OAKS16プログラミングテキスト

目次

はじめに ... 4
1.C言語の基礎知識 ... 6
1.1.　　C言語とは ... 6
1.2.　　組み込み用のC言語とは ... 7
1.3.データ型 ... 8
1.3.1.NC30 で扱う定数 ... 8
1.3.2.変数 ... 10
1.4.演算子 ... 11
1.5.制御文 ... 12
1.6.関数 ... 16
1.7 配列 ... 19
1.7.1. １次元配列 ... 19
1.7.2. ２次元配列 ... 20
1.8.構造体 ... 22
1.8.1.ビットフィールドの宣言 ... 24
1.9.共用体 ... 25
2.OAKS16 でC言語を使って開発する前に ... 27
2.1.スタートアッププログラム ... 30
2.1.1.NC30 のサンプルスタートアッププログラムの構成 ... 30
2.1.2.スタートアッププログラムの作成 ... 31
2.2.プリプロセッサの処理 ... 42
2.2.1. sfr設定の為のヘッダファイル ... 43
3.プログラミング ... 49
3.1.基本のプログラム ... 49
3.1.1.論理演算 ... 52
3.1.2.データの反転 ... 53
3.1.3.データのマスク処理 ... 56
3.2.スイッチ回路の考え方 ... 59
3.2.1.チャタリングとは ... 63
3.2.2.S/Wでのチャタリング除去 ... 64
3.3.割り込みプログラム ... 66
3.3.1.M16Cの割り込み ... 67
3.3.2.割り込みを使用するために必要な処理 ... 68
3.3.1.INT0 割り込み端子からの割り込み ... 70

3.3.2.タイマを使った割り込み..76
3.4.LCDモジュールの制御..84
3.4.1.LCDモジュールの構成..84
3.1.2.OAKS16LCDBOARDの配線..85
3.1.3.LCDモジュールの初期設定..86
3.1.4.LCDモジュール制御の為の関数..88
3.1.5.サンプルプログラム１..95
3.1.6.サンプルプログラム２..104

はじめに

このテキストは、OAKS16 基礎テキストの続編として作成されています。OAKS16 基礎テキスト終了後にお使いください。
このテキストは、前半は、OAKS16 のプログラムを C 言語で開発するための基礎知識、後半は OAKS16 のプログラミングについての説明をしています。C 言語の文法等はある程度説明していきますが、さらに詳しい文法を知りたい方は、市販の C 言語の入門書をご覧下さい。
また、テキスト中の例題は全て、TM で動作できるようにフォルダで添付してあります。「OAKS16 で TM をお使いになる方のために」を参考にして、動作を確認してください。

LCDボードの準備

本テキストではプログラムのターゲットとして OAKS16LCD ボードを使用します。OAKS16LCD ボードをご購入いただくか、添付の資料を参考に同等の回路を作成して下さい。

①ボードの準備

基礎テキストで使用していた EXBOARD から CPU ボードを取り外し、LCD ボード上に差し替えてください。(必ず、電源回路をはずした状態で行なってください。) その後、電源、パソコンと再び接続してください。

②LCD ボード部品配置図

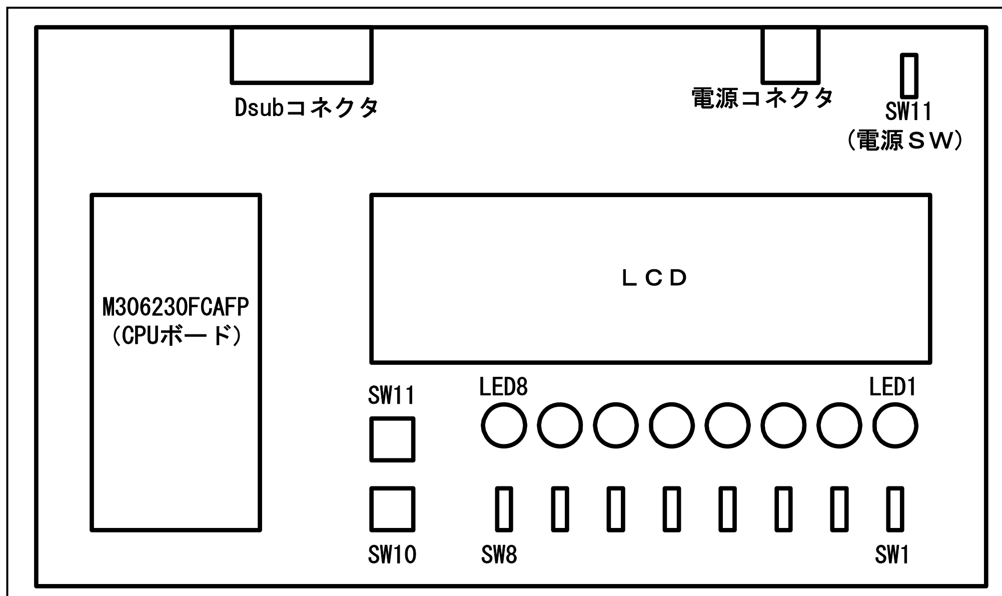


図 0.1

③ＬCD ボードインタフェース仕様書

LED1 から LED8

	点灯	消灯
LED1 (P00)	L	H
LED2 (P01)	L	H
LED3 (P02)	L	H
LED4 (P03)	L	H
LED5 (P04)	L	H
LED6 (P05)	L	H
LED7 (P06)	L	H
LED8 (P07)	L	H

SW1 から SW8

	ON	OFF
SW1 (P30)	L	H
SW2 (P31)	L	H
SW3 (P32)	L	H
SW4 (P33)	L	H
SW5 (P34)	L	H
SW6 (P35)	L	H
SW7 (P36)	L	H
SW8 (P37)	L	H

SW9,SW10

	ON	OFF
SW9 (P82)	L	H
SW10 (P83)	L	H

LCD モジュール

ー	ー	DB7(P43)	データバス
RS(P46)	RS=0(コマンド)、RS=1 (データ)	DB6(P42)	
R/W(P45)	R=1,W=0	DB5(P41)	
E(P44)	E=1 でイネーブル	DB4(P40)	

1.C言語の基礎知識

1.1.C言語とは

C 言語はミニコンピュータ用の OS である UNIX システムの開発用言語としてベル研究所の D.M. リッチーによって作られました。そのため C 言語は主に UNIX ユーザに多く使われてきました。その後、パソコン上の OS（windows など）上で動かす為のコンパイラが多く開発され、プログラミングの中心言語になってきました。一方、組み込み用のシステムでは、かなり長い間、アセンブリ言語が使われてきました。その理由は、アセンブリ言語が、機械語に一番近い言語で、プログラムサイズを小さくすることが出来たからです。組み込みシステムでは、メモリの容量が限られる為、出来るだけ小さいサイズのプログラムを作ることが求められていました。

しかし、近年組み込みシステムに使われる CPU も高性能になり、それに伴いアセンブリ言語でプログラムを記述することが難しくなってきました。その代わりに、メモリの作成技術も進歩し、容量の多いメモリが使えるようになりました。さらに、組み込み用の CPU のコンパイラも開発･改良され、比較的効率の良いプログラムを作成できるようになりました。

このような過程から、現在では、多くの組み込み用プログラムが C 言語で書かれています。

最近、書店のコンピュータ書籍のコーナーにはたくさんの C 言語の関連書が並んでいます。これらの本の大部分は、C 言語の文法を習得するためのものです。しかし、これらの本の例題はほとんどがパソコン上で動かす為のプログラムです。もちろん OAKS16 で開発するための C 言語の文法はこれらの本から学べるのです。しかし、汎用機（パソコン、ミニコンなど）で動かすプログラムと OAKS16 を動かすプログラムではいくつかの違いがあります。

本テキストでは、この違いを認識しながら、プログラミングの基礎技術を習得していただきます。

ANSI について

C 言語が普及されたのを受け、ANSI（American National Standards Institute：アメリカ国内標準規格協会）という委員会が発足しました。ここで決められた規格が、ANSI 規格と呼ばれるものです。ANSI 規格はコンパイラ作成のための規格ですが、コンパイラの仕様は、プログラムを記述する上で必要な知識です。現在、多くのコンパイラが ANSI 準拠を謳っていますが、＋αの機能を持ったコンパイラも多く存在します。今回使用する NC30 も、ANSI 準拠ですが、ANSI に規定されていない便利な機能も多く持っています。このテキストでは、全ての機能については説明できませんので、C 言語についてある程度理解できてきたら、NC30 のマニュアルにも目を通してください。

1.2.組み込み用のC言語とは

皆さんは、今までに C 言語の入門書をご覧になったことがおありでしょうか？まだの方は書店へ行ってほんの少しページをめくってください。ほとんどの本が CRT に“welcome C”もしくは似たような言葉を表示させるところから始まります。これは、C 言語を学習する為の環境として CRT、キーボードを標準の入出力として準備してあることが前提となっているからです。ANSI の企画書にもこれらの入出力装置を使うための関数が標準ライブラリ関数として仕様が規定されています。

しかし、皆さんにこれからプログラムを作っていただく OAKS16LCD ボードには、CRT もキーボードもありません。代わりになりそうなのは、LCD とスイッチそれに LED だけです。

はじめは、LCD を CRT の代わりとしてテキストを書いてみようと考えました。しかし、すぐに挫折することになりました。私たちが、LCD ボードにデータを表示させるのは、C 言語のテキストの著者が言うように一文の記述 printf(“welcome C”);ではだめなのです。なぜならば、ＣＲＴやキーボードのような規格が決まっているものは、コンパイラのサポート内ですが、OAKS16 の LCD ボードのような特殊な規格のものは、サポートされていないからです。そのために、OAKS16LCD ボードを使うためには、自分で制御用の関数を作らなければならないのです。そのためには C 言語の文法プログラミングの定石（？）そしてハードウエアの仕様を全て把握していかなければなりません。組み込み用の C 言語プログラマが大変なのは、複数の分野の知識を必要とするところなのです。

汎用マシンで動作させる為のC言語プログラム

① OS から起動し、終了後は OS に戻る。
② 標準関数を有効利用できる。
③ メモリや入出力機器（デバイス）のアドレスを意識しないでプログラミングする。

1.2.2.組み込み用C言語プログラム

① リセット解除後、プログラムを起動する→C 言語では記述できないので、アセンブリ言語でリセット解除後に実行されるプログラム(スタートアッププログラムと言う。)を記述し、そこから呼びだす形をとる。終了は基本的にないので、無限ループの形で記述する。
② システムによって、周辺機器が異なるので、標準関数は出来るだけ使わず、自作する。
③ ROM、RAM の違いを認識し、正しいメモリ領域に、プログラム、データ領域を配置する。

これらの事柄に注意しながら、このあと出来るだけ多くの例題を使用して、OAKS16 のためのプログラミングを説明して行きます。

1.3. データ型

コンピュータを制御する場合、数値や文字などのデータを多く扱います。これらのデータは、CPU 内のレジスタやメモリに格納され、演算されたり、加工されたりします。C 言語で扱うことの出来るデータ（**定数**）は、「整数」「実数」「1 文字」「文字列」の 4 種類です。それ以外に、プログラム実行中に値が変わるデータとして**変数**が存在します。**変数**は、メモリにデータを入れるための領域確保を行ないますので、容量をコンパイラに認識させる為に**型宣言**を行ないます。ANSI 規格ではデータ型をターゲット CPU の BIT に依存させていますので、データ型の名前と表す数値が統一されていません。今回は、NC30 のデータ型を使って説明していきます。（このあと、M16C 以外の CPU を使って開発することがあった場合は、そのコンパイラの仕様を確認してください。）

1.3.1. NC30 で扱う定数

整数定数

整数定数は 10 進数、16 進数、8 進数の 3 種類の方法で記述できます。（2 進数記述は出来ません）データ表記は大文字、小文字を区別しません。

表 1.1

種類	表記方法	例
10 進数	通常の数学表記（何もつけない）	127、+127、-56
16 進数	数値の前に「0x（ゼロ・エックス）または「0X」を付ける	0x3b、0X3B
8 進数	数値の前に「0」を付ける	07、041

実数定数（浮動小数点定）

符号付きの実数を 10 進数で表したものを「浮動少数点定数」と言います。表記方法としては通常の小数点表記と、「e」または「E」を用いた指数表記が記述できます。

① 通常の小数点表記　　　　　　　　　例：175.5、-0.007

② 「e」または「E」を用いた指数表記　　例：1.755e2、-7.0E-3

1 文字定数

1 文字定数はシングルクォーテーション（‘）で囲みます。英数字のほかに、制御コードも 1 文字定数として扱うことが出来ます。

文字列定数

英数字や制御コードの並びをダブルクォーテーション（“）で囲むと文字列定数として扱うことが出来ます。文字列定数ではデータの最後にヌルコード‘￥0’が自動的に付けられ、文字列の終わりであることを表します。

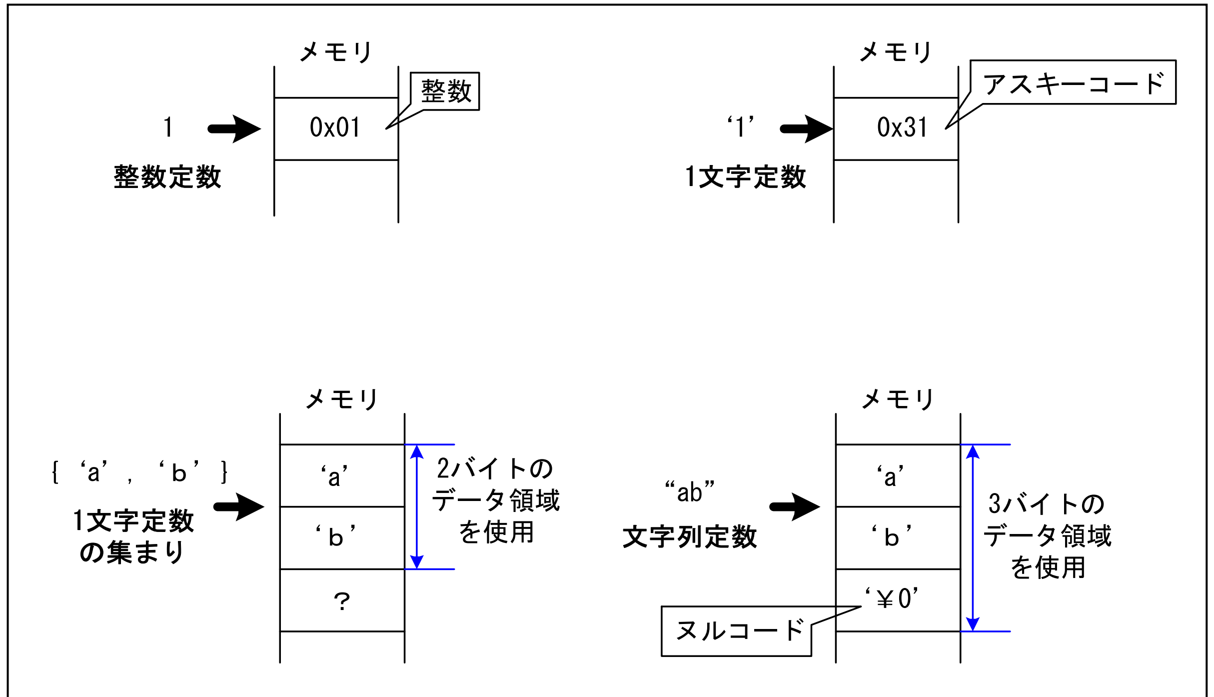


図 1.1

1.3.2. 変数

ここでは NC30 で扱える変数のデータ型と宣言方法について説明します。

NC30 の基本データ型

表 1.2

	データ型	ビット長	表現できる数値の範囲
整数	(unsigned) char	8 ビット	0～255
	signed char	8 ビット	-128～127
	unigned short (int)	16 ビット	0～65535
	(signed) short(int)	16 ビット	-32768～32767
	unsigned int	16 ビット	0～65535
	(signed) int	16 ビット	-32768～32767
	unsigned long (int)	32 ビット	0～65535
	(signed) long (int)	32 ビット	-32768～32767
実数	Float	32 ビット	有効桁数　9 桁
	Double	64 ビット	有効桁数　17 桁
	long double	64 ビット	有効桁数　17 桁

char は character のことを意味し、通常は文字データを格納するために使われます。
int は integer（整数）の意味です。

変数の宣言

変数の宣言は、**「データ型　変数名」**という書式で行ないます。

例：変数 c を char 型として宣言する。(8 ビットのデータ領域を a と言う名前で確保する)
char c ;
「データ型　変数名　＝　初期値；」と記述すると、宣言と同時にその変数に対して初期値を設定する事ができます。
例：char 型の変数 c に初期値として‘A’を設定する。
char c = ‘A’;
また、複数の変数名をカンマ（‘,’）で区切って列記すると、同じデータ型の変数を同時に宣言できます。

1.4. 演算子

NC30 で使用できる演算子を以下の表に示します。
詳しい内容は、プログラミングの中で説明していきます。

表 1.3

単項算術演算子	＋＋　－－　－
二項算術演算子	＋　－　＊　／　％
シフト演算子	＜＜　＞＞
ビット演算子	＆　｜　^　~
関数演算子	＞　＜　＞＝　＜＝　＝＝　！＝
論理演算子	＆＆　｜｜　！
代入演算子	＝　+＝　－＝　＊＝　／＝　％＝　＜＜＝　＞＞＝ ＆＝　｜＝　^＝
条件演算子	？：
sizeof 演算子	sizeof()
キャスト演算子	(型)
アドレス演算子	＆
ポインタ演算子	＊
コンマ演算子	,

1.5. 制御文

C 言語にはプログラムの流れを変えるための命令（制御命令）が存在します。これらを使うことにより、構造化された理解しやすいプログラムを記述することが出来ます。

構造化プログラミング

プログラムの流れをわかりやすくする為に、プログラムの流れを限定した手法だけを使って行なうというプログラミング手法が構造化プログラミングです。この手法は、「順次処理」「分岐処理」「繰り返し処理」の 3 つを使用して行ないます。

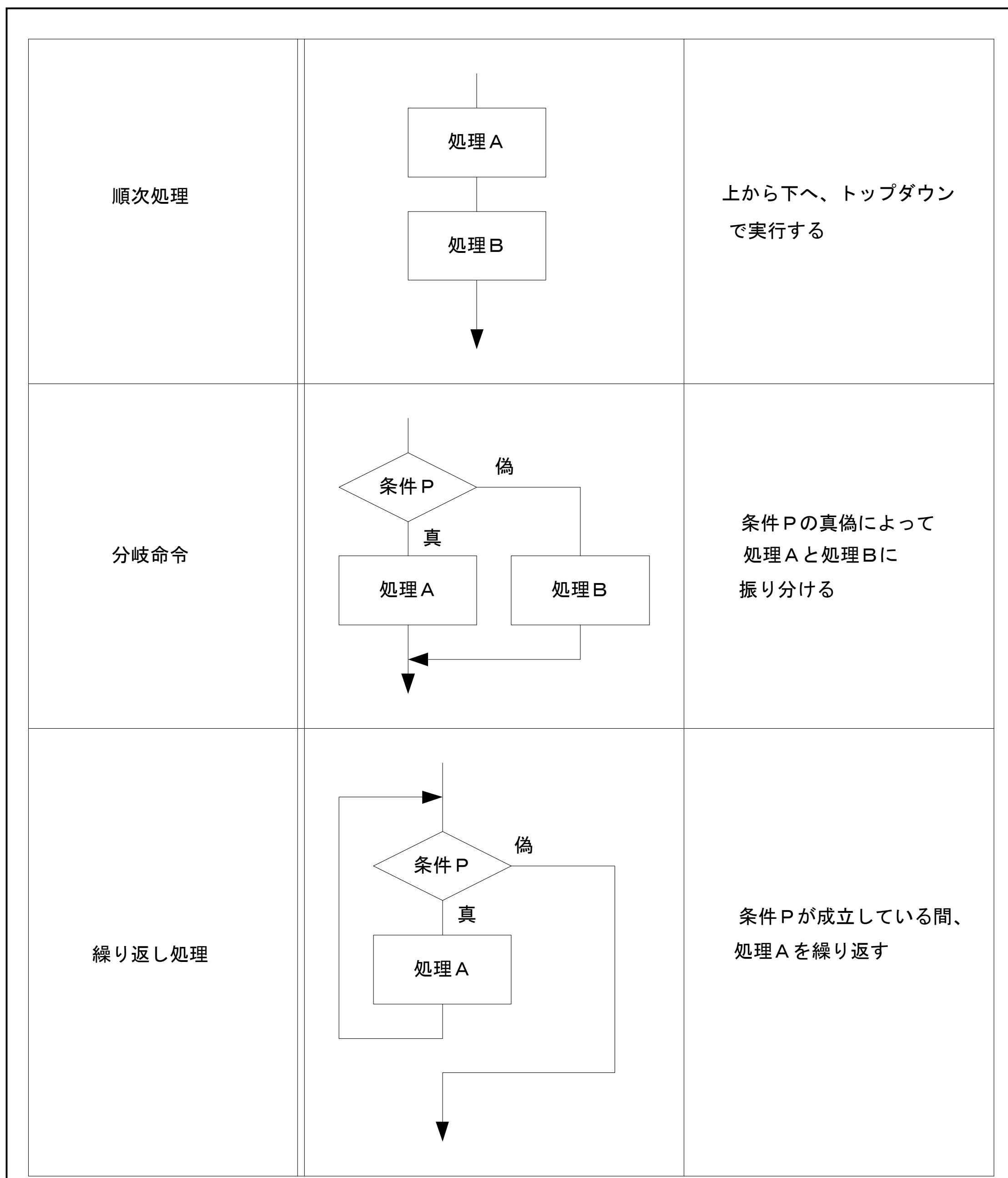


図 1.2

これらの制御文を実現する為の制御命令について説明します。

if 文

「もし～なら」という制御です。以下に代表的な記述と記述例を示します。

「if文」の構造

書き方	フローチャート	記述例
if（式）　文；	式 / 真 / 偽（=0） / 文	if(a=0)　a=a+1;
if（式）　文1； else　文2；	式 / 真 / 偽（=0） / 文1 / 文2	if(a!=0)　a=a+1; else　a=0;

if（式）　{
文1；
文2；
・
・
・
}
	式 / 真 / 偽（=0） / 文1 / 文2 / ・ / ・ / ・	if(a！=0){
		a=a+1;
		b=0;
		}

図 1.3

wlile 文

ある条件の間処理を繰り返します。

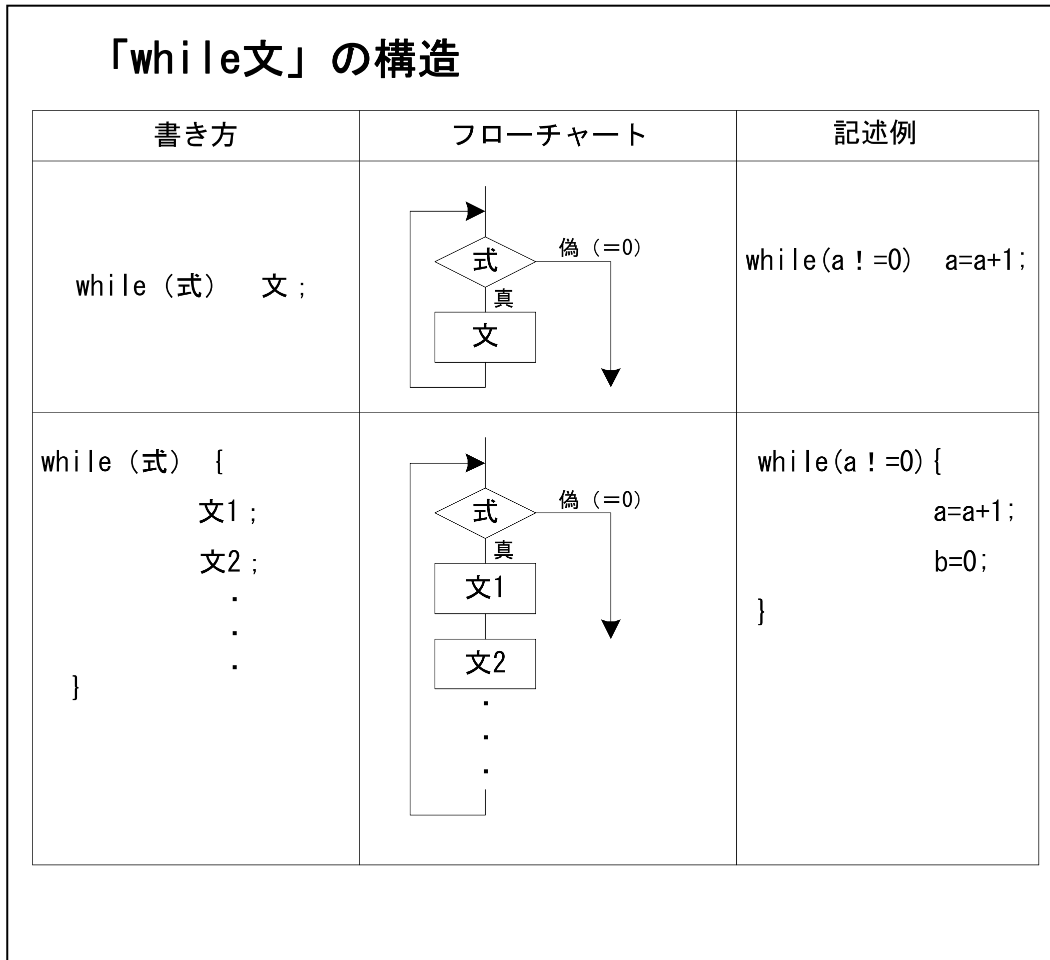
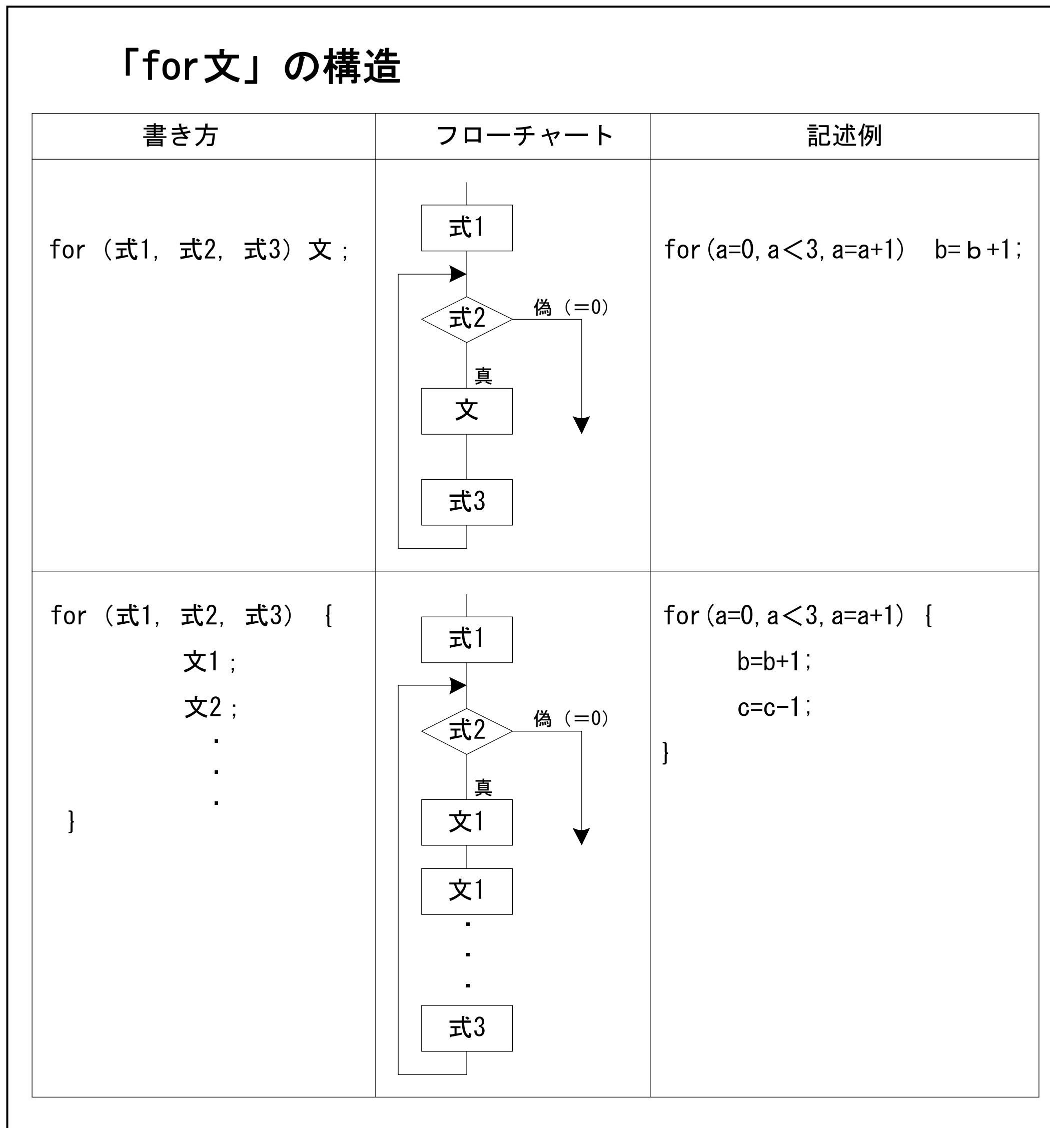


図 1.4

For 文

ある条件の間、処理を繰り返します。



「for文」の構造

書き方	フローチャート	記述例
for（式1，式2，式3）文；	式1 → 式2 → 真 → 文 → 式3（偽（=0）で抜ける）	for (a=0, a＜3, a=a+1)　b= b +1;

for（式1，式2，式3）{
文1；
文2；
・
・
・
}
	式1 → 式2 → 真 → 文1 → 文1 → ・・・ → 式3（偽（=0）で抜ける）	for (a=0, a＜3, a=a+1) {
		b=b+1;
		c=c-1;
		}

図 1.5

1.6. 関数

C 言語のプログラムは基本的に関数の集まりです。C 言語を理解する為には関数を知らなければなりません。

C 言語の関数の親分は main 関数です。全ての C 言語プログラムは main 関数から始まり、その中にいくつかの関数が書かれます。

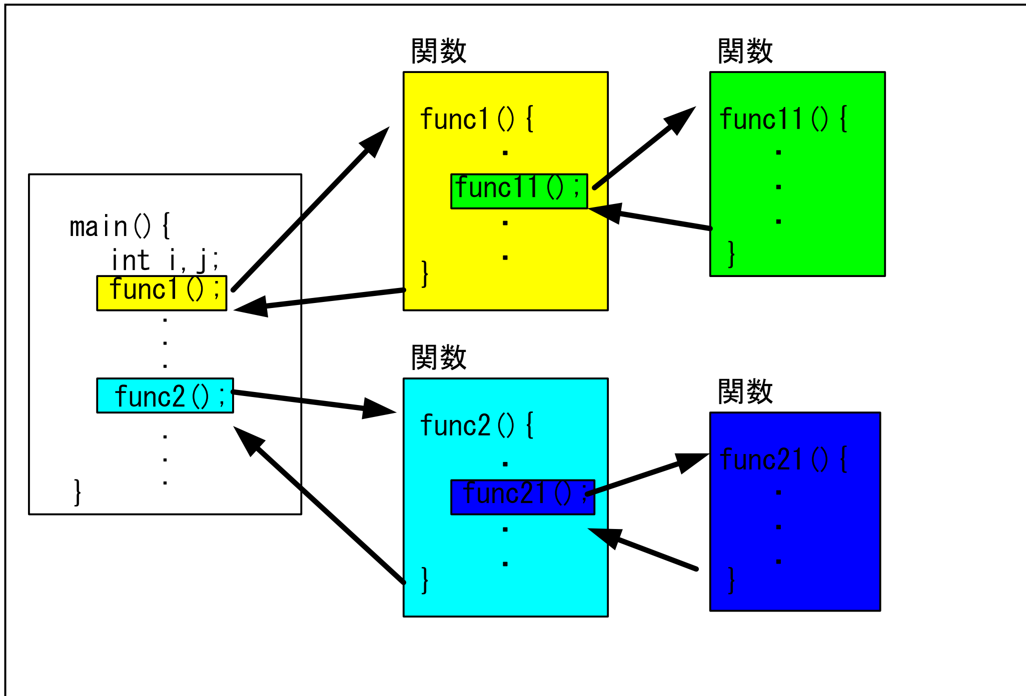


図 1.6

関数を作成するためには「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）

C 言語で関数を使用する前には、必ず関数の型宣言（プロトタイプ宣言）を行なわなければなりません。関数の型とは、関数の引数と戻り値のデータ型です。

```
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”という型を記述します。

関数の定義

関数本体では、引数を受け取る為の「仮引数」のデータ型と名称を定義します。また、戻り値は、「return 文」を使って返します。

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

関数の呼び出し

関数を呼び出すとき、その関数に対する引数を記述します。また呼び出した関数からの戻り値は代入演算子を用いて受け取ります。（代入演算子：1.3.4 演算子参照）

```
関数名（引数 1，・・・）；＋
```

戻り値があるとき

```
変数＝関数名（引数 1，・・・）；
```

関数の記述例

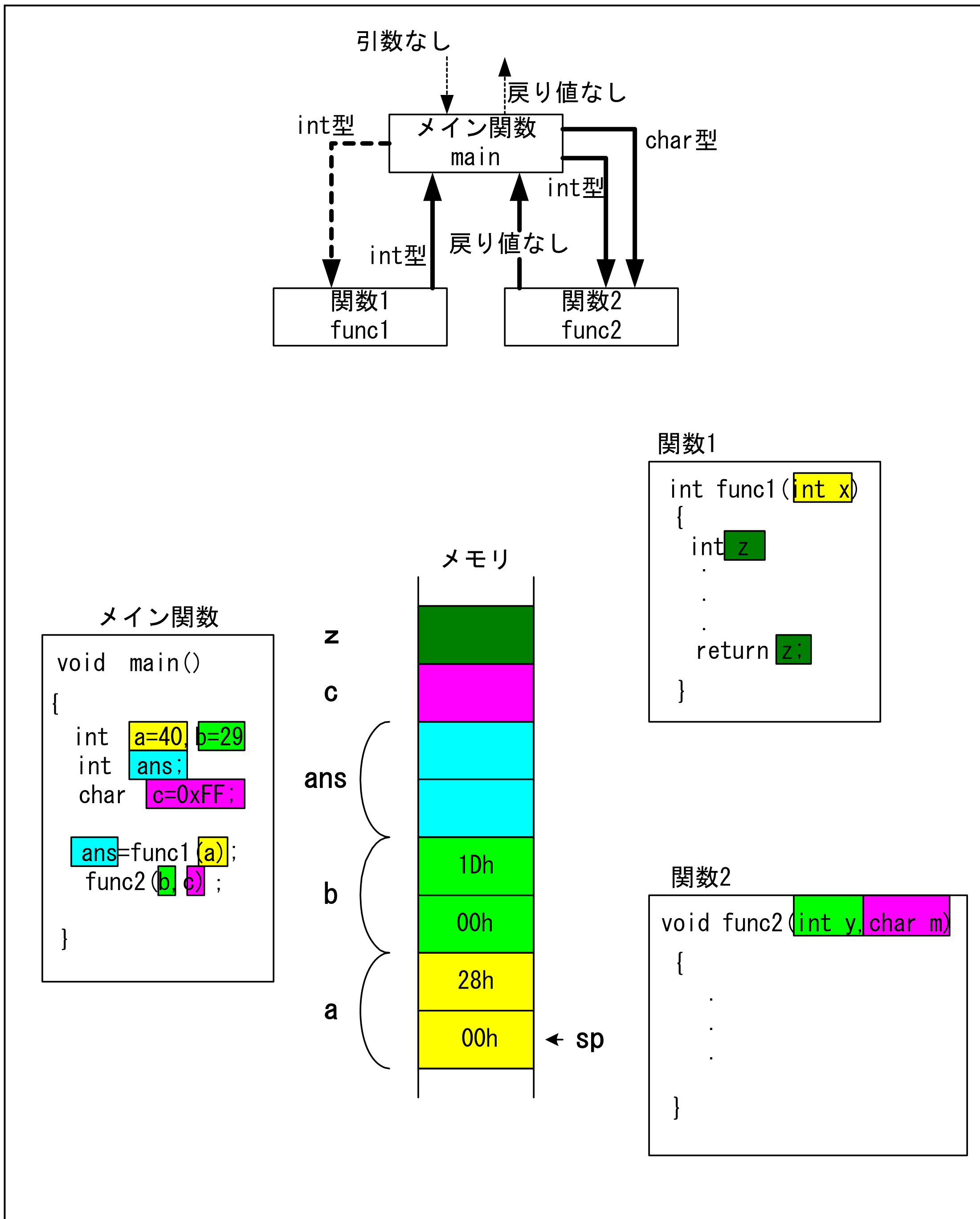


図 1.7

＊関数間でのデータの引渡しはスタックを使用して行なわれます。

1.7 配列

一つの変数名で複数のデータを格納することを配列といいます。この変数名を”配列名”と呼びます。この配列名の下なる各々のデータを“要素”と呼び、それを特定するのに 0 から始まる数を使います．個別の要素は、配列名[要素番号]で表します。
配列には 1 次元配列、2 次元配列などがあります。

1.7.1. 1 次元配列

1 次元配列は、配列の基本となるものです。

配列の宣言

データ型　配列名[要素数]；

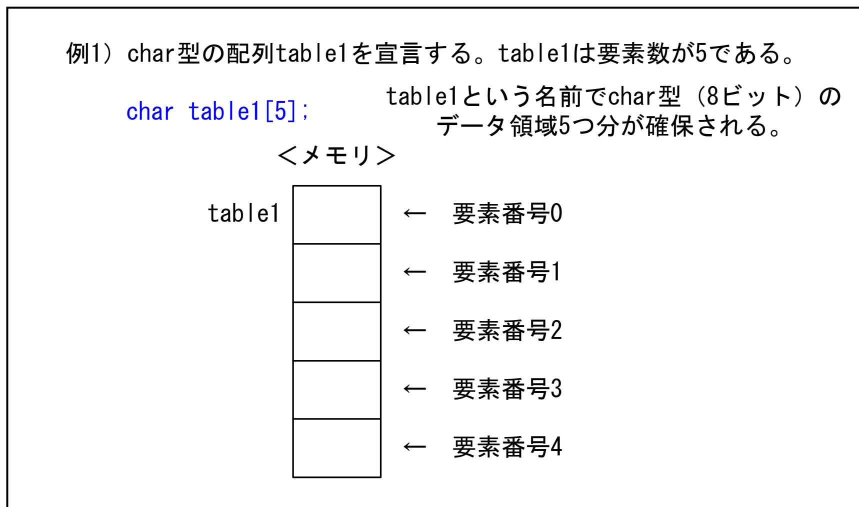


図 1.8

配列を宣言して初期化する

データ型　配列名[要素数]＝　{要素 0，要素 1，要素 2，・・・}；

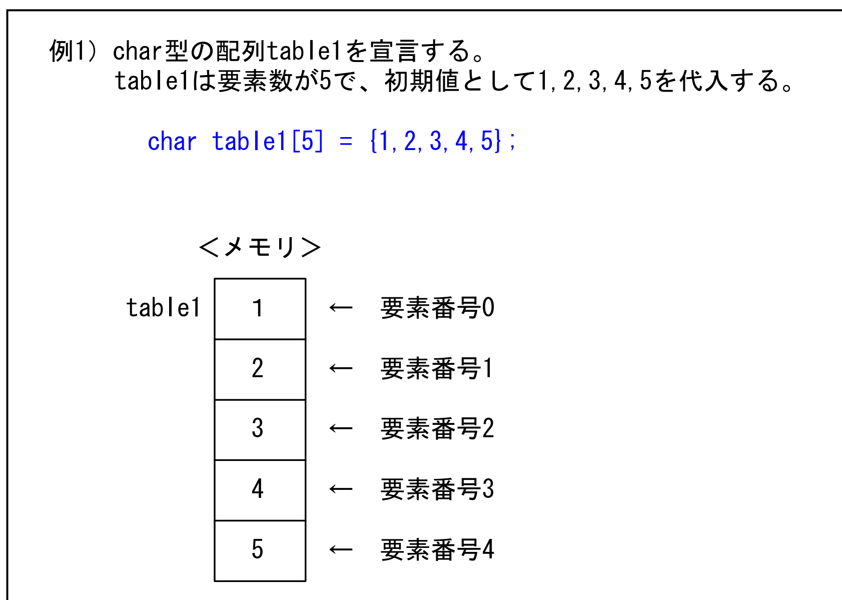


図 1.9

1.7.2. 2次元配列

配列の要素は変数や定数だけでなく配列も要素として扱うことができます。２次元配列は１次元配列を要素にもった配列です。

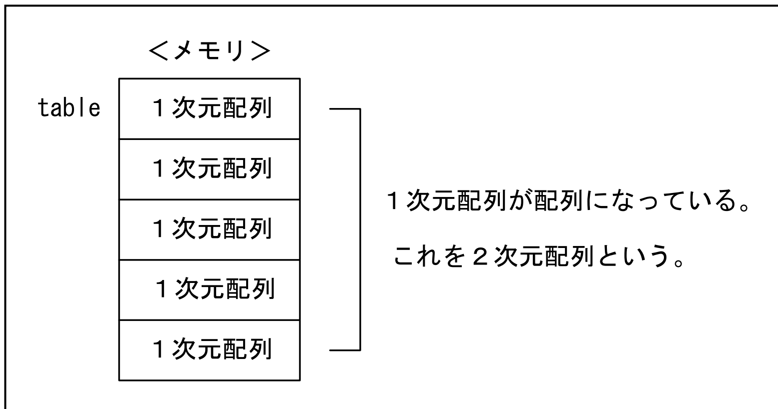


図 1.10

２次元配列は、行と列で表現されます。

配列の宣言

データ型　配列名〔行数〕〔列数〕;

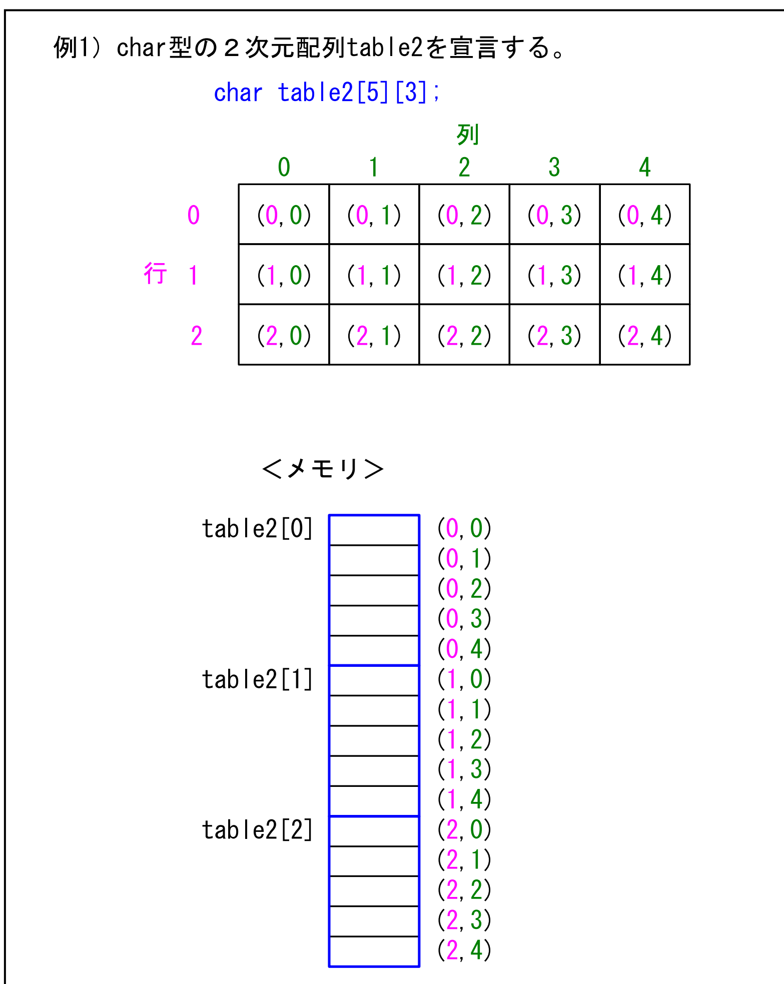


図 1.11

配列を宣言して初期化する

データ型　配列名[行数][列数]＝　{要素(0,0),要素(0,1),要素(0,2),・・・};

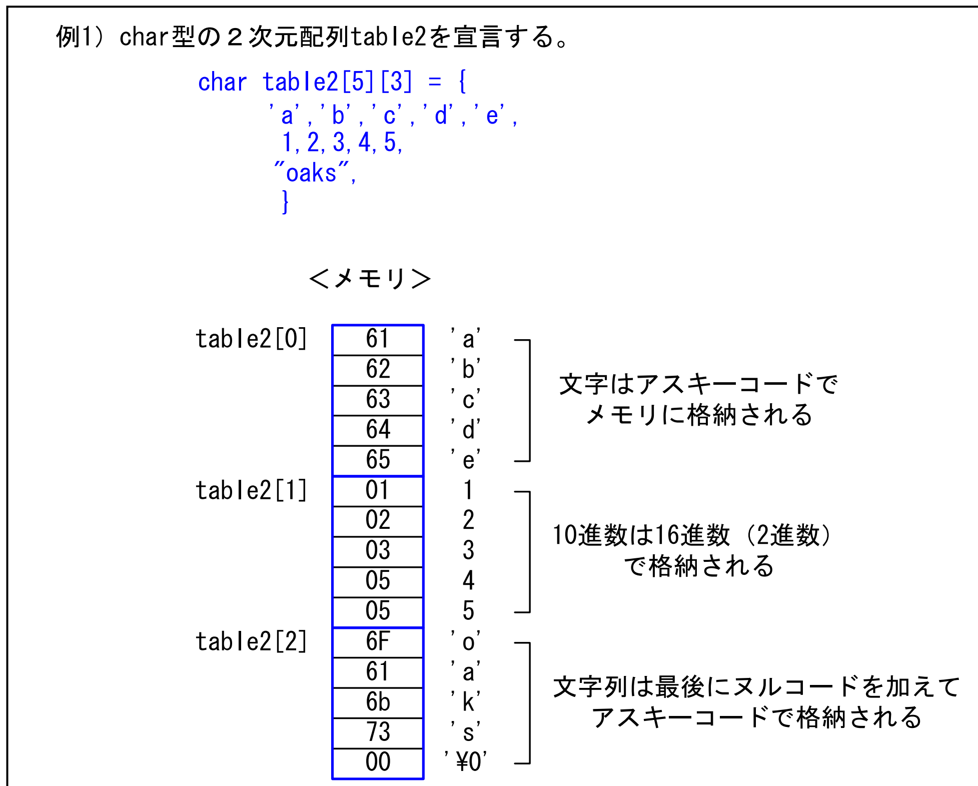


図 1.12

文字列の配列は、LCD 表示プログラムなどで使用することがあります。

1.8. 構造体

構造体はＣ言語のデータ型を自由に組み合わせて一つのまとまったかたまりとして管理するものです。この後説明する M16C の SFR を定義するヘッダファイル記述に使われているので、ここで説明しておきます。

構造体の宣言には、型枠（テンプレート）を宣言する機能と、型枠で示されたサイズのメモリ領域を実際に確保する機能があります。この二つの機能を一緒に宣言することもできます。

型枠の宣言

struct　構造体タグ　｛構造体リスト｝；

型を指定して記憶領域を確保する

struct　構造体タグ　構造体変数リスト；

型枠と記憶領域を同時に宣言する

struct　構造体タグ｛構造体リスト｝構造体変数リスト；

構造体タグは新しく定義された構造体型の名前（型枠名）です。構造体変数リストは単純変数や配列、ポインタをコンマ（,）で区切って並べます。

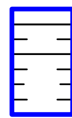
プログラム中での参照の仕方

“.”で構造体変数名とメンバ名をつなぎます。

＜構造体タグ宣言＞

struct　構造体タグ　{構造体リスト}

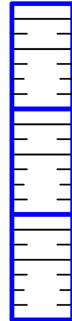
例）
```
struct  rei {
        char   c;
        int  i;
        long   l;
        } ;
```



＜構造体の型を指定して記憶領域を確保する＞

struct　構造体タグ　構造体変数リスト

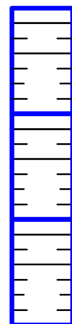
例）
```
struct rei data,table[2]
```



＜構造体型枠と記憶領域を同時に宣言する＞

struct　構造体タグ　{構造体リスト} 構造体変数リスト;

例）
```
struct  rei {
        char   c;
        int  i;
        long   l;
        } data,table[2];
```



＜構造体メンバを参照する。＞

構造体変数名. メンバ名

例）
```
data.char = 3;
table[2].long = 0xffffffff;
```

図 1.13

1.8.1. ビットフィールドの宣言

Ｃ言語では、構造体のメンバにビット単位で区別するビットフィールドが定義できます。
例として、全体が８ビットで上位から１ビット、１ビット、６ビットと割り当ててみます。

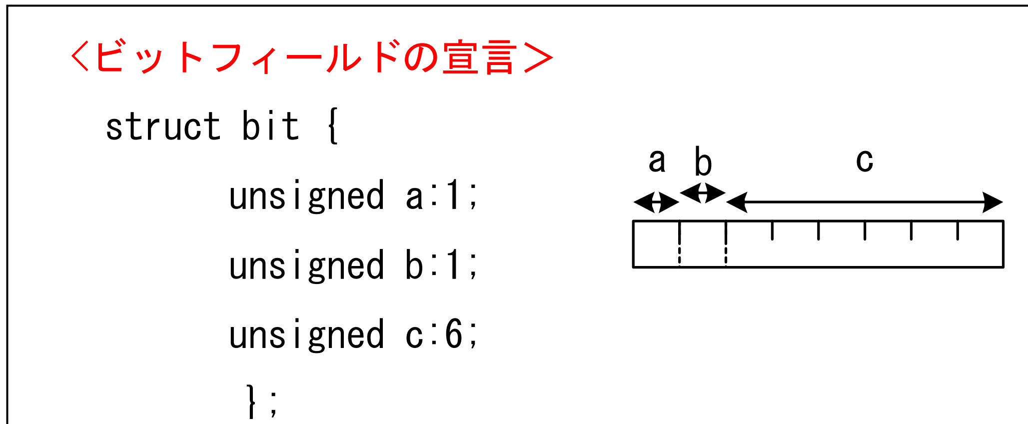


図 1.14

これによって、Ｃ言語では存在しないビット単位の変数を扱うことが出来るようになります。
（これは、後述の SFR の定義で使われます。）

1.9. 共用体

共用体とは、同一メモリ領域を別の複数の変数名を重ねて割り付けて操作できるようにするものです。書き方や参照の仕方は構造体とほぼ同じですが、struct の代わりに union を使います。

共用体の宣言には、型枠（テンプレート）を宣言する機能と、型枠で示されたサイズのメモリ領域を実際に確保する機能があります。この二つの機能を一緒に宣言することもできます。

型枠の宣言

union　共用体タグ　 {共用体リスト}；

型を指定して記憶領域を確保する

union　共用体タグ　共用体変数リスト；

型枠と記憶領域を同時に宣言する

union　共用体タグ {共用体リスト} 共用体変数リスト；

共用体タグは新しく定義された共用体型の名前（型枠名）です。共用体変数リストは単純変数や配列、ポインタをコンマ（,）で区切って並べます。

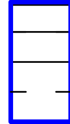
プログラム中での参照の仕方

“.”で共用体変数名とメンバ名をつなぎます。

＜共用体タグ宣言＞

union　共用体タグ　{共用体リスト}

例）
```
union　rei {
    char　 c ;
    int　 i;
    long　 l ;
    } ;
```

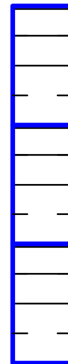


rei
このような型枠が作られます。
（まだメモリにはとられません。）

＜共用体の型を指定して記憶領域を確保する＞

union　共用体タグ　共用体変数リスト

例）
```
union rei data,table[2]
```

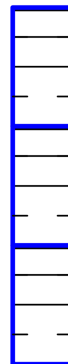


実際にメモリ上に、
共用体枠が取られます。

＜共用体型枠と記憶領域を同時に宣言する＞

union　共用体タグ　{共用体リスト} 共用体変数リスト;

例）
```
union　rei {
    char　 c ;
    int　 i;
    long　 l ;
    } data,table[2];
```



実際にメモリ上に、
共用体枠が取られます。

＜共用体メンバを参照する。＞

共用体変数名．メンバ名

例）
```
data.char = 3;
 table[2].long = 0xffffffff;
```

図 1.15

2.OAKS16 でC言語を使って開発する前に

「OAKS16 基礎テキスト」でも説明したように、組み込み用のプログラム開発に置いては、全ての記述を C 言語だけで行なうことは出来ません。パソコン上で動かすソフトが OS（Windows など）から呼び出されるように、電源投入後 C 言語 CPU が動き出したときから、C 言語の main 関数を呼び出すまでの作業をアセンブリ言語のプログラムで記述する必要があります。このプログラムを OAKS16 では「スタートアッププログラム」と呼んでいます。

スタートアッププログラムの内容

① スタックポインタの設定
② マイコンの初期設定
③ メモリ領域の初期化
④ 割り込みテーブルレジスタ“INTB”の設定
⑤ main 関数の呼び出し
⑥ 割り込みベクタテーブルの設定

①、③について：
メモリマップを確認します
OAKS16 のメモリマップでは、デバッグ時とフラッシュ ROM に書き込んだときのメモリマップが異なります。しかし、デバッグできない領域に開発することは無理があるので、デバッグ用のメモリマップに合わせて開発を行ないます。(モニタ領域に使われていた部分は使わないことにします。)
ROM：400h から 24BFh
RAM：E0000h から FBDFF h

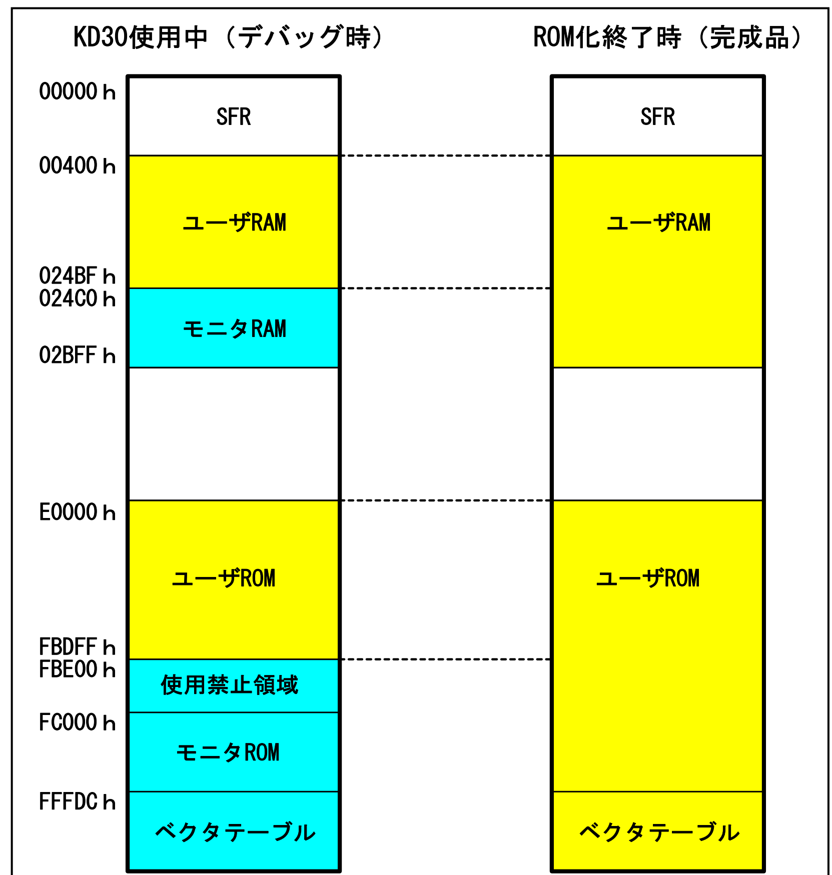


図 2.1

②について：
CPU モード：ワンチップモード
クロック：分周なし
（これらは、モニタプログラムの設定にあわせています。

異なる設定をしたい場合は、OAKS16 マニュアルとデータブックを読んで、ご自身の責任で行なってください。)

③について：

メモリの初期化は、この後説明していく C 言語のコンパイラのための設定です。C 言語のプログラムでは、データを型宣言という形で、どの領域を使うかを決めていきます。そのため、スタートアッププログラムでは、コンパイラの要求する名前でメモリ領域をシステムに合わせた容量で確保していきます。

NC30 のセクション

OAKS16 では NC30 というコンパイラを使用します。ここであえてこの件を確認するのは、コンパイラによってはメモリ領域の取り方、呼び方が異なる場合もあるからです。

NC30 のデータ領域についてここで説明しておきます。

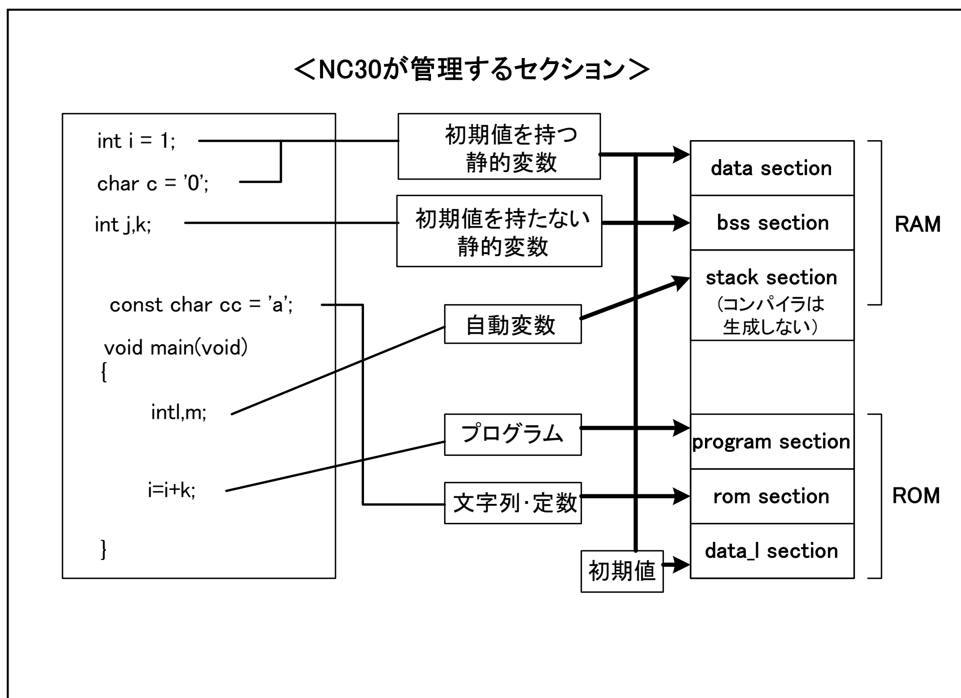


図 2.2

C 言語のデータ型については後で説明しますので、ここではデータ型の宣言によって、決められたメモリ領域に配置されると言うことだけ意識して置いてください。

セクションの属性：NC30 が生成するセクションは、初期値の有無、配置される療育、データサイズなどの属性によってさらに細かく分類されます。

表 2.1

属性	内容	対象セクションベース名
I	データの初期値を保持するセクション	data
N/F/S	N－near 属性（絶対番地 0～0FFFF の 64K バイトの領域） F－far 属性（0～FFFFF 番地の 1M バイト全メモリ領） S－SBDATA 属性 （SB 相対アドレッシングを使用できる領域）	data、bss、rom
E/O	E－データサイズが偶数 O－データサイズが奇数	data、bss、rom

ここでの“near”や“far”はこのあとのスタートアッププログラムの説明で出ますので、なにを意味するかを知っておいてください。

2.1.スタートアッププログラム

「OAKS16 基礎テキスト」の例題では、スタートアッププログラムは、最低限の設定だけをするものを紹介しました。しかし、三菱では、それぞれの CPU に対する標準のスタートアッププログラムを提供しています。TM を使ってプログラムを開発する場合でも、プロジェクト作成時に、標準のスタートアッププログラムを選択できるようになっています。

そこで、ここでは、M16C の標準スタートアッププログラムの内容説明と、OAKS16 に対応するように変更する方法説明をしていきます。

2.1.1.NC30 のサンプルスタートアッププログラムの構成

NC30 のサンプルスタートアッププログラムは、“ncrt0.a30”と“sect30.inc”の 2 つのファイルで構成されています。

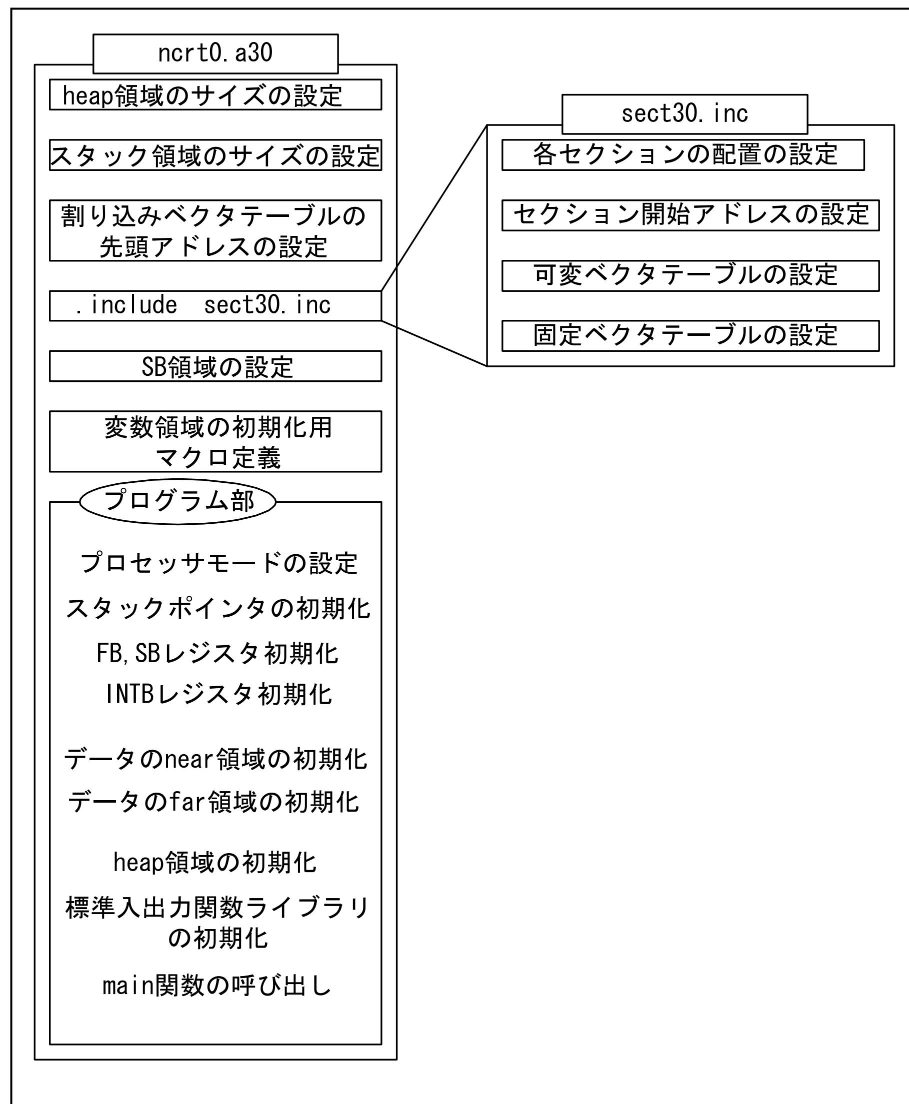


図 2.3

2.1.2.スタートアッププログラムの作成

サンプルスタートアッププログラムは、作成するプログラムに合わせて変更する必要があります。

サンプルスタートアッププログラムの変更点

ncrt.a30（スタートアッププログラム本体）

① スタック（SP、ISP）セクションのサイズ設定
② 可変割り込みベクタの先頭アドレス設定
③ プロセッサモードなどの各種レジスタの設定

sect30.inc（セクション定義ファイル）

① 各セクションの配置と先頭アドレスの設定
② 可変／固定割り込みベクタの設定
③ スペシャルページベクタの設定

では、これからサンプルスタートアッププログラムを変更していきます。
はじめに、sect30.inc の説明、続いて ncrt.a30 の説明を行ないます。

sect30.inc（セクション定義ファイル）

```
;*******************************************************************************
;
;     C Compiler for M16C/60
;        Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;        AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;        All Rights Reserved.
;
;     Written by T.Aoyama
;
;        sect30.inc      : section definition
;        This program is applicable when using the basic I/O library
;
;        $Id: sect30.inc,v 1.9 2000/06/20 09:07:11 simomura Exp $
;
;*******************************************************************************
;---------------------------------------------------------------
;
;     Arrangement of section
;
;---------------------------------------------------------------
; Near RAM data area
;---------------------------------------------------------------
; SBDATA area
      .section   data_SE,DATA
      .org  400H
data_SE_top:

      .section   bss_SE,DATA,ALIGN
bss_SE_top:

      .section   data_SO,DATA
data_SO_top:

      .section   bss_SO,DATA
bss_SO_top:

; near RAM area
      .section   data_NE,DATA,ALIGN
data_NE_top:

      .section   bss_NE,DATA,ALIGN
bss_NE_top:

      .section   data_NO,DATA
data_NO_top:

      .section   bss_NO,DATA
bss_NO_top:
;---------------------------------------------------------------
; Stack area
;---------------------------------------------------------------
      .section   stack,DATA
      .blkb STACKSIZE
stack_top:

      .blkb ISTACKSIZE
istack_top:

;---------------------------------------------------------------
;     heap section
;---------------------------------------------------------------
;     .section   heap,DATA
;heap_top:
;     .blkb HEAPSIZE
```

OAKS16 ではそのままでＯＫ

Near　RAM 領域の先頭アドレス設定

各データ領域の先頭にラベルを設定する

Ｅ属性セクションはアライメント指定し、偶数番地から配置する

スタック領域にユーザスタックと割り込みスタックの領域を確保する。

“STACKSIZE”と“ISTACKSIZE”は ncrt0.a30 の先頭で、“equ” 命令で設定する（今回は、300ｈ）

Heap セクションはメモリ管理関数を使うときに使用する。このテキストでは使わないのでコメントアウトする。(使用方法は、NC30 のマニュアルを参照のこと)

注）コメントアウト：
行頭に“；”を記述することによって
コメント文とし、あとで修正が可能なよう
にしておく

```
;-------------------------------------------------------
; Near ROM data area
;-------------------------------------------------------
;       .section   rom_NE,ROMDATA,ALIGN
;rom_NE_top:
;
;       .section   rom_NO,ROMDATA
;rom_NO_top:
;
;---------------------------------------------------------------
; Far RAM data area
;-------------------------------------------------------
;       .section   data_FE,DATA
;       .org       10000H
;data_FE_top:
;
;       .section   bss_FE,DATA,ALIGN
;bss_FE_top:
;
;       .section   data_FO,DATA
;data_FO_top:
;
;       .section   bss_FO,DATA
;bss_FO_top:
;
;
;-------------------------------------------------------
; Far ROM data area
;-------------------------------------------------------
        .section   rom_FE,ROMDATA
        .org       0F0000H
rom_FE_top:

        .section   rom_FO,ROMDATA
rom_FO_top:

;-------------------------------------------------------
; Initial data of 'data' section
;-------------------------------------------------------
        .section   data_SEI,ROMDATA
data_SEI_top:

        .section   data_SOI,ROMDATA
data_SOI_top:

        .section   data_NEI,ROMDATA
data_NEI_top:

        .section   data_NOI,ROMDATA
data_NOI_top:

;       .section   data_FEI,ROMDATA
;data_FEI_top:
;
;       .section   data_FOI,ROMDATA
;data_FOI_top:
;
```

OAKS16 ではこの領域に ROM を配置していないのでコメントアウトする

Near 領域（0ffffh 番地まで）に ROM が存在する場合のアドレス設定

Near 領域以外にに RAM が存在する場合のアドレス設定

OAKS16 ではこの領域に RAM を配置していないのでコメントアウトする

ROM 領域のアドレス設定
プログラムエリアは別に取るので、この場合は固定データの設定に使う領域を意味する

システムに応じて変える。今回はこのままで OK

初期値を持つデータのセクション設定
例）SEI：
SB データ･データサイズが偶数･データを保持するセクション

属性	内容
I	データの初期値を保持するセクション
N／F／S	N－near属性（絶対番地0～0FFFFの64Kバイトの領域） F－far属性（0～FFFFF番地の1Mバイト全メモリ領域） S－SBDATA属性（SB相対アドレッシングを使用できる領域）
E／O	E－データサイズが偶数 O－データサイズが奇数

<セクションの属性>

```
;------------------------------------------------------------------
; Switch Table Section
;------------------------------------------------------------------
;    .section        switch_table,ROMDATA
;switch_table_top:
;
;
;------------------------------------------------------------------
; code area
;------------------------------------------------------------------

     .section   program

     .section   interrupt
     ;.org ;must be set internal ROM area
     .section   program_S

;------------------------------------------------------------------
; variable vector section
;------------------------------------------------------------------
     .section   vector          ; variable vector table
     .org  VECTOR_ADR
;
;.if M62TYPE==1
;    .lword     dummy_int       ; BRK(vector 0)
;    .org  (VECTOR_ADR+16)
;    .lword     dummy_int       ; int3(for user)(vector 4)
;    .lword     dummy_int       ; timerB5(for user)(vector 5)
;    .lword     dummy_int       ; timerB4(for user)(vector 6)
;    .lword     dummy_int       ; timerB3(for user)(vector 7)
;    .lword     dummy_int       ; si/o4 /int5(for user)(vector 8)
;    .lword     dummy_int       ; si/o3 /int4(for user)(vector 9)
;    .lword     dummy_int       ; Bus collision detection(for user)(v10)
;    .lword     dummy_int       ; DMA0(for user)(vector 11)
;    .lword     dummy_int       ; DMA1(for user)(vector 12)
;    .lword     dummy_int       ; Key input interrupt(for user)(vect 14)
;    .lword     dummy_int       ; A-D(for user)(vector 14)
;    .lword     dummy_int       ; uart2 transmit(for user)(vector 15)
;    .lword     dummy_int       ; uart2 receive(for user)(vector 16)
;    .lword     dummy_int       ; uart0 transmit(for user)(vector 17)
;    .lword     dummy_int       ; uart0 receive(for user)(vector 18)
;    .lword     dummy_int       ; uart1 transmit(for user)(vector 19)
;    .lword     dummy_int       ; uart1 receive(for user)(vector 20)
;    .lword     dummy_int       ; timer A0(for user)(vector 21)
;    .lword     dummy_int       ; timer A1(for user)(vector 22)
;    .lword     dummy_int       ; timer A2(for user)(vector 23)
;    .lword     dummy_int       ; timer A3(for user)(vector 24)
;    .lword     dummy_int       ; timer A4(for user)(vector 25)
;    .lword     dummy_int       ; timer B0(for user)(vector 26)
;    .lword     dummy_int       ; timer B1(for user)(vector 27)
;    .lword     dummy_int       ; timer B2(for user)(vector 28)
;    .lword     dummy_int       ; int0 (for user)(vector 29)
;    .lword     dummy_int       ; int1 (for user)(vector 30)
;    .lword     dummy_int       ; int2 (for user)(vector 31)
```

コンパイルオプション“-fswitch_other_section(-fSOS)”を指定しない場合はコメントアウトする

このテキストでは使用しないのでコメントアウト

Main プログラム配置セクション

スタートアッププログラム配置セクション

可変ベクタアドレスの先頭アドレス

VECTOR_ADR は ncrt0.a30 の先頭で設定

ベクターテーブルが CPU のタイプで選択できるようになっているが、OAKS16 では CPU が決まっているので必要ないところは**コメントアウト**する(削除しても可)

コメントアウト

```
;.else
        .lword      dummy_int       ; vector 0 (BRK)
        .org  (VECTOR_ADR +44)
        .lword      dummy_int       ; DMA0 (for user)
        .lword      dummy_int       ; DMA1 2 (for user)
        .lword      dummy_int       ; input key (for user)
        .lword      dummy_int       ; AD Convert (for user)
        .org  (VECTOR_ADR +68)
        .lword      dummy_int       ; uart0 trance (for user)
        .lword      dummy_int       ; uart0 receive (for user)
        .lword      0fcb6bh             ; uart1 trance (for user)
        .lword      0fcb6bh             ; uart1 rec
        .lword      dummy_int       ; TIMER A0 (for user)
        .lword      dummy_int       ; TIMER A1 (for user)
        .lword      dummy_int       ; TIMER A2 (for user)
        .lword      dummy_int       ; TIMER A3 (for user)
        .lword      dummy_int       ; TIMER A4 (for user) (vector 25)
        .lword      dummy_int       ; TIMER B0 (for user) (vector 26)
        .lword      dummy_int       ; TIMER B1 (for user) (vector 27)
        .lword      dummy_int       ; TIMER B2 (for user) (vector 28)
        .lword      dummy_int       ; INT0 (for user) (vector 29)
        .lword      dummy_int       ; INT1 (for user) (vector 30)
        .lword      dummy_int       ; INT2 (for user) (vector 31)
;.endif
        .lword      dummy_int       ; vector 32 (for user or MR30)
        .lword      dummy_int       ; vector 33 (for user or MR30)
        .lword      dummy_int       ; vector 34 (for user or MR30)
        .lword      dummy_int       ; vector 35 (for user or MR30)
        .lword      dummy_int       ; vector 36 (for user or MR30)
        .lword      dummy_int       ; vector 37 (for user or MR30)
        .lword      dummy_int       ; vector 38 (for user or MR30)
        .lword      dummy_int       ; vector 39 (for user or MR30)
        .lword      dummy_int       ; vector 40 (for user or MR30)
        .lword      dummy_int       ; vector 41 (for user or MR30)
        .lword      dummy_int       ; vector 42 (for user or MR30)
        .lword      dummy_int       ; vector 43 (for user or MR30)
        .lword    dummy_int         ; vector 44 (for user or MR30)
        .lword      dummy_int       ; vector 45 (for user or MR30)
        .lword      dummy_int       ; vector 46 (for user or MR30)
        .lword      dummy_int       ; vector 47 (for user or MR30)
```

OAKS16 ではモニタが UART1 を使用しているので、必ずこのアドレスを設定する。

注） 可変割り込みのベクタ設定方法は、割り込みプログラムの項目で説明します。

```
;===============================================================
; fixed vector section
;---------------------------------------------------------------
     .section   fvector               ; fixed vector table
;===============================================================
; special page defination
;---------------------------------------------------------------
;    macro is defined in ncrt0.a30
;    Format: SPECIAL number
;
;---------------------------------------------------------------
;    SPECIAL 255
;    SPECIAL 254
;    SPECIAL 253
;        :
;        :
;    (omitted)
;        :
;        :
;    SPECIAL 24
;    SPECIAL 23
;    SPECIAL 22
;    SPECIAL 21
;    SPECIAL 20
;    SPECIAL 19
;    SPECIAL 18
;
;=====================================================================
; fixed vector section
;---------------------------------------------------------------
     .org 0fffdch
UDI:
     .lword     dummy_int
OVER_FLOW:
     .lword     dummy_int
BRKI:
     .lword     dummy_int
ADDRESS_MATCH:
     .lword     dummy_int
SINGLE_STEP:
     .lword     dummy_int
WDT:
     .lword     dummy_int
DBC:
     .lword     dummy_int
NMI:
     .lword     dummy_int
RESET:
     .lword     start
;
;*****************************************************************************
;
;    C Compiler for M16C/60
;       Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;       All Rights Reserved.
;
```

スペシャルページ関数呼び出しのベクタアドレス設定
飛び先番地の設定は ncrt0.a30 内で定義される

このテキストでは使用しないので**コメントアウト**

固定割り込みベクタの先頭アドレス設定

固定割り込みベクタの設定
「OAKS16 基礎テキスト」の例題では別ファイルに用意しましたが、三菱の標準スタートアッププログラムにはこのように記述されています

リセットベクタ

ncrt.a30（スタートアッププログラム本体）

```
;************************************************************************** ;
;     C COMPILER for M16C/60
;     Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;     AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;     All Rights Reserved.
;
;        ncrt0.a30 : NC30 startup program
;
;     This program is applicable when using the basic I/O librar
;
;        $Id: ncrt0.a30,v 1.13 2000/06/22 13:17:04 simomura Exp :
;
;**************************************************************************

;          .glb __BankSelect
;__BankSelect   .equ 0BH
;---------------------------------------------------------------------------
; HEEP SIZE definition
;---------------------------------------------------------------------------
;HEAPSIZE  .equ 300h

;---------------------------------------------------------------------------
; STACK SIZE definition
;---------------------------------------------------------------------------
STACKSIZE  .equ 300h

;---------------------------------------------------------------------------
; INTERRUPT STACK SIZE definition
;---------------------------------------------------------------------------
ISTACKSIZE .equ 300h

;---------------------------------------------------------------------------
; INTERRUPT VECTOR ADDRESS  definition
;---------------------------------------------------------------------------
VECTOR_ADR       .equ 0fa000h

;---------------------------------------------------------------------------
; special page definition
;---------------------------------------------------------------------------
;        macro define for special page
;
;Format:
;     SPECIAL    number
;

SPECIAL    .macro     NUM
     .org  0FFFFEH-(NUM*2)
     .glb  __SPECIAL_@NUM
     .word __SPECIAL_@NUM  & 0FFFFH
.endm
```

バンク切り替えの為の設定:このテキストでは使用しないのでコメントアウト

このテキストでは使用しないのでコメントアウト

Heap 領域のサイズ設定

ユーザスタック領域のサイズを設定する

スタックサイズの算出方法は後で説明するが、このテキストでは領域が十分あるので 300h のままで OK

割り込みスタック領域のサイズを設定する

可変割り込みベクタテーブルの先頭アドレスを設定

OAKS16 ではユーザが使用できるのは fbdffh までなので、余裕をとってこのアドレスを設定する。

スペシャルページマクロの定義

```
;-----------------------------------------------------------------------
; Section allocation
;-----------------------------------------------------------------------
     .list OFF
     .include sect30.inc
     .list ON

;-----------------------------------------------------------------------
; SBDATA area definition
;-----------------------------------------------------------------------
     .glb __SB__
__SB__     .equ data_SE_top

;=====================================================================
; Initialize Macro declaration
;-----------------------------------------------------------------------
N_BZERO    .macro    TOP_ ,SECT_
     mov.b#00H, R0L
     mov.w#(TOP_ & 0FFFFH), A1
     mov.w#sizeof SECT_ , R3
     sstr.b
     .endm

N_BCOPY .macro  FROM_,TO_,SECT_
     mov.w#(FROM_ & 0FFFFH),A0
     mov.b#(FROM_ >>16),R1H
     mov.w#TO_ ,A1
     mov.w#sizeof SECT_ , R3
     smovf.b
     .endm

BZERO.macro    TOP_,SECT_
     push.w    #sizeof SECT_ >> 16
     push.w    #sizeof SECT_  & 0ffffh
     pusha TOP_ >>16
     pusha TOP_ & 0ffffh
     .stk 8
     .glb _bzero
     .call_bzero,G
     jsr.a_bzero
     .endm

BCOPY.macro    FROM_ ,TO_ ,SECT_
     push.w    #sizeof SECT_  >> 16
     push.w    #sizeof SECT_  & 0ffffh
     pusha TO_ >>16
     pusha TO_  & 0ffffh
     pusha FROM_ >>16
     pusha FROM_  & 0ffffh
     .stk 12
     .glb _bcopy
     .call   _bcopy,G
     jsr.a_bcopy
     .endm
```

スタティックベースレジスタ（SB）の値をコンパイラに対して定義

RAM 領域初期化用マクロ定義

Near 領域の RAM クリアの為のマクロ

Near 領域の初期値コピー（ROM から RAM へ）の為のマクロ

Far 領域の RAM クリアの為のマクロ

Far 領域の初期値コピーの為のマクロ

```
;=====================================================================
; Interrupt section start
;---------------------------------------------------------------------
     .insf start,S,0
     .glb  start
     .section   interrupt
start:
;---------------------------------------------------------------------
; after reset,this program will start
;---------------------------------------------------------------------

     ldc   #istack_top,    isp   ;set istack pointer

     mov.b #02h,0ah

;    bset  1,0ah

     mov.b #00h,04h            ;set processer mode

;    bclr  1,0ah

     mov.b #00h,0ah


     ldc   #0080h,    flg

     ldc   #stack_top,     sp    ;set stack pointer

     ldc   #data_SE_top,   sb    ;set sb register

     ldintb     #VECTOR_ADR
```

リセット解除後はここからスタート

割り込みスタックポインタ設定

プロセッサモードレジスタ（04h）を書き換えるために、プロテクトレジスタ（0ah）のビット１（プロセッサモードレジスタへの書き込み許可ビットをセット（書き込み許可）

シングルチップモードに設定：
詳細はユーザーズマニュアル p 1-23 参照

プロテクトレジスタ(0ah)のビット１をクリア（書込み禁止）

フラグレジスタ設定：通常のスタックは USP（ユーザスタックポインタを使用）

ユーザスタックポインタの設定

スタティックベースレジスタ設定

割り込みテーブルレジスタ設定：このアドレスに加算する形で割り込み処理プログラムの先頭アドレスを決定するので、割り込みを使う場合には必ず設定する

```
;====================================================================
; NEAR area initialize.
;--------------------------------------------------------------------
; bss zero clear
;--------------------------------------------------------------------
     N_BZERO    bss_SE_top,bss_SE
     N_BZERO    bss_SO_top,bss_SO
     N_BZERO    bss_NE_top,bss_NE
     N_BZERO    bss_NO_top,bss_NO

;--------------------------------------------------------------------
; initialize data section
;--------------------------------------------------------------------
     N_BCOPY data_SEI_top,data_SE_top,data_SE
     N_BCOPY    data_SOI_top,data_SO_top,data_SO
     N_BCOPY    data_NEI_top,data_NE_top,data_NE
     N_BCOPY    data_NOI_top,data_NO_top,data_NO

;====================================================================
; FAR area initialize.
;--------------------------------------------------------------------
; bss zero clear
;--------------------------------------------------------------------
;    BZERO bss_FE_top,bss_FE
;    BZERO bss_FO_top,bss_FO
;
;--------------------------------------------------------------------
; Copy edata_E(O) section from edata_EI(OI) section
;--------------------------------------------------------------------
;    BCOPY data_FEI_top,data_FE_top,data_FE
;    BCOPY data_FOI_top,data_FO_top,data_FO
;
;    ldc  #stack_top,sp
;    .stk -40
```

Near 領域の初期設定

Near 領域の bss セクション（0ffffh 番地までで初期値を持たない静的変数領域）の初期化（0 クリア）

Near 領域の data セクション（初期値を持つ静的変数領域）の初期化（ROM 領域から初期値データをコピーする）

Far 領域の RAM の初期化
（OAKS16 ではこの領域に RAM を配置していないので使用しない→コメントアウトする）

```
;======================================================================
; heap area initialize
;----------------------------------------------------------------------
;     .glb __mbase
;     .glb __mnext
;     .glb __msize
;     mov.w#(heap_top&0FFFFH), __mbase
;     mov.w#(heap_top>>16), __mbase+2
;     mov.w#(heap_top&0FFFFH), __mnext
;     mov.w#(heap_top>>16), __mnext+2
;     mov.w#(HEAPSIZE&0FFFFH), __msize
;     mov.w#(HEAPSIZE>>16), __msize+2
;
;======================================================================
; Initialize standard I/O
;----------------------------------------------------------------------
;     .glb _init
;     .call_init,G
;     jsr.a_init
;
;======================================================================
; Call main() function
;----------------------------------------------------------------------
      ldc   #0h,fb     ; for debuger

      .glb _main
      jsr.a_main

;======================================================================
; exit() function
;----------------------------------------------------------------------
;     .glb _exit
;     .glb $exit
;_exit:                    ; End program
;$exit:
;     jmp   _exit
;     .einsf
;======================================================================
; dummy interrupt function
;----------------------------------------------------------------------
dummy_int:
      reit
      .end
;**********************************************************************
;
;     C COMPILER for M16C/60
;     Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;     AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
```

Heap 領域の初期化：このテキストではメモリ管理関数を使用しないのでコメントアウト

標準入出力関数の初期化：このテキストでは、標準入出力関数を使用しないのでコメントアウト（組み込み用のプログラムでは出来るだけ使わないことをお勧めします。）

Main 関数の呼び出し

Exit 関数部：組み込みマイコンのプログラムでは基本的には存在しません。ですが、リアルタイム OS などの使用によって必要な場合も考えられます。（このテキストでは使用しません）

ダミー割り込み処理関数部：想定していない割り込みが発生してしまった場合何も処理を行なわずに本のプログラムに復帰させる

以上が OAKS16 で使用するスタートアッププログラムの作り方です。このスタートアッププログラムをこれからあとの例題のスタートアッププログラムとして使用していきます。
変更したスタートアッププログラムはもとのプログラムの名前と同じですので、混同しないように注意して使用してください。

2.2. プリプロセッサの処理

C コンパイラには、プリプロセッサというコンパイルの前処理を行なうソフトが標準でついています。ここでは、マクロ展開や条件付コンパイル、及び指定されたファイルの包含などが行なわれます。プアリプロセッサへの指示（プリプロセスコマンド）は＃で始まります。

OAKS16 で使用している NC30 のプリプロセッサ ccp30 には＃include(ファイルの包含)や#define（マクロ定義）のほか、＃PRAGMA ではじまる NC30 特有の拡張機能が存在します。（＃PRAGMA ではじまる記述は、コンパイルの対象となる CPU に依存する記述を表します。）

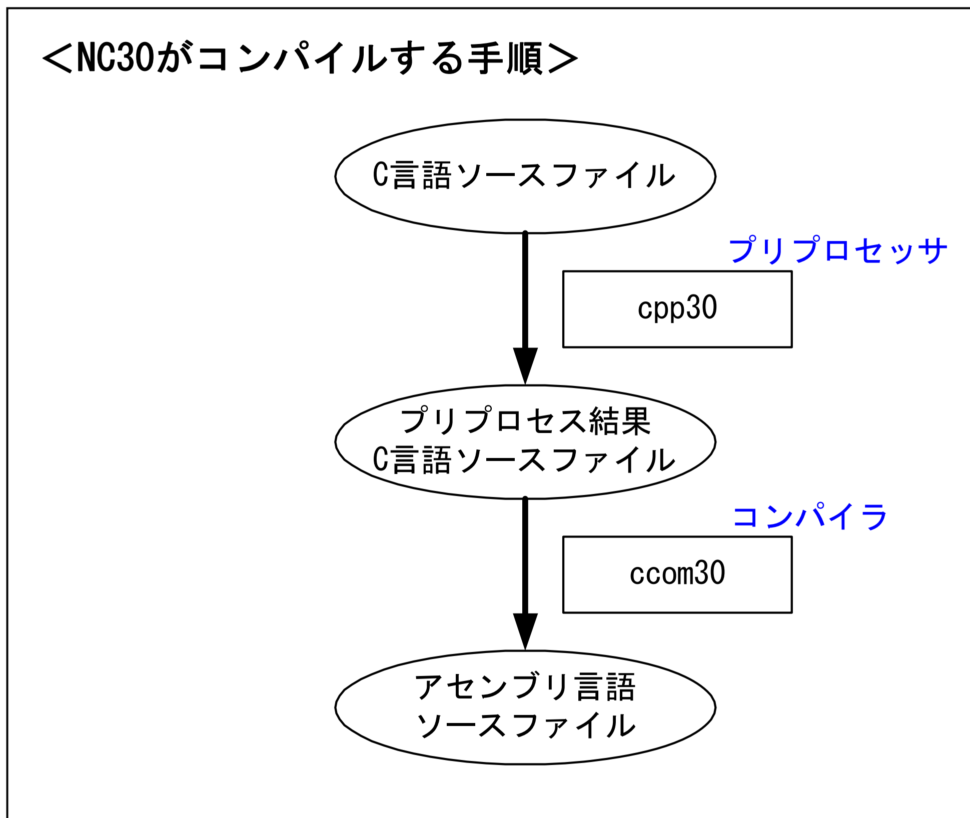


図 2.4

2.2.1. sfr設定の為のヘッダファイル

三菱のマイコンには SFR（スペシャルファンクションレジスタ）という領域が存在します。ここは、マイコンの動作モードや周辺 I/O の設定を行なうレジスタで、1 つのアドレスに 8 ビットのレジスタを持っています。この SFR に値を設定したり、SFR から値を読み出したりする場合、プログラム中に対象となるアドレスを記述すればよいのですが、それでは、後でリストをみたときに、何をしているのか解りにくいのです。ところで、SFR のそれぞれのレジスタにはシンボルやビットシンボルが決められています。プログラムのリスト中にこのシンボルで記述出来れば、プログラムがずっとわかりやすくなります。

OAKS16 では、M30620FCAFP の SFR のシンボル定義をまとめた oaks_sfr.h を用意しました。このファイルでは、プリプロセスコマンドで、SFR のマクロ定義やシンボル、ビットの定義を行なっています。

このファイルを MTOOL のフォルダ内（パスを通したフォルダ）内に入れて下さい。C 言語のソースファイルの先頭で

#include　<oaks_sfr.h>　（＃include はファイルを包含する為のプリプロセスコマンドです）

と記述すれば、シンボルやビットシンボルを使用して記述できます。

ここで、oak_sfr.h の説明をします。ファイルの内容が多いので、部分的に取り上げて説明しますので、ファイル内容全てを確認する為には付属のファイルをご覧下さい。

①#pragma ADDRESS による変数の絶対アドレスの指定

```
/*************************************************************************
*   ファイル名   : oaks_sfr.h                                            *
*                                                                        *
*   内容    : definition of M16C/62A's SFR                               *
*                                                                        *
*   作成    : oaks16kit support                                          *
*                                                                        *
*   Version      : 1.00  ( 2002- 3-8 ) Initial                           *
*************************************************************************/
/******************************************************
*   declare SFR addresses                             *
******************************************************/

#pragma ADDRESS     pm0_addr    0004H       /* Processor mode register 0 */

#pragma ADDRESS     pm1_addr    0005H       /* Processor mode register 1 */

#pragma ADDRESS     cm0_addr    0006H       /* System clock control register 0 */
              .
              .
              .
              .
#pragma ADDRESS     pur2_addr   03feH       /* Pull-up control register 2 */

#pragma ADDRESS     pcr_addr    03ffH       /* Port control register */
```

②8 ビット単位で使用する（ビットごとのアクセスができない）SFR の定義

```
/*******************************************************
*   declare  SFR char                                  *
*******************************************************/
unsigned char   da0_addr;              /* D-A register 0 */

#define      da0      da0_addr

unsigned char   da1_addr;              /* D-A register 1 */
#define      da1      da1_addr

/*------------------------------------------------------
    Up/down flag ; Use "MOV" instruction when writing to this regist
------------------------------------------------------*/
unsigned char   udf_addr;              /* UP/down flag */
```

ｄa0_addr という８ビットの変数を用意する

ｄa0 という記述は、da0_addr のことであるというマクロ宣言

①で#pragma ADDRESS　da0_addr　03d8h

と宣言しているので、この記述以降の da0 という記述は、**“03d8h 番地の 8 ビットのデータ領域”**を表すことになります。

da1,udf についても同様です。

③16 ビット単位で使用する SFR の定義

```
/*******************************************************
*   declare  SFR short                                 *
*******************************************************/
/*------------------------------------------------------
    Timer registers : Read and write data in 16-bit units.
------------------------------------------------------*/
unsigned short   ta11_addr;            /* Timer A1-1 register */

#define      ta11      ta11_addr

unsigned short   ta21_addr;            /* Timer A2-1 register */
#define      ta21      ta21_addr
          .
          .
          .

unsigned short   tb2_addr;             /* Timer B2 register */
#define      tb2      tb2_addr
```

ta11_addr という 16 ビットの変数を用意する

ta11 という記述は ta11_addr のことであるというマクロ宣言

テキスト内省略

①で#pragma ADDRESS　ta11_addr　0342h

と宣言しているので、この記述以降の ta11 という記述は、**“0342h 番地の 16 ビットのデータ領域”**を表すことになります。

④構造体と共用体を使ったビット宣言

ここでは、C 言語の記述“構造体”と“共用体”を使用して SFR のビット宣言をします。

SFR 領域のデータの中にはは、ビット一つ一つが意味を持つものがあります。これらをプログラム中で扱うにはビットシンボルで記述したほうがわかりやすくなります。NC30 ではビット記述でのコンパイルを可能にしていますので、ビットの定義をして使用します。

・8 ビット構成の SFR のビット定義

```
/********************************************************
*   declare SFR bit                                     *
********************************************************/
struct bit_def {
        char    b0:1;
        char    b1:1;
        char    b2:1;
        char    b3:1;
        char    b4:1;
        char    b5:1;
        char    b6:1;
        char    b7:1;
};

union byte_def{
    struct bit_def bit;
    char    byte;
};

/*------------------------------------------------------
    Processor mode register 0
------------------------------------------------------*/
union byte_def pm0_addr;

#define     pm0     pm0_addr.byte

#define     pm00        pm0_addr.bit.b0     /* Processor mode bit */
#define     pm01        pm0_addr.bit.b1     /* Processor mode bi
#define     pm02        pm0_addr.bit.b2     /* R/W mo
#define     pm03        pm0_addr.bit.b3     /* Software reset bit */
#define     pm04        pm0_addr.bit.b4     /* Multiplexed bus space select bit */
#define     pm05        pm0_addr.bit.b5     /* Multiplexed bus space select bit */
#define     pm06        pm0_addr.bit.b6     /* Port P40 to P43 function select bit */
#define     pm07        pm0_addr.bit.b7     /* BCLK output disable bit */


 （省略）

/*------------------------------------------------------
     dm1sl
------------------------------------------------------*/
#define     dm1sl       dm1sl_addr.byte

#define     dsel0_dm1sl dm1sl_addr.bit.dsel0        /* DMA request cause select bit */
#define     dsel1_dm1sl dm1sl_addr.bit.dsel1        /* DMA request cause select bit */
#define     dsel2_dm1sl dm1sl_addr.bit.dsel2        /* DMA request cause select bit */
#define     dsel3_dm1sl dm1sl_addr.bit.dsel3        /* DMA request cause select bit */
#define     dms_dm1sl   dm1sl_addr.bit.dms          /* DMA request cause expansion bit */
```

bit_def という名前の構造体ビットフィールドの宣言。b０から b７までの名前で 1 ビットずつの領域を確保する構造体枠を宣言している。

byte_def という名前の共用体の宣言。メンバーとして bit_def 型の bit という名の構造体と char 型の byte という名の変数を宣言している。

共用体の領域をメモリ上に確保する。(pm0_addr は 0004h 番地)

Byte データとビットデータのマクロ定義（詳細は次ページ参照）

ここで、もう少し詳しく順を追って、構造体、共用体宣言による SFR 定義の内容を説明します。

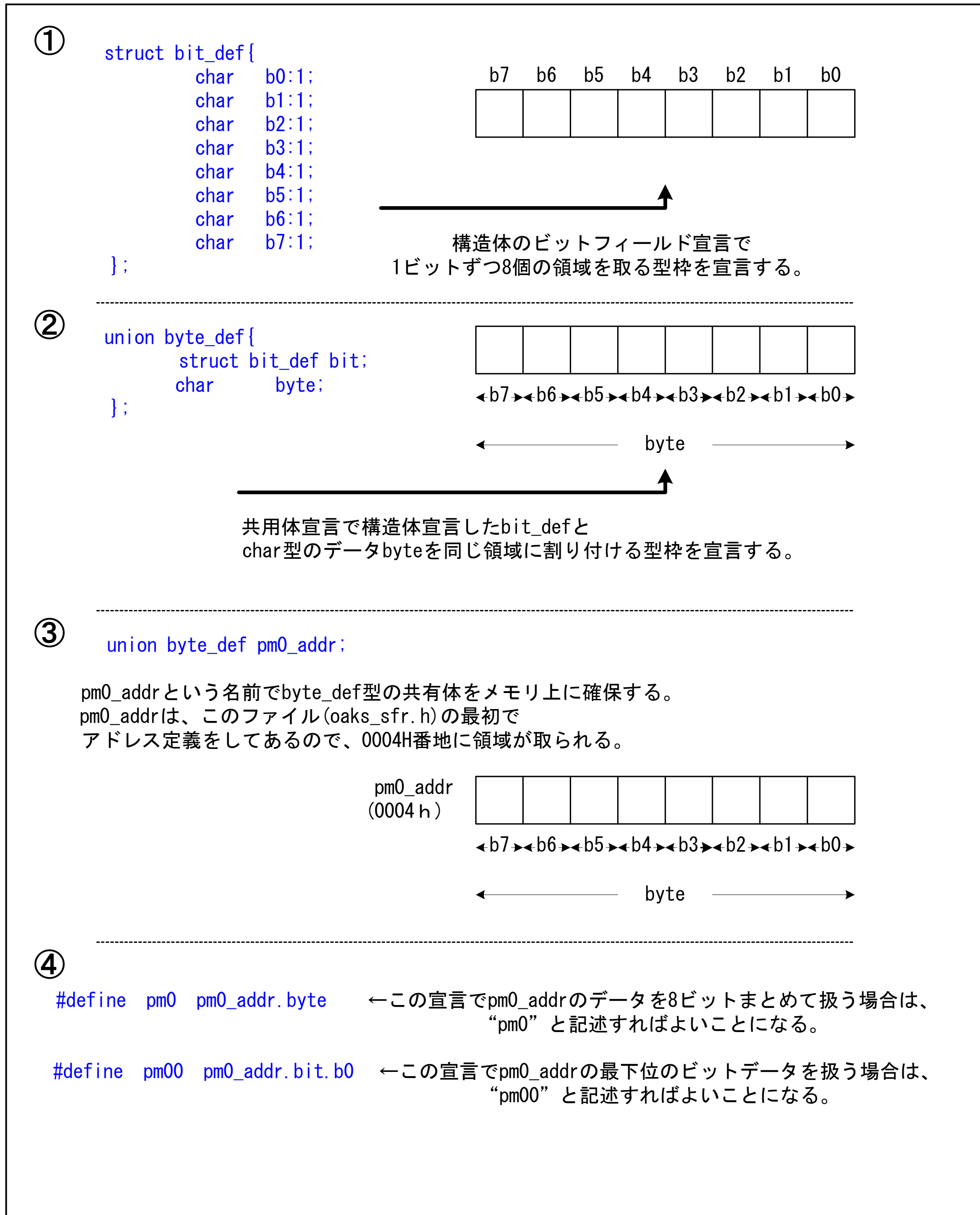


図 2.5

・共用体による SFR の定義

SFR の中には、16 ビット構成のもの、20 ビット構成のもの、2byte 構成のもの、3byte 構成のものなども存在します。それらの SFR も、シンボルやビットシンボルで扱えるように定義してあります。考えかたは、今まで説明してきたことと同じですので、確認して使用してください。

```
/*******************************************************
*   declare SFR union                                  *
*******************************************************/

union{
    struct{
        char    b0:1;
        char    b1:1;
        char    b2:1;
        char    b3:1;
        char    b4:1;
        char    b5:1;
        char    b6:1;
        char    b7:1;
        char    b8:1;
        char    b9:1;
        char    b10:1;
        char    b11:1;
        char    b12:1;
        char    b13:1;
        char    b14:1;
        char    b15:1;
        char    b16:1;
        char    b17:1;
        char    b18:1;
        char    b19:1;
    }bit;
    struct{
        char    low;                /* low  8 bit */
        char    mid;                /* mid  8 bit */
        char    high;               /* high 8 bit */
        char    nc;                 /* non use */
    }byte;
    unsigned long   dword;
}rmad0_addr,rmad1_addr,sar0_addr,sar1_addr,dar0_addr,dar1_addr;
#define     rmad0       rmad0_addr.dword        /* Address match interrupt register 0 32 bit */
#define     rmad0l      rmad0_addr.byte.low         /* Address match interrupt register 0 low  8 bit */
#define     rmad0m      rmad0_addr.byte.mid         /* Address match interrupt register 0 mid  8 bit */
#define     rmad0h      rmad0_addr.byte.high        /* Address match interrupt register 0 high 8 bit */
```

以上のような形で、SFR の宣言ができます。C 言語のリストの中に

#include　＜oaks_sfr.h＞

と記述して、使用してください。

　この記述は、NC30 のパスの通った INC フォルダに oaks_sfr.h ファイルがあることが前提です。C 言語ソースファイルと同じフォルダにヘッダファイルを置くこともできますが、その場合は C 言語ファイル中の記述を、

#include “oaks_sfr.h”

としてください。

（NC30 マニュアル内、#include 記述の項参照のこと）

3. プログラミング

ここから、実際に LCD ボードを使用して、プログラムを作っていきます。プログラムには、定石と言うような、定番となった考え方がいくつかあります。それらの方法をいくつか紹介していきます。

まずここで、LCD ボードを使用したプログラムを 1 つ作ってみましょう。C 言語の文法をおさらいするものとして、このプログラムを完成させ、そのプログラムを用途に応じて修正していくことによって、プログラム技法を紹介していくことにします。

3.1. 基本のプログラム

仕様

LCD ボードの sw1 から sw8 を led1 から led8 に対応させ、スイッチが ON のとき LED を点灯させ、OFF のとき LED を消灯させる。（例題 1）

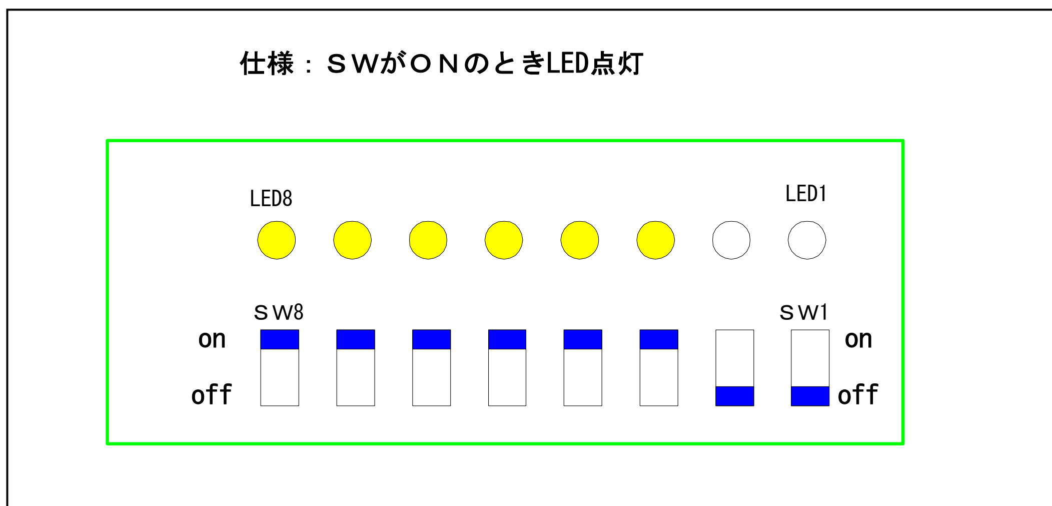


図 3.1

考え方

スタートアッププログラムは２章で作成したものをそのまま使います。

Ｃ言語ソースプログラム内では oaks_sfr.h をインクルードして記述します。

フローチャート

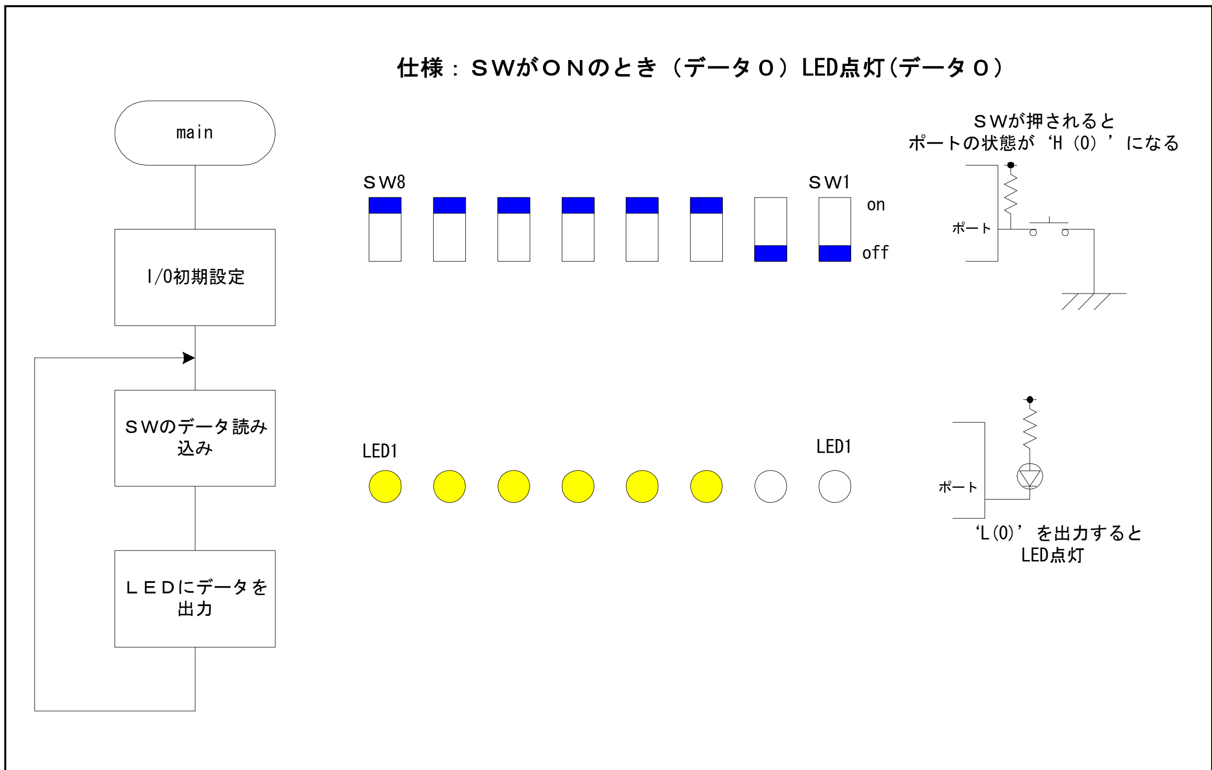


図 3.2

この仕様の場合、スイッチから入力したデータをそのまま LED に出力すれば OK です。
ですが、この後の仕様変更を考慮して、スイッチから入力したデータを一度メモリに入力して、それから LED に出力します。

書き方１：(メモリを使用しない)

```
p7=p3;
```

書き方２：(メモリを使用する)

```
unsigned char data; /* データ領域を 8 ビットで取りたいので char 型を使用する */
data＝p3;
p7=data;
```

リスト

```
/****************************************************/
/*   プロジェクト名: rei1                            */
/*   ファイル名：    rei1.c                          */
/*   内容：          SW の ON した LED 点灯          */
/*   日付：          2002.3.8                        */
/*   コンパイラ:     NC30WA (Ver.4.00)               */
/*   note:           OAKS16 対応                     */
/*   作成者：        OAKS16KIT support               */
/****************************************************/

#include  <oaks_sfr.h>

/* プロトタイプ宣言 */
void _main(void);    /* メインプログラム*/

/* マクロ定義 */
#define PORTIN  0x00 /* ポート方向レジスタを入力に設定する為のデータ */
#define   PORTOUT   0xff /* ポート方向レジスタを出力に設定する為のデータ */
#define   LEDOFF    0xff /* LED1 から LED8 までを消灯するデータ */



void main( void ){

     unsigned char data;  /* 変数宣言（スイッチの接続されているポートのデータを入れる） */

     p7=LEDOFF;      /* ポート 7 に消灯データを設定 */

     pd7=PORTOUT;    /* ポート 7 を出力に設定 */

     pd3=PORTIN;          /* ポート 3 を入力に設定 */


     for(;;){
          data = p3;/* ポート 3 (sw1 から sw8) のデータ読み込み */
          p7=data;   /* ポート 7 (LED1 から LED8) にデータ出力 */
     }

}
```

3.1.1. 論理演算

先ほどの基本のプログラムでは、入力したデータをそのまま出力していました。ここでは、論理演算を使用したデータの加工方法を説明します。論理演算とはデジタル回路等で 1（真と偽）の二つの値だけを扱う演算です。この演算はマイコンの制御でよく使われるものです。

用語の意味

真：値がＯＮになっている状態。1 も同等の意味。

偽：値が OFF になっている状態。0 も同等の意味。

真理値表：入出力の全てのパターンを表す表。

MIL 記号：論理回路上での論理演算を記号で表したもの。

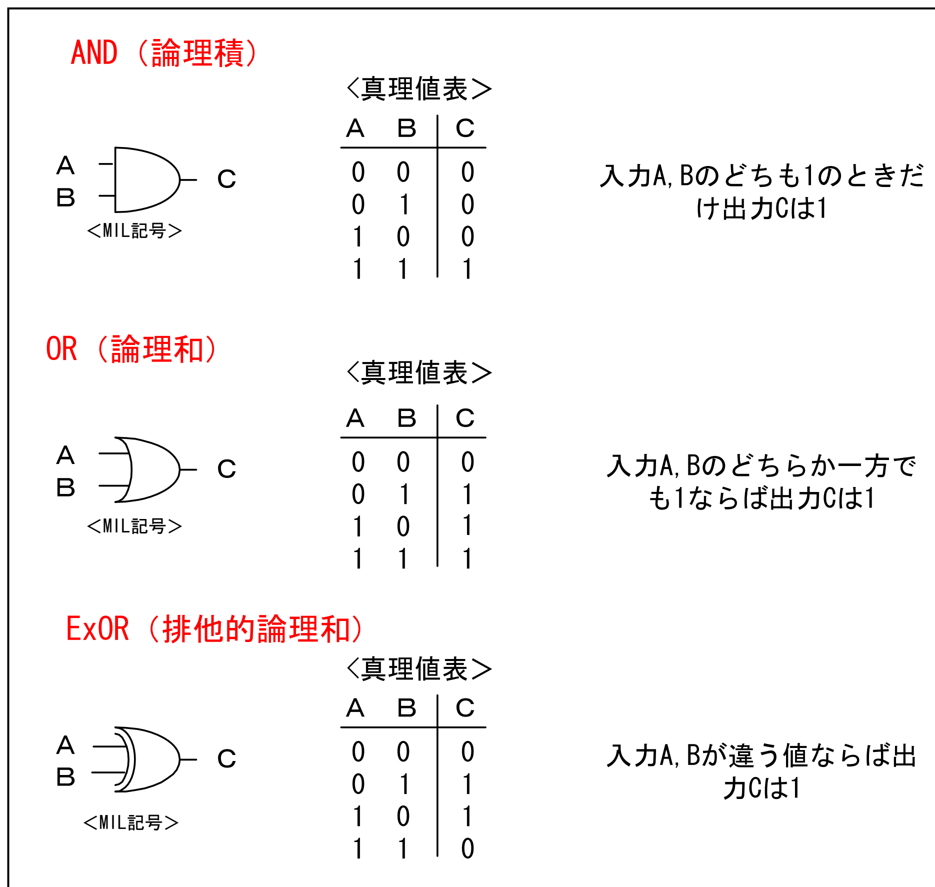


図 3.3

これらの論理演算はデータを加工する為に良く使われます。たとえば、データのあるビットを残して**他のビットをすべて 0 にしたいとき**（他のビットを「マスクする」といいます。）は、**論理積**を使います。0 と AND をとったビットはすべて 0 になり、1 と AND をとったビットはそのままの値が残ります。

また、特定のビット以外のビットを 1 にしたいときは、論理和を使います。1 と OR をとった

ビットはすべて 1 となり、0 と OR をとったビットだけがそのまま残ります。
また、**ビットを反転したいときには** 1 との**排他的論理和**をとります。これにより、0 は 1 に、1 は 0 になります。

3.1.2. データの反転

今度はデータのビットを反転させるプログラムを考えてみます。

仕様

SW が OFF のとき LED を点灯、ON のとき LED を消灯させて見ます。(例題 1_a)

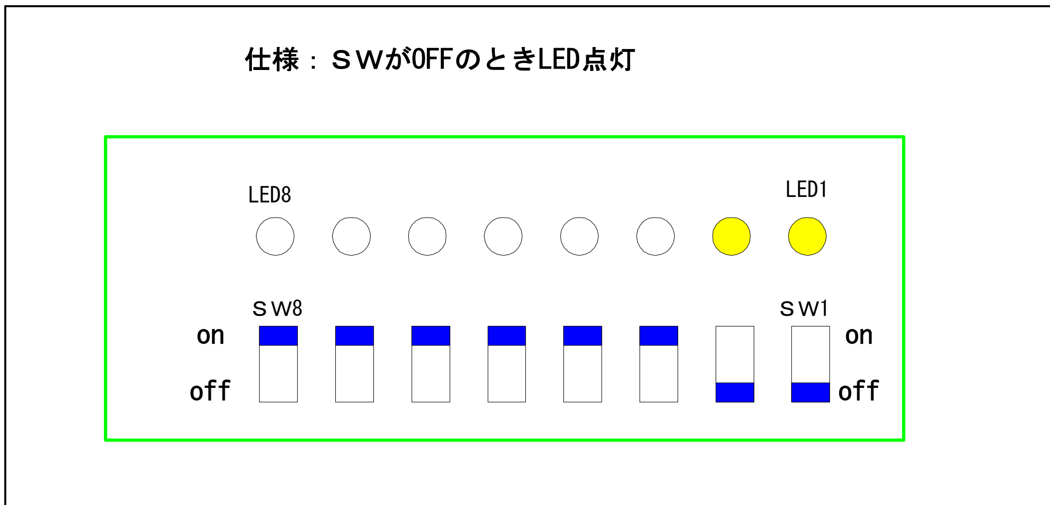


図 3.4

考え方

先ほどの**排他的論理和**を使用し、データを加工します。

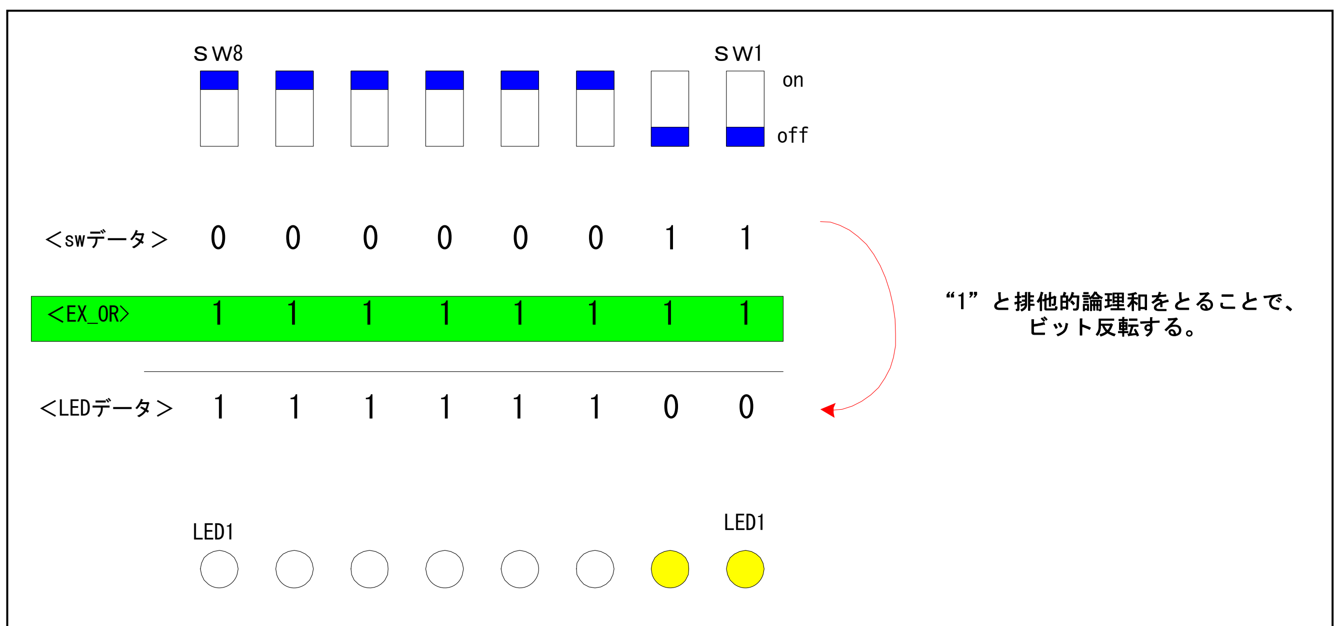


図 3.5

フローチャート

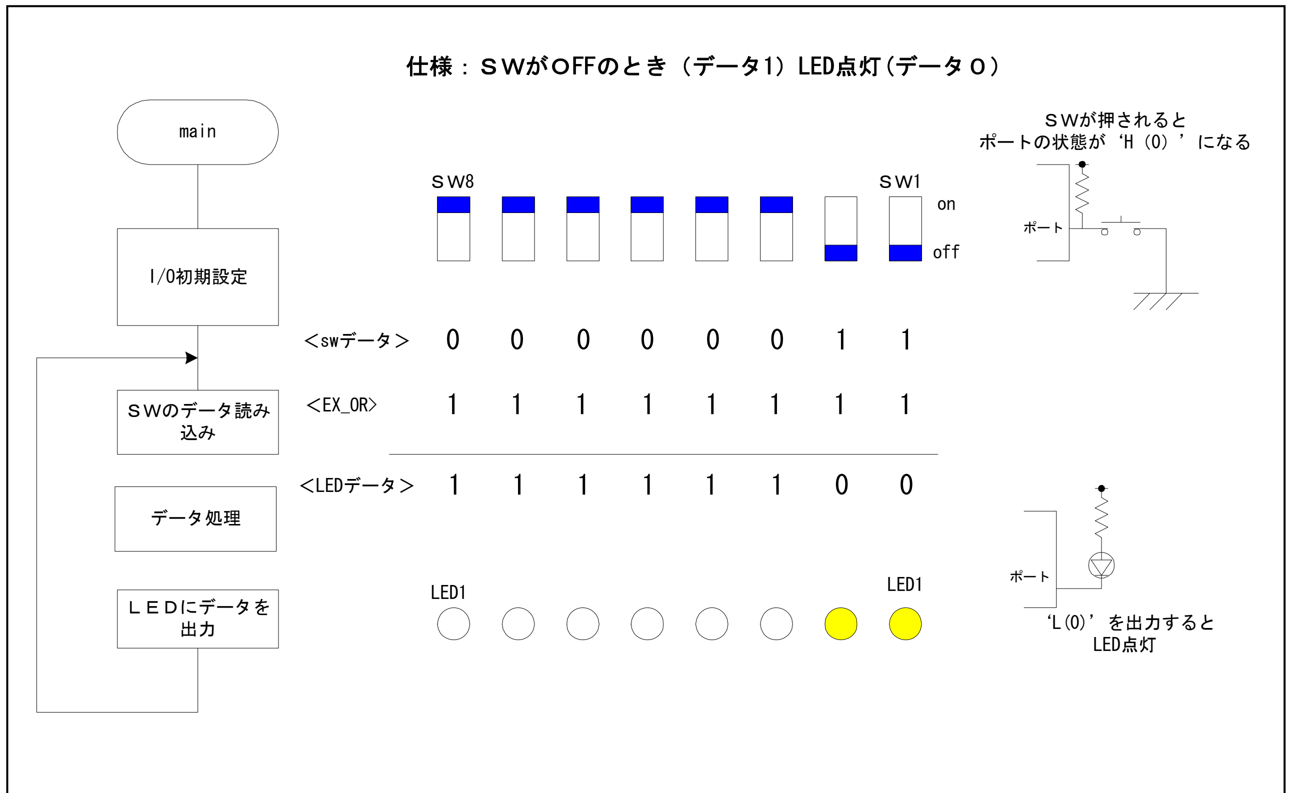


図 3.6

リスト

```
/*****************************************************/
/*    プロジェクト名: reil_a                          */
/*    ファイル名：    reil_a.c                        */
/*    内容：          sw の OFF した LED 点灯         */
/*    日付：          2002.3.8                        */
/*    コンパイラ:     NC30WA (Ver.4.00)               */
/*    note:           OAKS16 対応                     */
/*    作成者：        OAKS16KIT support               */
/*****************************************************/

#include   <oaks_sfr.h>

/* プロトタイプ宣言 */
void _main(void);    /* メインプログラム*/

/* マクロ定義 */
#define PORTIN  0x00 /* ポート方向レジスタを入力に設定する為のデータ */
#define    PORTOUT    0xff /* ポート方向レジスタを出力に設定する為のデータ */
#define    LEDOFF     0xff /* LED1 から LED8 までを消灯するデータ */


void main( void ){

     unsigned char data;       /* 変数宣言（スイッチの接続されているポートのデータを入れる） */

     p7=LEDOFF;      /* ポート 7 に消灯データを設定 */

     pd7=PORTOUT;    /* ポート 7 を出力に設定 */

     pd3=PORTIN;          /* ポート 3 を入力に設定 */


     for(;;){
          data = p3; /* ポート 3 (sw1 から sw8) のデータ読み込み */
          data ^=0xff;    /* data と 0xff の排他的論理和をとる */
          p7=data;   /* ポート 7 (LED1～LED8) にデータ出力 */
     }

}
```

一度取り込んだデータを FF と排他的論理和（EXOR）をとる

3.1.3. データのマスク処理

今度はデータのマスク処理について考えてみます。

仕様

SW1 がＯＮのとき LED 全てを点灯、０ＦＦのとき LED 全てを消灯させて見ます。（例題 1_ｂ）

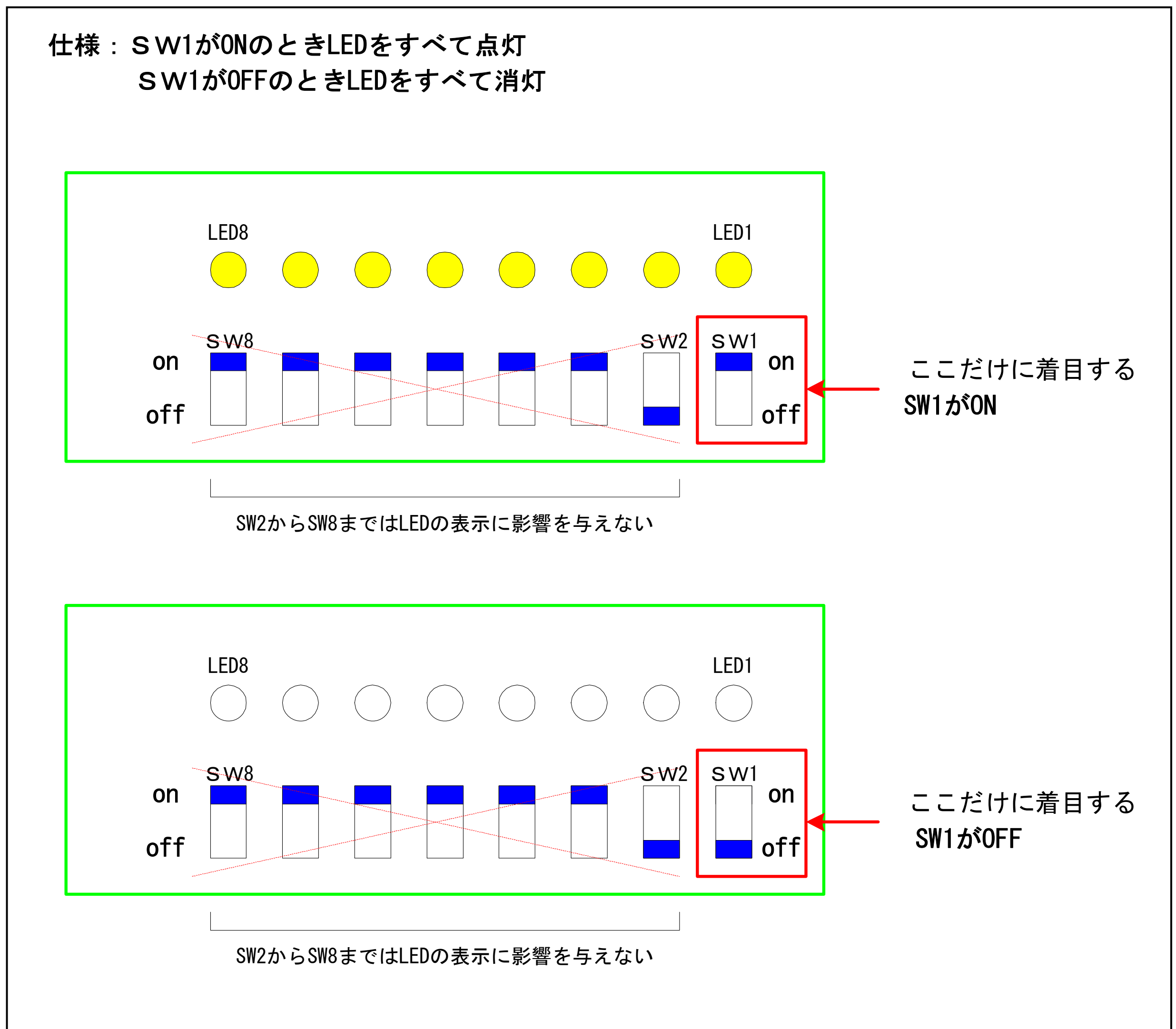


図 3.7

考え方

p3 のデータを読み込むと sw1 が ON の場合は 256×1/2 パターンあります。そのパターン全てを分岐条件にするのは大変なので、通常は判断に使用するビット以外は 0 にしてしまう（「マスクする」といいます）という方法をとります。これには先ほどの**論理積**を使用しますと 0 と AND をとったビットはすべて 0 になり、1 と AND をとったビットはそのままのこります。

フローチャート

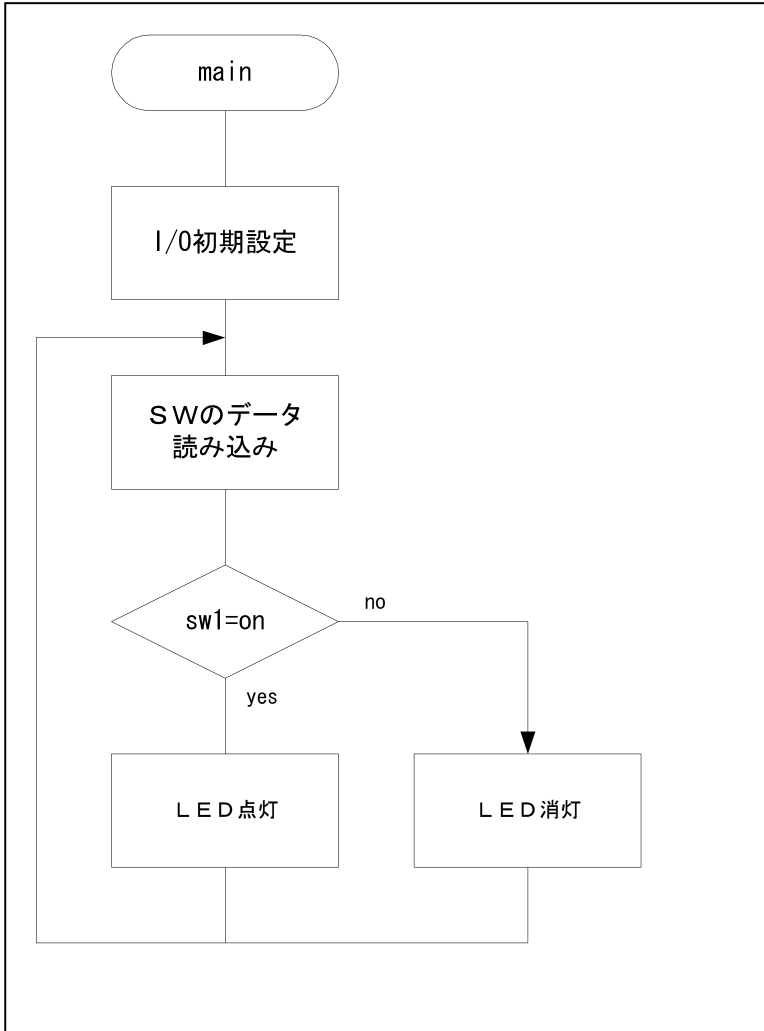


図 3.8

リスト

```
/****************************************************/
/*   プロジェクト名: rei1_b                       */
/*   ファイル名：    rei1_b.c                     */
/*   内容：          SW1 が ON なら LED すべて点灯  */
/*   日付：          2002.3.8                     */
/*   コンパイラ:     NC30WA (Ver.4.00)            */
/*   note:           OAKS16 対応                  */
/*   作成者：        OAKS16KIT support            */
/****************************************************/

#include  <oaks_sfr.h>

/* プロトタイプ宣言 */
void _main(void);    /* メインプログラム*/

/* マクロ定義 */
#define PORTIN  0x00 /* ポート方向レジスタを入力に設定する為のデータ */
#define    PORTOUT    0xff /* ポート方向レジスタを出力に設定する為のデータ */
#define    LEDOFF     0xff /* LED1 から LED8 までを消灯するデータ */
#define    LEDON 0x00 /* LED1 から LED8 までを点灯するデータ */

void main( void ){

     unsigned char data;  /* 変数宣言（スイッチの接続されているポートのデータを入れる）  */

     p7=LEDOFF;      /* ポート 7 に消灯データを設定 */

     pd7=PORTOUT;    /* ポート 7 を出力に設定 */

     pd3=PORTIN;          /* ポート 3 を入力に設定 */

     for(;;){
          data = p3;/* ポート 3 (sw) のデータ読み込み */
          data &= 0x01;/* sw1 以外のビットをマスク */
          if(data == 0){
               p7 = LEDON;/* SW1 が ON なら LED 点灯 */
          }
          else{
               p7 = LEDOFF;/* sw1 が off なら LED 消灯 */
          }
     }
```

Sw のデータを 00000001b と AND をとり、最下位ビット以外は強制的に“0”にする

3.2. スイッチ回路の考え方

今度は、スイッチを 1 回 ON するたびに LED をシフトすることを考えてみます。

仕様

最初に LED1 だけを点灯させておき、sw9 が一回押されたら LED2 だけが点灯する、もう一度 sw9 が押されたら LED3 だけが点灯するというように led の点灯位置を 1 つ左に移動させる。(例題 2)

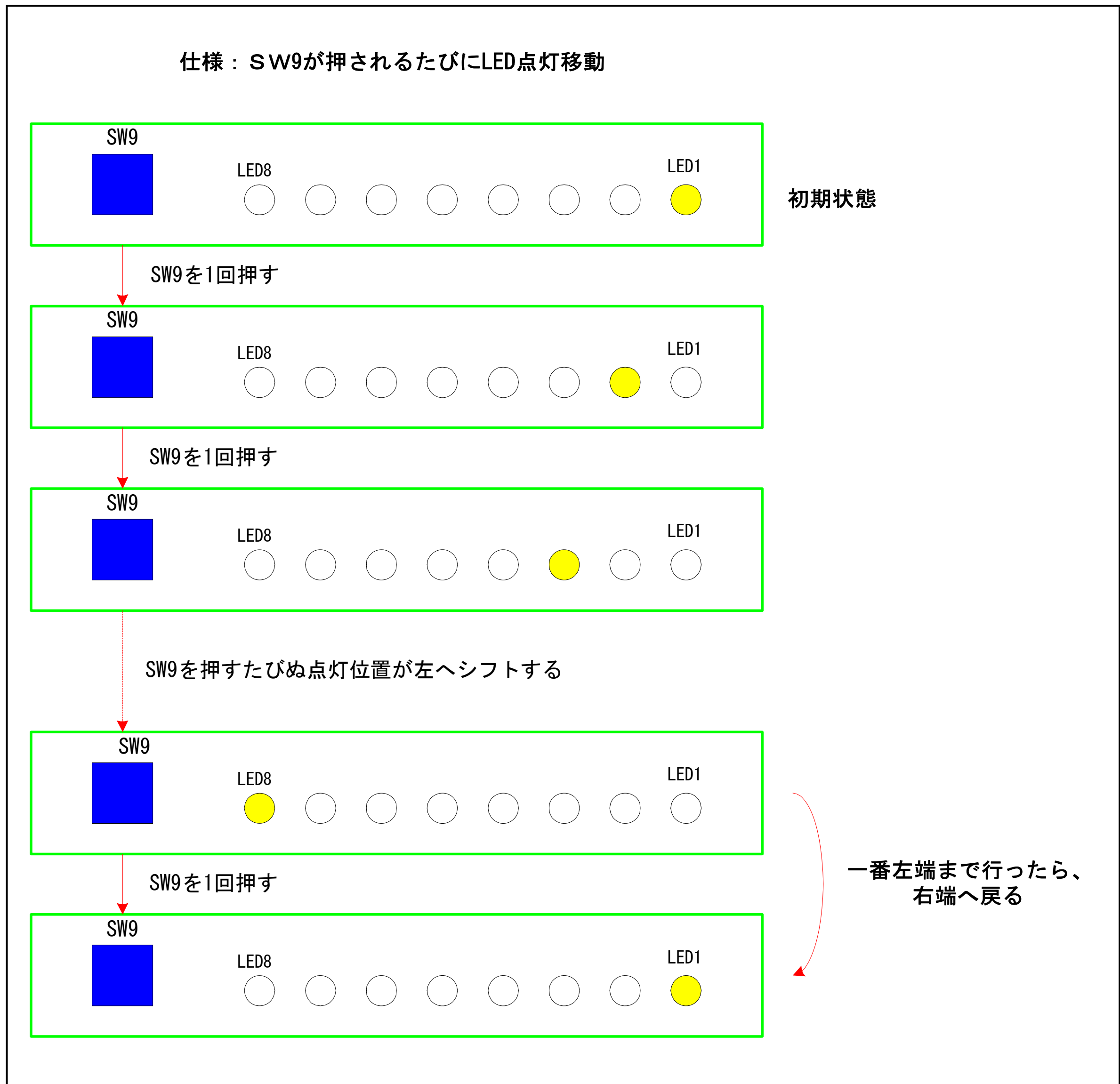


図 3.9

考え方

sw9 は port8 の bit2 に接続されています。この端子は割り込み端子（int0）としても使用できますがここではただのポートとして使用します。
Sw9 はプッシュスイッチですので、「一回押されたら」という判断はポートの状態をただ読むだけでは判断できません。「スイッチが OFF から ON になった」という変化を読み取る必要があります。「ON」になったと判断したら、一度「OFF」の状態が存在してから再び「ON」にならないと次のカウントをしてはいけません。このようなプログラムを作成するときには、判断の為のデータ（フラグと呼びます）を使用します。これは M16C の内部のフラグではなく、メモリに変数として確保します。今回の場合スイッチ処理終了フラグを用意し、sw9 が ON の場合の処理が終わったら、フラグをセットします。このフラグがセットされている間は、同じ処理は行わず、sw9 が OFF になったときに処理終了フラグをクリアします。次に SW9 が ON されれば最初の一回はスイッチ処理終了フラグがクリアなので処理を行い、またフラグをセットしておきます。このようにプログラムを作成することで、「プッシュスイッチを押すたびに」というプログラムを作成することが出来ます。

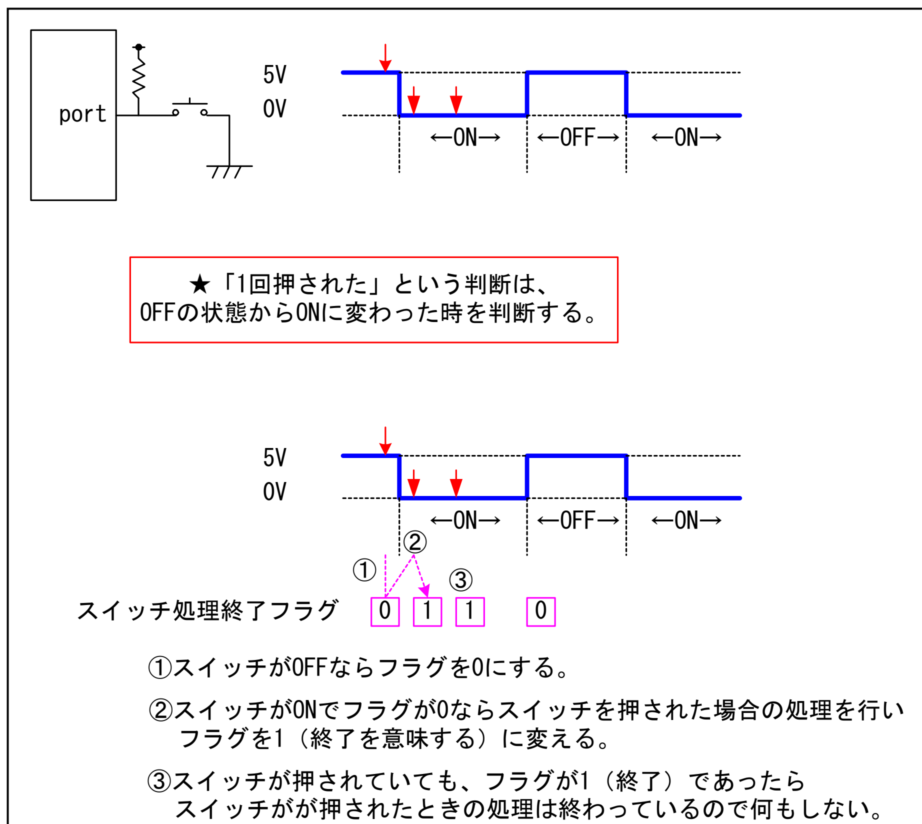


図 3.10

フローチャート

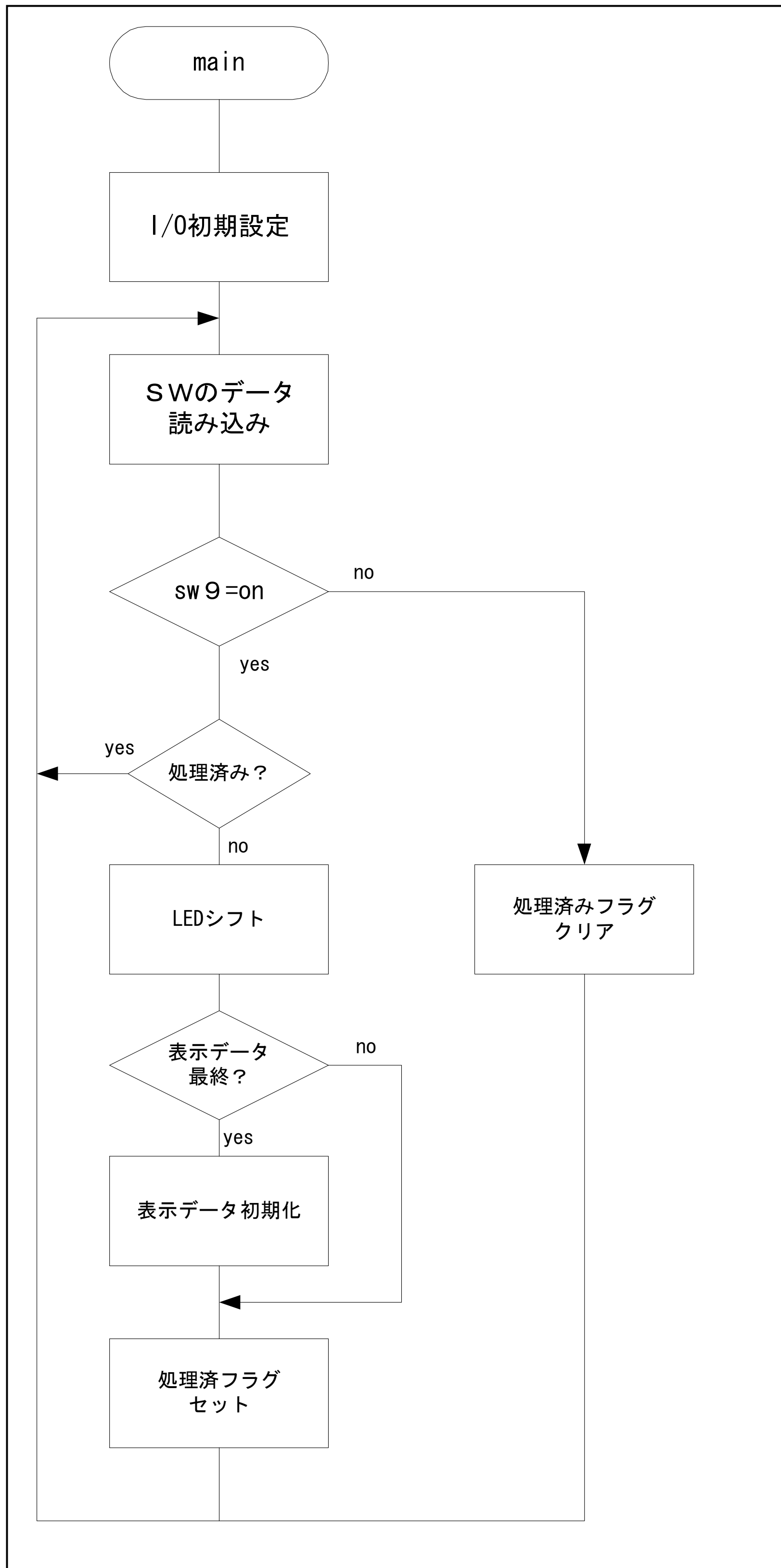


図 3.11

リスト

```
/*********************************************************************/
/*    プロジェクト名: rei2                                           */
/*    ファイル名：    rei2.c                                         */
/*    内容：          SW1 を ON するたびに LED の点灯位置をシフト    */
/*    日付：          2002.3.8                                       */
/*    コンパイラ:     NC30WA (Ver.4.00)                              */
/*    note:           OAKS16 対応                                    */
/*    作成者：        OAKS16KIT support                              */
/*********************************************************************/

#include  <oaks_sfr.h>

/* プロトタイプ宣言 */
void _main(void);     /* メインプログラム*/

/* マクロ定義 */
#define PORTIN  0x00 /* ポート方向レジスタを入力に設定する為のデータ */
#define    PORTOUT    0xff /* ポート方向レジスタを出力に設定する為のデータ */

void main( void ){

     unsigned char leddata;     /* LED に出力するデータのための変数 */
     unsigned char swdata;      /* SW の状態を表すデータのための変数 */
     unsigned char swflg; /* スイッチ処理終了フラグ */

     leddata=0x01;              /* LED データの初期値 */
     swflg=0;                   /* スイッチ処理終了フラグの初期値 */

     p7=leddata^0xff;           /* LED にデータ出力 */
     pd7=PORTOUT;               /* ポート 7 を出力に設定 */
     pd8=PORTIN;                     /* ポート 8 を入力に設定 */

     for(;;){
          swdata = p8 & 0x04;        /* ポート 8 のデータを読み、sw9 の状態だけをデータとして取り出す */
          if(swdata == 0){           /* スイッチが ON なら */
               if(swflg==0){         /* スイッチ処理終了フラグがクリアなら */
                    leddata<<=1;     /* LED データを左へ 1 シフト */
                    if(leddata==0)
                         leddata=0x01;
                    p7 = leddata^0xffff; /* LED データ出力 */
                    swflg = 1;                /* スイッチ処理終了フラグをセットする */
               }
          }
          else{
                    swflg=0;                  /* スイッチ処理終了フラグをクリアする */
          }

     }
}
```

3.2.1. チャタリングとは

例題３で思うような動作にならなかった方はいらっしゃいませんか？一度しか押していないのにとなりの LED ではなく、2,3 個となりの LED が点灯したりしていませんか？これはチャタリングと呼ばれる現象です。スイッチは押された瞬間に接点がバウンド（付いたり離れたりする）します。そのため、短い時間で処理を行なうマイコンでは、一度しか押されていないのに何度も押されたように判断してしまうのです。

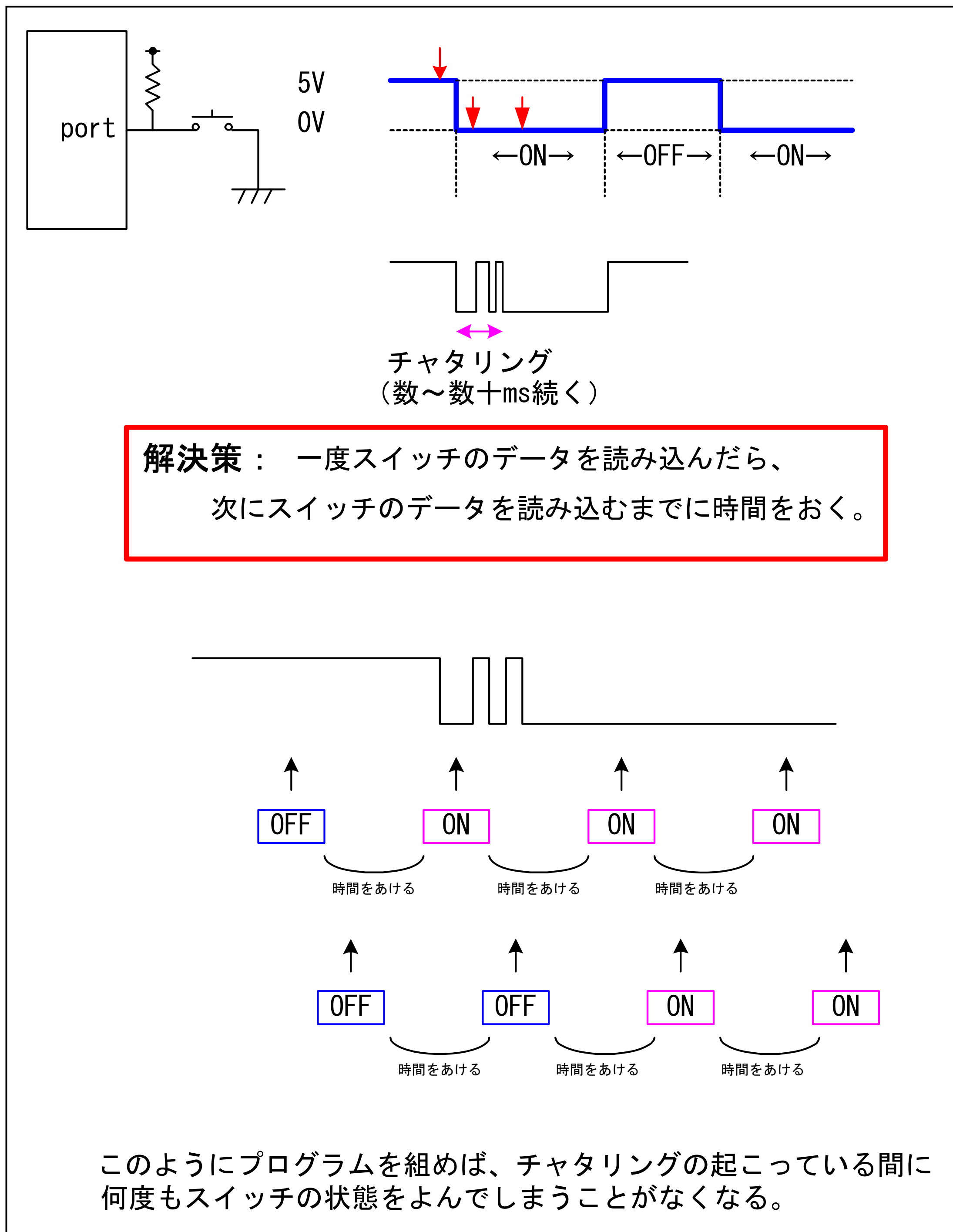


図 3.12

3.2.2.S/Wでのチャタリング除去

それでは、具体的にチャタリングを避けるためのプログラムを考えてみましょう。
方法はいくつかありますが、ここでは、プログラムの中に、「ソフトウエイト」を入れる方法を実行してみます。この、「ソフトウエイト」とは、入出力に何も影響を与えずに、必要な時間だけを経過させるというものです。以下にソフトウエイトのプログラム例を示します。

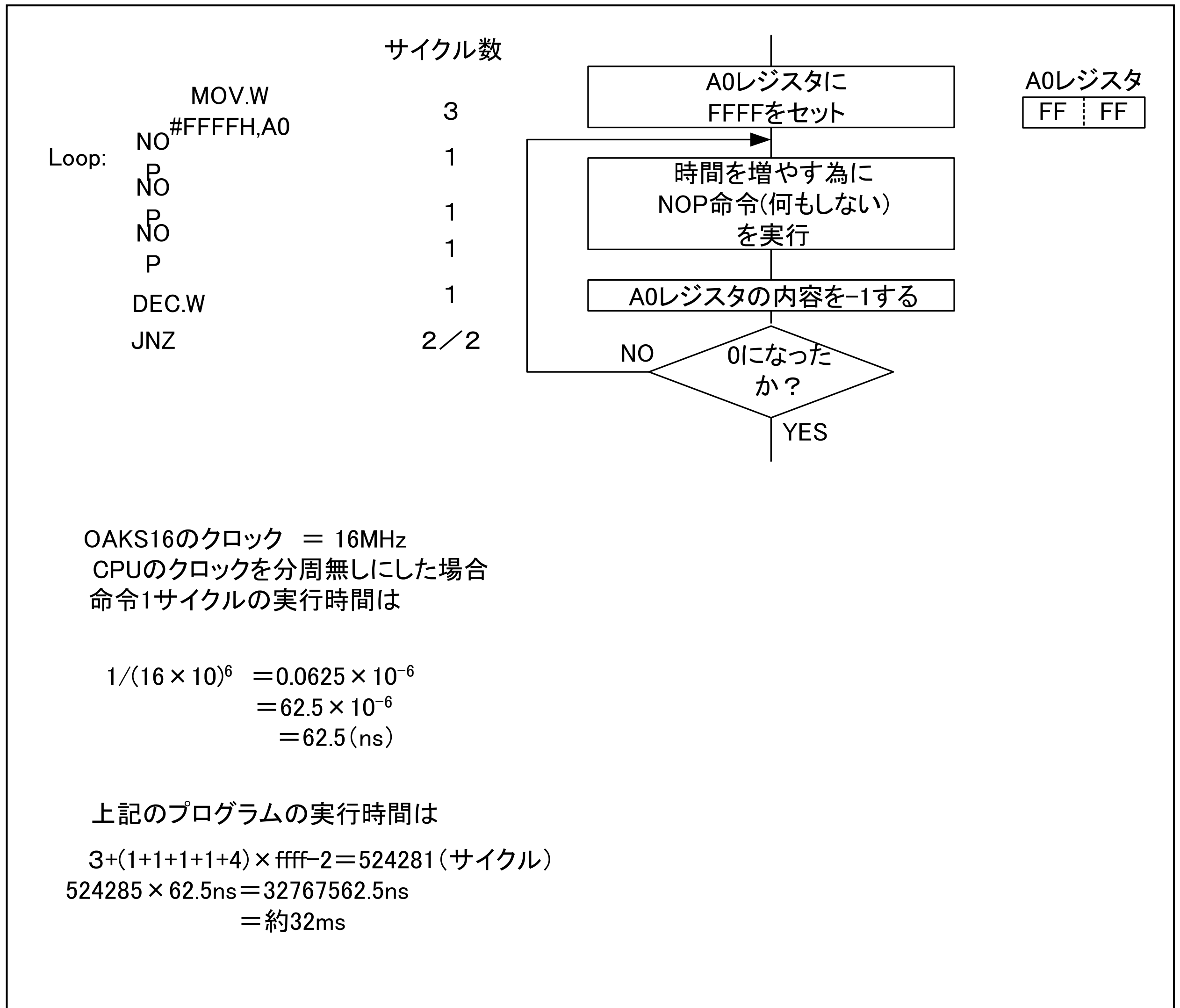


図 3.13

このプログラムはアセンブリ言語記述で書いています。C 言語ではどのようなアセンブリ言語に翻訳されるかわからないので、プログラムの実行時間を気にする処理では、アセンブリ言語記述を使用したほうが処理時間は、比較的明確になります。この場合、C 言語のリスト中に記述するので
#pragma　ASM～＃pragmaENDASM というアセンブリ言語記述のための書き方を使用します。以下に、このソフトウエイトを使用したリストを示します。

リスト

```
/*********************************************************************/
/*   プロジェクト名: rei2_a                                        */
/*   ファイル名 : r  ei2_a.c                                       */
/*   内容 :         SW1 を ON するたびに LED の点灯位置をシフト      */
/*                  チャタリング除去 (ソフトウエイト)              */
/*   日付 :         2002.3.8                                       */
/*   コンパイラ:     NC30WA (Ver.4.00)                              */
/*   note:          OAKS16 対応                                    */
/*   作成者 :       OAKS16KIT support                              */
/**********************************************************************/
#include   <oaks_sfr.h>

/* プロトタイプ宣言 */
void _main(void);    /* メインプログラム*/
void wait(void);     /* ソフトウエイトプログラム */

/* マクロ定義 */

#define PORTIN  0x00 /* ポート方向レジスタを入力に設定する為のデータ */
#define    PORTOUT    0xff /* ポート方向レジスタを出力に設定する為のデータ */

void main( void ){

     unsigned char leddata;     /* LED に出力するデータのための変数 */
     unsigned char swdata;      /* SW の状態を表すデータのための変数 */
     unsigned char swflg; /* スイッチ処理終了フラグ */

     leddata=0x01;/* LED データの初期値 */
     swflg=0;/* スイッチ処理終了フラグの初期値 */

     p7=leddata^0xff;      /* LED にデータ出力 */
     pd7=PORTOUT;          /* ポート 7 を出力に設定 */
     pd8=PORTIN;                /* ポート 8 を入力に設定 */

     for(;;){
          swdata = p8 & 0x04;       /* ポート 8 のデータを読み、sw9 の状態だけをデータとして取り出す */
          wait();                        /* チャタリング除去のための WAIT */
          if(swdata == 0){          /* スイッチが ON なら */
               if(swflg==0){        /* スイッチ処理終了フラグがクリアなら */
                    leddata<<=1;    /* LED データを左へ 1 シフト */
                    if(leddata==0)  /* 左端までのシフトが終了したら */
                         leddata=0x01;/* LED データ初期値を入れる */
                    p7 = leddata^0xffff;/* LED データ出力 */
                    swflg = 1;      /* スイッチ処理終了フラグをセットする */
               }
          }
          else{
                    swflg=0;   /* スイッチ処理終了フラグをクリアする */
          }

     }
}

void wait(void){           /* 約 32ms のウエイト */
#pragma ASM                     /*アセンブラ表記*/
     MOV.W      #0FFFFH,A0      /*カウンタ初期値セット*/
 LOOP:
     NOP
     NOP
     NOP
     DEC.WA0
     JNZ   LOOP       /*ループ終了？*/
#pragma ENDASM                  /*アセンブラ表記終了*/
}
```

3.3.割り込みプログラム

割り込みとは、元々CPU が実行していた処理に別の処理を割り込ませる機能です。割り込みが発生するとそのとき実行されていた処理は中断され、割り込み処理が実行されます。そして、割り込み処理が終了した後で、元の処理の続きが実行されます。M16C では外部端子や内部タイマなどの回路から割り込み信号を入力することによって割り込みを発生させる方法と、プログラム内で命令語を実行することによって割り込みを発生させる方法があります。割り込みのプログラムを実行する為には割り込みプログラムの記述、CPU の設定など必要な処理があります。

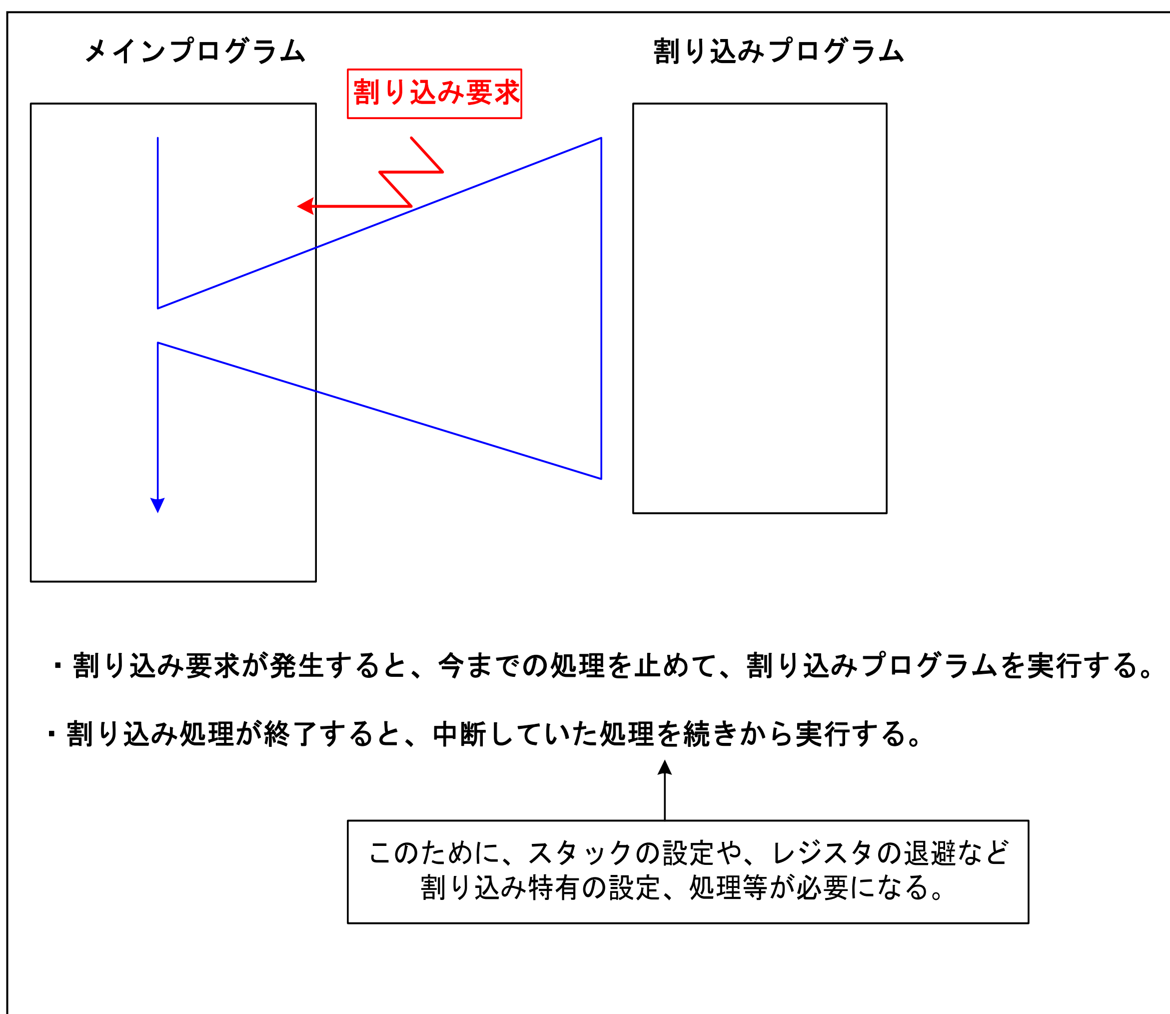


図 3.14

3.3.1.M16Cの割り込み

M16C には以下に示すような割り込みがあります。このテキストでは、ハードウエア周辺 I/O の中の INT1 による外部端子からの割込み、タイマ 0 を使用したタイマ割り込みの例を説明していきます。

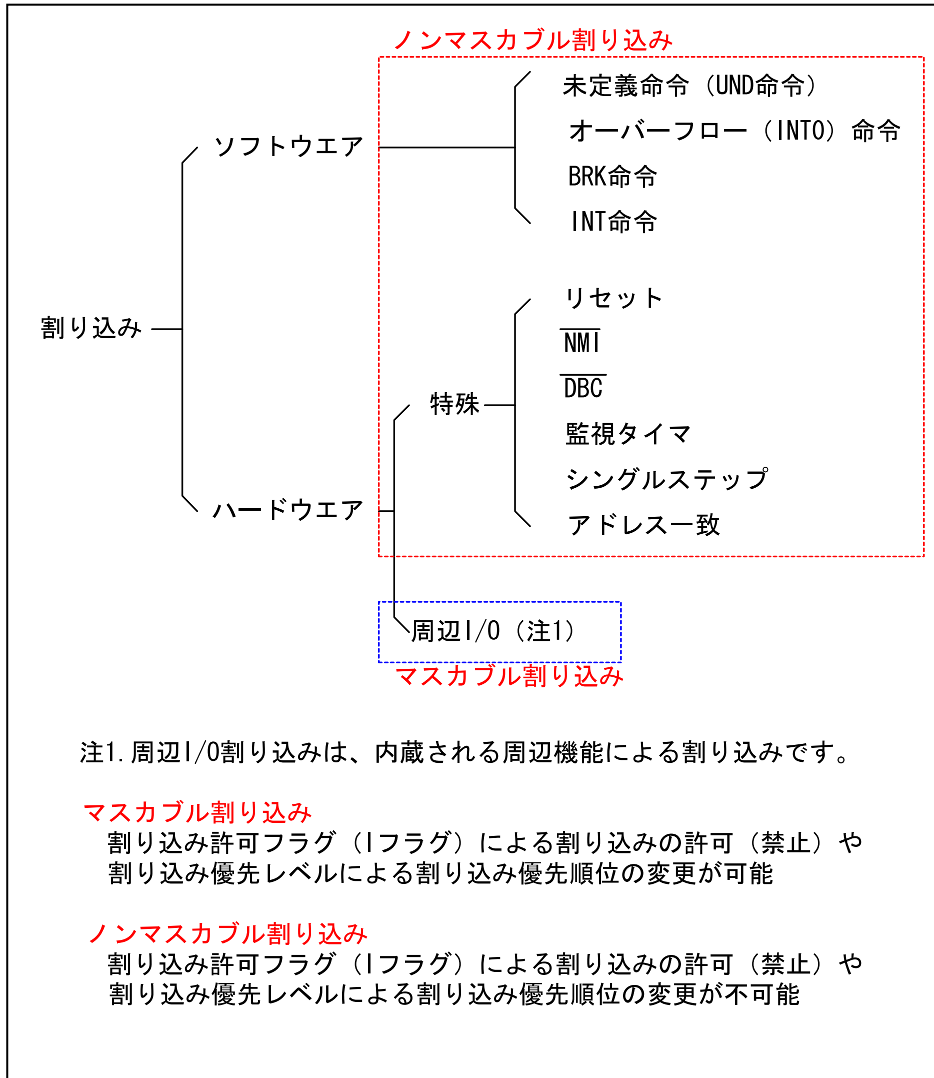


図 3.15

3.3.2. 割り込みを使用するために必要な処理

プログラムで割り込みを使用するためには、次の処理が必要です。今回は、C 言語でプログラム記述をしているので、割り込み処理も C 言語で記述します。

① 割り込み関数の記述
② 割り込みの許可
③ 割り込みベクタの設定

割り込み関数

割り込みで実行させたい処理を記述する関数です。

割り込み関数は**#pragma INTERRUPT 関数名** で宣言します。

割り込み関数は、元の処理を中断して実行されるので、割り込み関数の中で書き換えられたくないレジスタを割り込み前後に退避･復帰する必要があります。(C 言語で記述していると、レジスタを使用していないような気になりますが、C 言語ソースはアセンブリ言語に落とされますのでそこでレジスタは使用されています。）また、割り込み関数が終了したあと、元の関数へ戻る処理も必要です。**#pragma** INTERRUPT を使用することで、これらの処理を自動的に行ないます。

割り込み許可

割り込み許可は次の 2 条件件で許可され、両方の条件が揃ったときのみ、割り込みが受け付けられます。リセット後はどちらも禁止の状態になっています。

- **I フラグ（割り込み禁止フラグ）による許可**

  全てのマスカブル割り込みが許可されます。

- **割り込み要因ごとの割り込み優先レベルの設定による許可**

  該当する要因の割り込みのみを許可します。

  割り込み優先レベルは１～７のとき割り込み許可です。１～７では、数値が大きいほど優先レベルが高くなります。したがって、重要な割り込みが優先して実行されるように、割り込みに優先順位を設定します。

割り込みベクタの設定

割り込みベクタは、割り込み関数の先頭番地を格納しておくところです。割り込み関数は、他の関数と違い、別の関数から呼ばれて実行されるのではありません。割り込み要求が受け付けられたときに割り込みベクタに格納された番地からプログラムを実行するという動作によって割り込み関数が実行されます。割り込みベクタの設定は、2.1.2. スタートアッププログラムの作成を参照して下さい。

割り込みの処理順序

割り込みは次のような手順で実行されます。

① 割り込み要因からの割り込み要求

② 割り込み優先レベル判定

③ 割り込み処理

a)割り込みシーケンス（割り込み要求の受付から割り込み関数の先頭番地に到るまでの処理で、マイコンのハードウエアが自動的に行ないます。）

b)レジスタを退避します。

c)割り込み関数の実行

d)レジスタ復帰、元の処理への復帰

NC30 では#pragma INTERRUPT を使用すると、コンパイラが自動的に必要な命令語（レジスタの退避、復帰、リターン）を付加してくれます。そのため b)、d)の処理は自動的に行なわれ、割り込み関数内に記述する必要はありません。

割り込みシーケンス

① 00000h 番地を読むことで、CPU は割り込み情報（割り込み番号、割り込み要求レベル）を獲得する。その後、該当する割り込みの要求ビットが“0”になる。

（このため、プログラム中で 00000h 番地を読み出さないで下さい。割り込み要求がキャンセルされる場合があります。）

② 割り込みシーケンス直前のフラグレジスタ(FLG)の内容を CPU 内部の一時レジスタ（注 1）に退避する。

③ 割り込み許可フラグ（I フラグ）、デバッグフラグ（D フラグ）、及びスタックポインタ指定フラグ）を“0”にする。（ただし U フラグは、ソフトウエア割り込み番号 32～63 の INT 命令を実行した場合は変化しない）

（スタックポインタ指定フラグ“0”は、ISP（割り込みスタックポインタ）の使用を指定します。ISP は、リセット時に、00000h になっていますので、スタートアッププログラム内で、必ず、スタック領域の最終番地を指定してください。）

④ CPU 内部の一時レジスタの内容をスタック領域に退避する。

⑤ プログラムカウンタの内容をスタック領域に退避する。

⑥ プロセッサ割り込み優先レベル(IPL)に、受け付けた割り込みの割り込み優先レベルを設定する。

割り込みシーケンス終了後は、割り込みルーチンの先頭番地から命令を実行する。

3.3.1. INT0 割り込み端子からの割り込み

OAKS16LEDboard には、INT0 と INT1 割り込み端子にプッシュスイッチを接続してあります。残念ながら、このスイッチは H/W でチャタリングの除去をしていませんので、割り込みプログラムを書くときにもチャタリングを考えなければなりません。そのため、ここでは、1 回だけ割り込み処理を行なうプログラムをつかって、割り込みプログラムの記述方法を説明します。

INT0 を制御するレジスタ

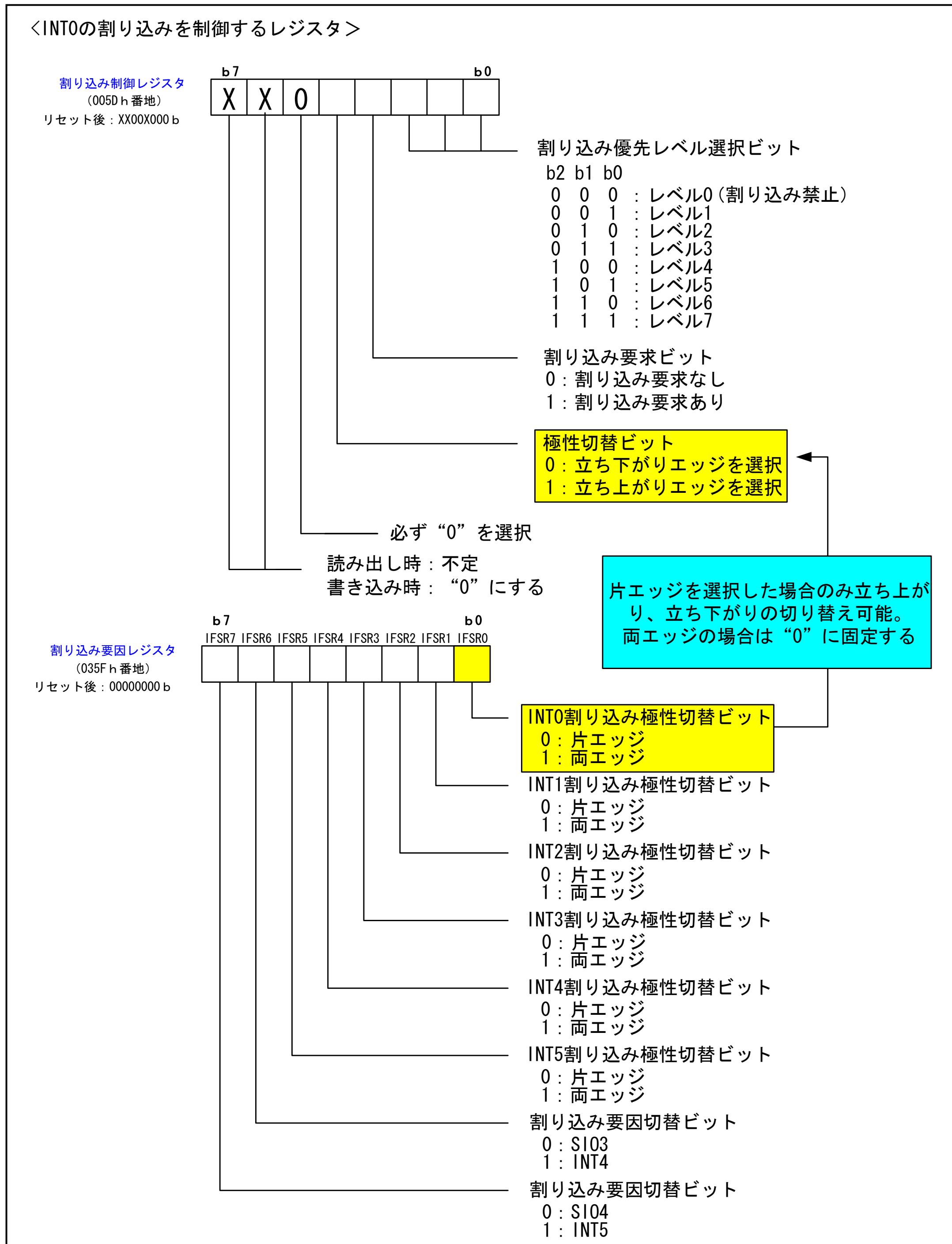


図 3.16

仕様

　プログラムスタート時 LED1~LED8 まで消灯。INT0 のスイッチが押されたら LED1～LED8 まで点灯。（例題 3）

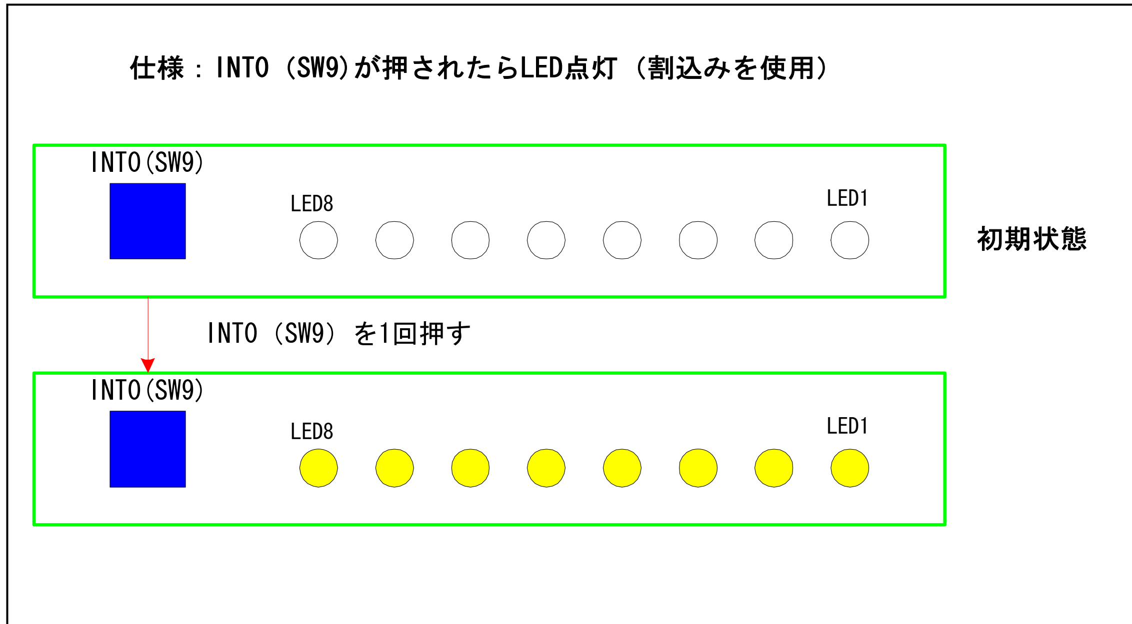


図 3.17

考え方

　main プログラム：LED を消灯させたら、割り込み待ちの無限ループを組む。
割り込みプログラム：LED を点灯させ割り込みから復帰させる。

フローチャート

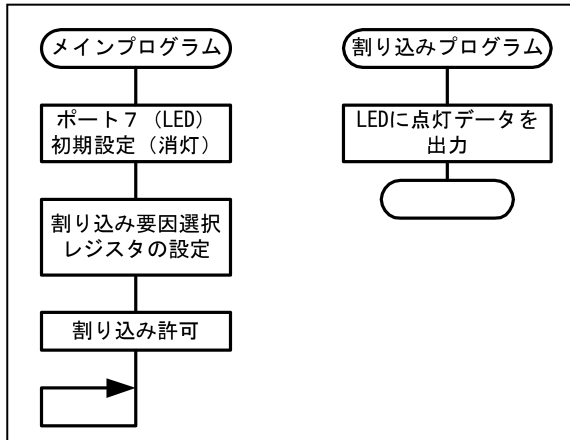


図 3.18

ファイルの分割

スタートアッププログラム：o_srt0.a30+o_sec30.inc

メインプログラムファイル：rei3.c＋OAKS16.h

ファイルの変更

① o_sect30.inc の割り込みベクタのアドレスを設定する。

② main プログラムに割り込みプログラムを記述する。

リスト

o_ncr0.a30(OAKS16 用のスタートアッププログラム：本テキスト 2.2.1.参照)

```
;*************************************************************************** ;
;     C COMPILER for M16C/60
;     Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;     AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;     All Rights Reserved.
;
;        ncrt0.a30 : NC30 startup program
;
;     This program is applicable when using the basic I/O library
;
;        $Id: ncrt0.a30,v 1.13 2000/06/22 13:17:04 simomura Exp $
;
;***************************************************************************
;          .glb  __BankSelect
;__BankSelect   .equ 0BH
;---------------------------------------------------------------------
; HEEP SIZE definition
;---------------------------------------------------------------------
;HEAPSIZE  .equ  300h

;---------------------------------------------------------------------
; STACK SIZE definition
;---------------------------------------------------------------------
STACKSIZE  .equ  300h

;---------------------------------------------------------------------
; INTERRUPT STACK SIZE definition
;---------------------------------------------------------------------
ISTACKSIZE .equ  300h

;---------------------------------------------------------------------
; INTERRUPT VECTOR ADDRESS  definition
;---------------------------------------------------------------------
VECTOR_ADR      .equ  0fa000h

 (省略)

start:
;---------------------------------------------------------------------
; after reset,this program will start
;---------------------------------------------------------------------
     ldc   #istack_top,    isp   ;set istack pointer
     mov.b #02h,0ah             ;プロセッサモードレジスタ書き込みイネーブル
;    bset  1,0ah
     mov.b #00h,04h             ;シングルチップモード
     mov.b #00h,05h             ;非拡張、ノーウエイト
;    bclr  1,0ah
     mov.b #00h,0ah             ;プロセッサモードディゼーブル
;
     mov.b #01h,0ah             ;クロックコントロールレジスタ書き込みイネ
     mov.b #08h,06h             ;発振
     mov.b #20h,07h             ;分周なし
     mov.b #00,0ah                   ;クロックコントロールレジスタ書き込みディゼーブル
;
;    ldc   #0080h,    flg         ;ユーザスタックを指定
     ldc   #stack_top,     sp    ;set stack pointer
     ldc   #data_SE_top,   sb    ;set sb register
     ldintb      #VECTOR_ADR
```

割り込み用のスタックサイズを定義する

可変ベクタの先頭アドレスを定義する。ユーザが使用できる ROM 領域で、ユーザプログラムに支障のない領域にする。また、できるだけ偶数番地を設定する。

ISP（割り込みスタックポインタ）に値をセットする。

ここでは USP を指定しているが、割り込み処理のときだけ自動的に ISP が使われる。ここで、ISP を設定して USP を使わないということも可能。

可変ベクタアドレスの設定

o_sect30.a30(OAKS16 用セクション定義ファイル：本テキスト 2.2.1.参照)

```
;****************************************************************************
;
;     C Compiler for M16C/60
;        Copyright 1995-1998 MITSUBISHI ELECTRIC CORPORATION
;        AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;        All Rights Reserved.
;
;     Written by T.Aoyama
;
;        sect30.inc      : section definition
;        This program is applicable when using the basic I/O library
;
;        $Id: sect30.inc,v 1.9 2000/06/20 09:07:11 simomura Exp $
;
****************************************************************************
(省略)
;---------------------------------------------------------------
; Stack area
;---------------------------------------------------------------
     .section   stack,DATA
     .blkb STACKSIZE
stack_top:

     .blkb ISTACKSIZE
istack_top:

(省略)
;---------------------------------------------------------------
; variable vector section
;---------------------------------------------------------------
     .section   vector          ; variable vector table
     .org  VECTOR_ADR
;
     .lword     dummy_int       ; vector 0 (BRK)
     .org  (VECTOR_ADR +44)
     .lword     dummy_int       ; DMA0 (for user)
     .lword     dummy_int       ; DMA1 2 (for user)
     .lword     dummy_int       ; input key (for user)
     .lword     dummy_int       ; AD Convert (for user)
     .org  (VECTOR_ADR +68)
     .lword     dummy_int       ; uart0 trance (for user)
     .lword     dummy_int       ; uart0 receive (for user)
     .lword     0fcb6bh              ; uart1 trance (for user)
     .lword     0fcb6bh              ; uart1 receive (for user)
     .lword     dummy_int       ; TIMER A0 (for user)
     .lword     dummy_int       ; TIMER A1 (for user)
     .lword     dummy_int       ; TIMER A2 (for user)
     .lword     dummy_int       ; TIMER A3 (for user)
     .lword     dummy_int       ; TIMER A4 (for user) (vector 25)
     .lword     dummy_int       ; TIMER B0 (for user) (vector 26)
     .lword     dummy_int       ; TIMER B1 (for user) (vector 27)
     .lword     dummy_int       ; TIMER B2 (for user) (vector 28)
     .glb  _int0
     .lword     _int0           ; INT0 (for user)
     .lword     dummy_int       ; INT1 (for us
     .lword     dummy_int       ; INT2 (for us
```

割り込みスタックを確保し、その採取番地に“istak_top”のラベルをつける

KD30 で使用しています。必ず、このように記述してください！

INT0 のプログラムの先頭番地を記述する。プログラムは、このファイルとは別ファイルになるので、“.glb”で宣言する。

rei3.c(メインプログラム+割り込みプログラム)

```
/****************************************************/
/*    プロジェクト名: rei3                           */
/*    ファイル名：    rei3.c                         */
/*    内容：          int0 割り込みチェック          */
/*    日付：          2001.10.5                      */
/*    コンパイラ:     NC30WA (Ver.4.00)              */
/*    note:           OAKS16 対応                    */
/*    作成者：        OAKS16KIT support              */
/****************************************************/

/* インクルードファイル */
#include   <oaks_sfr.h>              /* OAKS16 用定義ファイル */

/* プロトタイプ宣言 */
void main(void);
void int0(void);
#pragma    INTERRUPT  int0

/* マクロ定義 */
#define PORTIN  0x00
#define    PORTOUT    0xff

/*---------------------------------------------------------
/* 関数名  :main ()
/* 機能        :LED の初期設定をし、消灯させたまま割り込み
/*---------------------------------------------------------
void main( void ){

     p7=0xff;              /* LED に消灯データ出力 */
     pd7=PORTOUT;          /* ポート 7 を出力に設定 */

     int0ic=0x07;          /* int0 の割り込み要因設定（立下りエッジ、優先レベル 7 ）*/

     _asm("¥tFSET I");     /* 割り込み許可 */

     for(;;){              /* 何も処理を行なわず、割り込み
     }

}
/*------------------------------------------------------------
/* 関数名  :int0 ()
/* 機能        :int0 割り込み関数                                */
/*         :割り込みが発生したら LED に点灯データを出力する。       */
/*-------------------------------------------------------------------------
void int0(void){           /* 割り込みプログラム */
     p7=0x00;              /* LED に点灯データを出力 */
}
```

INT0 の割り込み関数の定義：
#pragma でレジスタの退避、復帰、割り込みからのリターンをコンパイラが自動的にコード生成する（記述しないで済む）

割り込み制御レジスタの設定。割り込み要因レジスタは、リセット時に片エッジ選択になっているので、変更しない場合は、設定しなくてもよい。

割り込み許可：この命令は、C 言語では記述できないので、インラインアセンブラ記述をする。

割り込みプログラム

3.3.2. タイマを使った割り込み

マイコン制御で時間を計測する為にはタイマを使用します。タイマの基本機能は基準になるクロックやパルスをカウントすることです。OAKS16 に搭載されている M30620FCAFP は、16 ビットタイマを 11 本内蔵しています。11 本のタイマは、持っている機能によってタイマ A（5 本）とタイマ B（6 本）の 2 種類に分類できます。タイマの詳しい機能はマニュアルを参照して下さい。これらのタイマは様々な機能を持っていますがここでは、タイマ A0 を基本機能であるタイマモードで使用して、一定時間ごとの割り込みをかける方法を説明します。

タイマ A0 の構成

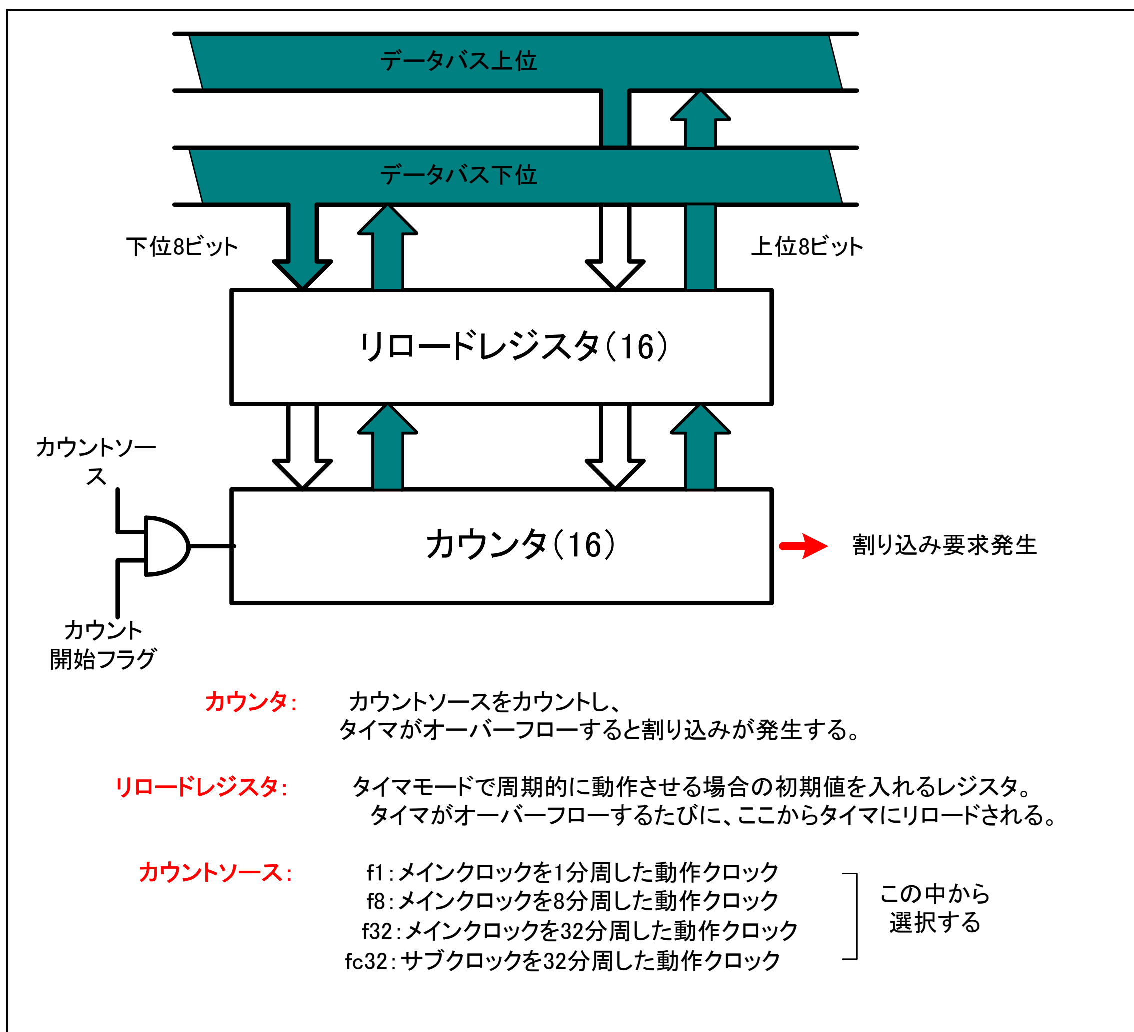


図 3.19

タイマ A0 関連レジスタ

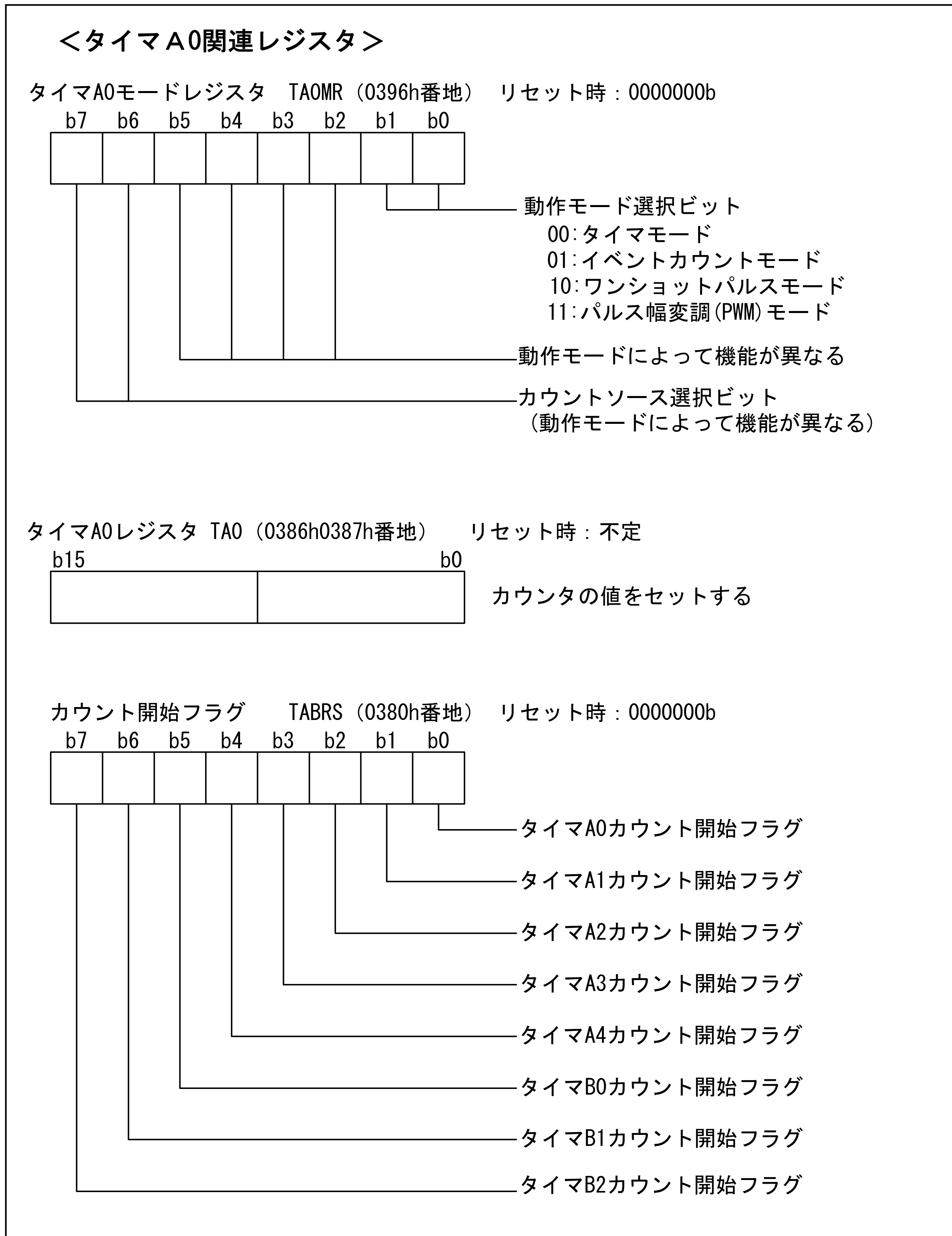


図 3.20

M16C のタイマは、様々な機能を持っているため、関連レジスタの設定が難しくなっています。そのため、ユーザーズマニュアル内では、使用する条件ごとに設定の説明がなされています。本テキストでも、タイマ割り込みを使用するために必要な設定に限定して説明します。その他の使用方法については、ユーザーズマニュアルを参照して下さい。

設定内容

・カウントソース f32（メインクロックを 32 分周したクロック）

↑一秒の計測をしたいのでカウントソースの周期の最大のもの

・パルス出力なし

・ゲート機能なし

＜タイマＡ0を使用するための設定＞

タイマA0モードレジスタ　TA0MR（0396h番地）　リセット時：0000000b

b7	b6	b5	b4	b3	b2	b1	b0
		0	0	0	0	0	0

b1 b0：動作モード選択ビット
00:タイマモード

b2：パルス出力機能選択ビット
0:パルス出力なし

b4 b3：ゲート機能選択ビット

b4	b3	
0	0	ゲート機能なし（TA0IN端子は通常のポート端子となる）
0	1	

b5：タイマモードでは“0”を設定する

b7 b6：カウントソース選択ビット

b7	b6		カウントソースの周期 f(XIN):16MHzの場合
0	0	f1	62.5ns
0	1	f8	500ns
1	0	f32	2μs
1	1	fC32	OAKS16ではサブクロックを接続していないので選択できない

タイマA0レジスタ TA0（0386h0387h番地）　リセット時：不定

b15	b0

カウンタの値をセットする
0000hからFFFFhを設定可能

カウント開始フラグ　　TABRS（0380h番地）　リセット時：0000000b

b7	b6	b5	b4	b3	b2	b1	b0

b0：タイマA0カウント開始フラグ
0:カウント停止
1:カウント開始

b7～b1：タイマA1～B2カウント開始フラグ

図 3.21

仕様

LED1～8 を 1 秒ごとに 2 進数でカウントアップさせる。(例題 4)

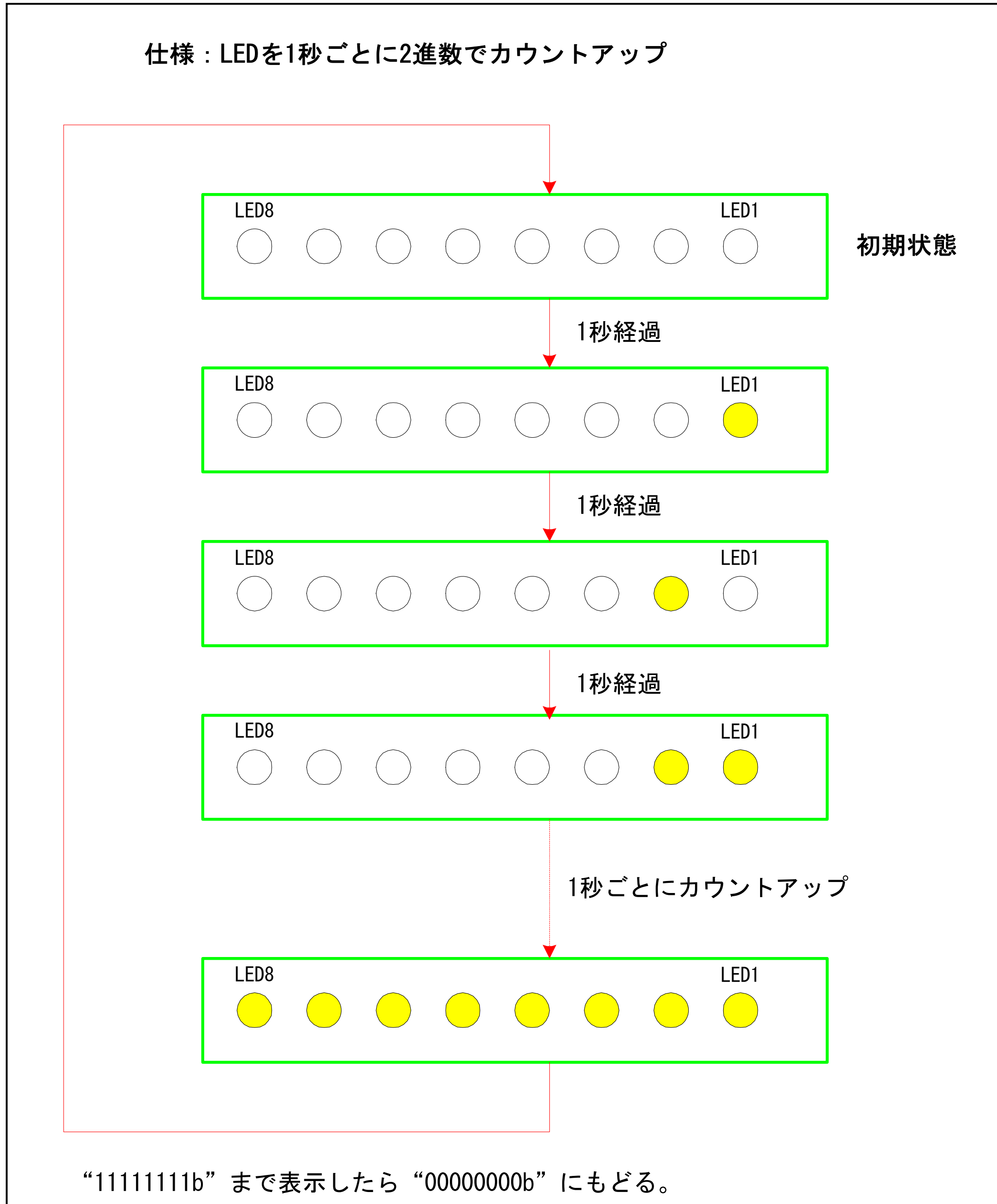


図 3.22

考え方

タイマ A0 を使用して、カウントソースを作る。

1 秒計時

タイマ A0 のカウントソースを f32 とすると 1 カウントは 1/(16MHz/32)=2μS

計時するには 1/2μs=500000 であるが 16 ビットカウンタなので 500000 回は設定できない。

1 秒=100ms×10 回とすると 100ms=2μS×50000 回
100ms の割り込みをかけて、これを 10 回カウントすればよいことになる。

フローチャート

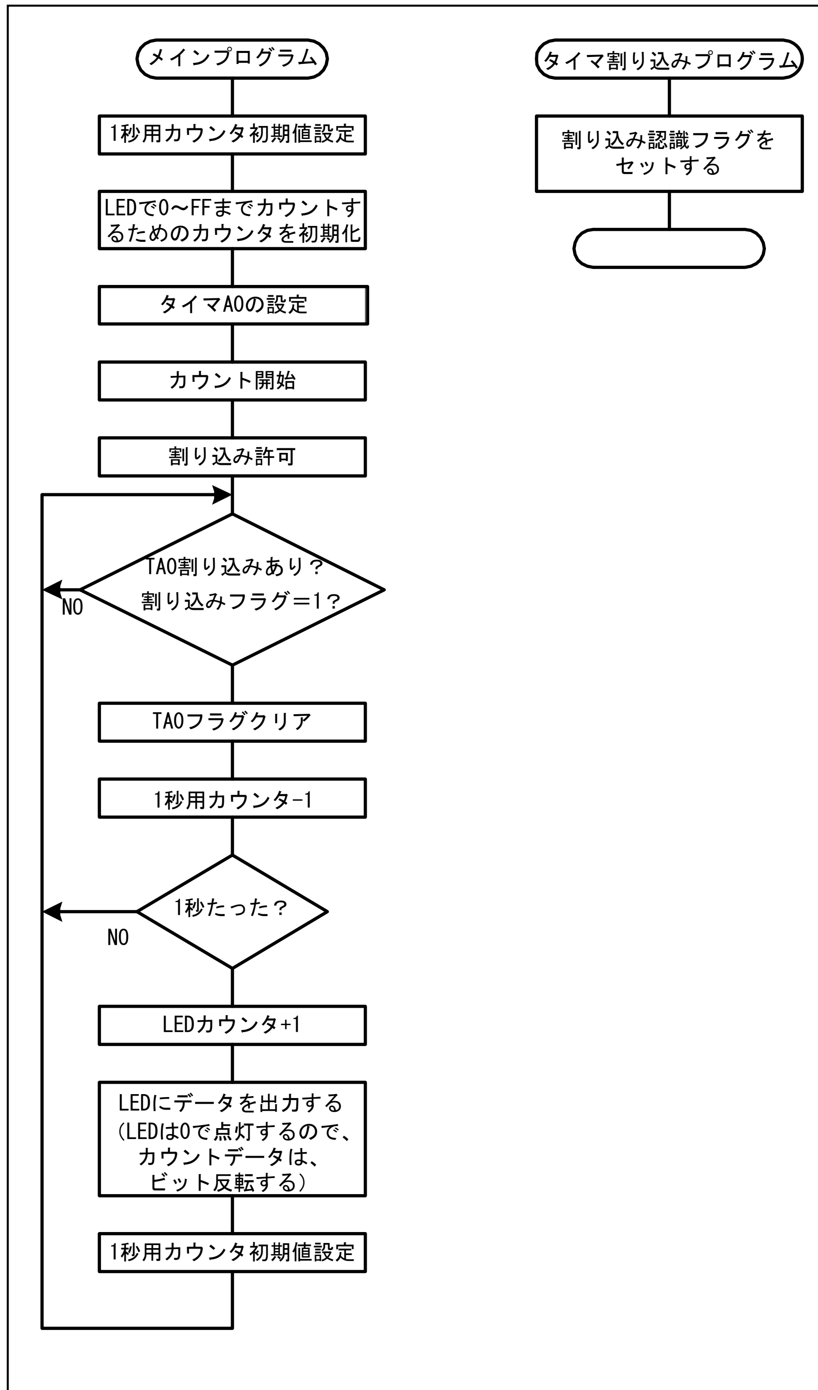


図 3.23

ファイルの分割

スタートアッププログラム：o_srt0. a30+o_sec30. inc

メインプログラムファイル：rei3. c＋OAKS16. h

ファイルの変更

③ o_sect30. inc の割り込みベクタのアドレスを設定する。

④ main プログラムに割り込みプログラムを記述する。

リスト

スタートアッププログラム（o_ncr0. a30）の変更は INT0 割り込みのときと同じなので省略します。

o_sect30. a30(OAKS16 用セクション定義ファイル)の変更も、基本的には INT0 のときと同じですが、INT0 は使用しないので,]ベクタの内容をダミーに戻し、TA0 のベクタに、TA0 割り込みプログラムの先頭アドレスをセットします。

```
;---------------------------------------------------------------
; variable vector section
;---------------------------------------------------------------
    .section   vector          ; variable vector table
    .org  VECTOR_ADR
;
    .lword     dummy_int       ; vector 0 (BRK)
    .org  (VECTOR_ADR +44)
    .lword     dummy_int       ; DMA0 (for user)
    .lword     dummy_int       ; DMA1 2 (for user)
    .lword     dummy_int       ; input key (for user)
    .lword     dummy_int       ; AD Convert (for user)
    .org  (VECTOR_ADR +68)
    .lword     dummy_int       ; uart0 trance (for user)
    .lword     dummy_int       ; uart0 receive (for user)
    .lword     0fcb6bh              ; uart1 trance (for user)
    .lword     0fcb6bh              ; uart1 receive (for user)

    .glb _ta0int
    .lword     _ta0int              ; TIMER A0 (for user)
    .lword     dummy_int       ; TIMER A1 (for user)
    .lword     dummy_int       ; TIMER A2 (for user)
    .lword     dummy_int       ; TIMER A3 (for user)
    .lword      dummy_int      ; TIMER A4 (for user) (vector 25)
    .lword     dummy_int       ; TIMER B0 (for user) (vector 26)
    .lword     dummy_int       ; TIMER B1 (for user) (vector 27)
    .lword     dummy_int       ; TIMER B2 (for user) (vector 28)
    .lword     dummy_int       ; INT0 (for user) (vector 29)
    .lword     dummy_int       ; INT1 (for user) (vector 30)
    .lword     dummy_int       ; INT2 (for user) (vector 31)
```

KD30 が使用するので必ずこのように記述する。

タイマ A0 割り込みプログラムの先頭番地設定

リスト（1/2）

```
/****************************************************/
/*   プロジェクト名: rei4                            */
/*   ファイル名 :    rei4.c                          */
/*   内容 :          LED のカウントアップ            */
/*   日付 :          2001.10.5                       */
/*   コンパイラ:     NC30WA (Ver.4.00)               */
/*   note:           OAKS16 対応                     */
/*   作成者 :        OAKS16KIT support               */
/****************************************************/

/* インクルードファイル */
#include  <oaks_sfr.h>             /* OAKS16 用定義ファイル */

/* プロトタイプ宣言 */
void main(void);              /* メイン関数  */
void ta0int();                /* 割込み関数  */
#pragma   INTERRUPT  ta0int

/*   マクロ定義*/
#define   PORTOUT   0xff               /* 出力ポート設定データ */

#define   CNT_TA0        50000-1            /* タイマ A0 カウンタ値    */
#define   CNT1S_TA0  10                /* タイマ A0 1 秒カウンタ値*/

/* 変数の宣言 */

static    char ta0flag;                     /* ta0 割り込みフラグ */
static    char cnt1s;                            /* 1 秒用カウンタ */
static    char ledcnt;                           /* led 表示用カウンタ */
/*-----------------------------------------------------------------------------------------------
----------*/
/* 関数名  :main ()                                                     */
/* 機能    : 1 秒ごとに LED1 から LED8 までを使って二進数カウントアップ */
/*         : 1 秒計時 (タイマ A0)                                       */
/*         :    タイマ A0 のカウントソースを f32 とする                 */
/*         :    1 カウントは 1/(16MHz/32)=2μS                           */
/*         :    計時するには 1/2μs=500000 であるが                      */
/*         :    16 ビットカウンタなので 500000 回は設定できない         */
/*         :    1 秒=100ms*10 回とすると                                */
/*         :    100ms=2μS*50000 回                                      */
/*-----------------------------------------------------------------------------------------------
-----------*/

void main(void)
{
     cnt1s=CNT1S_TA0;                    /* 1 秒用カウンタ初期化(10)              */
     ledcnt=0x00;                        /* led カウンタ初期化 */

     p7=ledcnt^0xff;                /* LED データ表示準備                   */
     pd7=PORTOUT;                        /* ポート 7 出力設定                     */

     udf = 0x00;                              /* ダウンカウント設定                     */
     ta0mr = 0x80;                       /* タイマモード クロック : 1/32      */
     ta0 = CNT_TA0;                      /* タイマ値の初期化                      */
     ta0ic = 0x06;                       /* 割り込みレベルの設定                  */
     tabsr = 0x01;                       /* カウント開始                          */

     _asm( "¥tFSET   I");           /* 割り込み許可                          */
```

リスト（2/2)

```
     while(1){
          if(ta0flag==1){
               ta0flag=0;                    /* ta0flag をクリア */
               cnt1s--;                      /* 1 秒用カウンタ-1  */
               if(cnt1s==0){                 /* 1 秒用カウンタが 0 なら（もし 1 秒たったら） */
                    ledcnt++;                     /* LED カウンタ+1     */
                    p7=ledcnt^0xff;
                    cnt1s = CNT1S_TA0;            /* 1 秒用カウンタ初期化(10)     */
               }
          }
     }

}

/*--------------------------------------------------------------------*/
/* 関数名 :ta0int ()                                     */
/* 機能   :ta0 割り込み関数                          */
/*        :割り込みのたびに ta0flag をセットする。*/
/*---------------------------------------------------------------------*/

void ta0int()
{
     ta0flag=1;                              /* ta0flag をセット */
}
```

3.4.LCDモジュールの制御

LCD モジュールとは、LCD 表示器に LCD コントローラが予め組み込まれた表示用の部品です。OAKS16 の LCD ボードで使用している SC1602BS は広く一般に使用されておりますが、あまり詳しい資料がありません。そこで、本テキストでは、LCD モジュールの使い方を含めて、プログラムの記述を説明していきます。

3.4.1.LCDモジュールの構成

以下に LCD モジュール（SC1602BS）の構成を示します。

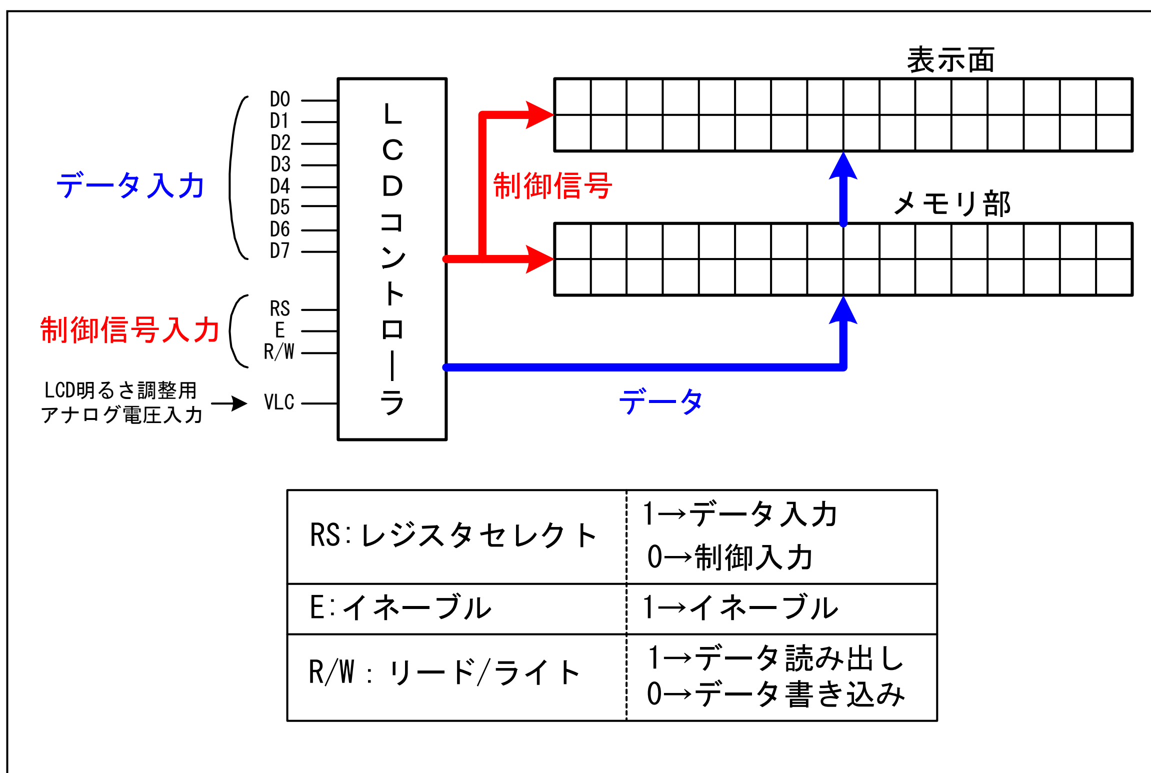


図 3.24

LCD モジュールに表示データを書き込むと、LCD モジュール内部で文字表示位置に対応するメモリ（DDRAM：表示データ RAM）に書き込まれます。それが書き改められない限りは、内部の LCD コントローラの制御により自動的に表示され続けます。

3.1.2.OAKS16LCDBOARDの配線

LCD モジュール（SC1602BS）は、データ線を 8 本持っていますが、4 本だけ接続して使用することができます。マイクロコンピュータで制御する場合、制御に使用するポートはできるだけ少ない方が好まれるので、4 本で制御される場合が多いようです。OAKS16 でも、4 本で制御するように配線してありますので、まず、配線を確認してください。

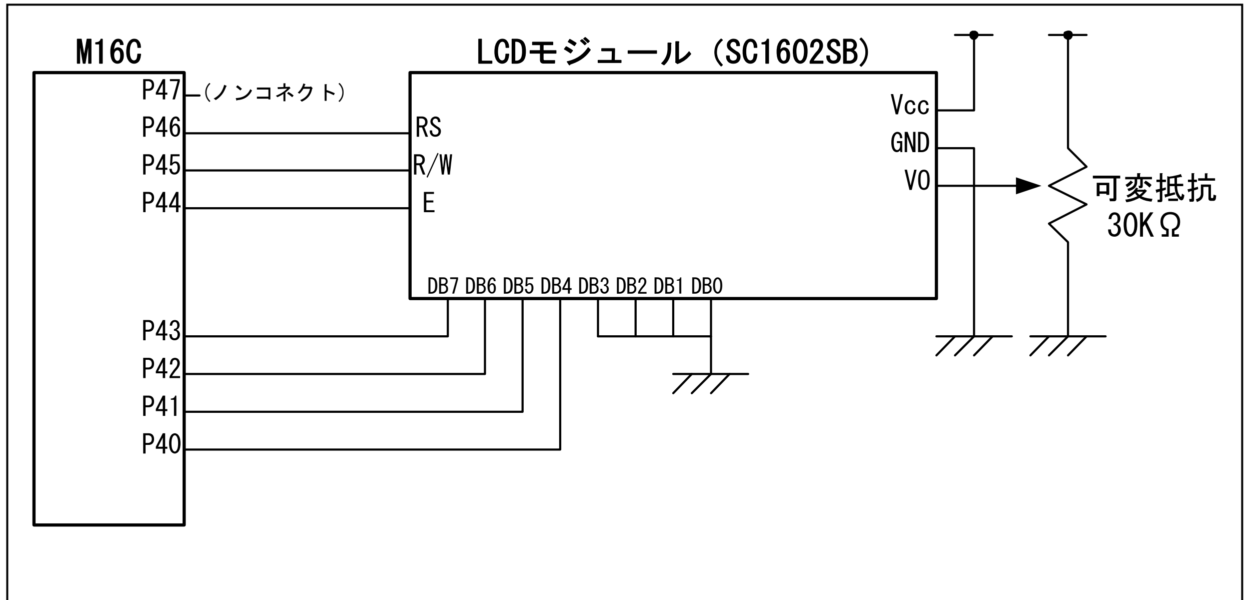


図 3.25

3.1.3.LCDモジュールの初期設定

SC1602SB は、電源 ON のあと、一連の初期設定を行ないます。これは、この部品を使用する際の決まりのようなものですので、無条件で実行してください。

① リセットのためのファンクションを機械的に 8 ビット転送で 3 回行ないます。

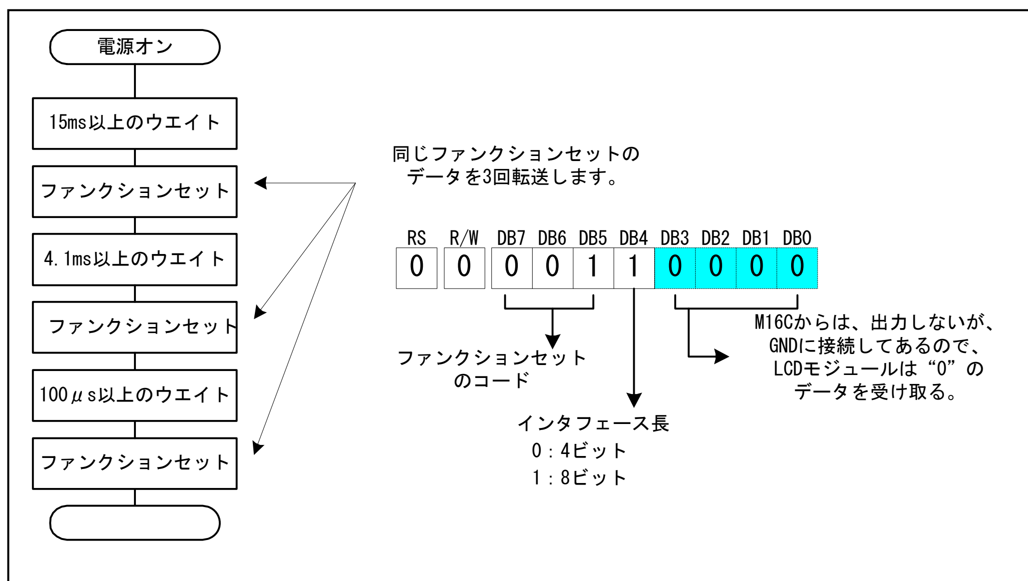


図 3.26

この段階で、実は LCD モジュールは 8 ビットでデータを受信します。OAKS16 の LCDBOARD では、LCD モジュールの下位 4 ビットは GND に接続してありますので、下位 4 ビットには "0" のデータが転送されます。ここで、3 回ファンクションデータを送ることにより、確実に LCD モジュールが 8 ビットでデータを受け取る状態にします。(この LCD モジュールは 4 ビットでデータを送る場合、時分割で 2 回に分けてデータを送受信しますので、4 ビット指定か 8 ビット指定かで動作が大きく異なります。そのため、まず標準である 8 ビットの転送を確実にします。

この後、4 ビットでデータを送るための設定、表示の初期化などの、本当の意味の初期設定が続きます。

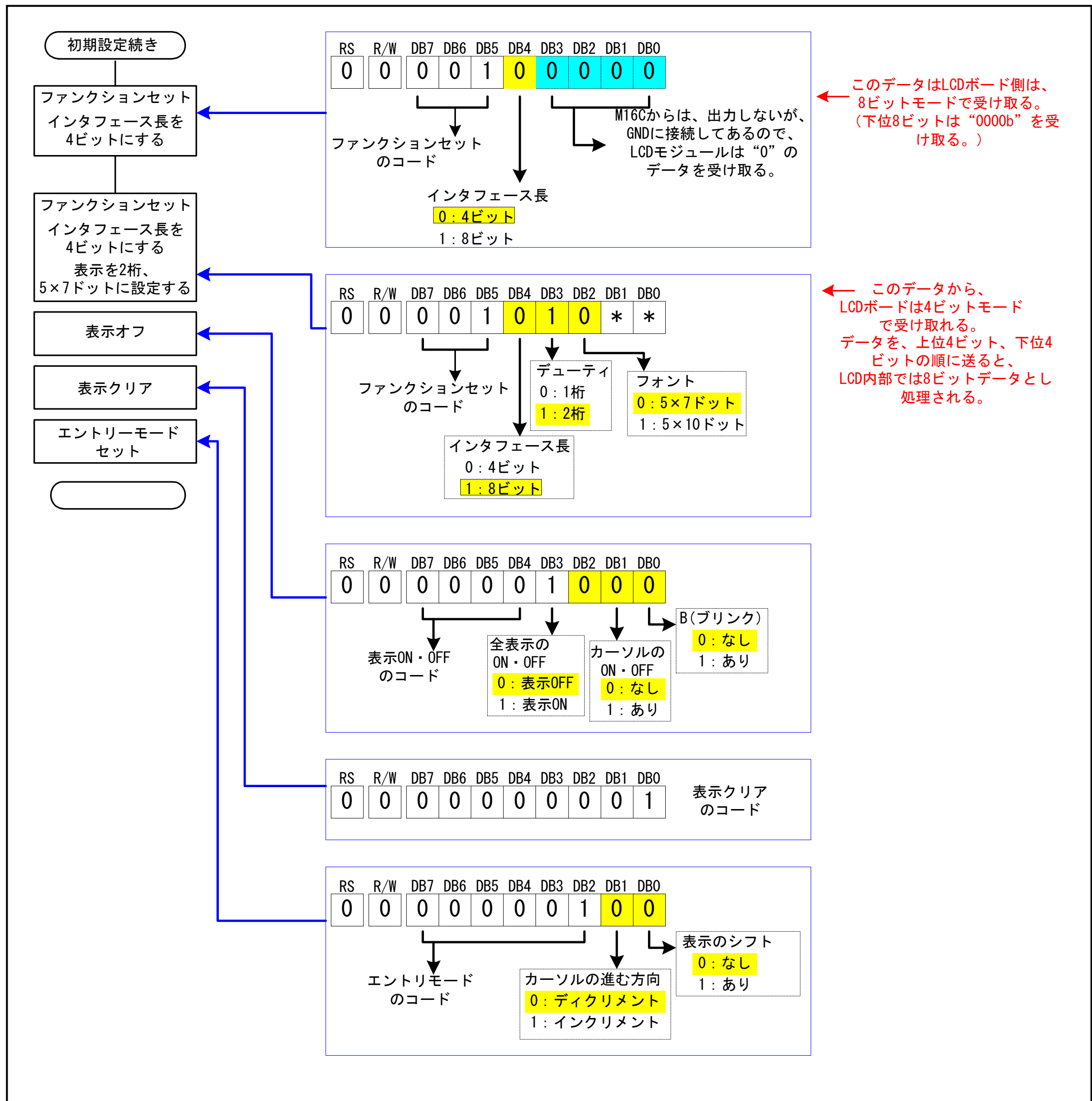


図 3.27

以上が、初期設定になります。

3.1.4.LCDモジュール制御の為の関数

ここで、LCD モジュールの制御の為の関数を作りましょう。

① ウエイト関数

今回の LCD ボードの制御には、ウエイト時間が 3 種類要求されています。
15ms、4.1ms、0.1ms のそれぞれのウエイト関数を作ります。
ここでは、チャタリング除去で使用したソフトウエイトを利用します。

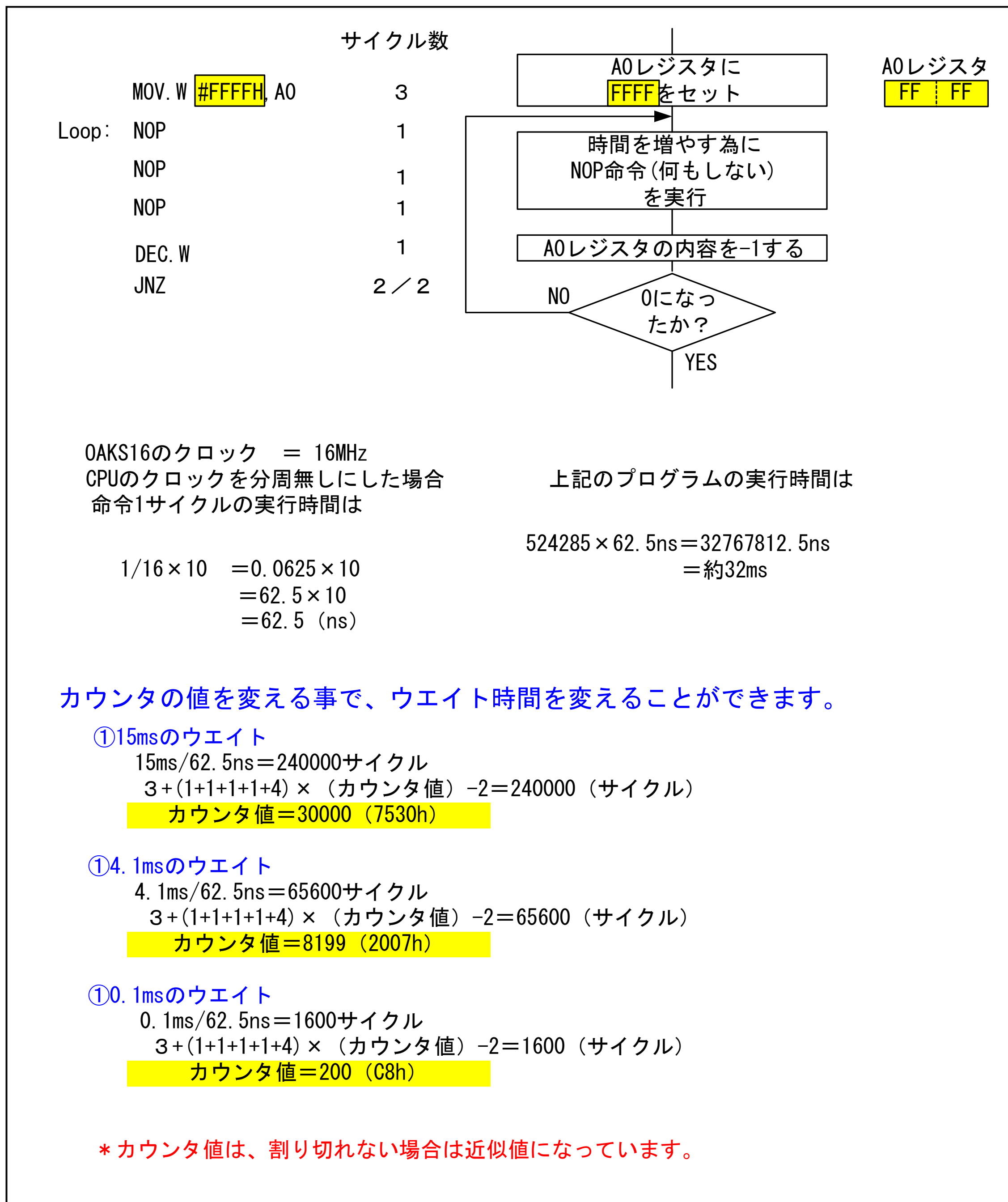
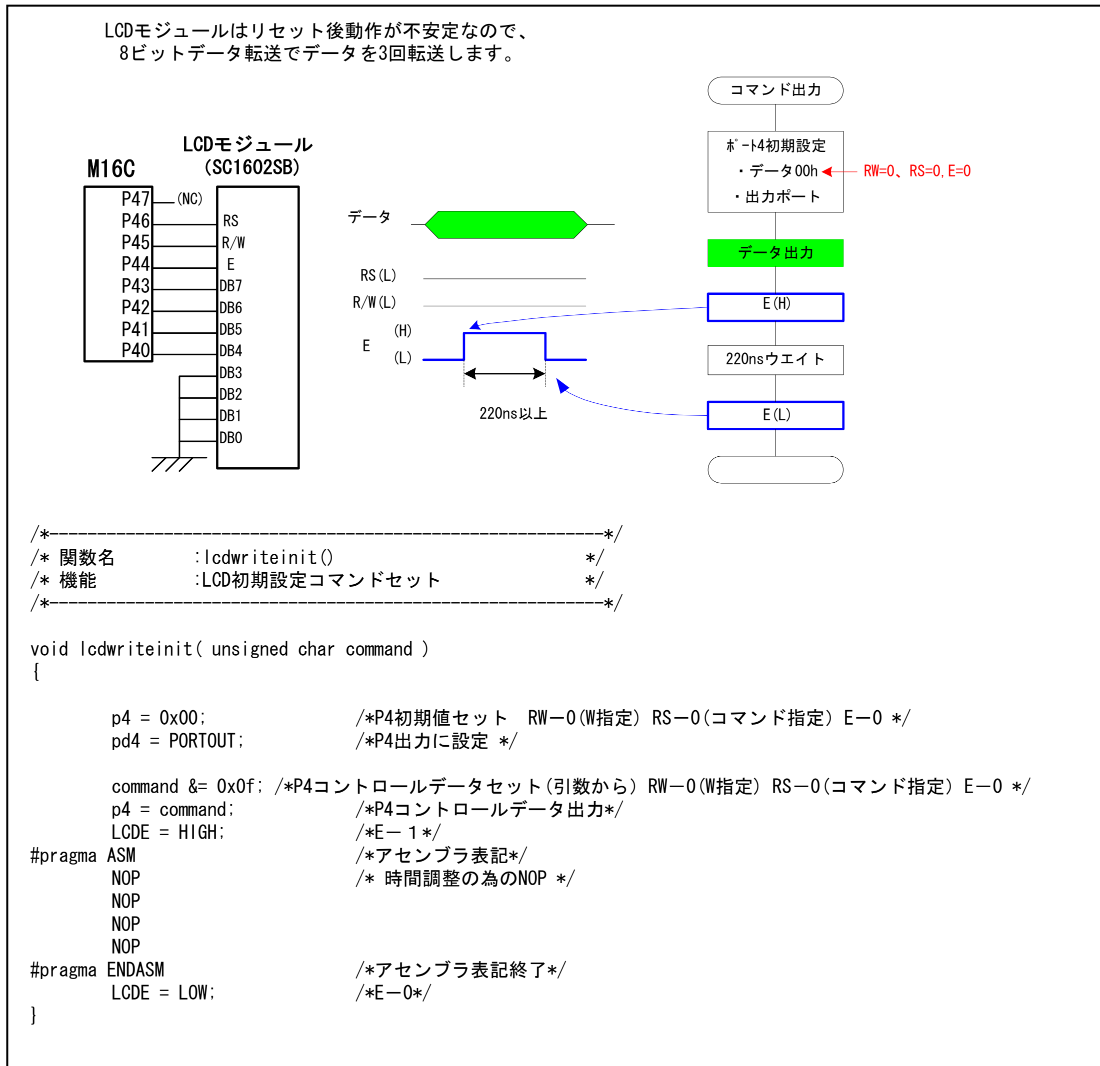


図 3.28

② ８ビットコマンド出力関数

初期設定の最初は、4 ビットでコマンドデータを送れません。そのため、8 ビットでデータを送る関数を作成します。

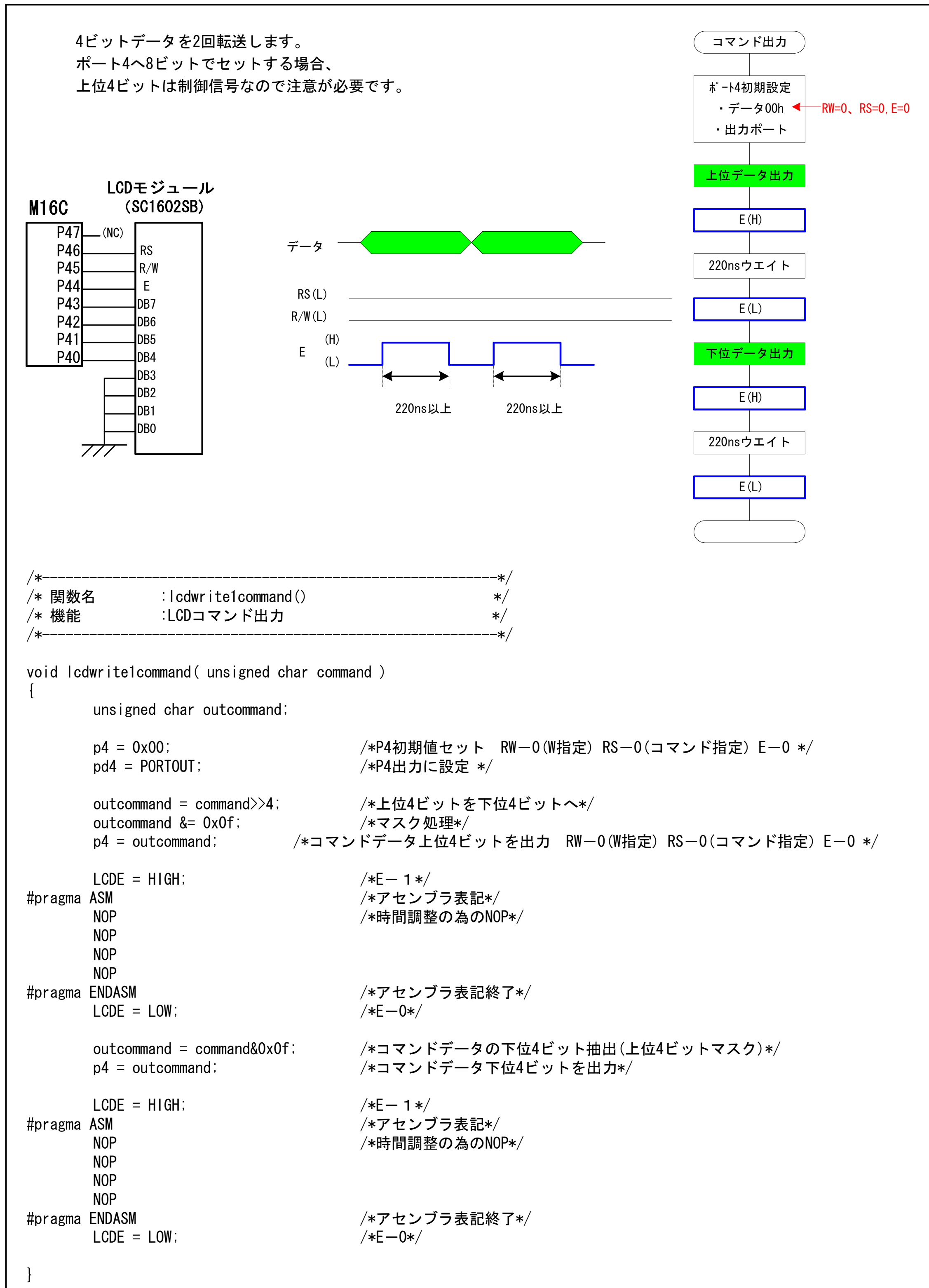



```
/*------------------------------------------------------------*/
/* 関数名          :lcdwriteinit()                          */
/* 機能            :LCD初期設定コマンドセット               */
/*------------------------------------------------------------*/

void lcdwriteinit( unsigned char command )
{

        p4 = 0x00;              /*P4初期値セット  RW—0(W指定) RS—0(コマンド指定) E—0 */
        pd4 = PORTOUT;          /*P4出力に設定 */

        command &= 0x0f; /*P4コントロールデータセット(引数から) RW—0(W指定) RS—0(コマンド指定) E—0 */
        p4 = command;           /*P4コントロールデータ出力*/
        LCDE = HIGH;            /*E— 1 */
#pragma ASM                     /*アセンブラ表記*/
        NOP                     /* 時間調整の為のNOP */
        NOP
        NOP
        NOP
#pragma ENDASM                  /*アセンブラ表記終了*/
        LCDE = LOW;             /*E—0*/
}
```

図 3.29

③ ４ビットモードでコマンドを送る関数

コマンドデータは、８ビットで意味をなしますが、４ビット転送では上位４ビット、かい４ビットの２回に分けて転送する必要があります。そのための関数です。




```
/*-----------------------------------------------------------*/
/* 関数名        :lcdwrite1command()                          */
/* 機能          :LCDコマンド出力                             */
/*-----------------------------------------------------------*/

void lcdwrite1command( unsigned char command )
{
        unsigned char outcommand;

        p4 = 0x00;                      /*P4初期値セット  RW−0(W指定) RS−0(コマンド指定) E−0 */
        pd4 = PORTOUT;                  /*P4出力に設定 */

        outcommand = command>>4;        /*上位4ビットを下位4ビットへ*/
        outcommand &= 0x0f;             /*マスク処理*/
        p4 = outcommand;        /*コマンドデータ上位4ビットを出力  RW−0(W指定) RS−0(コマンド指定) E−0 */

        LCDE = HIGH;                    /*E− 1*/
#pragma ASM                             /*アセンブラ表記*/
        NOP                             /*時間調整の為のNOP*/
        NOP
        NOP
        NOP
#pragma ENDASM                          /*アセンブラ表記終了*/
        LCDE = LOW;                     /*E−0*/

        outcommand = command&0x0f;      /*コマンドデータの下位4ビット抽出(上位4ビットマスク)*/
        p4 = outcommand;                /*コマンドデータ下位4ビットを出力*/

        LCDE = HIGH;                    /*E− 1*/
#pragma ASM                             /*アセンブラ表記*/
        NOP                             /*時間調整の為のNOP*/
        NOP
        NOP
        NOP
#pragma ENDASM                          /*アセンブラ表記終了*/
        LCDE = LOW;                     /*E−0*/

}
```

図 3.30

④4 ビットでデータを送る関数

コマンドとデータでは転送するときに RS の値を変えます。そのため、コマンド転送とは別に関数を作ります。

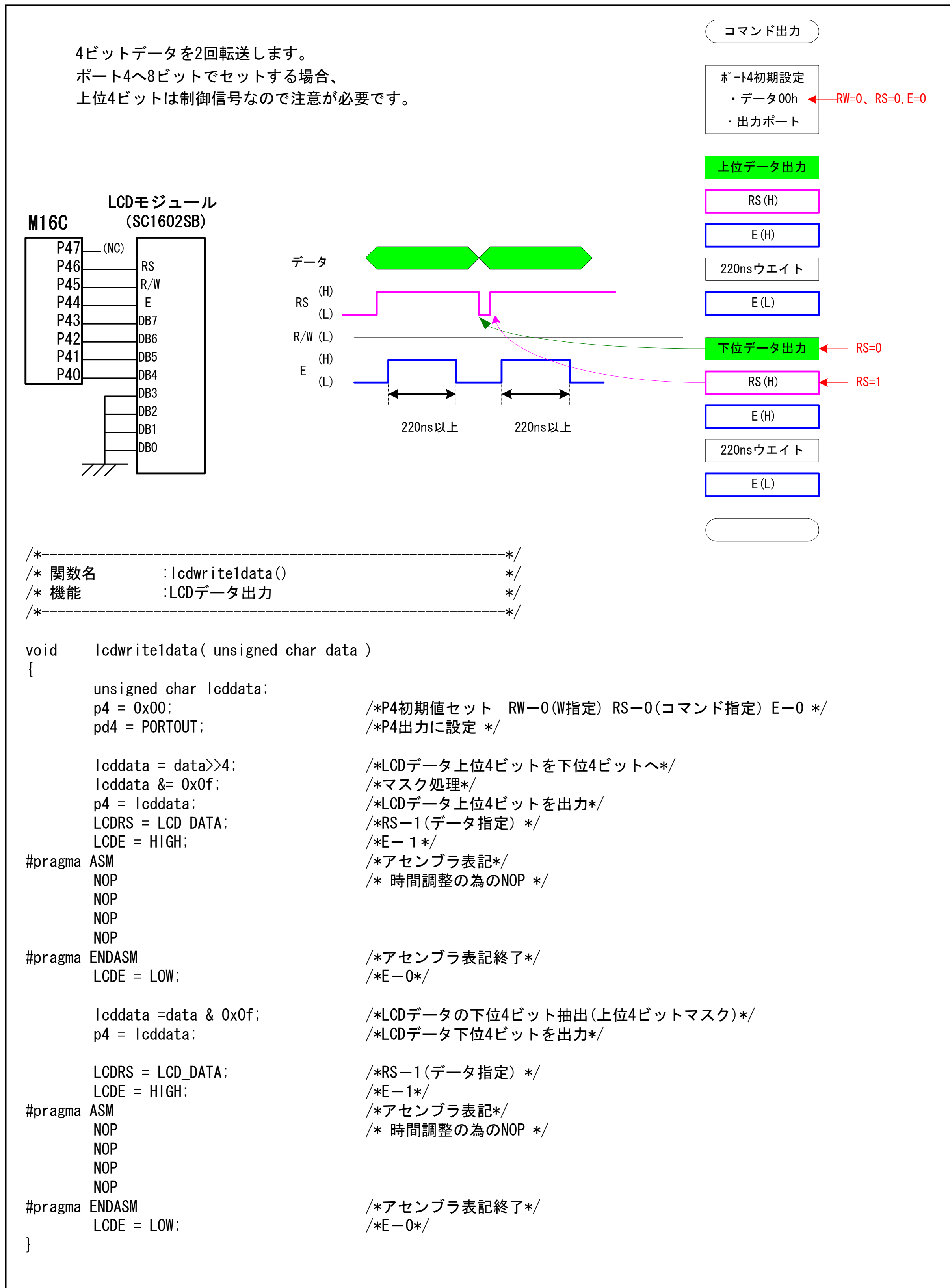


図 3.31

⑤ BUSY チェック関数

LCD モジュールは前の書き込みが終了しないと次のデータを送れません。書き込み中は、LCD モジュールの BUSY フラグが“1”になっていますので、“0”になるのを待って次の書き込みを行なう必要があります。このため、BUSY フラグを読み出す関数を作ります。

ビジーチェック関数内では、コマンドの呼び出しでビジーフラグを判断し、戻り値として、ビジーの有無（BUSY（1）・NOBUSY（0））を返します。

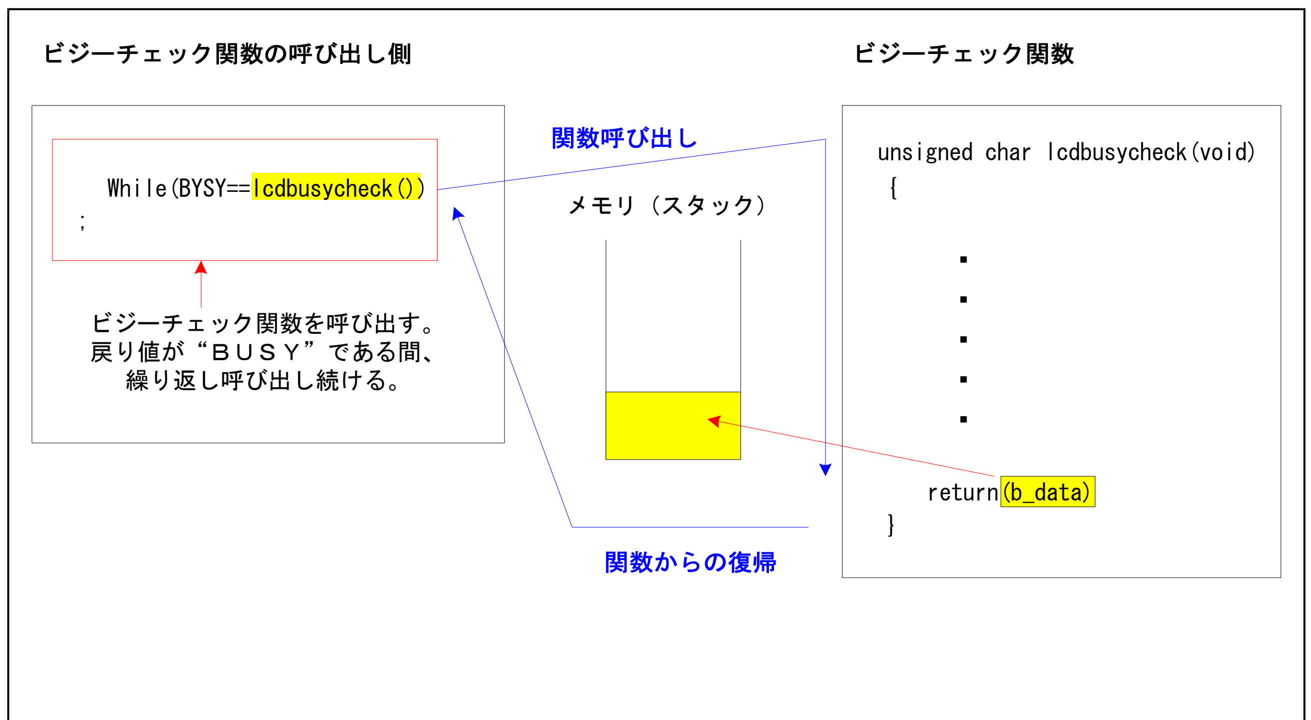
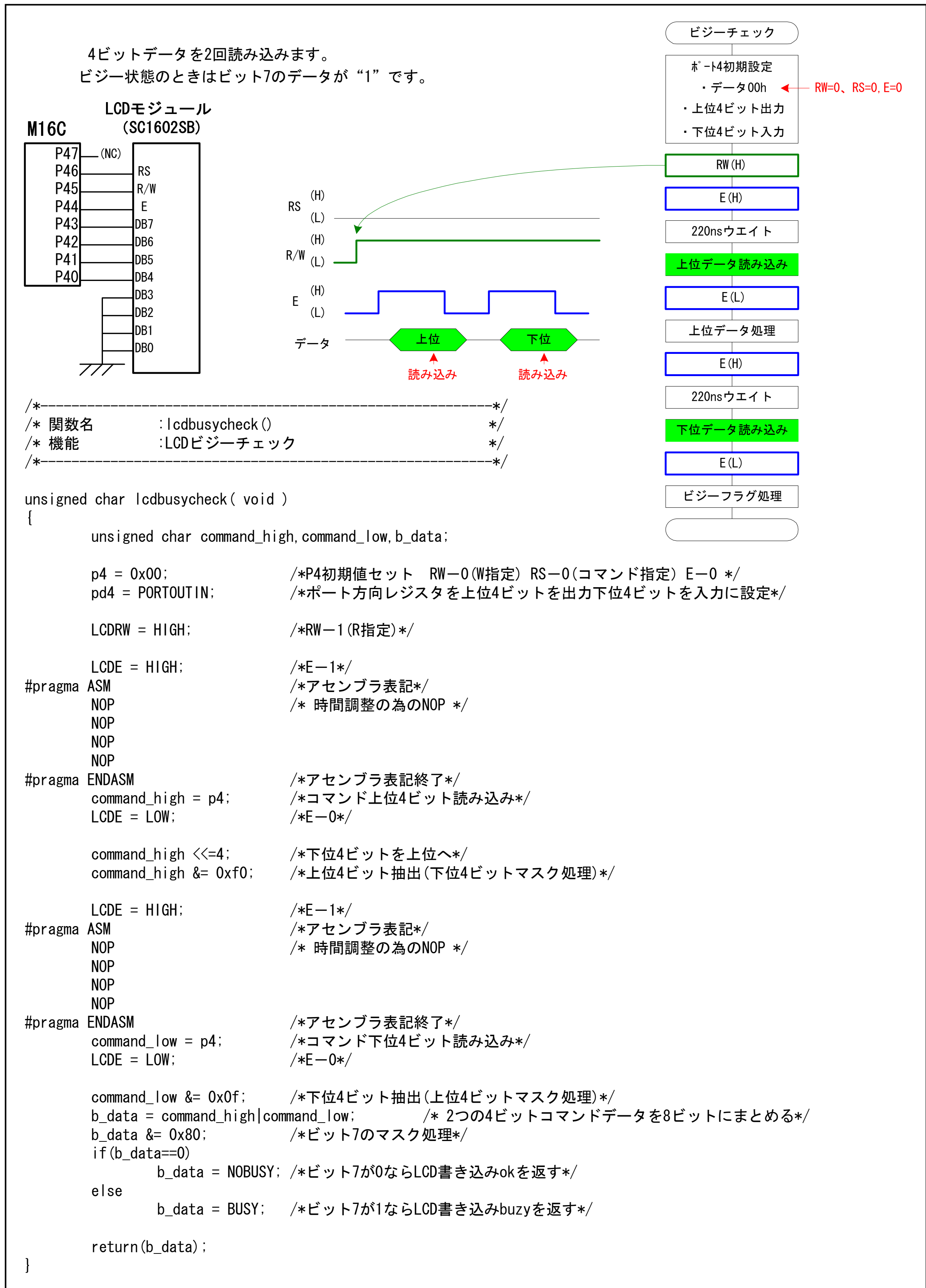


図 3.32




```
/*-----------------------------------------------------------*/
/* 関数名          :lcdbusycheck()                          */
/* 機能            :LCDビジーチェック                        */
/*-----------------------------------------------------------*/

unsigned char lcdbusycheck( void )
{
        unsigned char command_high,command_low,b_data;

        p4 = 0x00;              /*P4初期値セット  RW－0(W指定) RS－0(コマンド指定) E－0 */
        pd4 = PORTOUTIN;        /*ポート方向レジスタを上位4ビットを出力下位4ビットを入力に設定*/

        LCDRW = HIGH;           /*RW－1(R指定)*/

        LCDE = HIGH;            /*E－1*/
#pragma ASM                     /*アセンブラ表記*/
        NOP                     /* 時間調整の為のNOP */
        NOP
        NOP
        NOP
#pragma ENDASM                  /*アセンブラ表記終了*/
        command_high = p4;      /*コマンド上位4ビット読み込み*/
        LCDE = LOW;             /*E－0*/

        command_high <<=4;      /*下位4ビットを上位へ*/
        command_high &= 0xf0;   /*上位4ビット抽出(下位4ビットマスク処理)*/

        LCDE = HIGH;            /*E－1*/
#pragma ASM                     /*アセンブラ表記*/
        NOP                     /* 時間調整の為のNOP */
        NOP
        NOP
        NOP
#pragma ENDASM                  /*アセンブラ表記終了*/
        command_low = p4;       /*コマンド下位4ビット読み込み*/
        LCDE = LOW;             /*E－0*/

        command_low &= 0x0f;    /*下位4ビット抽出(上位4ビットマスク処理)*/
        b_data = command_high|command_low;      /* 2つの4ビットコマンドデータを8ビットにまとめる*/
        b_data &= 0x80;         /*ビット7のマスク処理*/
        if(b_data==0)
                b_data = NOBUSY; /*ビット7が0ならLCD書き込みokを返す*/
        else
                b_data = BUSY;   /*ビット7が1ならLCD書き込みbuzyを返す*/

        return(b_data);
}
```

図 3.33

⑥LCD モジュール初期設定の関数

LCD モジュールを初期設定する関数を作ります。LCD モジュールで決められた仕様通りに設定するだけなので、ここではリスト上で説明します。

```
/*----------------------------------------------------------*/
/* 関数名  :lcd_init()                                      */
/* 機能         :LCD 初期設定                               */
/*----------------------------------------------------------*/

void lcd_init(void)
{
     wait3();                        /*15ms ウエイト*/
     lcdwriteinit( 0x03 );                /*LCD ファンクションセット*
     wait2();                        /*4.1ms ウエイト*/
     lcdwriteinit( 0x03 );                /*LCD ファンクションセット*/
     wait1();                        /*0.1ms ウエイト*/
     lcdwriteinit( 0x03 );                /*LCD ファンクションセット*/
     wait1();
     lcdwriteinit( 0x02 );                /*LCD データを 4 ビット長に設定
     wait1();
     lcdwrite1command(0x28);         /*4bit、2 行文、5×7 ドットに設定*/
     wait1();

     /*   ここまでで 4 ビット×2 回のデータ転送ができるように設定されたの
     この後はビジーチェックをしてからデータ転送を行なう。 */

     while(BUSY == lcdbusycheck())   /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x08);         /*表示オフ*/

     while(BUSY == lcdbusycheck())   /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x01);         /*表示クリア*/

     while(BUSY == lcdbusycheck())   /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x06);         /*エントリーモード、インクリメント*/

     while(BUSY == lcdbusycheck())   /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x0c);         /*表示オン、カーソルオフ*/

}
```

ウエイトを入れて、同じコードを三回送る。(8 ビット転送のモードをきちんと動作させる。

4 ビット転送モードに変える。最初のコマンド転送では、上位 4 ビットしか読み取られていないので、2 度目のコマンド転送（4 ビット×2）で、はじめて全てのコマンドが LCD モジュールに受け取ってもらえる。

3.1.5. サンプルプログラム１

では、実際に LCD を使用したプログラムを作成してみます。

仕様

LCD に“welcome to oaks16”と表示する。

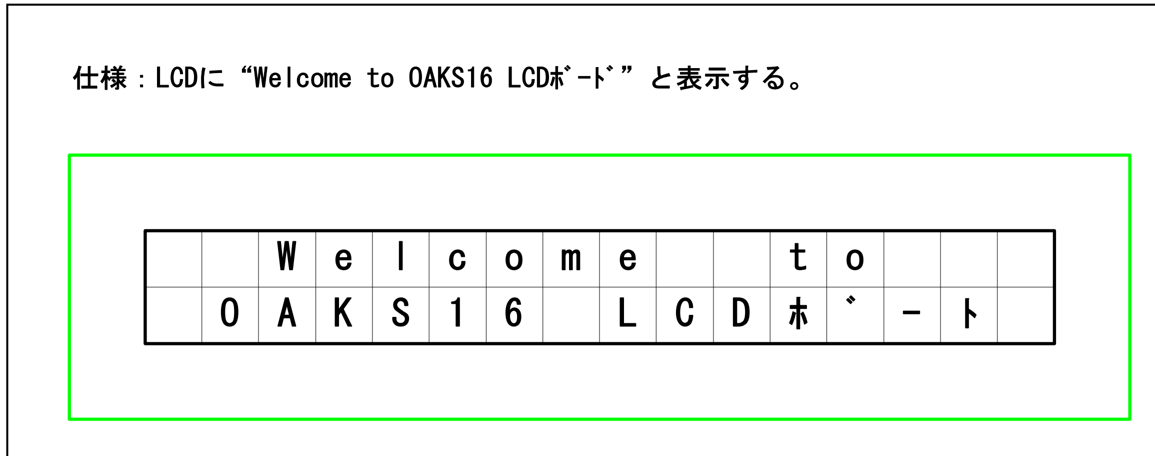


図 3.34

考え方

　ＬＣＤに表示示する為の文字列を配列を使って ROM 上に確保する。
LCD モジュールの初期化を行なった後、LCD 表示用配列から LCD モジュールの DDRAM にデータを出力する。

フローチャート

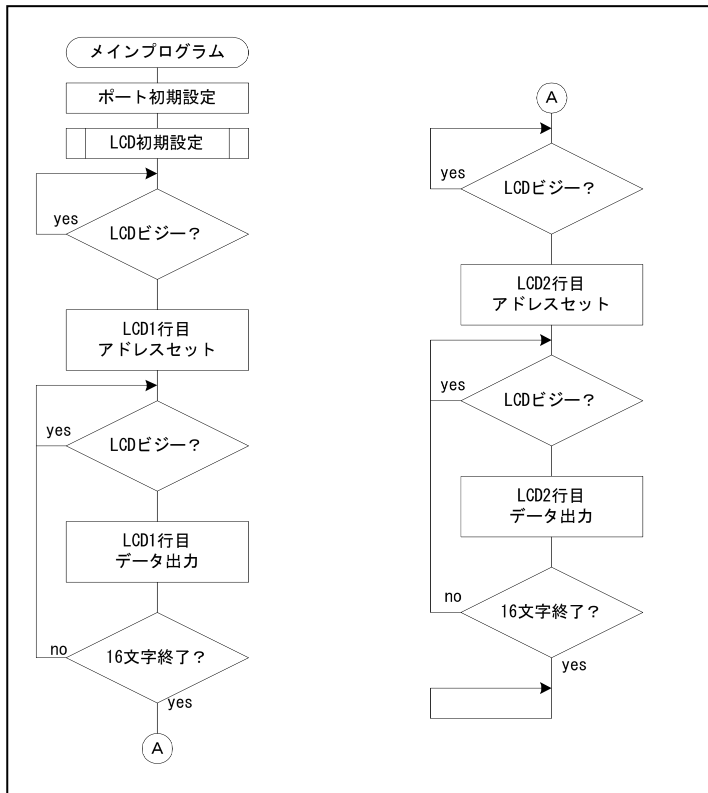


図 3.35

ファイルの分割

スタートアッププログラム：o_srt0.a30+o_sec30.inc

メインプログラムファイル：rei5.c＋OAKS16.h

ファイルの変更

スタートアッププログラムの変更無し

リスト

リスト(1/7)

```
/****************************************************/
/*   プロジェクト名: rei5                            */
/*   ファイル名 :     rei5.c                          */
/*   内容 :           LCD 表示                        */
/*   日付 :           2002.3.30                       */
/*   コンパイラ:      NC30WA (Ver.4.00)               */
/*   note:           OAKS16 対応                      */
/*   作成者 :         OAKS16KIT support               */
/****************************************************/

/* インクルードファイル */
#include   <oaks_sfr.h>                    /* OAKS16 用定義ファイル */

/* プロトタイプ宣言 */
void main(void);                                /* メイン関数 */
void lcd_init(void);                      /* LCD 初期設定 */
void lcdwriteinit( unsigned char);        /* LCD 初期設定コマンド出力 */
void lcdwrite1command( unsigned char);    /* LCD1 コマンド出力 */
void lcdwrite1data( unsigned char ); /* LCD1 データ出力 */
void wait1( void );                             /* 0.1ms ウエイト */
void wait2( void );                             /* 4.1ms ウエイト */
void wait3( void );                             /* 15ms ウエイト */
unsigned char lcdbusycheck( void );       /* LCD ビジーチェック */
/*   マクロ定義*/
#define PORTIN  0x00 /*ポート方向レジスタを入力に設定する為のデータ*/
#define   PORTOUT    0xff /*ポート方向レジスタを出力に設定する為のデータ*/
#define   PORTOUTIN 0xf0/*ポート方向レジスタを上位 4 ビットを出力*/
                    /*下位 4 ビットを入力に設定する為のデータ*/
#define   LCDRS p4_6          /* RS 端子 */
#define   LCDRW p4_5          /* RW 端子 */
#define   LCDE  p4_4          /* E 端子 */
#define   HIGH  1                  /* 端子出力“H” */
#define   LOW        0                  /* 端子出力“L” */
#define   LCD_COMMAND         0         /* RS-command 指定 */
#define   LCD_DATA       1         /* RS-data 指定 */
#define   BUSY  1                  /* LCD 書き込み buzy */
#define   NOBUSY     0                  /* LCD 書き込み OK */

/* 変数の宣言 */

const static char pat[2][16]={{' ',' ','W','e','l','c','o','m','e',' ',' ','t','o',' ',' ',' '},
                        {' ','O','A','K','S','1','6',' ','L','C','D','ﾎ','ﾞ','-','ﾄ','ﾞ'}};
                        /*LCD 表示用データ */
```

リスト(2/7)

```
/*------------------------------------------------------------*/
/* 関数名  :main ()                                            */
/* 機能    :LCDに'Welcom to OAKS16 LCDﾎﾞｰﾄﾞ'を表示する        */
/*------------------------------------------------------------*/

void main(void)
{
     int  i;                    /*ループカウンタ*/
/*** 初期設定  ***/
     p4 =  0x00;                /*ポート4(LCD)初期値設定*/
     pd4 = PORTOUT;        /*ポート4方向を出力に設定*/
     lcd_init();                     /*LCD初期設定*/

/*** LCD表示   ***/

     while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
     ;
     lcdwrite1command ( 0x80 );      /*1行目先頭アドレスセット*/

     for(i=0;i<16;i++)
     {
          while(BUSY == lcdbusycheck())/*LCDビジーチェック*/
          ;
          lcdwrite1data(pat[0][i]); /*LCDデータ1行目出力*/
     }

     while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
     ;
     lcdwrite1command ( 0xc0 );      /*2行目先頭アドレスセット*/

     for(i=0;i<16;i++)
     {
          while(BUSY == lcdbusycheck())/*LCDビジーチェック*/
          ;

          lcdwrite1data(pat[1][i]); /*LCDデータ2行目出力*/
     }

     while(1);                            /*無限ループ*/
}
```

リスト(3/7)

```
/*--------------------------------------------*/
/* 関数名  :lcd_init()                        */
/* 機能         :LCD 初期設定                 */
/*--------------------------------------------*/

void lcd_init(void)
{
     wait3();                           /*15ms ウエイト*/
     lcdwriteinit( 0x03 );              /*LCD ファンクションセット*/
     wait2();                           /*4.1ms ウエイト*/
     lcdwriteinit( 0x03 );              /*LCD ファンクションセット*/
     wait1();                           /*0.1ms ウエイト*/
     lcdwriteinit( 0x03 );              /*LCD ファンクションセット*/
     wait1();
     lcdwriteinit( 0x02 );              /*LCD データを 4 ビット長に設定*/
     wait1();
     lcdwrite1command(0x28);            /*4bit、2 行文、5×7 ドットに設定*/
     wait1();

     /*   ここまでで 4 ビット×2 回のデータ転送ができるように設定されたので
     この後はビジーチェックをしてからデータ転送を行なう。 */

     while(BUSY == lcdbusycheck())  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x08);            /*表示オフ*/

     while(BUSY == lcdbusycheck())  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x01);            /*表示クリア*/

     while(BUSY == lcdbusycheck())  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x06);            /*エントリーモード、インクリメント*/

     while(BUSY == lcdbusycheck())  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x0c);            /*表示オン、カーソルオフ*/

}
```

リスト(4/7)

```
/*--------------------------------------*/
/* 関数名 :lcdwriteinit()               */
/* 機能   :LCD 初期設定コマンドセット    */
/*----------------------------------------*/

void lcdwriteinit( unsigned char command )
{

     p4 = 0x00;               /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;           /*P4 出力に設定 */

     command &= 0x0f;         /*P4 コントロールデータセット(引数から) RW－0(W 指定) RS－0(コマンド指定) E
－0 */
     p4 = command;            /*P4 コントロールデータ出力*/
     LCDE = HIGH;             /*E－１*/
#pragma ASM                        /*アセンブラ表記*/
     NOP                           /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                /*アセンブラ表記終了*/
     LCDE = LOW;                   /*E－0*/
}
/*--------------------------------------*/
/* 関数名 :lcdwrite1command()           */
/* 機能   :LCD コマンド出力              */
/*--------------------------------------*/

void lcdwrite1command( unsigned char command )
{
     unsigned char outcommand;

     p4 = 0x00;               /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;           /*P4 出力に設定 */

     outcommand = command>>4;/*上位 4 ビットを下位 4 ビットへ*/
     outcommand &= 0x0f;      /*マスク処理*/
     p4 = outcommand;     /*コマンドデータ上位 4 ビットを出力  RW－0(W 指定) RS－0(コマンド指定) E－0 */

     LCDE = HIGH;             /*E－１*/
#pragma ASM                        /*アセンブラ表記*/
     NOP                           /*時間調整の為の NOP*/
     NOP
     NOP
     NOP
#pragma ENDASM                /*アセンブラ表記終了*/
     LCDE = LOW;                   /*E－0*/

     outcommand = command&0x0f;/*コマンドデータの下位 4 ビット抽出(上位 4 ビットマスク)*/
     p4 = outcommand;              /*コマンドデータ下位 4 ビットを出力*/

     LCDE = HIGH;             /*E－１*/
#pragma ASM                        /*アセンブラ表記*/
     NOP                           /*時間調整の為の NOP*/
     NOP
     NOP
     NOP
#pragma ENDASM                     /*アセンブラ表記終了*/
     LCDE = LOW;                        /*E－0*/
```

リスト(5/7)

```
/*--------------------------------------------*/
/* 関数名 :lcdwrite1data()                   */
/* 機能   :LCD データ出力                     */
/*--------------------------------------------*/

void lcdwrite1data( unsigned char data )
{
     unsigned char lcddata;
     p4 = 0x00;              /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;          /*P4 出力に設定 */

     lcddata = data>>4;      /*LCD データ上位 4 ビットを下位 4 ビットへ*/
     lcddata &= 0x0f;             /*マスク処理*/
     p4 = lcddata;           /*LCD データ上位 4 ビットを出力*/
     LCDRS = LCD_DATA;       /*RS－1(データ指定) */
     LCDE = HIGH;            /*E－１*/
#pragma ASM                       /*アセンブラ表記*/
     NOP                          /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM               /*アセンブラ表記終了*/
     LCDE = LOW;                  /*E－0*/

     lcddata =data & 0x0f;   /*LCD データの下位 4 ビット抽出(上位 4 ビットマスク)*/
     p4 = lcddata;           /*LCD データ下位 4 ビットを出力*/

     LCDRS = LCD_DATA;       /*RS－1(データ指定) */
     LCDE = HIGH;            /*E－1*/
#pragma ASM                       /*アセンブラ表記*/
     NOP                          /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM               /*アセンブラ表記終了*/
     LCDE = LOW;                  /*E－0*/
}

/*----------------------------------------------------------*/
/* 関数名 :wait1()                                          */
/* 機能   :0.1ms ウエイト                                   */
/*----------------------------------------------------------*/
void wait1(void){            /* 約 0.1ms のウエイト */
#pragma ASM                       /*アセンブラ表記*/
     MOV.W     #0C8H,A0      /*カウンタ初期値セット*/
LOOP1:
     NOP
     NOP
     NOP
     DEC.W A0
     JNZ  LOOP1              /*ループ終了？*/
#pragma ENDASM               /*アセンブラ表記終了*/
}
```

リスト(6/7)

```
/*-----------------------------------------------------------*/
/* 関数名 :wait2()                       */
/* 機能   :4.1msウエイト                 */
/*---------------------------------------*/
void wait2(void){              /* 約4.1msのウエイト */
#pragma ASM                           /*アセンブラ表記*/
      MOV.W      #2007H,A0       /*カウンタ初期値セット*/
LOOP2:
      NOP
      NOP
      NOP
      DEC.WA0
      JNZ   LOOP2              /*ループ終了?*/
#pragma ENDASM                 /*アセンブラ表記終了*/
}
/*---------------------------------------*/
/* 関数名 :wait3()                       */
/* 機能        :15msウエイト             */
/*---------------------------------------*/
void wait3(void){              /* 約15msのウエイト */
#pragma ASM                           /*アセンブラ表記*/
      MOV.W      #7530H,A0       /*カウンタ初期値セット*/
LOOP3:
      NOP
      NOP
      NOP
      DEC.WA0
      JNZ   LOOP3              /*ループ終了?*/
#pragma ENDASM                 /*アセンブラ表記終了*/
}
```

リスト(7/7)

```
/*----------------------------------------*/
/* 関数名 :lcdbusycheck()                  */
/* 機能      :LCD ビジーチェック           */
/*----------------------------------------*/

unsigned char lcdbusycheck( void )
{
     unsigned char command_high,command_low,b_data;

     p4 = 0x00;                /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUTIN;          /*ポート方向レジスタを上位 4 ビットを出力下位 4 ビットを入力に設定*/

     LCDRW = HIGH;             /*RW－1(R 指定)*/

     LCDE = HIGH;              /*E－1*/
#pragma ASM                         /*アセンブラ表記*/
     NOP                            /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                 /*アセンブラ表記終了*/
     command_high = p4;        /*コマンド上位 4 ビット読み込み*/
     LCDE = LOW;                    /*E－0*/

     command_high <<=4;        /*下位 4 ビットを上位へ*/
     command_high &= 0xf0;     /*上位 4 ビット抽出(下位 4 ビットマスク処理)*/

     LCDE = HIGH;              /*E－1*/
#pragma ASM                         /*アセンブラ表記*/
     NOP                            /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                 /*アセンブラ表記終了*/
     command_low = p4;         /*コマンド下位 4 ビット読み込み*/
     LCDE = LOW;                    /*E－0*/

     command_low &= 0x0f; /*下位 4 ビット抽出(上位 4 ビットマスク処理)*/
     b_data = command_high|command_low;   /* 2 つの 4 ビットコマンドデータを 8 ビットにまとめる*/
     b_data &= 0x80;           /*ビット 7 のマスク処理*/
     if(b_data==0)
          b_data = NOBUSY;     /*ビット 7 が 0 なら LCD 書き込み ok を返す*/
     else
          b_data = BUSY;       /*ビット 7 が 1 なら LCD 書き込み buzy を返す*/

     return(b_data);
}
```

3.1.6.サンプルプログラム２

仕様

LCDに“switchdata＝＊＊h”と表示する。＊＊にはSW1からSW8までで表される16進数を表示する。データはスイッチがONで“1”、OFFで“0”とする。

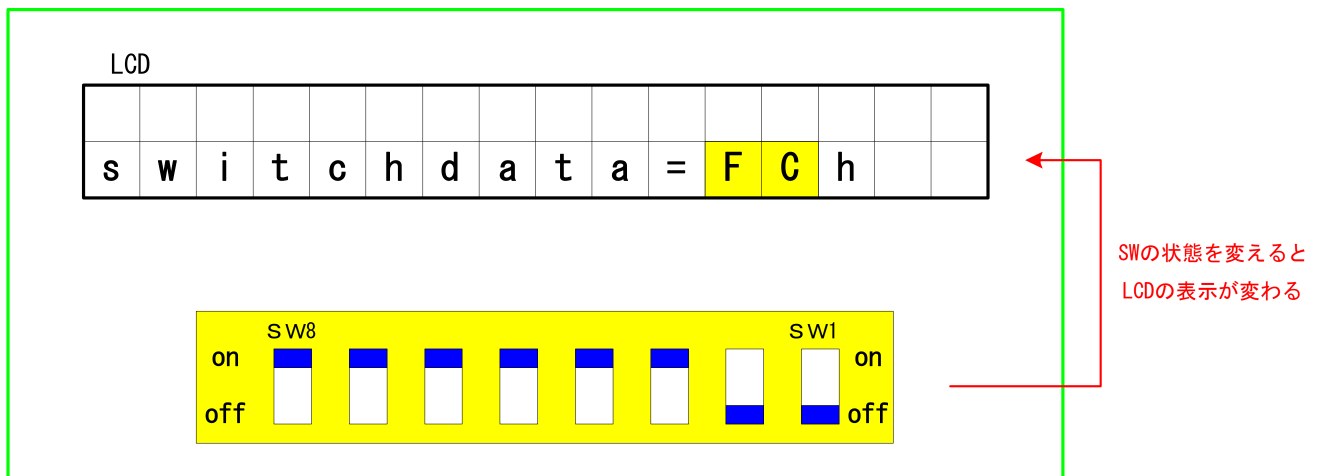


図 3.36

考え方

LCDに表示する為の文字列を配列を使ってROM上に確保する。

数値データをアスキーコードに変換する為のテーブルを用意する。

フローチャート

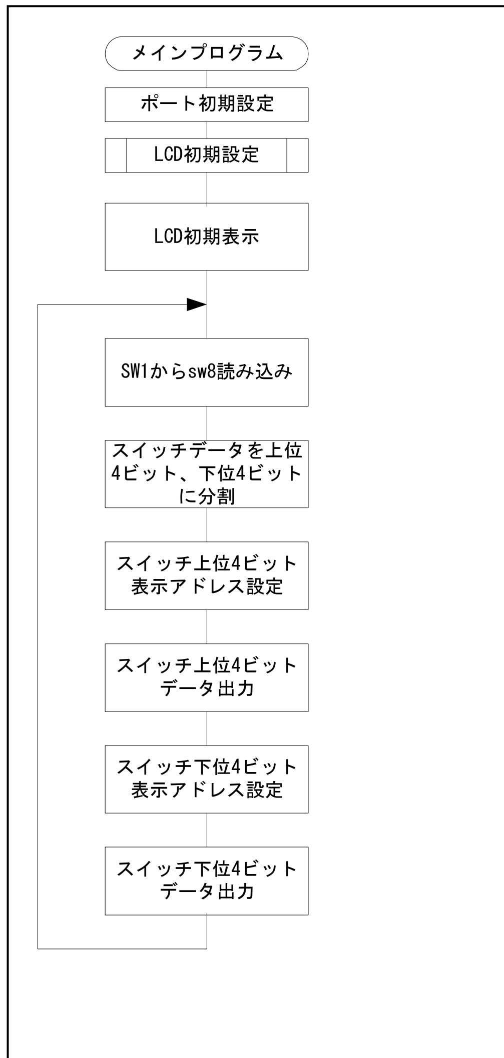


図 3.37

リスト

リスト(1/7)

```
/****************************************************/
/*  プロジェクト名: rei6                             */
/*  ファイル名 :    rei6.c                           */
/*  内容 :          swdata を LCD に表示する         */
/*  日付 :          2002.3.30                        */
/*  コンパイラ:     NC30WA (Ver.4.00)                */
/*  note:           OAKS16 対応                      */
/*  作成者 :        OAKS16KIT support                */
/****************************************************/

/* インクルードファイル */
#include  <oaks_sfr.h>                         /* OAKS16 用定義ファイル */

/* プロトタイプ宣言 */
void main(void);                                    /*メイン関数*/
void lcd_init(void);                           /*LCD 初期設定*/
void lcdwriteinit( unsigned char);        /*LCD 初期設定コマンド出力*/
void lcdwrite1command( unsigned char);    /*LCD1 コマンド出力*/
void lcdwrite1data( unsigned char );      /*LCD1 データ出力*/
unsigned char lcdbusycheck( void );            /*LCD ビジーチェック*/
void wait1( void );                            /*0.1ms ウエイト*/
void wait2( void );                            /*4.1ms ウエイト*/
void wait3( void );                            /*15ms ウエイト*/

/*  マクロ定義*/
#define PORTIN  0x00 /*ポート方向レジスタを入力に設定する為のデータ*/
#define    PORTOUT   0xff  /*ポート方向レジスタを出力に設定する為のデータ*/
#define    PORTOUTIN 0xf0/*ポート方向レジスタを上位 4 ビットを出力*/
                              /*下位 4 ビットを入力に設定する為のデータ*/

#define    LCDRS p4_6           /*RS 端子*/
#define    LCDRW p4_5           /*RW 端子*/
#define    LCDE  p4_4           /*E 端子*/
#define    HIGH  1
#define    LOW       0
#define    LCD_COMMAND          0         /*RS-command 指定*/
#define    LCD_DATA       1          /*RS-data 指定*/
#define    BUSY  1                   /*LCD 書き込み buzy*/
#define    NOBUSY    0                    /*LCD 書き込み OK*/

/* 変数の宣言 */

const static    char pat[17]={"switchdata=  h  "};        /*LCD 表示用データ*/
const static    char patdata[17]={"0123456789abcdef"};    /*パターン表示用データ*/
```

リスト(2/7)

```
/*-----------------------------------------------------------------------------------*/
/* 関数名 :main ()                                                   */
/* 機能   :LCDに'Welcom to OAKS16 LCDﾎﾞｰﾄﾞ'を表示する       */
/*-----------------------------------------------------------------------------------*/

void main(void)
{
unsigned char sw,sw_low,sw_high;     /*変数宣言(スイッチの接続されているポートのデータを入れる)*/

     int  i;                                   /*ループカウンタ*/
/*** 初期設定  ***/

     pd3=PORTIN;                               /*ポート3を入力に設定*/

     p4 =  0x00;                               /*ポート4(LCD)初期値設定*/
     pd4 = PORTOUT;                       /*ポート4方向を出力に設定*/
     lcd_init();                               /*LCD初期設定*/

/*** LCD表示  ***/
     while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
     ;
     lcdwrite1command ( 0xc0 );      /*2行目先頭アドレスセット*/

     for(i=0;i<16;i++)
     {
          while(BUSY == lcdbusycheck())/*LCDビジーチェック*/
          ;
                                               /*OAKS16 LCDﾎﾞｰﾄﾞを表示*/
          lcdwrite1data(pat[i]);          /*LCDデータ出力*/
     }
     for(;;)
     {
          sw=p3;
          sw=sw^0xff;
          sw_low=sw&0x0f;
          sw_high=(sw&0xf0);
          sw_high>>=4;

          while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
          ;
          lcdwrite1command ( 0xcb );      /*スイッチデータ上位表示アドレスセット*/

          while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
          ;
          lcdwrite1data(patdata[sw_high]);/*スイッチ上位4ビット表示データ出力*/

          while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
          ;
          lcdwrite1command ( 0xcc );      /*スイッチデータ下位表示アドレスセット*/

          while(BUSY == lcdbusycheck())   /*LCDビジーチェック*/
          ;
          lcdwrite1data(patdata[sw_low]);/*スイッチ下位4ビット表示データ出力*/
     }
}
```

リスト(3/7)

```
/*------------------------------------------------------------*/
/* 関数名 :lcd_init()                          */
/* 機能   :LCD 初期設定                         */
/*------------------------------------------------------------*/

void lcd_init(void)
{
     wait3();                                            /*15ms ウエイト*/
     lcdwriteinit( 0x03 );                               /*LCD ファンクションセット*/
     wait2();                                            /*4.1ms ウエイト*/
     lcdwriteinit( 0x03 );                               /*LCD ファンクションセット*/
     wait1();                                            /*0.1ms ウエイト*/
     lcdwriteinit( 0x03 );                               /*LCD ファンクションセット*/
     wait1();
     lcdwriteinit( 0x02 );                               /*LCD データを 4 ビット長に設定*/
     wait1();
     lcdwrite1command(0x28);                             /*4bit、2 行文、5×7 ドットに設定*/
     wait1();

     /*   ここまでで 4 ビット×2 回のデータ転送ができるように設定されたので
     この後はビジーチェックをしてからデータ転送を行なう。 */

     while(BUSY == lcdbusycheck())                  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x08);                             /*表示オフ*/

     while(BUSY == lcdbusycheck())                  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x01);                             /*表示クリア*/

     while(BUSY == lcdbusycheck())                  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x06);                             /*エントリーモード、インクリメント*/

     while(BUSY == lcdbusycheck())                  /*LCD ビジーチェック*/
     ;
     lcdwrite1command(0x0c);                             /*表示オン、カーソルオフ*/

}
```

リスト(4/7)

```
/*-----------------------------------------------------------*/
/* 関数名 :lcdwriteinit()                                     */
/* 機能   :LCD 初期設定コマンドセット                          */
/*-----------------------------------------------------------*/

void lcdwriteinit( unsigned char command )
{

     p4 = 0x00;                /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;            /*P4 出力に設定 */

     command &= 0x0f;          /*P4 コントロールデータセット(引数から) RW－0(W 指定) RS－0(コマンド指定) E－0 */
     p4 = command;             /*P4 コントロールデータ出力*/
     LCDE = HIGH;              /*E－１*/
#pragma ASM                         /*アセンブラ表記*/
     NOP                            /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                 /*アセンブラ表記終了*/
     LCDE = LOW;                    /*E－0*/
}
/*-------------------------------------------------------------*/
/* 関数名 :lcdwrite1command()                                   */
/* 機能   :LCD コマンド出力                                     */
/*-------------------------------------------------------------*/

void lcdwrite1command( unsigned char command )
{
     unsigned char outcommand;

     p4 = 0x00;                /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;            /*P4 出力に設定 */

     outcommand = command>>4;/*上位 4 ビットを下位 4 ビットへ*/
     outcommand &= 0x0f;       /*マスク処理*/
     p4 = outcommand;          /*コマンドデータ上位 4 ビットを出力  RW－0(W 指定) RS－0(コマンド指定) E－0 */

     LCDE = HIGH;              /*E－１*/
#pragma ASM                         /*アセンブラ表記*/
     NOP                            /*時間調整の為の NOP*/
     NOP
     NOP
     NOP
#pragma ENDASM                 /*アセンブラ表記終了*/
     LCDE = LOW;                    /*E－0*/

     outcommand = command&0x0f;/*コマンドデータの下位 4 ビット抽出(上位 4 ビットマスク)*/
     p4 = outcommand;               /*コマンドデータ下位 4 ビットを出力*/

     LCDE = HIGH;              /*E－１*/
#pragma ASM                         /*アセンブラ表記*/
     NOP                            /*時間調整の為の NOP*/
     NOP
     NOP
     NOP
#pragma ENDASM                      /*アセンブラ表記終了*/
     LCDE = LOW;                         /*E－0*/

}
```

リスト (5/7)

```
/*--------------------------------------------*/
/* 関数名 :lcdwrite1data()                    */
/* 機能   :LCD データ出力                     */
/*--------------------------------------------*/

void lcdwrite1data( unsigned char data )
{
     unsigned char lcddata;
     p4 = 0x00;               /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUT;           /*P4 出力に設定 */

     lcddata = data>>4;       /*LCD データ上位 4 ビットを下位 4 ビットへ*/
     lcddata &= 0x0f;         /*マスク処理*/
     p4 = lcddata;            /*LCD データ上位 4 ビットを出力*/
     LCDRS = LCD_DATA;        /*RS－1(データ指定) */
     LCDE = HIGH;             /*E－１*/
#pragma ASM                        /*アセンブラ表記*/
     NOP                           /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                /*アセンブラ表記終了*/
     LCDE = LOW;                   /*E－0*/

     lcddata =data & 0x0f;    /*LCD データの下位 4 ビット抽出(上位 4 ビットマスク)*/
     p4 = lcddata;            /*LCD データ下位 4 ビットを出力*/

     LCDRS = LCD_DATA;        /*RS－1(データ指定) */
     LCDE = HIGH;             /*E－1*/
#pragma ASM                        /*アセンブラ表記*/
     NOP                           /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                /*アセンブラ表記終了*/
     LCDE = LOW;                   /*E－0*/
}

/*--------------------------------------------*/
/* 関数名 :wait1()                            */
/* 機能   :0.1ms ウエイト                     */
/*--------------------------------------------*/
void wait1(void){             /* 約 0.1ms のウエイト */
#pragma ASM                        /*アセンブラ表記*/
     MOV.W     #0C8H,A0       /*カウンタ初期値セット*/
LOOP1:
     NOP
     NOP
     NOP
     DEC.W A0
     JNZ  LOOP1               /*ループ終了？*/
#pragma ENDASM                /*アセンブラ表記終了*/
}
```

リスト(6/7)

```
/*---------------------------------------*/
/* 関数名 :wait2()                       */
/* 機能    :4.1ms ウエイト                */
/*---------------------------------------*/
void wait2(void){               /* 約 4.1ms のウエイト */
#pragma ASM                          /*アセンブラ表記*/
     MOV.W      #2007H,A0       /*カウンタ初期値セット*/
LOOP2:
     NOP
     NOP
     NOP
     DEC.WA0
     JNZ  LOOP2                 /*ループ終了?*/
#pragma ENDASM                  /*アセンブラ表記終了*/
}
/*---------------------------------------*/
/* 関数名 :wait3()                       */
/* 機能    :15ms ウエイト                 */
/*---------------------------------------*/
void wait3(void){               /* 約 15ms のウエイト */
#pragma ASM                          /*アセンブラ表記*/
     MOV.W      #7530H,A0       /*カウンタ初期値セット*/
LOOP3:
     NOP
     NOP
     NOP
     DEC.WA0
     JNZ  LOOP3                 /*ループ終了?*/
#pragma ENDASM                  /*アセンブラ表記終了*/
}
```

リスト(7/7)

```
/*--------------------------------------*/
/* 関数名 :lcdbusycheck()               */
/* 機能   :LCD ビジーチェック           */
/*--------------------------------------*/

unsigned char lcdbusycheck( void )
{
     unsigned char command_high,command_low,b_data;

     p4 = 0x00;                 /*P4 初期値セット  RW－0(W 指定) RS－0(コマンド指定) E－0 */
     pd4 = PORTOUTIN;           /*ポート方向レジスタを上位 4 ビットを出力下位 4 ビットを入力に設定*/

     LCDRW = HIGH;              /*RW－1(R 指定)*/

     LCDE = HIGH;               /*E－1*/
#pragma ASM                          /*アセンブラ表記*/
     NOP                             /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                  /*アセンブラ表記終了*/
     command_high = p4;         /*コマンド上位 4 ビット読み込み*/
     LCDE = LOW;                     /*E－0*/

     command_high <<=4;         /*下位 4 ビットを上位へ*/
     command_high &= 0xf0;      /*上位 4 ビット抽出(下位 4 ビットマスク処理)*/

     LCDE = HIGH;               /*E－1*/
#pragma ASM                          /*アセンブラ表記*/
     NOP                             /* 時間調整の為の NOP */
     NOP
     NOP
     NOP
#pragma ENDASM                  /*アセンブラ表記終了*/
     command_low = p4;          /*コマンド下位 4 ビット読み込み*/
     LCDE = LOW;                     /*E－0*/

     command_low &= 0x0f; /*下位 4 ビット抽出(上位 4 ビットマスク処理)*/
     b_data = command_high|command_low;   /* 2 つの 4 ビットコマンドデータを 8 ビットにまとめる*/
     b_data &= 0x80;            /*ビット 7 のマスク処理*/
     if(b_data==0)
          b_data = NOBUSY;      /*ビット 7 が 0 なら LCD 書き込み ok を返す*/
     else
          b_data = BUSY;        /*ビット 7 が 1 なら LCD 書き込み buzy を返す*/

     return(b_data);
}
```

ＯＡＫＳ16 キット
ＯＡＫＳ16 プログラミングテキスト（Ver. 1.01）

資料番号：ＯＡＫＳ16 プログラミングテキスト（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

禁無断転載
本書の一部または全部を、当社に断りなく、いかなる形でも転載または複製することを堅くお断りします。