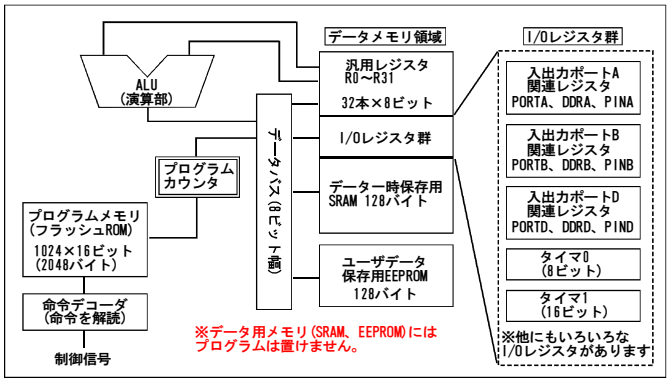
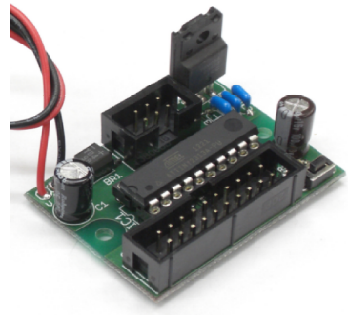
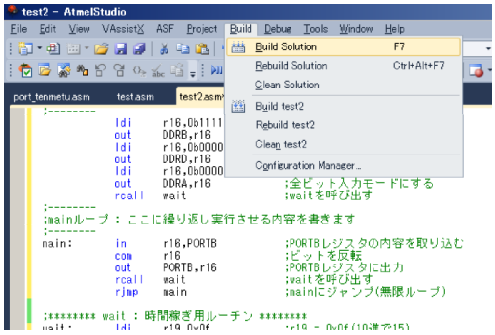
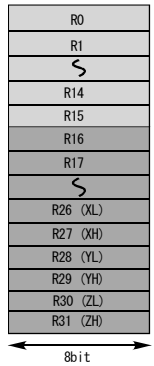
AVRマイコン アセンブラ超入門セット
(基礎知識&I/Oポート制御編)

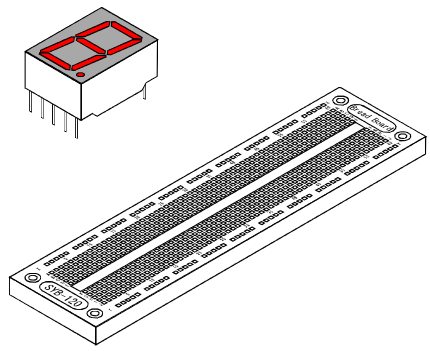
****** マイコンの動作を目(LED)で見てチェックする!! *******











このテキストの概要

このテキストは、Atmel社のAVRマイコン(ATtiny2313A)を使った簡単なプログラムを作って動かすことを通じて、マイコンのしくみと動作、プログラムの開発のしかた、I/Oポートの使いかたについて説明することを目的としています。

このテキストとセットのパーツセットは、皆さんが目で見たり、手で触ったりすることで、マイコンの動作を実際に確かめられるようになっています。

(1)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、入出力ポートの説明について学習します。

(2)実際にマイコン上で動くプログラムを作るにはどうするのかについて、AVRマイコンの開発環境「Atmel Studio 6」を使って、開発手順を説明します。

(3)簡単な実例のプログラムを実際に作って動かしてみます。LEDとCdSセルの動作を確認しながら、マイコンの入出力の基本的な使い方を説明します。

このテキストでは、これまでマイコンを触ったことのない人でも、I/Oポートを操ってLEDを点灯させたり、外部の状態をマイコンに取り込んだりできるようになることを目指します。

◎ 本キットにはAVRライタ(作成したプログラムをAVRマイコンに書き込むためのツール)は附属しません。
AVRライタをお持ちでない方は、AVRライタを別途お求めください。

◎ この説明書で使っているAVRライタは、デジットの「AVRWRT3」です。Atmel社純正の「AVRISP mk2」「AVR DRAGON」などを使う場合は、開発環境のヘルプを参照してください。

オーディオ・マイコン・メカトロ・電子パーツ
デジット
年中無休・営業時間:AM11:00～PM8:00
〒556-0005 大阪市浪速区日本橋4-6-7
TEL 06-6644-4555 / FAX 06-6644-1744
HP http://digit.kyohritsu.com
Blog http://blog.digit-parts.com Twitter @0666444555

1. はじめに ***** マイコンのプログラミングはとても簡単で面白い!! *****

このテキストでは、このテキストと一緒に入っている部品セットを使って、実際にプログラムを作り、マイコンに書き込んで、動作をみなさんの目や手で確かめていただくことで、マイコンがどのようにして動作するのか、マイコンを使って何かを制御したり、外部の状態をマイコンに取り込むには、何をすればできるのかを説明します。

この「AVRマイコン アセンブラ超入門セット」は、次のような方を対象に書かれています:

(1) はじめてマイコンを触る人
(2) C言語やその他の言語で動かしてみたことはあるけれども、マイコンがどのように動いているのかもっと知りたい人

このテキストでは、次のような理由から、マイコンのプログラム開発言語として、アセンブラを使っています。

(1) アセンブラなら、マイコン内部の汎用レジスタやプログラムカウンタを常に意識しながら、1ステップずつ動作をイメージしながらプログラミングするので、プログラムがどのような仕組みで動くのかを非常に理解しやすいです。

(2) 約10種類のよく使われる基本的な命令の使い方と動作をマスターすることで、はじめての人でもどんどんプログラムを作って動かせるようになります。(このキットに入っているATtiny2313Aの場合、全部で120種類の命令があります)

(3) アセンブラでのプログラミングを覚えておくと、C言語でプログラミングする場合でも、プログラムの細かい動作を思考しながらプログラムが作れるので、わかりやすくてバグの少ない、良いプログラムが作れるようになります。

このテキストは、次のような構成になっています。

Step 1: 汎用レジスタ、プログラムカウンタ、プログラムメモリ、データSRAM、I/Oレジスタといった、マイコンの中身について説明します。

Step 2: 簡単なプログラムを実際に書いてみることで、開発環境の使い方、アセンブラプログラムの書き方の基本的な約束事を説明します。

Step 3: マイコンのI/Oポートを使って、入出力の実験を行うことで、I/Oポート関連の設定レジスタ(PORTx、DDRx、PINx)の使い方を説明します。

このテキストを使って繰り返し実験していただくことで、はじめてAVRマイコンを触る人でも、LEDの点滅や7セグメントLEDによる数字の表示、センサからの外部入力の扱いができるようになり、自分でいろいろなことができるようになると思います。

AVRマイコン アセンブラ超入門セット
(基礎知識&I/Oポート制御編)
目次

◎はじめに ---- 2

◎実験の準備
回路の組み立て ---- 4
開発環境のインストール ---- 7
AVRライタのインストール ---- 7

◎AVRマイコン(ATtiny2313A)のしくみ
主な仕様 ---- 8
主要部ブロックダイヤグラム ---- 8
内部メモリの説明 ---- 9
プログラムはどのように実行されるのか ---- 10
汎用レジスタについて ---- 12

◎はじめてのプログラム[LEDの点滅]
プロジェクトの作成 ---- 13
プログラムを書いてビルドする ---- 15

◎I/Oポートの使い方
マイコン(ATtiny2313A)のピン配置 ---- 20
I/Oポート(デジタル入出力ポート)関係の設定レジスタの概要 ---- 20
実際にデジタルI/Oポートを使ってみる ---- 23

◎練習 ---- 30

◎資料編
よく使う命令とその概要 ---- 33
全命令セット一覧 ---- 34
よく使う擬似命令一覧 ---- 36
I/Oレジスタマップ ---- 37
数値(10進、2進、16進)早見表 ---- 38

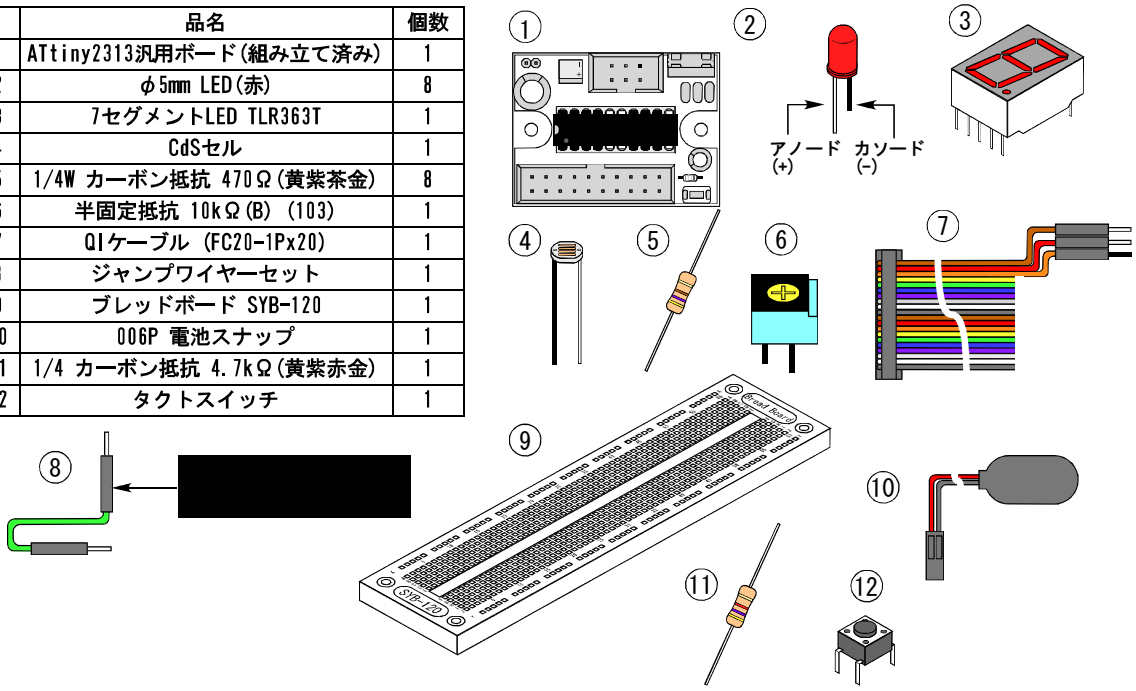
2. 実験の準備

2.1. ブレッドボードに回路を組み立てます

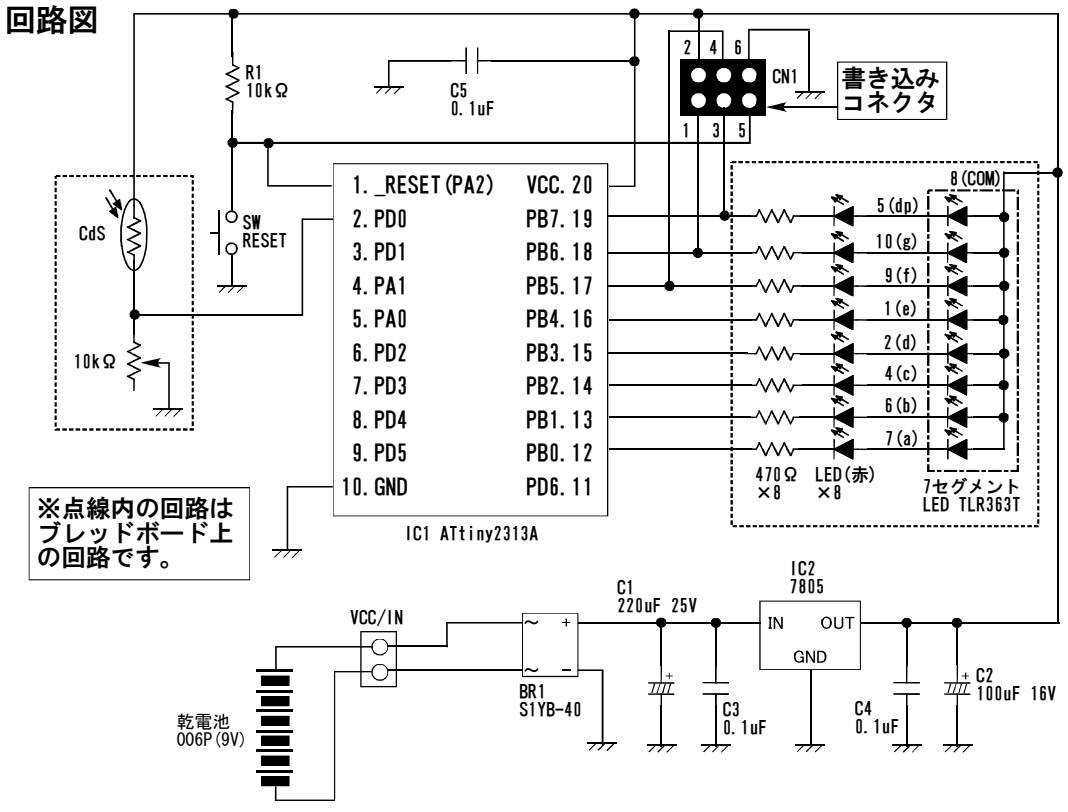
実験をはじめる前に、キットに附属の部品を使って、ブレッドボードに回路を組み立てます。

部品表 ※予告なく変更することがあります

	品名	個数
1	ATtiny2313汎用ボード(組み立て済み)	1
2	φ5mm LED(赤)	8
3	7セグメントLED TLR363T	1
4	CdSセル	1
5	1/4W カーボン抵抗 470Ω(黄紫茶金)	8
6	半固定抵抗 10kΩ(B) (103)	1
7	QIケーブル (FC20-1Px20)	1
8	ジャンプワイヤーセット	1
9	ブレッドボード SYB-120	1
10	006P 電池スナップ	1
11	1/4 カーボン抵抗 4.7kΩ(黄紫赤金)	1
12	タクトスイッチ	1

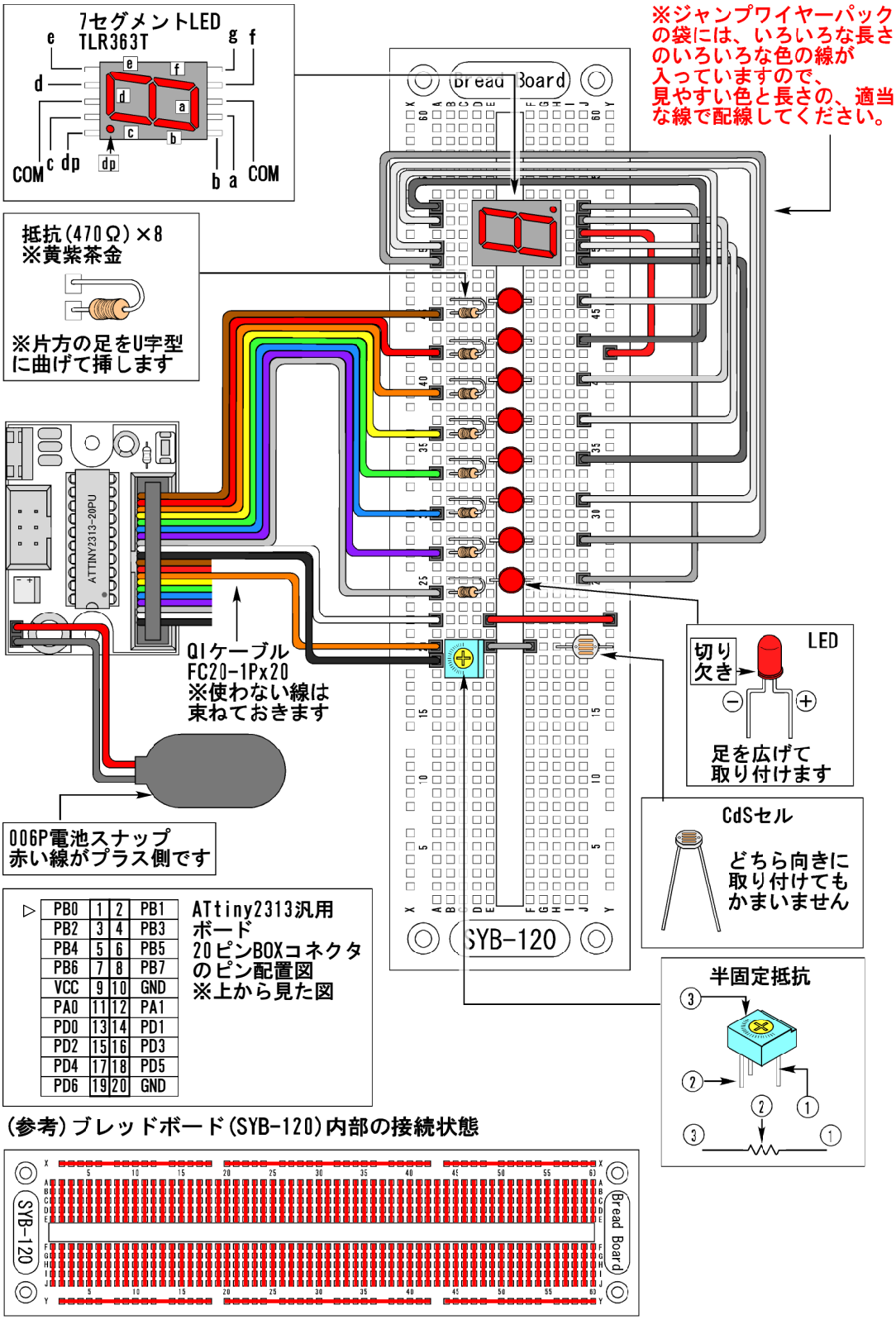


回路図



組み立てかた

下図のように、ブレッドボードに部品を挿して、ジャンプワイヤーで配線します。



PB0	1	2	PB1
PB2	3	4	PB3
PB4	5	6	PB5
PB6	7	8	PB7
VCC	9	10	GND
PA0	11	12	PA1
PD0	13	14	PD1
PD2	15	16	PD3
PD4	17	18	PD5
PD6	19	20	GND

ATtiny2313汎用ボード
20ピンBOXコネクタのピン配置図
※上から見た図

(参考)ブレッドボード(SYB-120)内部の接続状態

（補足：7セグメントLEDについて）

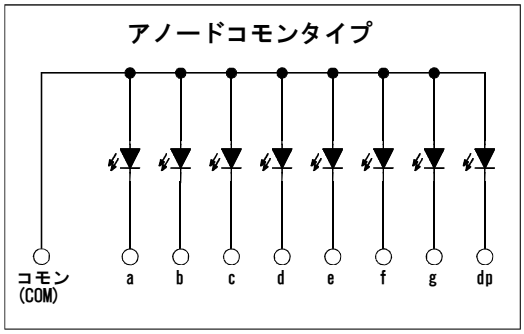
今回製作するブレッドボードには、バラのLEDとは別に、「7セグメントLED」(通称7セグ)というLEDモジュールが使われています。ブレッドボードの組み立てに入る前に、この7セグメントLEDについて少しだけ説明します。

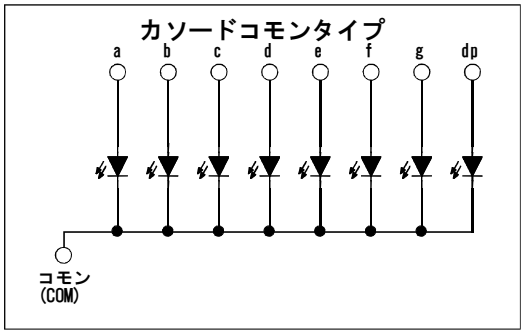
「7セグメントLED」は、左の写真のように、細長いLEDを漢字の「日」の字型に配置した表示モジュールです。
左の写真は、本キットに入っている1桁タイプのものです。

小数点表示を除いて、細長いLED(表示セグメント)が7つあるので、「7セグメントLED」または単に「7セグ」と呼ばれています。

7セグメントLEDの中には、小数点を含めてLEDが8個入っていますが、これらそれぞれのLEDから独立に足を引き出すと、足の本数が多くなり(16本)、配線が大変なので、LEDのアノード側とカソード側のどちらか一方は、7セグメントLEDのケースの中で共通に接続されています。

LEDのアノード側を共通(コモン)にしたものを、「アノードコモンタイプ」、カソード側を共通(コモン)にしたものを「カソードコモンタイプ」と呼びます。「アノードコモンタイプ」と「カソードコモンタイプ」それぞれの内部接続を、下の図に示します。

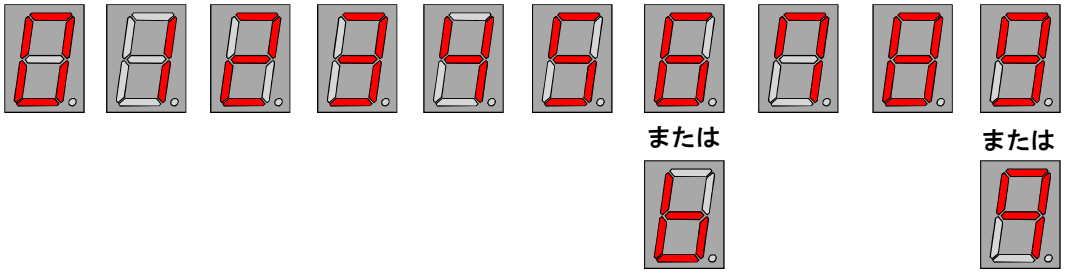




上の図の「a」～「g」と「dp」は、7セグメントの各セグメントの呼び名です。各セグメントは左図のように呼ぶ習慣になっています。ほぼ世界共通の習慣ですので、この機会に覚えておいてください。

このキットでは、アノードコモンタイプのもの(TLR363)を使用します。

この7セグメントLEDは、主に数字を表示するのに使います。数字の「0」～「9」までの点灯パターン例を、下の図に示します。自動販売機などでよく見かけると思います。



2.2. 開発環境(Atmel Studio 6)のインストール

「Atmel Studio 6」は、Atmel社のWebサイトからユーザ登録すると、ダウンロードできます。
(2013年9月現在、無料でダウンロード、使用できます。ユーザ登録も無料です)

マイコンをプログラミングして動かすためには、プログラムの開発環境を用意する必要があります。AVRマイコンのプログラミングは、「Atmel Studio 6」という統合開発環境上で行います。

「Atmel Studio 6」は、このテキストで扱うATtiny2313A以外にも、いろいろなAVRマイコンのプログラムを開発できます。

「Atmel Studio 6」をインストールすると、C言語の開発環境も一緒にインストールされます(※)ので、C言語でのプログラム開発もできます。
(※前バージョンの「AVR Studio」の場合、C言語の開発環境は「AVR Studio」とは別にダウンロードしてインストールする必要がありました)

下の説明を参照して、「Atmel Studio 6」をダウンロードして、パソコンにインストールしてください。
(※以下に説明するダウンロード手順は、2013年9月現在のものです。ダウンロードの手順は、予告なく変更されることがありますので、その場合は、Atmel社のWebページの指示に従ってダウンロードしてください。)

「Atmel Studio 6」ダウンロードのしかた (2013年9月現在):

(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。
※インストーラのアイコンをダブルクリックしてから、インストールがはじまるまで、少し時間がかかることがありますが、異常ではありません。

2.3. AVRライタ(AVRWRT3)のインストール

◎AVRライタ(AVRWRT3)は、本キットには附属しません。AVRライタをお持ちでない方は、別途お求めください。

◎Atmel社純正の書き込みツール(AVRISP mk2、AVR DRAGONなど)を使って書き込む場合の詳細については、開発環境のヘルプを参照してください。

開発環境「Atmel Studio 6」を使って開発したプログラムを、AVRマイコンに書き込むために、AVRライタを使います。このテキストでは、デジットオリジナルのAVRライタ(AVRWRT3)を使います。

AVRライタを使うには、AVRライタ用のデバイスドライバと書き込みソフト(AVRWRT.exe)をパソコンにインストールする必要があります。AVRライタ付属のCD-ROMに入っている、「インストールのしかた」をよく読んでから、インストール作業を行ってください。

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。

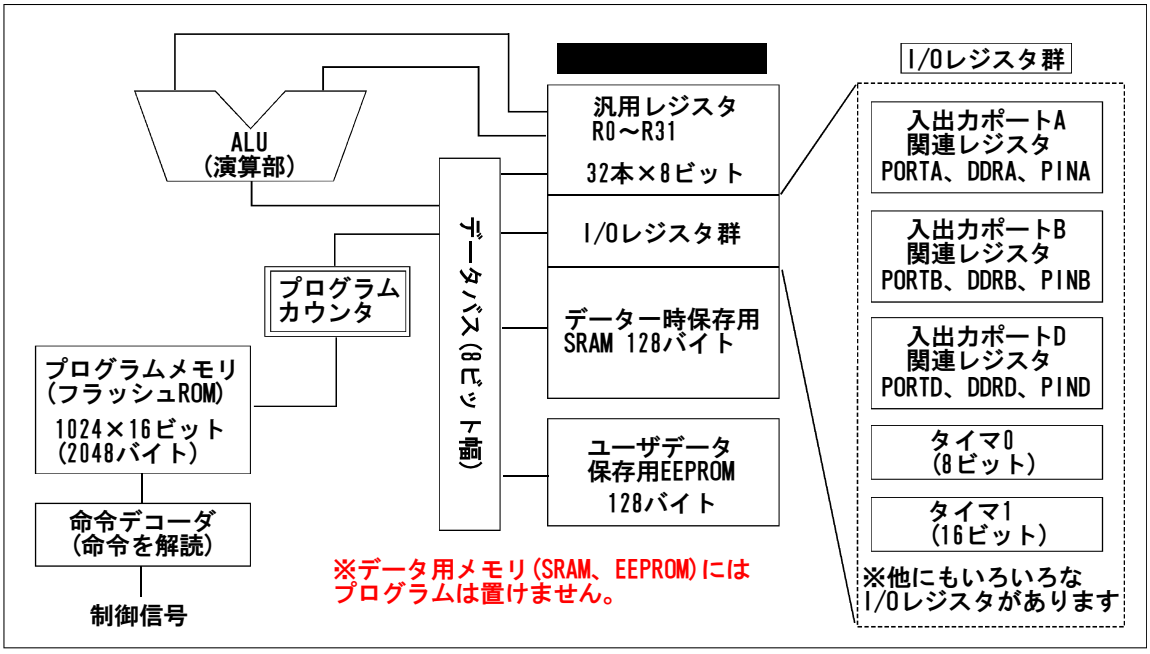
3. AVRマイコン(ATtiny2313)のしくみ

「AVRマイコン アセンブラプログラミング超入門セット」に入っているAVRマイコン、ATtiny2313について、ごく簡単に紹介します。

3.1. ATtiny2313Aの主な仕様

◎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

3.2 ATtiny2313Aの主要部ブロックダイヤグラム



AVRマイコンの内部は、上図のブロックダイヤグラムのように、非常にシンプルなつくりになっています。

AVRマイコンの基本機能ブロックには、次のものがあります。

(1)プログラムメモリ(フラッシュROM) : AVRライタでプログラム(命令)を書き込みます。
(2)プログラムカウンタ : プログラムメモリから取り出す命令の番地(アドレス)が入っています。
(3)SRAM領域 :
◎汎用レジスタ(R0～R31の32本) : データの一時保存や演算に使います。
◎I/Oレジスタ群 : 外部との入出力機能やタイマ機能、通信機能などの各種機能設定用のレジスタが装備されています。
◎データSRAM : ユーザデータを一時的に保存できます。
(4)データ用EEPROM : ユーザデータを保存できます。(電源を切っても中身は消えません)

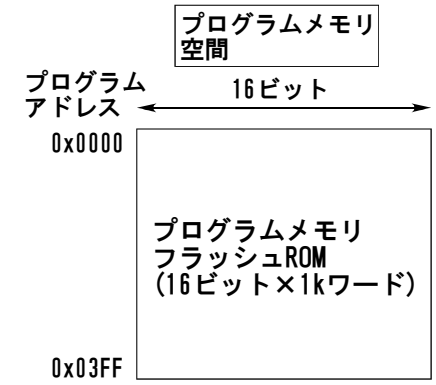
次のページ以降で、AVRマイコンのプログラムメモリ、データSRAM、データEEPROMと汎用レジスタについて、詳しく解説します。

3.3.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、内部メモリ空間として、プログラムを格納するプログラムメモリと設定情報やデータを記憶するデータメモリ空間、ユーザデータを保存するデータEEPROM空間があります。

ここでは、これらプログラムメモリとデータメモリ空間、データEEPROM空間について、詳しく解説します。

(1)プログラムメモリ(フラッシュROM)



※プログラムメモリはフラッシュROMですので、マイコンの電源を切っても中身は消えません。

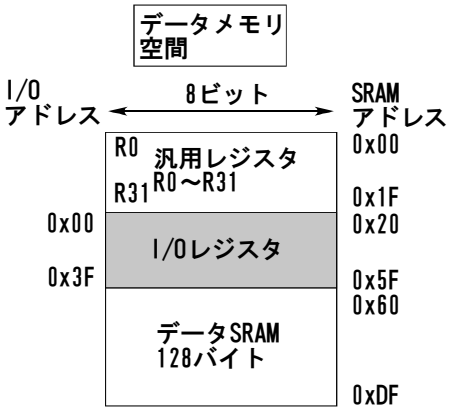
プログラムメモリはフラッシュROMです。AVRマイコンで実行されるすべての命令は、フラッシュROM上に置かれています。
この命令は、開発環境で作成して、AVRライタで外部から書き込みます。

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。この超入門セットに入っているATtiny2313Aの場合、プログラムメモリは2kバイト(2048バイト)ありますので、最大で1024ステップ分の命令を格納できます。

プログラムメモリの命令実行用アドレスは、データSRAMのアドレスとは別に割り振られていて、命令はプログラムカウンタが指示するアドレスから1個ずつ、順番に取り出されて実行されます。

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。

(2)データメモリ空間(データSRAM)



※I/Oレジスタ領域は、I/Oアドレスを使ってアクセスします。

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は、プログラム中で自由に読み書きできます。

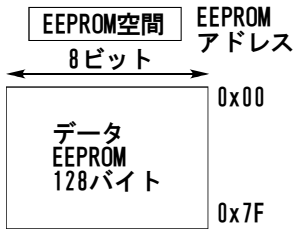
※データSRAM上には、プログラムは置けません。また、データSRAMの中身は、マイコンの電源を切ると消えてしまいます。

ATtiny2313の場合、SRAMアドレス0x60以降の領域にデータSRAMが置かれています。同じSRAM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。

(3)データEEPROM空間



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。一度データを書き込むと、次にデータを書き込むまで中身が保存された状態ですので、設定データなど、電源を切ったときに消えると困るデータを保存するのに使われます。

※EEPROMには、プログラムは置けません。

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。

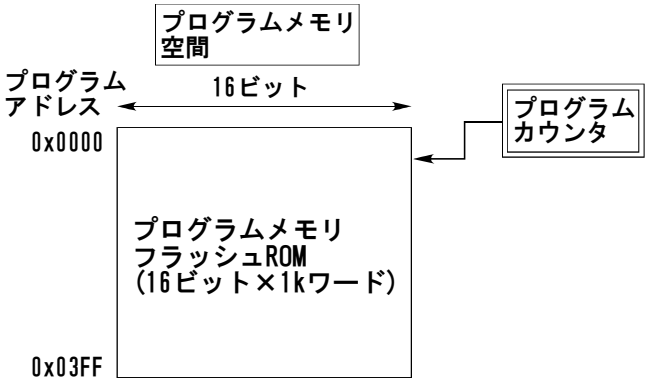
ポイント

◎AVRマイコンのプログラムは、すべてプログラムメモリ(フラッシュROM)上に置かれます。
データSRAM上やデータEEPROM上には、プログラムは置けません。

◎I/Oレジスタは、データメモリ空間に割り当てられていますが、SRAMアドレスとは別のI/Oアドレスを使って、IN/OUT命令でアクセスします。

3.4. プログラムはどのように実行されるのか?（プログラムカウンタの役目）

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。この章では、プログラムは一体どのようにして実行されるのか、プログラムカウンタの役目を中心に説明します。



AVRマイコンのプログラム(命令)は、プログラムメモリ(フラッシュROM)に書き込まれています。(8ページの説明を参照してください)

マイコンがプログラムを実行するためには、このプログラムメモリに書き込まれている命令を読み出す必要があります。

プログラムカウンタは、プログラムメモリの何番地の命令を読み出すかを指し示すカウンタです。プログラムを実行するために、マイコンはこのプログラムカウンタが指し示すプログラムメモリから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)

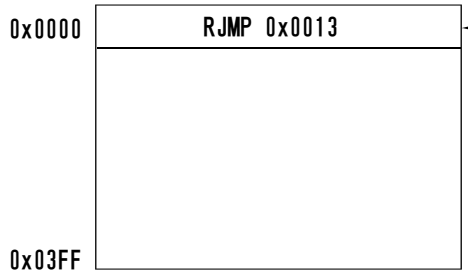
命令をプログラムメモリから取り出して実行すると、プログラムカウンタの値は自動的に1だけ増えて、プログラムメモリ中の次に実行する命令を指し示すようになっています(※)。
(※ジャンプ命令やサブルーチンコール命令を実行すると、プログラムカウンタの値は命令のジャンプ先アドレスになります)

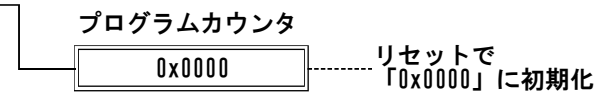
プログラムカウンタの値は、マイコンの電源を入れたあとや外部リセット信号が与えられたあと、自動的に「0x0000」に初期化されます。

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。下の図は、プログラムカウンタと命令の実行との関係を示します。

(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。

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(ATtiny2313A)で、どのような命令が実際に用意されているのかは、ATtiny2313A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。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てあります。

※ATtiny2313Aの全命令セットの概要を巻末の資料篇に載せていますので、参考にしてください。

(補足) サブルーチンコール命令とリターン命令

サブルーチンコール命令(RCALL命令)は、プログラム中でサブルーチン(よく実行される作業をひとかたまりにしたもの)を呼び出すのに使います。

リターン命令(RET命令)は、呼び出されたサブルーチンから戻るときに使います。

サブルーチンコール(RCALL命令)とリターン命令(RET命令)は、常にペアで使う必要があります。

RCALL命令を使う前に、スタックポインタ(SPLレジスタ)に、データSRAM領域の最後のアドレスを設定する必要があります。プログラムの最初の部分で、次のように書いて設定してください。

```
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。
                              ;「ldi r16,0xdf」と書いてもかまいません。
         out SPL,r16          ;「RAMEND」は、データSRAMの最後のアドレスで、
                              ; 開発環境の中で定義されています。
```

3.5.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。

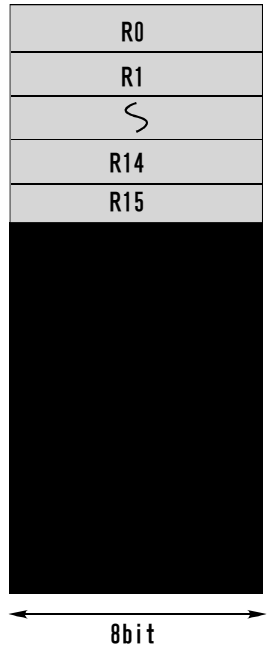
AVRマイコンの場合、内部SRAM上のデータ同士を直接足し算したり、比較論理演算したりすることはできません。また、I/Oレジスタ上のデータの設定も、直接I/Oレジスタにデータを書き込むことはできません(※)。

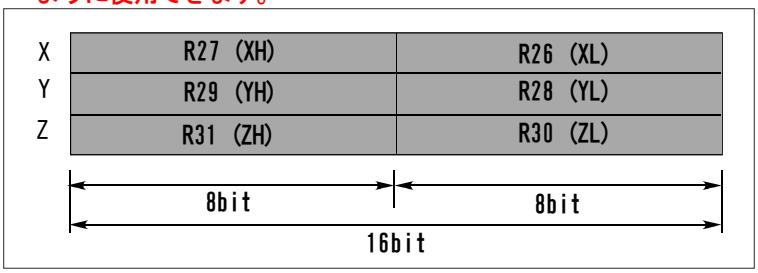
このため、ユーザのプログラムでは、汎用レジスタを介して処理を行い、内部SRAMに格納したり、I/Oレジスタにコピーしたりします。

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。

4. はじめてのプログラム[LEDの点滅]

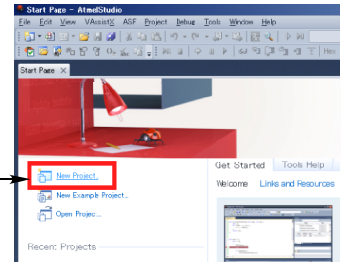
AVRマイコンのおおよその構造とプログラムの実行の仕組みについて説明しましたので、今度は実際にプログラムを作って動かして確かめてみます。I/Oポートのしくみと動作についてはまだ説明していませんが、後で詳しく説明します。

4.1. プロジェクトの作成

AVRマイコンの開発環境「Atmel Studio 6」では、作成するプログラムやビルドしてできた書き込み用のファイルなどを、「プロジェクト」という単位で管理しています。

ここでは、新しいプロジェクトを作り、その上でプログラムを実際に作って、マイコンに書き込んで動かしてみます。

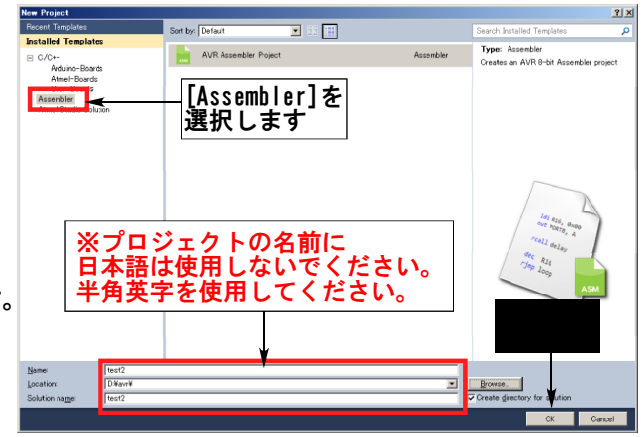
(1)Atmel Studioを起動すると、スタートページのウインドウが出ますので、ウインドウ左側の[New Project]をクリックします。



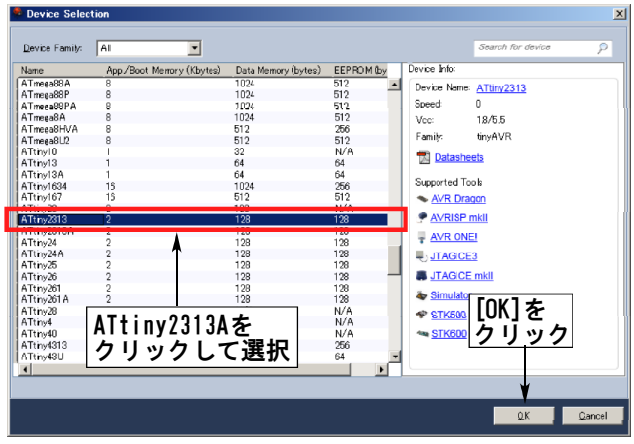
[New Project]をクリックすると、新しいプロジェクトを作るウインドウが出ますので、左側の「Installed Templates」中の[Assembler]をクリックします。

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。

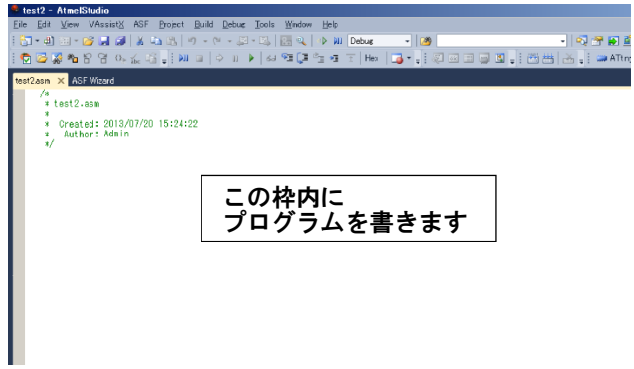
これから作るプロジェクトに好きな名前をつけた後、[OK]をクリックします。



(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ATtiny2313A)を選んで、[OK]ボタンをクリックします。

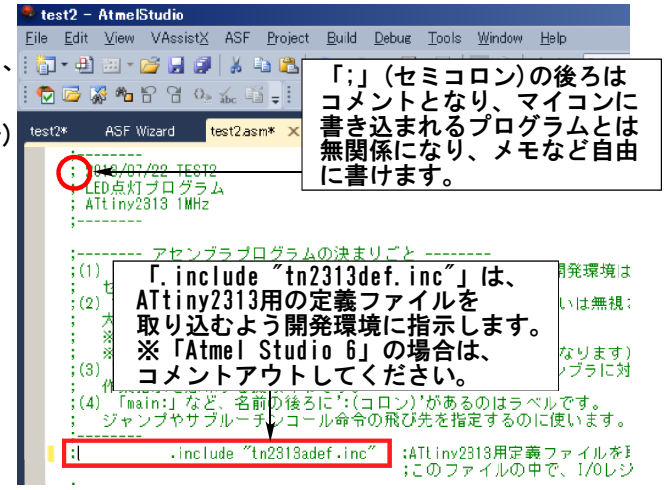


(3)「Finish」 のクリック後、ウィンドウが開きます。
これでアセンブリが出来る様になりましたので、開いたウィンドウにプログラムを書きます。



(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、マイコンに書き込まれるプログラムとは無関係になります。
デフォルトの設定では、緑字表示になります。



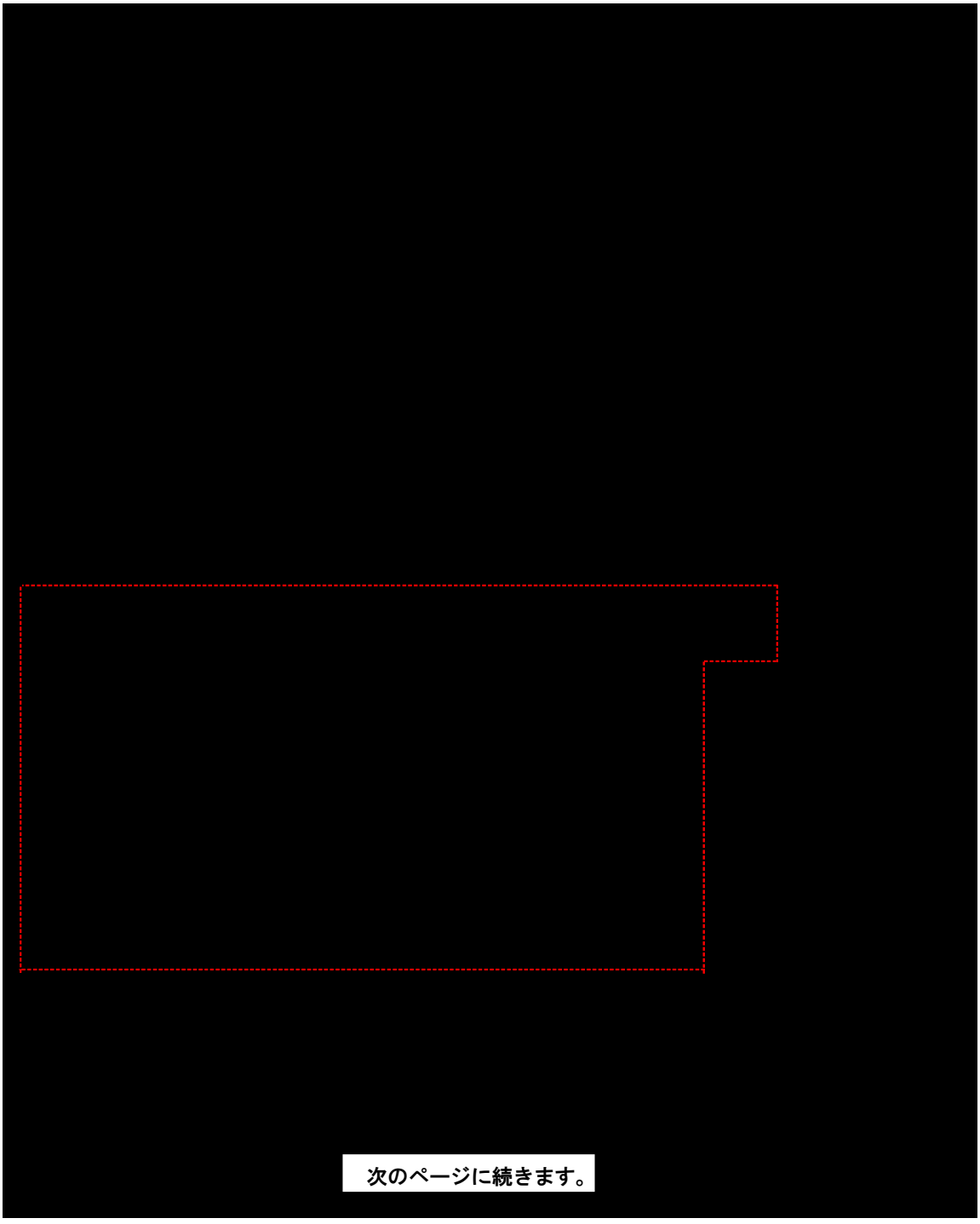
(5)「Atmel Studio 6」でプログラムを開発する場合は、プロジェクトを作成する段階で使用するマイコンを選択しますので、プログラム冒頭の「.include "tn2313adef.inc"をコメント記号(;)でコメントアウトしてください。
※「.include "tn2313adef.inc"」をコメントアウトしないと、ビルドするときに、「これこれの定義がだぶっています」という趣旨のエラーメッセージが大量に出て、ビルドに失敗します。

※「.include "tn2313adef.inc"」は、ビルド時に"tn2313adef.inc"という名前のファイルを取り込むよう、開発環境に指示する擬似命令です。前バージョンの「AVR Studio」を使って開発するときは、プログラム冒頭に「.include "tn2313adef.inc"」と書く必要があります。

4.2. プログラムを書いてビルドする

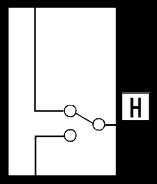
プロジェクトを作成して新しいプログラムを作る準備ができましたので、次のプログラムリストを入力します。入力を間違えてもあとで修正できますので、心配せずどんどん入力してください。
命令の詳細については、このテキストの巻末に命令セット一覧表を載せてありますので、参考にしてください。

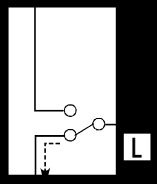
※アセンブラプログラムの中では大文字小文字を区別しません。プログラムは大文字小文字どちらで書いてもかまいません。



次のページに続きます。

前のページの続きです。





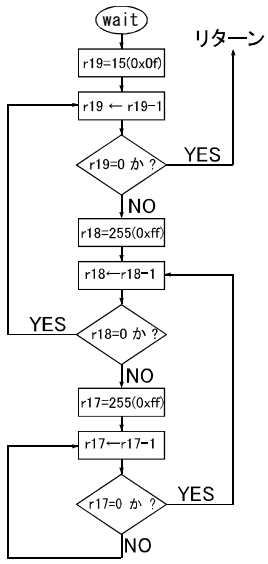
ポートが「0」のとき
点線の向きに電流が流れ
LEDが点灯します

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。

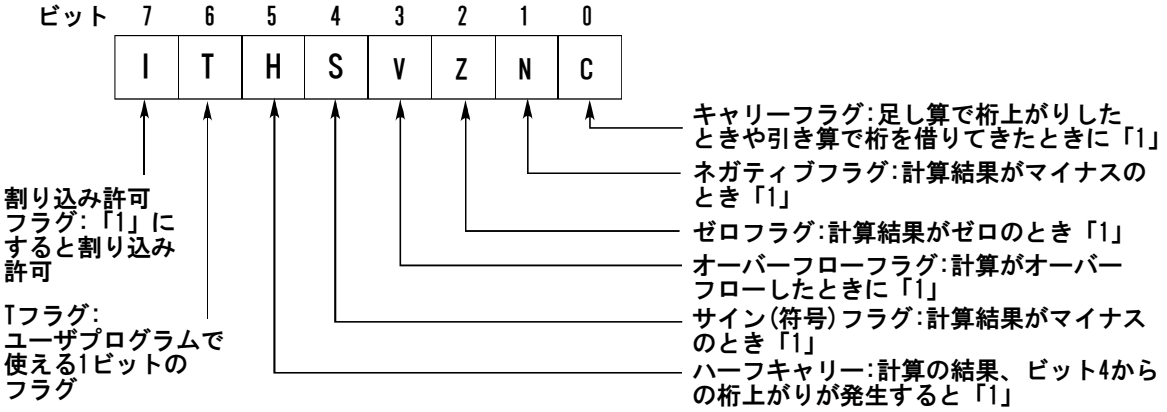
◎ CPI(シーピーアイ)命令は、引き算命令の一種です。
この例では、R17から数値「0」を引いて、結果が「0」ならば、ステータスレジスタ(SREG)のゼロ(Z)フラグが「1」になり、引き算の結果が「0以外の値」ならば、ゼロ(Z)フラグが「0」になります。

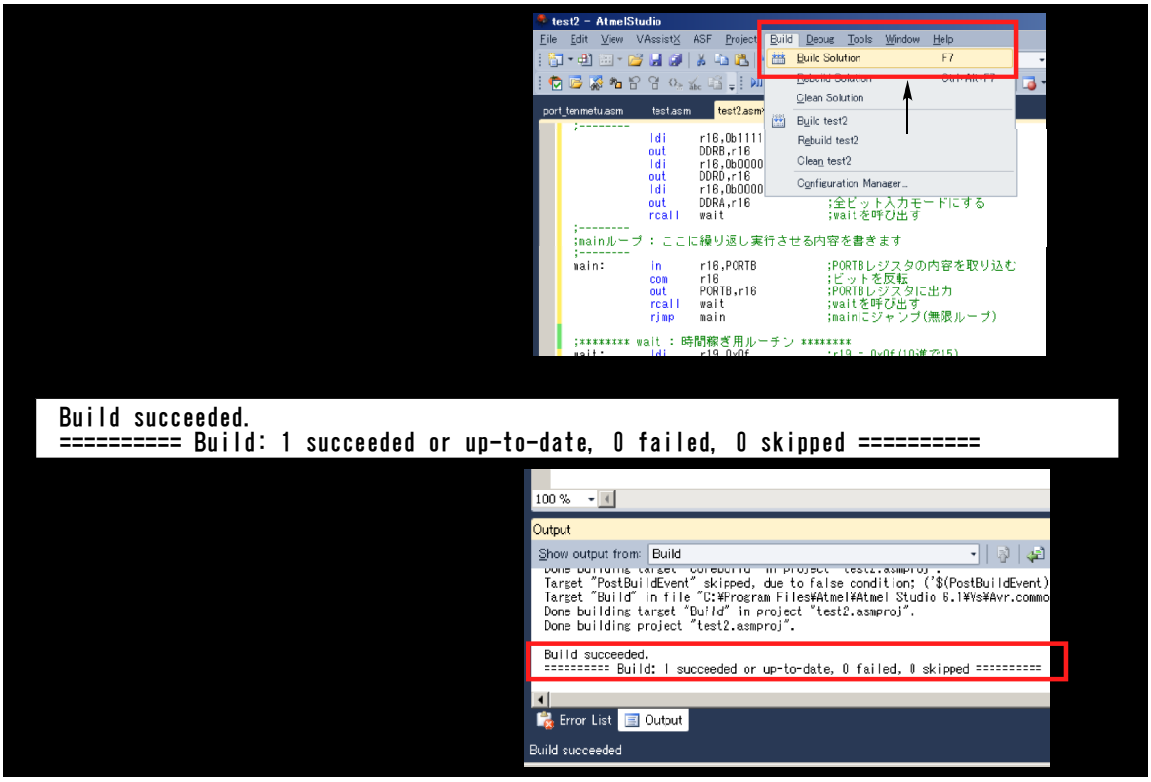
◎ BREQ(ブランチイコール)命令は、この例では、ステータスレジスタ(SREG)のゼロ(Z)フラグが「1」ならば、「wait2」のラベルに分岐し、「0」ならば何もせず次の命令に進みます。
「BREQ」の「EQ」は、ここでは「R17-0」の計算結果が「0」であること(ステータスレジスタの「Z」フラグが「1」)を指します。

◎ 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

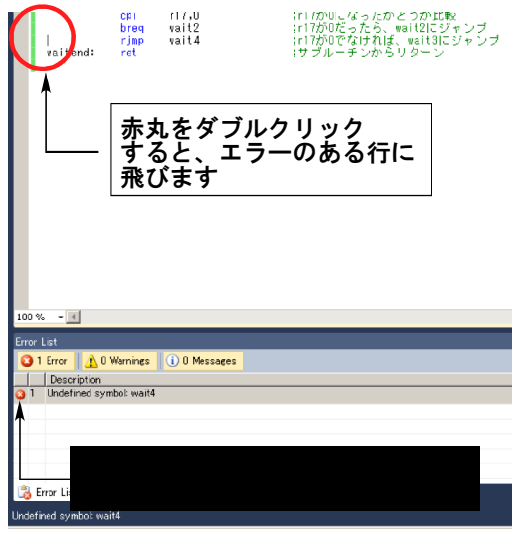
ステータスレジスタ(SREG)は、I/Oレジスタの1つで、汎用レジスタで足し算や引き算などの計算を行った結果のフラグが反映されているレジスタです。





(2)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)

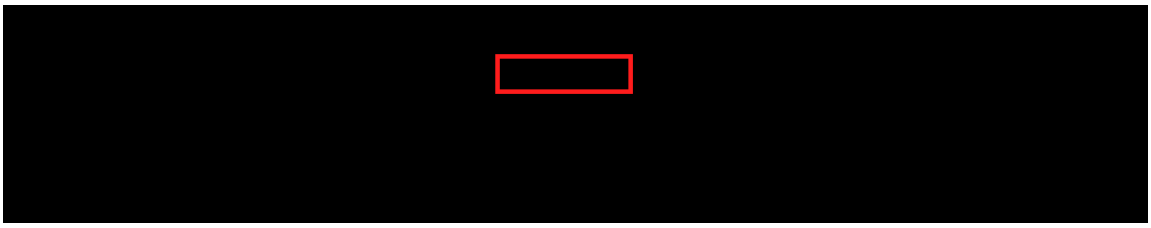


よくあるエラーの原因

◎ジャンプ先のラベルの名前が間違っている
◎コンマ(「,」)がピリオド(「.」)になっている(またはセミコロン「;」とコロン「:」の間違い)
◎命令とレジスタ名の間に空白がない
◎全角文字でプログラムを書いている
(特に全角の空白文字は気づきにくいので注意してください)

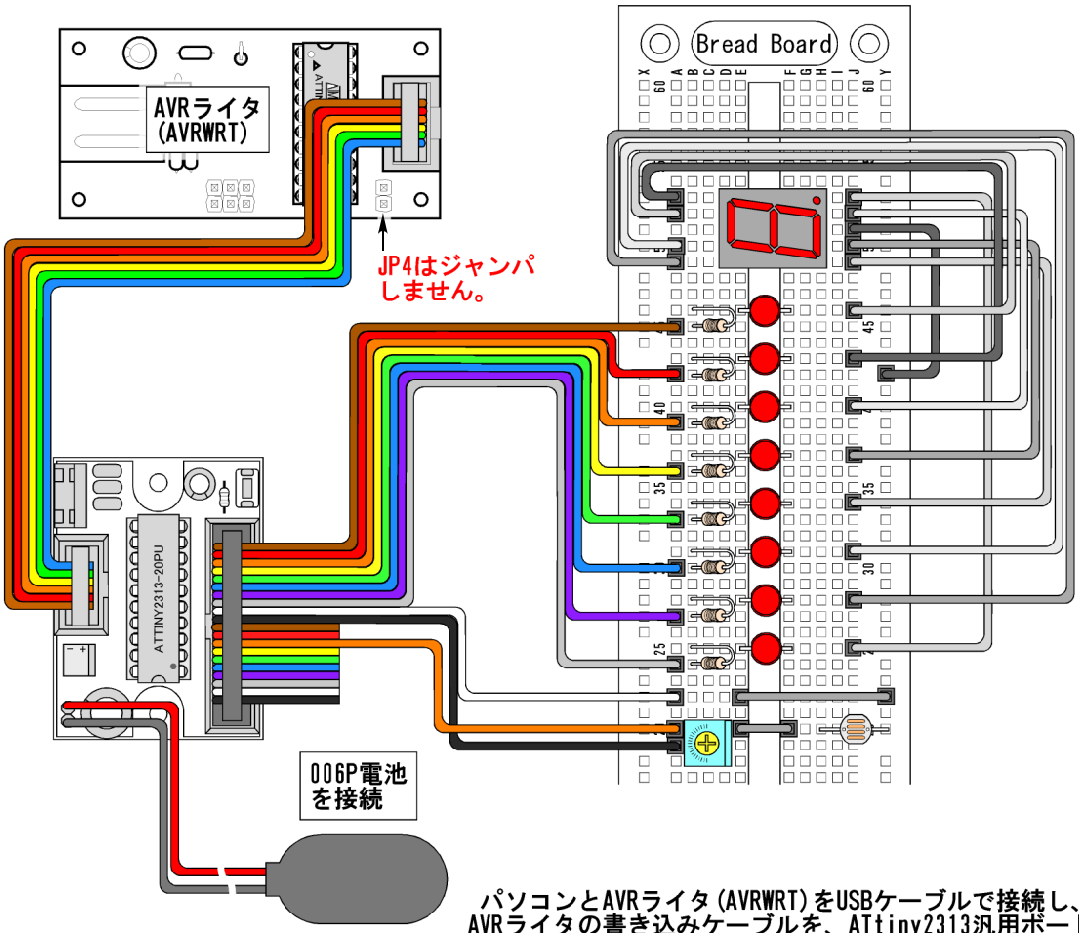
(3)ビルドしたプログラムをマイコンに書き込みます。

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。



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)4～5ページで回路を組み立てたブレッドボードを用意します。
電池スナップに006P乾電池を接続します。

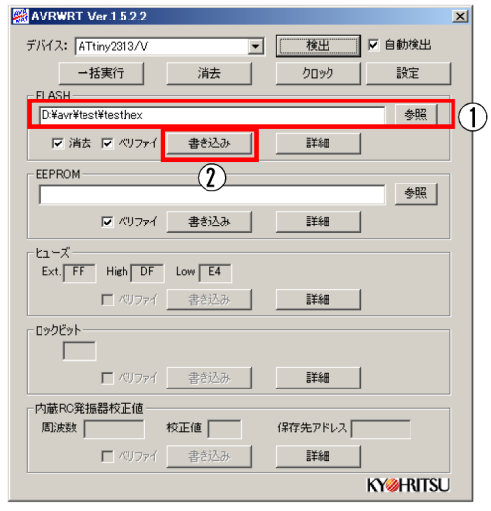


パソコンとAVRライタ(AVRWRT)をUSBケーブルで接続し、AVRライタの書き込みケーブルを、ATtiny2313汎用ボードの6ピンBOXコネクタに接続します。

(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。

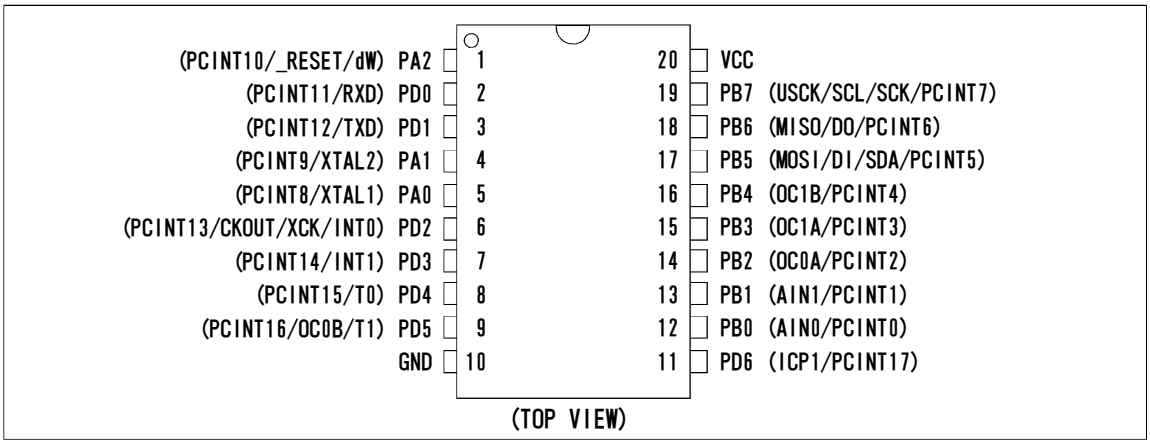


(3)AVRライタのソフトからの、「ベリファイOK」のメッセージが出たあと、
ブレッドボード上のLEDが点滅します。
7セグメントLEDのセグメントも同様に点滅しますので、よく観察してください。

5. I/Oポートの使い方

5.1.マイコン(ATtiny2313A)のピン配置

下の図は、ATtiny2313A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。



ATtiny2313A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、括弧内の信号名の機能は、機能設定することで使用できるようになります。
(詳細については、ATtiny2313Aのデータシートを参照してください)

このテキストでは、マイコンのピンをデジタルI/Oポートとして使うことを前提に説明しますので、上の図の括弧内の信号名については気にしなくてもかまいません。ただ、AVRマイコンでは、デジタルI/Oポートのピンと各種機能のピンが共用になっていて、機能設定することで、様々な機能を使えるようになるということは、覚えておいてください。

5.2.I/Oポート(デジタル入出力ポート)関係のI/Oレジスタの概要

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、ATtiny2313Aの、I/Oレジスタマップから、I/Oポート関係の部分のみ抜粋したものです。ATtiny2313Aには、ポートA、ポートB、ポートDの3つのI/Oポートがあることがこの表からわかります。設定用レジスタは、ポートA用、ポートB用、ポートD用に、それぞれ次のレジスタがあります。
(「x」のところには、ポートA用、ポートB用、ポートD用に応じて、それぞれA、B、Dが入ります)

◎PORTx ： 主にデータ出力に使用

◎DDRx ： 入出力の切り替え設定に使用

◎PINx ： 主に入力の状態を取り込むのに使用

次のページで、PORTx、DDRx、PINxの各レジスタの機能を説明します。

(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。
入出力ポートのピンに対応するDDRxのビットに「1」を書き込むとそのピンは出力、「0」を書き込むとそのピンは入力になります。
例えば、ICのピンPB0～PB7を出力ポートに設定するには、DDRBレジスタのビットに相当するビットに「1」を書きます。つまり、DDRBレジスタに、「0b11111111」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

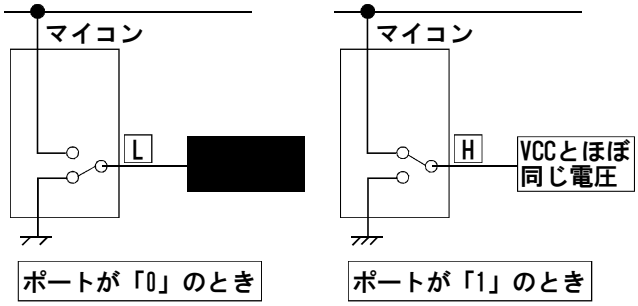
```
LDI     R16,0b10000000
OUT     DDRB,R16
```

※(参考)マイコンをリセットすると、DDRxレジスタの値は「0b00000000」に初期化されます。

(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。



このテキストでは、マイコンの設定レジスタの内容を「0」「1」と、実際のマイコンの入出力のピンの状態(電圧)を「L」「H」と書いています。

設定レジスタの「0」が電圧の「L」に、設定レジスタの「1」が電圧の「H」に対応します。
(左の図を参照してください)

(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
```

※(参考)マイコンをリセットすると、PORTxの値は「0b00000000」に初期化されます。

(補足)ATtiny2313Aの場合、出力ポートに流せる電流はポート1つにつき最大で40mA、マイコン全体の合計で最大200mAです。
これを超える電流を出力ポートに流すと、マイコンが壊れますので、注意してください。

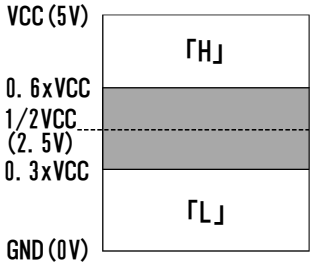
(3)PINxレジスタ(PINA、PINB、PIND)の機能

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」を取り込むことができます。

[redacted]

(補足)「しきい値」について

「しきい値」とは、マイコンを含むデジタル回路において、「H」レベルと「L」レベルを識別するときの基準となる電圧のことです。



AVRマイコンは「CMOS」という製法で製造されています。しきい値は電源電圧(VCC)の1/2の電圧です。

左の図は、ATtiny2313Aの入力電圧レベルを図にしたものです。入力電圧が0.6xVCC(5V動作のとき3V)以上あれば、「H」レベルと、0.3xVCC(5V動作のとき1.5V)以下であれば「L」レベルとして入力されることを表しています。

左の図で灰色になっている電圧範囲は、「H」と「L」のどちらとして認識されるか、不明の電圧範囲(不定領域)です。
マイコンで使用しない入力ピンは、抵抗でプルアップ(抵抗を介してVCCに接続)またはプルダウン(抵抗を介してGNDに接続)して、マイコンに入力される電圧レベルをはっきりさせてください。

※「しきい値」の詳しいことについては、デジタル回路の入門書を参照してください。

5.3.実際にデジタルI/Oポートを使ってみる

このテキスト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る実験を行います。

プログラムの説明については、この節の後半にあります。

(1)Atmel Studioで新しい別のプロジェクトを作成し、次のプログラムを書きます。

```
;--------
; 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
; ATtiny2313A 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、開発環境は
;      セミコロンより後ろを無視します。
;(2) アセンブラプログラムの中では名前や命令の大文字小文字の違いは無視されます。
;      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 "tn2313adef.inc" ;ATtiny2313A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 (ATtiny2313A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
;--------
; ATtiny2313Aの場合、プログラムは0x13番地以降からはじまるように書きます
;--------
```

点線の枠内は、今回のプログラムでは、書かなくてもかまいません。(使用しません)

次のページに続きます。

前のページの続きです。

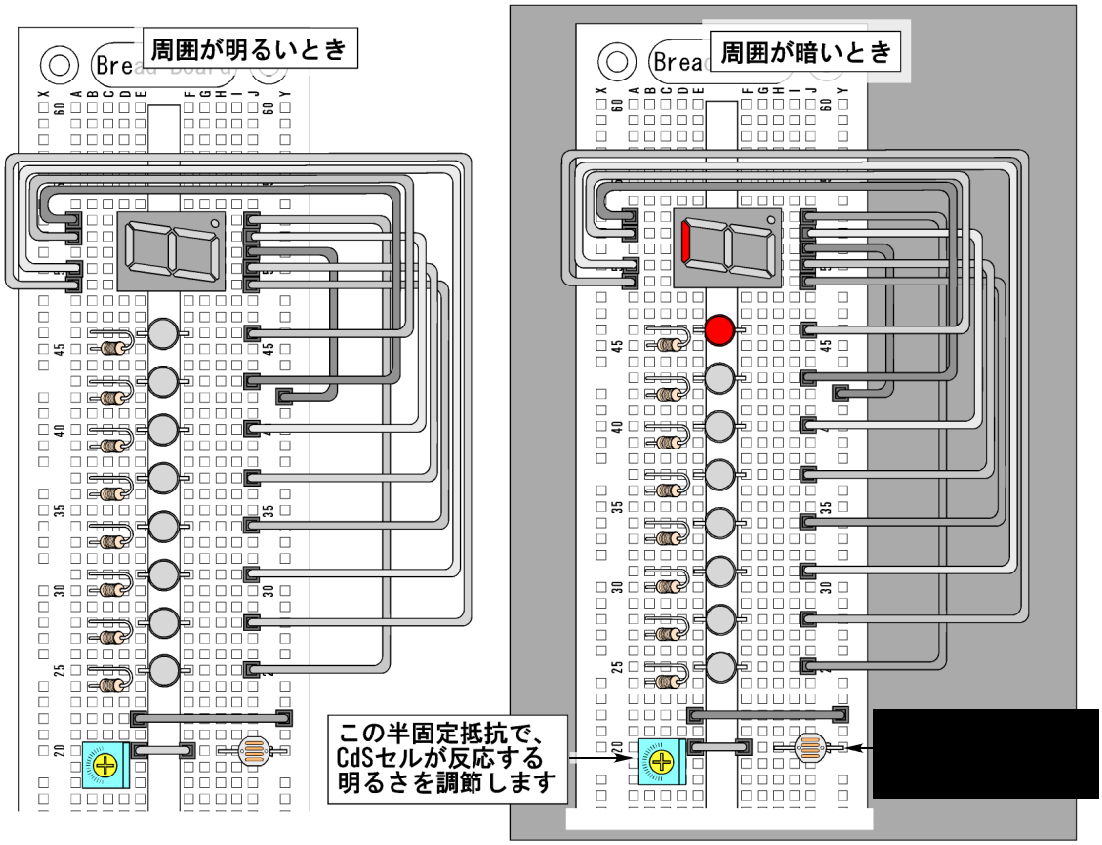
次のページに続きます。

前のページの続きです。

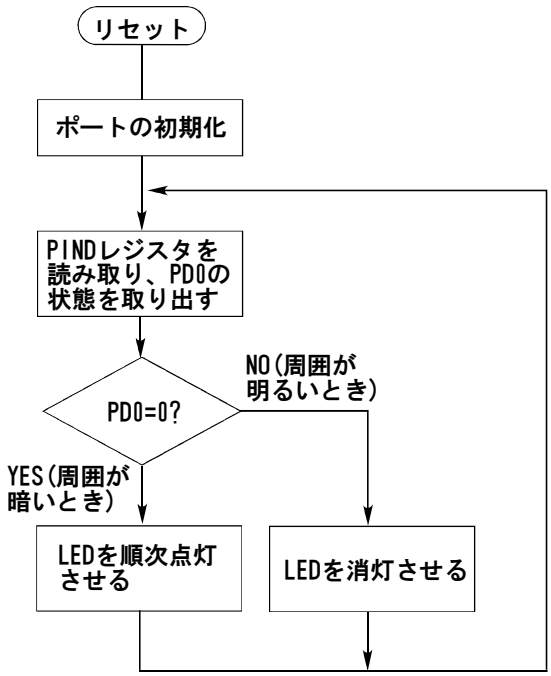
```
                ldi     r16, 0b11011111         ; PB5のLEDを点灯させます(r16=0b11011111)
                out     PORTB, r16              ; PORTBレジスタに出力
                rcall   wait                    ; 時間待ち
;
                ldi     r16, 0b10111111         ; PB6のLEDを点灯させます(r16=0b10111111)
                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
wait2:          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、wait1にジャンプ
                rjmp    wait2                   ;r17が0でなければ、wait2にジャンプ
waitend:        ret                             ;サブルーチンからリターン
```

(2)プログラムを書き終わったら、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。



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(23ページから25ページ)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)

15ページから17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(23ページから25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
            out     DDRB,r16
;---- 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
            out     DDRA,r16
```

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。(20ページの説明を見てください)

この実験では、CdSセルの状態を見たいので、CdSセルの接続先ポートである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。
(具体的には、「0b11111110」をDDRDレジスタに書き込んでいます)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。(21ページの説明を見てください)

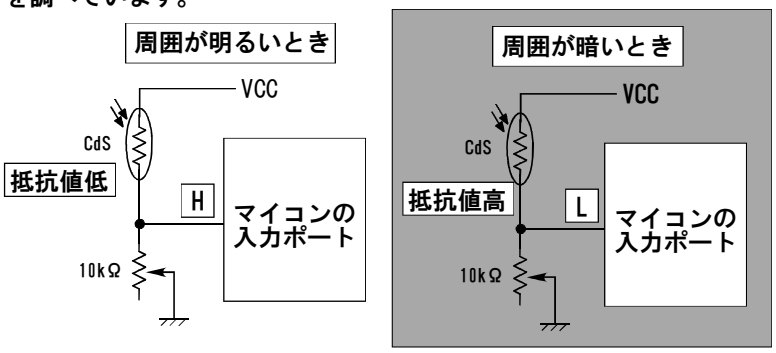
	(未使用)	PD6	PD5	PD4	PD3	PD2	PD1	PD0
DDRD	1	1	1	1	1	1	1	0
		出力	出力	出力	出力	出力	出力	入力

PD0を入力にするには、
DDRDを左のように設定します。
(赤字は変更された箇所)

(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。

この入門セット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。

※抵抗による分圧については、電子回路の入門書を参照してください。

ポートの特定のビットのみを使って条件判断するにはどうすればよいか?

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(アンド)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。

このようなことをしなくても、「in　　r16,PIND」でr16に読み取った実際のピンの状態を、次のように、単に「0」と比較すればよいようにも思えますが、これは正しい方法ではありません。

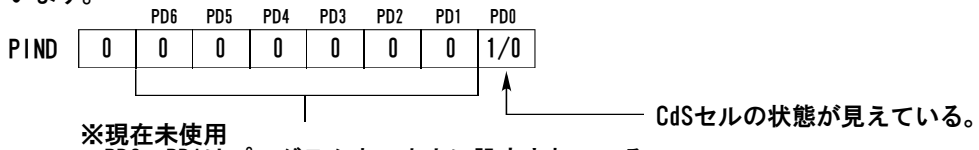
悪い例

```
main:           in      r16,PIND            ; CdSセルの状態を入力
                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)
```

「andi」命令がありません

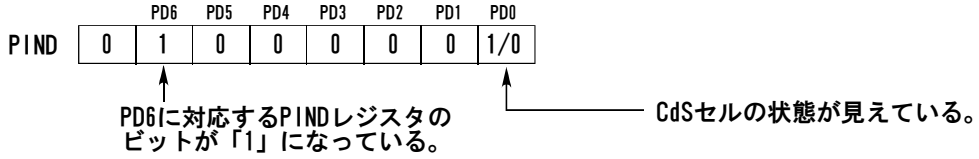
20～22ページのプログラムを、この「悪い例」のように変更しても、一応動きますが、PD1～PD6のいずれかのビットを、他の用途に使うと、動かなくなります。

なぜ動かなくなるのか、「in r16,PIND」命令を実行したときの、PINDレジスタの状態を考えてみます。「in r16,PIND」命令を実行したときのPINDレジスタは、下図のような状態になっています。



※現在未使用
PD6～PD1はプログラム中で出力に設定されている。
PINDレジスタは、ポートDの実際のピンの状態を反映しているので、ここでは偶然「0」になっている。
(これはAVRマイコンがリセット後、PORTDレジスタを自動的に「0b00000000」に初期化するため)

ここで仮に、「in r16,PIND」命令の前に、PD6を「1」にするようなOUT命令を実行していたとすると、「in r16,PIND」命令を実行したときのPINDレジスタは、下図のような状態になります。なぜ下図のような状態になるのかについては、19ページの説明を参照してください。



20～22ページのプログラムでは、次のように、r16の値(PINDレジスタの値が入っています)を「0」と比較し、「0と等しい」か「0と等しくない」かで条件分岐しています。

```
                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」命令の前にPD6を「1」にするようなOUT命令を実行していた場合、前ページの「悪い例」のようにプログラムを書いていたならば、「cpi r16,0」命令の結果は常に「0と等しくない」ことになり、LEDの点滅部分に分岐してくれないことがわかります。
この問題を解決するには、ポートDのほかのビットには関係なく、PD0に接続されたCdSの状態だけを取り出して調べる必要があります。

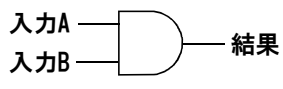
このような場合には、「andi」命令を使い、PD0に対応するビット0のみを「1」に、他のビットを「0」にしたデータ(0b00000001)とANDをとります。

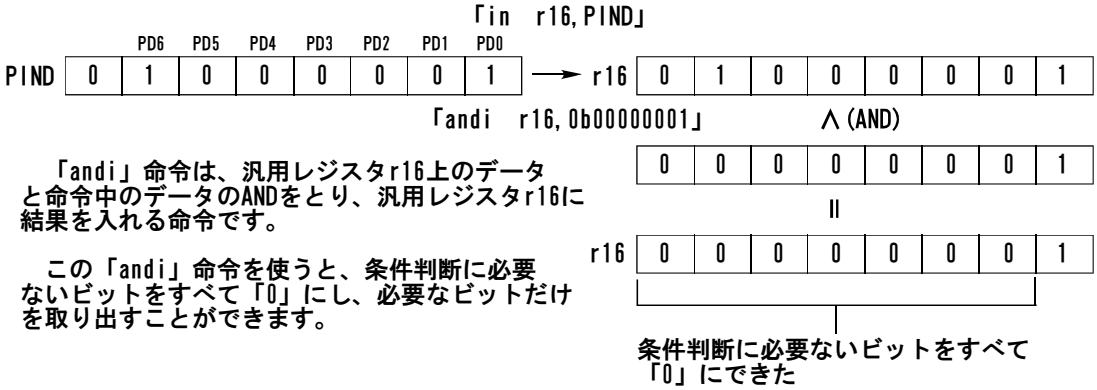
ANDの真理値表

入力A	入力B	結果
0	0	0
0	1	0
1	0	0
1	1	1

ANDは、2つの入力が両方とも「1」のときに限り結果が「1」になり、2つの入力のどちらか一方でも「0」ならば結果が「0」になる演算です。AND演算の真理値表と、論理回路の記号は、左の図表を見てください。

ここで仮に、PD6が「1」を出力しているものと仮定して、andi命令の働きを調べてみます。





「andi」命令は、汎用レジスタr16上のデータと命令中のデータのANDをとり、汎用レジスタr16に結果を入れる命令です。

この「andi」命令を使うと、条件判断に必要ないビットをすべて「0」にし、必要なビットだけを取り出すことができます。

正しいプログラム

```
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)
```

上の「正しいプログラム」中で、「andi r16,0b00000001」命令の次の命令は「cpi r16,0」です。直前のandi命令で、r16のビット7～ビット1はすべて「0」であることが保証されます。
つまり、「andi r16,0b00000001」命令を実行したあとの、r16の値は、「0」か「1」かのどちらかになります。

周囲が明るい場合、「cpi r16,0」の結果は「0と等しくない」(なぜならr16の値は「0b00000001」だから)ので、「cpi r16,0」の次の「breq　main_1」命令は実行されません。(それに続く命令がそのまま実行されます)

周囲が暗くなると、「cpi r16,0」の結果は「0と等しい」ので、「cpi r16,0」の次の、「breq　main_1」命令が実行され、LEDの順次点滅部分にジャンプします。

ポイント
◎I/Oレジスタの値(ここではPINDレジスタの値)を使って正しく条件判断するには、「andi」命令を使って、条件判断に関係しないビットをすべて「0」にする必要がある。

(補足)主な論理演算

主な論理演算には、次のようなものがあります。詳細については、デジタル回路の入門書を参照してください。

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

OR(オア)の真理値表

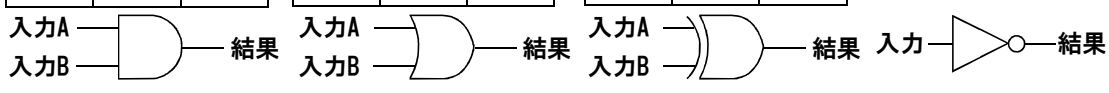
入力A	入力B	結果
0	0	0
0	1	1
1	0	1
1	1	1

XOR(エックスオア)の真理値表

入力A	入力B	結果
0	0	0
0	1	1
1	0	1
1	1	0

NOT(ノット)の真理値表

入力	結果
0	1
1	0

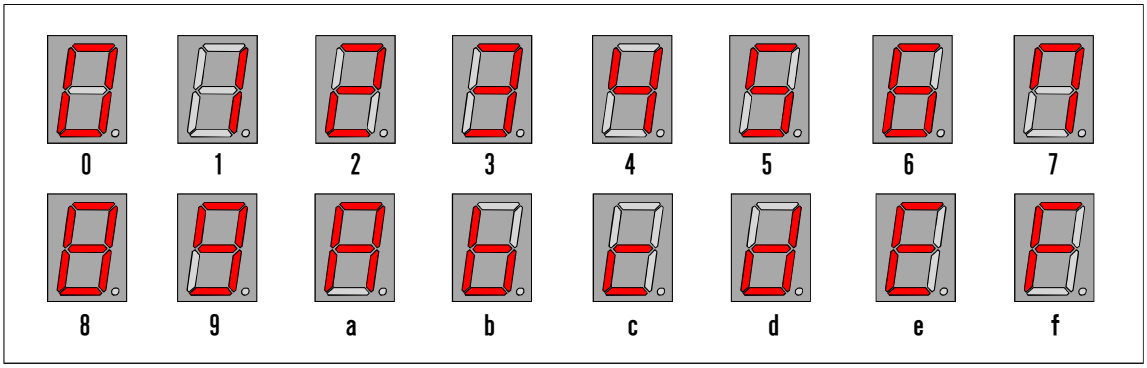


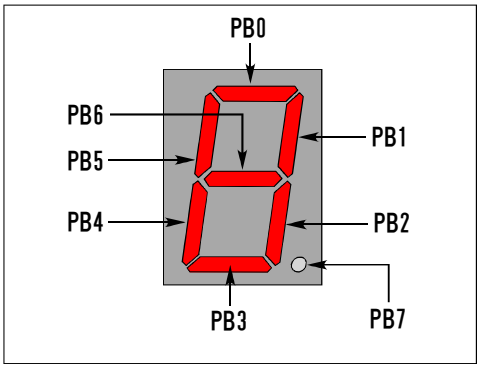
5. 練習

5.1. 7セグメントLEDで、数字を表示する

この入門セットのブレッドボードには、バラのLEDと一緒に、7セグメントLEDを載せてあります。
※7セグメントLEDについての簡単な解説が、5ページに載っていますので参考にしてください。

この7セグメントLEDを使って、実際に0から9までの数字と、aからfまでのアルファベットを表示させてみてください。表示させる数字の字形は次のとおりです。

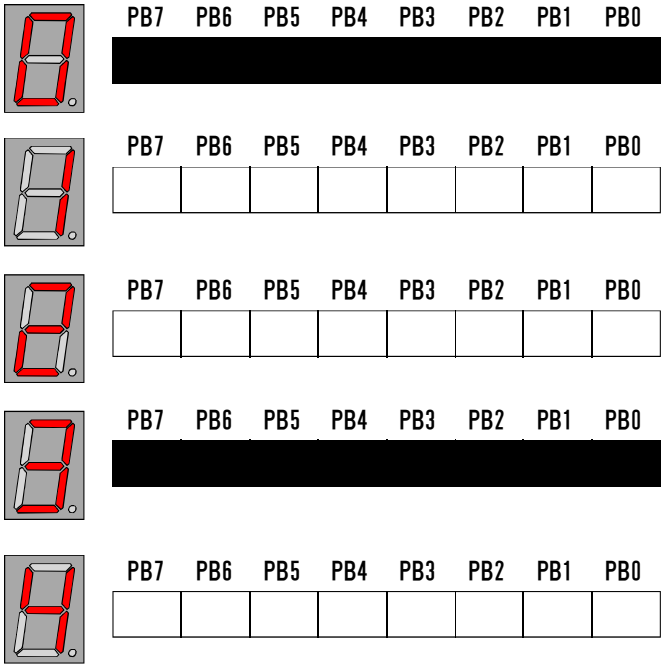


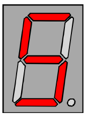


7セグメントLEDの各セグメントと、接続先のマイコンのポートとの関係は、左の図のとおりです。

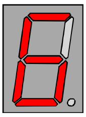
参考
このキットのブレッドボードの場合、7セグメントLEDのセグメントは、「0」で点灯、「1」で消灯します。

(1)各数字とセグメントの表示パターンから、ポートBに出力するデータを決めてください。

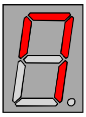




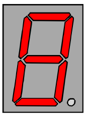
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



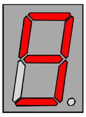
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



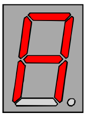
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



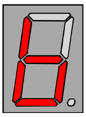
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



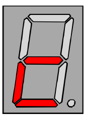
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



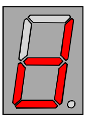
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



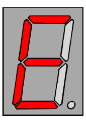
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



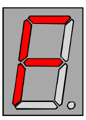
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0



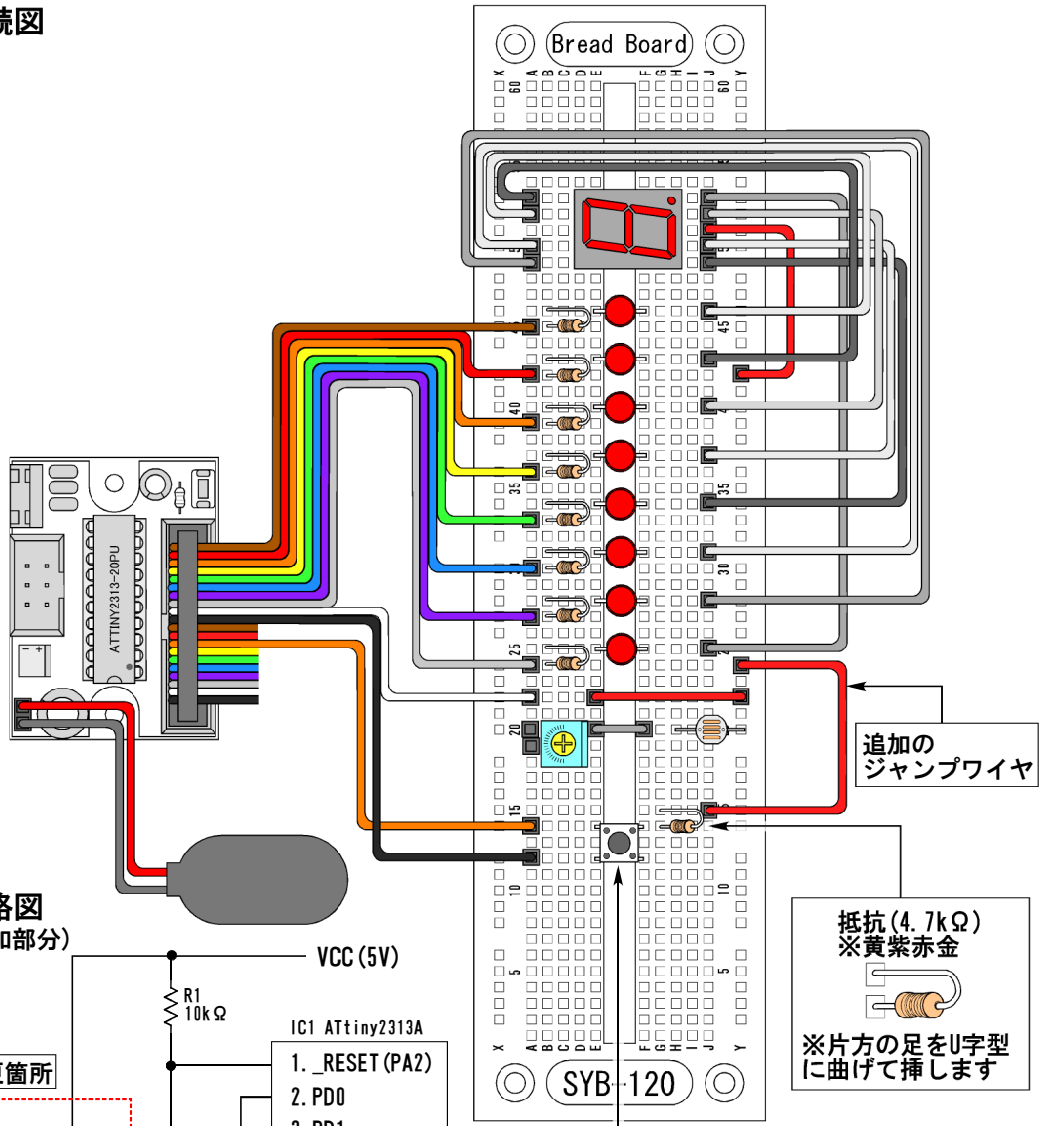
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0

(2)LED点滅プログラム(13ページから15ページ)の、LED点滅部分に、設計したビットパターンを書き、(パターンが足りなければ追加してください)、ビルドして書き込んでください。

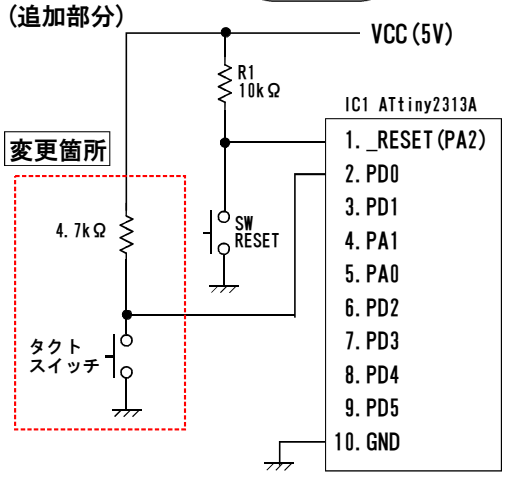
5.2. タクトスイッチ(押しボタンスイッチ)による入力の実験

この入門セットには、実験用パーツとして、タクトスイッチと4.7kΩの抵抗が付属しています。23～25ページの実験では、入力としてCdSセルと半固定抵抗を使いましたが、これをタクトスイッチと抵抗に置き換えてみます。

接続図



回路図
(追加部分)



プログラムは23～25ページのプログラムを使います。

23～25ページのプログラムをマイコンに書き込むと、タクトスイッチを押していないときにLEDが点滅し、タクトスイッチを押すとLEDが消灯するはずです。

タクトスイッチを押したときにLEDが点滅し、タクトスイッチを押していないときにLEDが消灯するようにするにはどうすればよいでしょうか? 巻末資料編のよく使う命令一覧を参考にして、プログラムを改造してみてください。

資料編

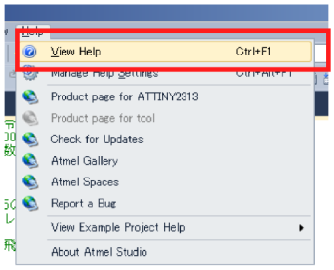
(1)よく使われる、主な命令とその操作の概要

ATtiny2313Aでよく使われる命令の一覧とその概要を説明します。

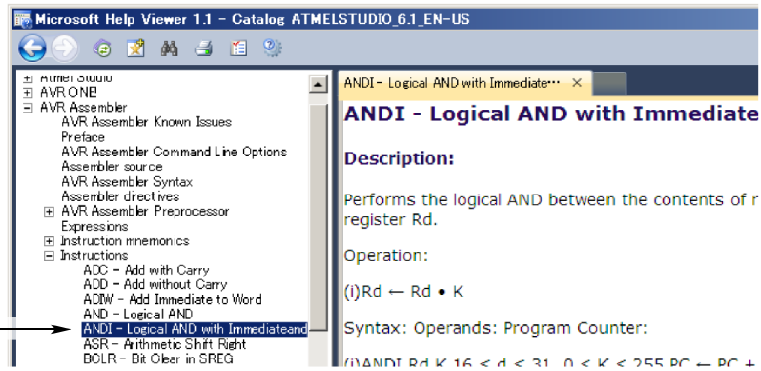
	命令	動作の概要
1	LDI Rd,K	汎用レジスタRdに、定数Kをコピーする
2	MOV Rd,Rr	汎用レジスタRrの値を、汎用レジスタRdにコピーする
3	OUT P,Rr	汎用レジスタRrの値を、入出力レジスタPにコピーする
4	IN Rd,P	入出力レジスタPの値を、汎用レジスタRdにコピーする
5	LDS Rd,MEM	データSRAMのMEM番地の値を、汎用レジスタRdにコピーする
6	STS MEM,Rr	汎用レジスタRrの値を、データSRAMのMEM番地にコピーする
7	INC Rd	汎用レジスタRdの値を、1増やす(インクリメントする)
8	DEC Rd	汎用レジスタRdの値を、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

ATtiny2313Aの命令についてもっと詳しく知りたいときは

※ATtiny2313A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。

ATtiny2313A 全命令の概要 (1/2)

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)「変化するフラグ」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)「クロック数」欄は、命令を実行するのに必要なクロック数です。

ニモニック	オペランド	概要	変化するフラグ	クロック数
Arithmetic and Logical Instrunctions				
ADD	Rd, Rr	Add two Registers	Z, C, N, V, H	1
ADC	Rd, Rr	Add with Carry two Registers	Z, C, N, V, H	1
ADIW	Rdl, K	Add Immediate to Word	Z, C, N, V, S	2
SUB	Rd, Rr	Subtract two Registers	Z, C, N, V, H	1
SUBI	Rd, K	Subtract Constant from Register	Z, C, N, V, H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Z, C, N, V, H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Z, C, N, V, H	1
SBIW	Rdl, K	Subtract Immediate from Word	Z, C, N, V, S	2
AND	Rd, Rr	Logical AND Registers	Z, N, V	1
ANDI	Rd, K	Logical AND Register and Constant	Z, N, V	1
OR	Rd, Rr	Logical OR Registers	Z, N, V	1
ORI	Rd, K	Logical OR Register and Constant	Z, N, V	1
EOR	Rd, Rr	Exclusive OR Registers	Z, N, V	1
COM	Rd	One' s Complement	Z, C, N, V	1
NEG	Rd	Two' s Complement	Z, C, N, V, H	1
SBR	Rd, K	Set Bit(s) in Register	Z, N, V	1
CBR	Rd, K	Clear Bit(s) in Register	Z, N, V	1
INC	Rd	Increment	Z, N, V	1
DEC	Rd	Decrement	Z, N, V	1
TST	Rd	Test for Zero or Minus	Z, N, V	1
CLR	Rd	Clear Register	Z, N, V	1
SER	Rd	Set Register	None	1
BRANCH INSTRUCTIONS				
RJMP	k	Relative Jump	None	2
IJMP		Indirect Jump to (Z)	None	2
RCALL	k	Relative Subroutine Call	None	3
ICALL		Indirect Call to (Z)	None	3
RET		Subroutine Return	None	4
RETI		Interrupt Return	I	4
CPSE	Rd, Rr	Compare, Skip if Equal if (Rd = Rr)	None	1/2/3
CP	Rd, Rr	Compare	Z, N, V, C, H	1
CPC	Rd, Rr	Compare with Carry	Z, N, V, C, H	1
CPI	Rd, K	Compare Register with Immediate	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared if (Rr(b)=0)	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set if (Rr(b)=1)	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared if (P(b)=0)	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set if (P(b)=1)	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	None	1/2
BREQ	k	Branch if Equal	None	1/2
BRNE	k	Branch if Not Equal	None	1/2
BRCS	k	Branch if Carry Set	None	1/2
BRCC	k	Branch if Carry Cleared	None	1/2
BRSH	k	Branch if Same or Higher	None	1/2
BRLO	k	Branch if Lower	None	1/2
BRMI	k	Branch if Minus	None	1/2
BRPL	k	Branch if Plus	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	None	1/2
BRHS	k	Branch if Half Carry Flag Set	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	None	1/2
BRTS	k	Branch if T Flag Set	None	1/2
BRTC	k	Branch if T Flag Cleared	None	1/2
BRVS	k	Branch if Overflow Flag is Set	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	None	1/2
BRIE	k	Branch if Interrupt Enabled	None	1/2
BRID	k	Branch if Interrupt Disabled	None	1/2
BIT AND BIT-TEST INSTRUCTIONS				
SBI	P, b	Set Bit in I/O Register	None	2
CBI	P, b	Clear Bit in I/O Register	None	2
LSL	Rd	Logical Shift Left	Z, C, N, V	1
LSR	Rd	Logical Shift Right	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	Z, C, N, V	1

ATtiny2313A 全命令の概要 (2/2)

ニモニック	オペランド	概要	変化するフラグ	クロック数
ROR	Rd	Rotate Right Through Carry	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	None	1
BSET	s	Flag Set	SREG(s)	1
BCLR	s	Flag Clear	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T	1
BLD	Rd, b	Bit load from T to Register	None	1
SEC		Set Carry	C	1
CLC		Clear Carry	C	1
SEN		Set Negative Flag	N	1
CLN		Clear Negative Flag	N	1
SEZ		Set Zero Flag	Z	1
CLZ		Clear Zero Flag	Z	1
SEI		Global Interrupt Enable	I	1
CLI		Global Interrupt Disable	I	1
SES		Set Signed Test Flag	S	1
CLS		Clear Signed Test Flag	S	1
SEV		Set Twos Complement Overflow.	V	1
CLV		Clear Twos Complement Overflow	V	1
SET		Set T in SREG	T	1
CLT		Clear T in SREG	T	1
SEH		Set Half Carry Flag in SREG	H	1
CLH		Clear Half Carry Flag in SREG	H	1
DATA TRANSFER INSTRUCTIONS				
MOV	Rd, Rr	Move Between Registers	None	1
MOVW	Rd, Rr	Copy Register Word	None	1
LDI	Rd, K	Load Immediate	None	1
LD	Rd, X	Load Indirect	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	None	2
LD	Rd, Y	Load Indirect	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	None	2
LD	Rd, Z	Load Indirect	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	None	2
LDS	Rd, k	Load Direct from SRAM	None	2
ST	X, Rr	Store Indirect	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	None	2
ST	-X, Rr	Store Indirect and Pre-Dec.	None	2
ST	Y, Rr	Store Indirect	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	None	2
ST	-Y, Rr	Store Indirect and Pre-Dec.	None	2
STD	Y+q, Rr	Store Indirect with Displacement	None	2
ST	Z, Rr	Store Indirect	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	None	2
STD	Z+q, Rr	Store Indirect with Displacement	None	2
STS	k, Rr	Store Direct to SRAM	None	2
LPM		Load Program Memory to R0	None	3
LPM	Rd, Z	Load Program Memory	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	None	3
SPM		Store Program Memory	None	-
IN	Rd, P	In Port	None	1
OUT	P, Rr	Out Port	None	1
PUSH	Rr	Push Register on Stack	None	2
POP	Rd	Pop Register from 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	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 ″tn2313adef.inc″ ; ATtiny2313A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) ATtiny2313A I/Oレジスタマップ

I/Oアドレス（16進）	レジスタの名前	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
0x3F	SREG	I	T	H	S	V	N	Z	C
0x3E	(予約)	*	*	*	*	*	*	*	*
0x3D	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C	OCR0B	タイマカウンタ0 コンペアレジスタB							
0x3B	GIMSK	INT1	INT0	PCIE0	PCIE2	PCIE1	*	*	*
0x3A	GIFR	INTF1	INTF0	PCIF0	PCIF2	PCIF1	*	*	*
0x39	TIMSK	TOIE1	OCIE1A	OCIE1B	*	ICIE1	OCIE0B	TOIE0	OCIE0A
0x38	TIFR	TOV1	OCF1A	OCF1B	*	ICF1	OCF0B	TOV0	OCF0A
0x37	SPMCSR	*	*	RSIG	CTPB	RFLB	PGWRT	PGERSW	SPMEN
0x36	OCR0A	タイマカウンタ0 コンペアレジスタA							
0x35	MCUCR	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00
0x34	MCUSR	*	*	*	*	WDRF	BORF	EXTRF	PORF
0x33	TCCR0B	FOC0A	FOC0B	*	*	WGM02	CS02	CS01	CS00
0x32	TCNT0	タイマカウンタ0 (8ビット)							
0x31	OSCCAL	*	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0
0x30	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	*	*	WGM01	WGM00
0x2F	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	*	*	WGM11	WGM10
0x2E	TCCR1B	ICNC1	ICES1	*	WGM13	WGM12	CS12	CS11	CS10
0x2D	TCNT1H	タイマカウンタ1 カウンタレジスタ上位							
0x2C	TCNT1L	タイマカウンタ1 カウンタレジスタ下位							
0x2B	OCR1AH	タイマカウンタ1 コンペアレジスタA 上位							
0x2A	OCR1AL	タイマカウンタ1 コンペアレジスタA 下位							
0x29	OCR1BH	タイマカウンタ1 コンペアレジスタB 上位							
0x28	OCR1BL	タイマカウンタ1 コンペアレジスタB 下位							
0x27	(予約)	*	*	*	*	*	*	*	*
0x26	CLKPR	CLKPCE	*	*	*	CLKPS3	CLKPS2	CLKPS1	CLKPS0
0x25	ICR1H	タイマカウンタ1 インプットキャプチャレジスタ 上位							
0x24	ICR1L	タイマカウンタ1 インプットキャプチャレジスタ 下位							
0x23	GTCCR	*	*	*	*	*	*	*	PSR10
0x22	TCCR1C	FOC01A	FOC1B	*	*	*	*	*	*
0x21	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
0x20	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
0x1F	(予約)	*	*	*	*	*	*	*	*
0x1E	EEAR	EEPROM アドレスレジスタ							
0x1D	EEDR	EEPROM データレジスタ							
0x1C	EECR	*	*	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
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
0x15	GPIOR2	GPIOレジスタ2							
0x14	GPIOR1	GPIOレジスタ1							
0x13	GPIOR0	GPIOレジスタ0							
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
0x0F	USIDR	USI データレジスタ							
0x0E	USISR	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0
0x0D	USICR	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC
0x0C	UDR	UART データレジスタ (8ビット)							
0x0B	UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
0x0A	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
0x09	UBRRL	UART ボーレート設定レジスタ 下位							
0x08	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
0x07	BODCR	*	*	*	*	*	*	BODS	BODSE
0x06	PRR	*	*	*	*	PRTIM1	PRTIM0	PRUSI	PRUSART
0x05	PCMSK2	*	PCINT17	PCINT16	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11
0x04	PCMSK1	*	*	*	*	*	PCINT10	PCINT9	PCINT8
0x03	UCSRC	UMSEL1	UMSEL0	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
0x02	UBRRH	*	*	*	*	UART ボーレート設定レジスタ 上位			
0x01	DIDR	*	*	*	*	*	*	AIN1D	AIN0D
0x00	USIBR	USI バッファレジスタ							

※ 表の中で「(予約)」となっているレジスタにはアクセスしないでください。
　また、「*」となっているビットは未使用のビットです。

(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