ＯＡＫＳ16基礎テキスト

目次

はじめに

このテキストは、OAKS16 を使って開発を行なう為の入門書です。

テキスト内では、演習を行ないますので、開発に必要な環境を揃えておく必要があります。ソフトウエア（コンパイラ、デバッガ、フラッシュ ROM ライタ）のセットアップをなされていない方は、『OAKS16 をお使いになる方のために』を参考にしてソフトウエアのセットアップを行なってください。そして、『OAKS16 で TM をお使いになる方のために』を参考にして、TM にソフトウエアツール（エディタ、NC30、KD30、flashstart）を設定し、ツールバーから起動できるようにしてください。

このテキストは、**C 言語で開発をしていくことを前提**としていますが、組み込み用のマイコンでは初期設定はアセンブリ言語で記述しますので、その部分に関してはアセンブリ言語の説明を入れてあります。C 言語もアセンブリ言語も、使用している命令、文法等は、テキスト内で出来るだけ解説していますので、マイコンをはじめてお使いになる方でも、OAKS16 の開発を簡単に行なっていただけると思います。

1. マイコンとは

これから、OAKS16 を使って組み込みマイコンの開発について説明していきますが、その前にマイコンの簡単な説明をしておきましょう。

『マイコン』とは、『マイクロコンピュータ』(『マイクロコントローラ』と呼ばれることもあります）の略称です。この、『マイコン』を使う目的は、入出力の制御にあります。(そのために『コントローラ』と呼ばれます）制御を行なう為には、ハードウエア（H/W)、ソフトウエア（S/W）の二つの要素が必要になります。(図 1-1)

ハードウエア：マイコンに入力回路と出力回路を接続すること、これが H/W の準備です。これから説明していきますスイッチやＬＥＤ（発光ダイオード）などの入・出力装置はそのままマイコンに接続するだけではありません。抵抗など電子部品を接続したり、電源や GND に接続したりといった配線が必要になります。これらのこと考慮した、マイコンシステムの作成が必要になります。

ソフトウエア：マイコンを動かす為のプログラムの準備です。マイコンはプログラムを読み込み、その命令に従って動作を行ないますので、入出力制御のプログラムを作成してマイコンシステムの中に入れておく必要があります。

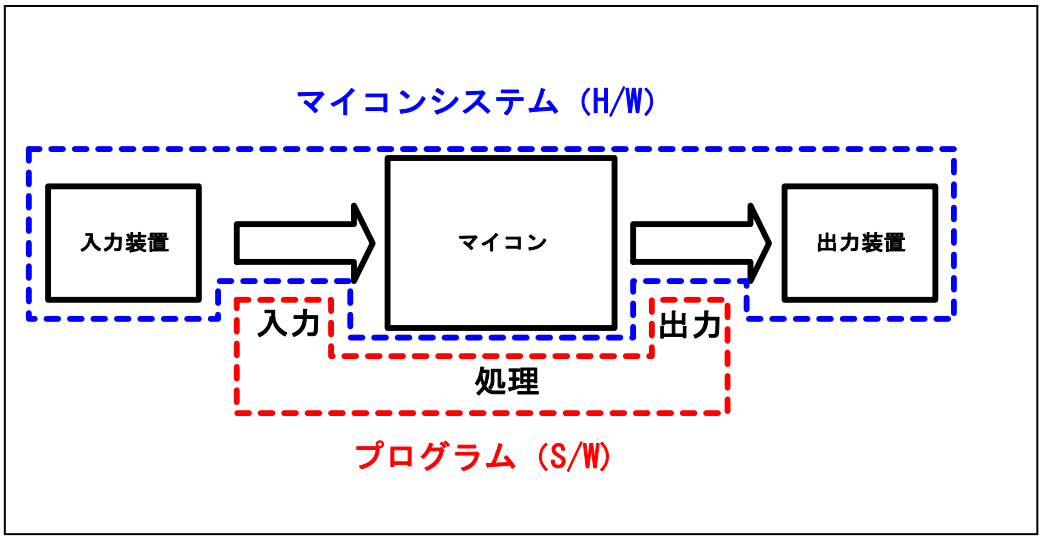


図 1-1

1.1. マイコンの基本構成

マイコンは普通、ＣＰＵ（central processing unit：中央演算処理装置）、メモリ（記憶装置）、周辺機器で構成されています。これらの構成要素は、バス（bus）と呼ばれる共通の信号線群で結ばれています。（図 1-2）

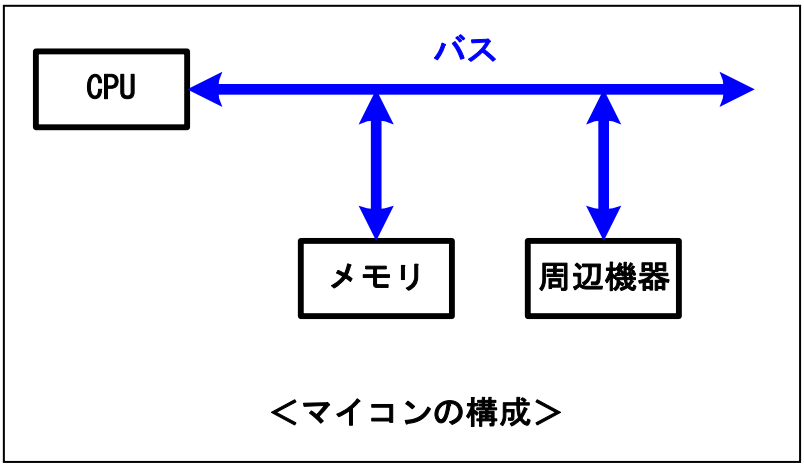


図 1-2

周辺機器は周辺インタフェース（peripheral interface）と呼ばれる回路ユニットを通してバスに結合されます。周辺機器は、大きく分けて入力機器と出力機器に分かれますので、周辺インタフェースも、入力インタフェース、出力インタフェースと呼ばれたりします。また、入力インタフェースと出力インタフェースをあわせた、入出力インタフェースを総称して I/O インタフェースと呼ぶこともあります。（図 1-3）

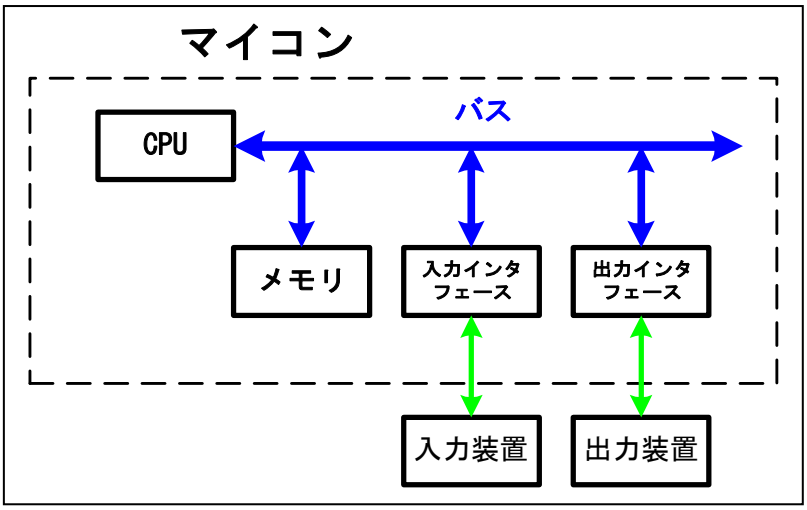


図 1-3

この **CPU、メモリ、I/O インタフェース**がマイコン構成の三要素です。

ＣＰＵ（中央演算処理装置）：マイコンの頭脳部分です。メモリに書き込んであるプログラムを順番に読み込み解読して、演算を行ったり、入出力の制御をしていきます。その実行過程において必要となるレジスタ（一時的に記憶する回路）やカウンタなどの補助回路も内蔵しています。

メモリ（記憶装置）：プログラムやデータをデジタル信号の形で書き込み保存しておく場所です。メモリは大きく分けてＲＯＭ（read only memory）とＲＡM(read write ramdom access memory)の 2 つの種類があります。

ＲＯＭは、ＣＰＵからの読み出し専用メモリで内容の変更が出来ません。ですが電源を切ってもデータが保持されるので、プログラムを書き込んでおくために使います。最初からプログラム等を工場で出荷時に書き込んでしまうマスク ROM、特別な書き込み装置・消去装置を使って何度か書き換えが出来る EPROM（Erasable and Programable ROM）、電気的に書き換え可能な EEPROM（Electrically Erasable and Programable ROM）などがあります。EEPROM の仲間には、フラッシュメモリという、ブロック単位でしか書き込みが出来ず、その単位ごとに一度消去しないと書き込みが出来ないというタイプのメモリもあります。

ＲＡＭは CPU から読み書きができるメモリですが、電源を切ってしまうとデータは保持されません。そのため、プログラムは書いておけませんが、プログラムの中で演算結果などの一時的なデータ保持領域として使用します。RAM には電源が供給されている間はデータを保持し続ける SRAM（static RAM）と、リフレッシュ回路というデータを保持する為の回路が必要な DRAM(dynamic RAM)があります。

最近は、FeRAM（Ferroelectric RAM）や MRAM(magnetic RAM：磁気によってデータを記憶する)など Flash メモリのように不揮発性メモリ（電源を切ってもデータが保持される）で SRAM のように書き換えが高速なメモリも登場し、ROM、RAM（プログラム専用メモリ、データ専用メモリ）という表現も曖昧になりつつあります。

Ｉ／Ｏインタフェース：マイコンと接続する外部機器とのやり取りをおこないます。

このマイコンの構成要素が一つのボード上に組み込まれているものを**ワンボードマイコン**、一つのチップ（IC:集積回路）に組み込まれているものを**ワンチップマイコン**と呼びます。（図 1-4）

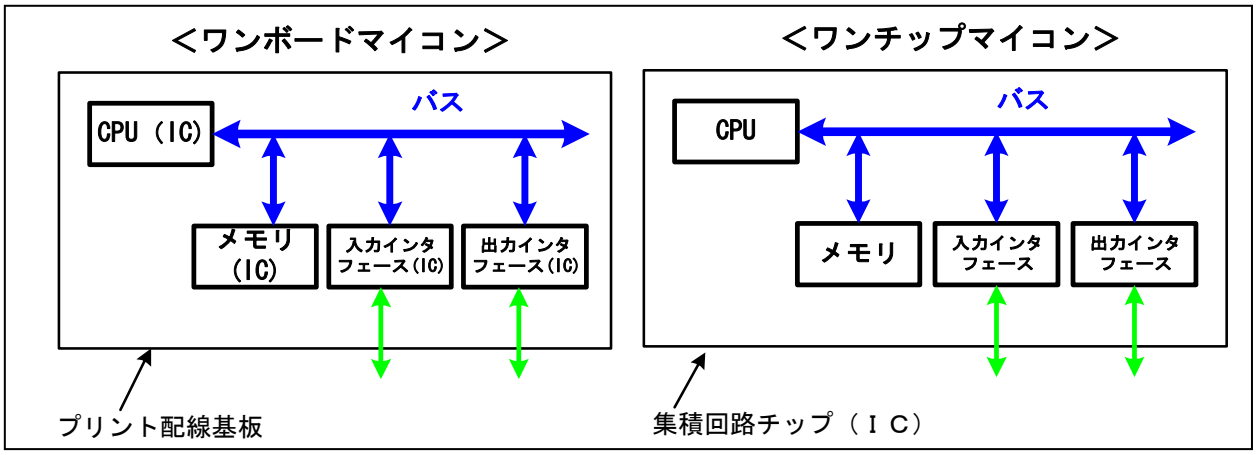


図 1-4

これらの構成要素は、バス（bus）と呼ばれる信号線で結線され情報の伝達が行なわれます。（バスは乗合バスを語源とし、同じ配線を色々な電気信号が時分割で共用することからその呼び名がつきました。）

バスは、その機能から、**アドレスバス**、**データバス**、**コントロールバス**の 3 つに分類されます。アドレスバス、データバスは複数まとめて 1 つの情報を表します。コントロールバスは、1 本で情報を伝達し、メモリや周辺機器からのデータの読み取りの際に使用する**リード(Read)信号**、書き込みのために使用する**ライト（write）信号**などがあります。（図 1-5）

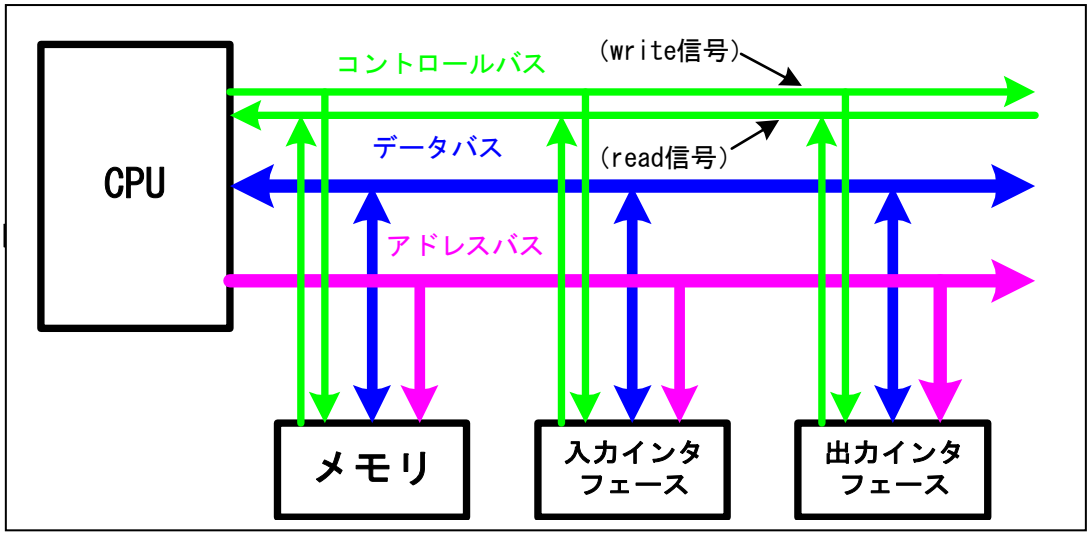


図 1-5

マイコンには、この基本構成要素のほかにどうしても必要な回路があります。(図 1-6)

クロック発生回路：マイコンは、クロックという、一定周期で“H”と“L”を繰り返す信号を基準として動作します。クロックは、ランニングをするときの笛の合図のようなものです。笛を吹く間隔が短ければ、足を前に出すタイミングも速くなり、スピードが上がります。マイコンもクロックの速度を上げればそれだけ処理速度は上がります。ですがもちろん限界というものがあるわけで、それをデータシートでしっかり確認して設計しないと、マイコンを壊す原因になります。クロック信号は、コンデンサと抵抗で発振させるＣＲ回路や、セラミック発振子・水晶発振子等を使った回路でマイコンのクロック入力端子から供給します。精度はＣＲ回路＜セラミック発振子＜クリスタル発振子の順でよくなります。マイコンによっては、クロック発生回路を内部に持っているものもあります。

リセット回路:マイコンには、電源を投入したとき、プログラムが暴走してしまったときなど、プログラムを一番最初から実行して欲しい時があります。そのための初期状態（リセット状態と呼びます。）にする為の回路です。マイコンは通常、一定時間、決まったレベルの電圧（信号）をリセット端子から入力すると、リセット状態になります。リセット時のマイコンの状態は CPU によって異なりますが、どのＣＰＵでもプログラムカウンタ（プログラムを順番に実行する為のカウンタ）を初期化し、プログラムを先頭から実行できるようにします。マイコンによっては、内部にリセット回路を持っているものもあります。

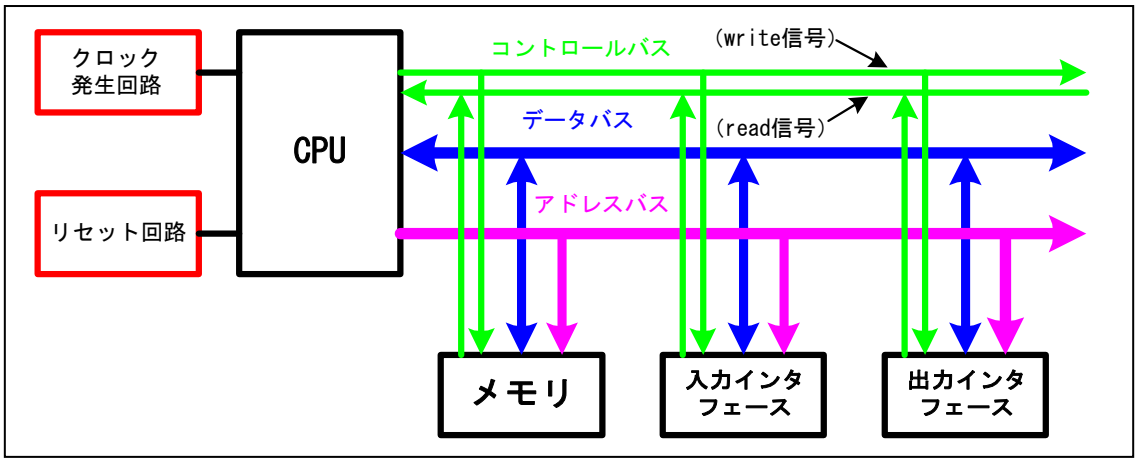


図 1-6

1.2. マイコンの基本動作

マイコンはメモリに書き込まれたプログラムを順次読み出して実行していきます。動作の内容は、フェッチ、デコード、エグゼキュートの 3 つで、リセット解除後この動作を繰り返し続けていきます。

リセット：リセット信号が一定時間入力された後、解除されると（リセット解除）マイコンが動作を開始します。

命令の読み込み（フェッチ）：ＣＰＵはプログラムカウンタの内容をアドレスバスから出力し、リード信号を出力した後、データバスから命令を入力します。

命令の解析（デコード）：CPU 内部で命令の解読をします。

命令の実行（エグゼキュート）：解読した命令を実行します。

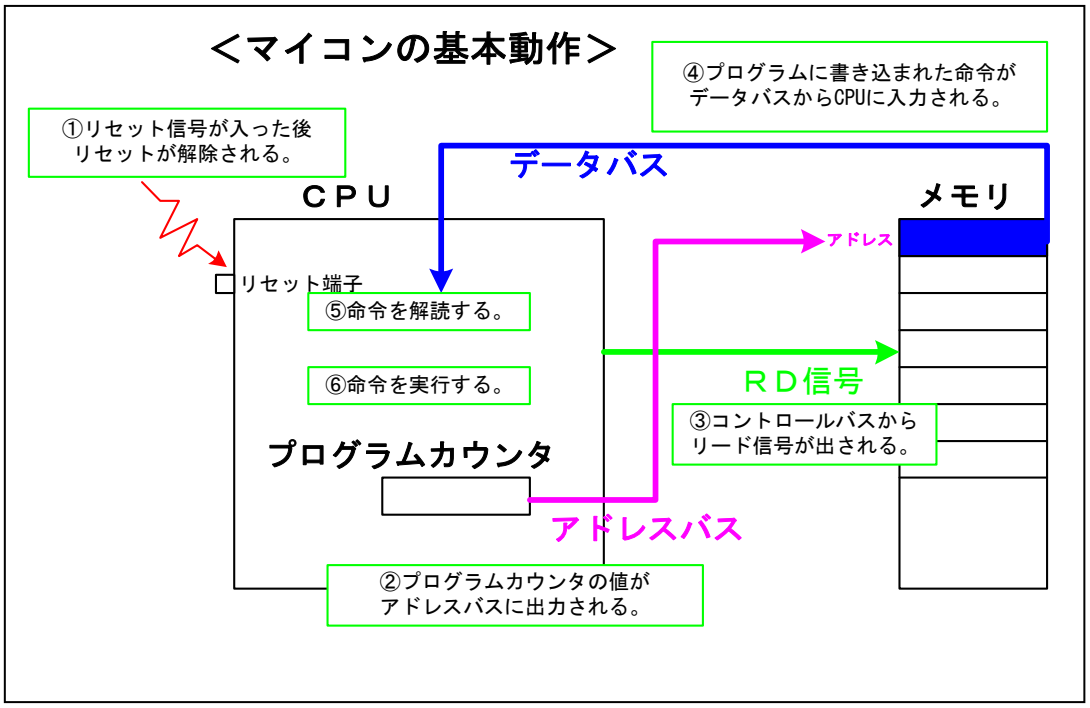


図 1-7

1.3. マイコンの扱うデータと言語

マイコンは全ての情報を“H”と“L”の二つの信号で伝達します。言い換えれば、マイコンの理解できる情報は、2 種類だけということになります。マイコンはこの 2 種類の情報を複数の信号線を使って、メモリや、I/O とやり取りするのです。この情報を“H”と“L”とで表現していてはとても分かりにくいので、とりあえず数字の“1”（H）と“0”（L）とに置き換えて考えるようになりました。“1”と“0”で表される 1 つの単位を**“ビット（ｂｉｔ）”**と呼びます。ビットとは指を意味します。このビットをいくつかまとめて、情報を表現するようになりました。2 ビットでは 2^2 で 4 通り。4 ビットでは 2^4 で 16 通り、8 ビットでは 2^8 で 256 通りのデータを表せます。

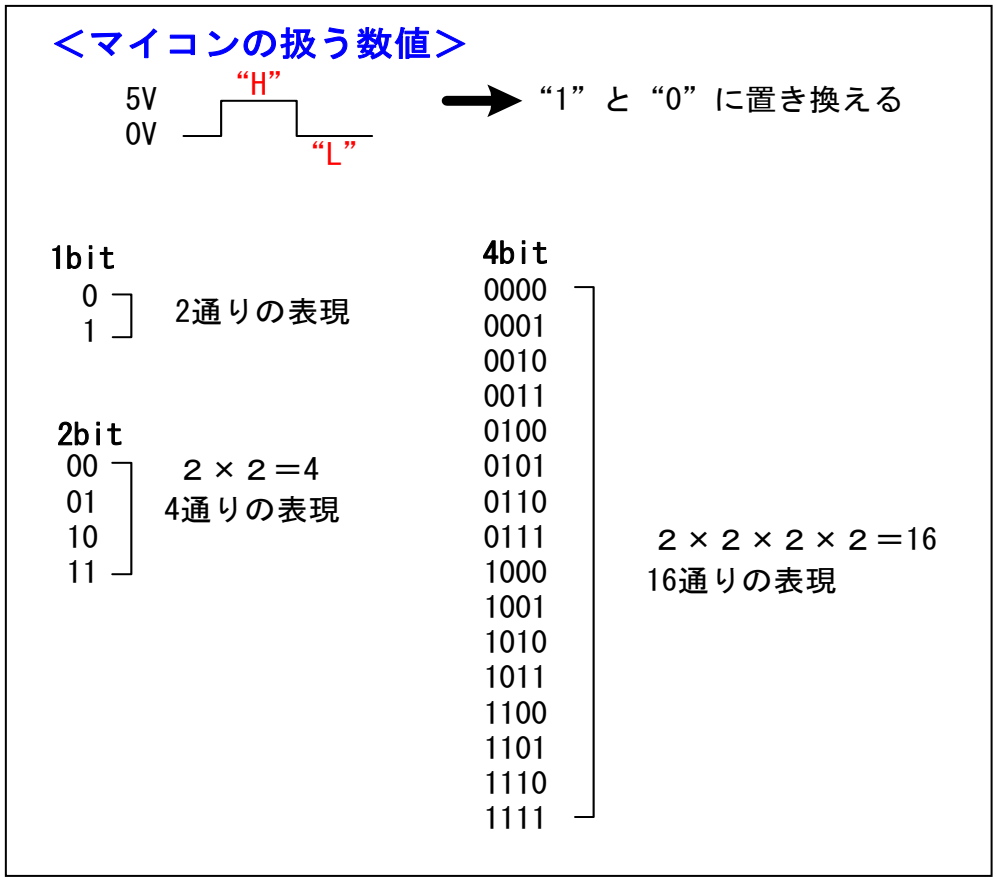


図 1-8

この表現方法で、マイコンのプログラムやデータなど全ての情報を表現します。

1.3.1. 数値を表すデータ

私たちが日常使用している数値は、0 から 9 までの数字を使って数を表します。この方法は、10 になるごとに桁上がりをするので、**10 進数**と呼びます。これに対して、マイコンの数値は、1 と 0 だけなので、2 になったら桁上がりをします。この方法を **2 進数**と呼びます。2 進数の表現は、数字の個数が多くなりすぎて解りにくいので、これを解りやすくするために **16 進数**の表記方法が出来ました。これは、**2 進数 4 ビットをまとめたもの**で、数字の 0 から 9 の後に、アルファベットの**A,B,C,D,E,F** を追加して、16 個の表記をするものです。（表 1-1）この 2 進数、16 進数、10 進数の区別をする為に、記号を使います。2 進数を表す記号としては“B”“%”、16 進数を表す記号としては“H”“0x”等がありますが、プログラムの中では、使える表現が決まっていますので、必要に応じて使い分けていきます。

表 1-1

2進数	16進数	10進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

表記例）

01010101B　　%01010101

（どちらも 2 進数を表します。）

55H　　0 x 55

（どちらも 16 進数を表します。）

マイコン側の、扱えるデータとしては、2 進数、16 進数ということになりますが、やはり人間界の 10 進数に少しでも近づけたいということで、**BCD（Binary Coded Decimal）**という表現方法が考えられました。これは、2 進数 4 ビット（16 進数 1 桁）で 10 進数の 1 桁を表すというものです。この表現を使うと、電卓など人間を相手にする用途では大変便利なので、BCD で計算を行なう命令語を持つマイコンもあります。

表 1-2

＜ＢＣＤ表記＞

2進数表現	16進数表現	10進数
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
00000101	5	5
00000110	6	6
00000111	7	7
00001000	8	8
00001001	9	9
00010000	10	10

1.3.3. 文字コード

コンピュータでは文字を取り扱う為にそれぞれの文字に 16 進コードを対応させています。その一つである ASCII コードはコンピュータ用の英数字コード体系として最も普及しているものです。その他のコードとして、JIS や EUC（拡張 UNIX コード）で決められた文字コードがあります。漢字は 16 ビットで表現されます。JIS には約 6,300 文字の漢字コードが登録されています。

＜ASCII コード＞

ASCII コードは、7 ビットで文字を表現しています。上位 3 ビットが 000 と 001 のコードは制御コードと呼ばれ、文字を表すのではなく改行やタブなどの文字表示の制御を行なう為のコードです。この他に、上位ビットを 3 ビットから 4 ビットに拡張し、日本語の半角カタカナを表現する拡張アスキーコードがあります。

表 1-3

		上位3ビット						
		000(0)	**001(1)**	**010(2)**	**011(3)**	**100(4)**	**101(5)**	**111(6)**	**111(7)**
下位4ビット	**0000（0）**	**NUL**	**DEL**	**SP**	0	@	P	‘	p
	0001（1）	**SOH**	**DC1**	!	1	A	Q	a	q
	0010（2）	**ETX**	**DC2**	“	2	B	R	b	r
	0011（3）	**EOT**	**DC3**	#	3	C	S	c	s
	0100（4）	**EOT**	**DC4**	$	4	D	T	d	t
	0101（5）	**ENQ**	**NAK**	%	5	E	U	e	u
	0110（6）	**ACK**	**SYN**	&	6	F	V	f	v
	0111（7）	**BEL**	**ETB**	‘	7	G	W	g	w
	1000（8）	**BS**	**CAN**	（	8	H	X	h	x
	1001（9）	**HT**	**EM**	）	9	I	Y	i	y
	1010（A）	**LF**	**SUB**	*	:	J	Z	j	z
	1011（B）	**VT**	**ESC**	+	;	K	[	k	{
	1100（C）	**FF**	**FS**	,	<	L	¥	l	|
	1101（D）	**CR**	**GS**	−	=	M	]	m	}
	1110（E）	**SO**	**RS**	.	>	N	^	n	~
	1111（F）	**SI**	**US**	／	?	O	_	o	

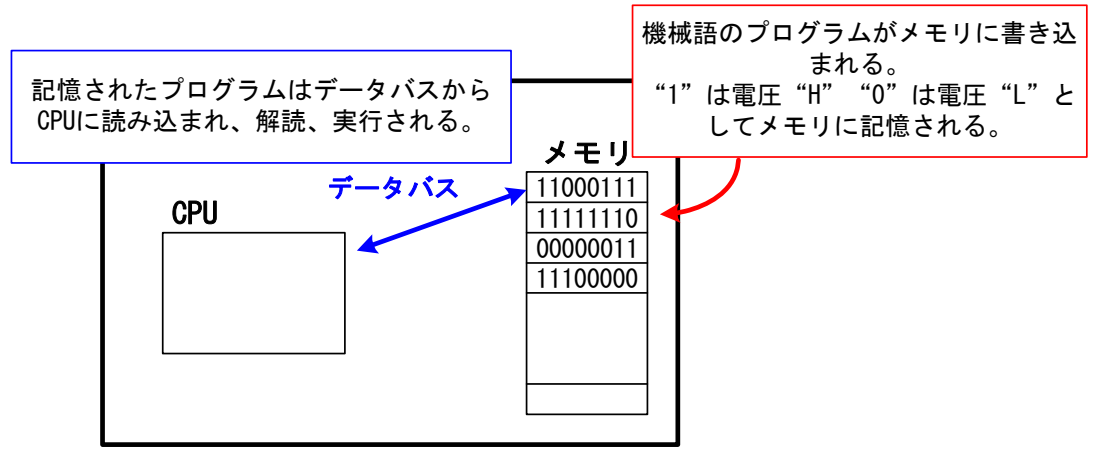
注）太字は制御コード

1.3.3. プログラムに使われるデータ（言語）

プログラムにもやはり機械語が使われます。プログラムを書く場合には、2 進数を 16 進数に置き換えてもやはり解りにくいことに変わりはありません。そこで、命令をニーモニック（mnemonic : 記憶を助けるというような意味です。）という記号に置き換える方法を考えました。これがアセンブリ言語です。アセンブリとは組み立てるという意味ですが、ニーモニックを使ってプログラムを記述する作業が組み立てを行なっているようなのでこう呼ばれるようです。

アセンブリ言語の問題は、CPU によってニーモニックが異なるということです。このためにプログラムの互換性（他のシステムでも使えるということ）がありません。そこで、どの CPU でも共通の記述で使えるプログラムのための言語が使われるようになりました。その 1 つがＣ言語です。Ｃ言語は、比較的簡単な記述で、プログラムを記述できます。

以下に、各言語記述と特徴を示します。（図 1-9）



言語	記述	特徴
マシン語（機械語）	11000111 ←転送する（命令部分） 11111110 ←feh（転送する値） 00000011 ） 11100000 ）←PORT0のアドレス	難解で、開発言語としては無理。
アセンブリ言語	MOV.B #0feh, port0 ～を，～へ 命令部分(転送する) （この記号を“ニーモニック”と呼ぶ）	機械語と1対1対応 ＣＰＵによって記述が異なる。 マイコンの詳しい知識が必要
C言語	Port0＝0xfe Port0に16進数feを代入する	CPUが変わっても共通の記述でOK
日本語	「Port0に、fehを出力する。」	開発言語ではない

図 1-9

このように、記述に便利な言語が出来ましたが、最終的にはやはり機械語に変換しないとマイコンでは動かすことが出来ません。このために、変換用のソフトウエアが存在します。
言い換えれば、変換用のソフトウエアがあれば、そのプログラム言語を使って開発が出来るということです。

C 言語からアセンブリ言語に変換する作業を**コンパイル**と呼び、その作業を行なうソフトウエアを**コンパイラ**と呼びます。
アセンブリ言語から機械語に変換する作業を**アセンブル**と呼び、その作業を行なうソフトウエアを**アセンブラ**と呼びます。
（ソフトウエアによっては、コンパイラが機械語まで変換してくれて、その一連の作業をコンパイルと呼ぶ場合もあります。）

このあと、OAKS16 を使ってこれらの作業を行なっていきます。

2.M30620FCAFP概要

OAKS16 には三菱の 16 ビットワンチップマイコンM16C/62AグループのM30620FCAFPが使われています。OAKS16 を使っていく上で、このマイコンがどのような機能をもっているかをある程度知っておかなければなりません。付属のCDROMには、このマイコンのデータシートとユーザーズマニュアルが入れてあります。これを全て読破して覚えればそれにこしたことはないのですが、それにはかなりの労力を伴います。ですから、ここでは最低限知って欲しいことだけを説明し、その他の部分については、必要になった段階で追加説明をしていきます。

M30620FCAFP は M16C/62A グループの 1 つで、内部 RAM（10KB）、フラッシュ ROM（128KB）のものです。それ以外の点ではすべてグループ共通の機能を持ちますので、ここから先は、M16C/62A として説明していきます。M16C/62A は、チップの内部にマイコンの基本構成要素であるＣＰＵ、メモリ、I/O ポートとタイマ等周辺機器を内蔵しています。

2.1. 中央演算処理装置（ＣＰＵ）

以下に CPU のレジスタ構成を示します。（図 2-1）

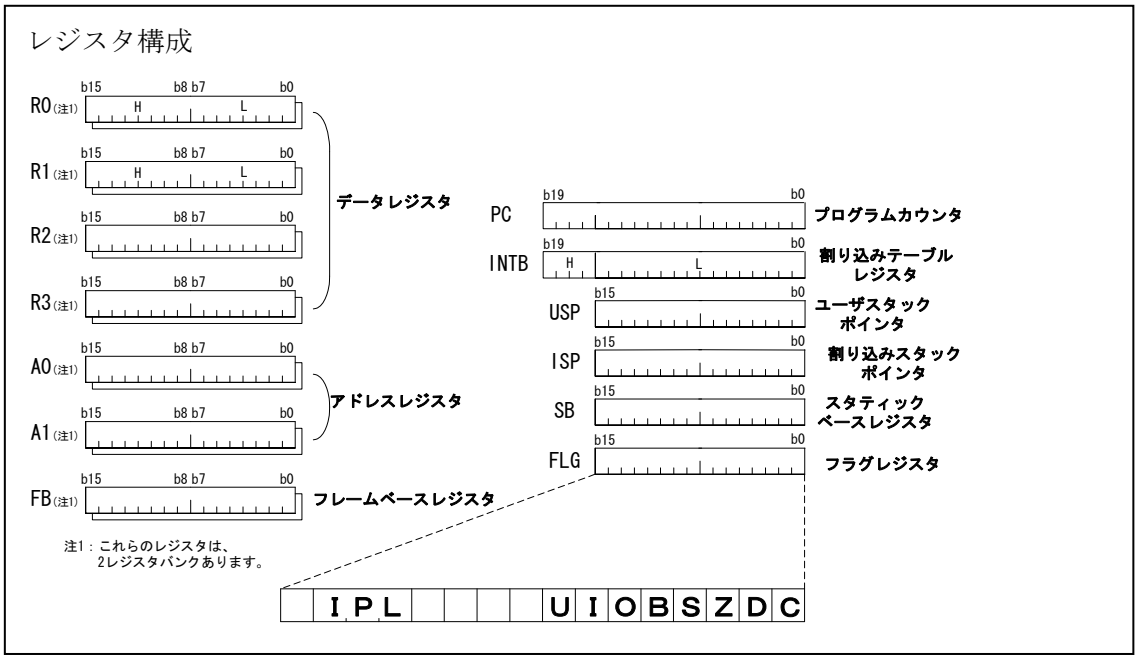


図 2-1

データレジスタ（R0/R1/R2/R3）：16 ビット構成ですが、命令語によっては、8 ビットまたは 32 ビットのレジスタとしても使用できます。

アドレスレジスタ（A0/A1）：データレジスタと同等の機能を持つ 16 ビットのレジスタです。アドレスレジスタ相対／間接アドレッシングで使用します。

フレームベースレジスタ（FB）：16 ビットのレジスタで、ＦＢ相対アドレッシングに使用します。

プログラムカウンタ（PC）：20 ビット構成で次に実行する命令の番地を示します。

割り込みテーブルレジスタ（INTB）：20 ビット構成で割り込みベクタテーブルの先頭番地を示します。

スタックポインタ（USP/ISP）：16 ビット構成でユーザスタックポインタ（USP）と割り込みスタックポインタ（ISP）の 2 種類があります。どちらを使用するかはスタックポインタ指定フラグ（フラグレジスタのビット 7 （U フラグ））で設定します。

スタティックベースレジスタ（SP）：16 ビットのレジスタで、SP 相対アドレッシングに使用します。

フラグレジスタ（ＦＬＧ）：１１ビットで構成されており、１ビット単位で使用します。アセンブリ言語でプログラムを記述するときの分岐の判断条件に使われたり、CPU の動作を設定する為に使われたりします。(図 2-2)

このレジスタと、フラグについては、アセンブリ言語でプログラムを書く場合には絶対に必要な知識となります。ですが、今回は C 言語でプログラム開発を行なっていきますので（一部、どうしてもアセンブリ言語で書かなければならない部分はありますが）それほどレジスタやフラグを意識したプログラミングはありません。

フラグレジスタのそれぞれの役割と使い方については、プログラム説明の中で必要に応じて説明していきます。詳細をお知りになりたい方は、ユーザーズマニュアル P1-13 をご参照ください。

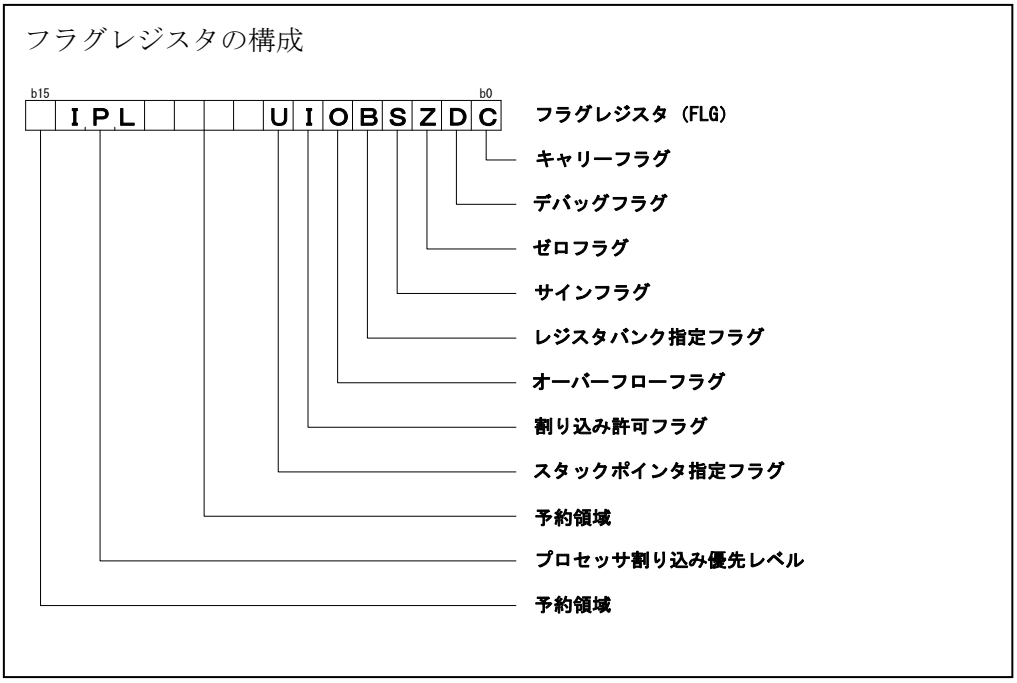


図 2-2

2.2. 周辺機能

M16C/62A の周辺機能を以下の図にまとめました。（図 2-3）
このテキストでは、入出力ポートについてだけ説明しますが、その他にタイマなど多くの機能を持っています。使用する場合は、ユーザーズマニュアルの各項目をお読みください。

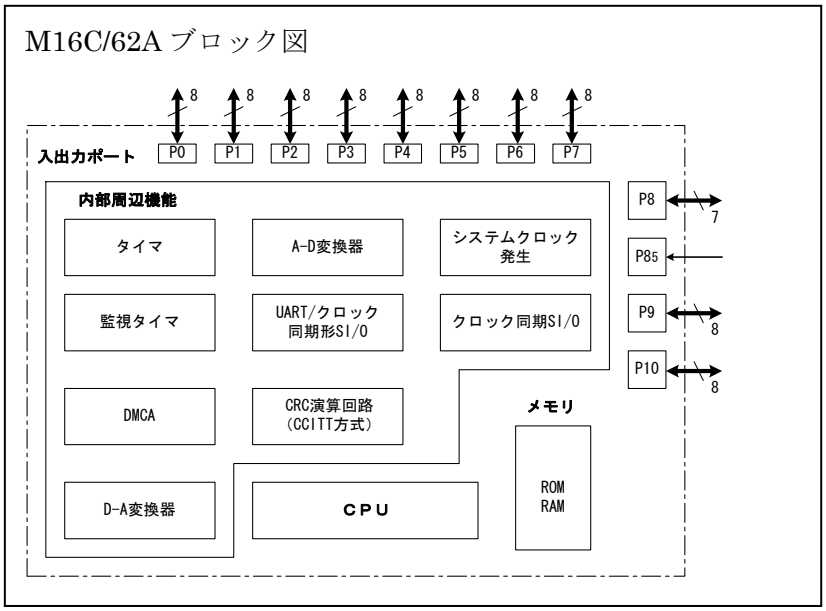


図 2-3

2.3. メモリマップ

メモリ配置図を示します。（図 2-4）アドレス空間は 00000h から FFFFFh までの 1 Mバイトあります。
M30620FCAFP は、シングルチップモード、メモリ拡張モード、マイクロプロセッサモードの 3 つのモードを持ち、メモリ拡張も可能です。
しかし OAKS16 ではシングルチップモードのみサポートしておりますので、ここではシングルチップモードのメモリマップについてだけ説明します。

SFR（スペシャルファンクションレジスタ）領域：ここには、CPU のモードや周辺機能の制御の為のレジスタが集められています。この領域の使用方法は、プログラミングの中で説明していきます。

内部 RAM：M30620FCAFP では 10K バイトの SRAM が使われています。

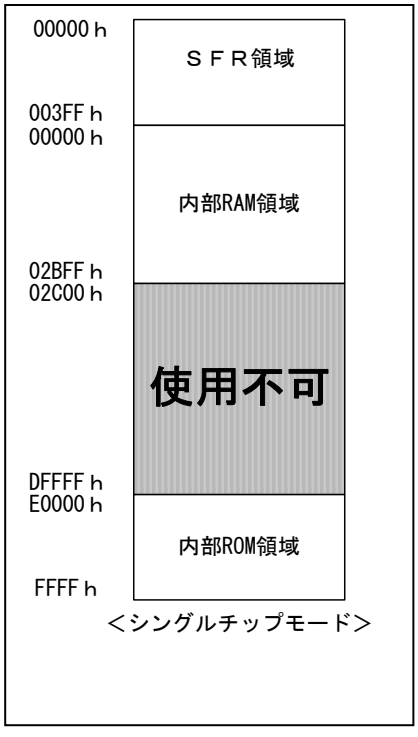


図 2-4

内部 ROM：M30620FCAFP では 125K バイトのフラッシュメモリが使われています。内部 ROM の一部（FFFDCｈ~FFFFFｈ）は、固定ベクタアドレスとなっており、割り込み発生時に実行するプログラムの先頭アドレスを格納するようになっています。（図 2-5）

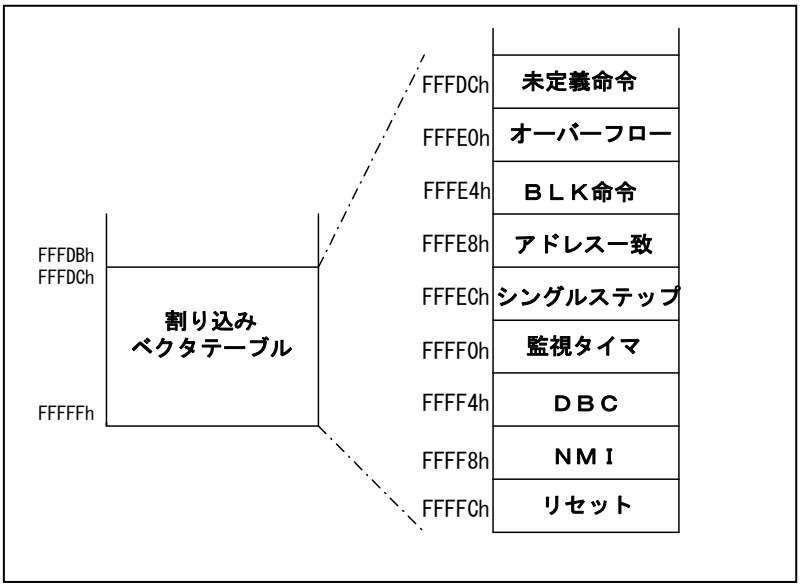


図 2-5

2.4. リセット

M16C/62A をリセット状態にする為には、ハード的にリセット信号を入力する方法と、ソフトウエアによりリセットする方法があります。通常のマイコンのシステムでは、『パワーＯＮリセット』といい電源投入時にマイコンの動作が安定するまでリセット状態を保ち、その後リセットを解除しマイコンの動作を開始するというような回路を組みます。OAKS16 では、電源投入時と SW1 を押したときにリセットがかかるように設計されています。

M16C/62A のリセット解除後の状態

① **SFR の一部は、決まった値にセットされます。**その設定を変更したい場合には、プログラムの中で設定していかなければなりません。方法については『4.OAKS16 のプログラミング』の中で説明していきます。

② **リセットが解除された後、リセットベクタテーブルに示されている番地からプログラムを実行します。**

リセット時の詳細は、ユーザーズマニュアルｐ1-15 リセットについての記述を参照して下さい。

2.5.M16Cの基本動作

以下に M16C の簡単な動作を示します。

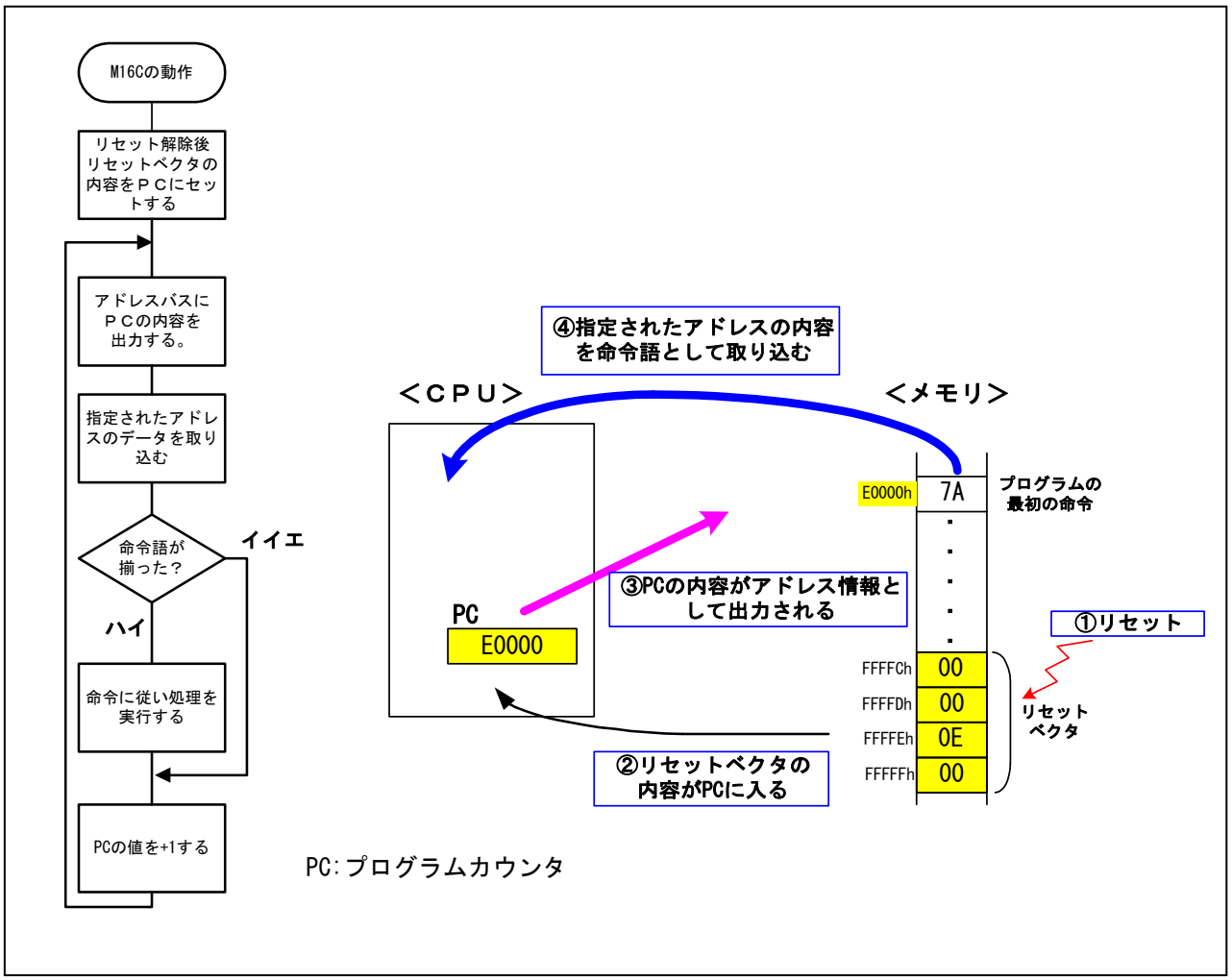


図 2-6

3. OAKS16 で開発するということ

プログラム開発は、プログラムを書いて実行するだけではありません。プログラムが、一度で仕様通りに動くことはまれで、ほとんどの場合、思うように動作してくれないものです。そこで、その原因を調べる作業を行なう必要が出てきます。この作業を**デバッグ**といいます。(**バグ**とは虫を意味し、プログラムの中のバグを探すことです。)

デバッグは全てのマイコン開発に必要な作業ですが、この作業を行なう為のツールをデバッガと呼びます。デバッガの代表は、ICE(インサーキットエミュレータ)ですが、メーカで出しているこのツールはとても高価なので、OAKS16 で使用しているような簡易デバッガ（モニタデバッガ）が多く使われるようになりました。このモニタデバッガは、パソコン側とターゲット側の両方にデバッグ用のプログラムが存在し、その間で通信を行ないながら、プログラムを実行したり、メモリの内容を参照したりするものです。このデバッガは、開発者が作成するプログラムと、デバッグ用のモニタプログラムが同じシステムのメモリに存在しますので、最初は少し戸惑われるかと思いますが、慣れてしまえば結構役に立つツールです。

OAKS16 で使っているデバッガは、KD30 というデバッガです。これから、KD30 を使って OAKS16 の開発をするために必要な内容を説明していきます。

3.1.OAKS16 の開発手順

図 3-1

まず、OAKS16 を使ったプログラムの開発手順を示します。この一連の作業は、TM で操作できます。TM を使用すると開発に必要な各ツール（ソフトウエア）を共通のツールバーから起動でき、開発作業がスムーズに行なえます。操作の詳細は『OAKS16 で TM をお使いになる方のために』を参照して下さい。

①コーディング：エディタ（テキストエディタ）を起動し、プログラムを記述します。このテキストでは、基本的には C 言語で記述します。しかし、後で詳しく説明しますが、M16C の開発をする場合、全てのプログラムを C 言語で記述することができません。M16C の初期設定の部分だけ、アセンブリ言語で記述します。

②コンパイル：詳しくは、**コンパイル**、**アセンブル**、**リンク**の作業を含みます。C 言語で記述したプログラムは、コンパイラ（NC30）でアセンブリ言語のプログラムに変換（コンパイル）します。その後、アセンブラ（AS30）でリロケータブルオブジェクトファイル（再配置可能な機械語ファイル）に変換（**アセンブル**）します。アセンブリ言語で書いたプログラムは、アセンブラ(AS30)でリロケータブルファイルに変換します。そしてリンカ(LN30)を起動して、すべてのリロケータブルオブジェクトファイルをまとめてアドレスを決定し、一つのアブソリュートオブジェクトファイル（実行ファイル）にします。（**リンク**）

③デバッグ：デバッガ（ＫＤ３０）を使ってプログラムの動作を確認します。仕様通りの動作をしない場合、GO-BREAK、メモリ参照などの機能を使って原因を突き止めます。そして再びエディタを起動してプログラムを修正し、コンパイルし、デバッグの作業を行ないます。

④ROM に書き込み：デバッガ上で仕様通りプログラムが動いたら、フラッシュ ROM ライタ(flashstart)でフラッシュ ROM にプログラムを書き込み、動作させます。ここで、仕様通りに動けば開発終了です。OAKS16 ではデバッガで動かすファイルと、フラッシュ ROM ライタで書き込むためのファイルが異なるので、ここでファイルの変換作業を行ないます。詳しい手順は、『ＯＡＫＳ16 でＴＭをお使いになる方のために』を参照して下さい。

3.2. メモリマップ

これは、OAKS16 のメモリマップです。前の章で説明した M3062FPAFP のメモリマップを思い出してください。その中の、RAM 領域の一部と、ROM 領域の一部をモニタが使用します。いままで、主流だったオンボード用のデバッガでは、ＲＯＭにモニタプログラムが入れてあり、ユーザプログラムは、RAM上に展開されていました。OAKS16 では、フラッシュメモリを使っている為、ROM 領域にプログラムを書き込んでデバッグができます。

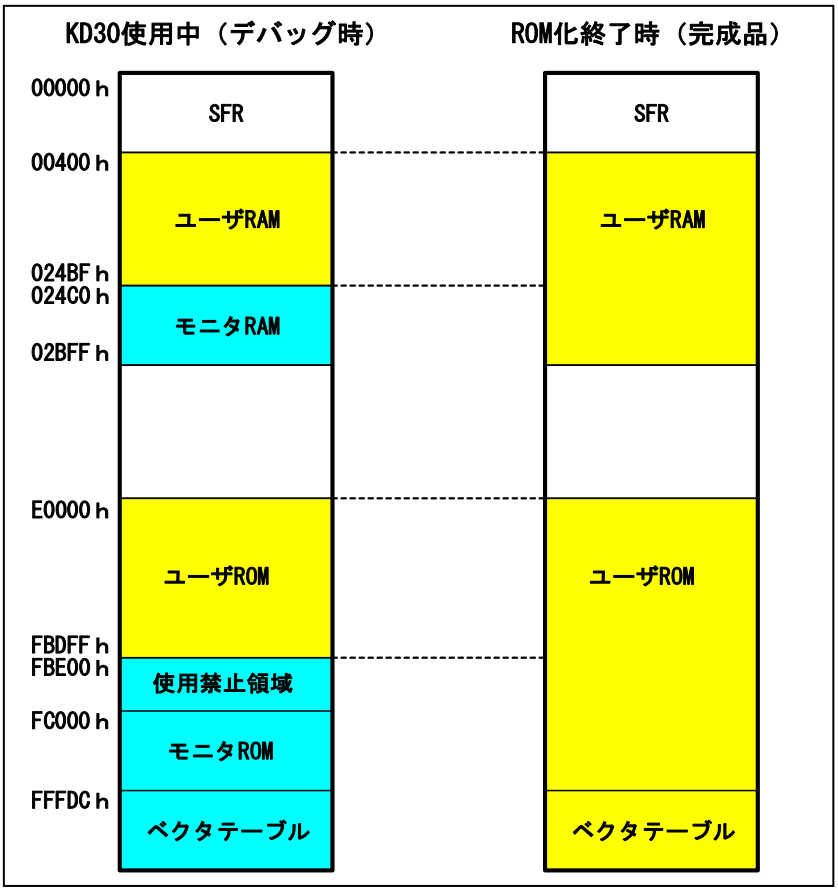


図 3-2

3.2.1 デバッグ時

まず、デバッグ時のメモリマップを見てください。モニタプログラムが、RAM、ROM の一部を使用しています。そのため、ユーザ使用領域は、RAM の 0400H～024BH 番地まで、ROM の E0000H 番地から FBDFFH 番地までになります。

M16C/62A では、リセット解除後（マイコンの動作スタート時）リセットベクタ（FFFFCH～FFFFF）で示されるアドレスからプログラムを実行します。しかし、デバッグ中は、KD30 を操作することによりプログラムを実行します。方法としては、まず、パソコンの KD30 画面上でプログラムカウンタの値（実行スタートアドレス）を設定します。そして『GO』コマンドを実行することでプログラムを動かします。プログラムの書き込みは、KD30 のプログラムダウンロード機能により行ないます。

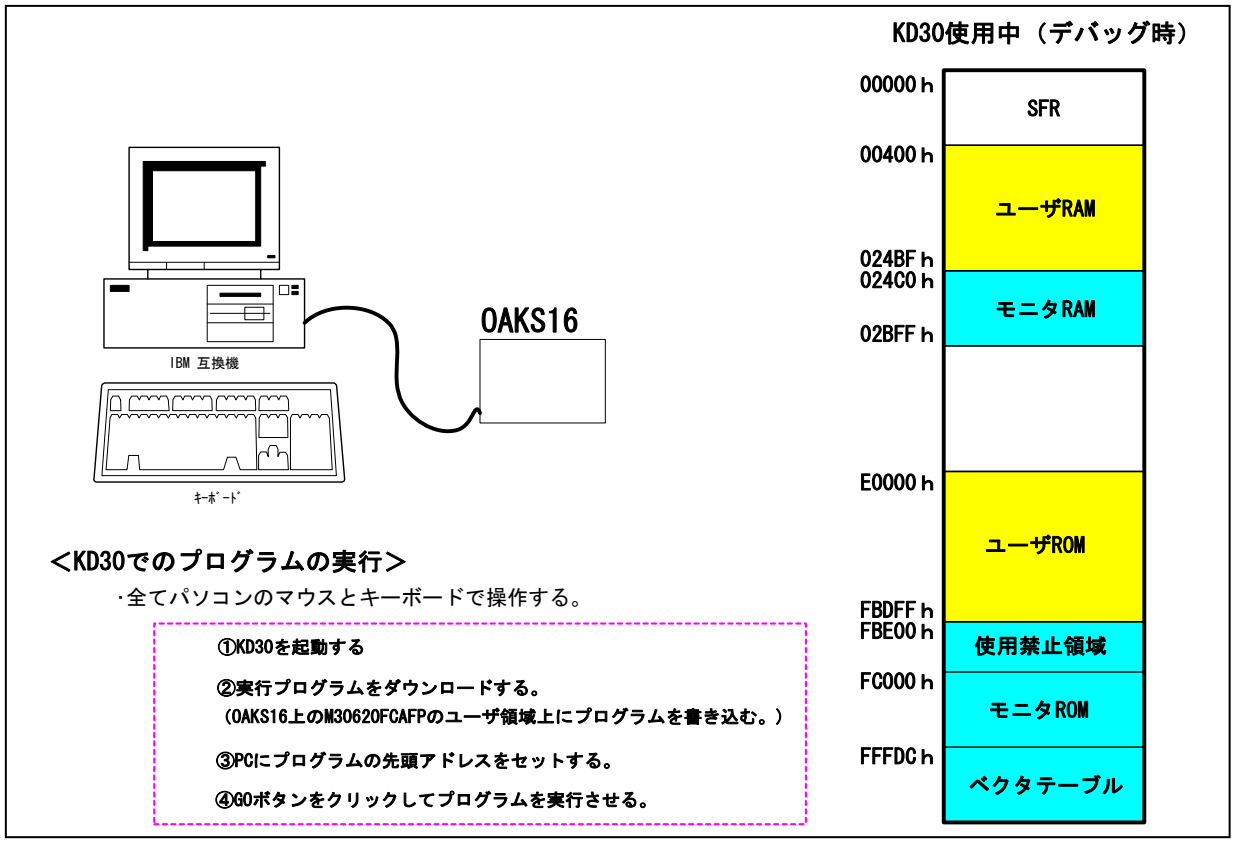


図 3-3

3.2.2. ROM書き込み時

デバッグが終了したら、今度は、フラッシュメモリにプログラムを直接書き込み、パソコンとのケーブルをはずし、OAKS16 単体で動作させます。

この場合のプログラムの書き込みは、KD30 ではできません。フラッシュメモリ書き込み用のソフト『flashstart』を起動します。『flashstart』は、M16C/62A のブート ROM に書かれているプログラムを使用します。（ブート ROM に通信用のプログラムが入っています）ブート ROM は、通常使用しているユーザ ROM の他にある、8K バイトのメモリで、出荷時に標準入出力モードでの書き換えプログラムが格納されています。ブート ROM はある条件を満たしたときだけ使用できます。（条件：$P5_5$ 端子“L”、CNV_{SS} 端子“H”、$P5_0$ 端子“H”としてリセットを解除）

そのため、フラッシュメモリに書き込む前（『フラッシュスタート』を起動する前）には、CPU ボードの J1 コネクタをショートします。これにより CNV_{SS} 端子が“H”になります。$P5_5$ 端子“L”、$P5_0$ 端子“H”は OAKS16 のＣＰＵボードですでに結線されています。（回路図参照のこと）

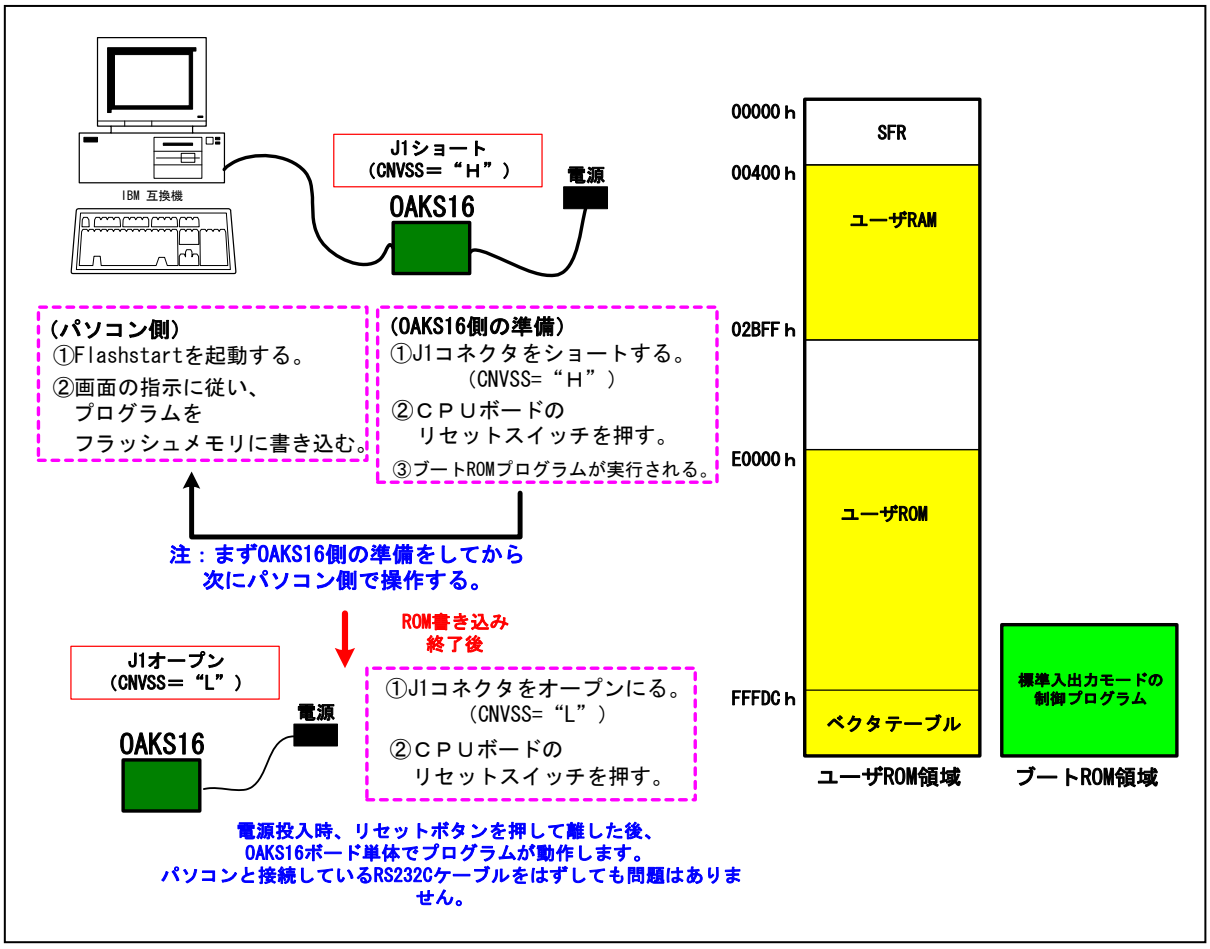


図 3-4

Flashstart でプログラムを書き込んだ後は、再び、CPU ボードの J1 コネクタをオープンにして、CPU ボードのリセットスイッチを押します。スイッチを離すと、作成したプログラムが単独で動作して、OAKS16 を制御します。
ここで書き込まれるプログラムの先頭アドレスは、フラッシュ ROM のリセットベクタに書き込まれていなければなりません。

この、プログラムの書き方、アドレスの決め方などを次の章で、例題の解説をしながら説明していきます。

4. OAKS16 のプログラミング（sampleA解説）

ここでは、OAKS16 のプログラミングについて、CDROM 内のサンプルプログラム sampleA を例にとって詳しく説明していきます。

OAKS16 でプログラムを開発する場合、いくつか注意することがあります。

①マイコンの初期設定：KD30 で動作させているときはモニタのほうで既に初期設定されている内容に関しては、ユーザプログラムで初期設定をしなくても正常に動作します。しかし ROM に書き込んで動作させる場合には初期設定をしていないと誤動作の原因となります。（特にスタックポインタの設定は必ずしなければなりません。）そのため、デバッグ用プログラムでも、初期設定は行なって下さい。特に、CPU の動作モードなどをモニタの設定と異なる設定をする場合注意が必要です。（モニタの初期設定の内容は、OAKS16Fullkit マニュアルまたは OAKS16Boardkit マニュアルを参照のこと）

②固定ベクタテーブル：モニタ（KD30）上でデバッグ中は、書き込みができませんが、ROM に書き込んで実行する場合には必ず必要なので、デバッグ用プログラムでも、記述しておいてください。

結論としては、デバッグ用、ROM 焼き用というようにプログラムを分けないで、同じプログラムを使った方が誤動作の原因が少なくてすみます。（デバッガで、フラッシュ ROM に書き込めないところは無視してくれるので、誤動作の原因にはならない。）

これらの点を考慮して sampleA のプログラムを考えてみましょう。

4.1. sampleAの仕様

では、sampleA のプログラムを考えます。制御する対象は、CPU ボード上の LED1 です。今回は、この LED を点滅させることを考えます。点滅の方法としてはタイマ割り込みを使って時間を制御する方法もありますが、ここでは、ソフトウエアウエイトを使うことにします。詳しいプログラムの書き方はあとで説明します。

4.1.1. I/O回路図

制御の対象となる、LED1 の接続回路を確認します。（図 4-1）LED 接続の詳しい説明は（『５．I/O 制御の演習』）で行ないますので、ここでは、配線された回路図を確認し、どのようなデータをポートから出力すればいいのかを考えます。

LED1 は $P0_0$ に接続されています。この回路図では $P0_0$ から“H”を出力すれば LED1 が消灯し、$P0_0$ から“L”を出力すれば LED１が点灯することがわかります。（詳細は、『５．I/O 制御の演習』参照のこと）

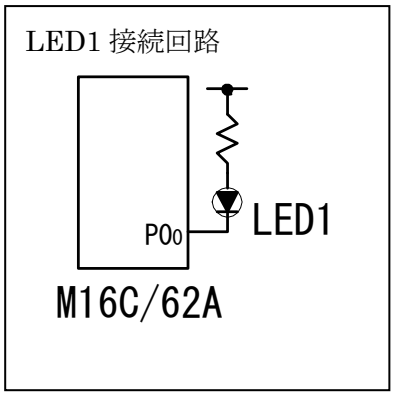


図 4-1

4.1.2. I/Oインターフェース仕様

今回制御する LED1 のインタフェース仕様は以下のようになります。(図 4-2)

インタフェース仕様

	点灯	消灯
LED1（$P0_0$）	L	H

図 4-2

4.1.3. フローチャート

以下に、このプログラムのフローチャートを示します。(図 4-3)

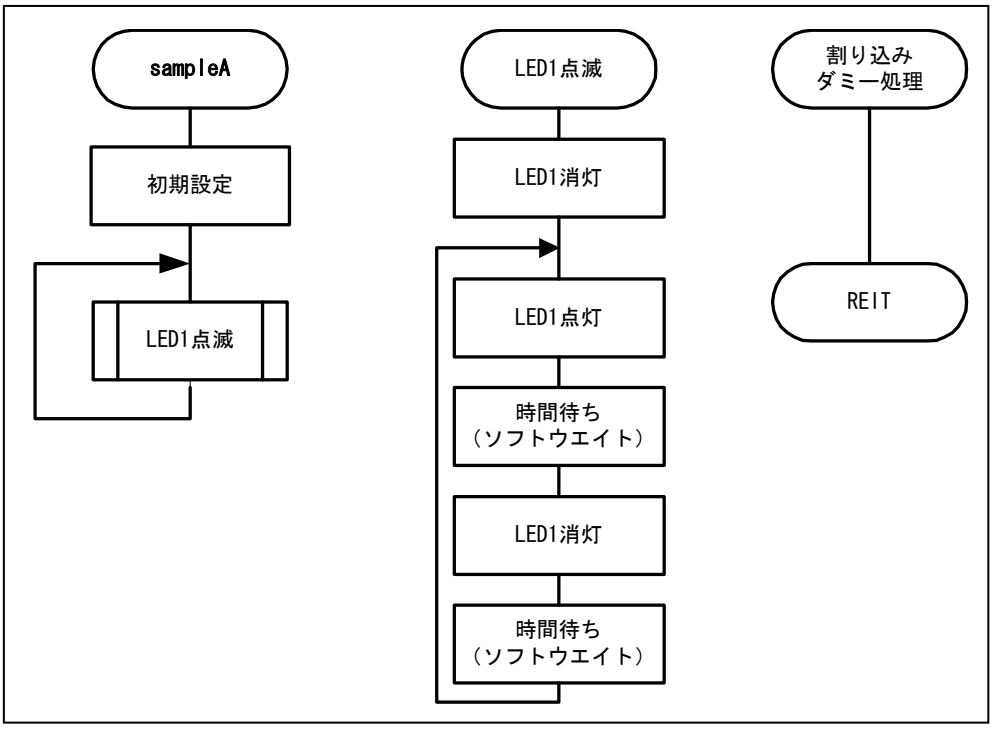


図 4-3

4.1.4. ファイル分割

<start0.a30>：スタートアッププログラムです。アセンブリ言語で記述します。内容は、M16C/62A の初期設定です。初期設定をしたあと、Ｃ言語で記述した main プログラムをサブルーチンコールします。割り込み処理のダミープログラムもアセンブリ言語で記述しますので、このファイルの中に一緒に入れておきます。このダミープログラムの内容は、何も処理を行なわずに、割り込みから復帰するというものです。

<test.c>：main プログラムを記述します。今回は、LED1 の点滅プログラムを記述します。

<vector.a30>：固定ベクタテーブルの設定をします。リセットベクタの領域に、satrt0.a30 に記述した初期設定の先頭アドレスを記述します。その他のベクタには、割り込みダミープログラムの先頭アドレスを記述します。

下記に、それぞれのファイルとプログラムの流れを示します。

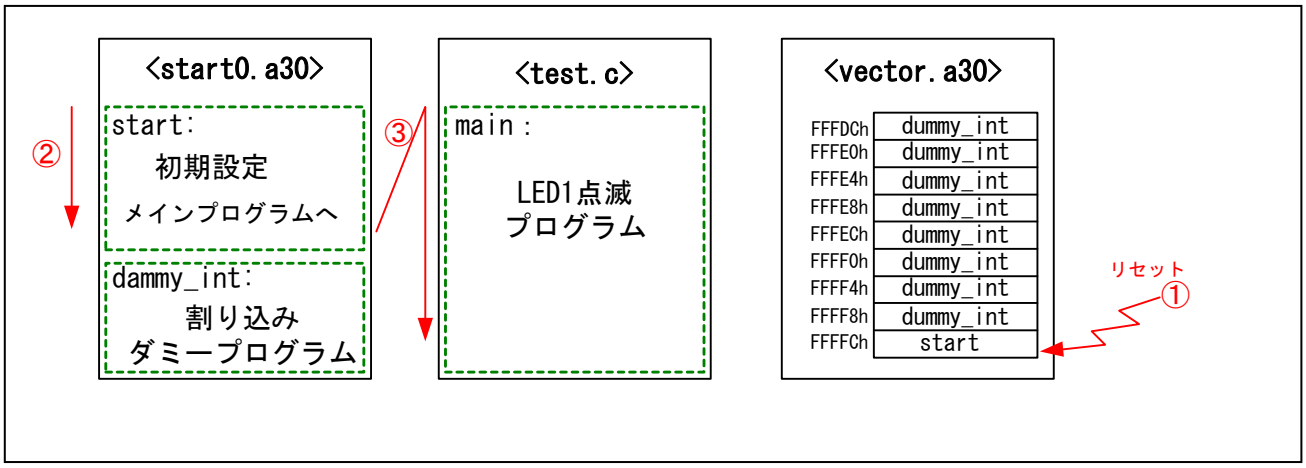


図 4-4

《プログラムの流れ》

①リセットが解除されると、リセットベクタに入れてあるアドレス情報を読み出します。

②リセットベクタに入れてあったアドレスからプログラムを実行します。

③プログラム記述により、メインプログラムに処理が移ります。

（start0.a30 の初期設定内のサブルーチンコール命令で test.c のメインプログラムからプログラムを実行します。）

4.1.5. ファイル構成とオブジェクトファイルができるまで

sampleA のファイル分割ができたところで、これらの、ソースファイルから OAKS16 で実行できるファイルができるまでの流れを説明します。OAKS16 では、KD30（デバッガ）が使用するファイルと、flashstart（フラッシュメモリ書き込みソフト）が扱えるファイルが異なります。これらのファイルを作成するために、NC30 を使ってコンパイルしていくわけですが、実際にはコンパイル→アセンブル→リンク→ROM 化ファイル生成という手順を踏みます。

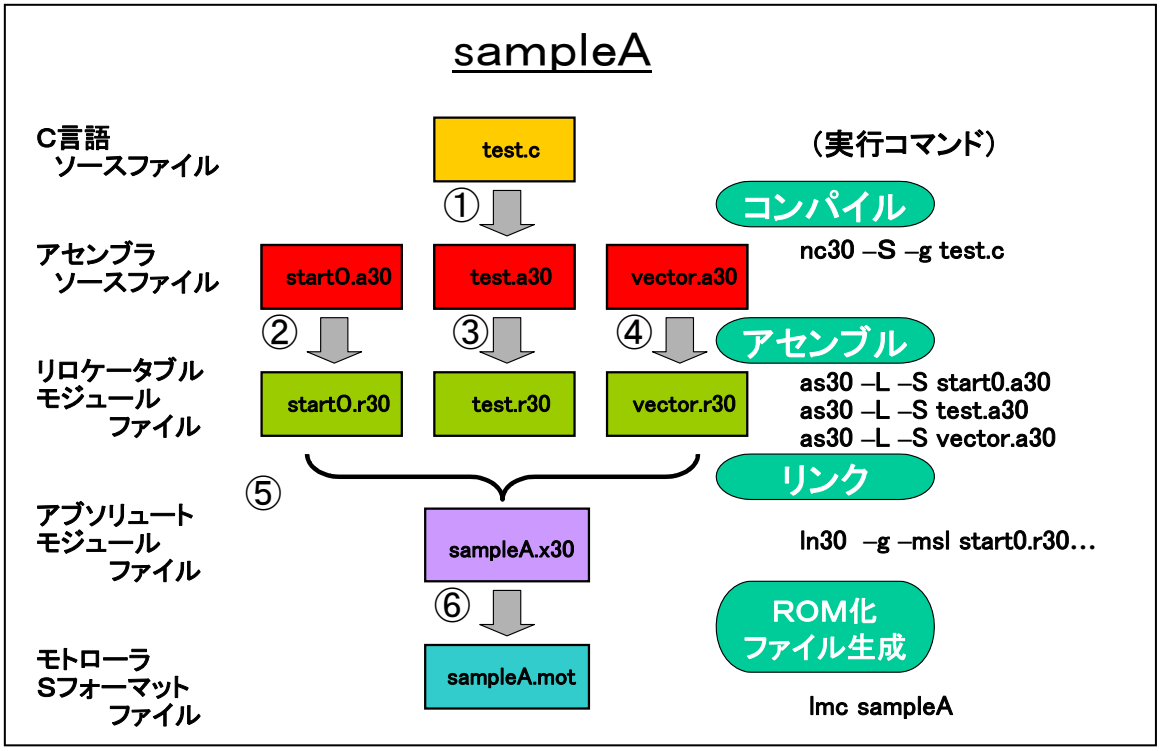


図 4-5

ソースファイルは test.c(C 言語ソースファイル)、start0.a30（アセンブリ言語ソースファイル)、vector.a30（アセンブリ言語ソースファイル）の 3 つです。

① test.c からはコンパイラ NC30 により、test.a30 というアセンブラソースファイルが生成されます。

② start0.a30 からはアセンブラ AS30 により、start0.r30 が生成されます。

③ test.a30 からはアセンブラ AS30 により、test.r30 が生成されます。

④ vector.a30 からはアセンブラ AS30 により、vector.r30 が生成されます。

⑤ ②③④で作成されたリロケータブルモジュールファイルをリンクして、sampleA.x30 というアブソリュートモジュールファイルを作成します。（デバッガ KD30 用）

⑥ ⑤で作成された sampleA.x30 から sampleA.mot というモトローラＳフォーマットのファイルを生成します。（FlashStart 用）

ここまでが、開発の流れです。

4.2. サブルーチンの考え方

プログラムの記述方法として知っておかなければならないサブルーチンについて説明します。

アセンブリ言語ではサブルーチンとして記述しますが、C 言語では関数として記述します。(C 言語をコンパイラでアセンブリ言語の記述に書き換えるときには、サブルーチンの形になります。)

サブルーチンとは、『メイン』のルーチンに対して、『サブ』のルーチンということで、まとまった処理を一つの独立したプログラムとして記述する方法です。

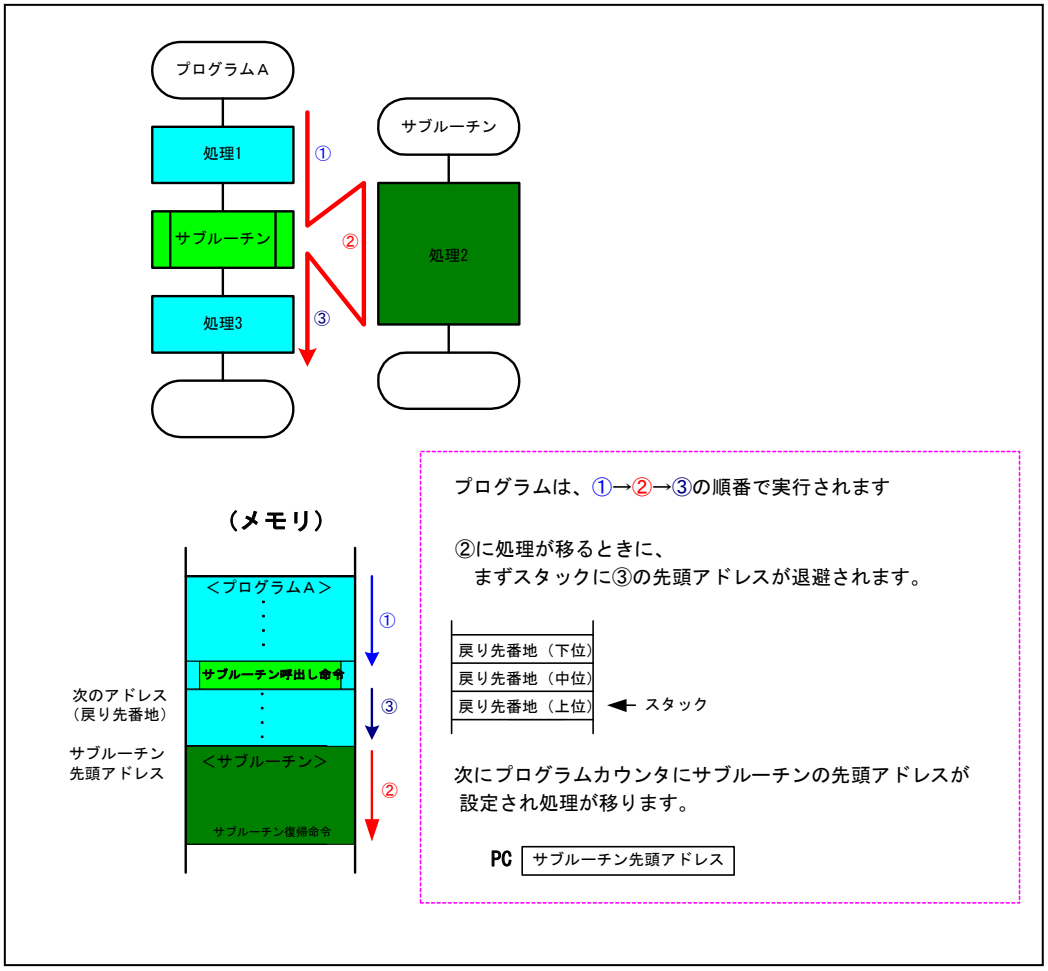


図 4-6

上に示すフローチャートのように、プログラムの途中でサブルーチンを呼び出すことによって、動作させます。

この方法を取ると、まとまった処理を分けて記述することができ、プログラムが見やすくなります。

また、NC30 では、異なったファイルに記述されているプログラムをリンクして最終プログラムを作成できるので、サブルーチン（C では関数）の形をとり、他のファイルから呼び出して使用できることは、仕事の分担ができたり、他のシステムでそのサブルーチンや関数を流用したりでき、開発効率を上げることもできます。

プログラムの書き方はこのあと説明していきますが、サブルーチンコール命令を使うことで、指定したアドレスからプログラムを実行することができます。この動作は、条件無しの分岐命令でもできますが、サブルーチンの形を取ると、サブルーチンの処理が終わるともとのプログラムの続きから処理を行ないます。サブルーチンからの復帰は、サブルーチンの最後にサブルーチン復帰命令を記述しておき、この命令の実行によって、もとのプログラムの続きから命令を実行していきます。

この、サブルーチンの動作のポイントとなるのが**“スタックポインタ”**です。サブルーチン呼び出し命令が実行されると、スタックポインタが示すメモリ（“スタック”と呼びます）に次の命令のアドレス（戻り先番地）を退避します。そして、プログラムカウンタにサブルーチンの先頭アドレスを入れ、サブルーチンのプログラムが実行されます。サブルーチンの最後では、サブルーチンの復帰命令が実行され、スタックに退避してあったアドレスが、プログラムカウンタに入り、サブルーチン呼び出し命令の次にあった命令を実行します。

4.3. アセンブリ言語の基礎知識

前に説明しましたように、M16C/62A のプログラムをＣ言語で開発する場合、初期設定の部分だけはアセンブリ言語で記述しなければなりません。このプログラムを**スタートアッププログラム**と呼びます。アセンブリ言語で記述する理由は理由は、C 言語で記述できない部分（スタックポインタの設定）があるためです。それ以外にも、レジスタの初期設定等はここで行ないます。

この、スタートアッププログラムを作成するために使用するアセンブリ言語をこれから説明していきます。

ファイルの記述の中には 2 種類の記述が混在します。一つは、**M16C/62A のニーモニック**（実際に機械語に直されてプログラムとなるもの）。もう一つは、アセンブラ記述のファイルを機械語の実行ファイルに変換する為のソフト AS30（アセンブラ）に対する**指示命令**です。
ここでは、ファイル内で使用されたものだけを説明していきます。さらに詳しい内容を知りたい方は、ニーモニックについては『M16C/60 M16C/20 シリーズソフトウエアマニュアル』を、指示命令については『AS30 ユーザーズマニュアル』をご覧下さい。

4.3.1. sampleAで使用するニーモニック（アセンブリ言語命令）

このファイル内で使用する命令語は、専用レジスタへの転送命令、転送命令、ビット操作命令、サブルーチンコール命令、割り込みからの復帰命令の 5 種類です。このうち、割り込みからの復帰命令は、割り込みの説明のときに行ないますのでここでは説明しません。

専用レジスタへの転送命令

ＬＤＣ命令

ニーモニック	記述形式	オペレーション
ＬＤＣ	LDC　src, dest	dest　←　src

・srcの内容をdestに転送します。

＜記述例＞

```
LDC   R0,SB
R0 1234ｈ   SB 0000ｈ  ＜実行前＞
R0 1234ｈ   SB 1234ｈ  ＜実行後＞

LDC   #024Bh,ISP
ISP  0000ｈ  ＜実行前＞
ISP  024Bｈ  ＜実行後＞
```

図 4-7

転送命令

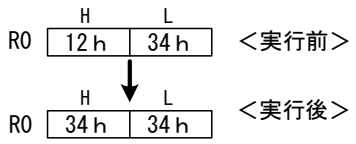
ＭＯＶ命令

ニーモニック	記述形式	オペレーション
ＭＯＶ	MOV.size　src, dest	dest ← src

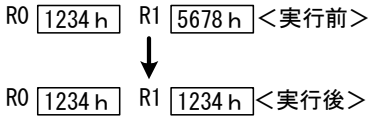
・srcの内容をdestに転送します。

＜記述例＞

MOV.B　ROL,ROH



MOV.W　R0,R1



MOV.B　#00100000B,0007H

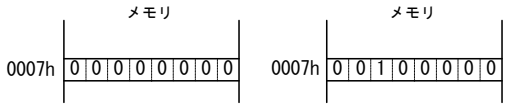


図 4-8

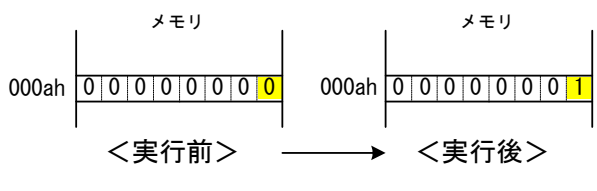
ビット処理命令

ＢＳＥＴ･ＢＣＬＲ命令

ニーモニック	記述形式	オペレーション	
ＢＳＥＴ	BSET　dest	dest ← 1	・destに１を格納します。
ＢＣＬＲ	BCLR　dest	dest ← 0	・destに０を格納します。

＜記述例＞

BSET　0,000AH



BCLR　0,000AH

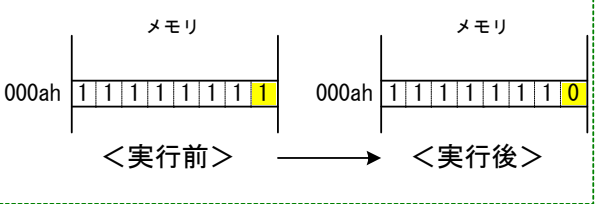


図 4-9

サブルーチン呼び出し命令

ＪＳＲ命令

ニーモニック	記述形式	オペレーション
ＪＳＲ	JSR(.length)label 注：(.length)は分岐距離指定子	SP ← SP-1 M(SP) ← (PC+n)H SP ← SP-2 M(SP) ← (PC+n)ML PC ← label 注：nは命令のバイト数

・labelへサブルーチン分岐します。

・分岐範囲は分岐距離指定子により異なります。

.length	分岐範囲
.W	命令の先頭番地を起点に-32767～+32768バイト（16ビットPC相対）
.A	フルアドレス空間（20ビット絶対）

＜記述例＞

```
jsr.a  label
```

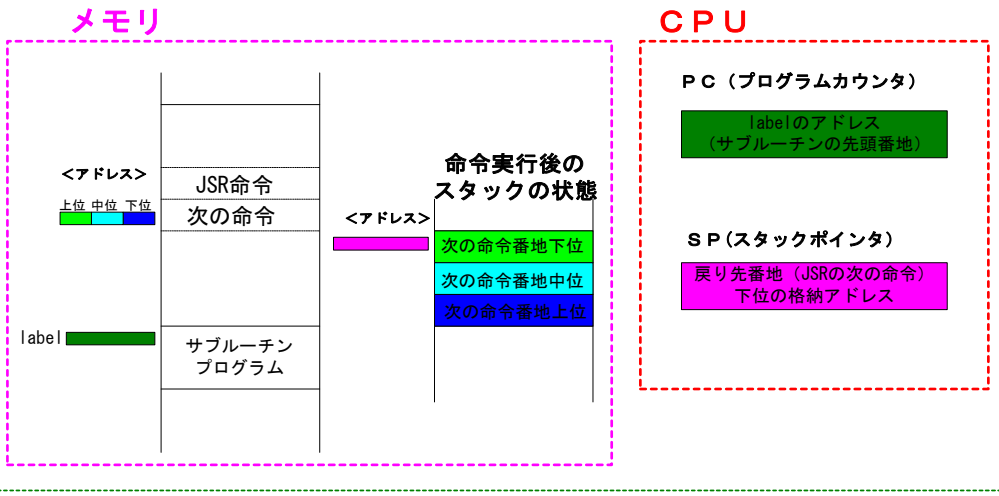


図 4-10

4.3.2. sampleAで使用する指示命令

AS30（アセンブラ）で使用できる指示命令について説明します。指示命令は、アセンブラ記述ファイルを機械語のオブジェクトファイルに変換するソフト（AS30）に対する指示をするものです。ここでは sampleA のスタートアッププログラム start0.a30 で使われる指示命令について説明します。その他の指示命令については、AS30 のユーザーズマニュアルを参照して下さい。

アセンブル制御指示命令

AS30 の実行について指示できます。

命令	機能	使い方･記述例
.EQU	シンボルを定義します。	命令の右側に記述された内容が、左側の記述方法で記述できるようになります。 例 1） intstack　.EQU　　024BFH ･この定義の後からは、“024BFH”と書くところを“intstack”と書くことができます。 例 2） PRCR　　.EQU　　000AH PRC0　　.BTEQU　0,PRCR ･この定義の後からは、“1,000AH”と書くところを“PRC0”と書くことができます。
.BTEQU	ビットシンボルを定義します。	
.END	アセンブラソースの終了を宣言します。	アセンブラのソースファイルの最後に記述します。

リンク制御指示命令

アドレス再配置制御の為の情報を定義できます。

命令	機能	使い方･記述例
.SECTION	セクション命を定義します。	例）.section　program リンクするときの情報になります。
.GLB	グローバルラベルを指定します。	例）.glb　main “main”というラベルが他のファイルから参照されることを表します。

アドレス制御命令

命令	機能	使い方･記述例
.ORG	アドレスを宣言します。	セクション指示命令 “.SECTION” の直後に記述してください。 例）.section　fvector 　　.org　　0fffdch
.LWORD	4 バイト長データで ROM 領域に格納します。	例）.LWORD　symbol（symbol は 4 バイトデータ：プログラムのアドレス等）

4.3.3. スタートアップファイル（start0.a30）の解説

sampleA の初期設定プログラムファイル（start0.a30）の説明をします。

フローチャート

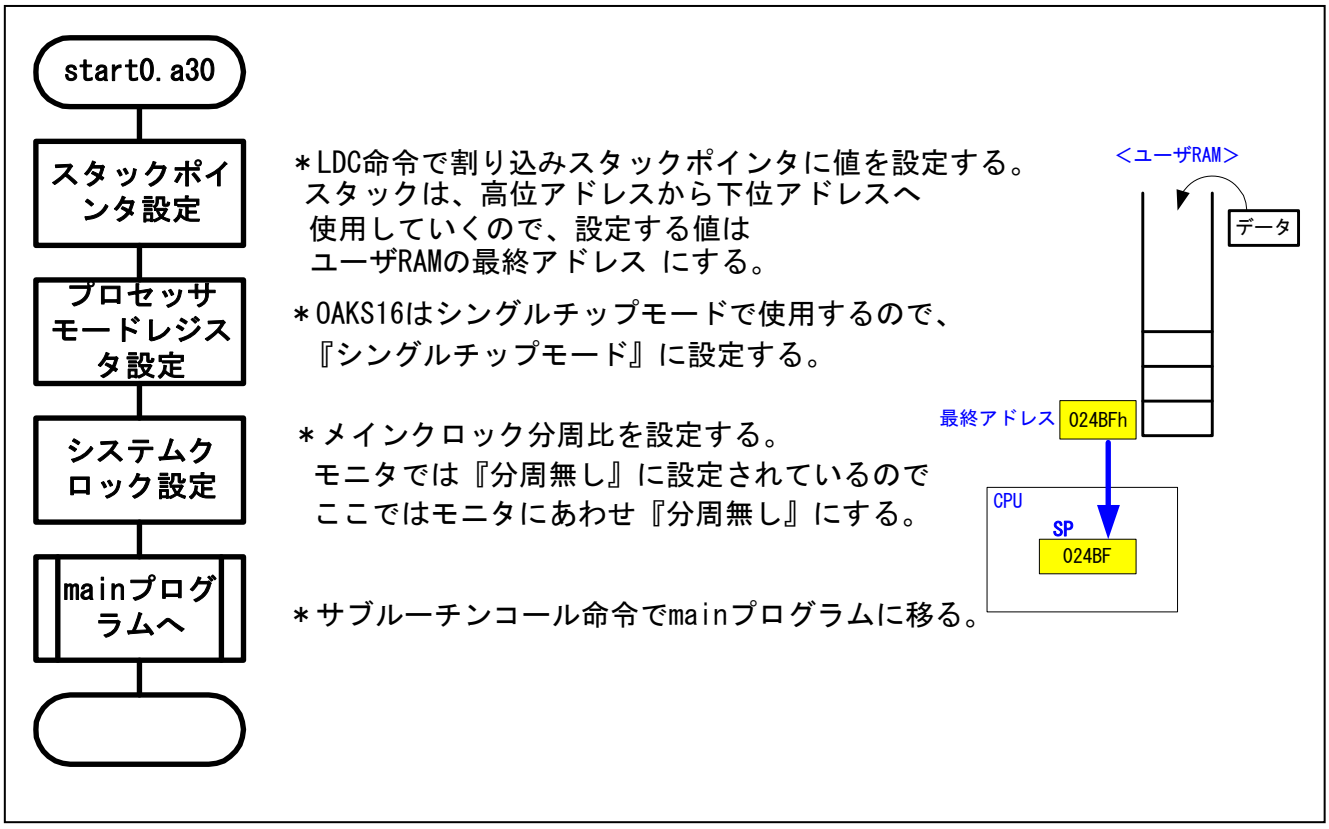


図 4-11

プロセッサモードレジスタ設定

OAKS16 のプロセッサモードを決めるレジスタで、プロセッサモード 0 とプロセッサモード 1 があります。これらのレジスタは、SFR 領域の 0004h,0005h 番地になります。OAKS16 の設定を以下の表で確認してください。

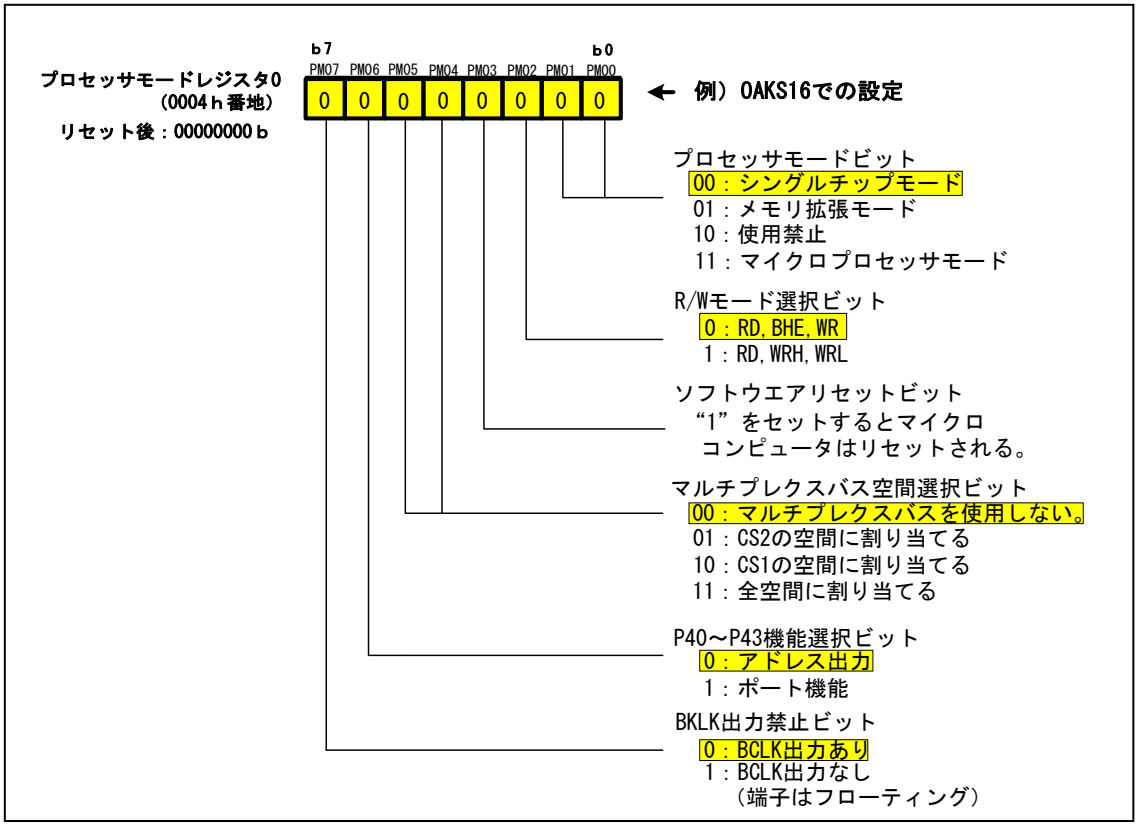


図 4-12

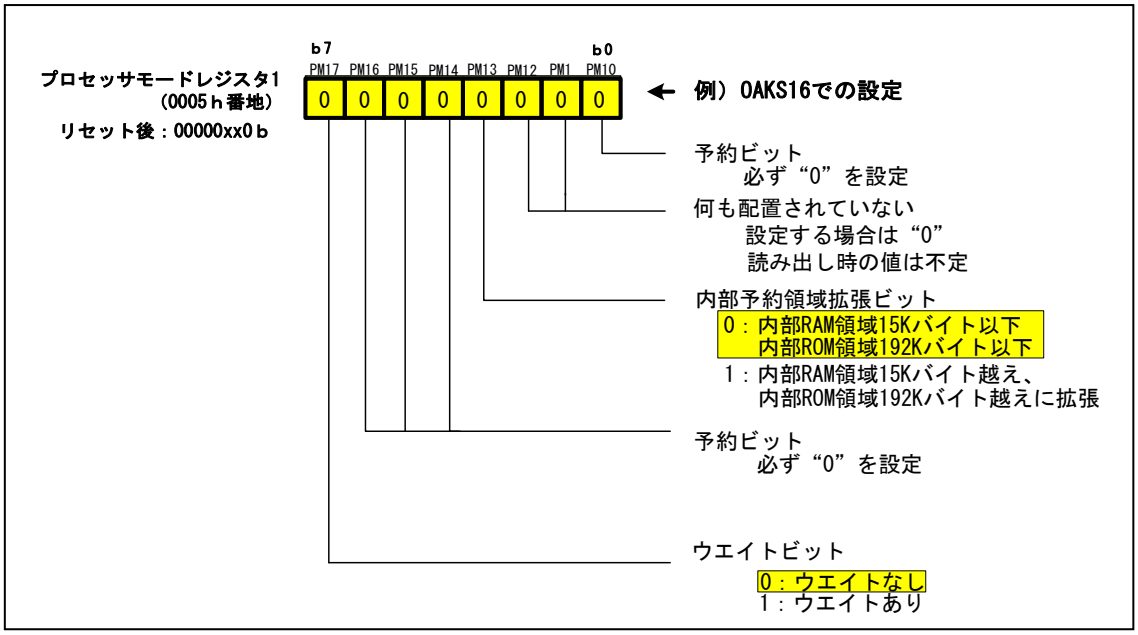


図 4-13

システムクロック制御レジスタ設定

このレジスタで、CPU の動作クロック源を設定します。CPU リセット時には、メインクロックの８分周が BCLK になります。OAKS16 ではモニタで、“分周なし”に設定していますので、ユーザプログラムの初期設定でも分周の設定をして下さい。このレジスタの値を設定しないと、モニタ起動時には、“分周なし”（モニタ設定により）、ROM 化時には、８分周（リセット時の値）ということが起こってしまいます。

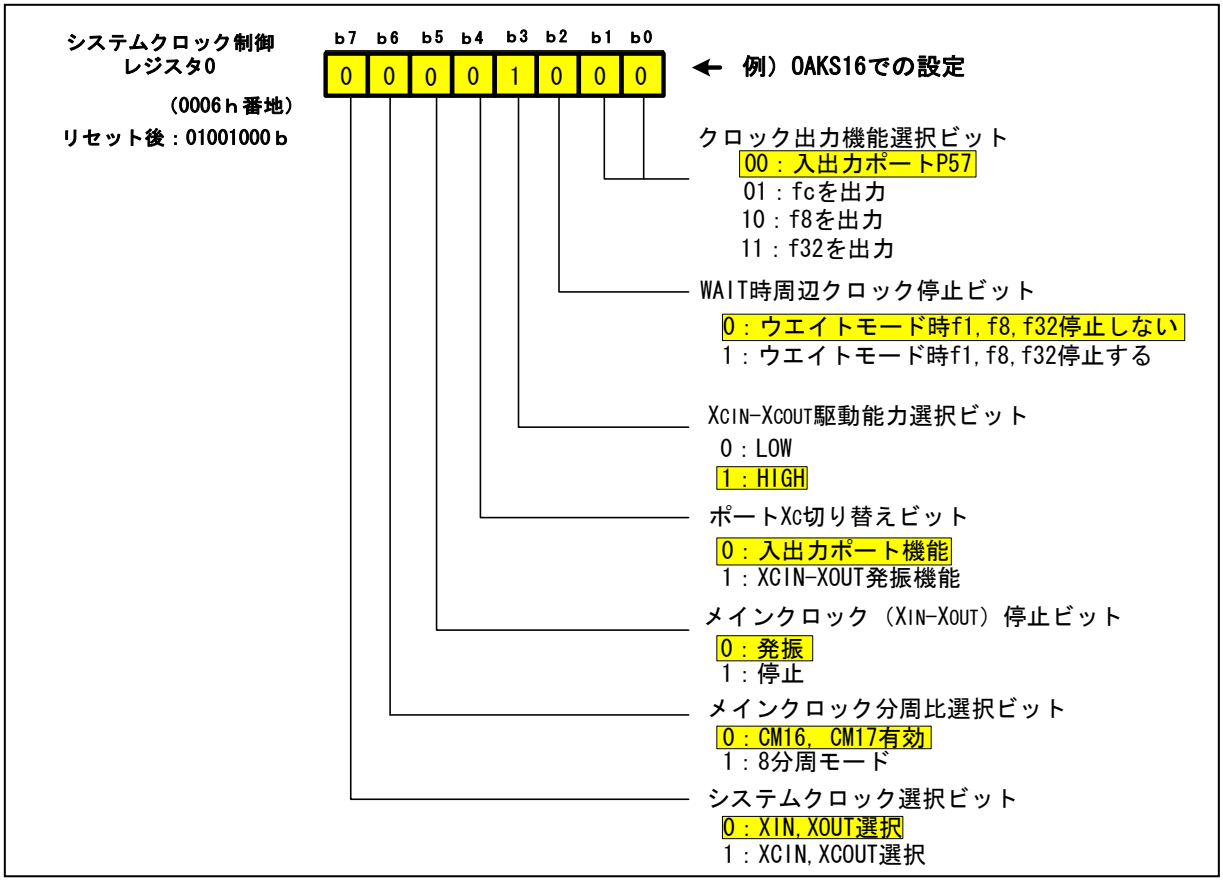


図 4-14

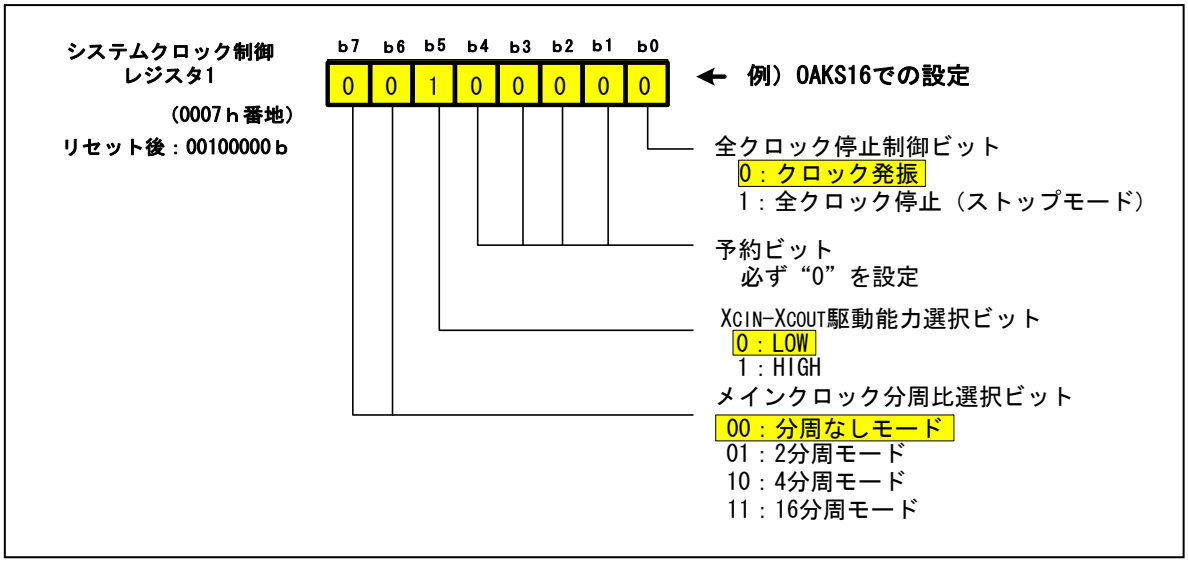


図 4-15

プログラム記述例

```
;/*-------------------------------------------*/
;/* ファイル名：      start0.a30              */
;/* 内容：            スタートアッププログラム  */
;/* 日付：            2001.9.19               */
;/* 作成者：          0AKS16KIT support       */
;/*-------------------------------------------*/

intstack.EQU    024BFH          ; スタックポインタ初期値(ユーザＲＡＭ最終番地)
PRCR .EQU       000AH           ; プロテクトレジスタ
PRC0 .BTEQU     0,PRCR          ; システムクロックレジスタビット
PRC1 .BTEQU     1,PRCR          ; プロセッサモードレジスタビット
;
PM0  .EQU 0004H     ; プロセッサモードレジスタ０
PM1  .EQU 0005H     ; プロセッサモードレジスタ１
;
CM0  .EQU 0006H     ; クロックモードレジスタ０
CM1  .EQU 0007H     ; クロックモードレジスタ１
C_CM0.EQU 00001000B ; CM0(Xcin-HIGH)
C_CM1.EQU 00100000B ; CM1(Xin-HIGH)
;
     .section   program
     .glb start
;
;*********************************************
;*    初期化ルーチン                         *
;*********************************************
;
start:
; ----- set interruptstack -----
     LDC  #intstack,ISP        ; 割り込みスタックポインタセット
; ----- set Processor mode -----
     BSET       PRC1       ; プロセッサモードレジスタ書き込みイネーブル
     MOV.B      #00000000b,PM0  ; シングルチップモード
     MOV.B      #00000000b,PM1  ; 非拡張、ノーウェイト
     BCLR       PRC1       ; プロセッサモードレジスタ書き込みディゼーブル
; ----- set System clock -----
     BSET       PRC0       ; クロックコントロールレジスタ書き込みイネーブル
     MOV.B      #C_CM0,CM0 ; 発信
     MOV.B      #C_CM1,CM1 ; 分周なし
     BCLR       PRC0       ; レジスタ書き込みディゼーブル
;
     .glb _main
;
;    ***** メインルーチンへ     *****
;
     jsr.a_main          ; --> main()
;
;*********************************************
;*    ダミープログラム                       *
;*********************************************
     .glb dummy_int
dummy_int:
     reit
;
     .end
```

ヘッダ：ファイル名や作成記録を記述します。
コメントは“；”の後に記述します。

ＳＦＲのアドレスやビットを定義します。

ここからのセクションを“program”という名前にします。

“start”というラベルを他のファイルから参照できるようにします。

M16C の初期設定です。
（ここからが実際に M16C で動かすプログラムです。）

“main”というラベルを他のファイルから参照できるようにします。

サブルーチンコールで main プログラムを呼び出します。

使用していない割り込みのダミープログラムです。何もしないで割り込みから復帰します。

プログラム記述の終了を示します。

4.3.4. ベクタテーブルファイル（vector.a30）の解説

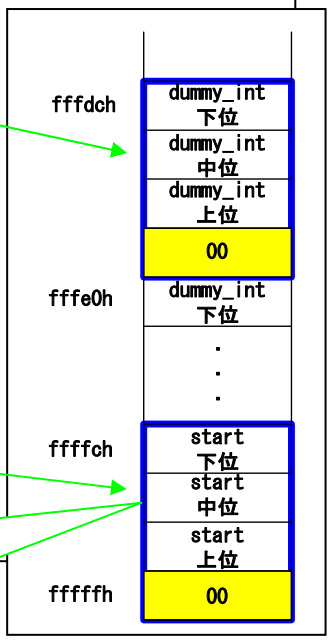
このファイルは、ベクターテーブルの定義をする為のファイルです。M16C/62A はリセットがかかると、リセットベクタと呼ばれるメモリのアドレスを読み込み、その内容をプログラムカウンタにセットしてプログラムを実行していきます。そのため、プログラムの先頭アドレスを、リセットベクタに設定しなければなりません。ベクタテーブルは、それぞれ 4 バイトずつデータ領域を持っているので、設定には指示命令“.lword”を使用します。

```
;/*-----------------------------------------------------------------------*/
;/* ファイル名： vector.a30                                               */
;/* 内容：       ベクタテーブル定義                                       */
;/* 日付：       2001.9.19                                                */
;/* 作成者：           OAKS16KIT support                                  */
;/* コメント：使用しないベクタは dummy_int を設定                         */
;/*-----------------------------------------------------------------------*/
;
        .glb    start
        .glb    dummy_int
        .section    fvector
        .org    0fffdch
;
        .LWORD  dummy_int       ; FFFDC～F Undefined instruction
        .LWORD  dummy_int       ; FFFE0～3 Overflow
        .LWORD  dummy_int       ; FFFE4～7 BRK instruction
        .LWORD  dummy_int       ; FFFE8～B Address match
        .LWORD  dummy_int       ; FFFEC～F Single step
        .LWORD  dummy_int       ; FFFF0～3 Watchdog timer
        .LWORD  dummy_int       ; FFFF4～7 DBC
        .LWORD  dummy_int       ; FFFF8～B NMI
        .LWORD  start       ; FFFFC～F RESET
;
        .end
```

ヘッダ：ファイル名や作成記録を記述します。

“start”と“dummy_int”は他のファイルを参照することを宣言します。

以下の記述内容を“fvector”というセクションで fffdch から配置する宣言です。



下位、中位、上位の順で配置される。余った上位ビットには全て“0”が入る。

4.4. C言語プログラムの基礎知識

4.4.1. sampleAで使用するC言語記述

ここで使われるＣ言語記述について説明します。Ｃ言語のプログラムはフリーフォーマット形式なので、ある一定の規則を守れば、後は自由にプログラムを記述できます。

C 言語の規則

(1)　プログラムは原則的に英小文字で記述する。
(2)　プログラムの実行文は「；」で区切る。
(3)　関数や制御文の実行単位は｛　｝で囲む。
(4)　関数や変数は型宣言が必要である。
(5)　予約語は識別し（関数名、変数名など）に使用できない。
(6)　コメントは「／＊　コメント　＊／」で記述する。

関数

C 言語では「関数」がプログラムの基本単位となります。関数を使用するためには「関数の型宣言（プロトタイプ宣言)」、「関数の定義」、「関数の呼び出し」の 3 つの手続きが必要です。

関数の型宣言（プロトタイプ宣言）　・・・　関数を使用する前に行なう。

＜書式＞

戻り値のデータ型　関数名（引数のデータ型のならび）；

＊戻り値や引数がないときは、空（から）を意味する“**void**”という型を記述する。

関数の定義（関数本体）

＜書式＞

```
戻り値のデータ型　関数名（仮引数１のデータ型　仮引数，・・・）
 {
                .
                .
                .
        return　戻り値；
}
```

関数の呼び出し

<書式>

関数名（引数 1…,）；　または　変数＝関数名（引数 1…,）；

test.c で使用する C 言語記述

① main () {
}

これは C 言語プログラムの中心になる main 関数です。この関数がプログラム全体の中心となり、ここからさらに、ほかの関数を呼び出したりします。
main の後の（）は引数が空であることを示します。引数がなくても（）が必要です。
｛　｝で囲まれた部分は関数の本体と呼び、処理手順等を記述します。

② for(式 1；式 2；式 3)｛･･･｝

繰り返しを行なう命令です。式 1 は for 文に入って最初に実行される為、初期値の設定に使われます。式 2 は for 文の繰り返しを終了する条件式です。式 3 は｛･･･｝が実行された後で実行される為、カウンタのインクリメントなどに使われます。式 2 の繰り返しの先頭で行なわれています。 f or 文の式 1、式 2、式 3 はどれも書かないで済ますこともできます。書かない場合には、その式がないものとみなされます。その結果 for（；；）｛･･･｝は無限ループと判断されます。

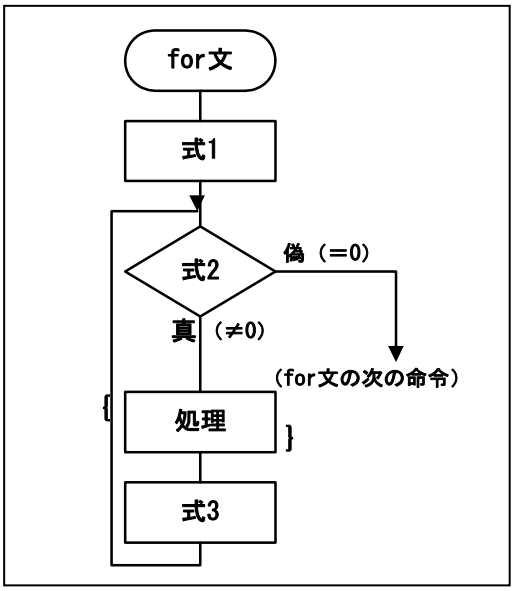


図 4-16

③ if(式)　文；

制御の流れを 2 方向に変える命令です。条件式が真のとき文を実行します。偽のとき、文を実行せずに次の命令に進みます。
If 文は文が一つの場合はセミコロン（；）で終わります。文が複数の場合は、それぞれの文をセミコロンで終わり、まとまりをカッコ｛｝でくくります。

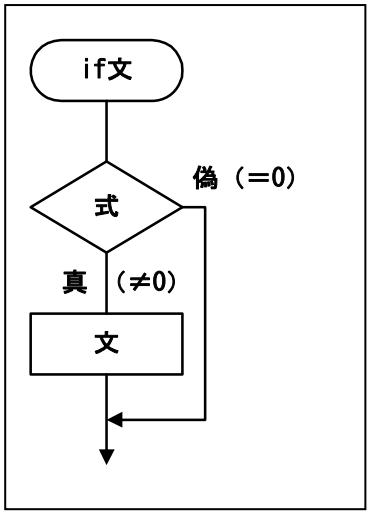


図 4-17

④ break；

繰り返しを中断してループから抜け出します。

⑤ ＃pragma ADDRESS　変数名　絶対アドレス

変数名を絶対アドレスに配置します。

⑥ ＃define

マクロ定義を行ないます。

⑦ 演算子

表 4-1

演算子	記述形式	内容
＝	式1＝式2	式2の値を式1へ代入する
－－	－－変数 変数－－	変数の値をディクリメント（-1）する
＝＝	式1＝＝式2	式1の値が式2の値と等しければ真、それ以外は偽

⑧ NC30 の予約語

（NC30 で使用しているため変数や関数名に使用できない記述です）

表 4-2

_asm	const	far	regisuter	switch
_far	continue	float	return	typedef
_near	default	for	short	union
asm	do	goto	signed	unsigned
auto	double	if	sizeof	void
break	else	int	static	volatile
case	enum	long	struct	while
char	extem	near		

4.4.2. test.cの解説

sampleA の main プログラムファイル（test.c）の説明をします。

フローチャート

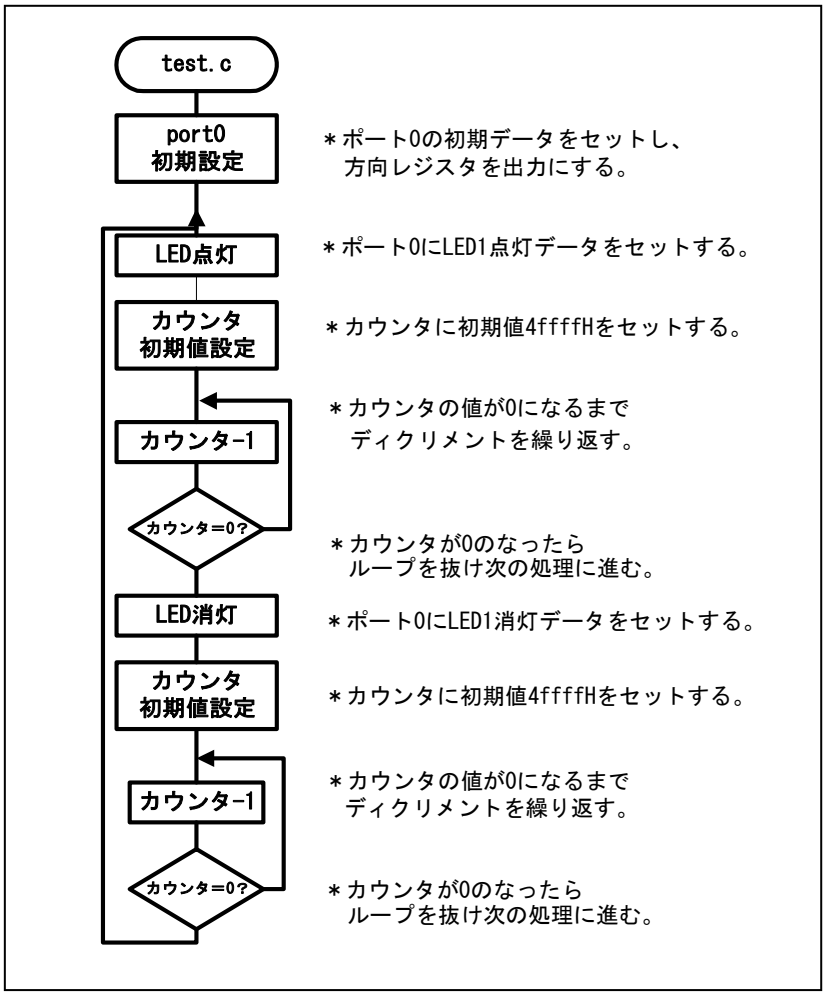


図 4-18

ポートの使い方（データ出力）

LED1 が接続されているポート P0 は、入出力兼用レジスタです。ポート P0 方向レジスタで、入力ポートとして使用するか、出力ポートとして使用するかを設定します。出力ポートとして設定した場合には、データレジスタに値を設定すれば、その値がポートの端子から出力されます。

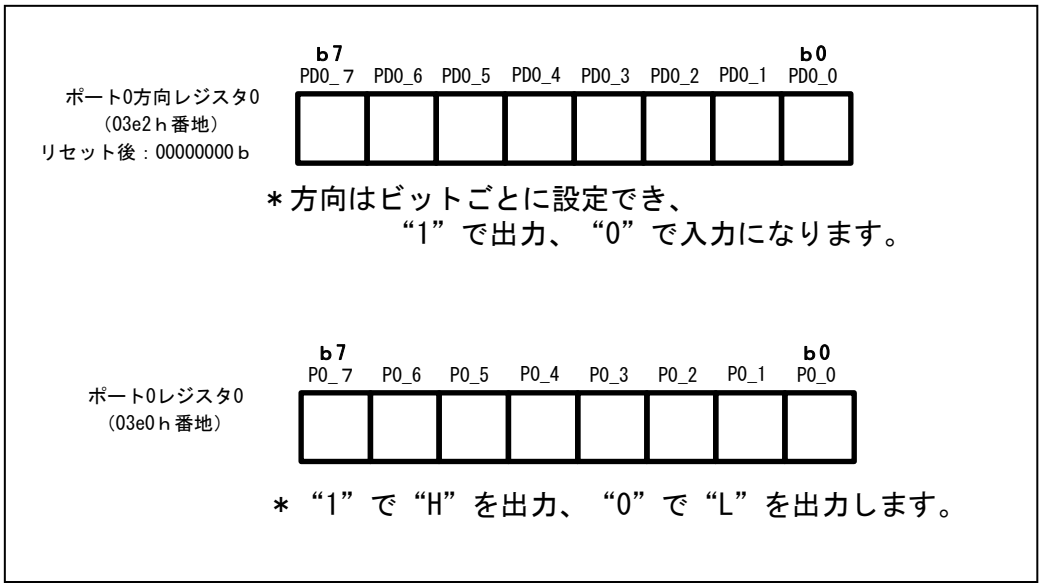


図 4-19

今回の回路は、ポート P00 に LED1 が接続されているだけで、P01 から P07 までの 7 ビットは何も接続されていません。ですから、P01 から P07 までは、入出力どちらに設定しても、どんなデータを設定しても、何も問題はありません。ここで、ポートへのデータの出し方ですが、NC30 というコンパイラはビット演算もサポートしているので、1 ビットだけ処理することも可能です。ですが、今回のプログラムでは、8 ビットまとめて、代入演算子で処理する方法をとります。

方向レジスタは全て出力（11111111ｂ）にします。LED 出力データは、LED1 以外は全て“H”データを出力することにし、点灯データ（11111110ｂ）、消灯データ（11111111ｂ）とします。

プログラム記述例

```
/*-------------------------------------------*/
/*  ファイル名：      test.c                 */
/*    内容：          LED 点灯（ソフトウエイト） */
/*    日付：          2001.9.19              */
/*    作成者：        OAKS16KIT support      */
/*-------------------------------------------*/

/* SFR 定義 */
#pragma ADDRESS p0   3e0H   /* Port P0 register  */
#pragma ADDRESS pd0  3e2H   /* Port P0 direction register */

/* プロトタイプ宣言 */
void _main(void);

/* 変数の宣言 */
unsigned char p0, pd0;


/* マクロ定義 */
#define LED_on  0xfe            /* LED 点灯 */
#define LED_off 0xff            /* LED 消灯 */

main(){
     unsigned long i;

     p0 = LED_off;              /* ポート 0 出力 H(LED 消灯)*/
     pd0 = 0xff;                /* ポート 0 方向出力 */

     for(;;)
     {
          p0 = LED_on;          /* LED 点灯 */
          for(i=0x4ffff;;){     /* 時間待ち(ソフトウエイト) */
               i--;
               if(i==0) break;
          }
          p0 = LED_off;         /* LED 消灯 */
          for(i=0x4ffff;;){     /* 時間待ち（ソフトウエイト）   */
               i--;
               if(i==0) break;
          }
     }

}
```

ヘッダ：ファイル名や作成記録を記述する。

ポート 0 の方向レジスタとデータレジスタのアドレス設定

このファイルで使用される関数の宣言です。

ポート 0 の方向レジスタとデータレジスタに値を入れるために変数として宣言します。

LED 点灯データ･消灯データをマクロ定義します。

カウンタとして使用する変数を宣言します。

ポート 0 初期設定

LED 点灯

ウエイト：ダウンカウント

LED 消灯

ウエイト：ダウンカウント

4.5. メモリ配置の基礎知識

OAKS16 で開発するプログラムは、最終的に ROM 化して動作させます。そのため、プログラムやデータをどこに配置するかが重要になります。基本的には、右の図が示す用に、ワーク用のデータ領域（カウンタなど）プログラム実行中のみ使用するものははは RAM に配置します。プログラムや固定データのように、プログラム実行途中で変更されないもの、電源を切ってもデータを保持したいものは ROM に配置します。

これらの配置は、最終的にリンカで決定されます。そのリンカに渡す情報として、“セクション”があります。このセクションの決め方は、アセンブラ記述と、C 言語記述では異なります。

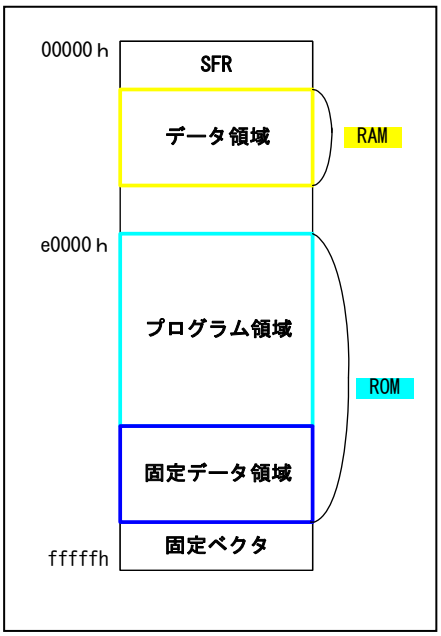


図 4-20

①アセンブラのセクションの決め方

“.SECTION”命令をソースプログラム内に記述することで決定します。

②C 言語でのセクションの決め方

コンパイラが自動的に設定します。

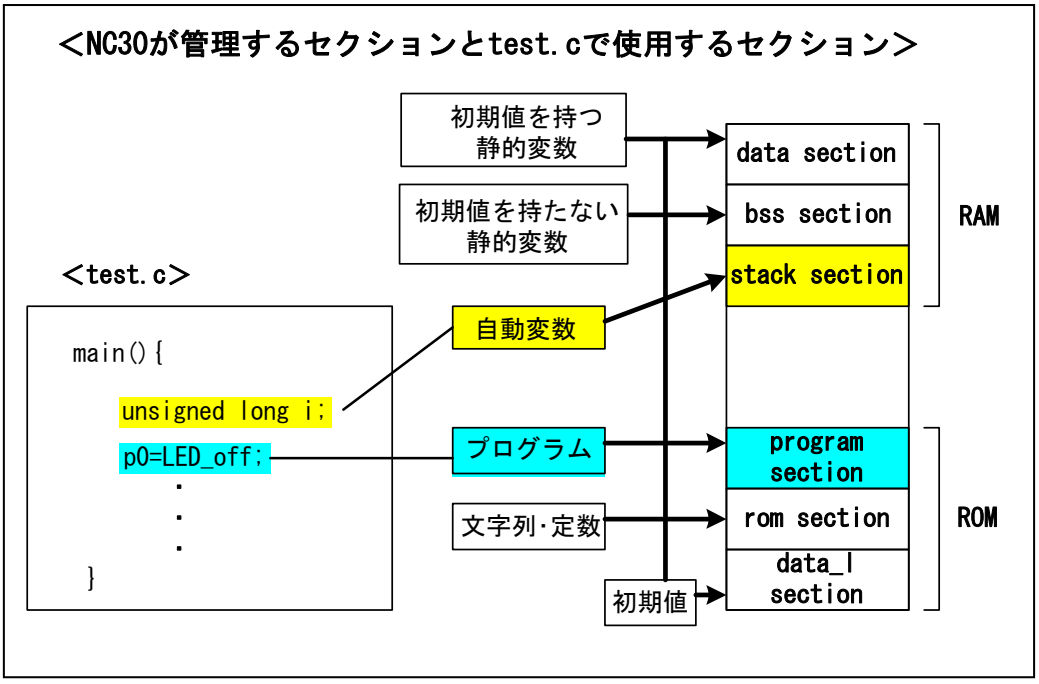


図 4-21

ファイル内のセクション記述と、メモリ配置

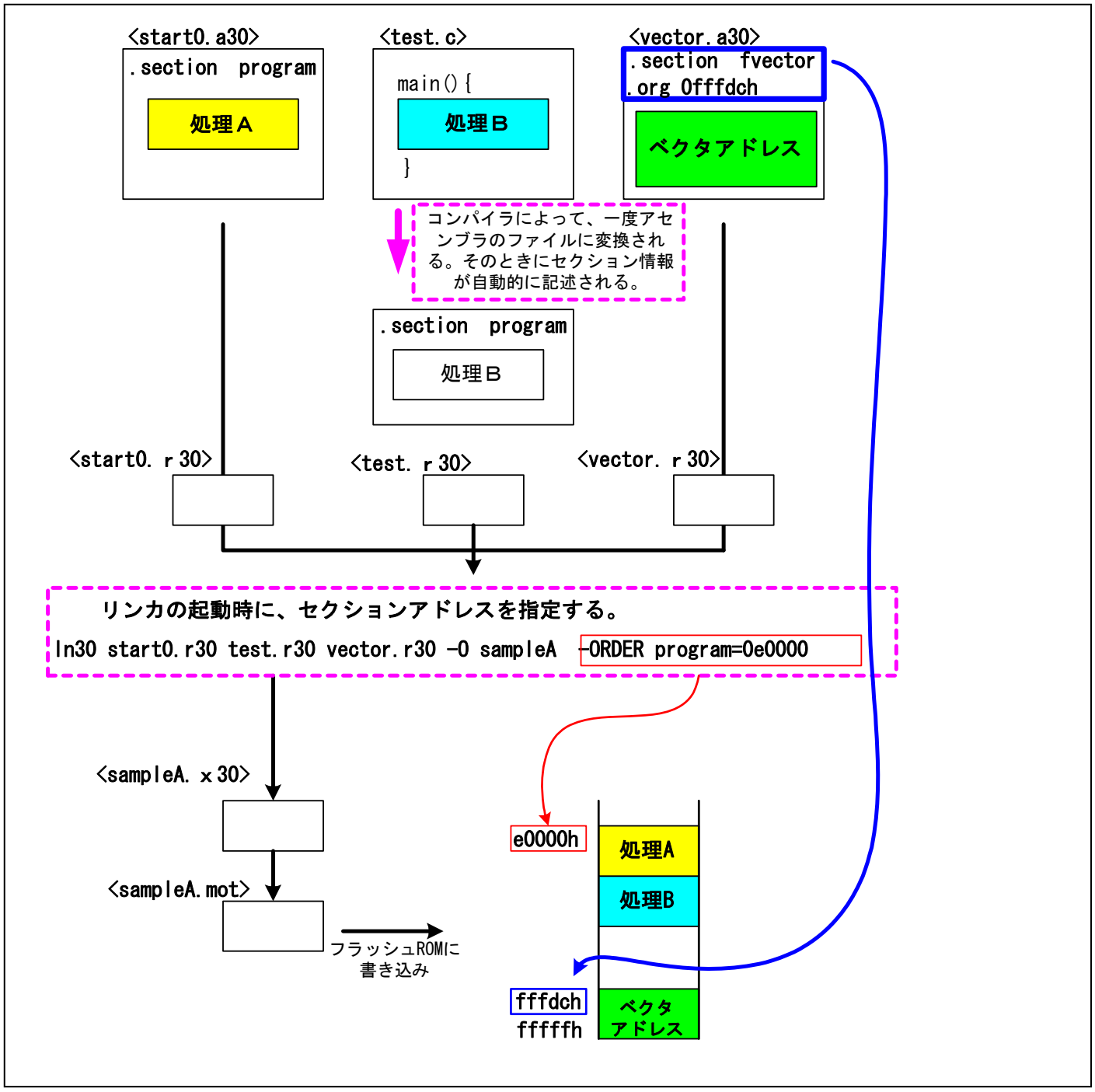


図 4-22

複数のファイルをコンパイル、アセンブル、リンクという一連の処理を行なうことによって一つのオブジェクトファイルを作成します。そこで、プログラムやデータをメモリのどこに配置されるかを決める方法として、アセンブラリストの中に指示命令“.org”で記述する方法と、リンカのオプション“-order”でアドレスを設定する方法があります。

固定ベクタなど、変更の必要のないものは、“.org”で設定しておきます。

プログラムやデータのアドレスなどは、変更がしやすいように、リストの中ではセクションだけを決めておき、リンク時にアドレスを指定するか、スタートアッププログラムに“.org”で配置する部分をまとめておき、そこだけを変更すればよいようにしましょう。

ここで説明している test.c はスタックを使用する自動変数しか使用していませんので、データ領域のセクションの配置は必要ありません。ですが、C 言語のプログラムの中で、静的変数、定数、初期値を持つ静的変数などを使用する場合には、アセンブリ言語のスタートアッププログラムで、セクションの配置と初期化を行なう必要があります。今回の説明に使用した start0.a30 にはデータ領域の初期化は含まれていません。静的変数、定数、初期値を持つ静的変数などを使用する場合には、NC30 に標準で添付されている“ncr0.a30”というスタートアッププログラムを使用することをお勧めします。使用方法は、TM でプロジェクトを作成するときに、“標準のスタートアッププログラムを使用する”を選択すれば、自動的にプロジェクトのフォルダにコピーされます。そこで、『M16C/60 M16C/20 シリーズ＜C 言語編＞プログラミングマニュアル　第２章 ROM 化技術』を参考にスタートアッププログラムの変更を行なってください。

5. Ｉ／Ｏ制御の演習

ここで、fullkit の EXBOARD の準備をします。EXBOARD についている部品を接続し、制御する為の回路を作成します。

5.1. LED

ここでは代表的な出力装置として LED をとりあげ、マイコンとの接続の方法、プログラムでの制御の考え方を説明していきます。

5.1.1. LEDの特性

LED は Light Emitting diod（発光ダイオード）の略では一定の電流を流すと発光する部品です。LED はダイオードの一種ですので極性があります。足の長い方が『アノード』といい電源につなぎます。足の短い方が『カソード』といい、GND につなぐ方です。電流は、アノードからカソードに向かってしか流れません。

5.1.2. LEDの接続回路

次に、LED の一般的な接続回路を示します。LED は、M16CA/62（マイコン）のポートに接続します。ここで LED とマイコンの間に抵抗が入れてあります。この抵抗がないと、LED とマイコンに多量の電流が流れ、壊れることがあります。そのために、電流を制限する為の抵抗を入れます。(『電流制限抵抗』と呼びます。)

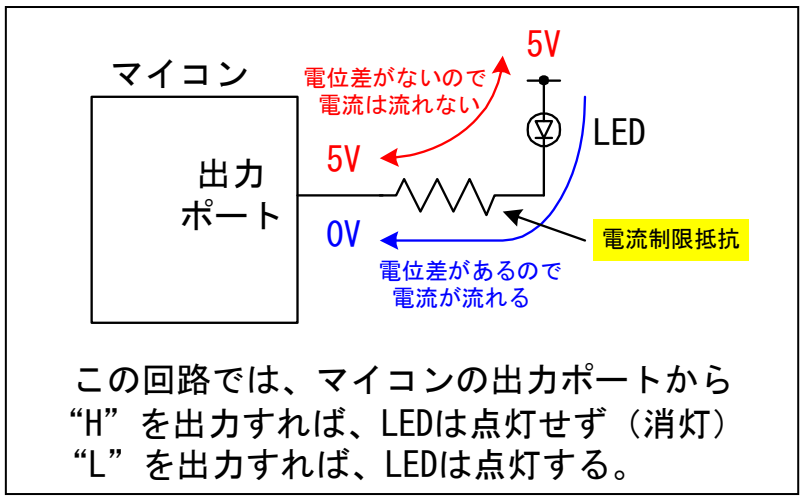


図 5-1

5.1.3. 電流制限抵抗の求め方

LED の制限抵抗を求めるには、まず、LED のデータシートで特性を調べます。

表 5-1

LED（SEL111R）の光特性（データシートより）

順電圧		
Typ	Max	条件
2.0（V）	2.5（V）	10(mA)

ここから、10mA の電流を流したとき、約 2.0(V)の電圧降下がおこる事がわかります。
次に、ポートの推奨動作条件を調べます。

表 5-2

M16C/62A 推奨動作条件（データシートより）

記号	項目	規格値（最大）
IOH（peak）	“H”尖頭出力電流	-10.0(mA)
IOH（avg）	“H”平均出力電流	-5.0(mA)
IOL（peak）	“L”尖頭出力電流	10.0（mA）
IOL(avg)	“L”平均出力電流	5.0（mA）

M16C/62A でポートから出力する電流の推奨値は５mA ですので、ここでは、5ｍＡで計算してみます。

R(抵抗値)＝Ｖ（電源電圧）－V_{LED}（ＬＥＤの電圧降下）／Ｉ（電流）
Ｒ＝（5-2）/0.005＝600（Ω）
ということで 600Ω位の抵抗をつければよいことになります。
OAKS16 の EXBOARD についている抵抗は、1ｋΩですので逆に計算してみると、1000＝（5-2）/Ｉで I=約 3.3mA 位の電流が流れることになります。電流は、流しすぎるとマイコンやＬＥＤをこわす原因になりますが、少なくても、輝度が問題なければＯＫです。実際につけていただければ、それほど問題のない輝度を得られることがわかっていただけると思います。
というわけで、LED の制御回路では計算した制限抵抗値より大きい値もので、近い値の抵抗をつけていただければよいということになります。

<参考＞

代表的な電子部品である**抵抗**は、その用途に応じて多くの種類が存在します。ここで使用しているのは、制度はあまりよくありませんが安価でよく使われる『**炭素皮膜抵抗器**』を使用しています。抵抗には一目で抵抗値がわかるようにカラーコードが表示されています。

抵抗カラーコード表

表 5-3

色	第 1 数字	第 2 数字	第 3 数字
黒	0	0	10^0 （1）
茶	1	1	10^1 （10）
赤	2	2	10^2 （100）
橙	3	3	10^3 （1000）
黄	4	4	10^4 （10000）
緑	5	5	10^5 （100000）
青	6	6	10^6 （1000000）
紫	7	7	10^7 （10000000）
灰	8	8	10^8 （100000000）
白	9	9	10^9 （1000000000）

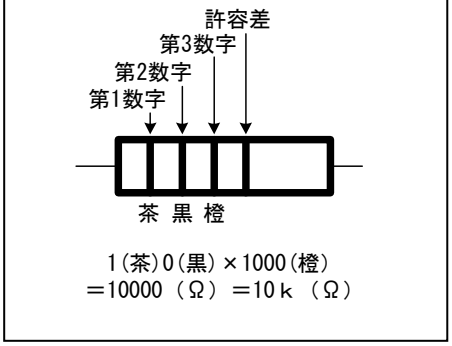


図 5-2

5.1.4. LEDの回路接続（演習 1）

下の回路図の通りに、LED2、LED3 を EXBOARD に接続して下さい。M30620FPAFP の端子はコネクタで EXBOARD に接続していますので、LED と接続した抵抗のもう一方の端子を回路図に示すポートが接続されるコネクタに接続してください。P70 は CN１の 26 番ピン、P71 は CN1 の 25 番ピンに接続されています。（OAKS16EXBOARD マニュアル参照）

LED 回路図

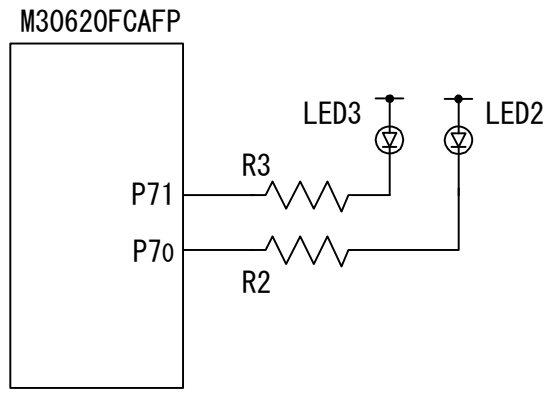


図 5-3

これで、LED の制御回路ができました。
この LED は、ポートから“H”が出力されると消灯し、“L”が出力されると点灯します。
これは、プログラムを作成するときに、非常に重要な情報です。この後説明するスイッチの回路の情報と一緒に『インターフェース仕様書』としてまとめておくと便利です。）

表 5-4

インタフェース仕様

	点灯	消灯
LED2（P70）	L	H
LED3（P71）	L	H

5.1.5. LED点灯プログラム（演習 2）

LED2 と LED3 を交互に点灯させるプログラムを作成して下さい。点灯の交代時間は、変化が分かる速さであれば OK です。

① ソースファイルの準備

ソースファイルは、sampleA を参考にしましょう。スタートアッププログラムと、ベクタファイルは同じものをそのまま使い、I/O を制御するメインプログラムだけを変更します。

(ｱ) ハードディスクに新フォルダを作成する。フォルダ名は TM のプロジェクト名になるのでここでは“ensyuA”とする。¥ensyuA に OAKS16CDROM 内¥sampleA から 3 つのソースファイル（test.c、start0.a30、vector.a30）をコピーする。

(ｲ) test.c の内容を変更するので、ファイル名を ensyuA.c に変更する。（内容の異なるプログラムを同じファイル名で持つことは、間違いの原因になりやすい。）

(ｳ) ensyuA.c をエディタで、仕様に合わせた内容に書き換える。

② コンパイル

TM でプロジェクトを作成し、コンパイルする。これにより、デバッガ（KD30）上で動作させるファイル（ensyuA.x30）とフラッシュ ROM ライタ（flashstart）で書き込むためのファイル（ensyuA.mot）が作成される。

③ デバッグ

デバッガで動作を確認する。

④ ROM 焼

フラッシュ ROM にプログラムを書き込み単体で動作させる。

以上が演習手順です。

演習 2 のサンプルプログラムはこのテキストに添付してあります。（ensyuA）ハードディスクのルートディレクトリにフォルダごとコピーして、TM で動作を確認してください。

＜参考＞

制御部分のフローチャート

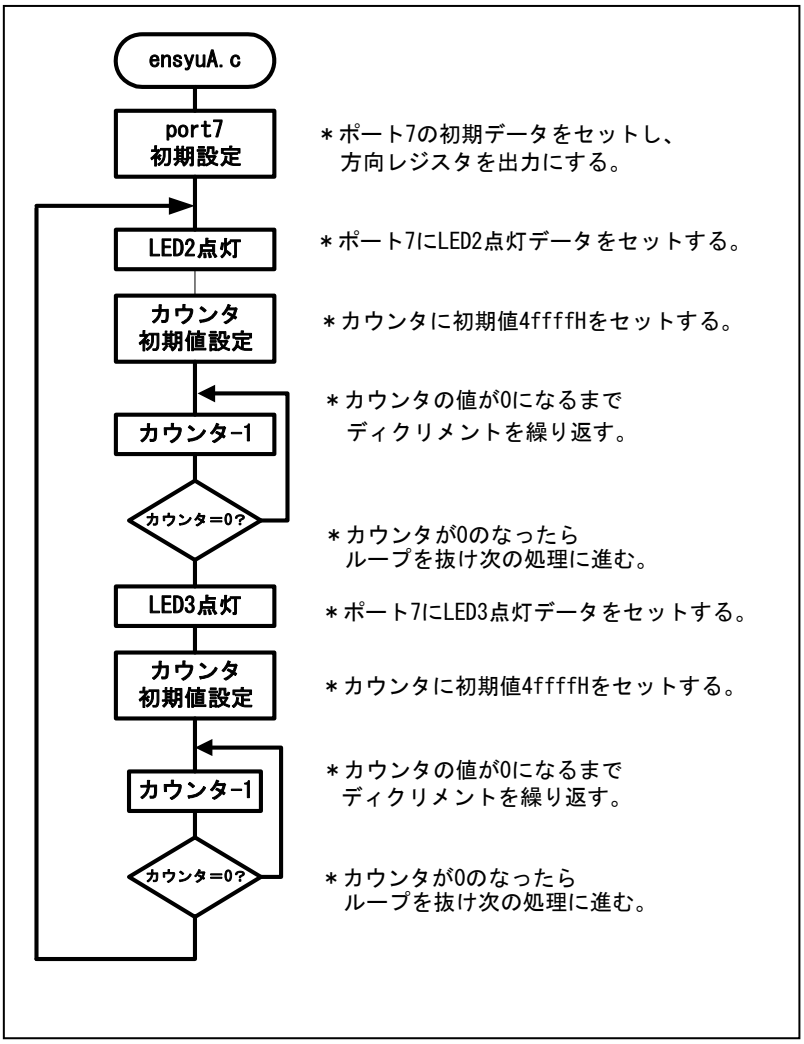


図 5-4

制御プログラム（ensyuA.c）リスト

```
/*------------------------------------------------------------------*/
/*   ファイル名：ensyuA.c                                           */
/*     内容：     LED2.3 点灯（ソフトウエイト）                     */
/*     日付：     2001.12.4                                         */
/*     作成者：   OAKS16KIT support                                 */
/*------------------------------------------------------------------*/

/* SFR 定義 */
#pragma ADDRESS p7          3edH        /* Port P7 register           */
#pragma ADDRESS pd7         3efH        /* Port P7 direction register */

/* プロトタイプ宣言 */
void _main(void);

/* 変数の宣言 */
unsigned char p7, pd7;


/* マクロ定義 */
#define LED_off 0xff            /* LED2.3 消灯 */
#define LED2_on     0xfe        /* LED2 点灯、LED3 消灯 */
#define LED3_on     0xfd        /* LED2 消灯、LED3 点灯 */

main(){
    unsigned long i;

    p7 = LED_off;                       /* ポート 7 出力 H(LED 消灯)    */
    pd7 = 0xff;                         /* ポート 7 方向出力            */

    for(;;)
    {
        p7 = LED2_on;                   /* LED2 点灯     */
        for(i=0x4ffff;;){               /* 時間待ち(ソフトウエイト)     */
            i--;
            if(i==0) break;
        }
        p7 = LED3_on;                   /* LED3 点灯     */
        for(i=0x4ffff;;){               /* 時間待ち（ソフトウエイト）   */
            i--;
            if(i==0) break;
        }
    }

}
```

P7 のアドレスを定義します。

ポート 7 の方向レジスタとデータレジスタに値を入れるために変数として宣言します。

LED に出力するデータを定義します。

P7 初期設定

LED2 点灯

LED3 点灯

5.2. スイッチ

OAKS16 EXBOARD には三種類のスイッチが用意されています。

① **トグルスイッチ**：ＯＮ、OFF の状態が固定できます。

② **8 素子ディップスイッチ**：ＯＮ、ＯＦＦの状態が固定できます。ﾄｸﾞﾙｽｲｯﾁが 8 個一緒になって小さな部品になったようなものです。

③ **プッシュスイッチ**：一番よく使われるスイッチです。押している間だけＯＮになります。

これらを M16C に接続する方法を考えましょう。

スイッチも LED と同様にポートに接続します。ですが、スイッチはこれを使ってマイコンに情報を与える部品です。スイッチが ON のとき、OFF のときの変化がマイコンに伝わるような回路を作成する必要があります。

5.2.1. スイッチの回路例

一般的なスイッチの接続回路例を示します。

スイッチ回路例 1

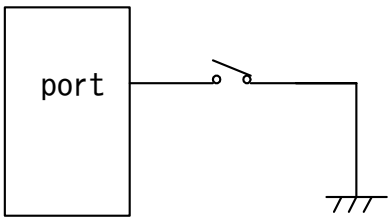


図 5-5

スイッチ回路例 2

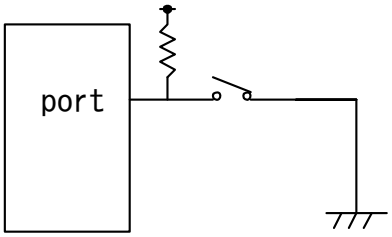


図 5-6

回路例 1 では、ON のときの電圧は“L”になりますが、OFF の時の電圧が決まりません。そこで、回路例 2 の様に抵抗を入れて電源に接続し、OFF の時“H”に決まるように配線します。

ここで、抵抗が入っていないと、スイッチを ON にした時、ショートしてしまいます。この抵抗をプルアップ抵抗といい、スイッチ回路では必要な回路です。プルアップ抵抗には、数ｋΩから数十ｋΩの抵抗を使います。

この後、詳しく説明しますが、OAKS16 に使われているマイコンでは、ポートにプルアップ抵抗が内蔵されており、これを使うこともできます。

5.2.2.M16C/62Aのプルアップ抵抗

M16C/62A のポートは内部にプルアップ抵抗を内蔵しており、プルアップ制御レジスタを設定することで、4 ビットごとにプルアップ抵抗の有無を設定できます。プルアップ制御レジスタは、プルアップ制御レジスタ 0（03FCh 番地)、プルアップ制御レジスタ 1（03FDh 番地)、プルアップ制御レジスタ 2（03FEh 番地）の 3 つで、それぞれ指定されたビットを“1”にすることでプルアップ抵抗を使うことができます。

下記に、$P3_0$、$P3_1$ にスイッチをつけた場合のプルアップ抵抗の設定を示します。ポート P3 のプルアップ制御レジスタはプルアップ制御レジスタ 0 です。$P3_0$ と $P3_1$ はビット 6 に割り当てられていますので、ビット 6 を“1”にします。$P3_2$、$P3_3$ は何も接続されていなくても 4 ビットごとの設定しかできませんので、強制的にプルアップされることになります。

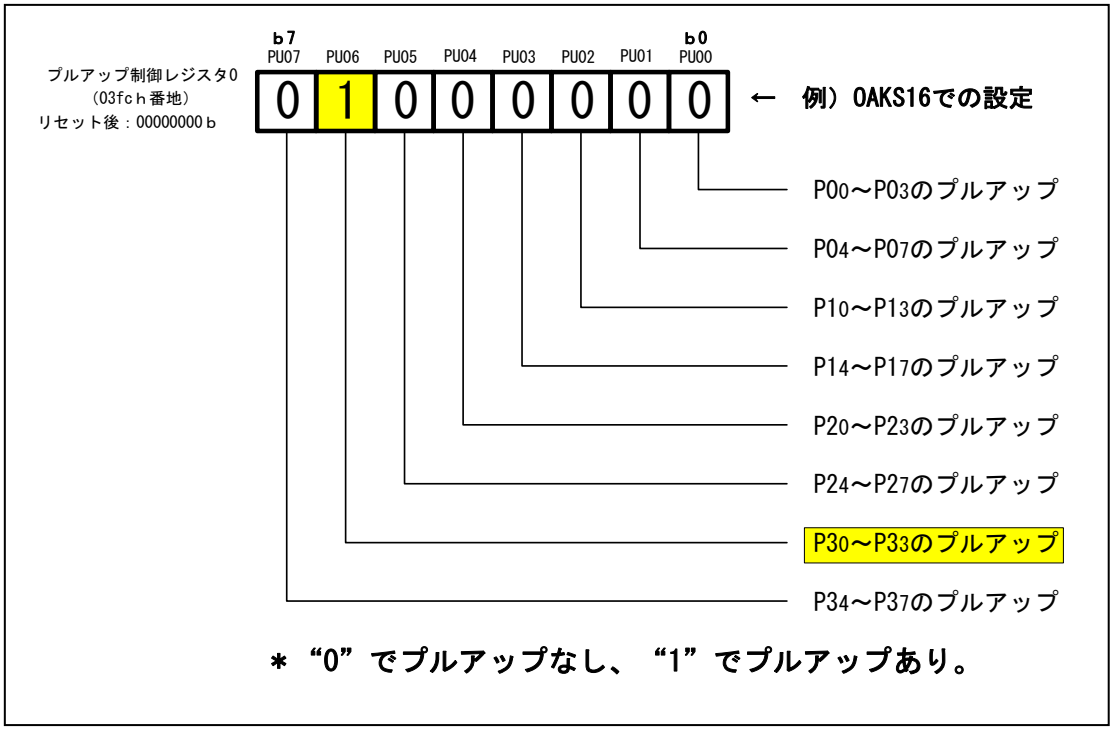


図 5-7

プルアップ抵抗値は、データシート P175『電気的特性』より、標準で 50.0kΩ です。

5.2.3. スイッチの回路接続（演習 3）

下の回路図の通りに、OAKS16EXBOARD の SW2、SW3 を接続して下さい。M30620FPAFP の端子はコネクタで EXBOARD に接続していますので、スイッチの一方を GND に、もう一方を回路図に示すポートが接続されるコネクタに接続してください。P30 は CN3 の 24 番ピン、P31 は CN3 の 23 番ピンに接続されています。（OAKS16EXBOARD マニュアル参照）
これらのスイッチは、OFF の時に“H”、ON の時に“L”になります。
プルアップ抵抗は、M16C/62A の内部プルアップ抵抗を使用します。

SW 接続

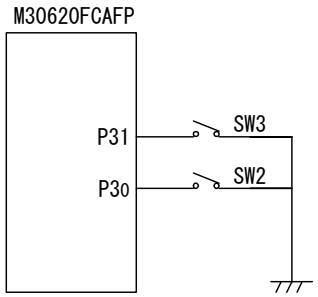


表 5-5

スイッチ 2 の状態とポート P30 の入力データ

	P30
SW2 が off	H
SW2 が on	L

スイッチ 3 の状態とポート P31 の入力データ

	P31
SW3 が off	H
SW3 が on	L

図 5-8

5.2.4. スイッチとLEDの制御プログラム（演習 4）

SW2 と LED2、SW3 と LED3 を対応させ、それぞれがＯＮのときに対応する LED が点灯するプログラムを作成してください。（OFF の時は、対応する LED は消灯です。）
演習手順は、4.1.5.LED 点灯プログラムと同じです。（プロジェクト名を ensyuB とします。）

データの考え方

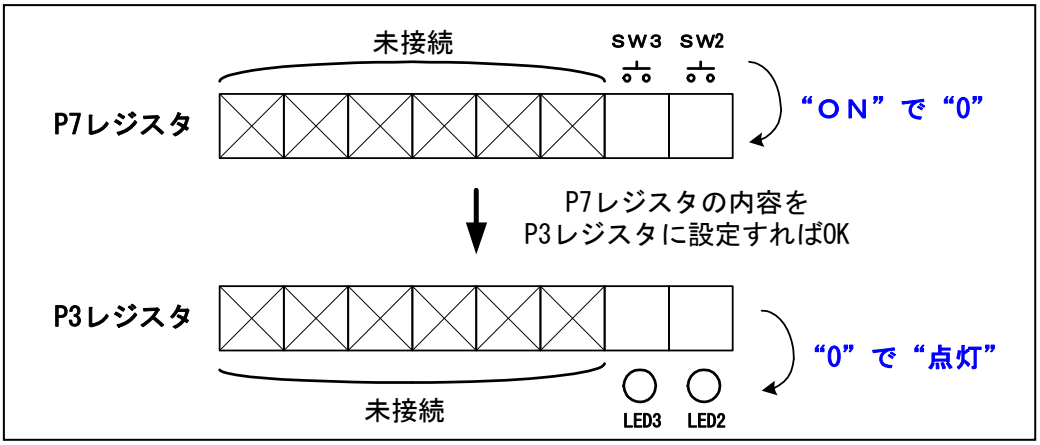


図 5-9

制御プログラムのフローチャートとリスト

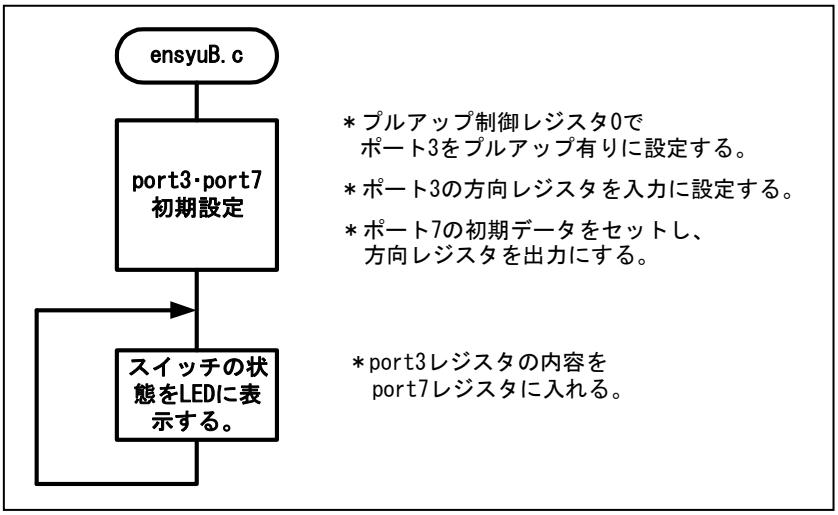


図 5-10

```
/*-------------------------------------------------------------------*/
/*   ファイル名：ensyuB.c                                            */
/*      内容：      LED と SW の制御                                 */
/*      日付：      2001.12.4                                        */
/*      作成者：    OAKS16KIT support                                */
/*-------------------------------------------------------------------*/

/* SFR 定義 */
#pragma ADDRESS p3          3e5H        /* Port P3 register          */
#pragma ADDRESS pd3         3e7H        /* Port P3 direction register */
#pragma ADDRESS p7          3edH        /* Port P7 register          */
#pragma ADDRESS pd7         3efH        /* Port P7 direction register */
#pragma ADDRESS pur0        3fcH        /* Port P0-3 pullup register */

/* プロトタイプ宣言 */
void _main(void);

/* 変数の宣言 */
unsigned char p3,pd3,p7, pd7,pur0;


/* マクロ定義 */
#define     LED_off         0xff        /* LED2.3 消灯 */
#define     p3pullup        0x40        /* p3b0-b3 pullup data */
#define     p3in            0x00        /* pd3 input   */
#define     p7out           0xff        /* pd7 output */

main(){

        pur0 = p3pullup;    /* ポート 3 をプルアップ抵抗ありに設定 */
        pd3 = p3in;         /* ポート 3 方向入力 */
        p7 = LED_off;       /* LED 消灯 */
        pd7 = p7out;        /* ポート 7 方向出力 */

        for(;;)
        {
                p7 = p3;    /* スイッチの状態を LED に出力 */
        }

}
```

スイッチが接続されているポート３のアドレスの定義を追加します。

プルアップ制御レジスタのアドレスを定義します。

ポートのデータレジスタ、方向レジスタ、プルアップレジスタに値を設定する為に変数宣言をする。

レジスタに設定する値をマクロ定義しておく。

ポートの初期設定

ポート 3 データレジスタの内容をそのままポート７データレジスタに入れる。

ファイルの変更が終了したら、TM を起動してプロジェクトを作成し、コンパイル、デバッグを行い、仕様通りの動作をするかどうか確認してください。動作が確認できたら flashstart でフラッシュ ROM に書き込み最終確認をして下さい。
演習 4 のサンプルプログラムはテキストに添付してあります。（ensyuB）ハードディスクのルートディレクトリにフォルダごとコピーして、TM で動作を確認してください。

以上で、OAKS16 基礎コースの説明と演習の解説は終了です。
さらに、M16C/62A の周辺機能、C 言語のプログラミングについてお知りになりたい方は OAKS16 応用テキストにお進み下さい。

ＯＡＫＳ16 キット

ＯＡＫＳ16 基礎テキスト（Ver. 1.01）

資料番号：ＯＡＫＳ16 基礎テキスト（Ver. 1.01）

発 行 所：オークス電子株式会社

〒101-0025　東京都千代田区佐久間町３丁目２１番地（第一千代田ビル３F）

ＴＥＬ：０３－３８６３－１１２１　　ＦＡＸ：０３－３８６３－１１３０

ホームページ　　http://www.oaks-ele.com