AVRマイコン アセンブラプログラミング入門篇

AVRマイコン(ATtiny2313)のしくみと開発のしかた

目次

2013年 7月

1. はじめに

この解説書は、コンパクトで高性能なAVRマイコン、ATtiny2313(Atmel社)を使って簡単なプログラムを作成することで、マイコンのしくみと動作、プログラムの開発のしかたについて学習することを目的としています。
最初に、ATtiny2313のしくみを解説することで、プログラムがマイコン内部で、一体どのように実行されるのか、プログラムカウンタの動作、よく使われる主な命令、入出力ポートの機能について学習します。
次に、AVRマイコンの開発環境、「Atmel Studio 6」を使って、マイコン上で動くプログラムを実際に作るにはどうするのか、開発手順を学習します。
次に、ATtiny2313の周辺機能(入出力ポート、タイマカウンタ)の基本的な使い方を、簡単な実例のプログラムを動かしてみることで、学習します。

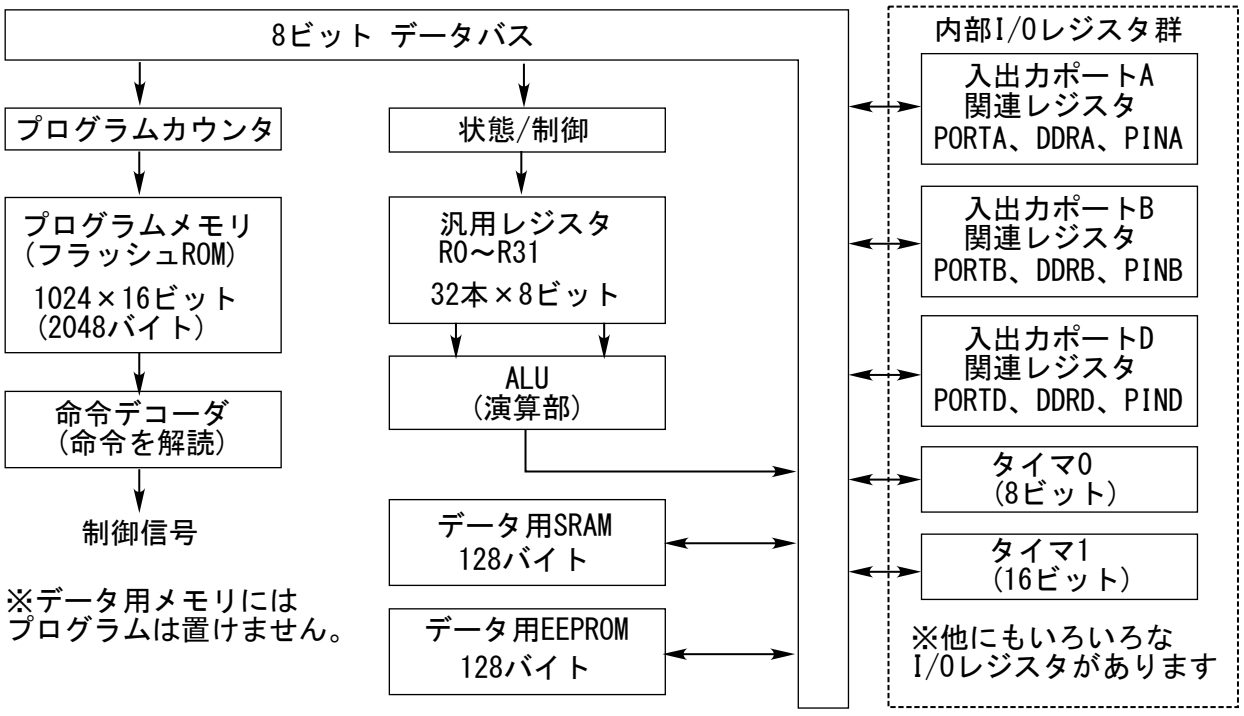
2. AVRマイコン(ATtiny2313)のしくみ

2.1. ATtiny2313の主な仕様

AVRマイコン、ATtiny2313は、Atmel社の高性能ワンチップマイコンです。
ATtiny2313の主な仕様は、次のとおりです:

◎フラッシュメモリ(プログラムメモリ) : 2048バイト(1024命令ワード)
◎データ用SRAM : 128バイト
◎データ保存用EEPROM : 128バイト
◎入出力ポート : 17本
◎8ビットタイマカウンタ×1
◎16ビットタイマカウンタ×1
◎シリアル通信機能 : USART×1チャネル、USI(汎用シリアルインターフェイス)×1チャネル
◎動作クロック : 1MHzまたは8MHzの内部クロック
※水晶発振子を外付けすることで、20MHzまでの外部クロックでも動作可能です。
◎ISP機能 : マイコンを基板に取り付けた状態でプログラムの書き換えができます。
◎電源電圧 : 1.8V～5V

2.2 ATtiny2313の主要部ブロックダイヤグラム



2.3.内部メモリの説明（プログラムメモリ/データSRAM/EEPROM)

AVRマイコンには、内部メモリとして、プログラムを格納するプログラムメモリとデータを記憶するRAM、EEPROMが用意されています。

プログラムメモリはフラッシュROMです。プログラムメモリの命令実行用アドレスは、データメモリのアドレスとは別に割り振られています。プログラムメモリは、1アドレスにつき、16ビットの幅があります。

また、プログラムメモリには、実行中に変更しないデータを配置することもできます。そのデータをプログラム中で利用するには、LPM命令を使います。

データSRAMは、データを一時的に記憶してプログラム中で利用するためのメモリです。アドレスはプログラムメモリのアドレスとは別に割り振られています。

※データSRAM上には、プログラムは置けません。

データSRAMは、プログラム中で自由に読み書きできます。

※データSRAMの内容は、マイコンの電源が切れると消えてしまいます。

ATtiny2313の場合、アドレス0x60からデータSRAMになっています。同じデータSRAMのアドレス空間上に、汎用レジスタ、I/Oレジスタも割り当てられています。

汎用レジスタとI/Oレジスタのアクセスには、普通はこのアドレスを使いません。汎用レジスタはレジスタの番号(R0～R31)を使ってアクセスします。

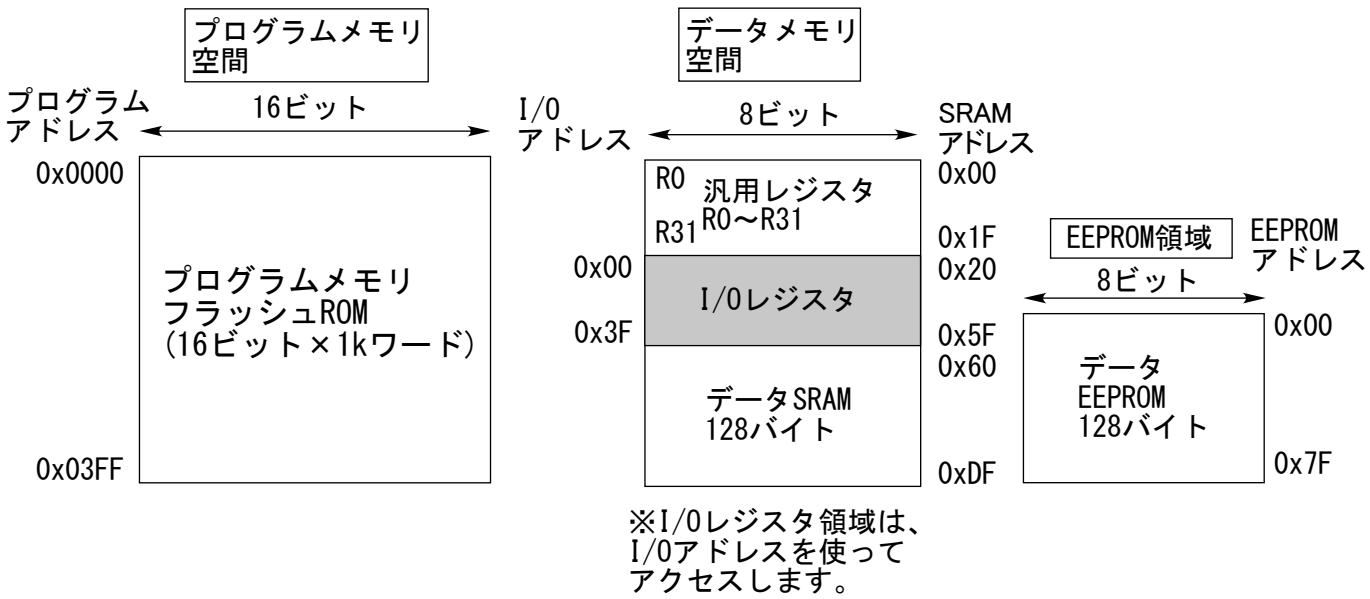
I/Oレジスタはデータ SRAMのアドレス空間中に割り当てられていますが、SRAMアドレスとは別に、I/Oアドレスという特別なアドレス(0x00～0x3F)が割り当てられていて、IN/OUT命令でアクセスします。

※たとえば、I/Oアドレスの0x00番地と、SRAMアドレスの0x20番地は、同じI/Oレジスタを指します。

データEEPROMは、マイコンの電源が切れても内容が消えないメモリです。

EEPROMのアドレスは、プログラムメモリ、データ用SRAMのアドレスとは別に割り振られています。

データEEPROMの読み書きは、I/Oレジスタにアクセスすることで行います。



2.4. プログラムはどのように実行されるのか?

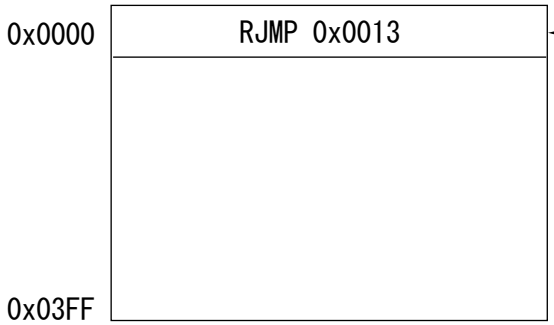
マイコンは、プログラムに書かれた命令に従って動きます。

プログラムを実行するために、マイコンはプログラムカウンタ中の値で指し示されるプログラムメモリの番地から1個ずつ命令を読み出します。(これを命令フェッチといいます)

マイコンは1マシンサイクル(AVRマイコンの場合は原則として1クロック)中に、1個の命令しか実行できません。この1個の命令は、汎用レジスタにデータを移動(コピー)する、マイコン内部のI/Oレジスタにアクセスする、プログラムカウンタの値を変更する(ジャンプする)といった、ごく単純な操作しかできません。

複雑な操作は、こうした単純な操作を組み合わせて、高速かつ順番に実行することで行います。

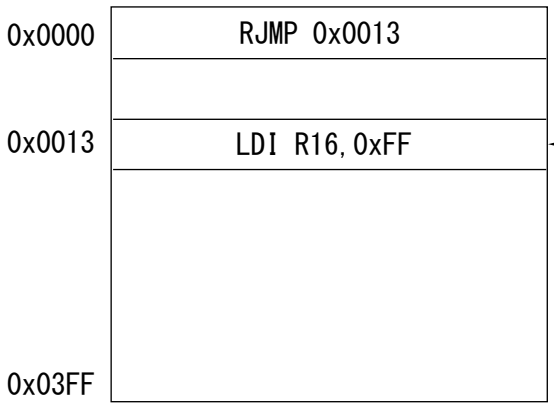
(1)リセット発生直後



マイコンの電源を入れたり、リセット信号を与えると、プログラムカウンタの内容は0x0000に初期化されます。

プログラムメモリ中の0x0000番地の命令(この例では、「RJMP 0x0013」)を読み出し、実行します。

(2)ユーザプログラムの1個目の命令の実行

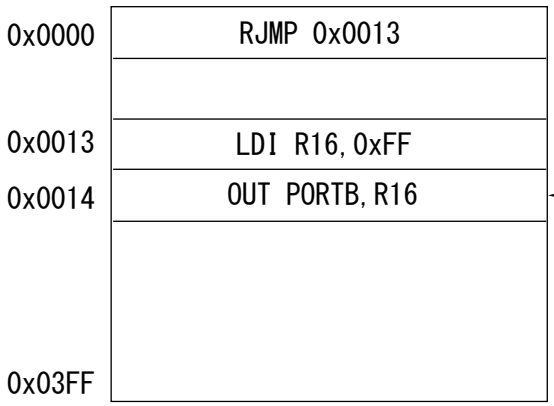


「RJMP 0x0013」という命令は、「プログラムカウンタの内容を0x0013に変更せよ」という命令です。

マイコンは、プログラムカウンタで指示された0x0013番地の命令(この例では、「LDI R16,0xFF」を読み出し、実行します。

命令を実行すると、プログラムカウンタの値は1だけ増えます。

(3)ユーザプログラムの2個目の命令の実行



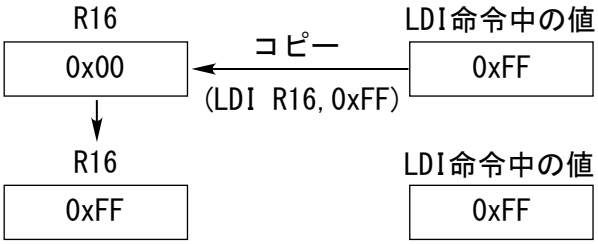
プログラムカウンタの値が1だけ増え、0x0014になりました。マイコンは、プログラムカウンタで指示された0x0014番地の命令(この例では、「OUT PORTB,R16」)を読み出し、実行します。

命令を実行すると、プログラムカウンタの値は1だけ増えて、次の番地の命令を指示するようになります。

このようなしくみで、マイコンはプログラムを実行します。AVRマイコン(ATtiny2313)で、どのような命令が実際に用意されているのかは、ATtiny2313のデータシートに載っています。全部で100種類以上の命令がありますが、実際によく使われる命令とその操作の一覧は、巻末の資料篇に載せます。

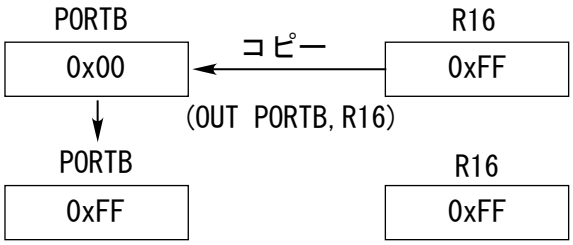
マイコンの命令で行われる操作(主な例)

(1)汎用レジスタにデータを移動する



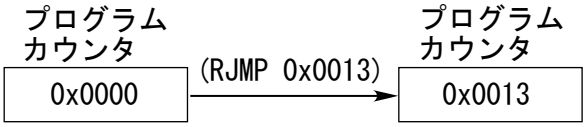
※マイコンの場合、「データの移動」とは「データのコピー」のことです。コピー元のデータは変化しません。

(2)I/Oレジスタの読み書き



※I/Oレジスタのアクセスの場合でも、コピー元のレジスタ(この場合汎用レジスタR16)のデータは変化しません。

(3)ジャンプする



プログラムカウンタの値を変更すると、次の命令の読み出し(命令フェッチ)は変更後のプログラムカウンタが指し示しているプログラムメモリのアドレスから行われます。

※ATtiny2313の全命令セットの概要を、ATtiny2313のデータシートより引用して33～34ページに載せていますので、参考にしてください。

2.5.汎用レジスタ(R0～R31)について

汎用レジスタは、各種I/Oレジスタに設定データをコピーするのに使ったり、内部SRAM上のデータを読み書きしたり、足し算、引き算、比較、論理演算などの演算を行ったりと、あらゆる場面で使われます。

AVRマイコンの場合、内部SRAM上のデータ同士を直接足し算したり、比較論理演算したりする命令はありません。また、I/Oレジスタ上のデータの処理も、I/Oレジスタ上では行えません(※)。

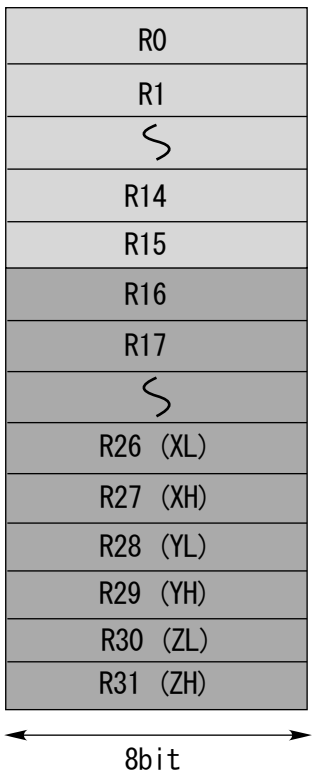
このため、ユーザのプログラムでは、汎用レジスタを介して処理を行い、内部SRAMに格納したり、I/Oレジスタにコピーしたりします。

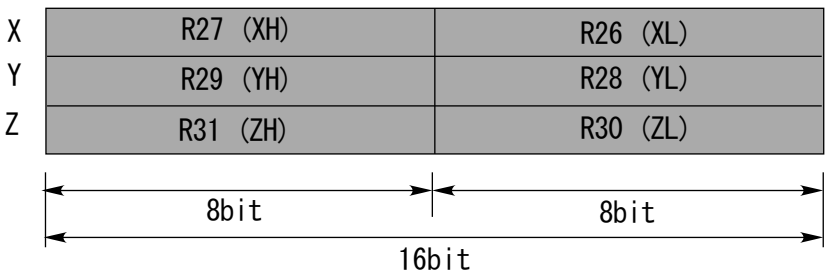
AVRマイコンには、R0からR31まで、32本の汎用レジスタがあります。

特に、右の図で濃いねずみ色をしているR16からR31のレジスタはほとんどの命令で使えます。右の図で薄いねずみ色をしているR0からR15は、LDI命令など、命令によっては使えないことがありますので使用に当たっては注意してください。

詳細は命令セットマニュアルを見てください。

※R16からR25までの10本の汎用レジスタは、プログラム上で全く同じように使用できます。



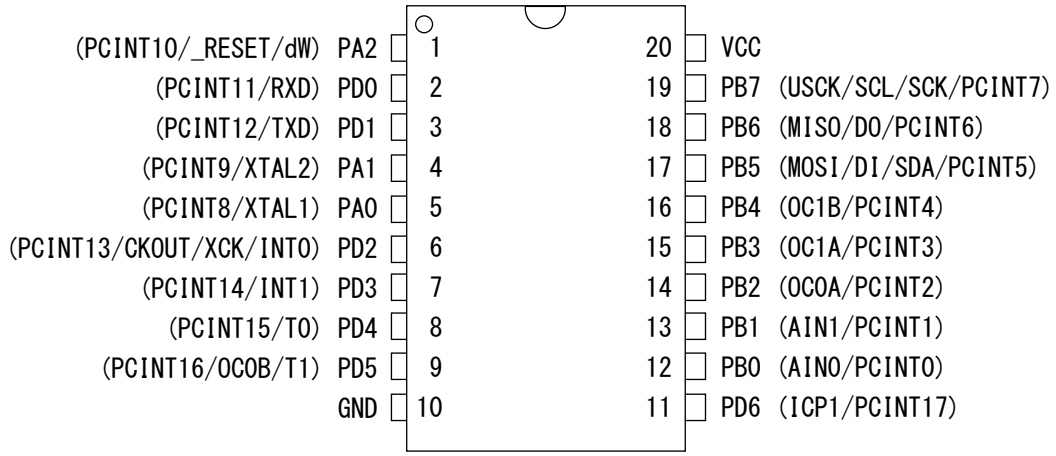


R26とR27、R28とR29、R30とR31を使って16bitレジスタとして使えます。それぞれXレジスタ、Yレジスタ、Zレジスタと呼びます。

※ I/Oレジスタ上のビットを直接「1」にセットしたり「0」にクリアしたりする命令もありますが、使用できるI/Oレジスタに制限があります。
このため、このチュートリアルでは、I/Oレジスタの操作も必ず汎用レジスタを介して行うことにします。

2.7.マイコンのピンと内部I/Oの関係

下の図は、ATtiny2313のピン配置です。ICの表側から見た状態で描いてあります。



ATtiny2313では、通常のI/Oポートの機能以外に、通信などのさまざまな機能を、同じピンを共通に使うことで実現しています。ポートの名前の横の、括弧内の信号名は、機能設定することで使用できるようになる、通常のI/Oポートの機能以外の機能を表しています。
(詳細については、ATtiny2313のデータシートを参照してください)

2.8.I/Oポート(デジタル入出力ポート)の入出力の設定

I/O アドレス	レジスタ名	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x1B	PORTA	-	-	-	-	-	PORTA2※	PORTA1	PORTA0
0x1A	DDRA	-	-	-	-	-	DDA2※	DDA1	DDA0
0x19	PINA	-	-	-	-	-	PINA2※	PINA1	PINA0
0x18	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x17	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x12	PORTD	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x11	DDRD	-	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x10	PIND	-	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0

※PA2は、リセット端子と共通になっています。通常は使用できません。

上の表は、ATtiny2313の、I/Oポート関係のI/Oレジスタの概要です。ATtiny2313には、ポートA、ポートB、ポートDの3つのI/Oポートがありますので、設定用レジスタもポートA用、ポートB用、ポートD用の3組あります。

(1) DDRxレジスタ(DDRA、DDRB、DDRD)の機能

DDRxは、ポートの入出力の方向を決めるためのレジスタです。出力に割り付けたいポートのピンに対応するDDRxのビットに「1」を書き込むと出力、「0」を書き込むと入力になります。

例えば、ICのピンPB0～PB7を出力ポートに割り付けるには、DDRBレジスタのビットに相当するビットに「1」を書きます。DDRBは、8ビット幅のI/Oレジスタです。

直接I/Oレジスタにデータを書く命令はありませんので、汎用レジスタ(例えばR16)を経由させて、DDRBにデータを入れます。

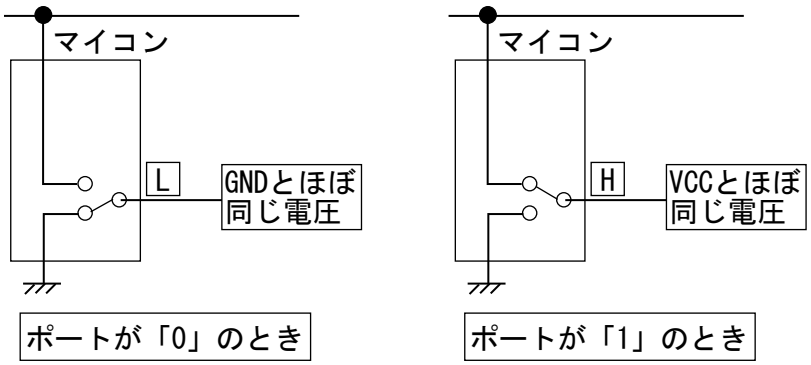
(例) PB7を出力に、PB6～PB0を入力に設定する場合

```
        LDI     R16,0b10000000
        OUT     DDRB,R16
```

(2)PORTxレジスタ(PORTA、PORTB、PORTD)の機能

PORTxは、ポートにデータを出力するためのレジスタです。PORTxを使ってデータを出力する前に、データを出力したいポートに対応するDDRxのビットを「1」にして、ポートを出力モードにする必要があります。

PORTxに「1」のビットを出力すると、出力は「H」レベル(ほぼ電源電圧いっぱい)、「0」のビットを出力すると、出力は「L」レベル(ほぼグラウンドレベル)になります。



(例) 出力に設定されたPB7に「H」レベルを出力する場合

```
LDI     R16,0b10000000      ; PB7を出力に設定する
OUT     DDRB,R16
LDI     R16,0b10000000      ; PB7に「H」を出力
OUT     PORTB,r16
```

(例) 出力に設定されたPB7に「L」レベルを出力する場合

```
LDI     R16,0b10000000      ; PB7を出力に設定する
OUT     DDRB,R16
LDI     R16,0b00000000      ; PB7に「L」を出力
OUT     PORTB,r16
```

(3)PINxレジスタ(PINA、PINB、PIND)の機能

PINxは、マイコン外部の実際のピンの状態が反映されているI/Oレジスタです。汎用レジスタを介してPINxレジスタを読み出すと、ポートを入力に設定しているか出力に設定しているかに関係なく、汎用レジスタにマイコン外部の実際のピンの状態が入ります。

(例) ポートBの実際のピンの状態を読み取る

```
IN      R16,PINB
```

3. 実際にマイコンを動かしてみる

この章では、ATtiny2313汎用ボードとブレッドボードを使って、実際にマイコンを動かしてみます。AVRマイコンの開発環境(Atmel Studio 6)を使った、プロジェクトの作成のしかた、プログラムの書き方について説明します。

3.1. 「Atmel Studio 6」のダウンロード

AVRマイコンのプログラムは、「Atmel Studio 6」という統合開発環境上で作成します。
今回のセミナーで使うATtiny2313以外のAVRマイコンのプログラム開発にも使えます。
また、アセンブラ以外にC言語のコンパイラも入っていますので、C言語でのプログラム開発にも便利です。
「Atmel Studio 6」は、アトメル社のWebサイトからダウンロードできます。
(2013年7月現在、無償でダウンロード、使用できます)

ダウンロードのしかた:
(1)はじめに、「http://www.atmel.com」にアクセスします。
トップページの上段の「Design Support」バーにマウスのカーソルを持っていくと、開発ツールの一覧が出ます。
「Development Tools」の中の、「Atmel Studio IDE」をクリックします。

(2)「Atmel Studio 6」の紹介ページに飛びます。「Atmel Studio 6.1 update 1.1 (build 2674) Installer -Full」のCDのアイコンがありますので、それをクリックします。

(3)ゲストダウンロードの画面に飛びますので、必要事項を記入して、[Submit]ボタンをクリックします。

(4)記入した電子メールアドレスあてに、ダウンロードの案内が書かれた電子メールが届きますので、その指示に従ってダウンロードします。

ダウンロードした「Atmel Studio 6」のインストーラのアイコンをダブルクリックして実行すると、インストールが行われます。インストーラの指示に従ってインストールしてください。

3.2. AVRライタ(AVRWRT)のインストール

AVRマイコンにプログラムを書き込むには、AVRライタを使います。今回のセミナーでは、デジットオリジナルのAVRライタ(AVRWRT 3)を使います。
AVRライタをパソコンに接続する前に、AVRライタ付属のデバイスドライバのインストールを行ってください。

重要!!

◎ AVRライタ付属のデバイスドライバのインストールが完了するまでは、AVRライタをパソコンに接続しないでください。
AVRライタは、デバイスドライバのインストールが完了してから、パソコンに接続してください。

◎ AVRライタのインストールのしかたについては、付属のCD-ROMに入っている「インストールのしかた」を参照してください。

◎ AVRライタの実行プログラムには、32ビット版Windows用と64ビット版Windows用の2種類がありますので、お使いのパソコンに合ったほうをパソコンのハードディスクにコピーしてください。

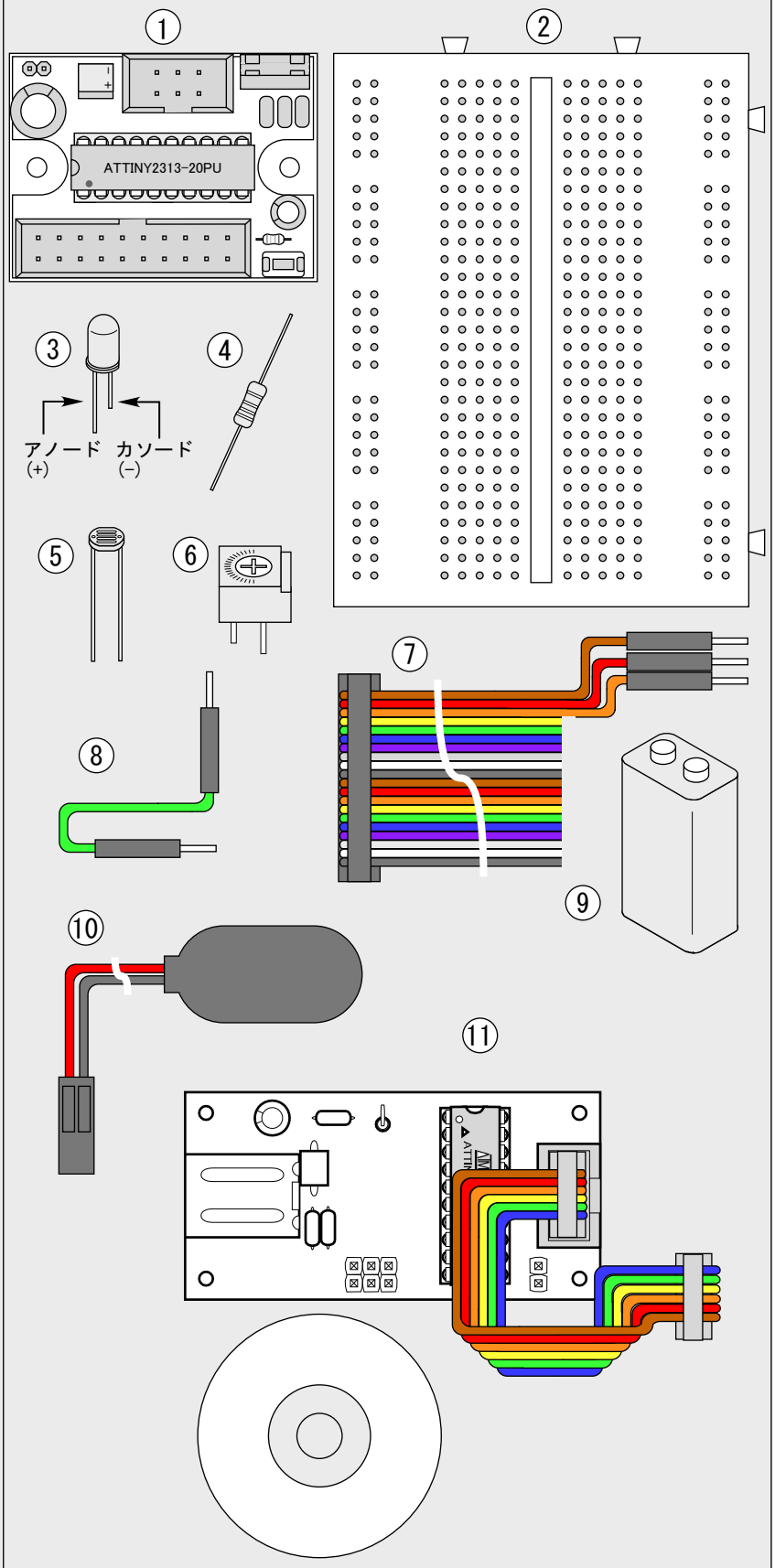
3.3 [LEDの点滅]

ここでは例として、LEDを点滅させるプログラムを実際に作って動かしてみます。

(1)ATtiny2313汎用ボードと
LED点滅ユニットを用意します。

部品セット内訳

①	ATtiny2313汎用ボード	1個
②	ブレッドボード	1個
③	5mm LED	8個
④	1/4W 470Ω抵抗(黄紫茶金)	8個
⑤	CdSセル	1個
⑥	半固定抵抗 10kΩ(103)	1個
⑦	QIケーブル(FC20-1Px20)	1個
⑧	QIケーブル(1P-1P 5cm)	1個
⑨	006P乾電池(9V)	1個
⑩	006P電池スナップ	1個
⑪	AVRライタ(AVRWRT3)	1セット



(2)QIケーブルで配線します。

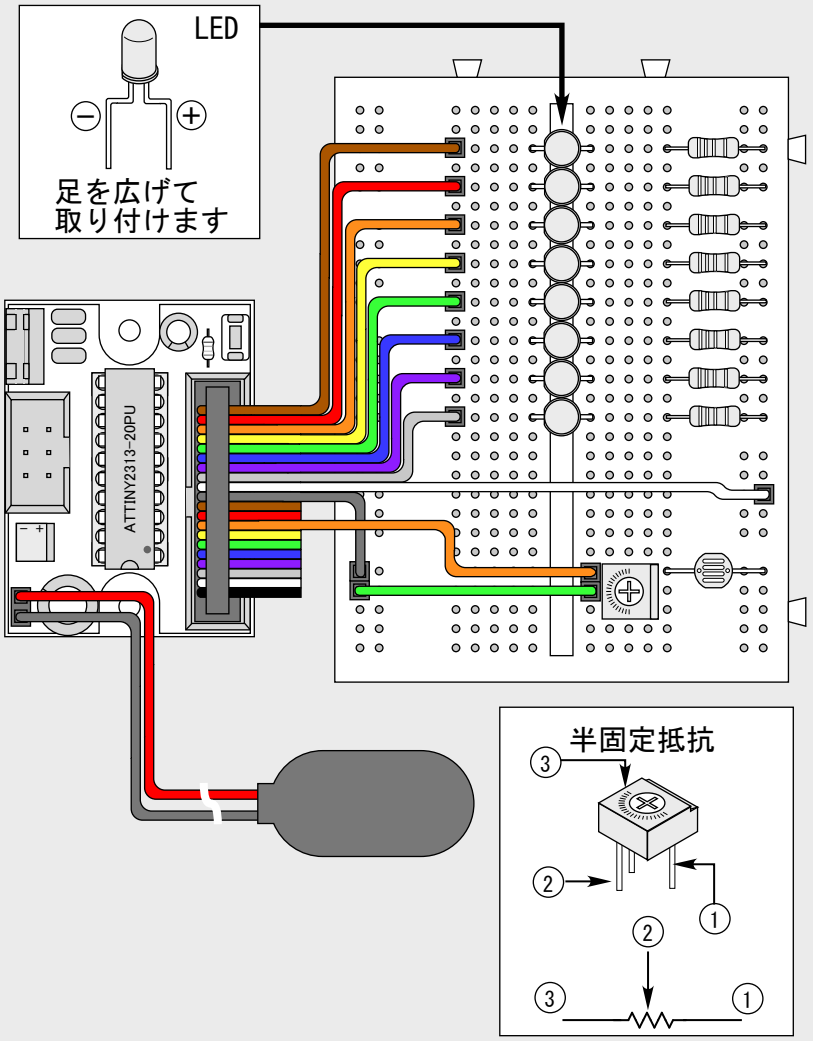
① 抵抗とLED、CdSセルと半固定抵抗を、ブレッドボードに挿します。

② ATtiny2313汎用ボードのCN2(20ピンBOXコネクタ)とブレッドボードの間を右図のように接続します。

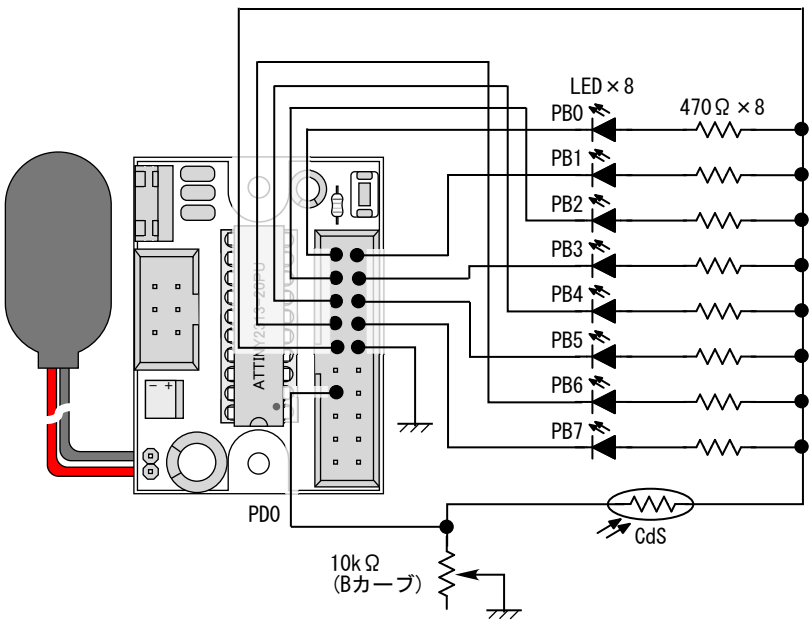
③ 電池スナップに、006P電池を接続します。

◎抵抗、CdSセルにはプラスマイナスの極性はありませんので、どちら向きに差し込んでもOKです。

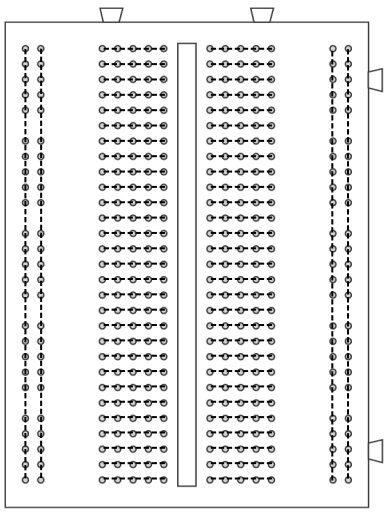
◎ LEDの極性については、右の図を見てください。



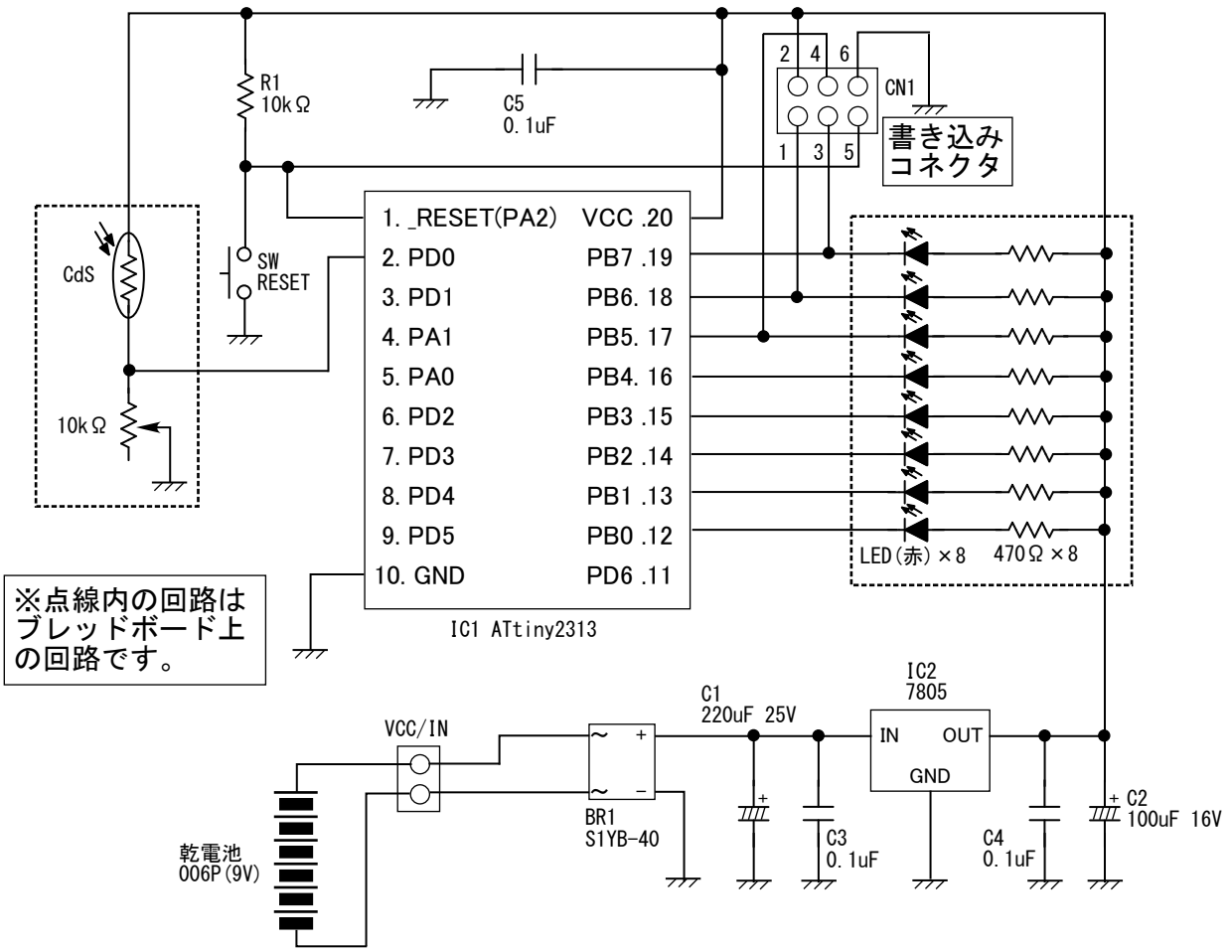
ブレッドボード上の回路図



ブレッドボードの
内部配線

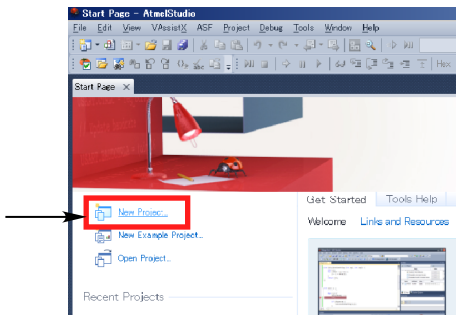


P-10の回路図



(3)Atmel Studioで新しいプロジェクトを作ります。

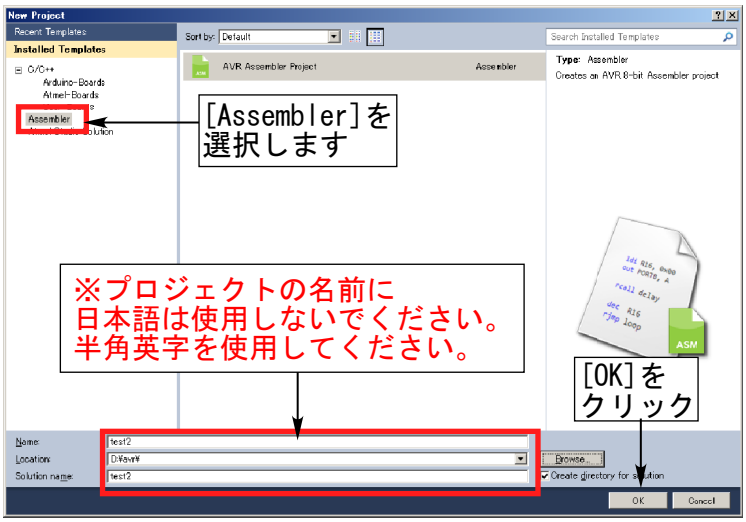
(1)Atmel Studioを起動すると、スタートページのウインドウが出ますので、ウインドウ左側の[New Project]をクリックします。



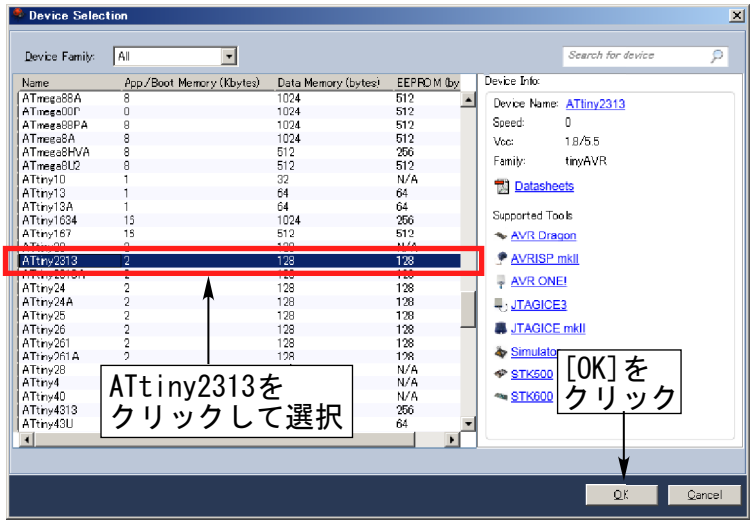
[New Project]をクリックすると、新しいプロジェクトを作るウインドウが出ますので、左側の「Installed Templates」中の[Assembler]をクリックします。

ウインドウ下側の[Name]欄に好きな名前を入れます。ここで入力した名前が、プロジェクトの名前になります。
※ここでは例として「TEST2」と入れています。

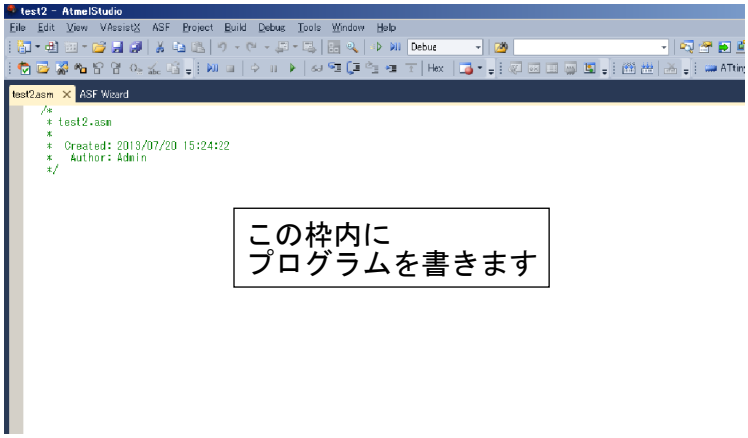
これから作るプロジェクトに好きな名前をつけた後、[OK]をクリックします。



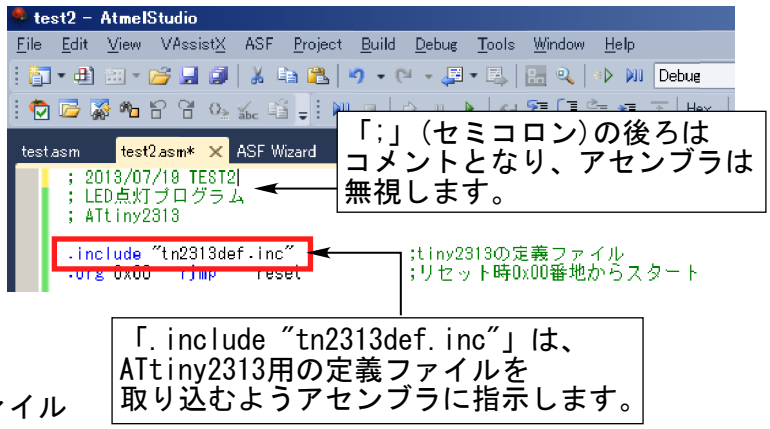
(2)「Device Selection」のウインドウが開き、デバイスの一覧が表示されます。
一覧のなかから、使用するマイコンの型番(この場合はATtiny2313)を選んで、[OK]ボタンをクリックします。



(3)「Finish」 のクリック後、ウィンドウが開きます。
これでアセンブリが出来る様になりましたので、開いたウィンドウにプログラムを入力していきます。



(4)プログラムを書くときに、プログラムに関するコメントを書いておくと、後で見直すときに確認しやすいです。
プログラムの行中で、「;」(セミコロン)の後ろはコメントになり、アセンブラは無視します。
デフォルトの設定では、緑字表示になります。



(5)AVRでは、プログラムの先頭に定義ファイルをインクルードしてからプログラムを入力します。

ATtiny2313汎用ボードは、ATtiny2313を使用していますので、定義ファイルは
「.include "tn2313def.inc」となります。

※「.include "(ファイル名)"」は、ビルド時に指定した名前のファイルを取り込むようアセンブラに指示する擬似命令です。

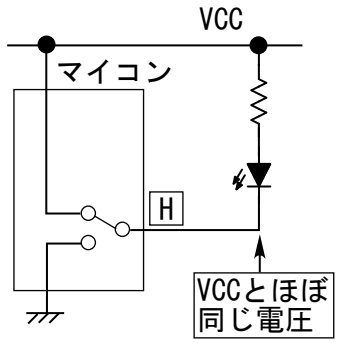
(5)LEDを点滅させるプログラムの例です。
※アセンブラは大文字小文字を区別しませんので、プログラムは大文字小文字どちらで書いても
かまいません。

```
;--------
; 2013/07/22 TEST2
; LED点灯プログラム
; ATtiny2313 1MHz
;--------

;-------- アセンブラプログラムの決まりごと --------
;(1) ';(セミコロン)'はコメント記号です。行の中に現れると、アセンブラは
;      セミコロンより後ろを無視します。
;(2) アセンブラは名前や命令の大文字小文字の違いを無視します。
;      大文字小文字どちらでプログラムを書いてもかまいません。
;      ※データ中の大文字小文字は区別されます。
;      ※コメント以外の場所では、全角文字は使えません(エラーになります)
;(3) '.include'など、頭に'.(ピリオド)'がついているのは開発環境に対する
;      作業指示を意味する擬似命令です。
;(4) 「main:」など、名前の後ろに':(コロン)'があるのはラベルです。
;      ジャンプやサブルーチンコール命令の飛び先を指定するのに使います。
;--------
      .include "tn2313def.inc"  ;ATtiny2313用定義ファイルを取り込みます
                                ;このファイルの中で、I/Oレジスタの名前などが
                                ;定義されています。
;--------
; 割り込みベクタ領域 (ATtiny2313の場合、0x0000番地～0x0012番地)
; 使う割り込みの頭のコメント記号(セミコロン)を外して使います。
;--------
.org 0x00    rjmp reset          ;各種リセット : 「reset:」というラベルのところに
                                 ;ジャンプします。
;.org 0x01   rjmp ext_int0       ;外部割り込み要求0
;.org 0x02   rjmp ext_int1       ;外部割り込み要求1
;.org 0x03   rjmp tim1_capt      ;タイマ/カウンタ1捕獲(キャプチャ)発生
;.org 0x04   rjmp tim1_compa     ;タイマ/カウンタ1比較A一致
;.org 0x05   rjmp tim1_ovf       ;タイマ/カウンタ1オーバーフロー
;.org 0x06   rjmp tim0_ovf       ;タイマ/カウンタ0オーバーフロー
;.org 0x07   rjmp usart_rxt      ;usart受信完了
;.org 0x08   rjmp usart_udre     ;usart送信バッファ空
;.org 0x09   rjmp uasrt_tx       ;usart送信完了
;.org 0x0a   rjmp ana_comp       ;アナログ比較器出力遷移
;.org 0x0b   rjmp pcint          ;ピン変化割り込み要求
;.org 0x0c   rjmp tim1_compb     ;タイマ/カウンタ1比較B一致
;.org 0x0d   rjmp tim0_compa     ;タイマ/カウンタ0比較A一致
;.org 0x0e   rjmp tim0_compb     ;タイマ/カウンタ0比較B一致
;.org 0x0f   rjmp usi_strt       ;usi開始条件検出
;.org 0x10   rjmp usi_ovf        ;usiカウンタオーバーフロー
;.org 0x11   rjmp ee_edy         ;eeprom操作可
;.org 0x12   rjmp wdt_ovf        ;ウォッチドッグ時計完了
;--------
; ※使わない割り込みベクタは書かなくてもかまいませんが、
; スペースはあけておきます
; (ATtiny2313の場合、プログラムを0x0013番地から書くようにします)
;--------
```

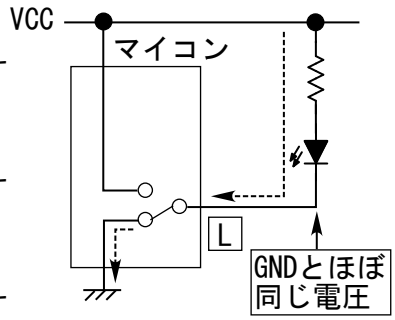
次のページに続きます。

前のページの続きです。

```
        .org 0x13                   ; .org <アドレス>は、アセンブラに命令を置く
                                    ; プログラムアドレスを指示するための擬似命令です。
                                    ; この場合、.org擬似命令に続くプログラムは、
                                    ; プログラムメモリの0x0013番地以降に置かれます。
                                    ; 頭が「0x」ではじまる数値は16進数です。

;--------
; リセット時の処理
; (1)スタックポインタの初期化
;    ATtiny2313の場合、RAMEND(内部SRAMの最終番地)は0x00df番地です。
;    RAMENDの値は、"tn2313def.inc"中で定義されています
;--------
reset:      ldi     r16,low(RAMEND)     ;「low」は16ビットの定数の下位8ビットを
                                        ;取り出すための演算子です。
                                        ;ビルドすると「ldi  r16,0xdf」になります。
            out     SPL,r16
;--------
; (2)ポートの初期化
;--------
            ldi     r16,0b11111111      ;ポートB(PB0～PB7)を
            out     DDRB,r16            ;全ビット出力モードにする
            ldi     r16,0b11111111      ;ポートD(PD0～PD6)を
            out     DDRD,r16            ;全ビット出力モードにする
            ldi     r16,0b11111111      ;ポートA(PA1～PA0)を
            out     DDRA,r16            ;全ビット出力モードにする
                                        ;頭が「0bではじまる数値は2進数です。
;--------
;mainループ : ここに繰り返し実行させる内容を書きます
; ※このチュートリアルの回路では、点灯させたいLEDのポートを「L」にする
; (PORTレジスタの対応するビットを「0」にする)とLEDが点灯します。
;--------
main:       ldi     r16,0b11111110      ; PB0のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11111101      ; PB1のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11111011      ; PB2のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11110111      ; PB3のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11101111      ; PB4のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11011111      ; PB5のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b10111111      ; PB6のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b01111111      ; PB7のLEDを点灯させます
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            rjmp    main                ; mainに飛びます(無限ループ)
```



ポートが「1」のとき
LEDは点きません



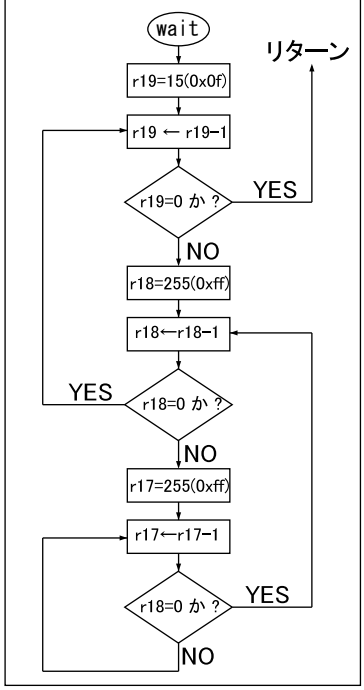
ポートが「0」のとき
点線の向きに電流が流れ
LEDが点灯します

次のページに続きます。

前のページの続きです。

```
;******** wait : 時間稼ぎ用ルーチン ********
wait:       ldi   r19,0x0f      ;r19 = 0x0f(10進で15)
wait1:      dec   r19           ;r19を1減らし
            cpi   r19,0         ;r19が0になったかどうか比較
            breq  waitend       ;0だったら、waitendへジャンプ
            ldi   r18,0xff      ;r19が0でなければ、r18を0xff
                                ;にする
wait2:      dec   r18           ;r18を1減らし
            cpi   r18,0         ;r18が0になったかどうか比較
            breq  wait1         ;r18が0だったら、wait1に
                                ;ジャンプ(繰り返す)
            ldi   r17,0xff      ;r18が0でなければ、r17を0xffに
                                ;する
wait3:      dec   r17           ;r17を1減らし
            cpi   r17,0         ;r17が0になったかどうか比較
            breq  wait2         ;r17が0だったら、wait2に
                                ;ジャンプ
            rjmp  wait3         ;r17が0でなければ、wait3に
                                ;ジャンプ
waitend:    ret                 ;サブルーチンからリターン
```

wait
時間稼ぎ用ルーチン
フローチャート



参考(比較と条件分岐)

◎ ここでいう「比較」とは、「R17-0」の計算結果を調べることで、R17の値が「0」であるかどうかを調べることを指します。
R17の値が「0」ならば、「R17-0」の計算結果は「0」です。
※ここでは、R17の値が「0」であれば「wait2」のラベルに分岐し、「0」でなければ次の命令に進んでいます。

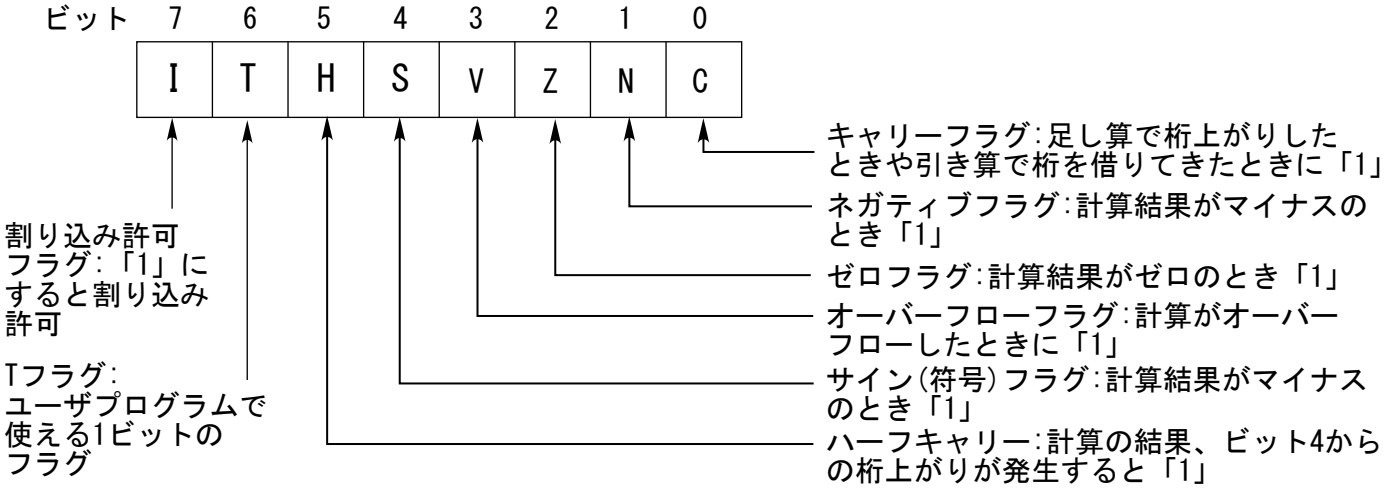
◎ CPI(シーピーアイ)命令は、引き算命令の一種です。
この例では、R17から数値「0」を引いて、結果が「0」ならば、ステータスレジスタ(SREG)のゼロ(Z)フラグが「1」になり、引き算の結果が「0以外の値」ならば、ゼロ(Z)フラグが「0」になります。

◎ BREQ(ブランチイコール)命令は、この例では、ステータスレジスタ(SREG)のゼロ(Z)フラグが「1」ならば、「wait2」のラベルに分岐し、「0」ならば何もせず次の命令に進みます。
「BREQ」の「EQ」は、ここでは「R17-0」の計算結果が「0」であること(ステータスレジスタの「Z」フラグが「1」)を指します。

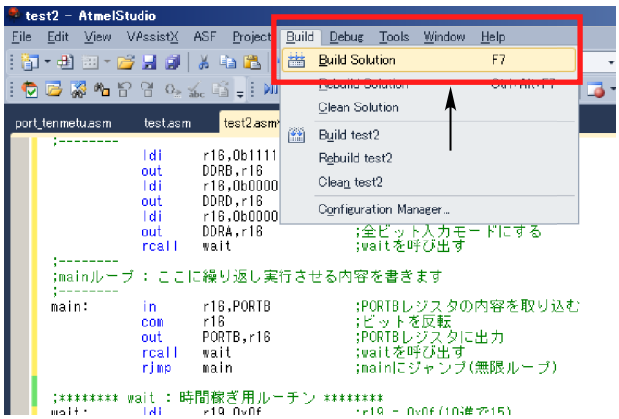
◎ 条件分岐命令は、比較命令の直後で実行してください。
間に他の命令が入ると、ステータスレジスタ(SREG)のフラグが変更されることがあるので、正しく条件判断できないことがあります。

(補足)ステータスレジスタ(SREG)について

ステータスレジスタ(SREG)は、I/Oレジスタの1つで、汎用レジスタで足し算や引き算などの計算を行った結果のフラグが反映されているレジスタです。



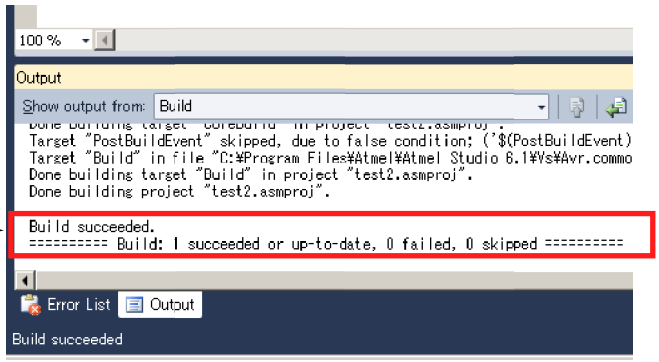
(6)プログラムを全て入力し終わったら、[Build]メニュー中の[Build]をクリックしてビルドします。



プログラムのビルドに成功すると、画面下のウインドウに、

```
Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

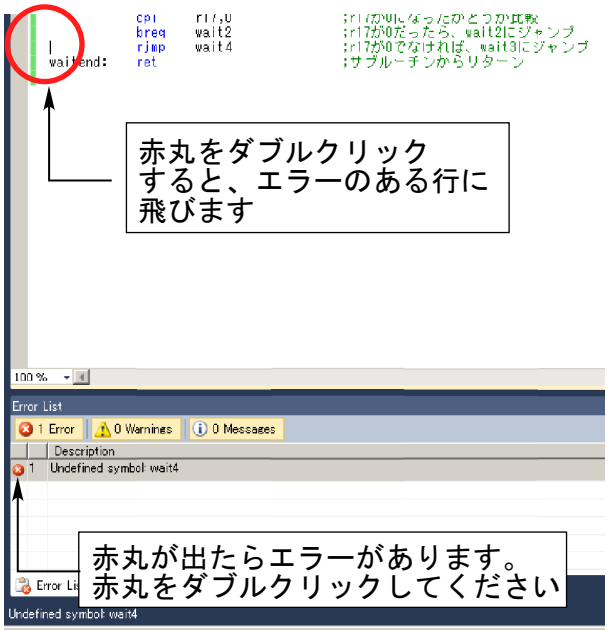
というメッセージが出ます。



プログラムのビルドがうまくいったメッセージです

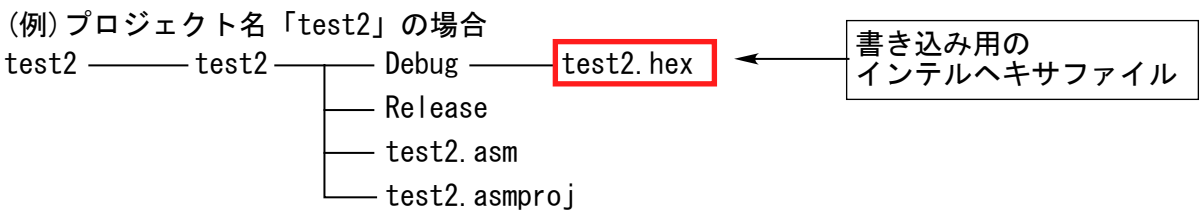
(7)命令の書き方が間違っているなど、ビルドでエラーが検出されれば、画面下に赤丸のエラー表示が出ますので、赤丸をダブルクリックしてください。

赤丸をダブルクリックすると、カーソルがエラーのある行に飛んでいって場所を教えてくれますので、修正してから再度「Build」しましょう。
(ここでは、wait3をwait4と書き間違えています)



(4)ビルドしたプログラムをマイコンに書き込みます。
プログラムのビルドが成功すると、次の場所に書き込み用のファイル(インテルヘキサ.hexファイル)ができます。

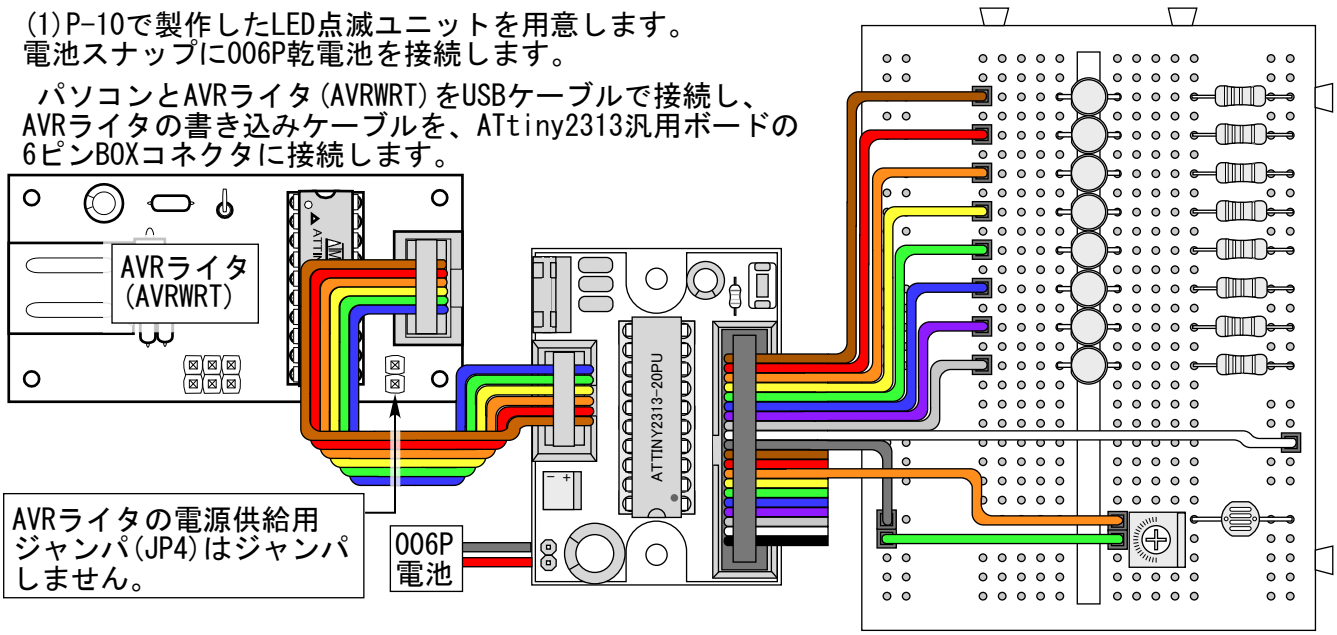
(例)プロジェクト名「test2」の場合



このインテルヘキサファイルを、書き込み器で開いてマイコンに書き込むと、マイコンがプログラムのとおりに動きます。

(1)P-10で製作したLED点滅ユニットを用意します。
電池スナップに006P乾電池を接続します。

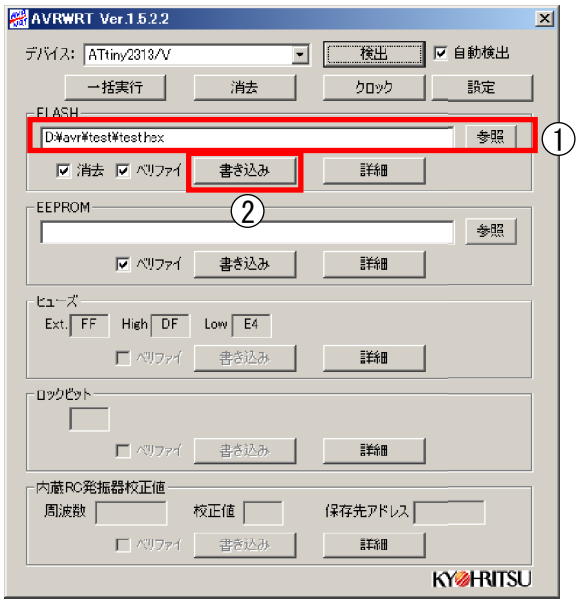
パソコンとAVRライタ(AVRWRT)をUSBケーブルで接続し、AVRライタの書き込みケーブルを、ATtiny2313汎用ボードの6ピンBOXコネクタに接続します。



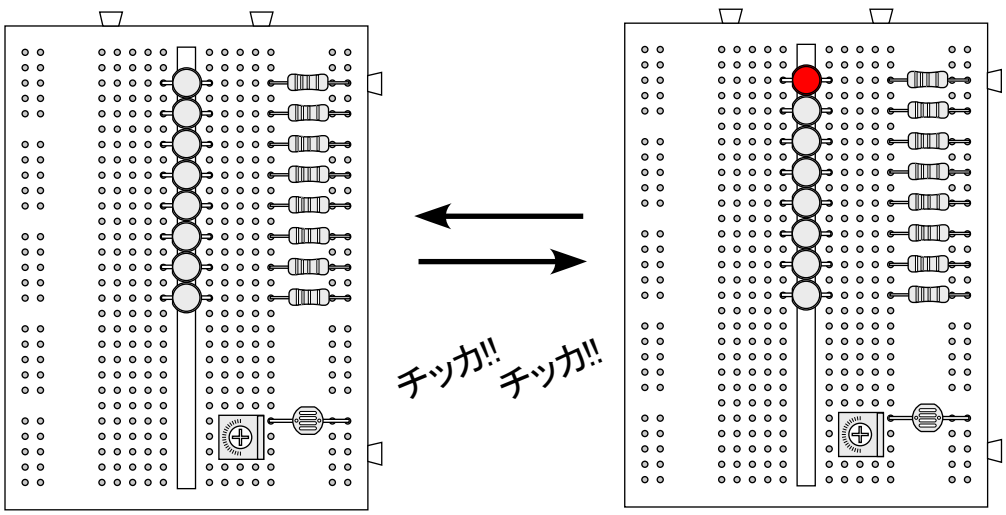
(2)AVRライタのソフトを立ち上げ、
プログラムを書きこみます。

①LED点滅プログラムのヘキサファイル
(ファイル名の末尾が「.hex」のファイル)を、
「参照」で選択します。
(「参照」ボタンをクリックすると、「ファイルを開く」ウインドウが出ますので、書き込むヘキサファイルを選択します)

②「書き込み」ボタンをクリックすると、
プログラムがマイコンに書き込まれます。



(3)LEDが点滅します。



3.4.外部の状態をマイコンの動作に反映させる

今回のチュートリアルで使うブレッドボードの回路には、CdSセルを1個使用しています。

CdSセルは、光が当たると抵抗値が下がる物質(CdS:硫化カドミウム)を塗って焼き付けたもので、周囲の明るさに合わせて抵抗値が変化する抵抗です。周囲が明るいときは抵抗値が低く、周囲が暗いときは抵抗値が高くなります。

CdSセルは、周囲の明るさに反応するセンサとして、夜になると点灯する街灯などに、広く使われています。

この章では、このCdSセルを使って、外部の状態(周囲の明るさ)をマイコンの動作に反映させてみます。

(1)Atmel Studioで新しいプロジェクトを作成し、次のプログラムを打ち込みます。

```
;--------
; 2013/07/22 CdS_input
; 周囲の明るさをCdSセルで検出するLED点灯プログラム
; ATtiny2313 1MHz
;--------

;-------- アセンブラプログラムの決まりごと --------
;(1) ';(セミコロン)'はコメント記号です。行の中に現れると、アセンブラは
;      セミコロンより後ろを無視します。
;(2) アセンブラは名前や命令の大文字小文字の違いを無視します。
;      大文字小文字どちらでプログラムを書いてもかまいません。
;      ※データ中の大文字小文字は区別されます。
;      ※コメント以外の場所では、全角文字は使えません(エラーになります)
;(3) '.include'など、頭に'.(ピリオド)'がついているのは開発環境に対する
;      作業指示を意味する擬似命令です。
;(4) 「main:」など、名前の後ろに':(コロン)'があるのはラベルです。
;      ジャンプやサブルーチンコール命令の飛び先を指定するのに使います。
;--------
      .include "tn2313def.inc"  ;ATtiny2313用定義ファイルを取り込みます
                                ;このファイルの中で、I/Oレジスタの名前などが
                                ;定義されています。
;--------
; 割り込みベクタ領域 (ATtiny2313の場合、0x0000番地～0x0012番地)
; 使う割り込みの頭のコメント記号(セミコロン)を外して使います。
;--------
.org 0x00    rjmp reset          ;各種リセット : 「reset:」というラベルのところに
                                 ;ジャンプします。
;.org 0x01    rjmp ext_int0       ;外部割り込み要求0
;.org 0x02    rjmp ext_int1       ;外部割り込み要求1
;.org 0x03    rjmp tim1_capt      ;タイマ/カウンタ1捕獲(キャプチャ)発生
;.org 0x04    rjmp tim1_compa     ;タイマ/カウンタ1比較A一致
;.org 0x05    rjmp tim1_ovf       ;タイマ/カウンタ1オーバーフロー
;.org 0x06    rjmp tim0_ovf       ;タイマ/カウンタ0オーバーフロー
;.org 0x07    rjmp usart_rxt      ;usart受信完了
;.org 0x08    rjmp usart_udre     ;usart送信バッファ空
;.org 0x09    rjmp uasrt_tx       ;usart送信完了
;.org 0x0a    rjmp ana_comp       ;アナログ比較器出力遷移
;.org 0x0b    rjmp pcint          ;ピン変化割り込み要求
;.org 0x0c    rjmp tim1_compb     ;タイマ/カウンタ1比較B一致
;.org 0x0d    rjmp tim0_compa     ;タイマ/カウンタ0比較A一致
;.org 0x0e    rjmp tim0_compb     ;タイマ/カウンタ0比較B一致
;.org 0x0f    rjmp usi_strt       ;usi開始条件検出
;.org 0x10    rjmp usi_ovf        ;usiカウンタオーバーフロー
;.org 0x11    rjmp ee_edy         ;eeprom操作可
;.org 0x12    rjmp wdt_ovf        ;ウォッチドッグ時計完了
;--------
; ※使わない割り込みベクタは書かなくてもかまいませんが、
; スペースはあけておきます
; (ATtiny2313の場合、プログラムを0x0013番地から書くようにします)
;--------
```

次のページに続きます。

前のページの続きです。

```
        .org 0x13                   ; .org <アドレス>は、アセンブラに命令を置く
                                    ; プログラムアドレスを指示するための擬似命令です。
                                    ; この場合、.org擬似命令に続くプログラムは、
                                    ; プログラムメモリの0x0013番地以降に置かれます。
                                    ; 頭が「0x」ではじまる数値は16進数です。

;--------
; リセット時の処理
; (1)スタックポインタの初期化
;    ATtiny2313の場合、RAMEND(内部SRAMの最終番地)は0x00df番地です。
;     RAMENDの値は、"tn2313def.inc"中で定義されています
;--------
reset:      ldi     r16,low(RAMEND)     ;「low」は16ビットの定数の下位8ビットを
                                        ;取り出すための演算子です。
                                        ;ビルドすると「ldi  r16,0xdf」になります。
            out     SPL,r16
;--------
; (2)ポートの初期化
;--------
            ldi     r16,0b11111111      ;ポートB(PB0～PB7)を
            out     DDRB,r16            ;全ビット出力モードにする
;---- CdSセルのつながっているPD0を入力モードにする
            ldi     r16,0b11111110      ;CdSセルのつながっているPD0を
            out     DDRD,r16            ;入力モードにする(他は出力モード)
            ldi     r16,0b11111111      ;ポートA(PA1～PA0)を
            out     DDRA,r16            ;全ビット出力モードにする
                                        ;頭が「0bではじまる数値は2進数です。
;--------
;mainループ : ここに繰り返し実行させる内容を書きます
; ※このチュートリアルの回路では、点灯させたいLEDのポートを「L」にする
; (PORTレジスタの対応するビットを「0」にする)とLEDが点灯します。
;--------
main:       in      r16,PIND            ; CdSセルの状態を入力
            andi    r16,0b00000001      ; PD0の状態のみをandi命令で取り出す
            cpi     r16,0               ; 周囲が暗い(抵抗値が高い)ときはr16=0
            breq    main_1
;--------
; 周囲が明るい(CdSセルの抵抗値が低い)ときはLEDを消灯させます
;--------
            ldi     r16,0b11111111      ; LEDを消灯
            out     PORTB,r16
            rjmp    main
;--------
; 周囲が暗い(CdSセルの抵抗値が高い)ときはLEDを点滅させます
;--------
main_1:
            ldi     r16,0b11111110      ; PB0のLEDを点灯させます(r16=0b11111110)
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11111101      ; PB1のLEDを点灯させます(r16=0b11111101)
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11111011      ; PB2のLEDを点灯させます(r16=0b11111011)
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
            ldi     r16,0b11110111      ; PB3のLEDを点灯させます(r16=0b11110111)
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait          ; 時間待ち
;
            ldi     r16,0b11101111      ; PB4のLEDを点灯させます(r16=0b11101111)
            out     PORTB,r16           ; PORTBレジスタに出力
            rcall   wait                ; 時間待ち
;
```

次のページに続きます。

前のページの続きです。

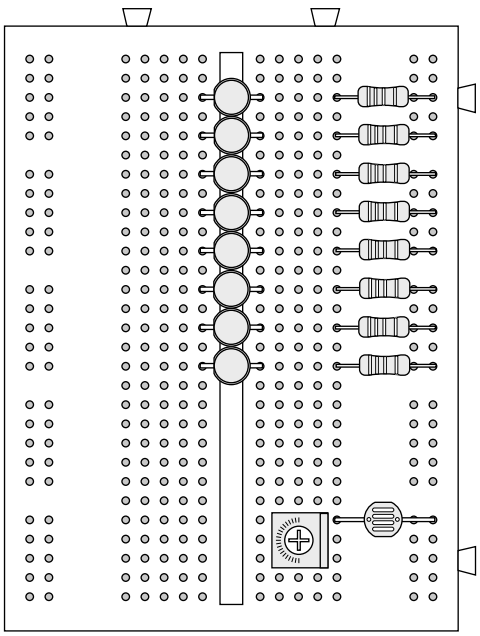
```
                ldi     r16,0b11011111      ; PB5のLEDを点灯させます(r16=0b11011111)
                out     PORTB,r16           ; PORTBレジスタに出力
                rcall   wait                ; 時間待ち
;
                ldi     r16,0b10111111      ; PB6のLEDを点灯させます(r16=0b10111111)
                out     PORTB,r16           ; PORTBレジスタに出力
                rcall   wait                ; 時間待ち
;
                ldi     r16,0b01111111      ; PB7のLEDを点灯させます(r16=0b01111111)
                out     PORTB,r16           ; PORTBレジスタに出力
                rcall   wait                ; 時間待ち
;
                rjmp    main                ; mainに飛びます(無限ループ)

;******** wait : 時間稼ぎ用ルーチン ********
wait:           ldi     r19,0x0f            ;r19 = 0x0f(10進で15)
wait1:          dec     r19                 ;r19を1減らし
                cpi     r19,0               ;r19が0になったかどうか比較
                breq    waitend             ;0だったら、waitendへジャンプ
                ldi     r17,0xff            ;r17を0xffにする
wait3:          dec     r17                 ;r17を1減らし
                cpi     r17,0               ;r17が0になったかどうか比較
                breq    wait1               ;r17が0だったら、wait2にジャンプ
                rjmp    wait3               ;r17が0でなければ、wait3にジャンプ
waitend:        ret                         ;サブルーチンからリターン
```

(2)プログラムを打ち込んだら、ビルドしてマイコンに書き込みます。

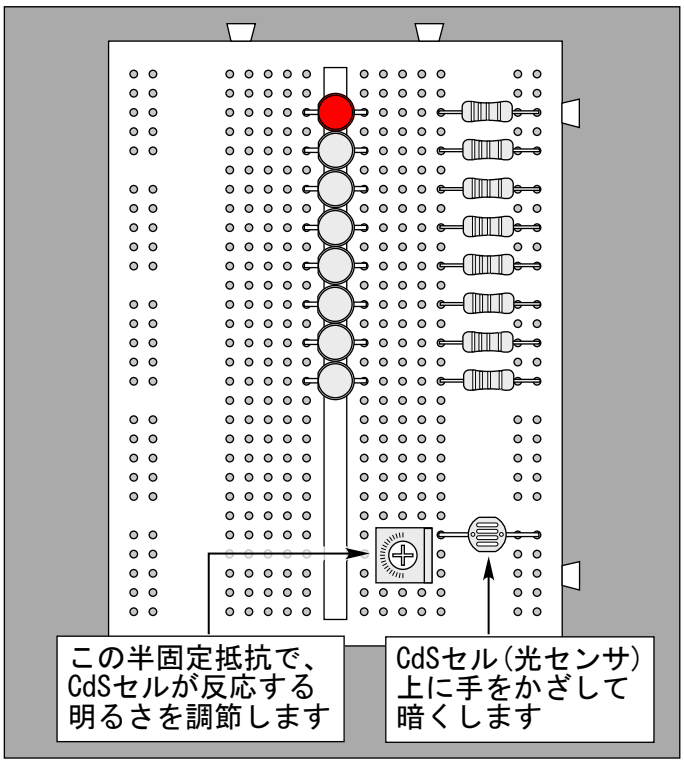
周囲が明るいときは、LEDが消灯しています。

CdSセルの上に手をかざして周囲の光をさえぎると、LEDの点滅がはじまります。

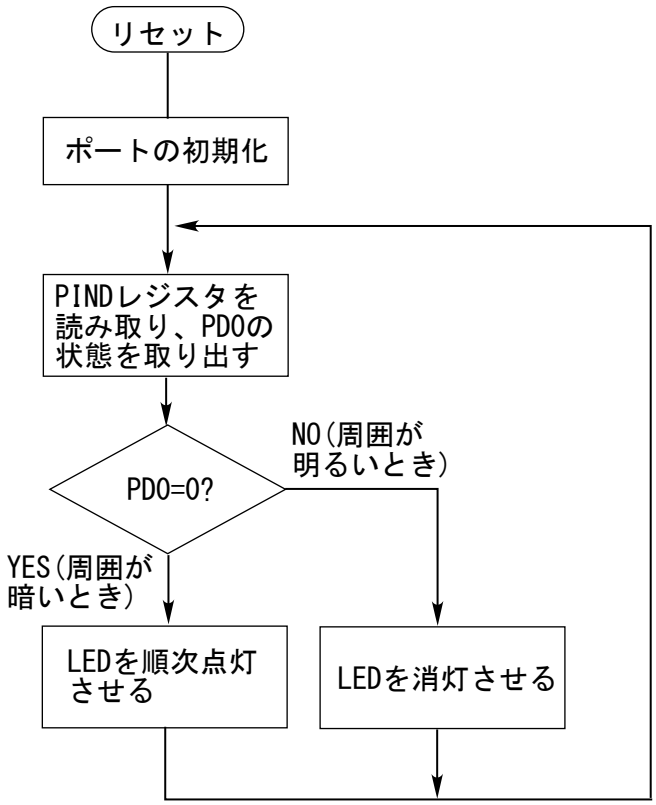
周囲が明るいとき



周囲が暗いとき



全体のフローチャート



CdSセルを使って、周囲の明るさに応じてマイコンの動作を変化させるプログラムの、全体のフローチャートは、左図のとおりです。

※全体のおおまかな動きがわかりやすいように、LEDの順次点灯の操作の詳細については省略しています。

フローチャート中のひし形の囲みは、条件に応じた動作の選択を表します。

(3)プログラムの解説(ポートの初期化)

13ページから15ページのLEDの順次点灯のプログラムでは、マイコンのI/Oポートを全部「出力」に設定しました。今回のプログラム(18ページから20ページ)では、PD0に接続されたCdSセルの状態をマイコンに入力したいので、次のように変更します。

```
;--------
; (2)ポートの初期化
;--------
            ldi     r16,0b11111111      ;ポートB(PB0～PB7)を
            out     DDRB,r16            ;全ビット出力モードにする
;---- CdSセルのつながっているPD0を入力モードにする
            ldi     r16,0b11111110      ;CdSセルのつながっているPD0を
            out     DDRD,r16            ;入力モードにする(他は出力モード)
            ldi     r16,0b11111111      ;ポートA(PA1～PA0)を
            out     DDRA,r16            ;全ビット出力モードにする
                                        ;頭が「0bではじまる数値は2進数です。
```

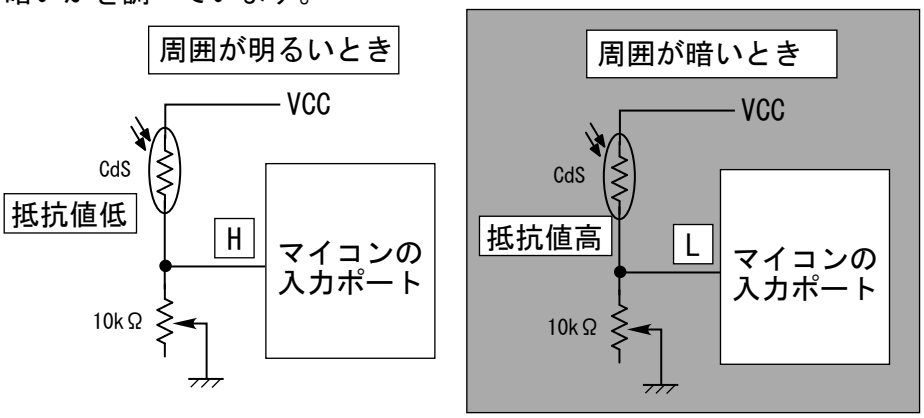
赤の点線で囲った部分が、変更した部分です。AVRマイコンのI/Oポート関係のレジスタは、PORTx、DDRx、PINx(xのところはA、B、Dのいずれかが入ります)の3つのレジスタが一組になっています。(6～7ページの説明を見てください)

ここでは、CdSセルの状態を見たいので、PD0を入力に設定します。そのため、DDRDレジスタのビット0を「0」に、残りのビットを「1」に設定しています。DDRxレジスタ中の、ビットが「1」に設定されたポートは出力に、ビットが「0」に設定されたポートは入力になります。
(6～7ページの説明を見てください)

(4)プログラムの解説(条件判断)

CdSセルは、本節のはじめに説明したように、光が当たったときには抵抗値が低く、光が当たっていないときには抵抗値が高くなる抵抗です。

このチュートリアルのブレッドボードでは、CdSセルを下図のように使って、周囲が明るいか暗いかを調べています。



周囲が明るいときはCdSセルの抵抗が低いので、マイコンの入力ポートは「H」レベルになります。
周囲が暗くなると、CdSセルの抵抗が高くなり、マイコンの入力ポートは「L」レベルになります。

CdSセルの状態をマイコンに取り込んで条件判断するために、プログラム中では次のようにします。

```
main:       in    r16,PIND          ; CdSセルの状態を入力
            andi  r16,0b00000001    ; PD0の状態のみをandi命令で取り出す
            cpi   r16,0             ; 周囲が暗い(抵抗値が高い)ときはr16=0
            breq  main_1
;--------
; 周囲が明るい(CdSセルの抵抗値が低い)ときはLEDを消灯させます
;--------
            ldi   r16,0b11111111    ; LEDを消灯
            out   PORTB,r16
            rjmp  main
;--------
; 周囲が暗い(CdSセルの抵抗値が高い)ときはLEDを点滅させます
;--------
main_1:
            (以下省略)
```

「in　　r16,PIND」で、マイコンのI/Oポートの、実際のピンの状態を、r16に読み取っています。
次の「andi　　r16,0b00000001」命令は、r16に取り込まれたPINDの状態と、定数「0b00000001」のANDをとっています。ANDすると、両方ともに「1」ならば結果は「1」に、どちらか一方でも「0」ならば結果は「0」になります。

ANDの真理値表

入力A	入力B	結果
0	0	0
0	1	0
1	0	0
1	1	1

※このプログラムでは、PINDのビット0の状態だけを見たいので、0b00000001とのANDをとっています。
次の「cpi　　r16,0」は、直前のandi命令の結果が「0である」か、「0でないか」を、0と比較することで調べています。
次の「breq　　main_1」で、比較の結果、r16が「0である」場合(周囲が暗い場合)に、LEDの順次点滅部分にジャンプさせています。

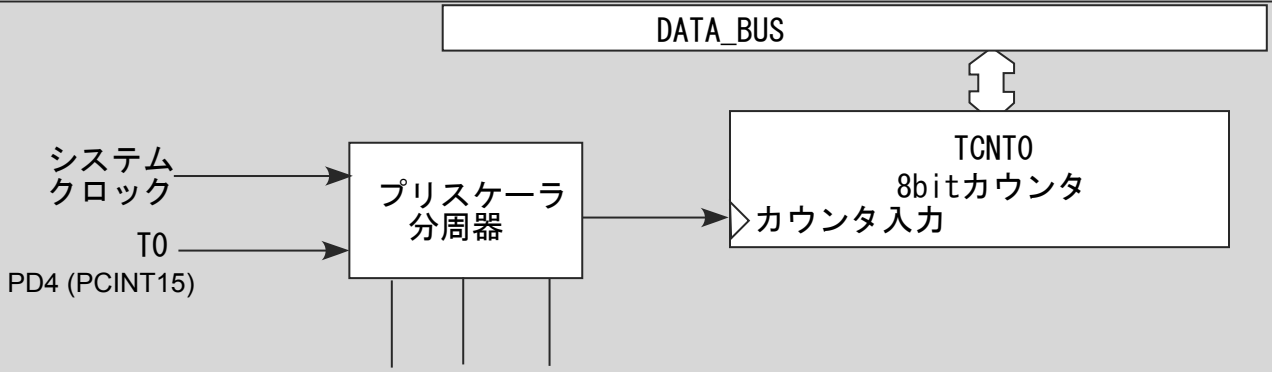
周囲が明るい場合、「cpi　　r16,0」の結果は「0でない」ので、breq命令は実行されず、そのまま次の命令に進みます。
(LEDを消灯します)

4.タイマカウンタ機能と割り込み

AVRマイコン(ATtiny2313)には、デジタルI/Oポート以外にも、多彩な周辺機能が搭載されています。この章では、その中から、タイマカウンタ0の簡単な使い方を、割り込み機能と合わせて説明します。

4.1.タイマカウンタ0の機能設定

タイマカウンタ0には文字とおり8ビットのカウンタが内蔵されており、これにクロックを与えると、カウントが始まります。カウンタの値は、読み書きできます。値がオーバーフローする(カウンタの値が8ビットで表現できる最大値の255を超えて0に戻る)と、再び0から数えます。
オーバーフローするとI/OレジスタTIFR0のTOV0ビットが1になります。



CS02	CS01	CS00	
0	0	0	タイマカウンタストップ
0	0	1	CKI/O
0	1	0	CKI/O/8
0	1	1	CKI/O/64
1	0	0	CKI/O/256
1	0	1	CKI/O/1024
1	1	0	T0ピン立ち下がりエッジ
1	1	1	T0ピン立ち上がりエッジ

I/O アドレス	レジスタ名	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x39	TIMSK	TOIE1	OCIE1A	OCIE1B	-	ICIE1	OCIE0B	TOIE0	OCIE0A
0x38	TIFR	TOV1	OCF1A	OCF1B	-	ICF1	OCF0B	TOV0	OCF0A
0x33	TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
0x32	TCNT0	タイマカウンタの値 (8ビット)							

※コンペア関係のレジスタは省略してあります。詳細についてはATtiny2313のデータシートを参照してください。

AVRマイコンのタイマカウンタは多機能で、オーバーフロー割り込みや、あらかじめ設定しておいた値と一致を検出するコンペア機能などがあります。ここでは、タイマカウンタ0がオーバーフローしたときに発生するオーバーフロー割り込みを使ってみます。
Atmel Studioを使って、次ページのプログラムを打ち込んでみてください。

```
;--------
; 2013/07/19 TOVO_TEST
; LED点灯プログラム(タイマ0オーバーフロー割り込み版)
; ATtiny2313 1MHz
;--------

;-------- アセンブラプログラムの決まりごと --------
;(1) ';(セミコロン)'はコメント記号です。行の中に現れると、アセンブラは
;      セミコロンより後ろを無視します。
;(2) アセンブラは名前や命令の大文字小文字の違いを無視します。
;      大文字小文字どちらでプログラムを書いてもかまいません。
;      ※データ中の大文字小文字は区別されます。
;      ※コメント以外の場所では、全角文字は使えません(エラーになります)
;(3) '.include'など、頭に'.(ピリオド)'がついているのはアセンブラに対する
;      作業指示を意味する擬似命令です。
;(4) 「main:」など、名前の後ろに':(コロン)'があるのはラベルです。
;      ジャンプやサブルーチンコール命令の飛び先を指定するのに使います。
;--------
      .include "tn2313def.inc"  ;ATtiny2313用定義ファイルを取り込みます
                                ;このファイルの中で、I/Oレジスタの名前などが
                                ;定義されています。

      .equ   rcunt = 0x60       ;ATtiny2313の内部SRAM領域は、RAMアドレス0x60
                                ;からはじまります。ここでは、RAMアドレス0x60番地
                                ;に、rcuntという名前をつけています。
;--------
; 割り込みベクタ領域 (ATtiny2313の場合、0x0000番地～0x0012番地)
; 使う割り込みの頭のコメント記号(セミコロン)を外して使います。
;--------
.org 0x00    rjmp reset           ;各種リセット : 「reset:」というラベルのところに
                                  ;ジャンプします。
;.org 0x01   rjmp ext_int0        ;外部割り込み要求0
;.org 0x02   rjmp ext_int1        ;外部割り込み要求1
;.org 0x03   rjmp tim1_capt       ;タイマ/カウンタ1捕獲(キャプチャ)発生
;.org 0x04   rjmp tim1_compa      ;タイマ/カウンタ1比較A一致
;.org 0x05   rjmp tim1_ovf        ;タイマ/カウンタ1オーバーフロー
.org 0x06    rjmp tim0_ovf        ;タイマ/カウンタ0オーバーフロー
;.org 0x07   rjmp usart_rxt       ;usart受信完了
;.org 0x08   rjmp usart_udre      ;usart送信バッファ空
;.org 0x09   rjmp uasrt_tx        ;usart送信完了
;.org 0x0a   rjmp ana_comp        ;アナログ比較器出力遷移
;.org 0x0b   rjmp pcint           ;ピン変化割り込み要求
;.org 0x0c   rjmp tim1_compb      ;タイマ/カウンタ1比較B一致
;.org 0x0d   rjmp tim0_compa      ;タイマ/カウンタ0比較A一致
;.org 0x0e   rjmp tim0_compb      ;タイマ/カウンタ0比較B一致
;.org 0x0f   rjmp usi_strt        ;usi開始条件検出
;.org 0x10   rjmp usi_ovf         ;usiカウンタオーバーフロー
;.org 0x11   rjmp ee_edy          ;eeprom操作可
;.org 0x12   rjmp wdt_ovf         ;ウォッチドッグ時計完了
;--------
; ※使わない割り込みベクタは書かなくてもかまいませんが、
; スペースはあけておきます
; (ATtiny2313の場合、プログラムを0x0013番地から書くようにします)
;--------
```

次のページに続きます。

前のページの続きです。

```
      .org 0x13                 ; .org <アドレス>は、アセンブラに命令を置く
                                ; プログラムアドレスを指示するための擬似命令です。
                                ; この場合、.org擬似命令に続くプログラムは、
                                ; プログラムメモリの0x0013番地以降に置かれます。
                                ; 頭が「0x」ではじまる数値は16進数です。

;--------
; タイマ0オーバーフロー割り込み処理
; タイマ0のとりうる値は0～255までです。タイマ0の値が255の
; ときカウント入力が入ると、タイマ0の値は0に戻り、TIFRレジスタの
; TOV0ビットが立ちます。(オーバーフロー)
; このとき、割り込みが許可されていれば、このルーチンに飛んできます。
;--------
tim0_ovf:
            push  r16           ;割り込み処理中に使うレジスタと
            in    r16,SREG      ;フラグレジスタをスタックに保存(プッシュ)
            push  r16
;--------
            lds   r16,rcunt
            cpi   r16,0
            brne  tim0_ovf1
            ldi   r16,0b11111110      ; PB0のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
;
tim0_ovf1:
            cpi   r16,1
            brne  tim0_ovf2
            ldi   r16,0b11111101      ; PB1のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
;
tim0_ovf2:
            cpi   r16,2
            brne  tim0_ovf3
            ldi   r16,0b11111011      ; PB2のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
;
tim0_ovf3:
            cpi   r16,3
            brne  tim0_ovf4
            ldi   r16,0b11110111      ; PB3のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
;
tim0_ovf4:
            cpi   r16,4
            brne  tim0_ovf5
            ldi   r16,0b11101111      ; PB4のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
;
tim0_ovf5:
            cpi   r16,5
            brne  tim0_ovf6
            ldi   r16,0b11011111      ; PB5のLEDを点灯させます
            out   PORTB,r16           ; PORTBレジスタに出力
            rjmp  tim0_ovf8
```

次のページに続きます。

前のページの続きです。

```
;
tim0_ovf6:
            cpi    r16,6
            brne   tim0_ovf7
            ldi    r16,0b10111111      ; PB6のLEDを点灯させます
            out    PORTB,r16           ; PORTBレジスタに出力
            rjmp   tim0_ovf8
;
tim0_ovf7:
            ldi    r16,0b01111111      ; PB7のLEDを点灯させます
            out    PORTB,r16           ; PORTBレジスタに出力
;--------
tim0_ovf8:
            lds    r16,rcunt           ;内部SRAM上のデータはlds命令で
            inc    r16                 ;汎用レジスタにコピーし、処理してから
            sts    rcunt,r16           ;内部SRAMにコピーする
            cpi    r16,8               ;コピーしても、コピー元のデータは変わらない
            brne   tim0_ovf_ex
            ldi    r16,0
            sts    rcunt,r16
;--------
tim0_ovf_ex:
            ldi    r16,256-100         ;TCNT0に値をセット
            out    TCNT0,r16
;--------
            pop    r16                 ;スタックに保存してあったフラグレジスタと
            out    SREG,r16            ;割り込み処理中に使用したレジスタを復帰(ポップ)
            pop    r16
            reti                       ;割り込みから戻る(作業再開)
;--------

;--------
; リセット時の処理
; (1)スタックポインタの初期化
;    ATtiny2313の場合、RAMEND(内部SRAMの最終番地)は0x00df番地です。
;     RAMENDの値は、"tn2313def.inc"中で定義されています
;--------
reset:      ldi    r16,low(RAMEND)     ;「low」は16ビットの定数の下位8ビットを
                                       ;取り出すための演算子です。
                                       ;ビルドすると「ldi  r16,0xdf」になります。
            out    SPL,r16
;--------
; (2)ポートの初期化
;--------
            ldi    r16,0b11111111      ;ポートB(PB0～PB7)を
            out    DDRB,r16            ;全ビット出力モードにする
            ldi    r16,0b11111111      ;ポートD(PD0～PD6)を
            out    DDRD,r16            ;全ビット出力モードにする
            ldi    r16,0b11111111      ;ポートA(PA1～PA0)を
            out    DDRA,r16            ;全ビット出力モードにする
;--------
; (3)タイマ0の初期化
;--------
            ldi    r16,0x05            ;1/1024分周
            out    TCCR0B,r16
            ldi    r16,256-100         ;TCNT0に値をセット
            out    TCNT0,r16
            ldi    r16,0x02            ;タイマ0オーバーフロー割り込みを許可
            out    TIMSK,r16
;--------
; (4)内部データSRAMの初期化
;--------
            ldi    r16,0x00            ; rcuntを0にする
            sts    rcunt,r16
```

次のページに続きます。

前のページの続きです。

```
;--------
            sei                                 ;マイコン全体の割り込みを許可
;--------
;mainループ : ここに繰り返し実行させる内容を書きます
;--------
main:
            rjmp   main                         ;mainにジャンプ(何もしません)
```

このプログラムをビルドしてマイコンに書き込むと、mainループの中で何もしていないにもかかわらず、LEDが点滅します。

タイマカウンタ0は、リセット直後は停止しています。タイマカウンタ0の設定レジスタ、TCCR0Bに、タイマカウンタの分周比を設定すると(設定値については上記プログラムと17ページの表を見てください)、カウントアップを開始します。

この例では、1MHzのマイコンのクロックを1024分周していますので、約1m秒ごとに1ずつカウントアップします。

タイマカウンタ0は8ビット幅ですので、0から255までの値をとります。もしタイマカウンタ0の値が255のときにカウントアップすると、再び0に戻ります。このとき、タイマフラグレジスタ、TIFRレジスタ中の、TOV0ビットが「1」になり、オーバーフローしたことを表示します。

このとき、タイマ割り込み許可レジスタ、TIMSKのTOIE0ビットが「1」にセットされており、マイコン全体の割り込みが「sei」命令によって許可されていると、タイマカウンタ0がオーバーフローしたときに割り込みを発生します。

※タイマフラグレジスタ、TIFRレジスタ中のTOV0ビットは、割り込みが発生すると自動的にクリアされます。

4.2.割り込みについて

はじめに少し説明したように、マイコンのプログラムは、プログラムカウンタの値にしたがって、順番に1個ずつ「命令フェッチ->解読実行」されます。

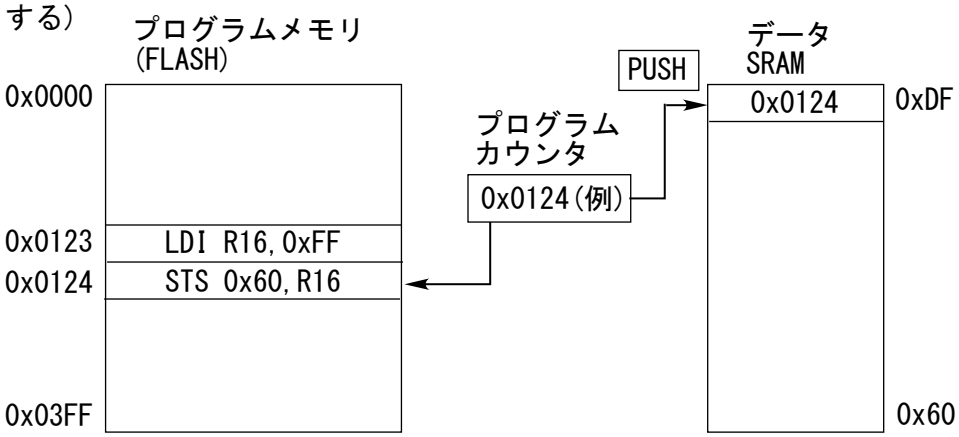
割り込み機能を使うと、24～27ページのプログラム例のように、マイコンのプログラムとは別の出来事によって、急ぎの仕事を行わせることができます。

AVRマイコンの場合、割り込みが発生して受け付けられると、今の命令の実行が終わり次第、次の動作が自動的に行われます。

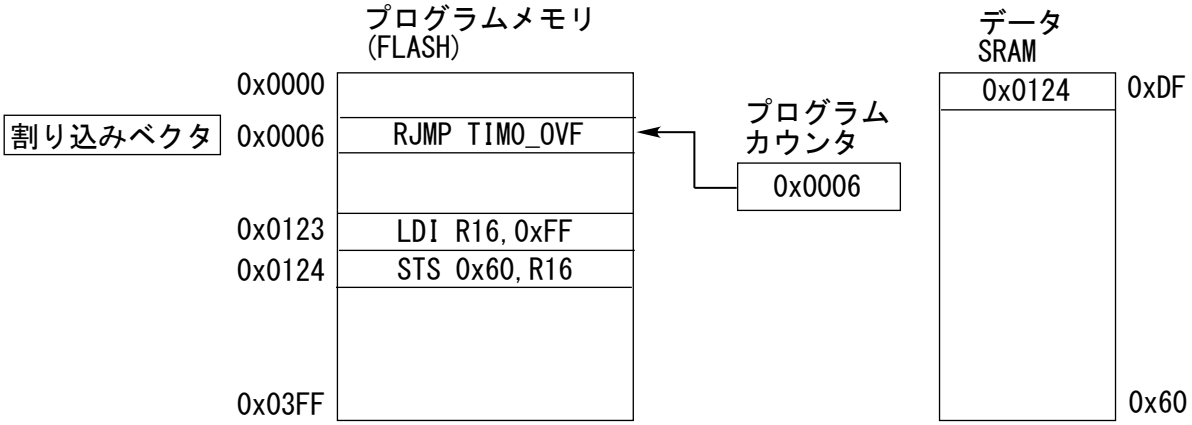
この例では、0x0123番地の命令を実行中に、タイマカウンタ0オーバーフロー割り込みが発生したとします。

(1)割り込み処理中にほかの割り込みが入らないように、以後の割り込みを禁止する

(2)次の命令のプログラムカウンタの値を、スタック(データSRAM上に確保された、データ一時置き場で、データを一時的に積み上げて置いておくことができる)に積んで保存する(PUSHする)

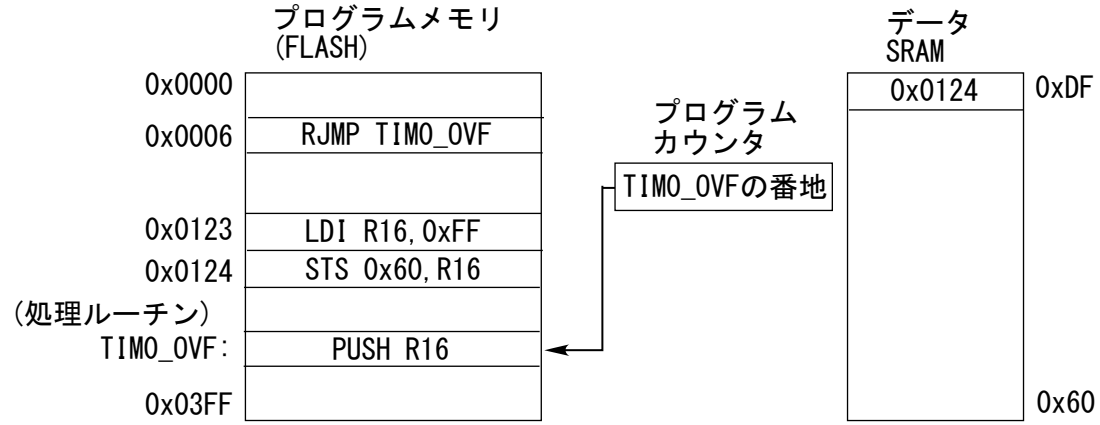


(3)受け付けられた割り込み要因に応じた割り込みベクタの番地が、プログラムカウンタに書き込まれる。(プログラムメモリ上の割り込みベクタの番地にジャンプします)
割り込みベクタ領域は、ATtiny2313の場合、プログラムメモリの0x0000番地からはじまっています。プログラムメモリ上のどの割り込みベクタにジャンプするかは、割り込みの要因別に、すでに決まっています。(ユーザ側では変えられません)



※この例では、タイマ0のオーバーフロー割り込みのベクタ(プログラムメモリ上の0x0006番地)にジャンプしています。

(4)この割り込みベクタの番地には、通常は割り込み処理プログラムへのジャンプ命令(RJMP)が置かれているので、割り込み処理ルーチンにジャンプする。
割り込み処理ルーチンは、割り込みベクタ領域(ATtiny2313の場合、0x0000番地から0x0012番地)以外であれば、プログラムメモリ上のどの番地にでも置くことができます。
(プログラムを書くユーザが自由に決められます)



この(1)～(4)の一連の動作は、割り込みが発生し、受け付けられたときに自動的に行われます。
※リセットだけは例外で、常に受け付けられます。また、プログラムカウンタは保存されません。

重要

次の場合は、タイマオーバーフローなどの割り込みのもととなる事象が発生しても割り込みとして受け付けられず、保留されます。
(1)割り込み処理ルーチンを実行中のとき:
　　ATtiny2313では、割り込みの処理中はほかの割り込みは受け付けられません。
(2)SREG(演算結果などのフラグを保持するI/Oレジスタ)中のIビットが「0」のとき:
　　SREG中のIビットをSEI命令で「1」にしないと割り込みが受け付けられません。
(3)各周辺機能のI/Oレジスタ設定で、割り込みを発生するように設定していないとき

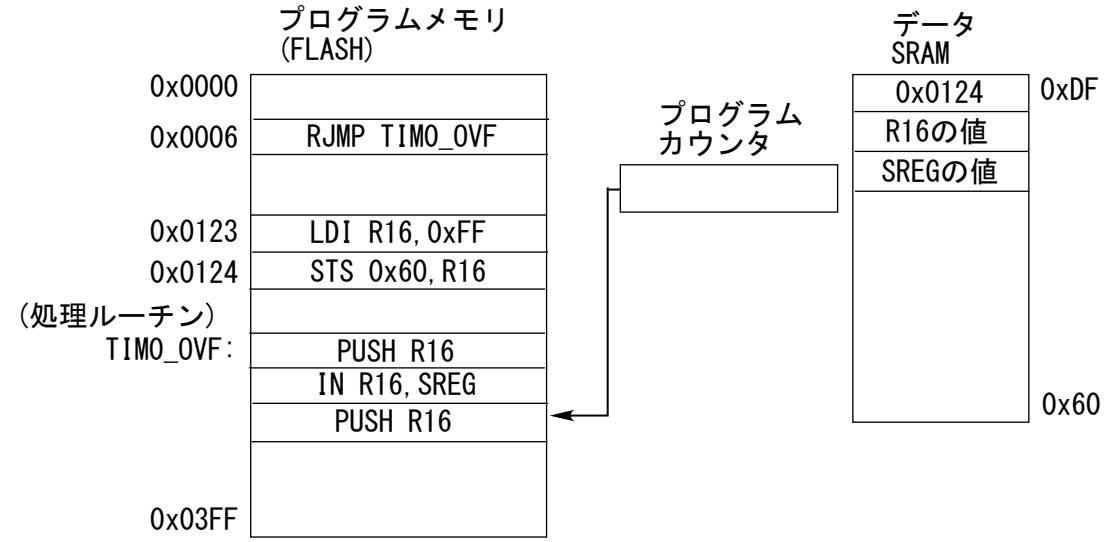
割り込みは、メインのプログラムの実行状況とは関係なく発生し、いつ起こるかわからないので、割り込みの処理を行うときには、必ず次の操作を行います。
この操作は自動的には行ってくれないので、ユーザのプログラム中で行います。

(5) 割り込み処理ルーチン中で使用する汎用レジスタをスタックに積んで保存する。
フラグレジスタ(I/Oレジスタ中のSREGレジスタ)も、割り込み処理中で演算などで変化するので、スタックに積んで保存する。

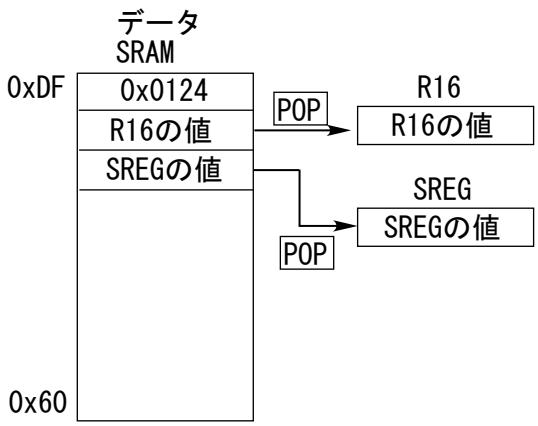
(例)

```
tim0_ovf:    push   r16          ; r16を割り込み処理中に使うのでプッシュ
             in     r16,SREG     ; フラグレジスタは直接メモリに保存できないので、
             push   r16          ; r16にコピーして、r16をプッシュする
```

もし割り込み処理中で使う汎用レジスタを保存していなければ、割り込み処理プログラム中で汎用レジスタの値が変更されてしまいますので、必ず保存してください。



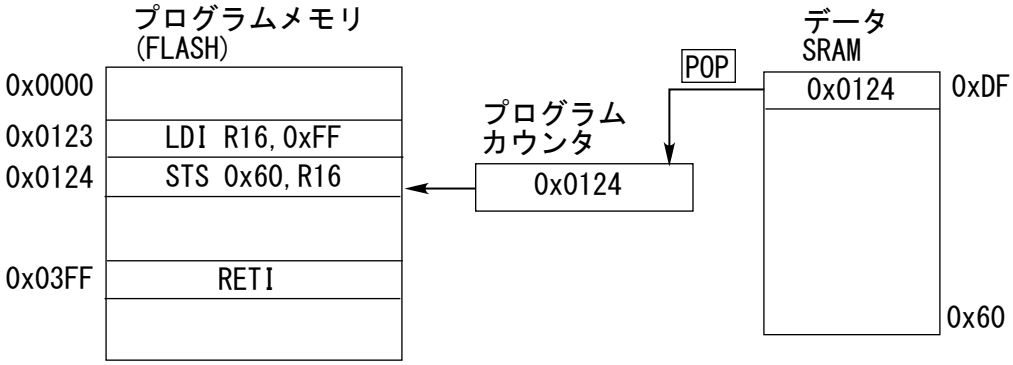
割り込み処理ルーチンが終わると、次の操作をしてもとのプログラムに戻ります。



(6) スタックに一時保存しておいたフラグレジスタ(SREG)と汎用レジスタを、POP命令でもとに戻す

割り込みから戻るときに汎用レジスタをPOPするときは、割り込みに入ったときにPUSHした順序と逆の順序でPOPします。

(7) 「割り込みからのリターン(RETI)」命令を実行すると、スタックに保存されていたプログラムカウンタの値がプログラムカウンタに書き戻され、中断されていたメインのプログラムの実行が再開される。



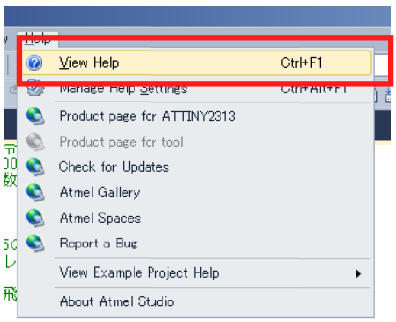
資料篇

(1)よく使われる、主な命令とその操作の概要

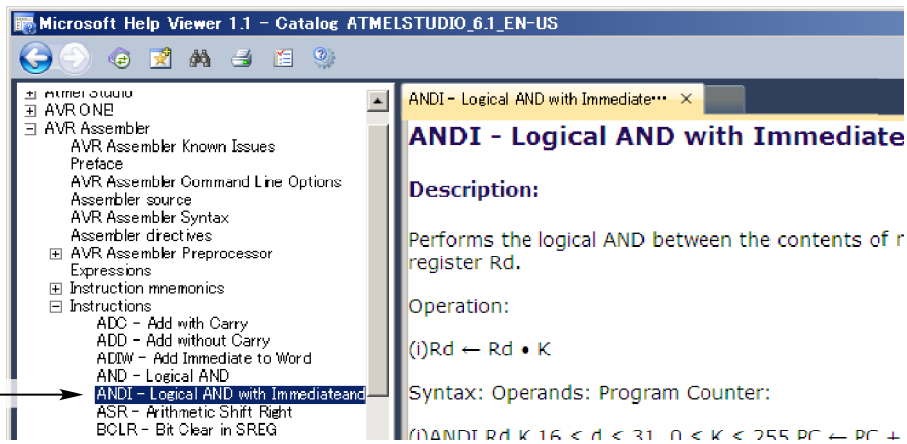
ATtiny2313でよく使われる命令の一覧とその概要を説明します。

	命令	動作の概要
1	LDI Rd,K	汎用レジスタRdに、定数Kをコピーする
2	MOV Rd,Rr	汎用レジスタRrの値を、汎用レジスタRdにコピーする
3	OUT P,Rr	汎用レジスタRrの値を、入出力レジスタPにコピーする
4	IN Rd,P	入出力レジスタPの値を、汎用レジスタRdにコピーする
5	LDS Rd,MEM	データSRAMのMEM番地の値を、汎用レジスタRdにコピーする
6	STS MEM,Rr	汎用レジスタRrの値を、データSRAMのMEM番地にコピーする
7	INC Rd	汎用レジスタRdの値を、1増やす(インクリメントする)
8	DEC Rd	汎用レジスタRdの値を、1減らす(デクリメントする)
9	LSL Rd	汎用レジスタRdのビットを、1だけ左にシフトする。 はみ出したビットは、Cフラグに反映される
10	LSR Rd	汎用レジスタRdのビットを、1だけ右にシフトする。 はみ出したビットは、Cフラグに反映される
11	COM Rd	汎用レジスタRdのビットを反転する
12	ANDI Rd,K	汎用レジスタRdのビットと、定数KのビットのANDをとり、 汎用レジスタRdに入れる
13	ORI Rd,K	汎用レジスタRdのビットと、定数KのビットのORをとり、 汎用レジスタRdに入れる
14	CPI Rd,K	汎用レジスタRdの値と、定数Kを比較する
15	BREQ LABEL	比較などの演算結果が0のとき、ラベルで指定されたアドレスにジャンプする
16	BRNE LABEL	比較などの演算結果が0でないとき、ラベルで指定されたアドレスにジャンプする
17	RJMP LABEL	ラベルで指定されたアドレスに無条件でジャンプする
18	RCALL LABEL	ラベルで指定されたアドレスのサブルーチンを呼び出す
19	RET	呼び出されたサブルーチンから戻る
20	LPM Rd,Z	Zレジスタで指定された、プログラムメモリ中のデータを、 汎用レジスタRdにコピーする

※ATtiny2313の全命令の詳細については、Atmel Studioのヘルプで調べることができますので、参照してください。



Atmel Studioの[Help]メニュー中の[View Help]をクリックすると、ヘルプ機能が立ち上がります。



調べたい命令をクリック

ヘルプ機能左側のウインドウ中の[AVR Assembler]をクリックし、[Instructions]をクリックすると、命令の一覧が出ます。調べたい命令をクリックすると、その命令の詳細が見られます。

ATtiny2313 全命令の概要 (1/2)
(ATtiny2313のデータシートより引用)

命令表の見かた

(1)「Operands」欄の「Rd」「Rr」は、汎用レジスタを、「K」は数値(定数)を表します。

(2)「FLAGS」欄は、ステータスレジスタ(SREG)中のフラグの中で、どのフラグが変更されるかを表します。

(3) 「#CLOCKS」欄は、命令を実行するのに必要なクロック数です。

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1

ATtiny2313 全命令の概要 (2/2)
(ATtiny2313のデータシートより引用)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

(2)よく使う擬似命令一覧

.include "**filename**" :

ソースプログラム中で、別のファイルを取り込みたいときに使用します。
filenameのところには、取り込みたいファイルの名前が入ります。
(例) .include "tn2313def.inc" ; ATtiny2313用レジスタ定義ファイルを取り込む

.org **address** :

この擬似命令の後ろの命令を、プログラムメモリ中の指定したアドレスから置くよう、
アセンブラに指示します。

```
(例) .org    0x0013              ; この後の命令をプログラムメモリの0x0013番地
                                 ; から配置する
```

.equ **name** = **式**

「式」で表される数値に、「name」で表される名前をつけます。
数値に名前をつけることで、プログラム中では名前を使って書くことができます。

```
(例) .equ    rcunt = 0x60        ; ATtiny2313の内部SRAMの先頭番地(0x60)に、
                                 ; "rcunt"と名前をつけた
                lds    r16,rcunt ; rcuntという名前で、0x60という値を使うことができる
```

.db **data0,data1,....**

プログラムメモリ中に、指定したデータを置くときに使用します。
このチュートリアルでは説明しませんでしたが、置かれたデータはLPM命令で汎用レジスタに
コピーすることができます。
※1行中のデータの個数は偶数個でないと、ビルド時に警告が出ます。

```
(例) .db     0x00,0x01,0x02,0x03      ;プログラムメモリ中に、0x00,0x01,0x02,0x03
                                      ;というデータを置く
```

※これ以外にもたくさんの擬似命令がありますが、よく使うもののみを取り上げました。
詳細については、Atmel Studioのヘルプで[Assembler Directives]を選ぶと見ることが
できますので、参考にしてください。

(3) ATtiny2313 I/Oレジスタマップ
(ATtiny2313のデータシートより引用)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	8
0x3E (0x5E)	Reserved	–	–	–	–	–	–	–	–	
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
0x3C (0x5C)	OCR0B	Timer/Counter0 – Compare Register B								85
0x3B (0x5B)	GIMSK	INT1	INT0	PCIE0	PCIE2	PCIE1	–	–	–	50
0x3A (0x5A)	GIFR	INTF1	INTF0	PCIF0	PCIF2	PCIF1	–	–	–	51
0x39 (0x59)	TIMSK	TOIE1	OCIE1A	OCIE1B	–	ICIE1	OCIE0B	TOIE0	OCIE0A	86, 115
0x38 (0x58)	TIFR	TOV1	OCF1A	OCF1B	–	ICF1	OCF0B	TOV0	OCF0A	86, 115
0x37 (0x57)	SPMCSR	–	–	RSIG	CTPB	RFLB	PGWRT	PGERS	SPMEN	176
0x36 (0x56)	OCR0A	Timer/Counter0 – Compare Register A								85
0x35 (0x55)	MCUCR	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00	36, 50, 68
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	44
0x33 (0x53)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	84
0x32 (0x52)	TCNT0	Timer/Counter0 (8-bit)								85
0x31 (0x51)	OSCCAL	–	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	30
0x30 (0x50)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	81
0x2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	110
0x2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	112
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								114
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								114
0x2B (0x4B)	OCR1AH	Timer/Counter1 – Compare Register A High Byte								114
0x2A (0x4A)	OCR1AL	Timer/Counter1 – Compare Register A Low Byte								114
0x29 (0x49)	OCR1BH	Timer/Counter1 – Compare Register B High Byte								114
0x28 (0x48)	OCR1BL	Timer/Counter1 – Compare Register B Low Byte								114
0x27 (0x47)	Reserved	–	–	–	–	–	–	–	–	
0x26 (0x46)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	30
0x25 (0x45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								114
0x24 (0x44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								114
0x23 (0x43)	GTCCR	–	–	–	–	–	–	–	PSR10	118
0x22 (ox42)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	113
0x21 (0x41)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	44
0x20 (0x40)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	53
0x1F (0x3F)	Reserved	–	–	–	–	–	–	–	–	
0x1E (0x3E)	EEAR	–	EEPROM Address Register							22
0x1D (0x3D)	EEDR	EEPROM Data Register								22
0x1C (0x3C)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	22
0x1B (0x3B)	PORTA	–	–	–	–	–	PORTA2	PORTA1	PORTA0	68
0x1A (0x3A)	DDRA	–	–	–	–	–	DDA2	DDA1	DDA0	68
0x19 (0x39)	PINA	–	–	–	–	–	PINA2	PINA1	PINA0	69
0x18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	69
0x17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	69
0x16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	69
0x15 (0x35)	GPIOR2	General Purpose I/O Register 2								23
0x14 (0x34)	GPIOR1	General Purpose I/O Register 1								23
0x13 (0x33)	GPIOR0	General Purpose I/O Register 0								23
0x12 (0x32)	PORTD	–	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	69
0x11 (0x31)	DDRD	–	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	69
0x10 (0x30)	PIND	–	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	69
0x0F (0x2F)	USIDR	USI Data Register								165
0x0E (0x2E)	USISR	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0	164
0x0D (0x2D)	USICR	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC	162
0x0C (0x2C)	UDR	UART Data Register (8-bit)								136
0x0B (0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	137
0x0A (0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	138
0x09 (0x29)	UBRRL	UBRRH[7:0]								140
0x08 (0x28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	167
0x07 (0x27)	BODCR	–	–	–	–	–	–	BODS	BODSE	37
0x06 (0x26)	PRR	–	–	–	–	PRTIM1	PRTIM0	PRUSI	PRUSART	36
0x05 (0x25)	PCMSK2	–	PCINT17	PCINT16	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	52
0x04 (0x24)	PCMSK1	–	–	–	–	–	PCINT10	PCINT9	PCINT8	52
0x03 (0x23)	UCSRC	UMSEL1	UMSEL0	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	139
0x02 (0x22)	UBRRH	–	–	–	–	UBRRH[11:8]				140
0x01 (0x21)	DIDR	–	–	–	–	–	–	AIN1D	AIN0D	168
0x00 (0x20)	USIBR	USI Buffer Register								166

(4)数値(10進、2進、16進)早見表

10進	2進 (0b)	16進 (0x)	10進	2進 (0b)	16進 (0x)	10進	2進 (0b)	16進 (0x)	10進	2進 (0b)	16進 (0x)
0	00000000	00	64	01000000	40	128	10000000	80	192	11000000	C0
1	00000001	01	65	01000001	41	129	10000001	81	193	11000001	C1
2	00000010	02	66	01000010	42	130	10000010	82	194	11000010	C2
3	00000011	03	67	01000011	43	131	10000011	83	195	11000011	C3
4	00000100	04	68	01000100	44	132	10000100	84	196	11000100	C4
5	00000101	05	69	01000101	45	133	10000101	85	197	11000101	C5
6	00000110	06	70	01000110	46	134	10000110	86	198	11000110	C6
7	00000111	07	71	01000111	47	135	10000111	87	199	11000111	C7
8	00001000	08	72	01001000	48	136	10001000	88	200	11001000	C8
9	00001001	09	73	01001001	49	137	10001001	89	201	11001001	C9
10	00001010	0A	74	01001010	4A	138	10001010	8A	202	11001010	CA
11	00001011	0B	75	01001011	4B	139	10001011	8B	203	11001011	CB
12	00001100	0C	76	01001100	4C	140	10001100	8C	204	11001100	CC
13	00001101	0D	77	01001101	4D	141	10001101	8D	205	11001101	CD
14	00001110	0E	78	01001110	4E	142	10001110	8E	206	11001110	CE
15	00001111	0F	79	01001111	4F	143	10001111	8F	207	11001111	CF
16	00010000	10	80	01010000	50	144	10010000	90	208	11010000	D0
17	00010001	11	81	01010001	51	145	10010001	91	209	11010001	D1
18	00010010	12	82	01010010	52	146	10010010	92	210	11010010	D2
19	00010011	13	83	01010011	53	147	10010011	93	211	11010011	D3
20	00010100	14	84	01010100	54	148	10010100	94	212	11010100	D4
21	00010101	15	85	01010101	55	149	10010101	95	213	11010101	D5
22	00010110	16	86	01010110	56	150	10010110	96	214	11010110	D6
23	00010111	17	87	01010111	57	151	10010111	97	215	11010111	D7
24	00011000	18	88	01011000	58	152	10011000	98	216	11011000	D8
25	00011001	19	89	01011001	59	153	10011001	99	217	11011001	D9
26	00011010	1A	90	01011010	5A	154	10011010	9A	218	11011010	DA
27	00011011	1B	91	01011011	5B	155	10011011	9B	219	11011011	DB
28	00011100	1C	92	01011100	5C	156	10011100	9C	220	11011100	DC
29	00011101	1D	93	01011101	5D	157	10011101	9D	221	11011101	DD
30	00011110	1E	94	01011110	5E	158	10011110	9E	222	11011110	DE
31	00011111	1F	95	01011111	5F	159	10011111	9F	223	11011111	DF
32	00100000	20	96	01100000	60	160	10100000	A0	224	11100000	E0
33	00100001	21	97	01100001	61	161	10100001	A1	225	11100001	E1
34	00100010	22	98	01100010	62	162	10100010	A2	226	11100010	E2
35	00100011	23	99	01100011	63	163	10100011	A3	227	11100011	E3
36	00100100	24	100	01100100	64	164	10100100	A4	228	11100100	E4
37	00100101	25	101	01100101	65	165	10100101	A5	229	11100101	E5
38	00100110	26	102	01100110	66	166	10100110	A6	230	11100110	E6
39	00100111	27	103	01100111	67	167	10100111	A7	231	11100111	E7
40	00101000	28	104	01101000	68	168	10101000	A8	232	11101000	E8
41	00101001	29	105	01101001	69	169	10101001	A9	233	11101001	E9
42	00101010	2A	106	01101010	6A	170	10101010	AA	234	11101010	EA
43	00101011	2B	107	01101011	6B	171	10101011	AB	235	11101011	EB
44	00101100	2C	108	01101100	6C	172	10101100	AC	236	11101100	EC
45	00101101	2D	109	01101101	6D	173	10101101	AD	237	11101101	ED
46	00101110	2E	110	01101110	6E	174	10101110	AE	238	11101110	EE
47	00101111	2F	111	01101111	6F	175	10101111	AF	239	11101111	EF
48	00110000	30	112	01110000	70	176	10110000	B0	240	11110000	F0
49	00110001	31	113	01110001	71	177	10110001	B1	241	11110001	F1
50	00110010	32	114	01110010	72	178	10110010	B2	242	11110010	F2
51	00110011	33	115	01110011	73	179	10110011	B3	243	11110011	F3
52	00110100	34	116	01110100	74	180	10110100	B4	244	11110100	F4
53	00110101	35	117	01110101	75	181	10110101	B5	245	11110101	F5
54	00110110	36	118	01110110	76	182	10110110	B6	246	11110110	F6
55	00110111	37	119	01110111	77	183	10110111	B7	247	11110111	F7
56	00111000	38	120	01111000	78	184	10111000	B8	248	11111000	F8
57	00111001	39	121	01111001	79	185	10111001	B9	249	11111001	F9
58	00111010	3A	122	01111010	7A	186	10111010	BA	250	11111010	FA
59	00111011	3B	123	01111011	7B	187	10111011	BB	251	11111011	FB
60	00111100	3C	124	01111100	7C	188	10111100	BC	252	11111100	FC
61	00111101	3D	125	01111101	7D	189	10111101	BD	253	11111101	FD
62	00111110	3E	126	01111110	7E	190	10111110	BE	254	11111110	FE
63	00111111	3F	127	01111111	7F	191	10111111	BF	255	11111111	FF

メモ

変更履歴

◎ 2013年 7月 初版発行