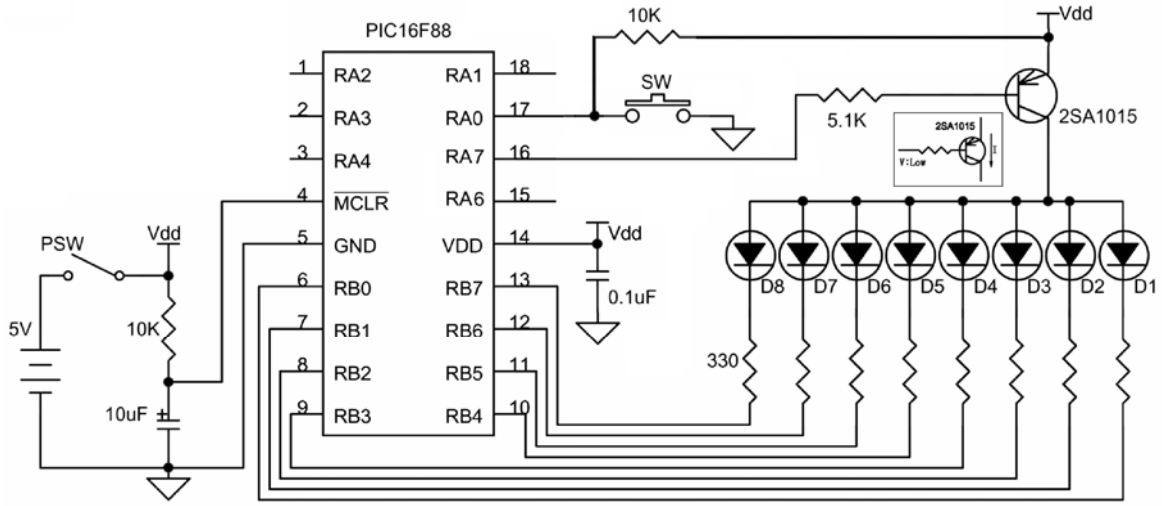
はじめに

本書は生理科学実験技術トレーニングコースにて我々が 2007 年度よりおこなってきた,「C 言語によるＰＩＣプログラミング」の実習で作成したテキストから抜粋して再編集したものです。PIC については書籍やインターネット等で詳しく紹介がされており本書では詳しく説明をすることはしません。また，デジタル回路の基本的なことやプログラムの作成には C 言語の知識が多少必要になります。したがって，本書の内容では PIC の魅力を分かっていただけないかも知れませんが，マイクロ・コントローラに興味はあるがまだ使ったことが無い人に少しでもお役にたてればと考えています。

（本書をお読みいただく前に）

図 1-1 はサンプルのプログラムで使用するテストボードの基本回路です。
さっそくですが，この回路図から何ができるのか想像できますか？



PIC16F88　　：PIC デバイス
330，5.1K，10K：抵抗（数値は抵抗の値）
10uF，0.1uF　：コンデンサ（数値は容量の値）
D1～D8　　：LED
2SA1015　　：トランジスタ
PSW，SW　　：スイッチ，押しボタン
VDD　　：電源＋
▽　　：グランド（GND）

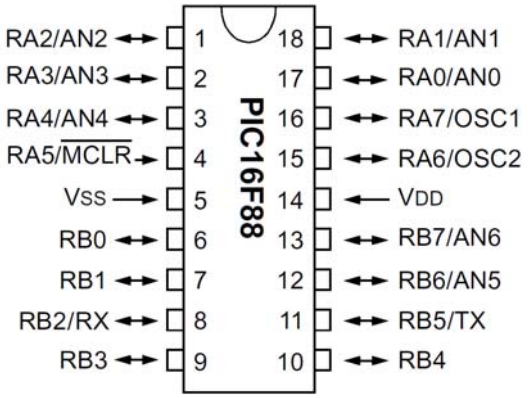
図 1-1，テストボート基本回路

PIC に何のプログラムがされているのか分からないので，どのような動作をするのかを答えるのは無理ですが，電源やスイッチ，LED や抵抗，それにトランジスタなどの基本的な電子部品の記号の見方，電圧や電流を加えたときの動作がある程度分かっていないと PIC のプログラムはできません。もし，回路の見方が分からないということでしたら，電気部品とデジタル回路について少し勉強をしてから本書を読んでいただくと理解が深まると思います。

【PIC について】

マイクロ・コントローラと呼ばれるプログラム可能なデバイスは数多く存在しますが, マイクロチップ・テクノロジー社で開発された PIC（Peripheral Interface Controller）は古くから電気工作好きの人々に利用されてきました。PIC には多くの種類があり，デジタル入出力，通信機能，アナログ入力，USB 機能など，用途に合わせて選ぶことができます。本書では，PIC16F88 を使用しています。

PIC とは Peripheral Interface Controller の略でこの名前が表すように周辺機器の制御をおこなうための小さなコンピュータです。PIC の内部には様々な仕事をする機能が組み込まれていますが，この内部に備わっている機能（＝装置）も周辺機器と呼びます。



＜仕様＞

プログラムメモリ（ROM）	: 4k ワード
データメモリ（RAM）	: 368 バイト
データ EEPROM	: 256 バイト
入出力ピン	: 16 本
タイマ（8/16 ビット）	: 2/1 個
アナログ入力	: 7 本

その他，通信機能など

図 1-2，PIC16F88

周辺機器と信号やデータのやり取りするための接続口のことを Input/Output ポート（略して，I/O ポート）といいます。また、PIC から出ている金属製の足のことを I/O ピンといいます。PIC の種類によって I/O ピンの数は様々です。PIC16F88 では 18 本のピンが出ていますが，電源：VDD，グランド：GND，リセット：MCLR，オシレータ：OSC1，OSC2 で 5 本を使用すると I/O で使用できるピンは 13 本となります。ただし，ピンの幾つかは I/O やアナログ入力と兼用することができるので I/O は最大で 16 本使用することができます。

例えば、押しボタンなどを接続する場合には I/O ピンを入力として設定します。また、LED などを接続する場合には I/O ピンを出力として設定します。

例）スイッチの状態検査
右図の場合、入力ピンの状態は
スイッチを押すと Low 状態
スイッチを離すと High 状態

例）LED の点灯・消灯
右図の場合は、出力ピンの状態を
Low 状態にすると LED は点灯
High 状態にすると LED は消灯

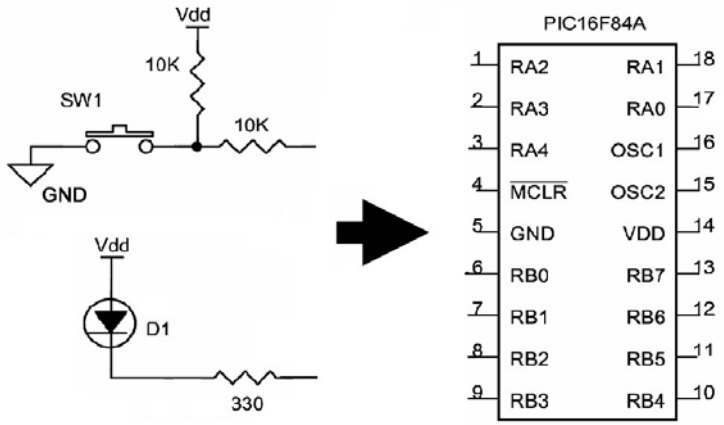


図 1-3，PIC に何を接続するかで，ピンの入出力を決める

I/O ピン毎の入出力回路の違いで入出力の動作が異なる場合があります。例えば，RA4 だけは出力として使用するとオープンドレインとなります。また，ポート B のピンを出力として使用する場合には弱プルアップの設定をすることができます。詳しくは PIC16F88 の資料を参照してください。

PIC16F88 は信号の電圧が高いか低いかの２値（デジタル：「１」or「０」）を扱えるだけでなく，電圧値（アナログ）を扱えます。このような PIC では１つのピンが複数の機能を持つことになり，それらの機能を使用するには TRIS*x* レジスタと他のレジスタも設定する必要があります。アナログ入力兼用の I/O ピンはリセット後の設定がデフォルトでアナログ入力となるものがあるので，ピンの設定は目的に応じて必ずおこないます。また，このようなピンはユーザーが I/O ピンとして使用することはできません。また，MCLR ピンや OSC ピンを I/O ピンとして使用できる PIC もあります。

周辺機器との接続口を I/O ポートといい、I/O ピンの ON/OFF を制御するレジスタ（PORTA、PORTB など）を指すこともあります。PIC の I/O ポートは特殊機能レジスタ内にメモリマップド I/O で配置されており，メモリをアクセスするのと同じ命令で周辺装置とデータをやり取りすることができます。

パソコンや PIC などの計算機の内部では全ての事柄が電圧の低いと高いで区別されます。電圧の低い状態を Low＝0，電圧の高い状態を High＝1 として扱うことを正論理といい，Low＝1，High＝0 として扱うことを負論理といいます。PIC のピンに入出力する信号は正論理で扱います。

PIC16F88 には PORTA と PORTB の 2 つのポートがあります。I/O ポートを制御するには，TRIS*x*（*x* はポート名）レジスタと PORT*x* レジスタを使用します。PIC の I/O ピンは TRIS*x* レジスタを設定することで，I/O ピンを入力または出力のどちらかで使用することができます。TRIS*x* レジスタのビットをセット(＝１)すると，そのビットに対応する I/O ピンが入力となり，I/O ピンがハイインピーダンスモードになります。TRIS*x* レジスタのビットをクリア(＝０)すると，そのビットに対応する I/O ピンが出力となり PORT*x* のビットが１なら H レベル(Vdd)，０なら L レベル（GND）の信号が出力されます。

例）ポートＢに対してライト命令やリード命令を実行したときの入出力ピンの状態

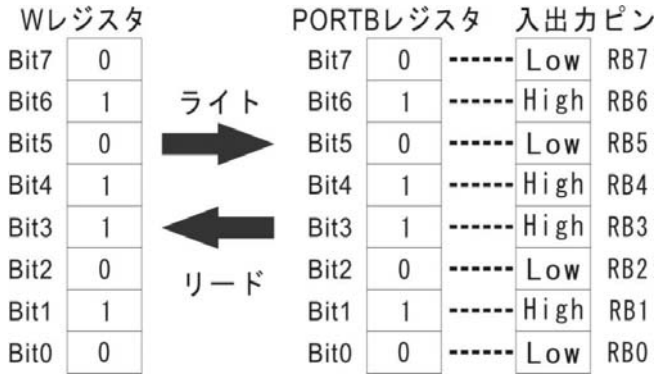


図 1-4，I/O ポートのリードとライト

図 1-3 の場合、押しボタンを押したときに入力は[0]となります。また、LED を点灯するには出力を[0]にします。

【テストボートについて】

本テキストのプログラムを動作させるためのテストボートの回路を図 2-1 に示します。

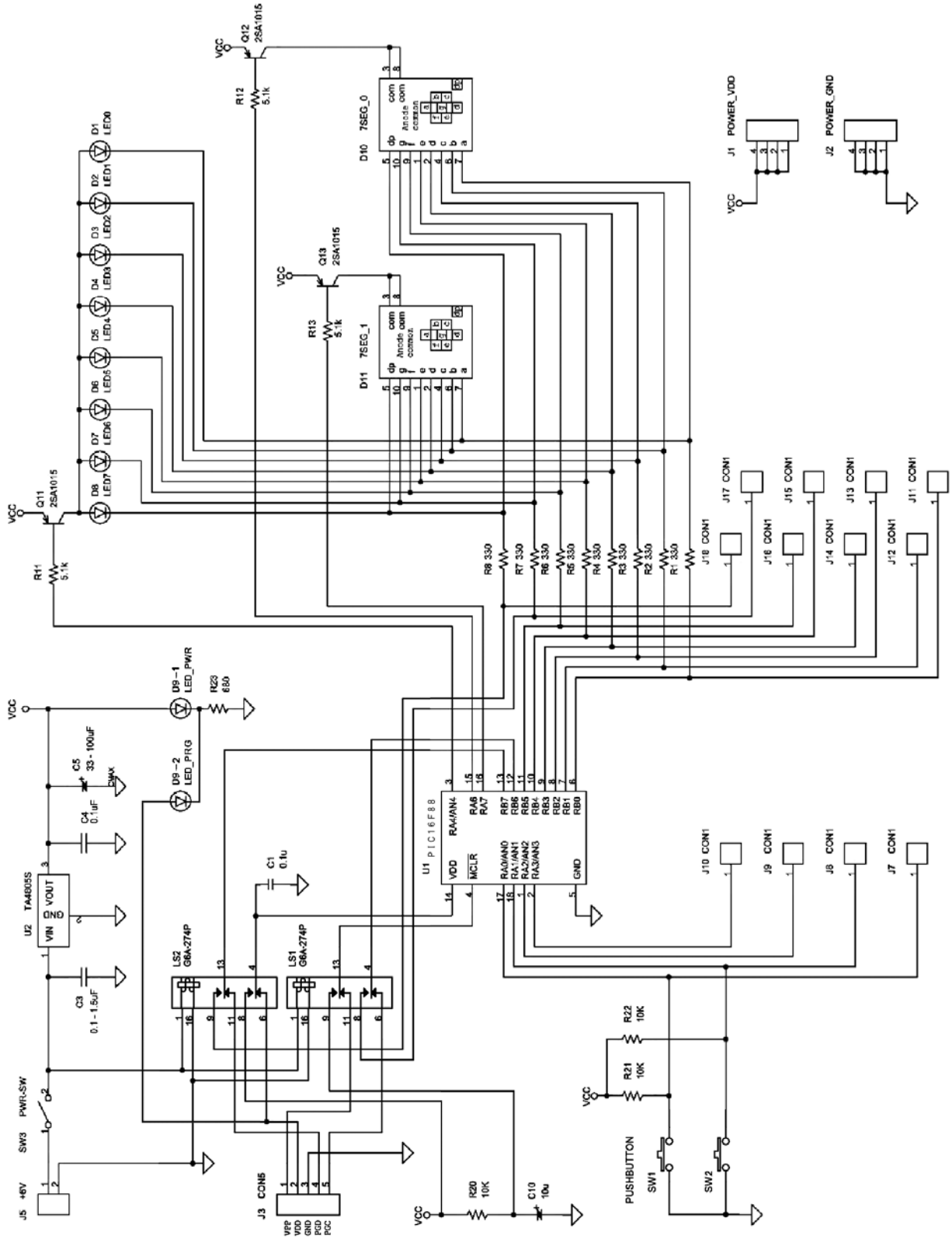


図 2-1，テストボードの回路図

図 2-2，テストボートの組み立て例です。図 2-3 はその部品配置と結線パターン図です。

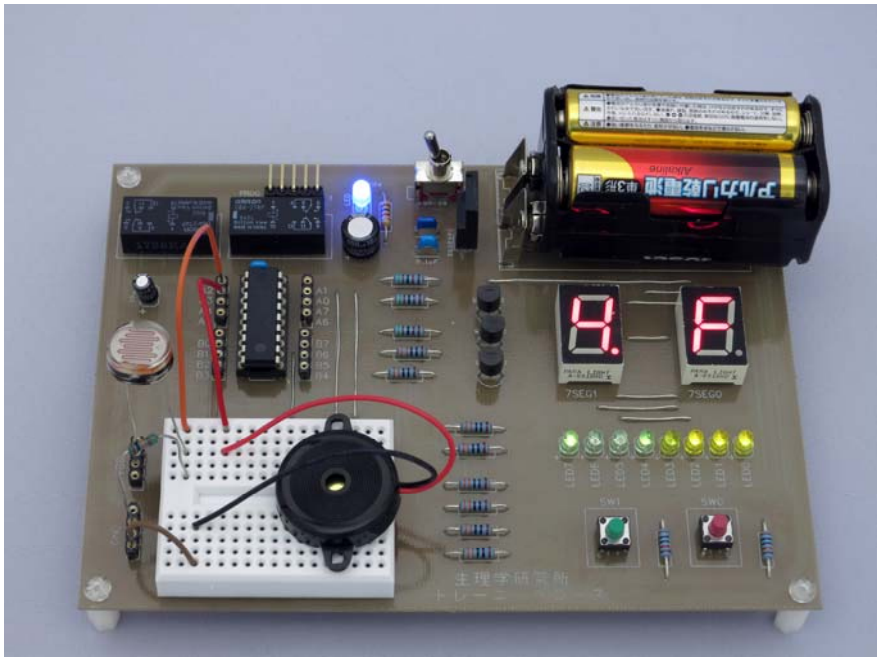
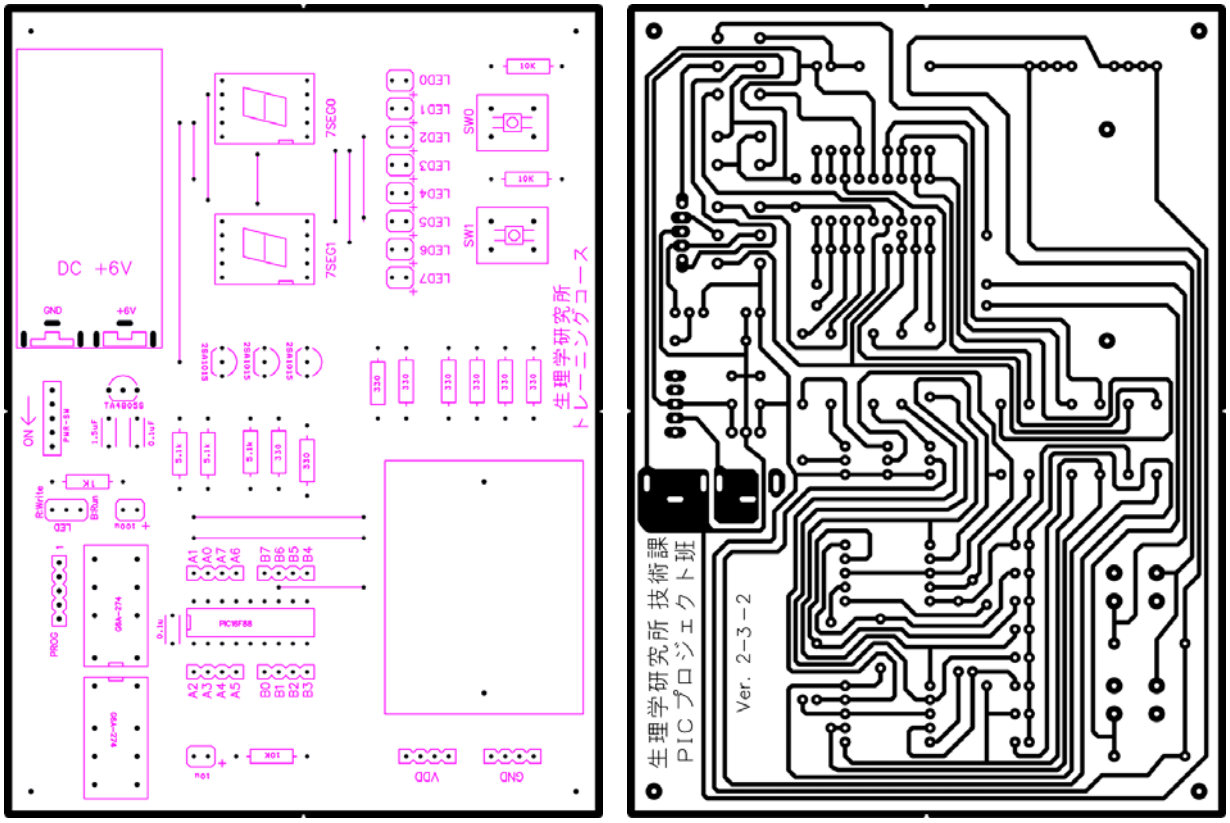


図 2-2，テストボートの組み立て例



部品面（表面）　　ハンダ面（裏面）

図 2-3，テストボードの部品配置と結線パターン例

・回路部品

部品番号	部品名称	値	部品の印字	備考
U1	PIC16F88	ー	PIC16F88	
	18 ピン IC ソケット	ー		
U2	レギュレータ	+5V	78L05	印字面手前，左から 1，2，3
C1，C3，C4	セラミックコンデンサ	0.1uF	104	
C5	電解コンデンサ	33-100uF	33-100uF	足の長い方が＋側
C10	電解コンデンサ	10uF	10uF	足の長い方が＋側
Q11，Q12，Q13	トランジスタ	2SA1015	A1015	印字面手前，左から 1，2，3
D1～D8	LED	ー		足の長い方が＋側
D9	2 色 LED	ー		足の長さに注意、向きあり
D10，D11	７セグ LED	ー		上下方向に注意
R1～R8	抵抗	330Ω	橙橙茶 金	
R11，R12，R13	抵抗	5.1KΩ	緑茶赤 金	
R20，R21，R22	抵抗	10KΩ	茶黒橙 金	
R23	抵抗	680Ω	青灰茶 金	
SW1，SW2	プッシュスイッチ	ー		
SW3	トグルスイッチ	ー		
J1～2，J11-18	シングル IC ソケット	ー		
J3	L 型ピンヘッダ　5Pin	ー		
J5	電池スナップ	ー		形状に注意
LS1,LS2	ミニリレー	ー	G6A-274P	オムロン社製
その他	プレップボード			演習で使用
	ジャンパ線			演習で使用
	CdS セル			演習で使用
	ブザー			演習で使用
	100Ω抵抗			演習で使用
	10KΩ抵抗			演習で使用
	押しボタンスイッチ			演習で使用

PIC がプログラムを処理するためにはクロックが必要になります。クロックは心臓の鼓動に相当し規則正しいリズムを刻んで CPU に命令を処理するタイミングを知らせます。この CPU クロックを発生する装置を発振器や発振回路といいます。クロックはそのままの周波数で使われたり，元の周波数に対して 1／n に分周したり n 倍に逓倍したりして利用します。他の書籍などで PIC には発振回路（発振器やオシレータなど）が必要と書いてあり，図 2-1 と回路部品には発振に相当するものが無いのを見て変に思われた人がいるかもしれません，PIC16F88 には発振器が内蔵されており最高 8MHz のクロックを内部で発振させることができます。本書の演習では内蔵発振器を利用しています。

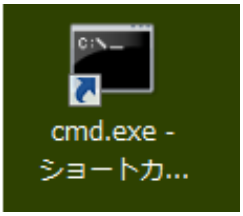
【開発環境のインストール】

1）コマンド プロンプトの用意

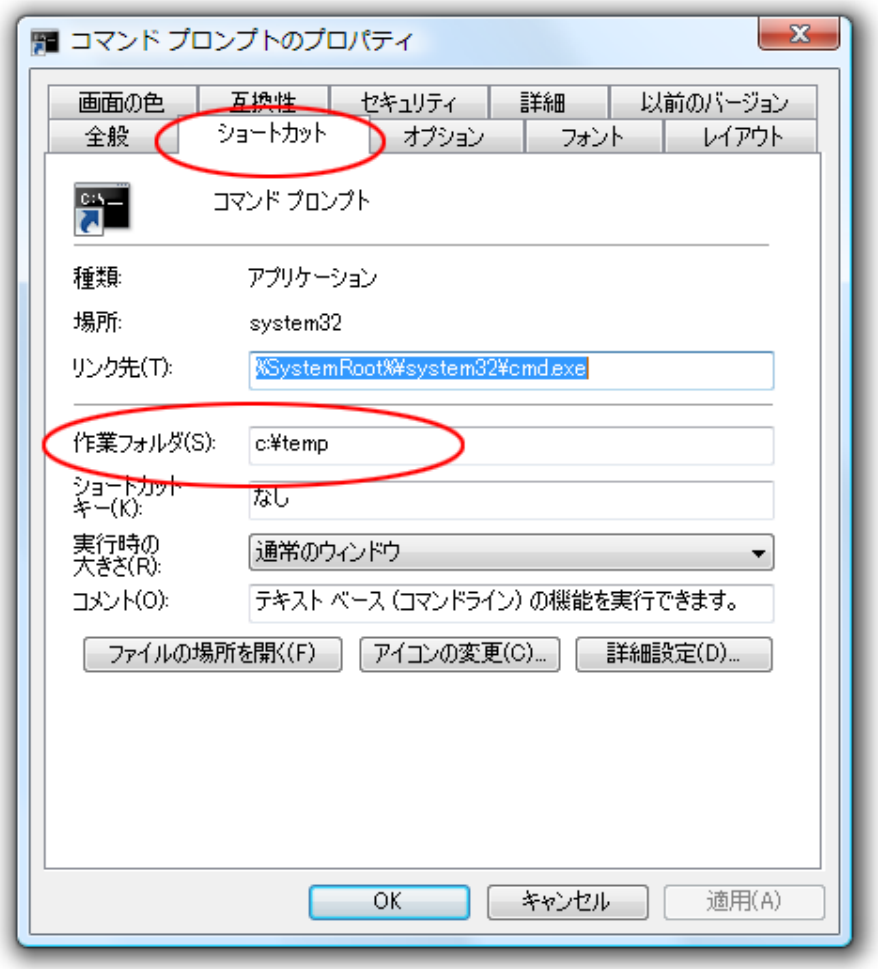
適当な場所に作業用のフォルダ（ディレクトリ）を作成します。

以下の説明は，ドライブ［Ｃ：］の下にファルダ［ｔｅｍｐ］を作成した場合です。

コマンド プロンプト（[スタート]メニュー →［プログラム］→［アクセサリ]の中にあります）のショートカットのコピーを作る。または，「C:¥Windows¥System32¥CMD.EXE」のショートカットを作ります。



プロパティを表示して，「ショートカット」タブの「作業フォルダ」を，先に作成した[c:¥temp]フォルダに変更します。



２）C 言語の開発環境の用意

PIC のプログラム開発には，C 言語のコンパイラやリンカなどのソフトウエアが必要になります。それに，作成したプログラムを PIC に書き込むための PIC ライタなどのハードウエアが必要になります。ソフトウエアについてはフリーで入手することができます。

・本書ではプログラム開発には Small Device C Compiler（以下，SDCC）を使用します。

・コンパイラ

http://sdcc.sourceforge.net/

・アセンブラ，リンカ

http://gputils.sourceforge.net/

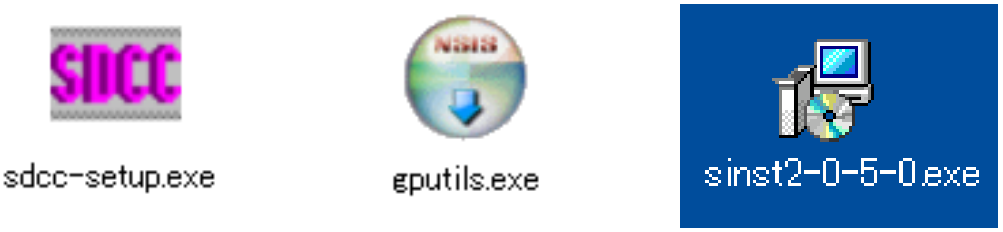
・ テキストエディタ，メモ帳などテキスト編集できるものなら何でも良いです。

参考）サクラエディタ

サクラエディタプロジェクト

http://sakura-editor.sourceforge.net/

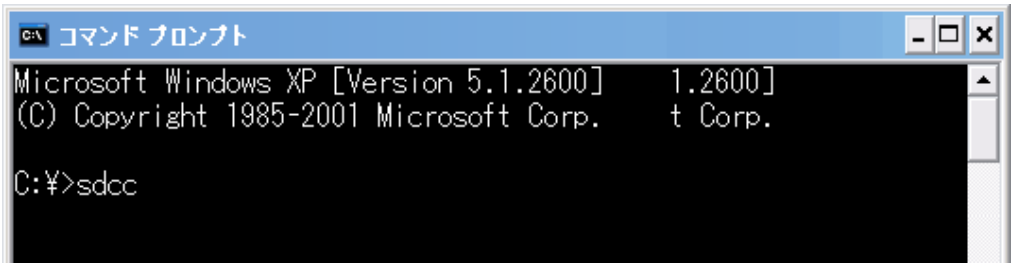
・ダウンロードしたコンパイラとリンカプログラム、テキストエディタをインストールします。

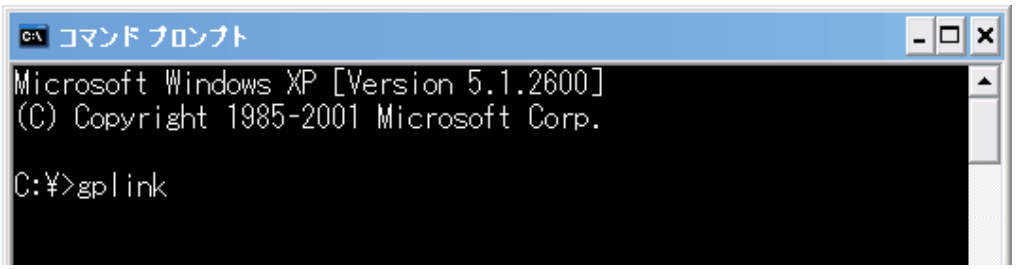


コマンド プロンプトを起動して，インストールができているか確認します。

「スタート」－「プログラム」－「アクセサリ」の「コマンド プロンプト」を選択する。または，「スタート」－「ファイル名を指定して実行]の名前に「ｃｍｄ」と入力して「OK」します。

DOS 窓が表示されたら「>」の後に，「ｓｄｃｃ」とタイプしてヘルプが表示されることを確認してください。次に，「ｇｐｌｉｎｋ」とタイプしてヘルプが表示されることを確認してください。





・バッチファイルの作成
バッチファイルは，コンパイル時の入力作業を軽減するために作成します。

テキストエディタを起動して,下記の内容を入力します。
入力が終わったら，作業フォルダに［cc.bat］の名前で保存します。

```
@echo off
sdcc -V -mpic14 -p16f88 %1
```

コンパイルのオプションについては，「コンパイルの仕方」のページを参照してください。

・プログラムライタの用意
PIC にプログラムを書き込むには専用の装置（プログラムライタ）が必要です。
テストボードには、マイクロチップ社（Microchip Technology Inc.）の PICkit2 または PICkit3 を直接接続できるように端子を設けています。

【プログラムの開発】

SDCC コンパイラについて

コンパイラにはコンパイラ毎の特有な命令や記述の仕方があります。
以下の内容は SDCC で有効なものであり，他のコンパイラでは異なる場合があります。

・サポートするデータ型

データの型	ビット長	データの範囲
unsigned char	8 ビット	0 ～ 255
(signed) char	8 ビット	-128 ～ 127
unsigned short int	16 ビット	0 ～ 65,535
(signed) short または int	16 ビット	-32,768 ～ 32,767
unsigned long	32 ビット	0 ～ 4,294,967,295
(signed) long	32 ビット	-2,147,483,648 ～ 2,147,483,647
float	32 ビット	3.4 * (10 の-38 乗) ～ 3.4 * (10 の+38 乗)

・レジスタへのアクセス

レジスタへ値を読み書きしてペリファラルを制御します。SDCC ではレジスタ名を変数のように扱えるように，ヘッダファイルで定義されています。

例）値の書き込み

```
PORTA = 0xaa;        //ポートＡに 0xaa（10 進では 170）をセットします
```

値の読み込み

```
unsigned char i;     //変数 i を unsigned char 型（8bit）で宣言
i = PORTA;           //ポートＡから値を取り出します
```

・ヘッダファイル

特殊機能レジスタ(SFR)にアクセスするには，ヘッダファイルで定義されているマクロ名を変数名（unsigned char 型)として使用します。定義されている変数名は PIC の仕様書に記載されているレジスタ名やビット名と同一になっています。

例えば，ポートやレジスタなどから信号を入出力する場合には以下のように記述する。

ポートおよびレジスタ指定

```
入力：  ｉ ＝ ＰＯＲＴＸ；        //Ｘは各ポートを示す，例えば，ＰＯＲＴＡなど
出力：  ＰＯＲＴＸ ＝ ｊ；        //Ｘは各ポートを示す，例えば，ＰＯＲＴＡなど
```

ビット指定

```
入力：  ｉ ＝ ＲＸ４；          //Ｘは各ポートを示す，例えば，ＲＡ０など
出力：  ＲＸ４ ＝ ｊ；          //Ｘは各ポートを示す，例えば，ＲＡ０など
                                //数字はポートのビット番号
```

コンパイルの仕方

1）先に作成しておいたコマンド プロンプトのショートカットから DOS 画面を起動します。
コマンド プロンプトを使い慣れていない方は，ファイルのコピーやディレクトリ（フォルダ）の作成などは WINDOWS のファイルマネジャーなどを使用してください。

・フォルダの移動
書式 >cd ファルダ名　　一つ下のフォルダ名のフォルダへ移動
>cd ..　　一つ上のフォルダへ移動

・ディレクトリ内のファイル名の表示
書式 >dir

2）DOS 画面で SDCC のコマンドを入力します。

書式 >sdcc *制御オプション* -m *デバイスの種類* -p *デバイス名* *ソースファイル名*

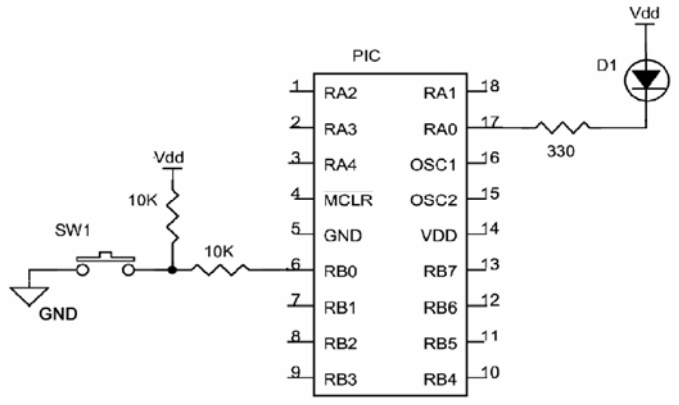
制御オプションには以下のものがあります。
-V ：コンパイルの進行状況を表示します。
-c ：コンパイルとアセンブルの後にリンクをしない。オブジェクトファイル(.o)を出力します。
オプションを省略すると，コンパイル，アセンブル，リンクがおこなわれ(.hex)ファイルができます。小規模なプログラムでソースファイルが１つのときにはオプションを省略します。規模の大きなプログラムを作成するときには，ソースファイルを分割して個別にコンパイルとアセンブルをおこない（.o）ファイルを生成し，全てのコンパイルがすんだらリンカで(.hex)ファイルを生成します。
-S ：アセンブラソース(.asm)を出力します。
-o ：出力ファイル名を指定します。
・デバイスの種類に PIC16F88 の場合は，「pic14」とします。
・デバイス名に PIC16F88 の場合は，「16F88」とします。

PIC16F88 のソースファイルをコンパイルする場合
１個のソースファイルをコンパイルからリンクまでする場合，
>sdcc -V -mpic14 -p16f88 *ファイル名*.c
コンパイルが正常に終われば，「*ファイル名*.HEX」というファイルができます。

複数のソースファイルのコンパイルをおこない後からリンクする場合
ファイルを個別にコンパイルします。
>sdcc -c -mpic14 -p16f88 *ファイル名１*.c
>sdcc -c -mpic14 -p16f88 *ファイル名２*.c
オブジェクトファイル，*ファイル名１*.o と *ファイル名２*.o をリンクします。
>gplink -I"c:¥program files¥SDCC¥lib¥pic"
-w -r -o *出力ファイル名* *ファイル名１*.o *ファイル名２*.o libsdcc.lib pic16f88.lib
例)
>sdcc -c -mpic14 -p16f88 *main.c*
>sdcc -c -mpic14 -p16f88 *sub.c*
>gplink -I"c:¥program files¥SDCC¥lib¥pic" -w -r -o*nips* *main.o sub.o* libsdcc.lib pic16f88.lib
リンクによって，nips.hex ファイルができます。

SDCC のプログラム書式

・I/O ポートの制御例



＜回路説明＞

・出力

LED-D1 の点灯／消灯

D1 を点灯するには RA0 を Low

D1 を消灯するには RA0 を High

・入力

スイッチ-SW1 の状態

SW1 が OFF のとき RB0 は High

SW1 が ON のとき RB0 は Low

A）上図の回路で LED-D1 を点灯する。

ポートＡの入出力モードを設定する。

特殊機能レジスタ(SFR)の TRISA レジスタの RA0 にあたる Bit0 の値を 0（出力モード）にします。

例）

```
TRISA =  0x00;  //この例では，ポートＡの RA0 以外のビットも出力に設定される。
```

RA0 のピンを Low（電圧が低い状態）にするため，ポート A の Bit0 に 0 を設定する。

例）

```
PORTA = 0x00;   //ポートＡの RA0 を Low:0 に設定,
                //この例では，ポートＡ全体に対して 0 を設定している
```

または

```
RA0 = 0;        //ポートＡの RA0 に対してだけ 0 を設定している
```

B）上図の回路でスイッチ-SW1 の状態を取得する

ポート B の入出力モードを設定する。

特殊機能レジスタ(SFR)の TRISB レジスタの RB0 にあたる Bit0 の値を 1（入力モード）にします。

例）

```
TRISB =  0x01;  //ポートＢの RB0 を入力に設定，この例では他は出力に設定される。
```

RB0 のピンの状態「スイッチ ON（Low のとき 0），スイッチ OFF（High のとき 1）」を取得する。

例）

```
int  swchk;
swchk = PORTB;                          //ポートＢレジスタのリード
if （ （ swchk & 0x01 ） == 0x00 ）{....}  //Bit0 の値を 0 かどうかチェック
```

または

```
if （ （ PORTB & 0x01 ） == 0x00 ） {....} //ポートＢをリードして，Bit0 の値をチェック
```

または

```
if （ RB0 == 0 ）{....}          //ポートＢの Bit0 をリードして，その値をチェック
```

・割り込み処理

割り込み処理は次のように記述します。

```
void Intr(void) __interrupt 0
{
        //割り込み内の処理
}
```

または

```
void Intr(void) interrupt 0
{
        //割り込み内の処理
}
```

「Intr」は関数名ですが，「Intr」でなくても構いません。
「__interrupt」または「interrupt」の後にはスペースを空けて割り込みレベル値を記述します。PIC16F88 などの 14bit の PIC では「0」レベルしかありません。

（重要）
割り込みレベルはレベル０の１つだけで，全ての割り込みはここにジャンプします。割り込み処理内（ void Intr(void) __interrupt 0 {} ）で何の割り込みが発生したか判定をおこない，割り込みが発生した要因毎にプログラムを記述します。

・ビット反転処理について
左側の C ソースをコンパイルすると，右側のようにアセンブラのコードが作られます（アセンブラのコードは処理の関係で毎回このようなものができるとは限りません)。この例では，４項目と７項目のアセンブラのコードができていません。SDCC においてはレジスタ（I/O ポート）の１ビットを示す変数を反転することはコンパイルエラーとならずアセンブラコードができないので注意すること。５項目のようにすることでビットの反転ができます。

	C 言語のコード	参考，コンパイル後のアセンブラコード
１：	PORTB = 0x00;	BCF STATUS, 5 CLRF _PORTB
２：	RB0 = 0x01;	BSF (_RB0 >> 3), (_RB0 & 7)
３：	PORTB = ~PORTB;	COMF _PORTB, W MOVWF _PORTB
４：	RB0 = ~RB0;	ビットの反転はプログラムコードができない。
５：	RB0 = RB0 ^ 1;	MOVLW (1 << (_RB0 & 7)) XORWF (_RB0 >> 3), F
６：	RA0 = RB0;	BCF (_RA0 >> 3), (_RA0 & 7) BTFSC (_RB0 >> 3), (_RB0 & 7) BSF (_RA0 >> 3), (_RA0 & 7)
７：	RA0 = ~RB0;	ビットの反転はプログラムコードができない

・PIC の特殊機能，SDCC でのコンフィグレーション指定

コンフィグレーションをプログラムで指定するには，

「int __at 0x2007 __config =」の後に指定したい項目を「&」で続けます。

SDCC の「PIC16F88.H」で定義されているコンフィグレーション

Configuration 1 Options は「int __at 0x2007 __config1 = 」に続けます。

_CP_ALL	Code protected (all protected)
_CP_OFF	Code protection off
_CCP1_RB0	CCP1 function on RB0
_CCP1_RB3	CCP1 function on RB3
_DEBUG_OFF	In-Circuit Debugger disabled
_DEBUG_ON	In-Circuit Debugger enabled,
_WRT_PROTECT_OFF	No program memory write protection
_WRT_PROTECT_256	First 256 program memory protected
_WRT_PROTECT_2048	First 2048 program memory protected
_WRT_PROTECT_ALL	All of program memory protected
_CPD_ON	Data EE memory code-protected
_CPD_OFF	Code protection off
_LVP_ON	Low-Voltage Programming enabled
_LVP_OFF	Low-Voltage Programming disabled
_BODEN_ON	Brown-out Reset enabled
_BODEN_OFF	Brown-out Reset disabled
_MCLR_ON	MCLR-pin function is MCLR
_MCLR_OFF	MCLR-pin function is digital RA5 or VDD
_PWRTE_OFF	Power-up Timer disabled
_PWRTE_ON	Power-up Timer enabled
_WDT_ON	Watchdog Timer enabled
_WDT_OFF	Watchdog Timer disabled
_EXTRC_CLKOUT	External Resistor/Capacitor Fosc/4 RA6
_EXTRC_IO	External Resistor/Capacitor I/O on RA6
_INTRC_CLKOUT	Internal Oscillator CLKOUT-RA6 and I/O-RA7
_INTRC_IO	Internal Oscillator I/O-RA6 and I/O-RA7
_EXTCLK	External Clock with I/O-RA6
_HS_OSC	HS oscillator
_XT_OSC	XT oscillator
_LP_OSC	LP oscillator

Configuration 2 Options は「int __at 0x2008__config2 = 」に続けます。

_IESO_ON	Internal External Switchover mode enabled
_IESO_OFF	Internal External Switchover mode disabled
_FCMEN_ON	Fail-Safe Clock Monitor enabled
_FCMEN_OFF	Fail-Safe Clock Monitor disabled

例）オシレータモード：HS，ウオッチドック：OFF，パワーアップタイマ：ON の場合

```
int __at 0x2007 __config1 = _HS_OSC & _WDT_OFF & _PWRTE_ON;
```

・データメモリについて

PIC16F88 の汎用レジスタの個数は 368 個（368 バイト）のため必要以上の変数や配列を取らないようにします。数値を変更することの無い変数や配列のテーブルデータなどは参照専用宣言（const）することでプログラム ROM 領域に置くようにします。

例）テーブルデータの作成

```
char data[4] = {0,1,2,3};          //汎用レジスタ RAM 領域に確保される
const char data[4] = {0,1,2,3};    //プログラム ROM 領域に確保される
```

・I/O ポートのアクセス

PIC の I/O ピンへのアクセスは、PORT レジスタへのライト（書込み）やリード（読込み）でおこないます。PORT レジスタはリード・モディファイ・ライトで処理されるので、ライト命令であってもリード処理の後にライト処理が実行されます。つまり、出力ピンとして設定している I/O ピンでも、一旦、リードにより I/O ピンの状態が読み込まれてから、目的とする値（0or1）がライトされます。

リード・モディファイ・ライトによる問題点、仮に、I/O ピンに接続された負荷の影響で信号が遅れて変化するような場合には、変化前の状態が I リードされることがあります。一部のコンパイラでは、同じ I/O ポートに対して連続して処理をおこなうソースコードを書いたときに、このような問題が起きないようにオブジェクトコードを生成するものがあります。

演習プログラム

PIC マイコンの応用システムを開発するためには，ハードウエアとソフトウエアを開発する必要があります。今回，ハードウエアにはテストボードを使用します。幾つかの演習プログラムを用意していますが，開発に慣れてきたら自身のアイデアで修正を加えて動作を確認してください。

開発手順

①テキストエディタで以下の演習のプログラムを入力してください。

②英数字でファイル名を付けて作業フォルダ（本書では，C:¥temp）に保存してください。

③コマンド画面でコンパイルをしてください。
書式　>sdcc -V -mpic14 -p16f88 *ファイル名.c*
コンパイルが終わると*ファイル名.HEX*というファイルができます。

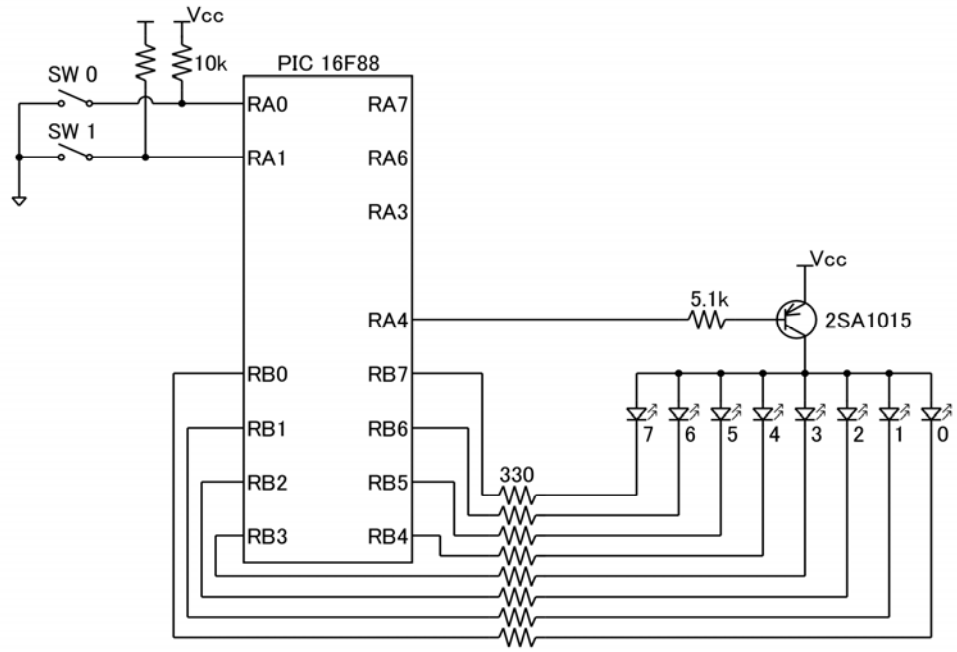
④*ファイル名.HEX のファイルを*PIC ライタで PIC に書き込みます。
PIC ライタとライタソフトの使い方については製品のマニュアルや開発者のホームページを見てください。

⑤書き込みが終わった PIC をテストボードに取り付けて動作を確認します。

⑥動作しないときには，①に戻ってプログラムを見直します。

＜演習１＞

Ⅰ．SW0 や SW1 のスイッチを押したとき，同じ番号の LED を点灯させる。

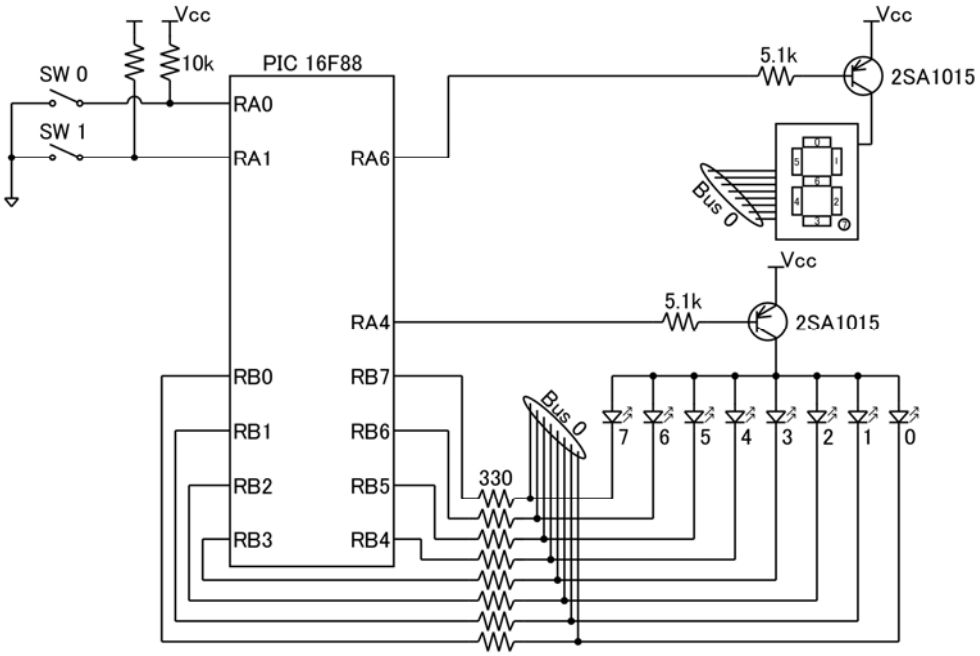



```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON& _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

// プログラムはここから始まります
void main () {

        // ここに，PIC の初期化処理を記述します
        OSCCON = 0x00;   // 内部クロック 31.25kHz で動作
        ANSEL  = 0x04;   // RA0-1,3 はデジタル入力，RA2 はアナログ入力
        TRISA  = 0x27;   // RA0-2,5 は入力，RA3-4,6-7 は出力
        TRISB  = 0x00;   // RB0-7 は出力

        PORTA  = 0xff;   // ポート A の初期化
        PORTB  = 0xff;   // ポート B の初期化

        RA4 = 0;         // RA4 のトランジスタ ON

        while(1) {       // 無限ループ
                // ポート A のスイッチ状態を読み込み，対応する LED を点灯させる
                PORTB = PORTA | 0xfc;
        }
}
```

＜演習２　ダイナミック点灯制御＞

Ⅰ．押しボタン（２進入力）に対応した数字を右の７セグメント LED に表示する。

Ⅱ．７セグメントと２進数表示の両方を点灯するように変更する。




```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

// ７セグメントの字形データ
//                    0     1     2     3     4     5     6     7     8     9
A     B     C     D     E     F
const char seg[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xd8, 0x80, 0x90, 0x88,
0x83, 0xc6, 0xa1, 0x86, 0x8e};

// プログラムはここから始まります
void main () {

        // ここに，PIC の初期化処理を記述します
        OSCCON = 0x00;   // 内部クロック 31.25kHz で動作
        ANSEL  = 0x04;   // RA0-1,3 はデジタル入力，RA2 はアナログ入力
        TRISA  = 0x27;   // RA0-2,5 は入力，RA3-4,6-7 は出力
        TRISB  = 0x00;   // RB0-7 は出力

        PORTA  = 0xff;   // ポート A の初期化
        PORTB  = 0xff;   // ポート B の初期化

        RA6 = 0;         // RA6 のトランジスタ ON

        while(1) {       // 無限ループ
                // ポート A のスイッチ状態を読み込み，対応する LED を点灯させる
                PORTB = seg[~PORTA & 0x03];
        }
}
```

＜演習３　演算＞

Ⅰ．コンパイラの演算(加算，減算，乗算，除算，除余算)の対応について確認する。

Ⅱ．num = num + 3;　の「+」の部分を「+, -, *, /, %」に変えたときに，コンパイラのメッセージや計算結果を確認する。

```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

char num;       // グローバル変数の宣言

// 7セグメントの字形データ
//                    0     1     2     3     4     5     6     7     8     9
A     B     C     D     E     F
const char seg[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xd8, 0x80, 0x90, 0x88,
0x83, 0xc6, 0xa1, 0x86, 0x8e};

// プログラムはここから始まります
void main () {

        // ここに，PICの初期化処理を記述します
        OSCCON = 0x00;  // 内部クロック31.25kHzで動作
        ANSEL  = 0x04;  // RA0-1,3はデジタル入力，RA2はアナログ入力
        TRISA  = 0x27;  // RA0-2,5は入力，RA3-4,6-7は出力
        TRISB  = 0x00;  // RB0-7は出力

        PORTA  = 0xff;  // ポートAの初期化
        PORTB  = 0xff;  // ポートBの初期化

        RA6 = 0;        // RA6のトランジスタON

        while(1) {      // 無限ループ
                // ポートAのスイッチ状態を読み込み，対応するLEDを点灯させる
                num = ~PORTA & 0x03;
                num = num + 3;
                PORTB = seg[num];
        }
}
```

＜演習４　タイマ０割り込み＞

Ⅰ．タイマ０割り込みを使用し，その回数を２進数で LED に表示する。

Ⅱ．INTERVAL，OPTION_REG，OSCCON を変更した場合の動作を推定し，確認する。

```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

#define INTERVAL 128            // 割り込み周期を決定する

static unsigned char led;       // グローバル変数の宣言

// 割り込み関数の定義
static void timer() __interrupt 0{
        TMR0 = INTERVAL;        // TMR0 カウント値設定

        led++;                  // 点灯パターンをインクリメント
        PORTB = ~led;           // LED を２進パターンで点灯

        T0IF = 0;               // フラグを倒す
}

// プログラムはここから始まります
void main(){

        // ここに，PIC の初期化処理を記述します
        OSCCON = 0x00;          // 内部クロック 31.25kHz で動作
        ANSEL  = 0x04;          // RA0-1,3 はデジタル入力，RA2 はアナログ入力
        TRISA  = 0x27;          // RA0-2,5 は入力，RA3-4,6-7 は出力
        TRISB  = 0x00;          // RB0-7 は出力

        OPTION_REG = 0x01;      // 内部クロック，プリスケーラ値設定 1:4
        TMR0       = INTERVAL;  // TMR0 カウント値設定

        PORTA  = 0xff;          // ポート A の初期化
        PORTB  = 0xff;          // ポート B の初期化

        RA4 = 0;                // RA4 のトランジスタ ON

        led = 0xff;             // 2 進パターンの初期化

        T0IE     = 1;           // タイマ割り込み許可
        GIE      = 1;           // 全体割り込み許可

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

参考）タイマ 0 の計算

クロックが 8MHz で，タイマ 0 に 10ms を設定するには以下のような計算をします。

設定したい時間(秒) ÷（クロック(秒) × 4）＝ 必要なカウント数

10ms ÷（0.125us × 4）＝ 20000

TMR0 レジスタは 8 ビットカウンタなので 256 までしかカウントできません。したがって，最適なプリスケーラを選択して TMR0 のカウント数を求めます。

必要なカウント数 ÷ プリスケーラの設定 ＝ TMR0 のカウント数

20000 ÷ 128 ＝ 156.25 ≒ 156 （プリスケーラに 1：128 を選択した場合）

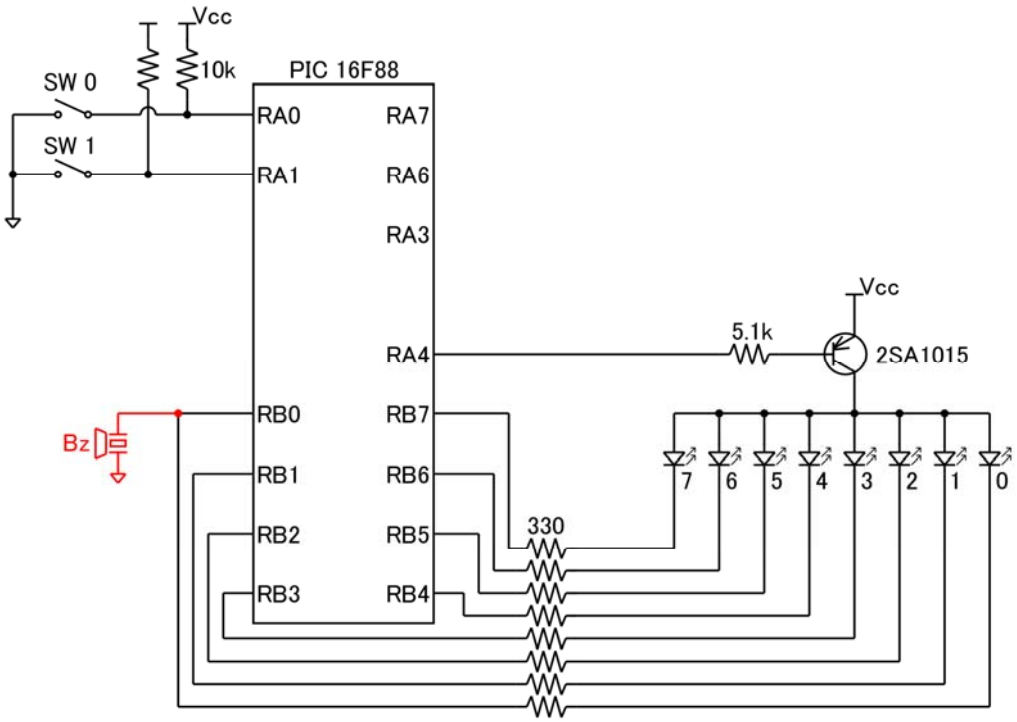
TMR0 はアップカウンタなので，255－156＝99 を設定すれば，約 10ms の時間を計ることができます。

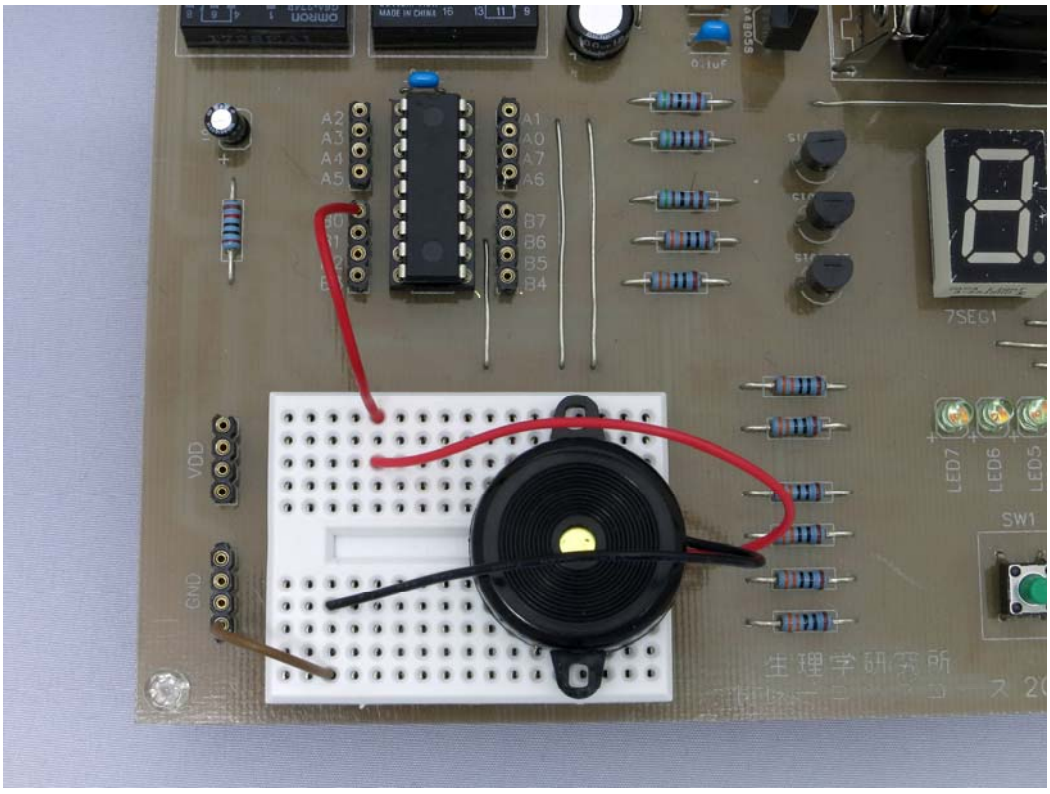
＜演習５　ブザー制御＞

Ⅰ．RB0 に圧電ブザーのプラス端子，GND にマイナス端子を接続する。

Ⅱ．タイマ０割り込みを使用して圧電ブザーを鳴らす。

Ⅲ．ボタン操作で音程が変更できるように改良する。





```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

#define INTERVAL 240            // 割り込み周期を決定する

// 割り込み関数の定義
static void timer() __interrupt 0{
        TMR0 = INTERVAL;        // TMR0 カウント値設定

        RB0 = RB0 ^ 1;          // High, Lo の反転

        T0IF = 0;               // フラグを倒す
}

// プログラムはここから始まります
void main(){

        // ここに, PIC の初期化処理を記述します
        OSCCON = 0x70;          // 内部クロック 8MHz で動作
        ANSEL  = 0x04;          // RA0-1,3 はデジタル入力, RA2 はアナログ入力
        TRISA  = 0x27;          // RA0-2,5 は入力, RA3-4,6-7 は出力
        TRISB  = 0x00;          // RB0-7 は出力

        OPTION_REG = 0x05;      // 内部クロック, プリスケーラ値設定 1:64
        TMR0       = INTERVAL;  // TMR0 カウント値設定

        PORTA  = 0xff;          // ポート A の初期化
        PORTB  = 0xff;          // ポート B の初期化

        RA4 = 0;                // RA4 のトランジスタ ON
        RB1 = 0;                // LED1 を点灯

        T0IE     = 1;           // タイマ割り込み許可
        GIE      = 1;           // 全体割り込み許可

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

＜演習６　PWM＞

Ⅰ．PWM を使用して LED の明るさやブザー音を調整する。

Ⅱ．PERIOD，DUTY の値を変化させ，LED の明るさやブザー音の違いを確認する。

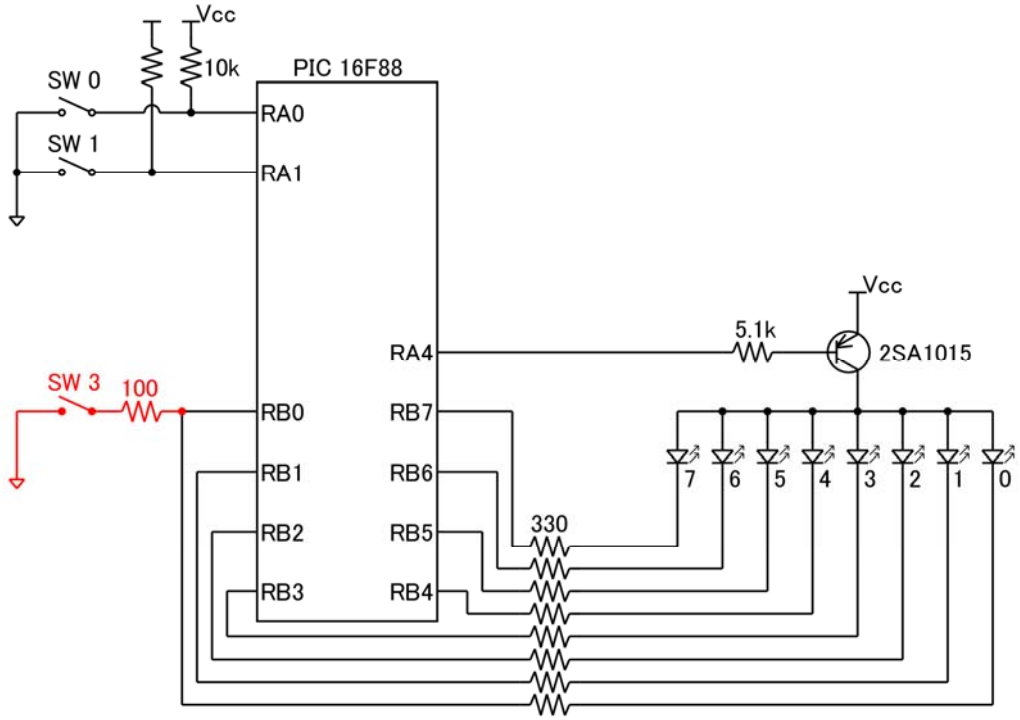
```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

# define PERIOD 96
# define DUTY   48

// プログラムはここから始まります
void main(){

        // ここに，PIC の初期化処理を記述します
        OSCCON = 0x70;          // 内部クロック 8MHz で動作
        ANSEL  = 0x04;          // RA0-1,3 はデジタル入力，RA2 はアナログ入力
        TRISA  = 0x27;          // RA0-2,5 は入力，RA3-4,6-7 は出力
        TRISB  = 0x00;          // RB0-7 は出力

        CCP1CON = 0x0c;         // CCP モジュールを PWM にする
        T2CON   = 0x02;         // TMR2 プリスケーラ 1:16
        PR2     = PERIOD;       // ピリオドを設定
        CCPR1L  = DUTY;         // デューティー比を設定
        CCP1X   = CCP1Y = 0;    // デューティー比の 9,10bit を設定
        TMR2    = 0;            // TMR2 の初期化

        PORTA  = 0xff;          // ポート A の初期化
        PORTB  = 0xff;          // ポート B の初期化

        RA4 = 0;                // RA4 のトランジスタ ON
        RB1 = 0;                // LED1 を点灯

        TMR2ON  = 1;            // TMR2 スタート

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

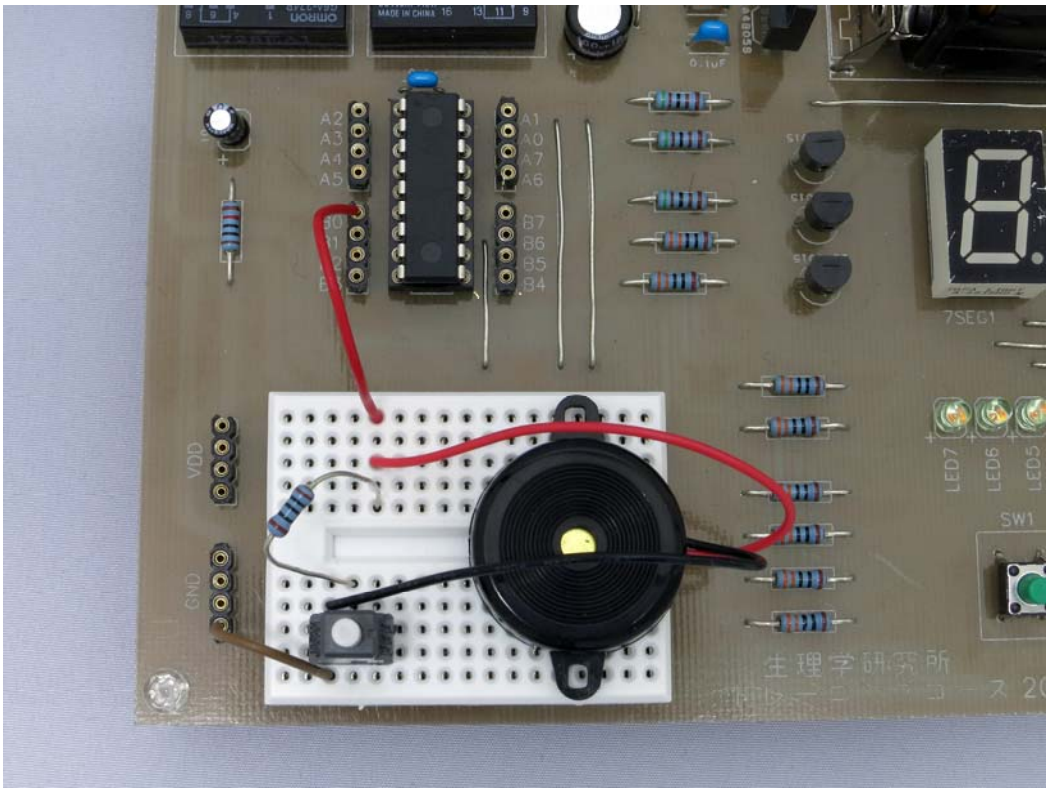
＜演習７　RB0/INT 割り込み＞

Ⅰ．抵抗 100Ω とスイッチを直列に，RB0 と GND 間に接続する(抵抗は RB0 側)。

Ⅱ．RB0/INT 割り込みを使用し，その回数を２進数で LED に表示する。

Ⅲ．タイマ０割り込みを使用し，その回数を２進数で LED 左４つに併せて表示する。



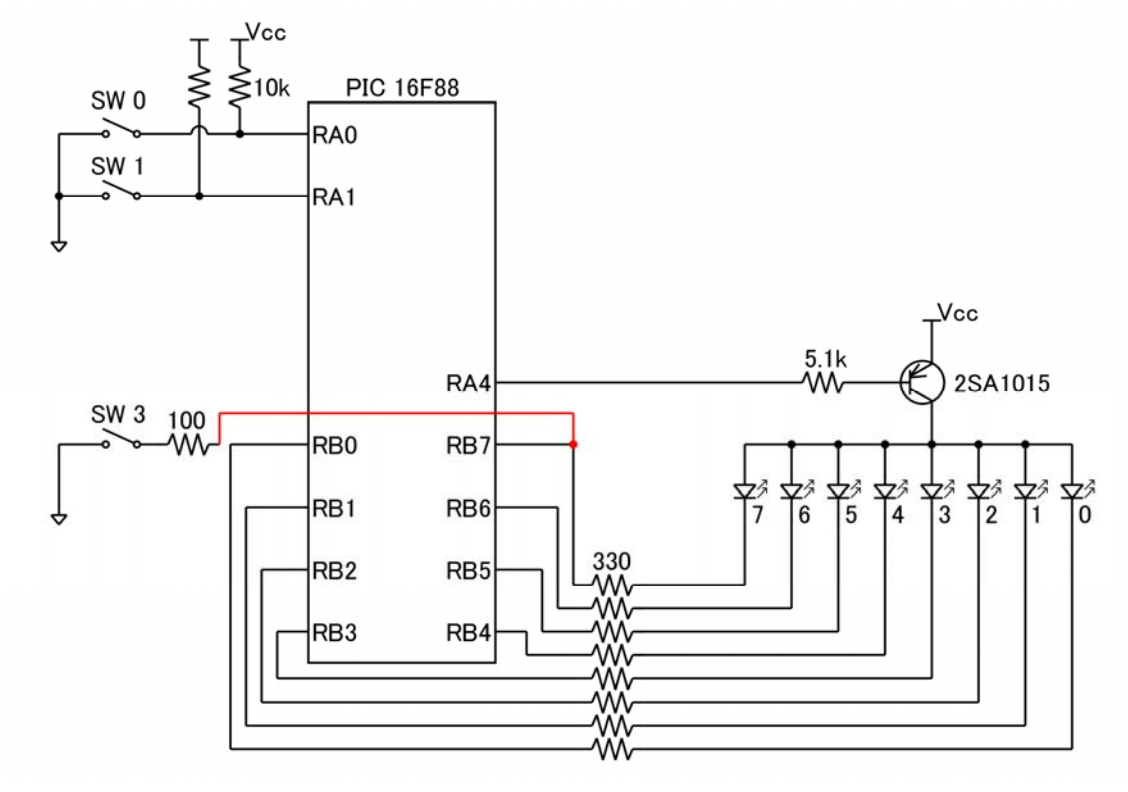


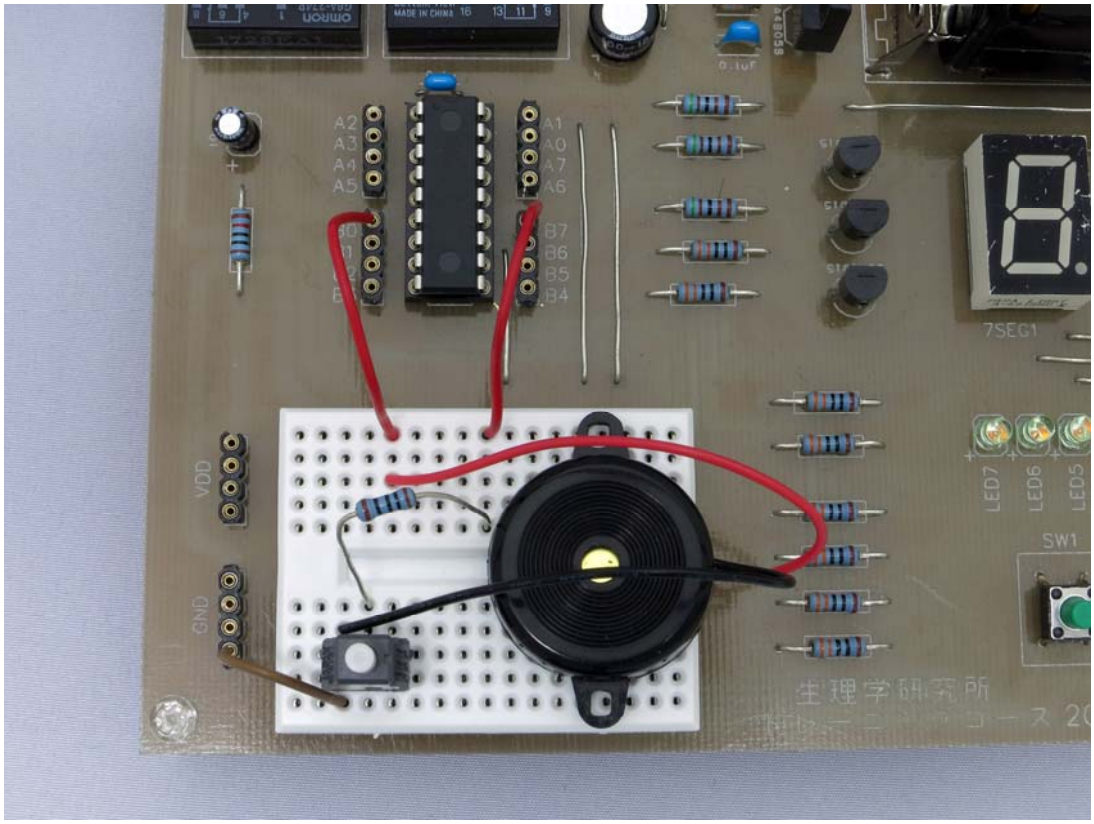
```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

static unsigned char count = 0; // グローバル変数の宣言

// 割り込み関数の定義
static void inte() __interrupt 0{
        if(count < 127){        // 表示は左へ 1bit シフトするので 7bit まで
                count++;
        }
        else{
                count = 0;
        }
        PORTB = ~(count << 1);

        INTF = 0;               // フラグを倒す
}

// プログラムはここから始まります
void main(){

        // ここに, PIC の初期化処理を記述します
        OSCCON = 0x00;          // 内部クロック 31.25kHz で動作
        ANSEL  = 0x04;          // RA0-1,3 はデジタル入力, RA2 はアナログ入力
        TRISA  = 0x27;          // RA0-2,5 は入力, RA3-4,6-7 は出力
        TRISB  = 0x01;          // RB0 は入力, RB1-7 は出力

        INTEDG = 1;             // RB0/INT の立ち上がりエッジによる割り込み

        PORTA  = 0xff;          // ポート A の初期化
        PORTB  = 0xff;          // ポート B の初期化

        RA4 = 0;                // RA4 のトランジスタ ON

        INTE   = 1;             // RB0/INT 割り込み許可
        GIE    = 1;             // 全体割り込み許可

        PORTB = ~(count << 1);  // RB0 は入力のため表示は左へ 1bit シフト

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

＜演習８　RB7:RB4 の状態変化割り込み＞

Ⅰ．抵抗 100Ω とスイッチを直列に，RB7 と GND 間に接続する(抵抗は RB7 側)。

Ⅱ．RB7:RB4 の状態変化割り込みを使用し，その回数を２進数で LED に表示する。





```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

static unsigned char count = 0; // グローバル変数の宣言

// 割り込み関数の定義
static void rbie() __interrupt 0{
        if(count < 127){        // RB7は入力のため表示は7bitまで
                count++;
        }
        else{
                count = 0;
        }
        PORTB = ~count;         // 状態を記憶する

        RBIF = 0;               // フラグを倒す
}

// プログラムはここから始まります
void main(){

        // ここに，PICの初期化処理を記述します
        OSCCON = 0x00;          // 内部クロック31.25kHzで動作
        ANSEL  = 0x04;          // RA0-1,3はデジタル入力，RA2はアナログ入力
        TRISA  = 0x27;          // RA0-2,5は入力，RA3-4,6-7は出力
        TRISB  = 0x80;          // RB0-6は出力，RB7は入力
        PORTA  = 0xff;          // ポートAの初期化
        PORTB  = 0xff;          // ポートBの初期化

        RBIE = 1;               // RB7:RB4の状態変化割り込み許可

        RA4 = 0;                // RA4のトランジスタON

        PORTB = ~count;         // 状態を記憶する

        RBIF = 0;               // フラグを倒す
        GIE  = 1;               // 全体割り込み許可

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

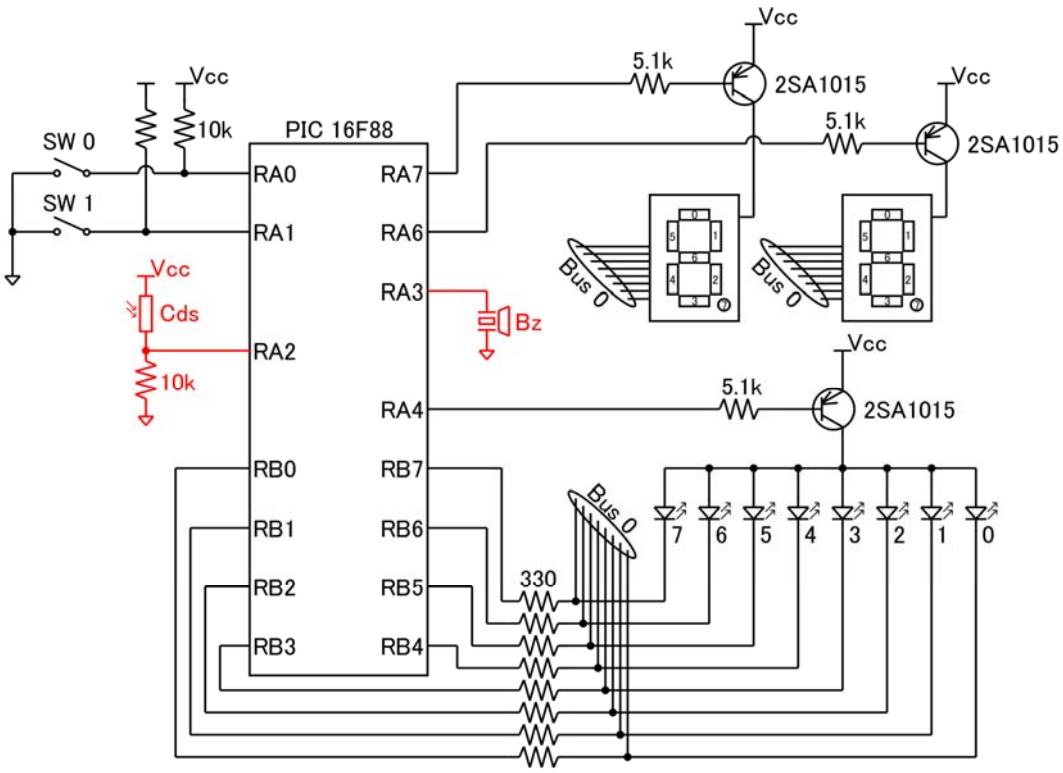
＜演習９　A/D 変換＞

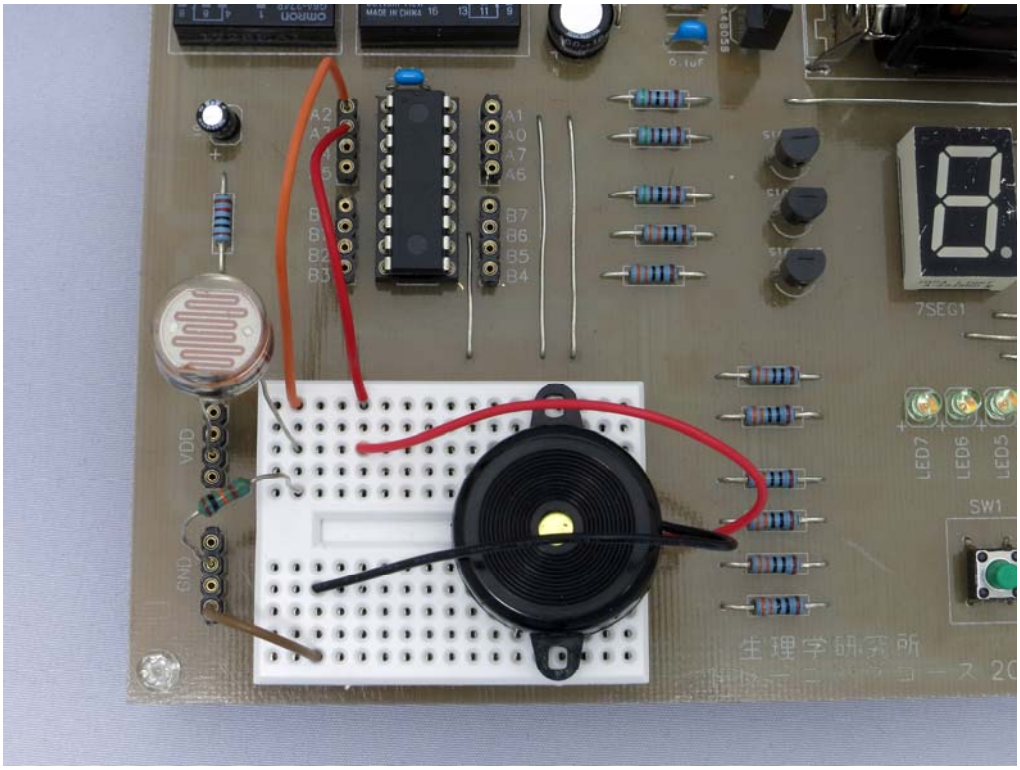
Ⅰ．RA3 に圧電ブザーのプラス端子，GND にマイナス端子を接続する。

Ⅱ．RA2 に CdS と抵抗 10kΩを接続する。残りの足を CdS は VCC に，抵抗は GND に接続する。

Ⅲ．A/D 変換を使用し，その電圧を７セグに表示する。

Ⅳ．電圧に対応した音も出すように変更する。





```
#include <pic/pic16f88.h>

// コンフィグレーションビットの設定
int __at 0x2007 __config1 = _INTRC_IO & _WDT_OFF & _PWRTE_OFF & _MCLR_ON
                          & _BODEN_ON & _LVP_OFF & _CPD_OFF & _WRT_PROTECT_OFF
                          & _DEBUG_OFF & _CCP1_RB0 & _CP_OFF;
int __at 0x2008 __config2 = _FCMEN_OFF & _IESO_OFF;

unsigned short d, v, i;                 // グローバル変数の宣言

// 7セグメントの字形データ
//                    0     1     2     3     4     5     6     7     8     9
A     B     C     D     E     F
const char seg[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xd8, 0x80, 0x90,
0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e};

// 表示ルーチン(アクィジション時間の確保)
void disp(){
        // アナログ値の読み込み (右詰10bit)
        d = (unsigned int) ADRESH << 8 | (unsigned int) ADRESL;
        v = (d * 50) / 1024;            // 電圧値の10倍に変換

        // 1Vの桁表示
        PORTB = seg[v / 10];
        RB7 = 0;
        RA7 = 0;
        for (i=0; i<100; i++);
        RA7 = 1;

        // 0.1Vの桁表示
        PORTB = seg[v - (v / 10) * 10];
        RA6 = 0;
        for (i=0; i<100; i++);
        RA6 = 1;

        // バーグラフ表示
        PORTB = 0xff >> ((v * 9) / 50);
        RA4 = 0;
        for (i=0; i<100; i++);
        RA4 = 1;
}
```

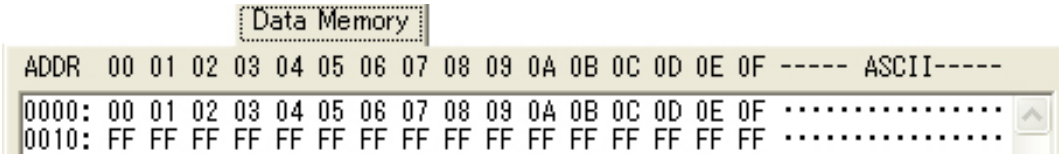
```
void main(){

        // ここに, PIC の初期化処理を記述します
        OSCCON = 0x70;          // 内部クロック 8MHz で動作
        ANSEL  = 0x04;          // RA0-1,3 はデジタル入力, RA2 はアナログ入力
        TRISA  = 0x27;          // RA0-2,5 は入力, RA3-4,6-7 は出力
        TRISB  = 0x00;          // RB0-7 は出力

        ADCON0 = 0x50;          // RA2(AN2)で AD 変換
        ADFM   = 1;             // 右詰で 10bit 出力
        ADCS2  = 1;     // ADCON0 で ADCS1=0,ADCS0=1 にしたので 16TOSC(2μS)
        VCFG1 = VCFG0 = 0;      // リファレンス AVDD-AVSS

        PORTA  = 0xff;          // ポート A の初期化
        PORTB  = 0xff;          // ポート B の初期化

        while(1){               // 無限ループ
                ADON = 1;       // AD 変換 ON
                disp();         // アクィジション時間待ち(20μS 以上)
                GO   = 1;       // AD 変換開始, フラグを立てる
                while(GO);      // フラグが倒れるまで待つ
                ADON = 0;       // AD 変換 OFF
        }
}
```

EEPROM の利用

・ＥＥＰＲＯＭからのデータの読み書きを確認する。
予め，下記のデータファイルを用意して，プログラムライタで書き込む。

```
:1042000000000100020003000400050006000700 92
:1042100008000900 0A000B000C000D000E000F00 42
```

データの確認は，プログラムライタで確認する。



//EEPROM の 0x00 番地から読み込んだデータに３を加えて 0x05 番地に書き込む。

```
#include <pic16f84.h>

// コンフィグレーションビットの設定
int at 0x2007 __config = _HS_OSC & _WDT_OFF & _PWRTE_ON;

unsigned char i;

// プログラムはここから始まる
void main()
{
        GIE = 0;                // 全割り込み禁止

        TRISA = 0x1c;   // ポートＡの設定、RA0-1＝出力, RA2-4＝入力
        TRISB = 0x00;   // ポートＢの設定、RB0-7＝出力

        PORTA = 0x03;   // トランジスタをON、PORTA=0x03でもOK

        PORTB = 0x01;   // 処理の進行状況確認のために  LED_D1:OFF

        EECON1 = 0x00;          // EECON1レジスタをクリアー
        EEADR = 0x00;           // EEPROMのアドレスを指定
        RD = 1;                 // 読み込み開始
        while( EECON1 & 0x01 ); // 読み込み処理中？
        i = EEDATA;             // EEPROMのデータを取り出す
        EECON1 = 0x00;          // EECON1レジスタをクリアー

        PORTB = 0x02;   // 処理の進行状況確認のために  LED_D2:OFF

        EEDATA = i + 3;         // 書き込みデータの準備
        EECON1 = 0x00;          // EECON1レジスタをクリアー
        EEADR = 0x05;           // EEPROMのアドレスを指定
        WREN = 1;               // 書き込み許可
        EECON2 = 0x55;          // 書き込み手順１
        EECON2 = 0xaa;          // 書き込み手順２
        WR = 1;                 // 書き込み開始
        while( EECON1 & 0x02 ); // 書き込み処理中？
        EECON1 = 0x00;          // EECON1レジスタをクリアー

        PORTB = 0x04;   // 処理の進行状況確認のために  LED_D3:OFF

        while(1);
}
```

参考文献

・PIC16F88 仕様書，マイクロチップ・テクノロジー・ジャパン㈱「http://www.microchip.co.jp/」
・SDCC についての詳細，「http://sdcc.sourceforge.net/」
・電子工作のための PIC 活用ガイドブック　後閑哲也　技術評論社
・おもしろい PIC マイコン PIC12F675 を使いこなす　中尾真治　オーム社
・たのしくできる C&PIC 実用回路　鈴木美郎志　東京電機大学出版局

平成22年 4月 第1版
平成25年10月 第2版

自然科学研究機構
生理学研究所 技術課
PICプロジェクト班