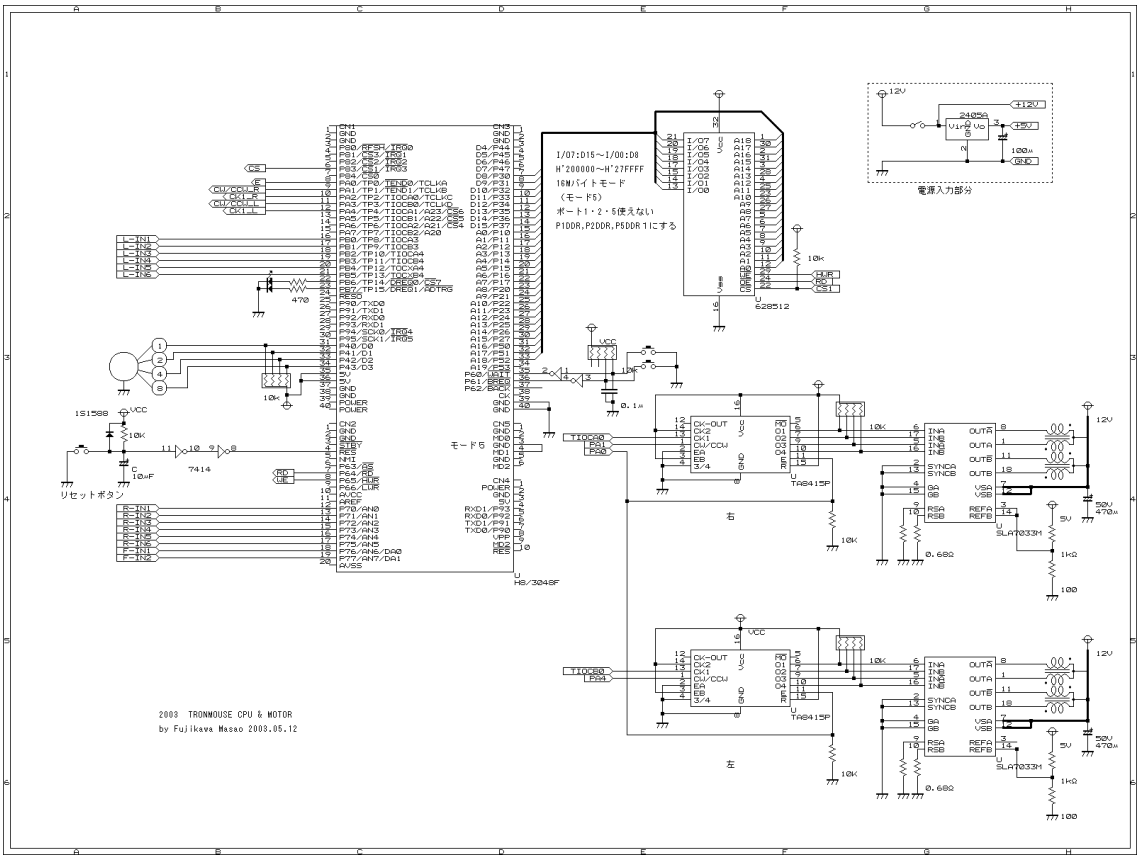
２．６．最終決定したマシンの仕様

以上より、次のような仕様のマシンを作成することとした。

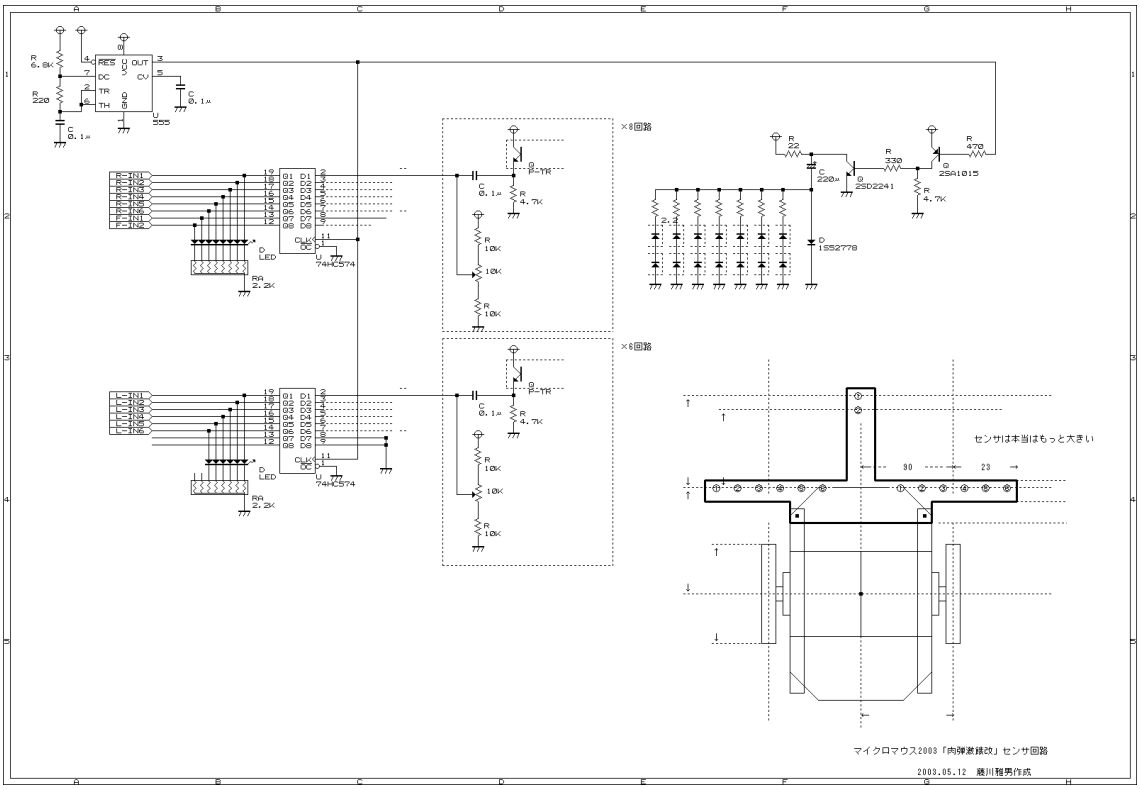
車体	藤川が以前作っていたものを流用。
－筐体	ガラスエポキシ製基板を組み合わせて使用。
－車輪	市販のアルミ製ホイール。上に粘性の高いゴムシートを貼り付けた。
CPU	H8-3048F（秋月電子 AKI-H8 3048F）を使用。
アクチュエータ	ステッピングモータ 2 つ
センサ	壁上面検出タイプ（ウィング形)。反射型フォトインタラプタ 14 個（オムロン EE-SY121。前 2 つ、左右各 6 つ）使用
電源	ニッケル水素電池単 3 型 1.2V、1700mAh（GP 社 170AAHC）12 本（14.4 ボルト)、3 端子レギュレータ（LM2940）で降圧。
プログラム開発の方法	C 言語による。コーディングはベストテクノロジー社の GCC Developer Lite を使用。 AKI-H8 へのファイルの転送には FlashWriter2.6 を使用。 転送ケーブルは RS232C ケーブルの一端をステレオジャックに

	付け替えたものを使用。受け側には、AKI-H8 の、脚の一部を上向きに取り付け、これに別に製作した転送ボードを挿すという形をとる。

以上の仕様に基づいて、4 月～5 月にかけて電子回路の設計・製作を行った。



ゼミ用 TRON 搭載マイクロマウスの設計図（CPU とモータ）



ゼミ用 TRON 搭載マイクロマウスの設計図（センサ）

マイクロマウス 2002 でセンサが電力を食いすぎ、三端子レギュレータが焼けてしまった体験から、このマシンではパルス出力により省電力を実現している。また、レギュレータもすぐには焼けないものを採用した。

外部 RAM の使用については、マイクロマウス東日本支部のものを参考にした。
マイクロマウス委員会東日本支部講習資料　http://www.fan.hi-ho.ne.jp/ogawa/h8/
ハイパーターミナルは「チェックをしない」にすると、rh.mot を動かすことができる。（rh.mot を ROM に焼いておくと､外部 RAM 上でプログラムを動かすことができるため､いちいち ROM 焼きを行わなくてすむ）
GCC Developer Lite を使った場合の外部 RAM 使用プログラムの使い方

1）ツールバーのツール→Gcc オプション→設定リストから h8/3048F　増設 SRAM（ターミナル要）を選びコンパイルする。
2）フラッシュライタで RH.MOT を ROM に入れる
3）SIMPLE TERM を起動し、通信速度を 38400bps、データを 8 ビット、ストップビットを１ビット、パリティをなし、フロー制御をなしにする。
4）ポートをオープンし、マイクロマウスの電源を入れると何か文字が表示される。
5）その後すぐにスペースキーを押すとファイルの転送ができる。そしてコンパイルしてできた～.bin ファイルを転送する。すると転送し終わると同時に実行される。

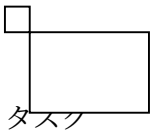
これで ROM に転送用のプログラムを入れ､それ経由で増設 RAM にプログラムを入れることができる。これをすることによって、ROM の書き込み制限（メーカーが保証するのは書き込みが 100 回まで）に縛られることがなくなった。

※日立製モニタデバッガを利用すると、メモリ増設がうまくいっていることを確認できる。モニタデバッガを ROM に入れると、PC から RS-232C 経由で特定の番地を指定して中身

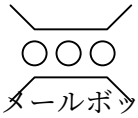
を見たり、数値を入力したりすることができる。設計したときに考えたとおり、H'20000 番地からの領域（増設 RAM 領域）に書き込み・読み出しを行うことができた。ただし、モニタデバッガは日立半導体グループがルネサスに名称変更してから配布されなくなってしまったようである。

付録　タスク設計に用いる記号

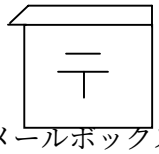
今後、よく使うであろう記号のテンプレートを以下にあげておく。



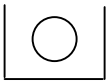
タスク



メールボックス



メールボックス



セマフォ

割込み

タイマ

これ以外は必要になったら適宜追加する。

http://www.linux.or.jp/JF/JFdocs/RTLinux-HOWTO.html

３．開発環境の整備

３．１．μITRON の選定

μITRON 仕様 OS にもさまざまなものがある。製品版は非常に高価であるので、フリーのものを用いる。フリーのμITRON 仕様 OS には以下のようなものがある。

・ TOPPERS
・ HOS

今回は実績の多い TOPPERS を用いることに決定した。

３．２．クロス開発環境

パソコン上でプログラム開発を行い、完成したものをマイコンに載せかえるという、クロス開発が主流である。環境には以下のようなものが考えられる。

①Windows　cygwin ベストテクノロジー社　GCC Developer Lite

直感的な操作で cygwin のことをまったく意識せずに gcc によるプログラム開発ができる。そのかわり、分割コンパイルができない、細かい設定ができないといった問題がある。しかし、さまざまな CPU・環境（増設 RAM あり・なしなど）に対応しており、非常に便利である。CINCS では長い使用実績がある。

②Windows　秋月コンパイラ

有償（ただし安価)。日立純正のコンパイラに近いものであり、たいていのことができるが、スタートアップルーチンを自分で書かねばならないなど、初心者には敷居が高い。

③Windows　cygwin　gcc

gcc を直接使用する。この環境が TOPPERS では推奨とされている。

④UNIX 系 OS　gcc

シミュレータなどが Windows 向けのものが多いため、不便なこともある。FreeBSDPress14 号に FreeBSD 上でクロス開発環境を作るための記事がある。

試行錯誤を繰り返した結果、③でうまくいく方法を確立できた。

３．３．環境の整備

通常の Windows プログラムと同様に自動インストールができるため、cygwin 自体のインストールは失敗することはない。

用語についてのメモ　http://www.zdnet.co.jp/help/tips/linux/l0302.html

「configure」は，これからインストールを行う OS の CPU やバージョンを調べたり，必要な関連ツールを調査する。そして，これらの状況を記述した Makefile ファイルを作成する。

「make」は，configure で作成された Makefile を基にしてソースコードをコンパイルする。ここでエラーが起こる場合には，configure で環境に沿ったパラメータを付加させなければならない。

「make install」は，make で生成されたバイナリファイルなどを規定のディレクトリにコピー（インストール）を行う。インストール先のディレクトリは，自分で設定したい場合には configure で「--prefix=/usr/xxxxxx」などとパラメータを付加させる必要がある。

３．３．１．Cygwin 環境構築（Administrator 権限が必要）

１．Cygwin を入れる。
- Install for を All Users にする
- Default Text File Type を Unix にする
- SelectPackages で All の横の Default をクリックし Install にする（時間がかかり応答なしになることもある）

２．準備
　以下のソースファイルを用意する。

アセンブラ・リンカ
　binutils-2.10.tar.gz
　　http://sources.redhat.com/binutils/
　　ftp://ftp.ring.gr.jp/pub/GNU/binutils
C コンパイラ
　gcc-2.95.2.tar.gz　　ftp://ftp.ring.gr.jp/pub/GNU/gcc/

標準 C ライブラリ
　newlib-1.8.2.tar.gz　　http://sources.redhad.com/pub/newlib/

３．binutils の作成
ソースを解凍
cd /usr/local/src
tar xzvf binutis-2.10.tar.gz

作業用ディレクトリ作成
mkdir binutis-2.10/work
cd binutils-2.10/work

コンフィグレーション
./configure --target=h8300-hms-coff --prefix=/usr/local/h8

コンパイル・インストール
make
make install

ここでインストールされたツールを実行できるようにするためにパスを通す。
PATH=$PATH:/usr/local/h8/bin

４．gcc の作成（http://www.ertl.ics.tut.ac.jp/%7Emuranaka/devel/h8-gcc.txt
　より）

newlib のヘッダーファイルを使用するので解凍する
cd /usr/local/src
tar xzvf newlib-1.8.2.tar.gz

gcc をコンパイル
tar xzvf gcc-2.95.2.tar.gz
mkdir gcc-2.95.2/work
cd gcc-2.95.2/work
../configure --target=h8300-hms-coff --prefix=/usr/local/h8 --with-newlib --wi
th-headers=/usr/local/src/newlib-1.8.2/newlib/libc/include
make LANGUAGES="c"
make install LANGUAGES="c"

５．newlib の作成

```
# cd /usr/local/src/newlib-1.8.2
# mkdir work
# cd work
# ../configure --target=h8300-hms-coff --prefix=/usr/local/h8
# make
# make install
```

以上で以下の H8 プロセッサ用のツールが使用できるようになる

h8300-hms-coff-as	アセンブラ
h8300-hms-coff-ld	リンカ
h8300-hms-coff-gcc	C コンパイラ
h8300-hms-coff-objcopy	オブジェクトファイル変換ツール
h8300-hms-coff-objdump	逆アセンブラ
h8300-hms-coff-nm	シンボル名表示
h8300-hms-coff-size	プログラムサイズ表示

短い名前で使用できるようにシンボリックリンクを張る

```
# cd /usr/local/bin
# ln -s /usr/local/h8/bin/h8300-hms-coff-as h8-as
# ln -s /usr/local/h8/bin/h8300-hms-coff-ld h8-ld
# ln -s /usr/local/h8/bin/h8300-hms-coff-gcc h8-gcc
# ln -s /usr/local/h8/bin/h8300-hms-coff-objcopy h8-objcopy
# ln -s /usr/local/h8/bin/h8300-hms-coff-objdump h8-objdump
# ln -s /usr/local/h8/bin/h8300-hms-coff-nm h8-nm
# ln -s /usr/local/h8/bin/h8300-hms-coff-size h8-size
```

これで各ツールが h8-xxxx で使用できる。

インストール後は /usr/local/src 以下は消しても良い。

６．使い方

[リンカースクリプトの編集]
ターゲットボードのメモリ配置に合うように ldscript を編集。また、ROM 化に対応させる。
オリジナルは /usr/local/h8/h8300-hms-coff/lib/ldscript/h8300h.x にあるのでこれを元にして編集する。(サンプル参照)

[スタートアップルーチン]
ROM 化に対応するためのスタートアップルーチンを作成。
オリジナルは /usr/local/src/newlib-1.8.2/newlib/libc/sys/h8300hms/crt0.S にあるのでこれを元にする。

スタートアップルーチンではスタックポインタの初期化、ハードウェア初期化(メモリ領域、メモリアクセスウェイト設定等)、初期化データ領域(DATA セクション)の初期化、非初期化データ領域(BSS セクション)の 0 初期化を行った後 main() 関数を呼び出す。
(サンプル参照)

[newlib 用グルーチン] (cygwin の場合)
newlib のターゲット依存部を作成する。
_read(), _wirte() にシリアル 1 文字入出力ルーチンを書くと printf(), scanf(), puts(), gets()等でシリアル経由の入出力が出来るようになる。ただし、printf()、scanf()を使用するとファイルサイズが膨大になるので要注意。
(サンプルの glue_newlib.c 参照)

[コンパイル方法]
```
# h8-gcc -T  -nostartfiles -mh -mrelax -O2 -o      ...
```

オプションの意味
-T : リンカスクリプトファイルの指定
-nostartfiles : デフォルトのスタートアップルーチンを使用しない
-mh : H8/300H を指定
-mrelax : 絶対アドレッシングを最適化
-O2 : 最適化レベル 2
-o : COFF 形式出力ファイル名
: 使用するスタートアップルーチン
: C 言語、アセンブラソースファイルまたはオブジェクトファイル

[バイナリ出力ファイル変換]
COFF 形式のバイナリファイルをモトローラ S レコードフォーマットファイル(S2)に変換する
```
# h8-objcopy -O srec
```
-O srec : モトローラ S レコードフォーマットを指定
: COFF 形式入力ファイル名
: S フォーマットファイル出力ファイル名

※ binutils-2.10 では S2 レコードを出力する(binutils-2.9.1 以前では S3)

Cygwin で X を使う
日本語を入力するには、［Ctrl］＋［¥］キーを押します。
日本語化の仕方が書いてあるページ
http://www.atmarkit.co.jp/flinux/special/cygwin2/cygwin01a.html
バイナリ等が置いてあるページ
http://www.on.ics.keio.ac.jp/%7Emaru/cygwin-xfree-jp-supplement/?05171600

※BINUTILS の configure ファイルから Make ファイルを作るときに cc がないというエラーとなったなので
```
alias gcc='cc'
```
と打ってからやるとエラーなく進めた。

３．４．コンパイルとリンク

３．４．１．メモリ領域の割り付け

プログラムを ROM 化してマイコンに書き込む場合、メモリ領域の使用法を考慮したうえで行わなければならない。

	名称	ｾｸｼｮﾝ名 下：gcc	書込み操作	初期値の有無	割り付け
1	プログラム領域	P .text	不可	有	ROM 領域に配置する
2	定数領域	C .tors	不可	有	ROM 領域に配置する
3	初期化データ領域	D .data	可	有	初期値は ROM 領域に配置し、実行時は RAM 領域を参照する
4	未初期化データ領域	B .bss	可	無	RAM 領域に配置する
5	スタック領域	（なし） .stack	可	無	RAM 領域に配置する

１．プログラム領域

２．定数領域

初期値があり書込み操作がない。リンケージの際に ROM 領域を指定する。

３．初期化データ領域

初期値があり書込み操作もある。リンケージの際は RAM 領域を指定し、ロードモジュール作成後に ROM ライタ上で初期値を ROM 領域に移動して ROM 化する必要がある。さらに、プログラム実行開始時に初期値を ROM 領域から RAM 領域にコピーしなければならない。

４．未初期化データ領域

初期値がなく書込み操作がある。リンケージの際は RAM 領域を指定する。C 言語の仕様上初期値はゼロとされているため、プログラム実行開始時にゼロクリアしなければならない。

「(i)変数に記憶場所を割りあてたときのみ、変数は初期化される。
(ii)明示的に初期化をしなかったとき、大域あるいは静的な変数にはすべて初期値 0 が割りあてられる。」（「プログラマのための ANSI C 全書」より）

具体的には、グローバル変数や static つきの変数が静的な変数にあたる。通常の（静的でない）変数（auto 変数）はスタックに置かれる。

５．スタック領域

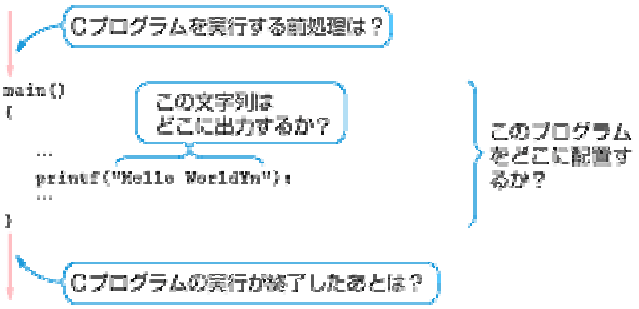
初期値がなく書込み操作がある。リンケージの際は RAM 領域を指定する。

リンクのときはスタートアップルーチン（拡張子 S）、rom や ram のセクションの設定を行うファイル（リンカスクリプト、拡張子 x）をプログラムファイル（拡張子 c）・ヘッダファイル（拡張子 h）とともに準備しておかねばならない。デフォルトでは rom プログラムは h'00000 番地から、vector（割り込みルーチンの設定を書く場所）は h'00100 番地まで等となっている。GCC Developer Lite の場合、利用者の手間を省くため h8ram.x（外部 RAM 使用のプログラム用）や h8rom.x（内蔵 ROM 使用のプログラム用）といったリンカスクリプトがあらかじめ準備してあるが、必要があれば自分で書き直さなければならない。

３．４．２．スタートアップルーチン

スタートアップルーチンは起動時の動作を書いたもので、普通はアセンブラで書かれる。具体的には「初期化データ領域のデータを rom から ram に転送する」「未初期化データ領域をゼロクリアする」「メイン関数にジャンプする」という動作を行う。

「しかし，組み込み用コンパイラではそうはいきません．図 4 に示すように，C プログラムの最初の関数を実行する前にやるべきことがあります．また，プログラムの実行を終了した場合，戻る OS が存在しないので，何らかの処理が必要でしょう．図 4 では printf 関数が使われていますが，この文字列はどこへ出力すべきなのでしょうか．そして，このプログラムは，メモリ上のどのアドレスに配置すべきかも検討する必要があります．



● スタートアップルーチン

スタートアップルーチンは, C 言語で記述したアプリケーションプログラムを実行する前に実行されるプログラムです．詳細は第 3 章で説明しますが，スタートアップルーチンでは，スタックの初期設定やメモリの初期化，その他の環境設定を行った後，C プログラムの最初の関数を呼び出します．

PC 用のアプリケーションプログラムでは，スタートアップルーチンを意識することはほとんどありません．C プログラムを書いてコンパイルさえすれば，そのプログラムは動作します．理由は，コンパイラが自動的にスタートアップルーチンをリンクするためです．

しかし，組み込みシステムの場合，システムは PC のように画一的ではありません．同じ CPU を使う場合でも，あるシステムには SDRAM が実装されていたり，また別のシステムでは液晶パネルが搭載されていたりと，用途により多種多様です．

つまり，C プログラム実行の前に，初期化しておかなければならないデバイスがあるのです．そのため，コンパイラからは基本となるスタートアップルーチンのサンプルのみ提供され，システム固有の初期設定は，プログラマ自身が責任をもって記述する必要があります．

また，PC 用コンパイラの場合，C プログラムで最初に実行される関数は main に固定されています. これは PC 用コンパイラに付属しているスタートアップルーチンから最初に呼ばれる C プログラムの関数が main 関数になっているからです．そのため，PC 用コンパイラの場合，main 関数を書かずにプログラムをコンパイルすると，リンク時に main 関数が未定義である旨のワーニングまたはエラーが発生します．

これに対して組み込みシステムの場合，スタートアップルーチンはプログラマが記述するため，C プログラムの最初の関数は自由に定義できます．

C プログラムから戻ってきた場合はどう処理すればよいのでしょうか. 通常は，無限ループや CPU を停止する命令を埋め込んでおきます.」

（以上　http://www.cqpub.co.jp/interface/toku/2002/200203/toku1_4.htm より）

※スタートアップルーチンの例（gcc developer lite 付属の h8crt0.S）
gas（GNU アセンブラ）で書かれている。Gas のセクション名については
http://www.bekkoame.ne.jp/~bero/gnudocj/binutil/gas/as_4.htm

に書かれている。

```
; h8/300 and h8/300h start up file.

        .h8300h         ;h8-300h を使っていることを宣言する
        .section .text  ;text セクションの宣言
        .global  _start ; 外部ラベル宣言
_start: ;ここが H'0 番地となる
        mov.l   #_stack,sp      ;スタックポインタを決める？

        ;; copy DATA from ROM   ROM から RAM にデータを転送する
        mov.l   #___dtors_end,er0
        mov.l   #_edata,er1
        mov.l   #___data,er2
.loop0: mov.w   @er0,r3
        mov.w   r3,@er2
        adds    #2,er0
        adds    #2,er2
        cmp.l   er1,er2
        blo     .loop0

        ;; clear BSS       BSS 領域をゼロクリアする
        mov.l   #_edata,er0
        mov.l   #_end,er1
        mov.w   #0,r2           ; not sure about alignment requirements
.loop:  mov.w   r2,@er0         ; playing it safe for now
        adds    #2,er0
        cmp.l   er1,er0
        blo     .loop
        jsr     @_main          ;main()関数にジャンプする
        jsr     _start          ;はじめに戻る（無限ループ）

        .section .stack
_stack: .long   1
```

自作する場合はこれを元に作成する。

スタートアップルーチンではスタックポインタの初期化、ハードウェア初期化(メモリ領域、メモリアクセスウェイト設定等)、初期化データ領域(DATA セクション)の初期化、非初期化データ領域(BSS セクション)の 0 初期化を行った後 main() 関数を呼び出す。

３．４．３．リンカスクリプト

リンカスクリプトの書き方については
http://www.semicon.panasonic.co.jp/micom/bs_part/gnu/linker-scripts.pdf
に詳細な解説があるため、それを参照すること。
以下　http://www.sra.co.jp/public/sra/product/wingnut/ld/ld-ja.html#SEC_Top
のサンプルに注を加えたもの。

```
OUTPUT_FORMAT("coff-h8300")
OUTPUT_ARCH(h8300h)
```

```
ENTRY("_start")
MEMORY {
        /* 0xc4 is a magic entry.  We should have the linker just
           skip over it one day... */

        vectors : org =0x00000, len =0x100   /*ベクタテーブルは 0x00000 番地から 0x100 番地分である、つまり 0x00000 番地から 0x000ff 番地までである*/
        rom     : org =0x00100, len =128k-0x100         /*rom 領域は 0x00100 番地から 128k-0x100 番地分である、つまり 0x00100 番地から 0x1ffff 番地までである*/
        ram     : org =0xfef10, len =4k

        /* The stack starts at the top of main ram.  */
        topram  : org =0xffefc, len =0x4
        /* At the very to of the address space is the 8-bit area.  */
        eight   : org =0xfff00, len =0x100
}
SECTIONS {
        .vectors : {    /*ベクタテーブルの設定　一行ごとに割り込み命令のアドレスが書いてある*/
          /*LONG(ABSOLUTE(_start))とはスタートアップルーチンのラベル_start の絶対アドレスのことである。*/
          /* LONG(DEFINED(_int_nmi)?ABSOLUTE(_int_nmi):ABSOLUTE(_start))　これは三項演算子で書いてある。「もし_int_nmi が定義されていたら、_int_nmi の絶対アドレス、そうでなかったら_start の絶対アドレスをここに書く」つまり、「C 言語のソースで int_nmi()が書かれていたら int_nmi()のアドレスを、そうでなかったらスタートアップルーチンのアドレスをここに書く」つまり、「nmi 割り込みが入ったとき、割り込み関数があったらそれを実行し、なかったら何もしない」*/
                /* Use something like this to place a specific function's address
                   into the vector table.  */
                /* 0x00 */
                LONG(ABSOLUTE(_start))
                LONG(ABSOLUTE(_start))
                LONG(ABSOLUTE(_start))
                LONG(ABSOLUTE(_start))
                /* 0x10 */
                LONG(ABSOLUTE(_start))
                LONG(ABSOLUTE(_start))
                LONG(ABSOLUTE(_start))
                LONG(DEFINED(_int_nmi)?ABSOLUTE(_int_nmi):ABSOLUTE(_start))
                /* 0x20 */
                LONG(DEFINED(_int_trap0)?ABSOLUTE(_int_trap0):ABSOLUTE(_start))
                LONG(DEFINED(_int_trap1)?ABSOLUTE(_int_trap1):ABSOLUTE(_start))
                LONG(DEFINED(_int_trap2)?ABSOLUTE(_int_trap2):ABSOLUTE(_start))
                LONG(DEFINED(_int_trap3)?ABSOLUTE(_int_trap3):ABSOLUTE(_start))
                /* 0x30 */
                LONG(DEFINED(_int_irq0)?ABSOLUTE(_int_irq0):ABSOLUTE(_start))
                LONG(DEFINED(_int_irq1)?ABSOLUTE(_int_irq1):ABSOLUTE(_start))
                LONG(DEFINED(_int_irq2)?ABSOLUTE(_int_irq2):ABSOLUTE(_start))
                LONG(DEFINED(_int_irq3)?ABSOLUTE(_int_irq3):ABSOLUTE(_start))
                /* 0x40 */
```

```
LONG(DEFINED(_int_irq4)?ABSOLUTE(_int_irq4):ABSOLUTE(_start))
LONG(DEFINED(_int_irq5)?ABSOLUTE(_int_irq5):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0x50 */
LONG(DEFINED(_int_wovi)?ABSOLUTE(_int_wovi):ABSOLUTE(_start))
LONG(DEFINED(_int_cmi)?ABSOLUTE(_int_cmi):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0x60 */
LONG(DEFINED(_int_imia0)?ABSOLUTE(_int_imia0):ABSOLUTE(_start))
LONG(DEFINED(_int_imib0)?ABSOLUTE(_int_imib0):ABSOLUTE(_start))
LONG(DEFINED(_int_ovi0)?ABSOLUTE(_int_ovi0):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0x70 */
LONG(DEFINED(_int_imia1)?ABSOLUTE(_int_imia1):ABSOLUTE(_start))
LONG(DEFINED(_int_imib1)?ABSOLUTE(_int_imib1):ABSOLUTE(_start))
LONG(DEFINED(_int_ovi1)?ABSOLUTE(_int_ovi1):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0x80 */
LONG(DEFINED(_int_imia2)?ABSOLUTE(_int_imia2):ABSOLUTE(_start))
LONG(DEFINED(_int_imib2)?ABSOLUTE(_int_imib2):ABSOLUTE(_start))
LONG(DEFINED(_int_ovi2)?ABSOLUTE(_int_ovi2):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0x90 */
LONG(DEFINED(_int_imia3)?ABSOLUTE(_int_imia3):ABSOLUTE(_start))
LONG(DEFINED(_int_imib3)?ABSOLUTE(_int_imib3):ABSOLUTE(_start))
LONG(DEFINED(_int_ovi3)?ABSOLUTE(_int_ovi3):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0xA0 */
LONG(DEFINED(_int_imia4)?ABSOLUTE(_int_imia4):ABSOLUTE(_start))
LONG(DEFINED(_int_imib4)?ABSOLUTE(_int_imib4):ABSOLUTE(_start))
LONG(DEFINED(_int_ovi4)?ABSOLUTE(_int_ovi4):ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0xB0 */
LONG(DEFINED(_int_dend0a)?ABSOLUTE(_int_dend0a):ABSOLUTE(_start))
LONG(DEFINED(_int_dend0b)?ABSOLUTE(_int_dend0b):ABSOLUTE(_start))
LONG(DEFINED(_int_dend1a)?ABSOLUTE(_int_dend1a):ABSOLUTE(_start))
LONG(DEFINED(_int_dend1b)?ABSOLUTE(_int_dend1b):ABSOLUTE(_start))
/* 0xC0 */
LONG(ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
LONG(ABSOLUTE(_start))
/* 0xD0 */
LONG(DEFINED(_int_eri0)?ABSOLUTE(_int_eri0):ABSOLUTE(_start))
LONG(DEFINED(_int_rxi0)?ABSOLUTE(_int_rxi0):ABSOLUTE(_start))
LONG(DEFINED(_int_txi0)?ABSOLUTE(_int_txi0):ABSOLUTE(_start))
```

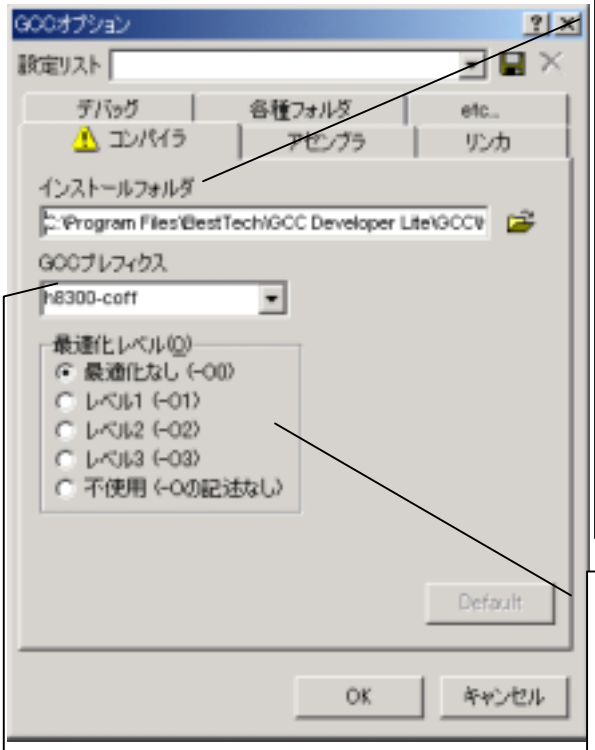
```
                LONG(DEFINED(_int_tei0)?ABSOLUTE(_int_tei0):ABSOLUTE(_start))
                /* 0xE0 */
                LONG(DEFINED(_int_eri1)?ABSOLUTE(_int_eri1):ABSOLUTE(_start))
                LONG(DEFINED(_int_rxi1)?ABSOLUTE(_int_rxi1):ABSOLUTE(_start))
                LONG(DEFINED(_int_txi1)?ABSOLUTE(_int_txi1):ABSOLUTE(_start))
                LONG(DEFINED(_int_tei1)?ABSOLUTE(_int_tei1):ABSOLUTE(_start))
                /* 0xF0 */
                LONG(DEFINED(_int_adi)?ABSOLUTE(_int_adi):ABSOLUTE(_start))

                FILL(0xff)
                }   > vectors
・*以下、セグメントの定義*・
        .text : {
                *(.text)
                *(.strings)
                *(.rodata)
                 _etext = . ;
                } > rom
        .tors : {
                ___ctors = . ;
                *(.ctors)
                ___ctors_end = . ;
                ___dtors = . ;
                *(.dtors)
                ___dtors_end = . ;
                }   > rom
        .data : AT ( ADDR(.tors) + SIZEOF(.tors) ){
                ___data = . ;
                *(.data)
                *(.tiny)
                 _edata = . ;
                } > ram
        .bss : {
                 _bss_start = . ;
                *(.bss)
                *(COMMON)
                 _end = . ;
                }   >ram
        .stack : {
                 _stack = . ;
                *(.stack)
                }   > topram
        .eight : {
                *(.eight)
                }   > eight
        .stab 0 (NOLOAD) : {
                [ .stab ]
                }
```

```
        .stabstr 0 (NOLOAD) : {
                [ .stabstr ]
                }
}
```

３．４．５．コンパイラオプション（GCC Developer Lite の場合）

ここでは GCC Developer Lite のヘルプを見やすく書き直したものをあげておく。本格的な gcc のオプションについてはマニュアルを参照すること。

コンパイラタブ



■インストールフォルダ(I)
GNUPro Toolkit のインストールされた中に GNU ツールが入っているフォルダ「bin」があります。このフォルダをフルパスで指定してください。

GNUPro Toolkit はデフォルトで「C:¥cygnus」へインストールされます。通常は「C:¥cygnus¥gnupro-99r1p1¥H-i686-cygwin32¥bin」を指定してください。

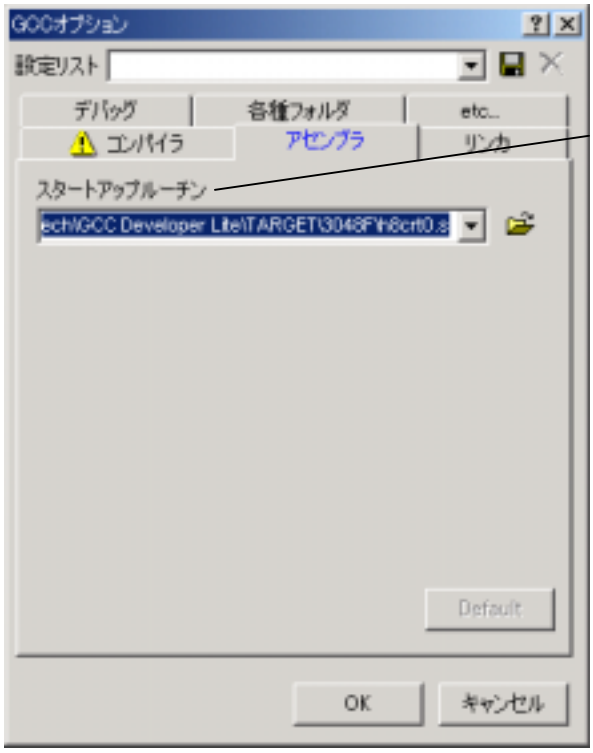
■最適化レベル(O)
プログラムの最適化レベルを指定します。
-O0　最適化を行いません。
-O1　最適化を行います。「-O」が指定されると、コンパイラは、コードのサイズと実行時間を削減するよう試みます。
-O2　さらに最適化を行います。サポートされている最適化のうち、サイズとスピードとのトレードオフを伴わないほとんどすべてのものを実行します。
-O3　さらに一層、最適化を行います。「-O3」は「-O2」により指定されるすべての最適化を有効にした上に、さらに「inline-functions」オプションも有効にします。
※　最適化の原理と出力コードの関係が把握できるまで「-O0」の指定をお勧めします。

■GCC プレフィクス(P)
GNUPro Toolkit のツール名に付加されるプレフィクスを指定します。

GNU ツールは通常下記の名称を使います。
GNU CC...　gcc
GNU Assembler....　as
GNU Linker...　ld

GNUPro Toolkit のばあい、これらの名称の前にターゲット CPU 別の識別文字列が付加されます。たとえば gcc を例にあげると、日立 H8/300 シリーズの場合は「h8300-hms-gcc」
この「-gcc」より前に付加される文字列を指定してください。

アセンブラタブ



■スタートアップルーチン(S)
標準システム・スタートアップ・ファイル以外のスタートアップルーチンを記述したアセンブリファイルもしくはアセンブル済みオブジェクトファイルを指定します。

スタートアップルーチンとは、ターゲット CPU 特有の初期化処理や使用されるデータエリアのクリア等の処理を行う部分を言います。
独自の処理をアセンブラレベルで記述したい場合に指定してください。

コマンドラインでコンパイルするときのオプションのつけ方

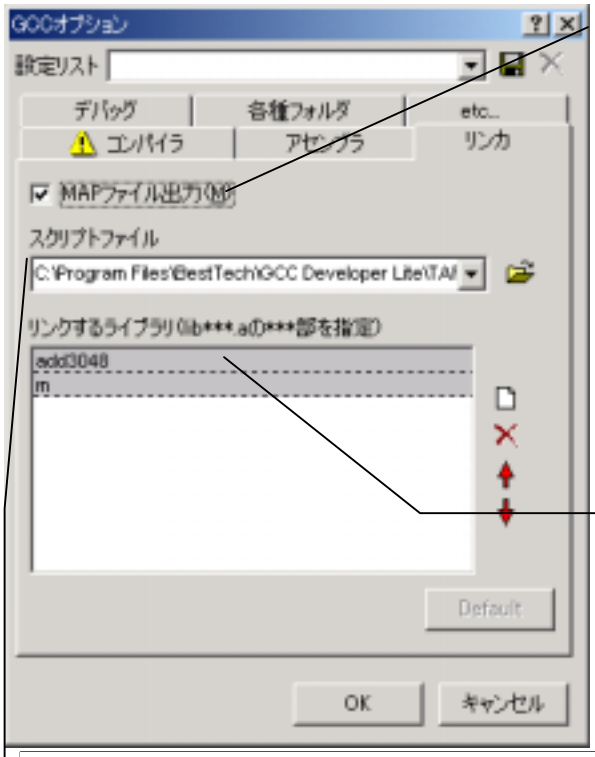
gcc -nostartfiles <startup-file>

-nostartfiles　　：デフォルトのスタートアップルーチンを使用しない

<startup-file>　　：使用するスタートアップファイル

現在は h8crt0.s を使っている。

リンカタブ



■マップファイル出力(M)
チェックされているとリンクマップファイルを作成します。

リンクマップには、オブジェクトやライブラリに含まれる関数やデータのメモリレイアウトが保存されます。
特に組み込みシステムにおけるメモリマップ情報はデバッグに有用です。

■リンクするライブラリ(L)
リンクするライブラリを指定します。

基本的にライブラリは GNUPro Toolkit が提供するものを指定します。
ただし指定する名前は、ライブラリファイル名が「libabc.a」という名称の場合は先頭の「lib」と拡張子「.a」を除いた「abc」のみとなることに注意してください。

また、GNUPro Toolkit で提供される標準ライブラリ以外のライブラリファイルを指定する場合は、各種フォルダの「追加ライブラリフォルダ」で指定したフォルダから検索されます。

■スクリプトファイル(S)
リンク用スクリプトファイルを指定します。
リンカスクリプトファイルは、プログラム上で使用されるセクションの確保と配置イメージを宣言したものです。
OS が管理する上で動作するプログラムではそれほどメモリの配置を意識することはありませんが、組み込みシステム等における RAM や ROM といったメモリの制約や、割込みベクタといった特殊なメモリエリアを正確に管理する必要があります。

スクリプトファイルには

内蔵 FLASHROM に書き込む時には　h8rom.x　を使用し
増設 SRAM（ターミナル要）に書き込むときには　h8ram.x を使用する。
これは設定リストを変えることにより自動的に変わる。

コマンドラインでコンパイルするときのオプションのつけ方

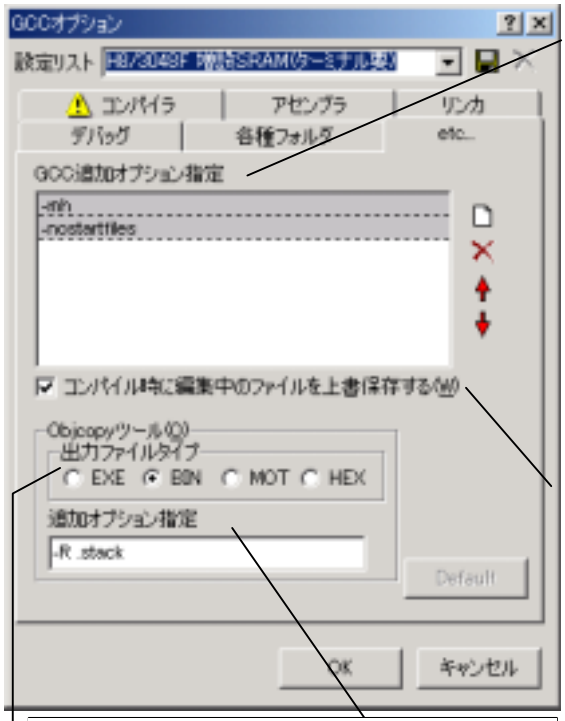
gcc -T <linker-script>
-T <linker-script>：リンカスクリプトファイルの指定

各種フォルダ

■追加インクルードフォルダ
ソースファイルで使用されるヘッダファイルを検索するフォルダを指定します。

標準ヘッダファイルの場所は指定する必要はありません。また、存在しないフォルダは指定できません。

■追加ライブラリフォルダ
リンカオプションで指定されたライブラリファイルを検索するフォルダを指定します。
標準ライブラリの場所は指定する必要はありません。また、存在しないフォルダは指定できません。



■GCC 追加オプション指定(A)
gcc へ与えるオプションを指定します。
この GCC Developer ではファイルやフォルダに関するコンパイラオプションを生成することが目的ですので、実際には多くのオプション(ターゲット CPU 毎に異なる設定や細かい最適化設定等)を別途指定する必要があります。

■コンパイル時に編集中のファイルを上書き保存する
コンパイル時に編集中のソースファイルを上書き保存した上で gcc を起動します。
新規作成されたソースのばあいは、コンパイル時にファイル保存のダイアログボックスが現れます。
また、このオプションを指定しておかないと編集したソースの内容が適用されない状態でコンパイルされてしまいます。

■追加オプション指定
objcopy 実行時のオプション指定を追加します。
出力ファイルタイプ(-O)以外のオプションを指定してください。

■Objcopy 出力ファイルタイプ
gcc にてコンパイルされたプログラムの変換を行うファイル形式を一つ選択します。
EXE：gcc がデフォルトで出力するファイル形式(拡張子が「.exe」)そのもので、objcopy にて変換は行わない。デバッガ環境等で使用できる形式。
BIN：gcc がデフォルトで出力する exe ファイルを、メモリイメージのバイナリファイル形式(拡張子が「.bin」)に Objcopy ツールにて変換(オプション「-O binary」を使用)する。
MOT：gcc がデフォルトで出力する exe ファイルを、インテル S レコードファイル形式(拡張子が「.mot」)に変換(オプション「-O srec」を使用)する。

コマンドラインでコンパイルするときのオプションのつけ方

-mh：H8/300H を指定
-mrelax：絶対アドレッシングを最適化

設定リストについて

RH.MOT などの ROM モニタを ROM に入れて、増設 RAM 上から作ったプログラムを実行するときに選ぶ。

作ったプログラムを普通に ROM から実行する時に使う。

３．４．６．TOPPERS

http://www.mit.pref.miyagi.jp/embedded/TOPPERS/　：豊橋技術科学大学 情報工学系 組込みリアルタイムシステム研究室

http://www.tron.org/　　：TRON 協会

http://www.interface.co.jp/cpu/sh_howto/sh_howto10.asp

うまくいかないため､7月1日に「Free BSD Press」14号の記事「H８システム用のITRON開発環境」に書いてあるとおりの環境作りをしてみた。

筆者は TOPPERS のプログラムを FreeBSD 上で開発しているらしく、記事は以下のような内容となっている。

１．ITRON と TOPPERS

１．１．ITRON 概要

１．２．TOPPERS/JSP

２．TOPPERS/JSP 開発環境の構築

２．１．クロス開発環境の準備と STL ライブラリのインストール

２．２．binutils のファイルのコピー

２．３．TOPPERS/JSP 開発環境のインストールとサンプルプログラムのコンパイル

３．まとめ

まず E 教室のマシンに FreeBSD をインストールし､記事にしたがってソフトのインストールを行った。FreeBSD を扱ったことのあるものがいなかったため、書いてあることにほぼ忠実に行うことにした｡FreeBSD(4.７patch)はFreeBSDマガジン14号の付録CD-ROMからインストールした。

（記事２．１による）

構築する TOPPERS/JSP 開発環境は以下のとおりである。

○ターゲット（H8/8300）用クロス開発環境

・ binutils：2.13
・ gcc：3.2
・ newlib：1.10

○ホスト環境　（FreeBSD　4.7R）

・ gcc：2.95.4
・ gnu make：3.79（記事では 3.80 だったが、見つからなかったため 3.79 とした）
・ perl：5.005_3
・ STL：4.5.3（ports から devel/stlport をインストール）

FreeBSD では web 上からファイルを直接取ってきてインストールをすることができる。/stand/sysinstall コマンドで選択画面になるので、ここで FTP からインストールを選ぶ。するとサーバを選ぶ画面になるので、適当に日本のサーバを選ぶ。ここで入れたいパッケ

ージを探し、「インストール」を選ぶと、自動的にインストールが行われる。
　gcc のためのパッチ「h8300-hms-gcc-3.1-1.patch」は http://h8300-hms.sourceforge.net/ から入手するとのことだったが、うまくパッチを当てることができなかったのでこれは省略した。
　newlib-1.10.tar.gz は ftp://source.redhat.com からダウンロードするとのことだったが、これはサーバがなくなっていたため別のところから探してきた。
　それ以外のファイルは GNU のサイトから入手した。
　P142 のインストール手順に従ってインストールした。シェルが csh だったので指示に従ってそれ用のコマンドを用いた。
　（記事２．２による）
　次に binutils のファイルを/＜H8/8300 の開発環境をインストールしたディレクトリ＞/h8300-hms/bfd にコピーした。具体的には p143 のとおりである。ただし
　libbfd.h は存在しなかったので libbfd.a を、libintl.a は存在しなかったので libintl.h をそれぞれコピーした。

どうしても make depend の後の make がうまくいかない。調べたところ
http://www.mit.pref.miyagi.jp/embedded/TOPPERS/
で、以下の説明を見つけた。

・make 時にエラーになり、makefile.depend を確認したところ<command line>と<built-in>が残っているようです。

Release1.3 では make depend の時に呼び出している perl スクリプトの処理が gcc3.x 系のプリプロセッサの出力フォーマットに対応していません。
・/utils/makedep の書き換え
　１２４行目を次のように変更します。

変更前
<　　if ($line =~ /^¥#¥s*([0-9]+)¥s*¥"([^¥"]+)¥"/) {

変更後
>　　if ($line =~ /^¥#¥s*([0-9]+)¥s*¥"([^¥"¥<¥>]+)¥"/) {

これを修正してもう一度 make すると目的のファイルができた。

しかし、実際に ROM に焼いて実行してみたが何も起こらなかった。サンプルプログラムが文字を返すプログラムなので表示するところが必要だが通信速度やパリティの有り無しなどがわからずモニタデバッガも何か特別なものを使わないといけないのかなどがわからずそのまま終わった。（2003 年 7 月末時点）

とりあえずここでおいておき、シミュレータの使い方を調べることにした。

３．４．７．TOPPERS シミュレータ

TOPPERS とともに提供されており、使用方法は添付文書にも載っているが、わかりづらいため、「Design Wave」2003 年 4 月号のサンプルを見るのがよい。VB のプログラムでデバイスのエミュレートを行い、VC++でコンパイルしたプログラムのシミュレートをすることができる。処理のほとんどはスクリプトで自動化されており、雑誌のサンプルを実行する場合問題は出なかった。

OS があると処理中に割り込んだ場合にちゃんと割り込みを処理していたが、OS 無しだと優先度の低い処理を実行中に優先度の高い処理がきた場合にちゃんと割り込んだものが実行されない、ということが実感できる。

なお、μITRON の感覚をつかむにはμITRON トレーナーというエミュレータもある。（これは TOPPERS とは直接関係がない）

まとめ

前期の研究はここまでである。今後は

- 実機でも動くプログラムの作成方法の確立
- 実際のプログラム開発
- OS 作成のための準備

といったことを進めていく必要がある。

※ 今までの参考文献

H8

- H8S,H8/300H シリーズ C 言語コーステキスト（日立半導体セミナー）
- H8 マイコン完全マニュアル（藤沢幸穂、オーム社）
- 自立型ロボット製作バイブル（西山一郎ほか、オーム社）
- ‘99･’00 年技術交流の記録（マイクロマウス委員会東日本支部）
- 初心者のためのロボットの作り方第六版（マイクロマウス委員会東日本支部）
- マイコン技術教科書 H8 編（今野金顕、CQ 出版社）
- H8 ビギナーズガイド（電気通信大学出版局）

リアルタイム OS

- μITRON4.0 標準ガイドブック（パーソナルメディア社）
- 実用 組込み OS 構築技法―情報通信を支える基礎技術 RTOS 入門（永井 正武 (著), 権藤 正樹 (著), 沢田 勉 (著)）
- 組み込みネット http://www.kumikomi.net/
- 組み込み Linux 入門（TECH I　16 号、CQ 出版）
- FreeBSD Press14・15・16 号
- Design Wave 2003 年 4 月号・5 月号
- HI シリーズテキスト（日立半導体セミナー）
- Embedded UNIX１～３（CQ 出版）
- TOPPERS 付属ドキュメント
- リアルタイム／マルチタスクシステムの徹底研究（TECH I　15 号、CQ 出版）
- リアルタイム OS と組み込み技術の基礎（TECH I　17 号、CQ 出版）←買うこと

以上