AKI-H8-3048マイコンキット

HITACHI H8/3048F使用マイコンボード

■128KBフラッシュROM内蔵。
■16MHz高速動作。
■専用フラッシュROMライター・
　コントロールソフト・
　アセンブラソフト付

ＨＩＴＡＣＨＩ　Ｈ８／３０４８Ｆ使用

ＡＫＩ-Ｈ８マイコンボード

１２８ＫＢフラッシュＲＯＭ内蔵　１６ＭＨｚ高速動作　ＡＤ・ＤＡ内蔵

専用フラッシュＲＯＭライター・コントロールソフト・アセンブラソフト付き

★日立製１６ビットＣＰＵ Ｈ８／３０４８Ｆを使用したマイコンボードです。１チップにＲＯＭ／ＲＡＭ・周辺回路を全て内蔵しており、ボードは、シンプルかつ高性能です。
★内部アーキテクチャ３２ビットで１６ＭＨｚの高速動作を実現しています。また、乗算・除算命令もサポートしています。１命令１２５ｎｓ(加算命令@１６ＭＨｚ動作時)
★１２８Ｋバイト大容量フラッシュメモリをＣＰＵチップに内蔵しています。プログラムを１００回以上書き替え可能です。従来のＣＰＵに不可欠なＥＰ－ＲＯＭを取り付ける必要がなくなりました。メモリ空間は最大１６Ｍバイトでさらにに ＲＯＭ・ＲＡＭを拡張することもできます。
★高速・高分解能Ａ／Ｄ・Ｄ／ＡコンバータをＣＰＵチップに内蔵しています。
★標準で７８本（最大）のＩ／Ｏポートを装備しています。
★高速ＲＳ２３２ドライバ・レシーバＩＣを使用しており、パソコンや他のマイコンとの通信も容易に行なえます。
★ボードは超小型で名刺サイズより小さく、機器組み込みに最適です。ピンヘッダー付き
★フラットＩＣはすでに基板実装済みです。
★専用ＲＯＭライター・ＷＩＮＤＯＷＳ用ＲＯＭライターコントロールソフト・アセンブラ付きですので、ＤＯＳ／Ｖ，ＰＣ９８シリーズでＨ８マイコンの開発ができます。
添付フロッピーディスクは、１．２ＭＢ(PC98)フォーマットです。

■Ｈ８／３０４８Ｆの主な仕様

メモリ	ＲＯＭ	１２８Ｋバイト	外部拡張可能
	ＲＡＭ	４Ｋバイト	外部拡張可能
周辺回路	ＩＴＵ	１６ビットタイマ×５ＣＨ	
	ＴＰＣ	４ＣＨパルス出力	
	ＷＤＴ	ウオッチドックタイマー	インターバルタイマーとして使用可
	ＳＣＩ	独立２ＣＨ	
	Ａ／Ｄ	１０ビット分解能×８ＣＨ	サンプルホールド内蔵
	Ｄ／Ａ	８ビット分解能×２ＣＨ	
	Ｉ／Ｏポート	入出力端子７８本（最大）	

概要

本キットにはＨ８マイコンボード及びＣＲパーツ，Ｈ８専用マザーボード（ライター兼用）、内蔵フラッシュＲＯＭライター用コントロールソフト（ＷＩＮＤＯＷＳ版）、アセンブラソフト、Ｈ８／３０４８Ｆハード・ソフトの技術資料（ＣＤ－Ｒ）　が入っていますので、このキットでボードの製作、ソフトの開発、ソフトの書込みができます。

Ｈ８マイコン製作は『Ｈ８マイコン製作』、Ｈ８専用マザーボードについては『マザーボード』の取説　、ソフト開発は『ソフトについて』をごらんください。

あらかじめ、部品の数量等を部品表等とてらしあわせて確認してください。

■Ｈ８マイコン製作■

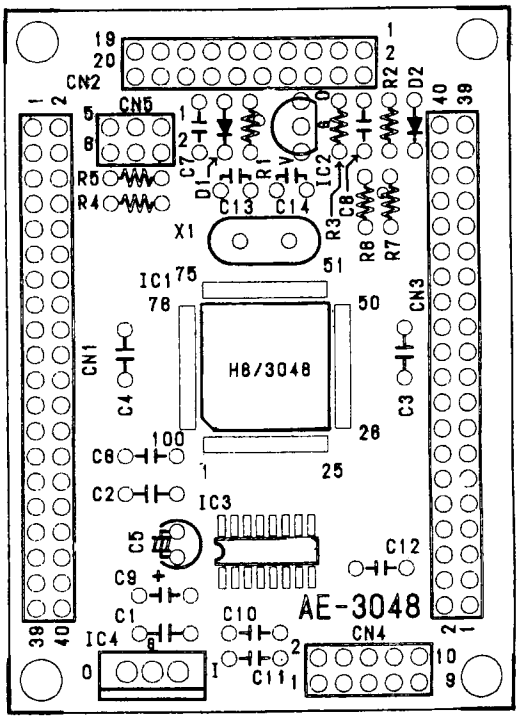
■部品表（Ｈ８マイコンボード）

番号	部品名	数	備考
	専用基板ＡＥ－３０４８	1	Ｈ８／３０４８Ｆ・ＡＤＭ２３２実装済
ＩＣ２	ＰＳＴ５２０（PST600）	1	５Ｖ用リセットＩＣ
ＩＣ４	７８Ｎ０５（７８０５）	1	＋５Ｖレギュレータ（他社相等品の場合有）
C1~4,8	１～２．２μＦ	5	積層セラミックコンデンサ
Ｃ５	１００μＦ(47~100μF)	1	電解コンデンサ
Ｃ６、７ C9~12	０．１～０．２２μＦ	6	積層セラミックコンデンサ
C13,14	１５ｐＦ	2	セラミックコンデンサ
Ｒ１	４．７ＫΩ	1	１／６Ｗカーボン抵抗
Ｒ２	１０ＫΩ	1	１／６Ｗカーボン抵抗
Ｒ３	４７Ω	1	１／６Ｗカーボン抵抗
Ｒ４～７	１０ＫΩ	4	１／６Ｗカーボン抵抗
Ｄ１，．２	小信号ダイオード	2	小信号ダイオード(各社相等品)
Ｘ１	クリスタル	1	１６ＭＨｚ水晶
その他	ピンヘッダ		ＣＮ１～４用　１１０ピン分

■Ｈ８マイコン基板組立

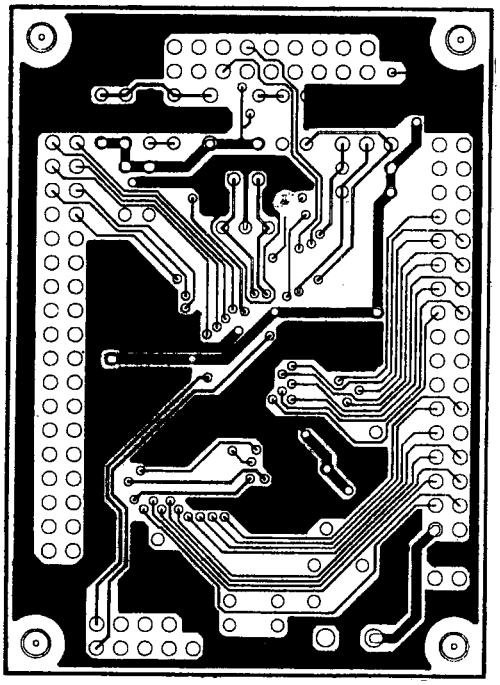
①基板と部品配置図をよく照らし合わせ、十分部品配置を確認してください。
②ダイオード、ＩＣ、電解コンデンサーには極性がありますので注意してください。
③セラミックコンデンサ、抵抗、ダイオードの順に取り付けます。
④電解コンデンサ、ＩＣ、水晶（１６ＭＨｚ）などの大きい部品を取り付けます。電解コンデンサは基板「＋」印に合わせて取り付けます。７８Ｎ０５は文字面を内側に向けて取り付けます。
⑤ピンヘッダを４０Ｐ２本、２０Ｐ１本、１０Ｐ１本に切り、ＣＮ１～４に取り付けます。
ＣＮ４はライターに接続しますので、下向きに取り付けてください。
ＣＮ１～３はお客さまのシステムに合わせて、どちらの向きに取り付けてもかまいません。
ＣＮ５はＣＰＵモード設定用です。
通常はモード７で使用しますので何も取り付けません。（プルアップ済）

■Ｈ８マイコン部品配置図

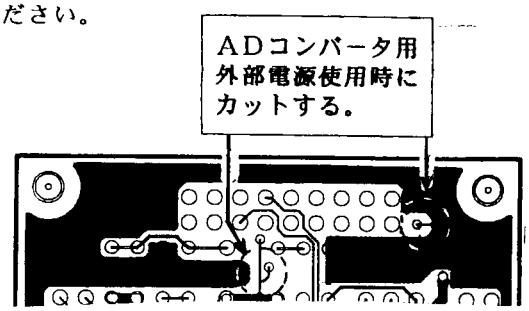


★ＣＮ１～ＣＮ４の相互の間隔は、２．５４ｍｍ×ｎ倍になっています。
２．５４ｍｍピッチの基板（当社Ｂ型ユニバーサル基板等）に、そのまま差さるようになっています。

■パターン図（半田面）



□ＡＤコンバータ
Ｈ８マイコンには、ＡＤコンバータが８ＣＨ内蔵されています。アナログ電源、アナログ基準電源は、ボード内の５Ｖに、接続されていますので、そのまま使用することが出来ます。
外部のアナログ電源、アナログ基準電源を使用する場合は、ボードのパターンを「カット部」で、カットしてください。
アナログＧＮＤは電源のＧＮＤに接続してください。



□動作周波数
メーカーの動作保証範囲は２ＭＨｚ～１６ＭＨｚです。
当社実験では５Ｖ時２０ＭＨｚまで書込および動作を確認しています。

書込プログラムは１６ＭＨｚ用になっています（書込まれたプログラムはどの周波数でも動作します。）ので、他のクリスタルで内蔵ＲＯＭを書込む場合は書込プログラムを用意する必要があります。

□動作電源
動作電源は三端子レギュレータ７８Ｎ０５による安定化回路がのっていますので、７．５Ｖ以上で２００ｍＡ以上供給出来るものをご用意ください。安定化された５Ｖがある場合は三端子レギュレータをパスしてください。

□シリアルインターフェイス
Ｈ８マイコンには、ＲＳ２３２Ｃ用ＳＣＩ（ｼﾘｱﾙｺﾐﾆｹｰｼｮﾝｲﾝﾀｰﾌｪｲｽ）が２チャンネルついています。また、２３２用ドライバ、レシーバのＩＣが基板に実装されています。
ポート９の０－３がＲＳ２３２用ＳＣＩです。また、ＳＣＩは内蔵フラッシュＲＯＭの書き込み時に使用します。

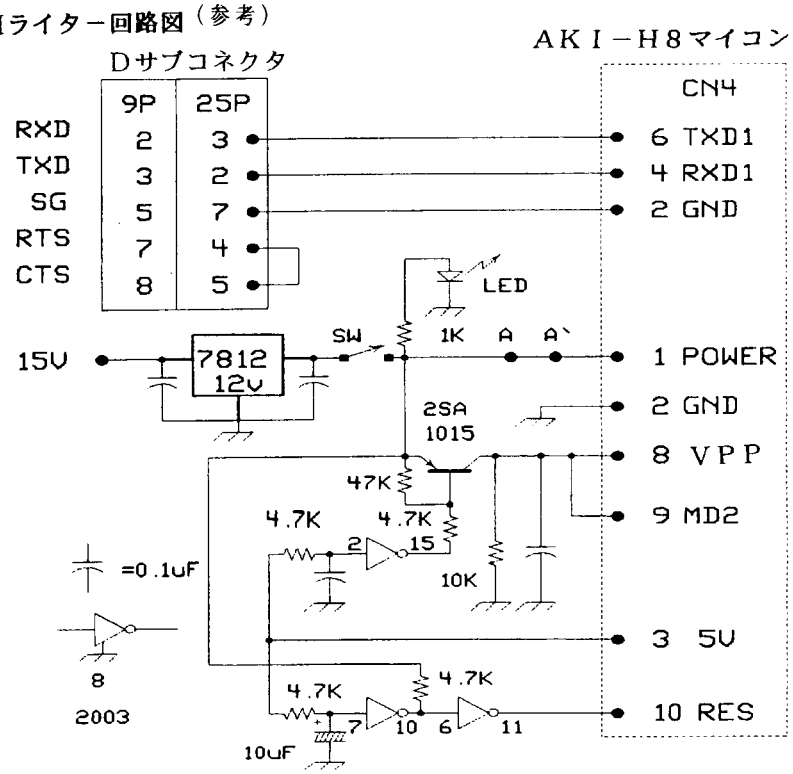
クロック同期通信で外部にクロックを出力する場合は、使用しないチャンネルのＴＸＤをＴＴＬＩ／Ｏの入力に設定し、そこにクロック出力（ＳＣＫ）を接続してください。
★注意
ポート９の０－３は２３２用ドライバＩＣの出力に接続されていますので、一般のＩ／Ｏとして使用することはできません。また、ポート９の２、３は絶対にＴＴＬＩ／Ｏの出力に設定してはいけません。
ユーザープログラムでＳＣＩを使用しない場合は、プログラムでＨ８／３０４８のポート９の０、１をＴＴＬＩ／Ｏの出力に、ポート９の２、３をＴＴＬＩ／Ｏの入力に設定してください。　（リセット時は全て入力ピンです。）

■ライター製作■
このセットにはＲＯＭライター機能付のマザーボードが付属しています。マザーボードの製作マニュアルに従いマザーボードを組立てご使用ください。
以下の『製作（参考）』はマザーボード以外に専用のフラッシュＲＯＭライターを製作する場合の参考資料です。（部品は付属していません）
通常はマザーボードのＲＯＭライター機能をご使用ください。

■部品表（ライター）（参考）

種類	部品名	数	備考
	ユニバーサル基板	1	Ｃタイプ ユニバーサル基板
ＩＣ	２００３	1	７回路入りトランジスタアレー
ＩＣ	７８１２	1	１２Ｖレギュレータ（他社相等品の場合有）
Ｃ	０．１～０．２２μＦ	4	積層セラミックコンデンサ
Ｃ	１０μＦ	1	電解コンデンサ　６Ｖ以上
抵抗	１ＫΩ	1	１／４Ｗカーボン抵抗
抵抗	４．７ＫΩ	4	１／４Ｗカーボン抵抗
抵抗	１０ＫΩ	1	１／４Ｗカーボン抵抗
抵抗	４７ＫΩ	1	１／４Ｗカーボン抵抗
ﾄﾗﾝｼﾞｽﾀ	２ＳＡ１０１５	1	ＰＮＰトランジスタ
ＬＥＤ	発光ダイオード	1	発光ダイオード
その他	ピンフレーム	1	１０Ｐ用に切ってご使用ください。
	Ｄサブコネクタ	1	２５Ｐ　Ｄサブ（メス）

■ＲＯＭライター回路図（参考）



■製作（参考）

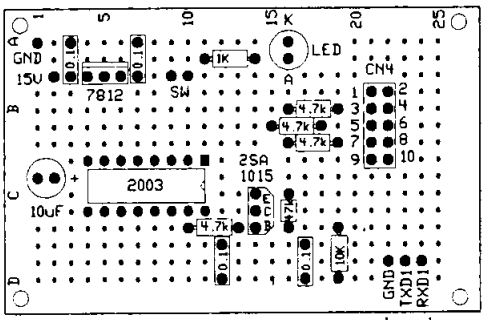
①部品配置図にしたがい抵抗、コンデンサから取り付けていきます。
②１０μＦ電解コンデンサ・ＬＥＤ・トランジスタ・２００３は極性が有りますので向きを間違えずに取り付けてください。
③ピンフレーム１０Ｐは部品面側に取り付けてください。
④ユニバーサル基板ですのでパターンはメッキ線等で、半田付けして作ってください。
⑤Ｄサブコネクタのピン番号はピン側の根本か穴のそばにありますので間違えないようにビニール線等で基板のＴＸＤ１、ＲＸＤＩ、ＧＮＤに接続してください。
ＲＴＳをＣＴＳにＤサブコネクタで接続し、信号が折り返すようにします。
⑥製作後は部品配置、パターン、半田付け等を十分にチェックし、Ｈ８マイコンを接続する前にＳＷ（スイッチ）ＯＮでＬＥＤが点灯しピンフレームの１番ピンに１２Ｖがきていることを確認してください。

Ｈ８マイコンを７８Ｎ０５をパスする形で製作した場合はＲＯＭライター回路図のＡ－Ａ'間を切り、そこに７８Ｎ０５を入れ、ピンフレームの１番ピン（ＰＯＷＥＲ）に５Ｖが入る様にＲＯＭライターを改造してください。

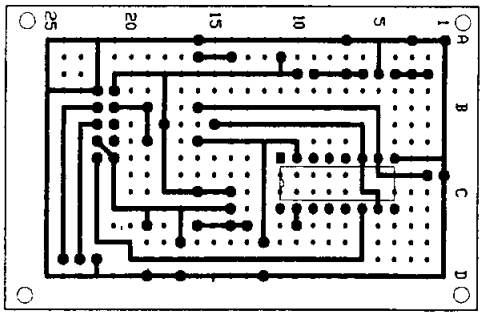
□ＲＯＭライターの電源

電源は１５Ｖ、２００ｍＡ以上とれる電源をご用意ください。
Ｈ８マイコン内蔵ＲＯＭのＶＰＰは１２Ｖ±５％です。この範囲を超えると正しく書き込みができませんので、ＲＯＭライター用電源は、リップル等が少なく、余裕のあるものをご用意ください。

■ＲＯＭライター部品配置図（参考）



■ＲＯＭライターパターン図



Ｄサブコネクタへ

■ＲＯＭライターの使い方■

（ＲＯＭライターソフトの使用方法もあわせてお読みください。）

①ＲＯＭライターに電源を接続する。
②ライターのＳＷ（スイッチ）をＯＦＦにし、Ｈ８マイコンのＣＮ４をＲＯＭライターに差し込む。　（Ｈ８マイコンの他のコネクタは何も接続しない。）

□重要
Ｈ８／３０４８ＦはモードMD２がＶＰＰ１２Ｖ端子と兼用になっています。
ＣＰＵモードが１－４になっている場合は必ずMD２－ＧＮＤ（ＣＮ５　５－６）間の接続をはずしてください。

③Ｄサブコネクタをパソコンのｒｓ２３２（ＣＯＭポート）に接続する。
④ＷＩＮ用ＲＯＭライターソフトに従い、書き込みを行なう。
ＲＯＭライターソフトのインストール、使用方法等は　ソフト説明　及び　フロッピーＤＩＳＫのＤＯＣをお読みください。
⑤書き込み終了後ＳＷ（スイッチ）をＯＦＦにしてからＨ８マイコンをはずしてください

■ ソフト説明■
このキットにはアセンブラ、ライター、サンプルユーティリティソフトが付属していますので、それらを使用してソフトの開発、書き込みまでできます。
アセンブラはMS－DOS用です。ＷＩＮＤＯＷＳのＤＯＳプロンプト（ＤＯＳ窓）でも動作します。アセンブラソースファイルからＨ８マイコンで実行できるＨＥＸファイルを作ります。
ＲＯＭライターソフトはＷＩＮＤＯＷＳ専用です。アセンブラで作ったＨＥＸファイルをＨ８マイコンの内蔵フラッシュＲＯＭに書き込みます。

□フロッピーの内容　（ＣＤ－Ｒの場合はあらかじめ解凍したものがはいっています。）

ディレクトリ名	ファイル名	内容
ＡＳＭ	Ｈ８ＡＳＭ．ＥＸＥ	アセンブラ関係のソフトを自己解凍型にまとめて圧縮したもの
	（解凍した内容）	
	Ａ３８Ｈ．ＥＸＥ	アセンブラ
	Ｌ３８Ｈ．ＥＸＥ	リンカー
	Ｃ３８Ｈ．ＥＸＥ	コンバータ
	Ｈ３８ＨＭＡＮ．ＤＯＣ	マニュアル（ワード文書）
	Ｈ３８ＨＭＡＮ．ＴＸＴ	マニュアル（テキスト文書）
ＷＲＩＴＥＲ	ＳＥＴＵＰ．ＥＸＥ	ＷＩＮＤＯＷＳ用ＳＥＴＵＰソフト
	ＳＥＴＵＰ．ＩＮＦ	ＳＥＴＵＰ内容
	ＦＬＡＳＨ．ＥＸＥ	ライタープログラム本体
	３０４８．ＩＮＦ	フラッシュメモリーインフォメーション
	３０４８．ＳＵＢ	書き込み専用プログラム
ＳＡＭＰＬＥ	サンプルソフトが入っています。内容・使い方は、ＲＥＡＤＭＥをお読みください。	
ＭＡＮ	Ｈ８／３０４８Ｆマニュアル（ＭＡＮ．ＤＯＣをお読みください。）	

■アセンブラソフト■
アセンブラソフトはＨ８マイコン用ソースファイル（ユーザーが製作）をコンパイルし、ＲＯＭライター用のＨＥＸファイルに変換するものです。

□圧縮の解凍　（ＣＤ－Ｒの場合はあらかじめ解凍したものがはいっています。）
アセンブラ関係のソフトは、圧縮プログラムＬＨＡで自己解凍型に圧縮してあります。
あらかじめハードＤＩＳＫ等にアセンブラを入れるディレクトリを作っておき、ＤＩＳＫ内のＨ８ＡＳＭ．ＥＸＥをコピーします。
ＭＳ－ＤＯＳまたは、ＷＩＮＤＯＷＳのＤＯＳプロンプト（ＤＯＳ窓）から、Ｈ８ＡＳＭ．ＥＸＥを実行すると、ソフト及びＤＯＣファイルが生成されます。

Ａ３８Ｈ．ＥＸＥ	アセンブラ	ソースファイルをＨ８用機械語に翻訳します。
Ｌ３８Ｈ．ＥＸＥ	リンカー	アセンブラで作った複数の機械語や、ライブラリを統合します。
Ｃ３８Ｈ．ＥＸＥ	コンバータ	ＲＯＭライター用のＨＥＸファイル（Ｓフォーマット）に変換します。
Ｈ３８ＨＭＡＮ．ＤＯＣ		アセンブラソフトのマニュアルで、ＷＩＮＤＯＷＳのワード用文書です。
Ｈ３８ＨＭＡＮ．ＴＸＴ		同じ内容のテキスト形式のファイルです。

★ＬＨＡは吉崎栄泰氏のアーカイバーソフトで、日本を代表する圧縮解凍プログラムです。
大変有用なソフトを公開していただき、この場にてお礼を申し上げます。

□アセンブラソフトの実行
アセンブラはＭＳ－ＤＯＳ用です。ＷＩＮＤＯＷＳのＤＯＳプロンプト（ＤＯＳ窓）でも動作します。
詳しい使用方法はマニュアル（Ｈ３８ＨＭＡＮ）をごらんください。

★実行例　　ＴＥＳＴ．ＭＡＲというソースファイルから、ＨＥＸファイルをつくる。

Ａ３８Ｈ．ＥＸＥ　ＴＥＳＴ．ＭＡＲ　（リターン）
この処理により、ＴＥＳＴ．ＯＢＪが生成される。
⇩
Ｌ３８Ｈ．ＥＸＥ　ＴＥＳＴ．ＯＢＪ　（リターン）
この処理により、ＴＥＳＴ．ＡＢＳが生成される。
⇩
Ｃ３８Ｈ．ＥＸＥ　ＴＥＳＴ．ＡＢＳ　（リターン）
この処理により、ＴＥＳＴ．ＭＯＴが生成される。

ＴＥＳＴ．ＭＯＴが
ＲＯＭライタ用ＨＥＸファイル
（Ｓフォーマット）です。

■ＲＯＭライターソフト■
□インストール
ＷＩＮＤＯＷＳ上からＤＩＳＫ内のＳＥＴＵＰ．ＥＸＥを実行してください。
ディレクトリ名はドライブ番号からすべて打ち込んでください。

□ライターソフトの起動
ＦＬＡＳＨ内のＦ－ＺＴＡＴ（ＦＬＡＳＨ．ＥＸＥ）がＲＯＭライタープログラムです。
ＷＩＮＤＯＷＳから起動してください。

□ＲＯＭライタープログラム（ＦＬＡＳＨ．ＥＸＥ）の使用方法
①モード選択
通常はそのまま「設定」をクリックしてください。
「モード選択」は、ブートモードを選択してください。ユーザープログラムモードはサポートしていません。
「タイムアウト時間」は消去の待時間設定です。通常は５～１０秒に設定してください。
「デバイス選択」は３０４８を選択してください。
②ブートモード設定
Ｈ８マイコンは、ＲＯＭライターのＳＷ（スイッチ）をＯＮにすることで、ブートモードになります。
ＲＯＭライターのＳＷ（スイッチ）をＯＮにし、その後「ＯＫ」をクリックしてください。パソコンから、書き込み専用プログラム（３０４８．ＳＵＢ）のＲＡＭ転送が開始され、１００％まで転送後、つづいて内蔵フラッシュＲＯＭを自動消去し終了します。
③ユーザープログラムの転送とＲＯＭ書き込み
ＷＩＮＤＯＷＳのメニューバーの「ＷＲＩＴＥ」をクリック後、画面に従い書き込みファイル名（例Ｃ：￥ＴＥＳＴ．ＭＯＴ）を「参照」で選んでください。「ＯＫ」をクリックすると、書き込みが開始します。小さなプログラムの場合、画面を注意して見ていないと、すぐ終了してしまいます。
④書き込み終了後、ＲＯＭライターのＳＷ（スイッチ）をＯＦＦにしてからＨ８マイコンを取り外してください。

注意
ブートモードは内蔵フラッシュＲＯＭの消去を自動で行います。そのため、あらかじめ内蔵フラッシュＲＯＭ内に書かれている内容をパソコンに読み込むことや、書き込み後の内容を読み出すことは、できません。
ＲＡＭ　Ｅｍｕｌａｔｉｏｎは、フラッシュＲＯＭの一部をＲＡＭで置き換えてエミュレーションするものです。このライターでは、対応していません。

■書き込みモードについて■

Ｈ８／３０４８Ｆの内蔵フラッシュＲＯＭを書き込むには、つぎの３モードがあります。このキットのライターはブートモード専用です。（Ｈ８マイコン本体はすべてのモードに対応しています。）

ブートモード　書き込み専用プログラムをパソコンから、ＲＡＭに転送し、そのプログラムにより、パソコンと通信しユーザープログラムを書き込む。

ユーザーモード　あらかじめ書き込み専用プログラムとＲＡＭ転送プログラムをブートモードで、フラッシュＲＯＭに書いておき、書き込みプログラムをＲＡＭに転送し、そのプログラムでユーザープログラムを書き込む。

ＰＲＯＭモード　２８Ｆ１０１用対応のＲＯＭライターを使用する方法

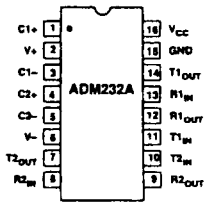
各モードの詳しい内容はハードウェアマニュアルをごらんください。

■各ＩＣの資料と外形

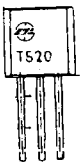
高速、+5V、0.1μF
RS-232ドライバ／レシーバ

特長
200kB/秒の転送レート
小容量（0.1μF）値のチャージ・ポンプ用コンデンサ
+5V単一電源動作
EIA-232-EおよびV.28規格に適合
2個のドライバと2個のレシーバ
DC-DCコンバータを内蔵
+5V電源で±9Vの出力振幅
±30Vのレシーバ入力レベル
MAX222／MAX232A／MAX242とピン・コンパチブル

ADM２３２AANR

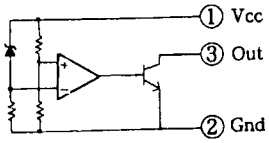


ＰＳＴ５２０
ＰＳＴ６００

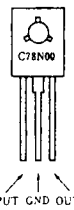


①②③

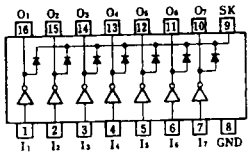
■等価回路



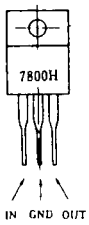
７８Ｎ０５



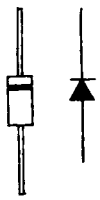
２００３



７８１２
７８０５

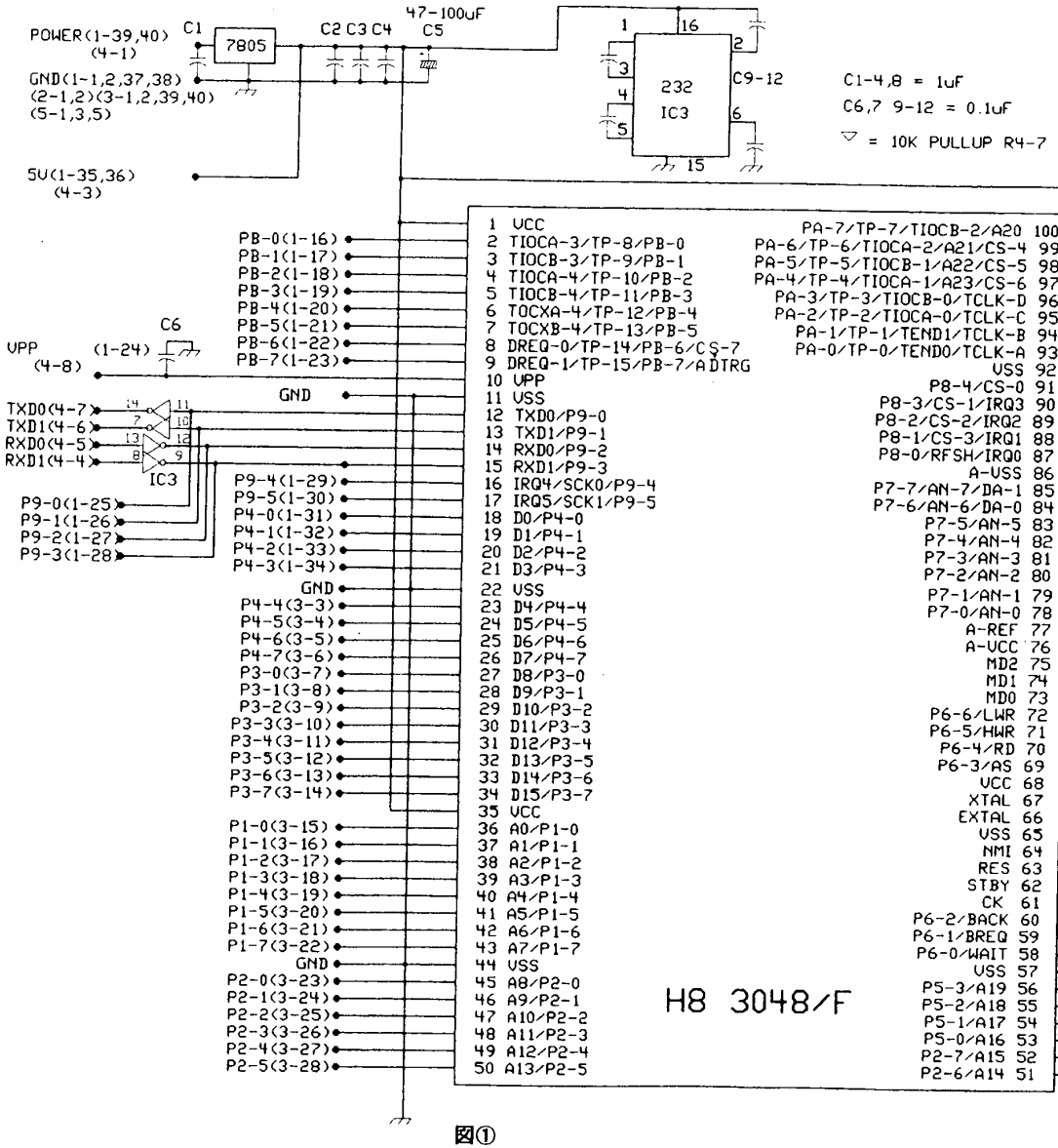


小信号ダイオード
ガラスモールド



Ｈ８－３０４８使用　Ｈ８マイコンキット
１９９７．１０　（有）秋月電子通商　ＫＡＫＥ　Ｍ.Ｏ
質問・お問い合わせ等は、往復葉書又は返信用切手同封の封書にてお願いいたします。
〒１５８　東京都世田谷区瀬田５－３５－６

■回路図



図①

■ＣＰＵモードの種類と設定

動作モード	MD2	MD1	MD0	内蔵ROM	内蔵RAM	アドレス空間
モード１	0	0	1	無効	有効	１Ｍバイト
モード２	0	1	0	無効	有効	１Ｍバイト
モード３	0	1	1	無効	有効	１６Ｍバイト
モード４	1	0	0	無効	有効	１６Ｍバイト
モード５	1	0	1	有効	有効	１Ｍバイト
モード６	1	1	0	有効	有効	１６Ｍバイト
モード７	1	1	1	有効	有効	１Ｍバイト

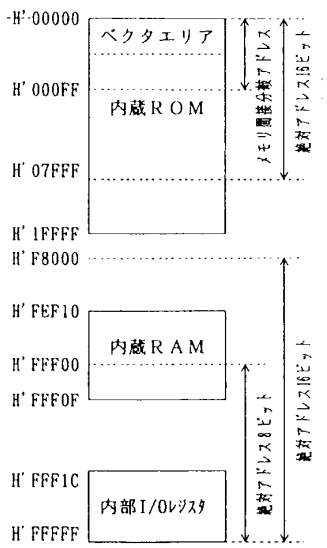
ＣＮ５		ＣＮ５	
１－２	接続　ＭＤ０＝０	１－２	無接続　ＭＤ０＝１
３－４	接続　ＭＤ１＝０	３－４	無接続　ＭＤ１＝１
５－６	接続　ＭＤ２＝０	５－６	無接続　ＭＤ２＝１

■ＣＰＵモード設定

Ｈ８／３０４８ＦはＲＯＭ，ＲＡＭを内蔵していますが、外部に拡張することも可能です。
ＣＮ５のＭＤ０～２がモード設定用コネクタです。
ＭＤ０～２はそれぞれ５ＶにプルアップされていますのでＣＮ５オープンの状態でモード７（シングルチップモード・内蔵ＲＯＭＲＡＭのみ有効）になっています。通常はモード７で使用します。

各モードの詳細はハードウエアマニュアル第３章「ＭＰＵ動作モード」をご覧ください。

■メモリマップ（ＣＰＵモード７）



■コネクター→ＩＣピン配置表

CN1 ピン番号	名称	CN2 ピン番号	名称	CN3 ピン番号	名称
1	GND	1	GND	1	GND
2	GND	2	GND	2	GND
3	87 P8－0	3	62 STBY	3	23 P4－4
4	88 P8－1	4	63 RES	4	24 P4－5
5	89 P8－2	5	64 NMI	5	25 P4－6
6	90 P8－3	6	69 P6－3	6	26 P4－7
7	91 P8－4	7	70 P6－4	7	27 P3－0
8	93 PA－0	8	71 P6－5	8	28 P3－1
9	94 PA－1	9	72 P6－6	9	29 P3－2
10	95 PA－2	10	76 AVCC	10	30 P3－3
11	96 PA－3	11	77 AREF	11	31 P3－4
12	97 PA－4	12	78 P7－0	12	32 P3－5
13	98 PA－5	13	79 P7－1	13	33 P3－6
14	99 PA－6	14	80 P7－2	14	34 P3－7
15	100 PA－7	15	81 P7－3	15	36 P1－0
16	2 PB－0	16	82 P7－4	16	37 P1－1
17	3 PB－1	17	83 P7－5	17	38 P1－2
18	4 PB－2	18	84 P7－6	18	39 P1－3
19	5 PB－3	19	85 P7－7	19	40 P1－4
20	6 PB－4	20	86 AVSS	20	41 P1－5
21	7 PB－5			21	42 P1－6
22	8 PB－6	CN4		22	43 P1－7
23	9 PB－7	1	POWER	23	45 P2－0
24	10 RES0	2	GND	24	46 P2－1
25	12 P9－0	3	5V	25	47 P2－2
26	13 P9－1	4	U2-8 RXD1	26	48 P2－3
27	14 P9－2	5	U2-13 RXD0	27	49 P2－4
28	15 P9－3	6	U2-7 TXD1	28	50 P2－5
29	16 P9－4	7	U2-14 TXD0	29	51 P2－6
30	17 P9－5	8	10 VPP	30	52 P2－7
31	18 P4－0	9	75 MD2	31	53 P5－0
32	19 P4－1	10	63 RES	32	54 P5－1
33	20 P4－2			33	55 P5－2
34	21 P4－3	CN5		34	56 P5－3
35	5V	1	GND	35	58 P6－0
36	5V	2	73 MD－0	36	59 P6－1
37	GND	3	GND	37	60 P6－2
38	GND	4	74 MD－1	38	61 CK
39	POWER	5	GND	39	GND
40	POWER	6	75 MD－2	40	GND

POWER　1－39, 40　　5V　1－35, 36
GND　　1－1, 2, 37, 38　2－1, 2　3－1, 2, 39, 40

■ＩＣ→コネクタピン配置表

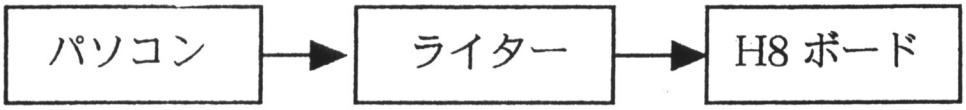
ICピン番号	名称	コネクタ	ICピン番号	名称	コネクタ
1	VCC	5V	51	P2－6	3－29
2	PB－0	1－16	52	P2－7	3－30
3	PB－1	1－17	53	P5－0	3－31
4	PB－2	1－18	54	P5－1	3－32
5	PB－3	1－19	55	P5－2	3－33
6	PB－4	1－20	56	P5－3	3－34
7	PB－5	1－21	57	VSS	GND
8	PB－6	1－22	58	P6－0	3－35
9	PB－7	1－23	59	P6－1	3－36
10	RES0	1－24	60	P6－2	3－37
11	VSS	GND	61	CK	2－38
12	P9－0 (TXD0)	1－25	62	STBY	2－3
13	P9－1 (TXD1)	1－26	63	RES	2－4
14	P9－2 (RXD0)	1－27	64	NMI	2－5
15	P9－3 (RXD1)	1－28	65	VSS	GND
16	P9－4	1－29	66	EXTAL	
17	P9－5	1－30	67	XTAL	
18	P4－0	1－31	68	VCC	5V
19	P4－1	1－32	69	P6－3	2－6
20	P4－2	1－33	70	P6－4	2－7
21	P4－3	1－34	71	P6－5	2－8
22	VSS	GND	72	P6－6	2－9
23	P4－4	3－3	73	MD－0	5－2
24	P4－5	3－4	74	MD－1	5－4
25	P4－6	3－5	75	MD－2	5－6
26	P4－7	3－6	76	AVCC	2－10
27	P3－0	3－7	77	AREF	2－11
28	P3－1	3－8	78	P7－0	2－12
29	P3－2	3－9	79	P7－1	2－13
30	P3－3	3－10	80	P7－2	2－14
31	P3－4	3－11	81	P7－3	2－15
32	P3－5	3－12	82	P7－4	2－16
33	P3－6	3－13	83	P7－5	2－17
34	P3－7	3－14	84	P7－6	2－18
35	VCC	5V	85	P7－7	2－19
36	P1－0	3－15	86	AVSS	2－20
37	P1－1	3－16	87	P8－0	1－3
38	P1－2	3－17	88	P8－1	1－4
39	P1－3	3－18	89	P8－2	1－5
40	P1－4	3－19	90	P8－3	1－6
41	P1－5	3－20	91	P8－4	1－7
42	P1－6	3－21	92	VSS	GND
43	P1－7	3－22	93	PA－0	1－8
44	VSS	GND	94	PA－1	1－9
45	P2－0	3－23	95	PA－2	1－10
46	P2－1	3－24	96	PA－3	1－11
47	P2－2	3－25	97	PA－4	1－12
48	P2－3	3－26	98	PA－5	1－13
49	P2－4	3－27	99	PA－6	1－14
50	P2－5	3－28	100	PA－7	1－15

『AKI-H8/3048F（フラッシュメモリ内蔵）
超高性能マイコンボード』

製作編

これは、H8-3048 キットを実際に作った記録です。皆さんがキットを組み立てるときの参考にして下さい。

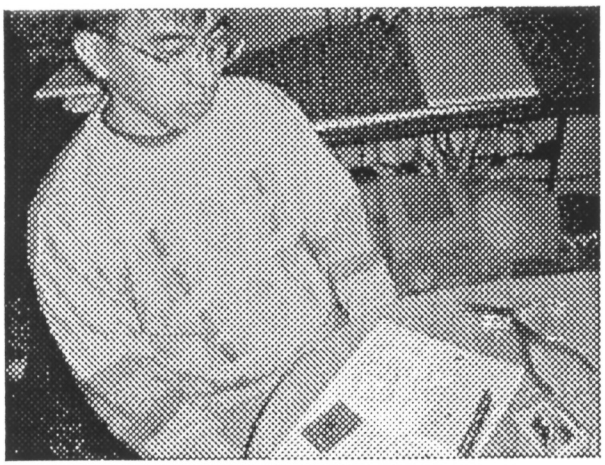
AKI-H8 マイコンボードの構成は次のようになっています。



パソコンでプログラムを作りライター経由でマイコンボードにそのプログラムを書き込みます。ライターとマイコンボードは自分で部品を半田付けして組み立てます。
パソコンは Win95 が動作しているものが必要です。

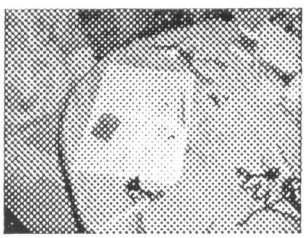
この文書では、ライターとマイコンボードの組み立てを説明します。

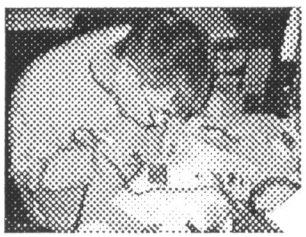
－－組み立てる人のプロフィール－－
中学校でインターホンを作ったことのある、大学生の吉田さんです。



<<第１日目>>
まず、ＣＰＵボードの半田付けです。15:20 から 17:30 まで、およそ２時間かかりました。ライターボードは第２日めに作ります。
半田こての台がないのがちょっとさみしいですね。半田こては２０W程度の小形のものを用意します。

抵抗やコンデンサといった小さな部品から取り付けます。普通は基板の裏から半田付けをするのですが、なかなか難しいので、部品側から半田付けをしてもかまいません。
あとでも書きますが、ダイオード、IC、電解コンデンサには極性があるので正しい向きで取り付けましょう。電解コンデンサは基板にはプラス（+）側に印がついていますが、コンデンサ自体にはマイナス（－）側に印がついています。





－－－部品が分からない－－－
吉田さんは、コンデンサと抵抗の区別はなんとか付きますが、ちょっと自信がありません。というわけで、簡単に説明しておきます。

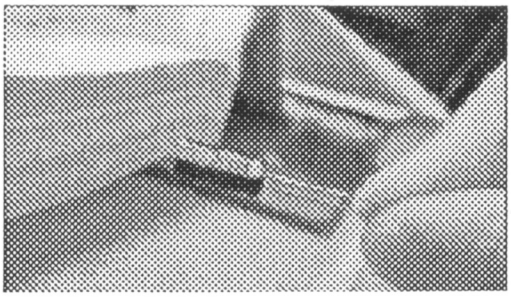
■1、1■　コンデンサ
　電解コンデンサ（ケミコン）と積層セラミックコンデンサの２種類あります。あ、もう１種類ありました。ちいさなセラミックコンデンサです。

電解コンデンサは大きいのが一つだけです。
電解コンデンサにはプラスとマイナスの電極があります。それがどちらか分からないんですね。電解コン

デンサをよく見ると白く大きなマイナス記号が書かれています。そこがマイナスです。あるいは、足の短い方がマイナスになります。

回路図や、基板にはプラスの印がありますが、電解コンデンサには、プラスではなく、マイナス側に印が付いています。

電解コンデンサは基板に根元まで差し込んでもかまいません。



積層セラミックコンデンサは青いコンデンサで、容量の異なるものが２種類あります。
104 と書かれたものと、155 の２種類はいっていました。ほかの値が入っているかもしれません。
１０４が 0.1 μＦ（マイクロファラッド）、１５５は 1.5μＦです。
コンデンサに書かれた値が大きいほうが、容量も大きいのです。５個入っているほうが容量の大きいほうです。
部品を基板に挿します。そのとき、部品が落ちない程度に足を広げます。それから半田付けして、あまった足をニッパで切り落とします。切り落とした足はもう一つの ROM ライター基板の半田付けに使うことにします。
基板を半田こてで暖めるようにして、半田が穴に吸い込まれるように半田付けします。

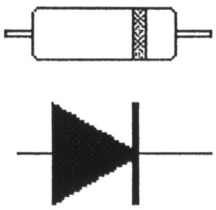
他にちいさなセラミックコンデンサが一つ。
これは一つだけなので分かります。

■１、２■　抵抗は色が分からないです。
茶色１で、赤は２・・・。右端に金色がくるように見て・・・
47 オームは黄・紫・黒・金 4 7 0 = 47 × 10^0 = 47 × 1。１本。
4.7K Ωは、黄・紫・赤・金 4 7 2 = 47 × 10^2 = 47 × 100。１本。
10 K Ωは、茶・黒・橙・金 1 0 3 = 10 × 10^3 = 10 × 1000。全部で５本。説明書では分けて書いてありますが同じ物です。
抵抗はコの字型に足を曲げてから基板に差し込み、半田付けします。

1 0 3
= 10 x 10³ = 10,000Ω

■１、３■　足３本の部品はＩＣです。みんなトランジスタかと思ってました。
小さなリセットＩＣと大きな５ボルトレギュレータ。向きがありますから間違えないように取り付けます。
一般的にこの形のＩＣは根元まで差し込みません。基板から５ミリくらい足が残るようにします。
５ボルトレギュレータは、文字のかかれた方が基板の内側になるように取り付けます。
吉田さんは５ボルトレギュレータを逆に取り付けてしまいました。足をしっかり折り曲げて半田付けしてしまったので、もうとりはすずことはできません。しかたがないのでニッパで足を切ってしまいました。
そのあと、基板の穴（スルーホールといいます）をはんだ吸い取りあみや、すいとり器できれいにしておきます。
本当は、そこに新しい部品を取りつけるとよいのですが、手持ちの部品がなかったため、半田付けのときにあまった抵抗の足をその穴に半田付けしておき、その先に先ほど取り外した５ボルトレギュレータを半田付けしました。

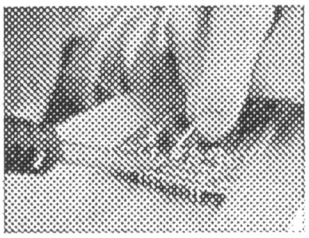
■1、4■　ダイオード
小信号ダイオード。片方に青い線が引いてあり、その方へ電流が流れます。
実物と回路図の関係を示します。



■1、5■　クリスタル
かんずめのような金属製の小さな部品です。

■1、6■　そのほか
ライターの基板とつながる CN4 の取り付け方は注意が必要です。部品が載っている側から半田付けをするように、端子が基板の裏側を向くように取り付けます。

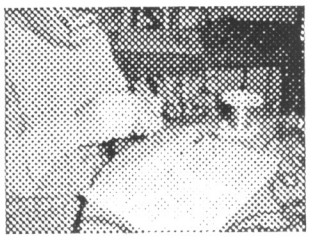
ピンヘッダ。CN4 は下向きに取り付けます。秋月さんの広告に載っている写真はどう見ても上向きです。これは間違って作ったものでしょうか?
ピンヘッダで、４０Ｐとか、２０ＰのＰとはなんでしょう。これは、 2列を両方合わせたピン数です。１０本ずつ2列なら２０Ｐになります。



■1、7■　１日めが終わりました
これだけでは、まだ完成ではありません。
さらに、今回作った CPU ボードにプログラムを書き込むライターボードを作らなければなりません。
<<第２日>>

キットに『ＡＫＩ－Ｈ８マザーボード基板』が入っている場合は、ここで説明している「ROMライターボード」を製作せず、マザーボード基板を製作してください。マザーボードには、ライター機能もあり、専用基板になっているため、製作が簡単で便利です。

2、CPU ボードにプログラムを転送する ROM ライターボードの制作



ユニバーサル基板に作ります。パターンも自分で作らなければならないので CPU ボードよりも時間がかかります。
ダイオード、IC、電解コンデンサには極性があるので正しい向きで取り付けましょう。 LED は足の長いほうがプラス側のアノード (A) になります。
抵抗やコンデンサの切り取った残りで配線できますから、わざわざすずめっき線を購入することはありません。

では、部品の確認をしましょう。

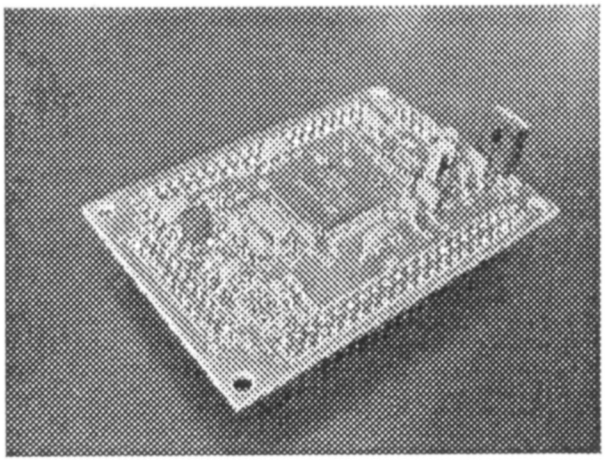
■2、1■　積層セラミックコンデンサ
１種類しか入っていません。
104 で 0.1 μF（マイクロファラッド）４本。

■2、2■　抵抗の取り付け
４種類の抵抗が入っています。
１ＫΩ　茶・黒・赤 1 0 2 = 10 × 10^2 = 10 × 100　　１本
4.7ＫΩ　　黄・紫・赤 4 7 2 = 47 × 10^2 = 47 × 100　　４本
10ＫΩ 茶・黒・橙 1 0 3 = 10 × 10^3 = 10 × 1000　　１本
47ＫΩ 黄・紫・橙 4 7 3 = 47 × 10^3 = 47 × 1000　　１本

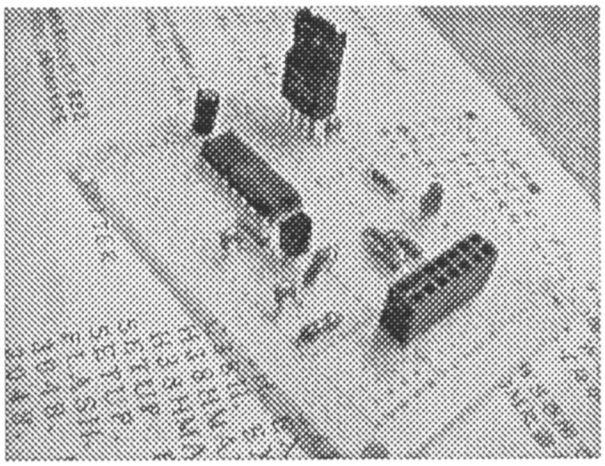
■2、3■　電解コンデンサ
マイナスには記号がついていて、足の長いほうがプラスです。

■2、4■　LED （発光ダイオード）
足の長いほうがプラス。
アノードが足の長いほうでプラス、カソードがマイナス。

■2、5■　１２ボルトレギュレータ
数字の書いてある面が内側に向くように取り付けます。文字の書いてある面を正面にして、左が入力で右が１２ボルトの出力になります。
この部品だけ背が高いので裏側の配線のときに邪魔になってしまいます。折り曲げておいてから取り付けたほうがよいかもしれません。



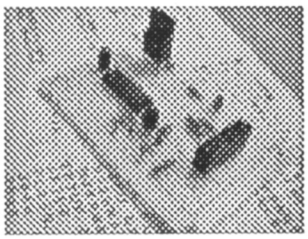
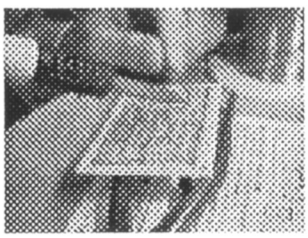
CPU ボード



Writer ボード

■2、6■　ピンフレーム１０Ｐ
実際には１４Ｐ（７×２）のピンフレームが付属してきました。説明書には切るようにと書かれていますが、そのままで使いました。
部品面に取り付けます。部品面というのは、部品が取り付けられている面ということで、部品を取りつけたのと同じ向きにします。この向きは、CPU ボードの CN4 とは裏表が逆です。CPU ボードの CN4 は部品面の裏側の半田面に取り付けました。

■2、7■　裏側の配線をします。
すでに、１時間半が経過しています。
吉田さんは、裏側の配線を半田だけでやってしまおうと考えました。でも、半田でつなげるのではなく、抵抗やコンデンサを半田付けしたときに切り取った足を使います。捨ててしまった人は、すずめっき線などを使いましょう。

■2、8■　２日めが終了しました。
基板はでき上がりました。つぎは簡単なプログラムを書き込んで正常に動作することを確認します。

その方法は、次のソフト編を参照ください。

■［付録Ａ］■抵抗の値
抵抗にはいくつかの色の帯が印刷されています。その色の帯で、抵抗の値をあらわしているのです。
例えば、抵抗値はつぎのように計算して求めます。
103 → 10 × 103Ω = 10 KΩ
472 → 47 × 102Ω = 4.7 KΩ
473 → 47 × 103Ω = 47 KΩ

ここで 103 を色であらわすと、茶－黒－橙となります。色の帯が数字になっているのです。
ここで使われている色と数値の関係を次に示します。

色	黒	茶	赤	橙	黄	緑	青	紫	灰	白
値	0	1	2	3	4	5	6	7	8	9

■［付録Ｂ］■コンデンサの値
コンデンサの値も抵抗の値と同じような考え方で表現されています。
例えばつぎのようになります。

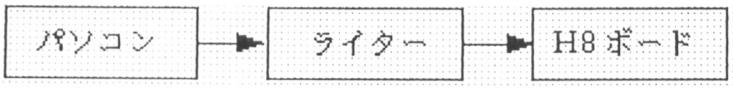
101 → 10 × 101 pF = 100 pF
103 → 10 × 103 pF = 10000 pF = 0.01 μF
104 → 10 × 104 pF = 100000 pF = 0.1 μF
155 → 15 × 105 pF = 1500000 pF = 1.5 μF

『AKI-H8/3048F（フラッシュメモリ内蔵）
超高性能マイコンボード』

H8/3048F編

（ソフト編）

全体の構成
AKI-H8 マイコンボードの構成は次のようになっています。



パソコンでプログラムを作りライター経由でマイコンボードにそのプログラムを書き込みます。ライターとマイコンボードは自分で部品を半田付けして組み立てます。
パソコンは Win95 が動作しているものが必要です。

この文書では、パソコンの準備のしかた、そしてプログラムの作り方を説明します。

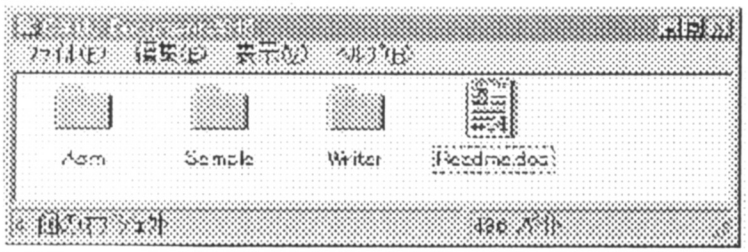
パソコンの準備
パソコン上でプログラムを作り、ライターを経由してマイコンボードにプログラムを書き込みます。
この章では、プログラムを作るためのアセンブラやリンカの準備、それにライター経由でプログラムを書き込むためのライターソフトを使えるようにします。

それでは、フロッピーに入っているアセブラなどをハードディスクにコピーしましょう。

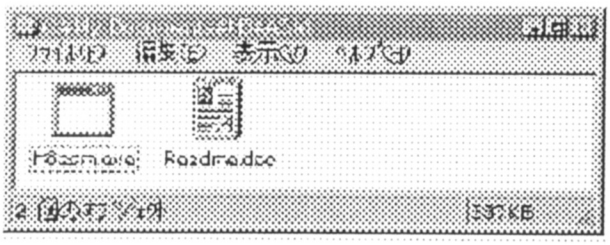
付属しているフロッピーは今は亡き NEC の９８００フォーマット （1.2 M byte） ですから、フツーのＤＯＳマシン (1.4 M byte) では読めない場合があります。
私のマシンで読み込むためにはドライバをインストールする必要がありました。それでも不安定で読めないファイルがあるという状態でした。たまたまフロッピーが２セットあったので、もう１枚のフロッピーで試したところこちらはうまく読むことができました。
読めなかったフロッピーも NEC の９８００では読めました。

こういうこともあるので、もしも読めなかった場合は NEC の９８００を持っている人を探しましょう。そこで 1.4 M byte に変換してもらうか、 2DD タイプの 720 K byte のフロッピーに書き込んでもらいましょう。NEC の古い９８００では 1.4 M byte に対応していない場合がありますから 2DD の 720 K byte のフロッピーに書き込んでもらうのが無難でしょう。

フロッピーの内容はつぎのようになっています。



ＡＳＭフォルダの中には「H8asm.exe」と「ReadMe.doc」が入っています。これがアセンブラかと思うと大違いで、アセンブラやリンカが圧縮されたファイルなのでした。

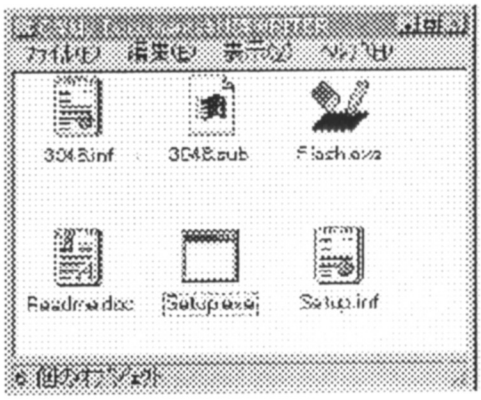


Ｓａｍｐｌｅ にはいろいろなサンプルプログラムやその説明のファイルが入っています。

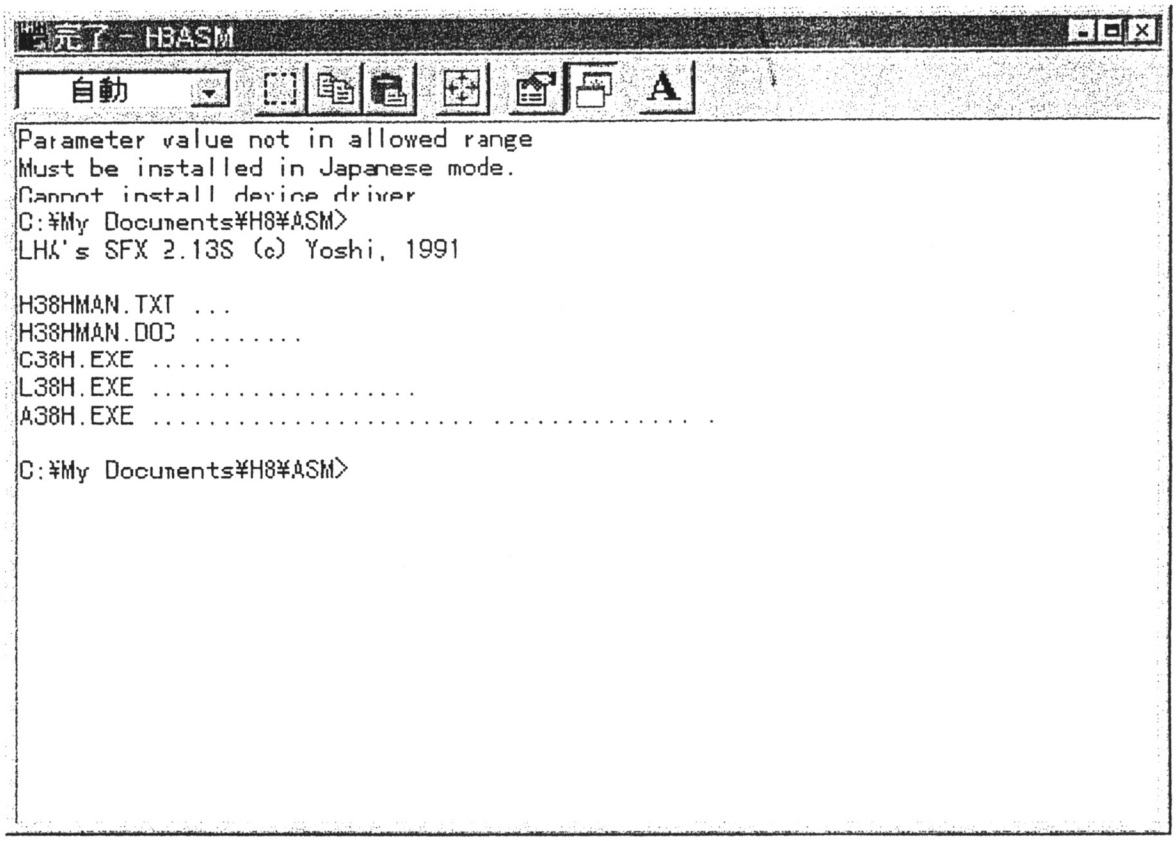
もう一つのフォルダ、Ｗｒｉｔｅｒ にはライター関係のファイルが入っています。

そうそう、これらの画面は表示メニューのオプションで、「.doc」や「.exe」などの拡張子を表示するように設定してあります。あなたのパソコンには「H8asm.exe」ではなく「H8asm」となっているかもしれません

プログラムを使う前にしなければならないのは、圧縮されているファイルを元に戻し、ライター関係のプログラムをインストール（セットアップ）することです。
それでは順番に説明しましょう。



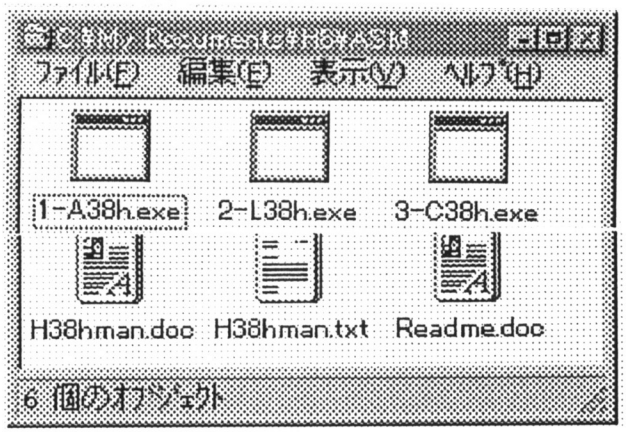
<<アセンブラやリンカを解凍する>>
asm フォルダにある「H8asm.exe」をダブルクリックします。次の画面のように解凍作業が行われて停止するので、ウインドウを閉じて終了します。



解凍が終了すると「A38h.exe」、「C38h.exe」、「L38h.exe」それに２つのドキュメントファイルが生成されます。

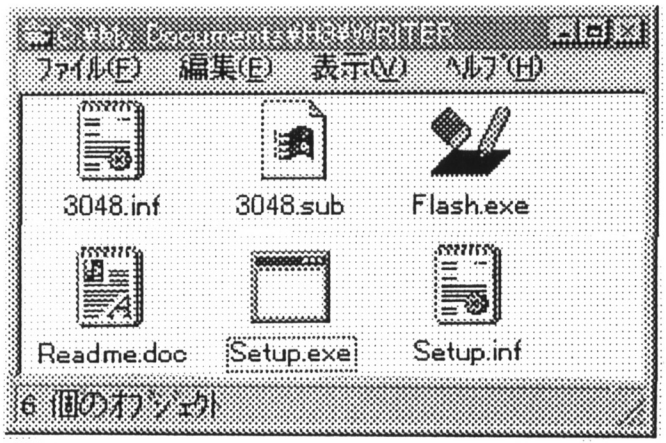
使わないものは捨ててしまいます。それから使いやすいように「アセンブラ」「リンカ」「コンバータ」の順に番号をつけておきます。
そうやって、整理したウインドウの状態を次に示します。

「A38h」がアセンブラ、
「L38h」がリンカ、
「C38h」がコンバータ

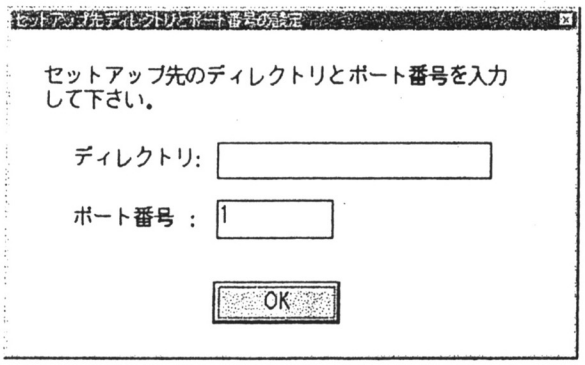


さて、次は ROM ライターソフトを使えるようにします。

<<ROM ライターソフトのインストール>>
WRITER フォルダにある Setup.exe を開きます。



Setup. exe を起動すると次のダイアログが表示されます。

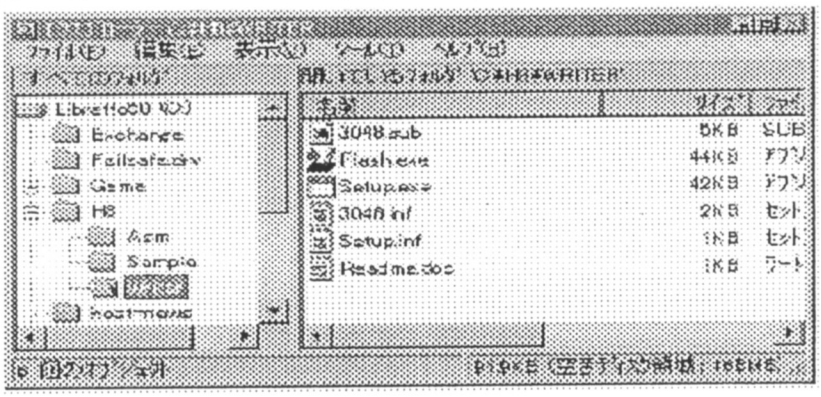


これがわかりません。ディレクトリといわれてもそんなもの覚えていませんし、ポート番号もわかりません。
ディレクトリを選ぶために「参照..」ボタンをぜひつけてほしいものです。もうひとつのポート番号は、マックならモデムポートとか、プリンタポートあるいは PC カードスロットという具合に文字で表示されてその中から選ぶようになっています。おまぬけ Win95 はこんなユーザーインターフェイスでも許されるのでしょうか。

こまりました。

ポート番号はパソコンのマニュアルを見ればわかるかもしれないということにして、ディレクトリの問題を解決しましょう。

えー、たとえば My Documents の H8 3048 の ROM ライターソフト といった長いファイル名、あ、正確にはパス名が、このインストーラソフト Setup.exe でゆるされるのか？といった問題もありますし、こんなことで作業が進まなくなるのはいやですから、短いパス名で覚えやすいものにしてしまいます。DOS で通用する名前にするのです。
ハードディスクのルート「C:¥」あれ？「C¥:」でしたか？はやい話が、マイコンピュータのふつーは「C」になっているハードディスクを開いたところに「H8」というフォルダを移動しましょう。しょせん Win 95 は DOS なんだな、と改めて感じてしまうのでした。はやいとこ MacOS for Intel を使いたいものです。



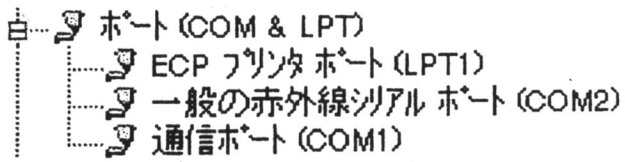
はい、これで Setup.exe のある場所は「C:¥H8¥Writer¥」になりました。これをコピーして Setup.exe のダイアログにペーストしようとしてもだめです。ペーストは受け付けませんでした。パス名をしっかり覚えるかメモをとって、間違えないようにキーボードから打ち込まなければなりません。

さて次はポート番号です。パソコンと ROM ライターはシリアルポートでつながります。シリアルでつながるというだけで、もうなんだかわかっちゃう人は、この説明を飛ばしてどんどんインストール作業へ進みましょう。

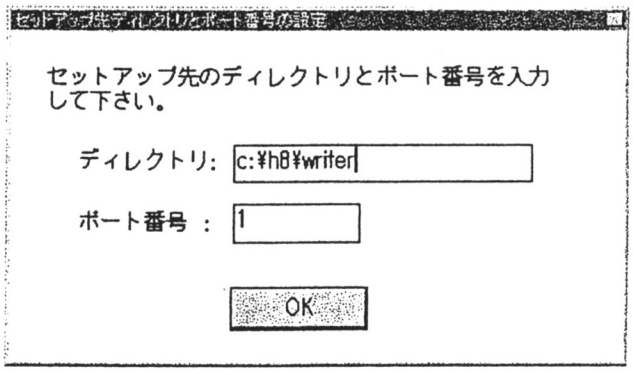
シリアルポートと対になって用いられる言葉がパラレルポートです。シリアルは朝ご飯で牛乳をかけて食べるものではなく、この場合は直列という意味です。それでパラレルは並列。なにが直列になったり並列になったりしているのでしょうか。これはプリンタやモデム、はたまたデジカメなどとパソコンをつないでデータをやり取りするときのお話になります。
一方、パソコンのデータの最小単位は、ま、ビットなのですが、ビットでは0か1しかありませんから、通常は8ビットを一まとめにしたバイトと呼ばれる固まりをデータやり取りの最小単位にしています。
それで、話はパソコンとモデムなどとの間でデータがいったりきたりするときに戻りますが、このいったりきたりするデータが8ビット （以上） をまとめて1バイト（以上）になっているのがパラレル、1ビットずつばらされているとシリアルというのです。最近は USB や FireWire といったシリアルものが流行ですね。

パソコンにはシリアルポートが4つあるようです。あなたがこれから ROM ライターとつなげようとしているシリアルポートが何番なのか設定しなければなりません。

私のパソコンのポートをコントロールパネルにある「システム」で表示されるシステムのプロパティの中のデバイスマネージャを見ると次のようになっていました。



これによるとポート番号は「1」であるらしいことがわかります。
ではさっそく Setup.exe を起動してディレクトリとポート番号を設定しましょう。
私の場合はディレクトリが「C:¥H8¥Writer¥」で、ポート番号が「1」です。



マイコンボードの動作を確認しましょう
マイコンボードとライターが完成し、プログラムも作ることができるようになって、やっとマイコンボードのチェックをすることができます。

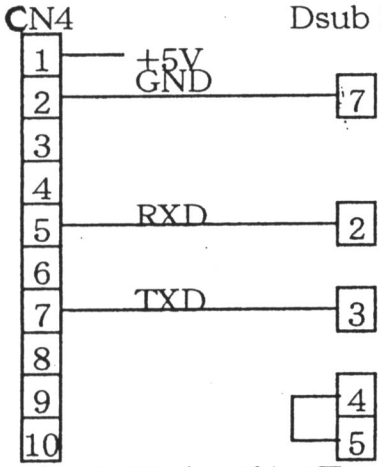
動作試験用の簡単なプログラムを作成してマイコンボードに書き込み、正しく動作していることを確かめます。後ほど詳しく解説しますが、まずその大まかな手順を次に示します。

プログラムを打ち込むのはたいへんなので、フロッピーディスクに入っているサンプルプログラムを使おうかとも思ったのですが、動作確認がたいへんなので、新たに打ち込みましょう。

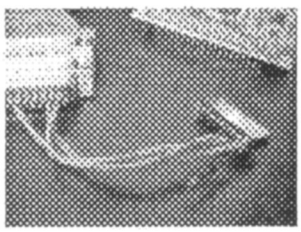
1） プログラムを作る
「メモ帳」や「ワードパッド」などでプログラムを打ち込みましょう。それを、Test.mar として最初の動作試験用プログラムとします。
2） ライターに転送してプログラムを書き込む
Flash.exe を使ってアセンブル・リンクしたプログラムを、ライターを使って書き込みます。
3） 動作させてチェックする
書き込んだプログラムを実行します。

どんな内容のプログラムかというと、シリアル通信用のポートを使って、受信した文字をそのまま送信します。パソコンの通信用ソフト「ハイパーターミナル」を動作させて、マイコンボードと通信を行うのですが、入力した文字がそのまま返ってくるだけですから、あまり、面白いものではありません。
それに、大変心苦しいのですが、マイコンボードとパソコンを接続するケーブルも自作しないといけません。まず、それから説明します。

接続ケーブルの制作
マイコンボードとライターを接続するＣＮ４に、シリアル通信用の信号が出ています。これを使ってパソコンと接続します。ライターボードを改造する方法もあるかも知れませんが、別に作ったほうが簡単でしょう。
まず、回路図を示します。



シリアルケーブルの図


シリアルケーブル回路図

マイコンボードに４つあるシリアルポートのうちのチャネル０を使います。
パソコン側は、ハイパーターミナルなどの通信ソフト（ターミナルソフト）を使います。

はい、それではプログラムを実行する手順です。

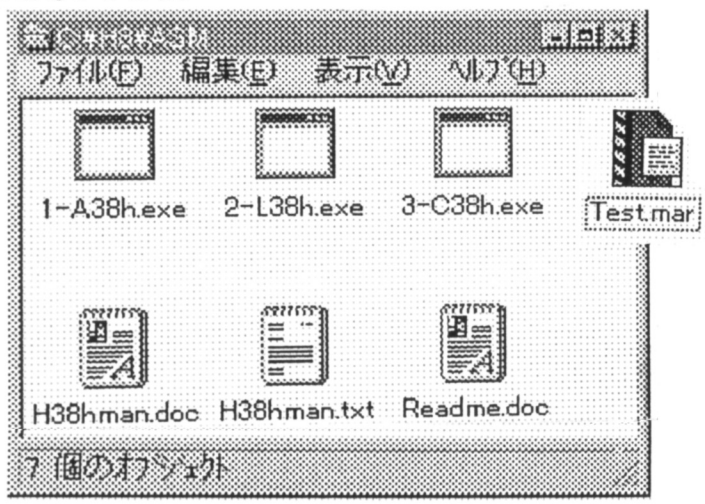
1) プログラムを入力する
リストを次に示します。
これを、「メモ帳」や「ワードパッド」、あるいはワープロなどで入力し、テキスト（テキスト文書形式）で保存します。その時のファイル名を「Test.mar」とします。

「;」から右側は注釈です。その部分はこのとおりでなくてもかまいませんが、ほかの部分は、一応このまま入力しておいてください。

```
        .heading        'Test for H8/3048 98.01.16'
        .print  src,nocref,nosct,list
;------------------------------
;       Simple In/Out for H8/3048
;       H8/3048 SingleChip Advanced Mode
;------------------------------
;
        .cpu 300ha:20           ; H8/300H Adavanced Mode, 20 bit addr
;
;
;------------------------------
;       reset vector table
;------------------------------
        .section rom,data,locate=h'0
;
        .data.l  reset
;
;------------------------------
reset:
        mov.l   #stack,sp
        bsr     inisci
loop:
        bsr     inch
        bsr     outch
        bra     loop
;
;------------------------------
;       inisci  InitSCI
;------------------------------
inisci:
        mov.b   #0,r0l
        mov.b   r0l,@sci0scr    ; clear all flags
        mov.b   #0,r0l
        mov.b   r0l,@sci0smr    ; Ascnc, 8bit, NoParity, (Even), stop1, 1/1
        mov.b   #51,r0l
        mov.b   r0l,@sci0brr    ; 9600 baud (CPU=16MHz)
        mov.w   #280,r0         ; wait 1 bit time (1/9600 sec)
inisc1: dec.w   #1,r0
        bne     inisc1
        mov.b   #h'30,r0l
        mov.b   r0l,@sci0scr    ; scr=0011 0000 (TE=1,RE=1)
        mov.b   @sci0ssr,r0l    ; Dummy Read
        mov.b   #h'80,r0l
        mov.b   r0l,@sci0ssr    ; Clear Error Flag (TDRE=1)
        rts
;------------------------------
;       inch to r0l
;               if err then carry = 1
;------------------------------
inch:   bsr     inchxx          ; to upper
        cmp.b   #'a',r0l
        blo     inch2
        cmp.b   #'z',r0l
        bhi     inch2
        and.b   #h'df,r0l       ; 'a' (0x61) -) 'A' (0x41)
inch2:  rts
```

```
inchxx: mov.b   @sci0ssr, r0l
        and.b   #h'78,  r0l
        beq     inchxx          ; RDRF=ORER=FER=PER=0
        btst.b  #6,@sci0ssr:8
        beq     incher
        mov.b   @sci0rdr, r0l
        bclr.b  #6,@sci0ssr
        andc.b  #h'fe, ccr      ; clear carry
        rts
;
incher: mov.b   #h'80, r0l
        mov.b   r0l,@sci0ssr
        orc.b   #1, ccr
        rts
;-------------------------------
;       outch from r0l
;-------------------------------
outch:  btst.b  #7,@sci0ssr
        beq     outch
        mov.b   r0l,@sci0tdr
        bclr.b  #7,@sci0ssr
        rts
;
        .section        stk, stack, locate=h'ff400
stack:  .equ    $
;
ramend: .equ    h'fff0f
;
;-------------------------------
;       internal I/O defs       h'fff1c - h'fffff
;-- sci defs
        .section        io, data, locate=h'fffb0
sci0smr .res.b  1       ; h'fffb0               ; serial mode register (h'fffb0)
sci0brr .res.b  1       ; h'fffb1               ; serial bit rate register
sci0scr .res.b  1       ; h'fffb2               ; serial control register
sci0tdr .res.b  1       ; h'fffb3               ; serial TX data
sci0ssr .res.b  1       ; h'fffb4               ; serial status register
sci0rdr .res.b  1       ; h'fffb5               ; serial RX data
;
        .end
```

2) アセンブルする
Sample フォルダにある Test.mar をアセンブルします。どんなファイルが作られたかわかるように ASM フォルダの中にコピーしておきましょう。

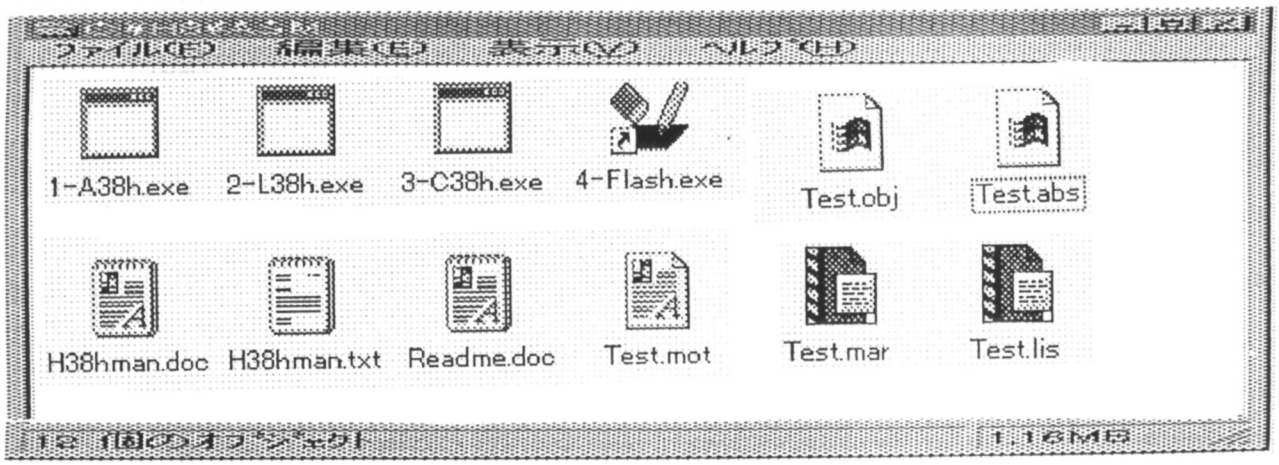


アセンブルは簡単です。「Test.mar」をドラッグし、アセンブラ「1-A38h.exe」にドロップします。（「Test.mar」の上でマウスボタンを押し、そのままずるずると「Test.mar」をひきずっていって、「1-A38h.exe」の上でマウスボタンをはなします）
この操作で「Test.lis」と「Test.obj」の２つのファイルが作られます。

3) リンクする
今度はリンクです。リンクはアセンブルで作られた（通常、複数の）「・・・.obj」ファイルをまとめて（リンクして）一つの「Test.abs」ファイルを作ります。
この操作もアセンブルと同様に、アセンブルで作られたファイル「Test.obj」をリンカ「2-L38h.exe」にドラッグアンドドロップします。
リンカは Test.abs を作成します。

4) 変換する
リンクしてできた「Test.abs」ファイルをモトローラSフォーマットに変換します。このフォーマットでなければ書き込むことができません。
この操作もアセンブルと同様に、「Test.abs」をコンバータ「3-C38h.exe」にドラッグアンドドロップします。
コンバータは「Test.mot」を作成します。

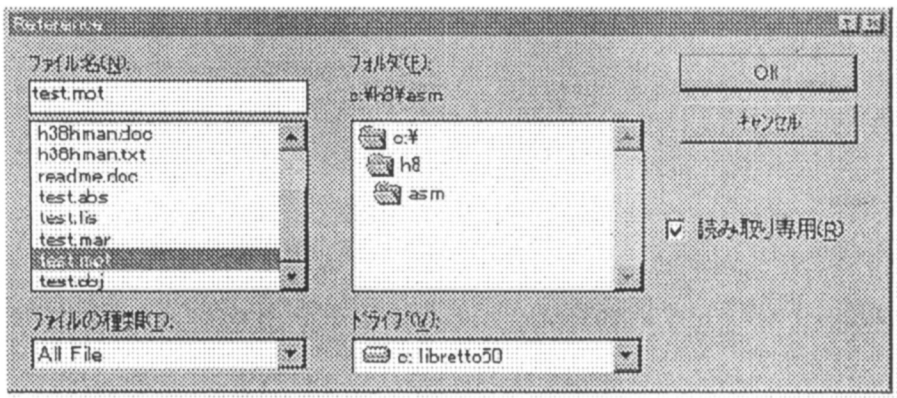
4) 書き込む
さて、CPUボードに書き込みます。
次の図のように、書き込みツール「4-Flash.exe」のショートカットをアセンブラなどと同じフォルダに入れておくと便利です。



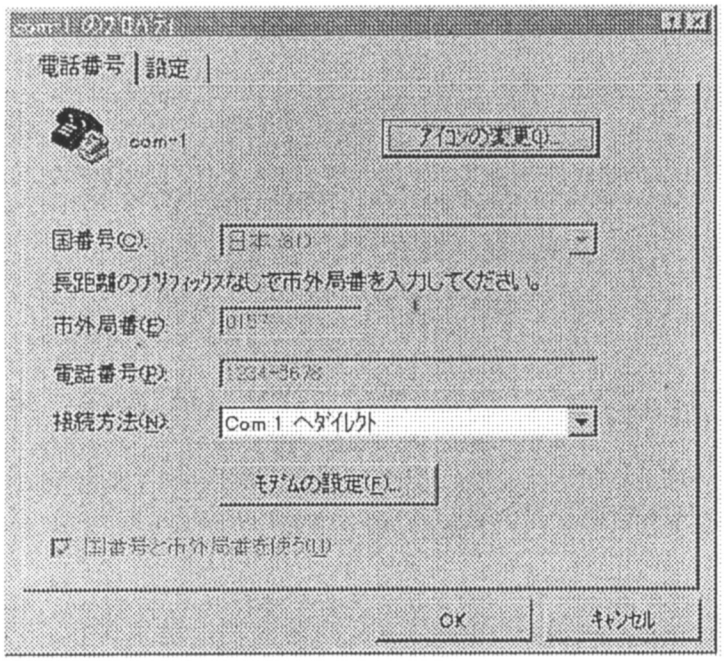
Flash.exe を開きます。
フラッシュメモリブロック情報ファイルは「3048inf」にしておきます。

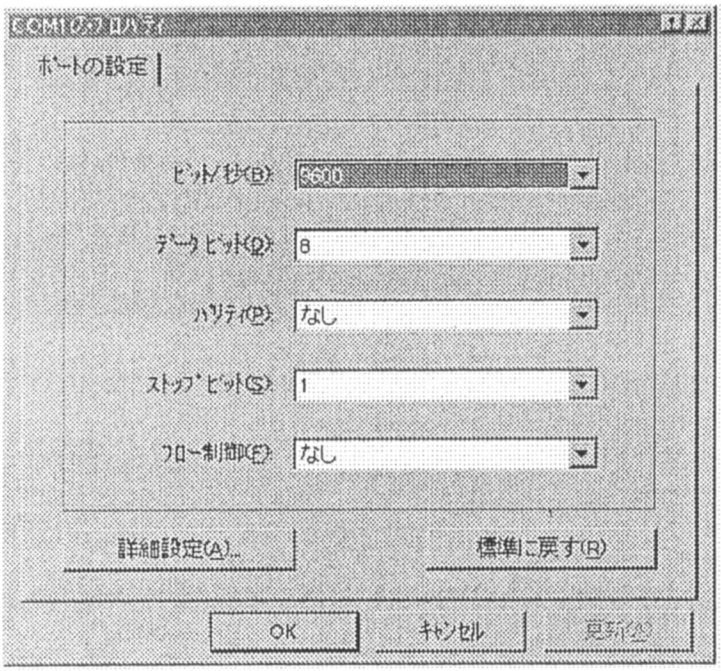
書き込むファイル Test.mot を選んで CPU ボードへ転送します。
ちいさなプログラムなので、転送は一瞬で終了します。

ライターソフトで表示されるダイアログを示しておきます。



5) 実行する
ハイパーターミナルん設定も必要ですから、先にそれをしてしまいましょう。
モデムの設定（?）が、9600 ボー８ビットで、「接続方法」は、接続するポートを各自のシステムにあわせて設定してください。私のマシンでは、Com1 が通信ポートになっています。





入力した文字がそのまま表示されればＯＫです。電源を切ると、キーボードから文字を入力しても、表示されなくなってしまうことを確認しておいてください。あ、そうそう、小文字は大文字に変換されます。この変換はＨ８がやっています。

めでたく動作しましたか？
アセンブル段階のエラーは、うちこみまちがいですから、ソースをよく見て訂正してください。
アセンブルは正常で、ソースもそのままなのに動作しない場合は、残念ながら、回路上の問題です。半田付け不良や、部品の向きなどを確認してください。回路が動作していないと、書き込みもできないかもしれません。

H8-3048 のハードウエアの基本
日立製作所から出されている資料を読むと次のように書かれています。
1）内部３２ビット構成
2）１６本の１６ビット汎用レジスタ
3）高速動作指向の簡潔で最適化された命令セット
4）16 M バイトのリニアなアドレス空間
5）多くの周辺機能
ITU: １６ビット５チャネルインテグレイテッドタイマ
TPC: プログラマブルタイミングパターンコントローラ
WDT: ウオッチドッグタイマ
SCI: シリアルコミュニケーションインターフェイス
ADC: 10 bit 8 ch アナログデジタル変換器
DAC: 8 bit 2 ch デジタルアナログ変換器
DMAC: ショートで 4 ch、ロングで 2 ch の DMA コントローラ
70 bit Input、8 bit Input I/O ポート
リフレッシュコントローラ
6）128 K Byte EEPROM（書き込み消去可能）、4 K Byte RAM

I／Oポートの信号端子と外部へのアドレス端子が共通なので、１６メガバイトのリニアなアドレス空間と７０ビットの入力ポートなどの周辺機能の両方を同時に使うことはできません。アドレス空間を１６メガバイトにすると入出力ポート数が減ってしまいます。

このキットの動作モードはモード７で、アドレス空間は１メガバイトになっています。１メガバイトとは、２０ビットで、H100000 から H1FFFFF となります。ここで「H1」はこの数値が１６進数であることをあらわしています。

アドレスをまとめると次のようになります。（H8/3048 シングルチップアドバンストモード・モード７）

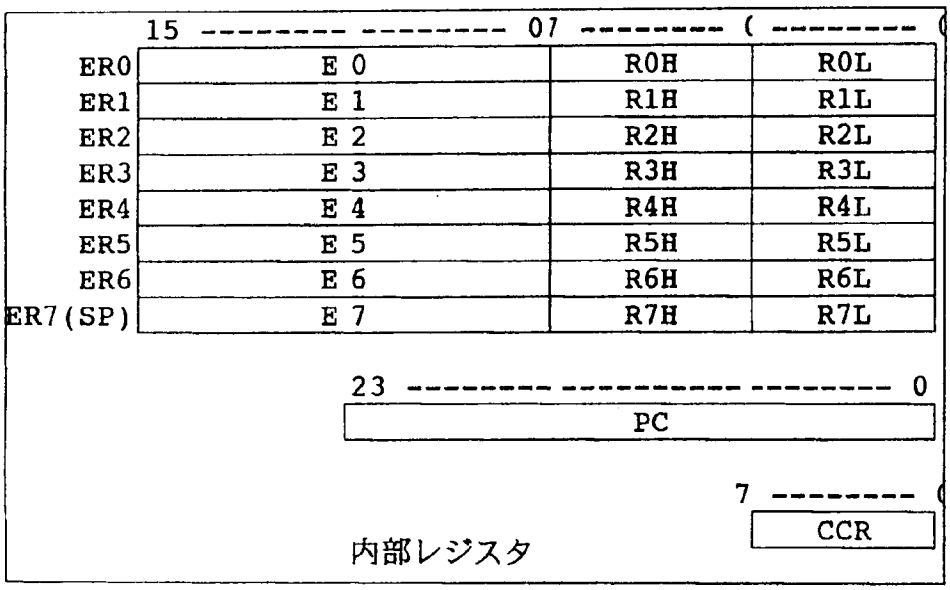
H100000 から H11FFFF までは１２８キロバイトのＲＯＭ。
H1FEF10 から H1FFF0F までは４キロバイトＲＡＭ。
H1FFF1C から H1FFFFF までは内部Ｉ／Ｏ。

＝＝＝＝＝＝＝＝ １６進数 ＝＝＝＝＝＝＝＝
１６進数の説明は必要ですか？
必要なさそうですが、簡単に説明しておきます。
１０で桁が一つあがる１０進数が絶対的なものだと思っていませんか？２進数とか、１６進数は、数学の世界だけで出てくる抽象的なものだといいうのは、はっきりって間違いです。
たまたまわれわれは１０進数を使っている、というのが正しい認識です。
で、コンピュータは１６進数で表現すると便利なところがあるのでよく使われます。１６進数では、１０進数で言うところの１０から１５までの文字が不足するので、そこをＡからＦまでのアルファベットを使っているのです。
１０進数と１６進数を区別するために、H8 のアセンブラでは、１６進数の頭に「H1」を付けます。CやＣ＋＋では「0x」を付けます。６８Ｋなどのアセンブラやパスカル（ちょっと記憶があいまい）では「$」を付けます。ＡからＦまでの文字は小文字でもかまわないのが普通です。
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

CPU の内部レジスタ
H8/3048 はＣＰＵとＩ／Ｏポートとタイマーなどの周辺回路が一つにまとまった部品です。
ここでは、ＣＰＵ内部の説明をします。ＣＰＵ内部にあって、プログラムを作る上で注意を払わなければならないのは汎用レジスタとコンディションコードレジスタ（ステイタスレジスタ）くらいです。それらのレジスタについて説明しておきます。

汎用レジスタ
８つの３２ビットのレジスタ ERn (n = 0..7) があります。ER0 から ER7 ですね。これら８つのレジスタは同じように使うことができますが、ER7 は特別にスタックポインタ（ＳＰ）として使いますから、プログラムを作るときは別扱いとします。

それぞれのレジスタ ERn は上位１６ビットと下位１６ビットの半分に分けて使うこともできます。このとき ERn は En と Rn の１６ビットレジスタになります。
この Rn はさらに上位８ビットと下位８ビットのレジスタ、RnH と RnL に分かれます。n は0から7までの数字が入ります。



内部レジスタ

汎用レジスタのほかに、２４ビットのプログラムカウンタ（ＰＣ）とコンディションコードレジスタ（ＣＣＲ）があります。

プログラムカウンタ（ＰＣ）
プログラムカウンタは、ＣＰＵが次に実行する命令のアドレスを示します。プログラムカウンタ相対アドレッシングで使われる以外は、レジスタの値として直接意識することはないでしょう。

コンディションコードレジスタ（ＣＣＲ）
もう一つのレジスタである８ビットのコンディションコードレジスタ（ＣＣＲ）はなにをしているのでしょう？これは大変重要なレジスタです。
計算の結果や、値チェックの結果がこのレジスタに反映されて、条件分岐で使われます。ＣＣＲ を直接アクセスすることはありませんが、どんなフラグがあって、この種類の命令ではどんなフラグが変化するのかということをある程度覚えておく必要があります。
命令の説明を見ると、その命令の処理内容といっしょに CCR の変化のしかたが書かれています。

7	6	5	4	3	2	1	0
I	UI	H	U	N	Z	V	C

CCR

ＣＣＲはＣＰＵの内部状態を示しています。
１ビットずつ説明しておきます。

I - ビット７：割り込みマスクビット
１にすると NMI 以外のハードウエア割り込み（例外処理）が禁止されます。これをマスクするといいます。例外処理に入ったときには１にセットされます。
UI - ビット６：ユーザービット/割り込みマスクビット
ユーザーが自由に使えるフラグビットです。LDC や ORC など、このビットを直接操作する命令以外では変化しません。
割り込みコントローラの設定で割り込みマスクビットになります。初期状態ではユーザービットです。
H - ビット５：ハーフキャリフラグ
加算や減算で発生する繰り上がり繰り下がりを示します。このチェックの対象となるのは演算レジスタのビット３です。ビット７の場合はキャリーフラグ（C）にその結果が現れます。
このフラグは ＢＣＤ 演算で使います。ＢＣＤ は４ビットを１０進の１桁として表現する方式です。

U - ビット４：ユーザービット
ユーザーが自由に使えるフラグビットです。LDC や ORC など、このビットを直接操作する命令以外では変化しません。
N - ビット３：ネガティブフラグ
加減算あるいは転送命令のあとで、その結果がマイナスなら１になります。マイナスというのは、２の補数表現での話ですから結局、その最上位ビットが１なら、このフラグは１になります。
Z - ビット２：ゼロフラグ
加減算あるいは転送命令で、その結果が０なら１になります。
V - ビット１：オーバーフローフラグ
加減算でオーバーフローが発生したら１になります。オーバーフローとキャリーとは違うの？とお考えのあなた、そうなんです、これは説明がめんどうです。
２の補数表現での桁あふれがオーバーフローです。プラスの数とプラスの数を加えてマイナスになるとオーバーフローです。このときキャリーは発生しません。８ビットであれば２の補数表現ではプラス側は最大１２７まで、マイナス側は -１２８までです。２の補数ではなく０から２５５までの値を取ると考えることもできます。１２０＋４０を計算するとその結果は１６０となり、０から２５５の８ビットの範囲に入っていますからキャリーは発生しません。しかし、２の補数表現で見たときには最大値１２７を越えていますから-３２となり、プラスとプラスを足してマイナスになっていますからオーバーフローフラグが１にセットされます。
同様にマイナスからマイナスを引いてプラスになってもオーバーフローです。
C - ビット０：キャリーフラグ
加減算での繰り上がり繰り下がりで１になります。シフトやローテートでも変化します。

プログラムを作る上で、その他に必要なハードウエアの知識
スタック
スタックとは何か・・・
サブルーチンの戻り番地を保存したり、一時的な作業領域として使われます。プログラムの最初に必ずスタックポインタを設定し、スタック領域を確保しておかなければなりません。
汎用レジスタのＥＲ７を使います。プッシュとポップはＣ風に書くと次のようになります。スタックポインタ（ＳＰ）は奇数にしてはいけません。奇数にするとどうなるのか、確認しておく必要はあります。

```
プッシュ        *(--ER7) = x;
ポップ          x = *(ER7++);
```

プッシュは、スタックにある値を保存するときで、ポップは、スタックからとりだすときの動作をいいます。
この CPU では、スタックポインタを更新してから保存します。スタックからとりだすときは、スタックポインタからとりだしてからポインタを更新します。
エンディアン
上位データが最初のアドレスになります。
たとえば H'1234 の１６ビットデータを H'1000 番地に書き込むと H'1000 番地は H'12 、H'1001 番地は H'34 となります。１６ビットのワードデータ、それに３２ビットのロングワードデータは先頭アドレスが偶数になるように配置します。奇数番地に対してワード・ロングワードデータをアクセスするとエラーではなく、最下位ビットが０として扱われます。エラーにはならないので、バグの温床となりそうです。
ベクタ
０番地から例外処理ベクタテーブルが置かれています。
すぐに使うテーブルは、０番地から４バイトのリセットベクタです。ここにプログラムの開始番地を設定します。これを正しく設定しないとプログラムが動きません。
スタックポインタの値はベクタテーブルにはありませんから、プログラムの先頭でスタックポインタ設定をしなければなりません。

アセンブラの注意事項
データ領域と align
CPU に対する命令は、必ず偶数番地から始まる必要があります。ところが、途中に文字列データなどが入ると、そのあとの命令が偶数であるとは限らなくなります。そこを、必ず偶数にしてしまうのが、.align です。
section で指定できる align は、そのアドレスだけで機能します。そのセクション全体に有効というわ

けではありません。
そのため、偶数番地から命令を始めようとすると、そのつど .align=2 とやる必要があります。

.datab が使えなませんでした。その代わりに .res を使いましょう。
次に、文字列データを確保する場合などの例を示します。参考にして下さい。

```
        .cpu 300ha
        .section a,data,locate=h'0
bega:   .data.l start
;
        .section b,code,locate=h'0400
        .org    h'0400
;
start:
        mov.l   @myData,er0
        mov.b   @byteD,r0l
        bra     start

        .section c,data,align=2
myData: .data.l h'12345678
byteD:  .data.b h'a
        .align 2
wordD:  .data.w h'b
charD:  .data.b 'a','b','c','d','e','f','g'
strD:   .sdata  "string data"(h'0)
        .align 2
longD:  .data.l h'9abcdef

enda:   .res    1
        .end
```

全体の構成
ハードウエアの内部構造の説明
内蔵周辺モジュールについて簡単に説明します。

割り込みコントローラ
割り込みコントローラで割り込みの許可／禁止、優先順位を決めることができます。
バスコントローラ
使わない？
リフレッシュコントローラ
DRAM リフレッシュコントローラです。使いませんね。
DMA コントローラ (DMAC)
メモリとメモリの間、あるいはメモリと１／Ｏの間でデータを転送します。これは面白そうなので使ってみたいですね。
１／Ｏポート
１０本の入出力ポートと、１本の入力ポートがあります。

#	入出力	入力プルアップ	出力	LED	シュミット
1	8入出力		1-TTL, 90pF	OK	
2	8入出力	ON/OFF	1-TTL, 90pF	OK	
3	8入出力		1-TTL, 90pF		
4	8入出力	ON/OFF	1-TTL, 90pF		
5	4入出力	ON/OFF	1-TTL, 90pF	OK	
6	7入出力		1-TTL, 90pF		
7	8入力専用		1-TTL, 90pF		0-7
8	5入出力		1-TTL, 90pF		0, 1, 2
9	6入出力		1-TTL, 30pF		
A	8入出力		1-TTL, 30pF		
B	8入出力		1-TTL, 30pF	OK	0-3

アドレスやデータ、割り込み入力にシリアル入出力などを兼用しています。
このキットはモード７なので、アドレスやデータは外部に出ませんから、それらの信号は入出力信号ラインとして使えますが、割り込みやその他の内蔵周辺モジュールとの兼用端子はどこかで切り替える必要があります。
また、すべてが８ビットというわけではありません。

その他の説明

割り込み・ベクタテーブル
リセット、割り込み、トラップ命令を例外処理と呼びます。
リセットは最も優先度の高い例外処理です。
割り込みには、その信号線が外に出ている「外部」と、内蔵周辺モジュールに接続されている「内部」の２種類あります。
外部端子からの割り込みはマスク不可能な割り込みＮＭＩ１本、マスク可能な割り込みＩＲＱ６本です。

内部割込みは内蔵周辺モジュールから発生する割り込みです。その数は３０です。内蔵周辺モジュールからの割り込み要因は、インターバルタイマやDMAC、シリアルインターフェイス、A/Dなどです。
割り込みコントローラで割り込みの許可／禁止、優先順位を決めることができます。
例外処理が発生したときには、現在処理中の命令終了とともに決められた番地へ処理を移します。０番地から始まるベクタテーブルには例外ごとの番地を設定します。

１６ビットインテグレーテッドタイマユニット（ITU）
５チャネルの１６ビットタイマから構成される１６ビットインテグレーテッドタイマユニットです。と言われてもなんだかわかりません。何ができるのかというと、最大１２種類のパルス出力、または最大１０種類のパルス入力処理。
ある周期で割り込みを発生することなどが可能になります。

プログラマブルタイミングパターンコントローラ（TPC）
これもなんだかよく分かりませんが、１６ビットインテグレーテッドタイマユニットの動作をもとに、いろいろな（ビット？）パターンで出力可能のようです。

ウオッチドッグタイマ（WDT）
ウオッチドックタイマはシステム監視に使われます。システムが定期的にウオッチドッグタイマにアクセスすることでウオッチドッグタイマはシステムが正常に動作していると判断します。しかし、ある時間が経ってもシステムからのアクセスがない場合は、システムが正常に動作していないとし、リセット信号を発生します。
H8-3048 に内蔵しているウオッチドッグタイマは、このほかに通常のインターバルタイマ機能も持っています。

シリアルコミュニケーションインタフェイス（SCI）
RS-232C や 422 などのシリアル通信用のインタフェイスで、２チャネル分用意されています。

スマートカードインタフェイス
シリアルコミュニケーションインタフェイスを使ったＩＣカードインタフェイスです。

Ａ／Ｄ変換器
８チャネルのアナログ入力を持つ１０ビット A/D 変換器です。Ａ／Ｄ変換器はアナログ信号をデジタル信号に変換します。気温や湿度、明るさなどのアナログ信号はそのままではコンピュータは処理できません。Ａ／Ｄ変換器を使ってディジタル信号に変換する必要があります。
この８本の入力端子はＩ／Ｏポート７の入力端子と兼用していますから、ポート７のディジタル信号入力とアナログ入力とは同時に使うことができません。
16 MHz クロックで変換時間は最大 8.4 μＳです。

Ｄ／Ａ変換器
２チャネルの８ビットＤ／Ａ変換器です。ディジタル信号をアナログ信号に変換します。
変換時間は最大１０μＳです。

ベクタテーブル一覧

#	Addr	*1: Reserved
0	h'00000	Reset
1	h'00004	*1
2	h'00008	*1
3	h'0000C	*1
4	h'00010	*1
5	h'00014	*1
6	h'00018	*1
7	h'0001C	NMI
8	h'00020	Trapa #0
9	h'00024	Trapa #1
10	h'00028	Trapa #2
11	h'0002C	Trapa $3
12	h'00030	IRQ0
13	h'00034	IRQ1
14	h'00038	IRQ2
15	h'0003C	IRQ3
16	h'00040	IRQ4
17	h'00044	IRQ5
18	h'00048	*1
19	h'0004C	*1

#	Addr	*1: Reserved
20	h'00050	WOVI Interval Timer
21	h'00054	CMI Compare Match
22	h'00058	*1
23	h'0005C	*1
24	h'00060	IMIA0 CompMatch/InpCapA0
25	h'00064	IMIB0 CompMatch/InpCapB0
26	h'00068	OVI0 Over Flow 0
27	h'0006C	*1
28	h'00070	IMIA1 CompMatch/InpCapA1
29	h'00074	IMIB1 CompMatch/InpCapB1
30	h'00078	OVI1 Over Flow 1
31	h'0007C	*1
32	h'00080	IMIA2 CompMatch/InpCapA2
33	h'00084	IMIB2 CompMatch/InpCapB2
34	h'00088	OVI2 Over Flow 2
35	h'0008C	*1
36	h'00090	IMIA3 CompMatch/InpCapA3
37	h'00094	IMIB3 CompMatch/InpCapB3
38	h'00098	OVI3 Over Flow 3
39	h'0009C	*1
40	h'000A0	IMIA4 CompMatch/InpCapA4
41	h'000A4	IMIB4 CompMatch/InpCapB4
42	h'000A8	OVI4 Over Flow 4
43	h'000AC	*1

#	Addr	*1: Reserved
44	h'000B0	DEND0A DMAC
45	h'000B4	DEND0B DMAC
46	h'000B8	DEND1A DMAC
47	h'000BC	DEND1B DMAC
48	h'000C0	*1
49	h'000C4	*1
50	h'000C8	*1
51	h'000CC	*1
52	h'000D0	ERI0 SCI 0 RX Error
53	h'000D4	RXI0 SCI 0 RX Data Full
54	h'000D8	TXI0 SCI 0 TX Data Empty
55	h'000DC	TEI0 SCI 0 TX End
56	h'000E0	ERI1 SCI 1 RX Error
57	h'000E4	RXI1 SCI 1 RX Data Full
58	h'000E8	TXI1 SCI 1 TX Data Empty
59	h'000EC	TEI1 SCI 1 TX End
60	h'000F0	ADI AD End
61	h'000F4	
62	h'000F8	
63	h'000FC	

『AKI-H8/3048F（フラッシュメモリ内蔵）
超高性能マイコンボード』

応用編

STEP 1.　パラレルＩ／Ｏポートから出力

H8/3048 には１０個のパラレルＩ／Ｏポートを持っています。そのうちの一つは入力専用ですが、ほかは入出力兼用です。
ここでは、ポート１に接続された８つのＬＥＤを順番に点滅させるプログラムを作ります。

#	入出力	入力プルアップ	出力	LED	シュミット
1	8入出力		1-TTL, 90pF	OK	
2	8入出力	ON/OFF	1-TTL, 90pF	OK	
3	8入出力		1-TTL, 90pF		
4	8入出力	ON/OFF	1-TTL, 90pF		
5	4入出力	ON/OFF	1-TTL, 90pF	OK	
6	7入出力		1-TTL, 90pF		
7	8入力専用		1-TTL, 90pF		0 - 7
8	5入出力		1-TTL, 90pF		0, 1, 2
9	6入出力		1-TTL, 30pF		
A	8入出力		1-TTL, 30pF		
B	8入出力		1-TTL, 30pF	OK	0 - 3

１、準備
８つのＬＥＤを順番に一つずつ点灯させたり、一つだけ点灯させて、その場所を順番に変えたりします。

プログラムを作る前にはっきりさせておかなければならないことを書いておきましょう。
（１）点灯する場所の指定方法
ＣＰＵのレジスタをシフトさせながら、そのレジスタを LED が接続されたポートに出力することで、点灯する場所を変えることもできますが、点灯する順番や個数を簡単にかえることはできません。ここでは、点灯場所データをあらかじめ用意することにします。そのデータを修正することでプログラムを変えずに、光らせかたを変更できます。
（２）Ｉ／Ｏポートの初期化方法
ポート１を出力ポートに設定します。
ポート１は入出力ポートとしてしか使えませんが、１０個のＩ／Ｏポートの中にはタイマーポートになったり、Ａ／Ｄポートになったりするものもあります。
プログラムのはじめの部分でポート１を出力ポートに設定する必要があります。
（３）ＬＥＤの接続方法
説明書によるとポート１は外部回路なしで、ＬＥＤを接続し点灯させることができます。
ＬＥＤの接続方法ですが、出力ポートとグラウンドの間にＬＥＤを接続し、ポートをＨ（５ボルトくらい）にしたときに点灯する方式と、ポートと５ボルト電源の間にＬＥＤを接続して、ポートをＬ（０ボルトくらい）にしたときに点灯する方式の２種類あります。通常は後者の５ボルト電源とポートの間にＬＥＤを接続し、ポートに電流を流し込む方式が用いられます。
０を書き込むと LED が点灯します。
（４）タイミングの取り方
たとえば、ＬＥＤを左から順番にひとつずつ点灯させて、点灯している場所が左から右へずれていくようにするプログラムを作るとします。プログラムの処理手順としては、まず、全部消去してから、一番左側を点灯させる。そこを消して、一つ右を点灯させる。さらに、そこを消してもう一つ右を点灯させる。といったことを繰り返すことで、順番に点灯しているように見えます。
ひとつのＬＥＤが点灯し、次のＬＥＤに移るまでの時間が１秒程度になるようにプログラムを作ります。

その時間の決め方は、何もしないループを何度もくり返すことで、時間かせぎをします。

２、ソースリスト
次にソースリストを示します。
最初の .data.1 rsv はベクタエリアです。今は使っていませんが、将来的に使うかもしれませんから、入れておいてください。最低限最初の .data.1 reset が必要です。ほかの .data.1 は省略してもかまいません。
プログラム本体は reset: から始まって１ページくらいの範囲です。

tabl: のデータを変えると、点灯の仕方が変わります。「1」になっているところが点灯しますから、「1」を2ヵ所に増やすと点灯するところも2ヵ所になります。
また、wait: にある mov.l #h'100000,er0 の「100000」を小さくすると速く、大きくするとゆっくりとした移動速度になります。いろいろ変えてみてください。

プログラムは「reset:」から始まります。0番地がリセットのベクタアドレスになっていて、リセット時には0番地に書き込まれている値を番地として、そこから処理が実行されます。

プログラムの最初にするべきことは、スタックポインタの初期化です。スタックはサブルーチンへ移動したときに、戻るアドレスを保存したり、レジスタの値を保存したり、一時的な作業領域にすることもあります。

loop: からはループになります。LED の点灯処理をいつまでも繰り替えします。

サブルーチンを説明しておきます。
initio: I/O ポートの初期化です。ポート1を出力に設定しています。
wait: だいたい1秒くらい、16進で 100000 回ぐるぐるまわって時を無駄に過ごします。

最後に I/O ポートの値を定義しているところがあります。その後ろがスタック領域の確保です。

```
.heading 'LED ON/OFF for H8/3048 LED.mar'
        .print  src,nocref,nosct,list
;------------------------------
;       H8/3048 SingleChip Advanced Mode
;
;------------------------------
;
        .cpu 300ha:20    ; H8/300H
                         ; Adavanced Mode,
                         ; 20 bit addr
;
;       Registers
;------------------------------
;       reset vector table
;------------------------------
        .section rom,data,locate=h'0
;
        .data.l reset
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l nmi
        .data.l trapa0
        .data.l trapa1
        .data.l trapa2
        .data.l trapa3
        .data.l irq0
        .data.l irq1
        .data.l irq2
        .data.l irq3
        .data.l irq4
        .data.l irq5
        .data.l rsv      ; 18
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv      ; 30
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv      ; 40
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv      ; 50
        .data.l rsv
        .data.l rsv
```

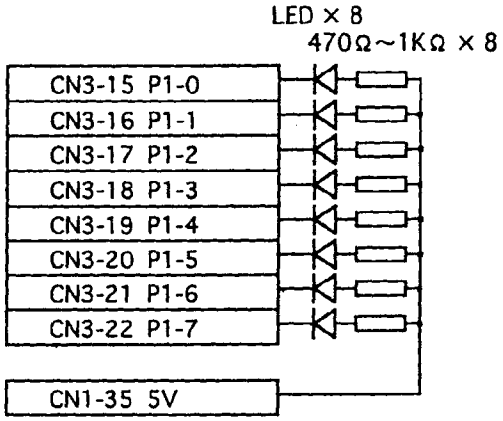
```
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv     ; 60
        .data.l rsv
        .data.l rsv
        .data.l rsv
;
        .page
;
;       .section rom,code,locate=h'00100
;-------------------------------
;       power on reset
;-------------------------------
reset:
        mov.l   #stack,sp
        bsr     initio
begin:  mov.l   #tabl,er1       ; 点灯データ表
        mov.w   #14,r2          ; 点灯データ表
                                ; のデータ数
loop:   mov.b   @er1+,r0l
        not.b   r0l
        mov.b   r0l,@p1dr
        bsr     wait
        dec.w   #1,r2
        bne     loop
        bra     begin
;
tabl:   .data.b b'10000000
        .data.b b'01000000
        .data.b b'00100000
        .data.b b'00010000
        .data.b b'00001000
        .data.b b'00000100
        .data.b b'00000010
        .data.b b'00000001
        .data.b b'00000010
        .data.b b'00000100
        .data.b b'00001000
        .data.b b'00010000
        .data.b b'00100000
        .data.b b'01000000
;
initio: mov.b   #h'ff,r0l       ; ポート１を
                                ; 出力に設定
        mov.b   r0l,@p1ddr
        rts

wait:   mov.l   er0,@-sp
        mov.l   #h'100000,er0
wait1:  nop
        dec.l   #1,er0
        bne     wait1
        mov.l   @sp+,er0
        rts
;
;-------------------------------
;       trap and irq
;-------------------------------
;reset:
nmi:    rte
trapa0: rte
trapa1: rte
trapa2: rte
trapa3: rte
irq0:   rte
irq1:   rte
irq2:   rte
irq3:   rte
irq4:   rte
irq5:   rte
rsv:    rte
;
        .align  2
prgend: .equ    $
;
        .page
;-------------------------------
;
; internal I/O defs     h'fff1c - h'fffff
;
;-- io defs
        .section io,data,locate=h'fffc0
p1ddr:  .res.b  1       ; h'fffc0
p2ddr:  .res.b  1
p1dr:   .res.b  1       ; h'fffc2
p2dr:   .res.b  1
p2pcr:  .res.b  1
;
;


       .page
;-------------------------------
;
;       work area h'fef10 - h'fff0f
;
        .section work,data,locate=h'fef10   i
;
stkbtm: .equ    $
;
        .section stk,stack,locate=h'ff400   ;
stack:  .equ    $
;
ramend: .equ    h'fff0f
        .end
```

３、回路図
点灯させる８個のＬＥＤをポート１に接続します。
ポートへ電流を流すようにして、５Ｖ電源とＬＥＤの間に４７０Ωから１ＫΩの抵抗を挿入し、ＬＥＤへの電流を１０ミリアンペア以下にします。

ポート１のあるＣＮ３には５ボルトの電源が無いので、ＣＮ１の３５か３６ピンから取り出します。
この回路はほかのプログラムでも使おうと思います。



STEP 2.　タイマーを使う

パラレル I/O ポートからの出力では、LED を順番に点灯させました。
その例ではソフトでループを作って、ある一定時間待つことで、点灯速度を決めています。
その速度は、ループを構成しているコマンドのクロック数を計算することで求めることができます。しかし、その計算は難しいため、計算途中で挫折してしまうことになるでしょう。そこで、タイマーを使います。タイマーを使うことで正確な周期で処理をすすめることが可能になります。

「STEP 1.　パラレルＩ／Ｏポートから出力」で作成したプログラムを修正して、タイマーを使って、正確な周期で LED を点灯させてみましょう。

１、準備　タイマーをどう使えば良いのか
H8/3048 に内蔵されているタイマーは、正式には「１６ビットインテグレーテッドタイマユニット（ＩＴＵ）」といいます。
１６ビットのタイマーユニットが５つありますが、今回は一つだけを使います。機能もたくさんあるので、マニュアルもたくさんのページを使っています。

やりたいことは、ある値をタイマーに設定すると、その値に対応する時間が経過したことをタイマーが教えてくれる、というものです。タイマーにはＣＰＵのクロックが接続されているので、そのクロック周波数と設定する値（とタイマーのモード）からその値に対応する時間が（ＣＰＵの命令クロック数を求めるより）簡単な計算で分かります。

ある時間が経過したということをチェックする方法もいくつか考えられます。一番簡単なのは、タイマーのあるビットが変化するのをずーっと待っているというもの。ほかにすることが無いのならこれで十分です。普通は、ほかにも仕事をしながらタイマーのチェックもしなければならないという状況になるでしょう。そんなときには「割り込み」を使います。今回は割り込みを使わず、あるビットを監視しつづけるという方法を採用します。

２、タイマーの初期設定
思い通りにタイマーを使うためには、たくさんあるレジスタにそれなりの値を設定しなければなりません。それがタイマーの初期設定（あるいは初期化）です。

やりたいことを、もう一度書いておきます。
「ある値を設定すると、その値に対応した時間が経過したことをタイマーが教えてくれる。」
ここで、「ある時間が経過した」ということをタイマーはどんなふうにして調べるのでしょうか。ちょっと（２秒くらい）考えて出てきたのは次の２つです。

・設定された値を減らしていって、0になるときをチェックする
・設定された値と0から始まるカウンタの値を比較して、一致したことをチェックする

今回は、2つめの値を比較して一致したことをチェックするほうを使います。

レジスタがたくさんあるので、順番にメモしていきます。使うレジスタには◎印を付けておきます。たくさんあるので、◎印がついているところだけ読んでもかまいません。（1から5までのレジスタは5チャネルあるタイマーすべてに共通しています）

1）タイマスタートレジスタ（TSTR）◎使います！
ビット0（STR0）を0にするとタイマー0の動作停止、1で動作開始です。準備終了後、1にしましょう。
2）タイマシンクロレジスタ（TSNC）
使わないで置きます。
3）タイマモードレジスタ（TMDR）
PWM（パルス幅変調）モードなどを決めます。ここも使いません。
4）タイマファンクションコントロールレジスタ（TFCR）
チャネル3，4の相補PWMモードなどの設定を行います。何の事だか分からないので、さわらないでおきます。
5）タイマアウトプットマスタイネーブルレジスタ（TOER）
チャネル3，4の出力制御なので、使いません。
6）タイマアウトプットコントロールレジスタ（TOCR）
相補PWMモードなどなどです。さわりません。

7）タイマカウンタ（TCNT）
16ビットのレジスタです。これは、実際にタイマーがカウンタとして使っているレジスタのようで、カウント中の値を見ることができるのかも知れませんが、見てもしょうがないですから、ここもさわりません。
次のGRA，GRBと比較されて一致すると0にリセットすることが可能です。その時をどこかでチェックすることができれば、今回の目的は達成できます。
8）ジェネラルレジスタA，B（GRA，GRB）◎これだ！
各チャネル2本のジェネラルレジスタがあります。ここに値を設定し、前述したタイマカウンタと比較一致をチェックすればよいのです。
アウトプットコンペアレジスタとして用いましょう。そのときは、GRA・GRBとTCNTが比較され、一致するとTSR（後述）のIMFA／IMFBフラグが1になります。これですね。これを使えばいいのです。やっと見つけました。
9）バッファレジスタA，B（BRA，BRB）
使いません。
10）タイマコントロールレジスタ（TCR）◎これもだ！
TCR0を使いましょう。
ここで設定できることは何でしょうか。
・カウンタクリア要因（ビット6と5）
ビット6を0、ビット5を1にすると「GRAのコンペアマッチ／インプットキャプチャでTCNTをクリア」します。比較して一致したらカウンタを0にして、もう一度はじめからカウントし直す。ということで、この設定を使います。
・クロックエッジ（ビット4と3）
0で立ち上がり、1で立ち下がりです。どちらでもいいので、初期値の「0：立ち上がりエッジでカウント」にしておきましょう。
・タイマプリスケーラ（ビット2，1、0）
カウントする信号を内部のCPUクロックを使うのか、はたまた、外部の信号を使うのか、どちらにするか決めるところです。さらに、そのクロックをそのまま使うのか、何分の1かに分周して使うのかも設定します。
今回は、外部信号は使わず、内部CPUクロックを1／8して使うことにしましょう。
ビット2を0、ビット1と0を1にします。

結局設定する値は次のようになりました。ビット7（ここは0）を左にして・・・
「0 01 00 011」

１６進で表現すると h'23 となります。
１１）タイマＩ／Ｏコントロールレジスタ（ＴＩＯＲ）
前述のＧＲＡ／ＧＲＢをアウトプットコンペアレジスタにするか、インプットキャプチャレジスタにするかを決めます。
リセット後の初期状態で、希望するアウトプットコンペアレジスタになってますから、そのままにしておきます。
１２）タイマステータスレジスタ（ＴＳＲ）◎使います！
コンペアマッチの状態表示です。ここをチェックすることで、所定の時間が過ぎたことをチェックできます。
ビットＯのインプットキャプチャ／コンペアマッチフラグＡ（ＩＭＦＡ）をチェックし、１になったら時間が経過したことを意味します。このフラグは自分でＯにリセットする必要があります。
１３）タイマインタラプトイネーブルレジスタ（ＴＩＥＲ）
初期状態では割込み禁止です。そのままにしておきます。

ながながと書いてしまいました。
使うところだけをまとめておきましょう。まとめると少ない！

１）タイマスタートレジスタ（ＴＳＴＲ）：ビットＯ（ＳＴＲＯ）を１にする。
８）ジェネラルレジスタＡ（ＧＲＡＯ）：時間を設定する。
１０）タイマコントロールレジスタ（ＴＣＲＯ）：h'23 を設定する。
１２）タイマステータスレジスタ（ＴＳＲＯ）：ビットＯが１で時間経過。

初期設定の処理手順は、きっと次のようになります。
・ＴＣＲに h'23 を設定する
・ＧＲＡに時間（後で計算しましょう）を設定する
・ＴＳＴＲのビットＯを１にする。

時間経過チェック手順は次のようになるでしょう。
・ＴＳＲのビットＯが１になるのを待つ
・ビットＯが１になったら、Ｏにリセットして、決められた仕事をする
・最初の、ビットＯが１になるのを待つところへ戻る

ここで、決められた仕事というのは、今回はＬＥＤに表示するデータを更新する処理です。

プログラムを作る前に、レジスタの番地を調べないと行けません。
これがなかなか、どこを調べると書いてあるのかわからないんですね。

略称	番地	初期値	設定値
TSTR	h'ff60	h'e0	h'e1
TCR0	h'ff64	h'80	h'23
TSR0	h'ff67	h'f8	ビット０を０にする
GRA0H	h'ff6a	h'ff	設定値上位バイト
GRA0L	h'ff6b	h'ff	設定値下位バイト

３、タイマー設定値の計算
タイマーによって、一定周期でＬＥＤの点灯場所を更新します。
ここでは、１秒ごとに更新するものとして、その時に設定する値を計算で求めてみましょう。

タイマコントロールレジスタ（ＴＣＲ）に設定する値で、ＣＰＵのクロックである１６ＭＨｚを１／８した２ＭＨｚでカウントするようにしてあります。この２ＭＨｚの周期である５００ナノ秒がいくつあれば１秒になるのか、という問題です。
ま、単純に割り算すればいいわけで、１÷５００×１０-9＝２０００００ となります。１６進では h'1E8480 です。

この値をＧＲＡに設定します。が、しかし、ここで重大なことに気がつきました。ＧＲＡは３２ビットではなく、１６ビットですから、こんな巨大な値を書き込むことはできません。ざんねんがっくり。

１６ビットをフルに使ったとして、最大カウント周期は、およそ 0.03 秒になります。しかたがないので、0.02 秒で更新することにします。同様に計算すると、設定値は次の値になります。
０．０２÷５００×１０-9＝４００００＝ h'9C40

この０．０２秒を５回で０．１秒、５０回で１秒になります。タイマーのカウントアップが５０回発生したらＬＥＤを更新することにします。これで１秒ごとになります。

４，プログラム

「STEP 1.　パラレル I／Oポートから出力」で作ったプログラムのI／Oポート初期化をしている「initio」と、更新するまでの時間をつぶす「wait」を修正します。ほかはそのままでＯＫです。

１）初期化処理の修正

ちょっと前にも書きましたが、初期設定の処理手順は、次のようになります。
・ＴＣＲに h'23 を設定する
・ＧＲＡに時間（後で計算しましょう）を設定する
・ＴＳＴＲのビットＯを１にする。

これをアセンブラで書くとこうなります。イタリック部分は修正前の initio です。

```
initio: mov.b   #h'11,r0l          ; ポート１を出力に設定
        mov.b   r0l,@p1ddr
        mov.b   #h'23,r0l          ; タイマーの初期化
        mov.b   r0l,@tcr0
        mov.w   #h'9c40,r0
        mov.w   r0,@gra0
        mov.b   #h'e1,r0l
        mov.b   r0l,@tstr
        rts
```

２）時間つぶし処理の修正

時間経過チェック手順は次のようになります。
・ＴＳＲのビットＯが１になるのを待つ
・ビットＯが１になったら、０にリセットして、決められた仕事をする
・最初の、ビットＯが１になるのを待つところへ戻る

０．０２秒のタイマーカウントを５０回繰りかえす処理です。従来のプログラムは、時間つぶしのために１００００回何もしない命令（nop）を実行していました。それを５０回タイマーチェックを実行する、というように変更します。１００００を５０に、nop をタイマーチェックに変えるだけです。タイマーチェックはサブルーチンにして別に作りましょう。

＜＜旧wait＞＞

```
wait:   mov.l   er0,@-sp
        mov.l   #h'100000,er0
wait1:  nop
        dec.l   #1,er0
        bne     wait1
        mov.l   @sp+,er0
        rts
```

＜＜新wait＞＞

```
wait:   mov.l   er0,@-sp
        mov.l   #h'50,er0
wait1:  bsr     timchk
        dec.l   #1,er0
        bne     wait1
        mov.l   @sp+,er0
        rts
```

タイマーチェック（timchk）のサブルーチンはこうなります。
ＴＳＲのビット０が１になるまで待ち、１になったらＴＳＲのビット０を０にしてループを終了します。

```
<<新timchk>>
timchk: mov.l   er0,@-sp
timch1: mov.b   @tsr0,r0l
        and.b   #1,r0l
        beq     timch1
        and.b   #h'fe,r0l
        mov.b   r0l,@tsr0
        mov.l   @sp+,er0
        rts
```

3）タイマーのレジスタアドレス定義
実はもう少しやることがありました。
上のプログラムで、TSTR や、 GRA0 などを使いましたが、その値を定義していませんでした。次のようにします。.res.b を使いたかったのですが、番地が飛び飛びなので、今回は .equ を使いました。

```
;-- io defs
        .section        io,data,locate=h'fffc0
p1ddr:  .res.b  1       ; h'fffc0
p2ddr:  .res.b  1
p1dr:   .res.b  1       ; h'fffc2
p2dr:   .res.b  1
p2pcr:  .res.b  1
tstr:   .equ    h'fff60
tcr0:   .equ    h'fff64
tsr0:   .equ    h'fff67
gra0h:  .equ    h'ff6a
gra0l:  .equ    h'ff6b
gra0:   .equ    gra0h
```

5、リスト
全体のソースリストを示します。

```
.heading 'LED ON/OFF with TIMER LED.mar'
        .print  src,nocref,nosct,list
;------------------------------
;       H8/3048 SingleChip Advanced Mode
;------------------------------
        .cpu 300ha:20   ; H8/300H
                        ; Adavanced Mode,
                        ; 20 bit addr
;------------------------------
;       reset vector table
;------------------------------
        .section rom,data,locate=h'0
;
        .data.l reset
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l nmi
        .data.l trapa0
        .data.l trapa1
        .data.l trapa2
        .data.l trapa3
        .data.l irq0
        .data.l irq1
        .data.l irq2
        .data.l irq3
        .data.l irq4
        .data.l irq5
        .data.l rsv     ; 18
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
```

```
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv        ; 30
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv        ; 40
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv        ; 50
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv        ; 60
        .data.l  rsv
        .data.l  rsv
        .data.l  rsv
;
        .page
;
;       .section rom,code,locate=h'00100
;-------------------------------
;       power on reset
;-------------------------------
reset:
        mov.l   #stack,sp
        bsr     initio
begin:  mov.l   #tabl,er1       ; 点灯データ表
        mov.w   #14,r2          ; 点灯データ
                                ; 表のデータ数
loop:   mov.b   @er1+,r0l
        not.b   r0l
        mov.b   r0l,@p1dr
        bsr     wait
        dec.w   #1,r2
        bne     loop
        bra     begin
;
tabl:   .data.b b'10000000
        .data.b b'01000000
        .data.b b'00100000
        .data.b b'00010000
        .data.b b'00001000
        .data.b b'00000100
        .data.b b'00000010
        .data.b b'00000001
        .data.b b'00000010
        .data.b b'00000100
        .data.b b'00001000
        .data.b b'00010000
        .data.b b'00100000
        .data.b b'01000000
;
initio: mov.b   #h'ff,r0l       ; ポート１を
                                ; 出力に設定
        mov.b   r0l,@p1ddr
        mov.b   #h'23,r0l       ; タイマーの
                                ; 初期化
        mov.b   r0l,@tcr0
        mov.w   #h'9c40,r0
        mov.w   r0,@gra0
        mov.b   #h'e1,r0l
        mov.b   r0l,@tstr
        rts
;
wait:   mov.l   er0,@-sp
        mov.l   #h'50,er0
wait1:
        bsr     timchk
        dec.l   #1,er0
          bne     wait1
        mov.l   @sp+,er0
        rts
;
timchk: mov.l   er0,@-sp
timch1: mov.b   @tsr0,r0l
        and.b   #1,r0l
        beq     timch1
        and.b   #h'fe,r0l
        mov.b   r0l,@tsr0
        mov.l   @sp+,er0
        rts
;-------------------------------
;       trap and irq
;-------------------------------
;reset:
nmi:    rte
trapa0: rte
trapa1: rte
trapa2: rte
trapa3: rte
irq0:   rte
irq1:   rte
irq2:   rte
irq3:   rte
irq4:   rte
```

```
 irq5:   rte
rsv:     rte
;
         .align  2
prgend:  .equ    $
;
         .page
;------------------------------
;        internal I/O defs h'fff1c - h'fffff
;-- io defs
         .section io,data,locate=h'fffc0
p1ddr:   .res.b  1          ; h'fffc0
p2ddr:   .res.b  1
p1dr:    .res.b  1          ; h'fffc2
p2dr:    .res.b  1
p2pcr:   .res.b  1
;
```

```
tstr:    .equ    h'fff60
tcr0:    .equ    h'fff64
tsr0:    .equ    h'fff67
gra0h:   .equ    h'fff6a
gra0l:   .equ    h'fff6b
gra0:    .equ    gra0h
;
         .page
;------------------------------
;        work area h'fef10 - h'fff0f
         .section work,data,locate=h'fef10
stkbtm:  .equ    $
         .section stk,stack,locate=h'ff400
stack:   .equ    $
ramend:  .equ    h'fff0f
         .end
```

STEP 3.　A/D 変換器を使う

A/D 変換器からサーミスタを使い温度を入力して、LED に棒グラフのように表示させてみましょう。

1、どんなことができるのか
H8/3048 には、８つの入力チャネルを持つ１０ビットの A/D 変換器を持っています。
ここで、８つの入力チャネルとは何かというと、A/D 変換器自体は一つなのですが、８つの入力端子があって、そのうちのどの入力端子を選ぶか決めることができるようになっているということです。８つの入力端子を持つ入力切替器が A/D 変換器の前についているのです。
１０ビットの A/D 変換器というのは、アナログからデジタルに変換された値が、１０ビットの値であるということです。１０ビットというのは、2の１０乗、すなわち１０２４段階にわけられるということを意味します。

ここでは、サーミスタを使った回路の出力電圧をアナログ入力端子に加えます。サーミスタは温度によりその抵抗値が変化する電子部品なので、温度を入力するということです。出力は STEP 1 で作成したＬＥＤに棒グラフのように表示させましょう。

アナログからデジタルへの変換速度は１０μ秒以下ですから、普通の音声信号であればデジタルに変換可能です。このボードではメモリが少ないのですが、音声信号の変化分をビット単位で記録すれば少ないメモリ容量でもそこそこの長さの音声信号を記録することができるでしょう。それをＰＷＭなどで出力すれば、簡単な音声メモを作ることができそうです。

2、どう使えばよいのか
H8/3048 の A/D 変換器にはサンプルアンドホールド（Ｓ＆Ｈ，Ｓ／Ｈなど）がついています。Ｓ＆Ｈの働きを説明しておきます。アナログ信号をデジタルに変換するためには、ある時間が必要です。その時間内にアナログ信号が変化するとデジタル信号に正しく変換されないことがあります。そのため、変換中は一定のアナログ信号にしてしまうのがサンプルアンドホールド回路の役目です。

A/D 変換器は、通常、あるビットを変化させるなどで変換を開始すると、変換終了まで待つことになります。変換が終了すれば割り込みを発生したり、どこかのビットが変化します。そこで、デジタル信号が格納されているレジスタを読み込むのです。

８チャネルあるアナログ入力はのうち、チャネル0を使うことにします。
この A/D 変換器には単一モードとスキャンモードという２つのモードがあります。今回は、連続して変換するスキャンモードを使うことにしましょう。

3、Ａ／Ｄコンバーターの初期設定
タイマーほどではありませんが、A/D 変換器にもたくさんのレジスタがあります。
初期設定をどうするのか決める前に、ざっとレジスタの働きを見てみましょう。何か設定が必要なレジスタには◎印を付けておきます。

1）A／Dデータレジスタ A～D（ＡＤＤＲＡ～D）◎変換結果です
８チャネルのアナログ入力は４つのグループに分けられて、この１６ビットの読み出し専用レジスタにデジタル値として格納されます。
2）A／Dコントロール／ステータスレジスタ（ＡＤＣＳＲ）◎使います
チャネル選択や終了を知るためのフラグがあります。
・ビット７：A／Dエンドフラグ（ＡＤＦ）
１で変換終了をあらわします。クリアするためには、このビットに０を書き込みます。
・ビット６：A／Dインタラプトイネーブル（ＡＤＩＥ）
０で終了割り込み発生を禁止します。初期値が０ですから、０のままにしておきます。
・ビット５：A／Dスタート（ＡＤＳＴ）
１で変換開始です。単一モードなら、変換終了で自動的に０にリセットされますが、今回使おうとしているスキャンモードでは、変換終了でも０にはならず次の変換を実行します。０にリセットしない限り、変換を続けるのです。

・ビット４：スキャンモード（ＳＣＡＮ）
０で単一モード、１でスキャンモードです。１にします。
・ビット３：クロックセレクト（ＣＫＳ）
変換時間の切替です。０で２６６ステート、１で１３３ステートになります。速度が違うと何が違ってくるのか不明です。初期値は０ですから、０のままにしておきましょう。
・ビット２，１，０：チャネルセレクト（ＣＨ２～０）
すべて０にするとチャネル０です。初期状態ではチャネル０が選ばれています。ここもそのまま０になるようにします。

結局、このレジスタの初期設定では次の値を書き込みましょう。
０００１００００＝h' 10
スキャンモードでチャネル０を選択します。
3）A／Dコントロールレジスタ（ＡＤＣＲ）
外部から A/D 開始を制御することを許します。今回は外部からの制御は許しませんから、使いません。
初期値の０のままにしておきます。

まとめると、次のようになります。
＜＜初期設定＞＞
ＡＤＣＳＲに h' 10 を書き込みます。これは、スキャンモードでチャネル０を選んでいます。
＜＜変換開始＞＞
ＡＤＣＳＲのビット５のＡＤＳＴを１にします。
今回は、連続して変換しますから、この設定も初期設定に含めてしまいましょう。ということで、初期設定ではＡＤＣＳＲに h' 30 を書き込むことになります。
＜＜変換終了チェック＞＞
連続して変換するのでチェックするのは止めます。
STEP 2 で使ったタイマーをここでも使って、１秒ごとに変換結果をＬＥＤに表示することにしましょう
＜＜変換結果の読み込み＞＞
変換結果は１６ビットレジスタＡＤＤＲＡに格納されます。

４、結果の表示
８つのＬＥＤに棒グラフとして温度を表示します。AD の結果は上位１０ビットに格納されています。実際に使うのは、点灯する棒グラフの LED が８個なので、上位３ビットだけになります。

５、回路
サーミスタをつけて温度を測定しようと考えたのですが、回路がたいへんなので、アナログ入力にボリュームをつけることにしました。
＋５ボルトとグラウンドの間にボリュームをつけて、０から５ボルトまで変化する電圧をアナログ入力の０に接続します。
６、プログラム
今回使うレジスタの番地を表にしました。

略称	番地	初期値	設定値
ADDRAH	h'ffe0	h'00	変換結果H
ADDRAL	h'ffe1	h'00	変換結果L
ADCSR	h'ffe8	h'00	h'30
ADCR	h'ffe9	h'7e	（何もしない）

7、全リスト
構造は、いままでのプログラムとほとんど同じです。
initio: で I/O の初期化をして、ループでぐるぐる回ります。
今回は、AD と I/O のほかに、AD データを読み込むタイミングをきめるために Timer も使っています。さらに、デジタルに変換した値をシリアルポートから出力していますから、パソコンのターミナルソフトでその値を見ることができます。

begin: からのルーチンはデジタルに変換された値を読み込み、その上位８ビットを１６進２桁になおして、シリアルポートから出力します。さらに、jデジタル値の上位３ビット０～７まで８段階を、LED の棒グラフデータにテーブルを使って変換します。
その後、タイマーを使って１秒間待ってから、デジタル値を読み込むところへ戻り、ループになります。

サブルーチンの説明をしておきましょう。
initio: はタイマーや AD の初期化です。inisci: で、シリアルポートの初期化もしています。シリアルポートは９６００ボー、８ビット、パリティーなしに設定されます。

シリアルポート関連のサブルーチンが（使っていないものも含めて）たくさんあります。
inch: シリアルポートから１文字読み込みます。何かエラーがあればキャリーがセットされます。
inchxx: シリアルポートから１文字読み込む inch: を使っていますが、入力した文字が小文字なら大文字に変換しています。
outch: シリアルポートから１文字出力します。
out8h: レジスタ ER0 のデータを８桁の１６進数の文字になおして、シリアルポートから出力します。
out6h:, out4h:, out2h: は、out8h: と同様に、ER0、R0、R0L のデータを１６進文字列に変換してシリアルポートから出力します。
outhex: は、R0L の下位４ビットを１６進文字列に変換して、シリアルポートから出力します。
crlf: はシリアルポートに対して、改行コードを出力します。

```
.heading        'ADC for H8/3048 ADC.mar'
        .print  src,nocref,nosct,list
;------------------------------
;       H8/3048 SingleChip Advanced Mode
;
;------------------------------
;
        .cpu 300ha:20   ; H8/300H
                        ; Adavanced Mode,
                        ; 20 bit addr
;
;       Registers
;------------------------------
;       reset vector table
;------------------------------
        .section rom,data,locate=h'0
;
        .data.l reset
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l nmi
        .data.l trapa0
        .data.l trapa1
        .data.l trapa2
        .data.l trapa3
        .data.l irq0
        .data.l irq1
        .data.l irq2
        .data.l irq3
        .data.l irq4
        .data.l irq5
        .data.l rsv     ; 18
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv     ; 30
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
```

```
        .data.l rsv
        .data.l rsv     ; 40
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv     ; 50
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv
        .data.l rsv     ; 60
        .data.l rsv
        .data.l rsv
        .data.l rsv
;
        .page
;
;       .section rom,code,locate=h'00100
;-------------------------------
;       power on reset
;-------------------------------
reset:
        mov.l   #stack,sp
        bsr     initio

        bsr     crlf
        mov.b   #'*',r0l
        bsr     outch

begin:
        mov.w   @addra,r0       ; 9876 543
                                ; 10. ... .
                                ; 上位8ビ
        mov.b   r0h,r0l         ; .... ...
                                ; 9876 543
                                ; 下位8ビ
        bsr     out2h
        bsr     crlf

        shlr.b  r0l             ; .... ...
                                ; .... .98
                                ; 上位3ビ
                                ; だけにす
        shlr.b  r0l
        shlr.b  r0l
        shlr.b  r0l
        shlr.b  r0l
        mov.l   #tabl,er1       ; tabl[ADdata
                                ; >> 13]
        extu.w  r0
        extu.l  er0
        add.l   er0,er1
        mov.b   @er1,r0l

        not.b   r0l
        mov.b   r0l,@p1dr
        bsr     wait
        bra     begin
;
tabl:   .data.b b'10000000      ; 0
        .data.b b'11000000      ; 1
        .data.b b'11100000      ; 2
        .data.b b'11110000      ; 3
        .data.b b'11111000      ; 4
        .data.b b'11111100      ; 5
        .data.b b'11111110      ; 6
        .data.b b'11111111      ; 7
;
initio: mov.b   #h'ff,r0l       ; ポート1を
                                ; 出力に設定
        mov.b   r0l,@p1ddr

        mov.b   #h'23,r0l       ; タイマーの
                                ; 初期化
        mov.b   r0l,@tcr0
        mov.w   #h'9c40,r0
        mov.w   r0,@gra0
        mov.b   #h'e1,r0l
        mov.b   r0l,@tstr

        mov.b   #h'30,r0l       ; Init ADC
        mov.b   r0l,@adcsr

        bsr     inisci

        rts
;
wait:   mov.l   er0,@-sp
        mov.l   #h'50,er0
wait1:
        bsr     timchk
        dec.l   #1,er0
        bne     wait1
        mov.l   @sp+,er0
        rts
;
timchk: mov.l   er0,@-sp
timch1: mov.b   @tsr0,r0l
        and.b   #1,r0l
        beq     timch1
        and.b   #h'fe,r0l
        mov.b   r0l,@tsr0
        mov.l   @sp+,er0
        rts
```

```
;------------------------------
;       sci
;       inisci  InitSCI
;------------------------------
inisci:
        mov.b   #0,r0l
        mov.b   r0l,@sci0scr    ; clear
                                ; all flags
        mov.b   #0,r0l
        mov.b   r0l,@sci0smr    ; Asenc,
                                ; 8bit,
                                ; NoParity,
                                ; (Even),
                                ; stop1, 1/1
        mov.b   #51,r0l
        mov.b   r0l,@sci0brr    ; 9600 baud
                                ; (CPU=16MHz)
        mov.w   #280,r0         ; wait 1 bit
                                ; time
                                ; (1/9600 sec)
inisc1: dec.w   #1,r0
        bne     inisc1
        mov.b   #h'30,r0l
        mov.b   r0l,@sci0scr    ; scr=0011
                                ; 0000
                                ; (TE=1,RE=1)
        mov.b   @sci0ssr,r0l    ; Dummy Read
        mov.b   #h'80,r0l
        mov.b   r0l,@sci0ssr    ; Clear Error
                                ; Flag (TDRE=1)
        rts
;
;------------------------------
;       inch to r0l
;               if err then carry = 1
;------------------------------
inch:   bsr     inchxx          ; to upper
        cmp.b   #'a',r0l
        blo     inch2
        cmp.b   #'z',r0l
        bhi     inch2
        and.b   #h'df,r0l       ; 'a' (0x61) ->
                                ; 'A' (0x41)
inch2:  rts

inchxx: mov.b   @sci0ssr,r0l
        and.b   #h'78,  r0l
        beq     inchxx          ; RDRF=ORER
                                ; =FER=PER=0
        btst.b  #6,@sci0ssr:8
        beq     incher
        mov.b   @sci0rdr,r0l
        bclr.b  #6,@sci0ssr


        andc.b  #h'fe,ccr       ; clear carry
        rts
;
incher: mov.b   #h'80,r0l
        mov.b   r0l,@sci0ssr
        orc.b   #1,ccr
        rts
;
mskn8l  .equ    h'0001
mskbyt  .equ    h'00ff
;
;------------------------------
;       outch from r0l
;------------------------------
outch:  btst.b  #7,@sci0ssr
        beq     outch
        mov.b   r0l,@sci0tdr
        bclr.b  #7,@sci0ssr
        rts
;------------------------------
;       out8h (er0 hexdata)
;------------------------------
out8h:  mov.l   er0,@-sp
        mov.w   e0,r0
        bsr     out4h
        mov.l   @sp,er0
        bsr     out4h
        mov.l   @sp+,er0
        rts
;------------------------------
;       out6h (er0 hexdata)
;------------------------------
out6h:  mov.l   er0,@-sp
        mov.w   e0,r0
        bsr     out2h
        mov.l   @sp,er0
        bsr     out4h
        mov.l   @sp+,er0
        rts
;------------------------------
;       out4h (r0 hexdata)
;------------------------------
out4h:  mov.l   er0,@-sp
        mov.b   r0h,r0l
        bsr     out2h
        mov.l   @sp,er0
        bsr     out2h
        mov.l   @sp+,er0
        rts
;------------------------------
;       out2h (r0l hexdata)
;------------------------------
out2h:  mov.l   er0,@-sp
        shlr.b  r0l
        shlr.b  r0l
        shlr.b  r0l
        shlr.b  r0l
        bsr     outhex
        mov.l   @sp,er0
        bsr     outhex
        mov.l   @sp+,er0
```

```
        rts
;-------------------------------
;       outhex (r0l hexdata)
;-------------------------------
outhex: and.b   #msknbl, r0l
        or.b    #'0', r0l
        cmp.b   #'9', r0l
        bls     outhx2
        add.b   #7, r0l
outhx2: bsr     outch
        rts
;-------------------------------
;       CR/LF
;-------------------------------
crlf:   mov.l   er0, @-sp
        mov.b   #h'd, r0l
        bsr     outch
        mov     #h'a, r0l
        bsr     outch
        mov.l   @sp+, er0
        rts
;
;-------------------------------
;       trap and irq
;-------------------------------
;reset:
nmi:    rte
trapa0: rte
trapa1: rte
trapa2: rte
trapa3: rte
irq0:   rte
irq1:   rte
irq2:   rte
irq3:   rte
irq4:   rte
irq5:   rte
rsv:    rte
;
        .align  2
prgend: .equ    $
;
        .page
;-------------------------------
;       internal I/O defs h'fff1c - h'fffff
;-------------------------------
;-- sci defs
        .section iosci, data, locate=h'fffb0
sci0smr .res.b  1       ; h'fffb0
                        ; serial mode reg.
sci0brr .res.b  1       ; h'fffb1
                        ; serial bit rate reg.
sci0scr .res.b  1       ; h'fffb2
                        ; serial control reg.
sci0tdr .res.b  1       ; h'fffb3
                        ; serial TX data
sci0ssr .res.b  1       ; h'fffb4
                        ; serial status reg.
sci0rdr .res.b  1       ; h'fffb5
                        ; serial RX data
;
;-------------------------------
;-- io defs
        .section io, data, locate=h'fffc0
; Para I/O
p1ddr:  .res.b  1       ; h'fffc0
p2ddr:  .res.b  1
p1dr:   .res.b  1       ; h'fffc2
p2dr:   .res.b  1
p2pcr:  .res.b  1
;
; Timer
tstr:   .equ    h'fff60
tcr0:   .equ    h'fff64
tsr0:   .equ    h'fff67
gra0h:  .equ    h'fff6a
gra0l:  .equ    h'fff6b
gra0:   .equ    gra0h
;
; ADC
addrah: .equ    h'fffe0 ; h'00  変換結果H
addral: .equ    h'fffe1 ; h'00  変換結果L
addra:  .equ    addrah
adcsr:  .equ    h'fffe8 ; h'00  h'30
adcr:   .equ    h'fffe9 ; h'7e   (何もしない) 1
;
        .page
;-------------------------------
;       work area h'fef10 - h'fff0f
;
        .section work, data, locate=h'fef10
;
stkbtm: .equ    $
        .section stk, stack, locate=h'ff400
stack:  .equ    $
ramend: .equ    h'fff0f
        .end
```