コンピュータ工学 講義プリント(12 月 4 日)

今回は、前回の講義で解説した、教科書 P.108 の図 5.2 の回路を用いて、LED の制御を行う方法について解説する。

・GPIO の利用方法

PIC16F84A にはポート A、ポート B と呼ばれる、2 つの GPIO インターフェースが搭載されている。

GPIO とは、設定により出力ピンにも入力ピンにもできる、汎用の入出力ピン(の集合)の事である。

教科書 P.20 の図 2.5 および表 2.2 を見れば分かるが、ポート A には RA0～RA4 の 5 本の端子が、またポート B には RB0～RB7 の 8 本の端子が割り当てられている。

なお、GPIO の一部のピンは、他の機能と共用になっており、設定により GPIO と他の機能とを排他的に利用する(つまり同時には利用できない)様になっている。具体的には、3 番ピンはポート A の RA4 ピンと、タイマ 0 の T0CKI(クロック入力)ピンの共用になっており、6 番ピンはポート B の RB0 ピンと、外部割込み信号入力用の INT ピンの共用になっている。

ポート A の制御には、図 1 に示すように、PORTA と TRISA の 2 つの SFR(特殊レジスタ)を使う。

表 1、ポート A の制御に関するレジスタ

アドレス	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
05H	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0
85H	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

TRISA は、ポート A の各ピン(RA0～RA4)を、入力ピンにするか、出力ピンにするかを設定するレジスタである。ピンごとに入出力の方向が設定でき、TRISA0～TRISA4 のビットに 0 を設定すると、RA0～RA4 の対応するピンが出力ピンになり、TRISA0～TRISA4 のビットに 1 を設定すると、RA0～RA4 の対応するピンが入力ピンになる。なお、電源投入時には TRISA0～TRISA4 は全て 1 に初期化されるので、RA0～RA4 までのピンは、TRISA レジスタの初期設定をするまでは、全て入力ピンとなる。

RA0～RA4 までのピンの論理レベルは、PORTA レジスタを読めば取得できる。例えば PORTA の RA2 ビットを読めば、RA2 ピンの論理レベルが読める。RA2 が入力ピンの場合(TRISA2=1)は、読んだ値は、外部から RA2 ピンに入力された信号の論理レベルになるし、RA2 が出力ピンの場合(TRISA=0)の場合は、RA2 ピンで出力している信号の論理レベルになる。

出力ピンに信号を出力したい場合は、PORTA レジスタに、出力したい論理レベルを書き込めば良い。例えば RA2 が出力ピンの場合(TRISA=0)、PORTA の RA2 ビットに 0 を書き込むと RA2 ピンから 0 が出力される。1 を書き込むと、RA2 ピンから 1 が出力される。

なお、RA2 が入力ピンの場合(TRISA=1)、PORTA の RA2 ビットに何を書き込んでも、RA2 ピンの状態は変化しない。

ポート B の制御についても、扱う SFR が、表 2 の様に PORTB と TRISB になるだけで、制御方法はポート A と同様である。

表 2、ポート B の制御に関するレジスタ

アドレス	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
06H	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
86H	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0

・LED の接続されている GPIO ピンの初期化

教科書 P.108 の図 5.2 の回路図を見れば分かるように、LED0～LED7 の 8 つの LED が、それぞれ RB0～RB7 のポート B のピンに接続されている。

全ての GPIO ピンは、電源投入時には入力ピンとして初期化されるため、そのままでは LED を点灯させる事ができない。そのため、LED の点灯を制御するプログラムを作成するには、まず RB0～RB7 のピンを全て出力ピンに設定する必要がある。

RB0～RB7 の全てのピンを出力ピンにするには、TRISB の全てのビットに 0 を設定すればよい。そのためには、

```
CLRF    TRISB
```

という命令を実行すればよい。CLRF は、教科書 P.85 に解説があるように、ファイルレジスタに 0 を代入する命令である。

ただし、ここで TRISB レジスタのアドレスが 86H である事に注意が必要である。10 月 30 日の講義でも説明したが、PIC16F84A のファイルレジスタは、バンク 0 とバンク 1 に分かれている。(教科書 P.25 参照)

STATUS レジスタの RP0 ビット(5 ビット目)を 0 にするとバンク 0 が選択され、00H～7FH の範囲のアドレスのファイルレジスタにアクセスできる。電源投入時は、バンク 0 が選択された状態で起動する。

一方で、STATUS レジスタの RP0 ビットを 1 にすると、バンク 1 が選択され、80H～FFH の範囲のアドレスのファイルレジスタにアクセスできるようになる。

TRISB レジスタはバンク 1 のレジスタであるから、バンク 1 に切り替えてから CLRF TRISB を実行しないと、06H 番地の PORTB レジスタに 0 を代入してしまうことになる。そこで、CLRF TRISB の前に、バンク 1 を選択する命令を追加する必要がある。また、CLRF TRISB の実行後に、バンク 1 を選択したままにしておくと、後々にバグを誘発する原因になりかねないので、バンク 0 に戻しておく方が無難である。

以上の事をまとめると、LED の接続されている RB0～RB7 のピンを全て出力ピンに設定するための初期化コードは、次の様になる。

```
BSF     STATUS,RP0      ; バンク 1 を選択
CLRF    TRISB           ; RB0～RB7 を出力ピンに設定
BCF     STATUS,RP0      ; バンク 0 に戻す
```

なお、BSF 命令(教科書 P.89 参照)と BCF 命令(教科書 P.88 参照)は、それぞれファイルレジスタの特定のビットに 1 または 0 を代入する命令である。

・LED 点灯プログラム(教科書 P.112 参照)

LED の点灯制御の例として、教科書 P.113 の図 5.13 に示す様なパターンで、8 つの LED の内 4 つを点灯させるプログラムを作ってみる。図 5.13 の点灯パターンにするには、RA0～RA7 を出力ピンに設定した後に、PORTB に 01010101B、すなわち 55H を代入すればよい。

図 5.13 の様に LED を点灯させた後は、無限ループで CPU を停止させるとすれば、LED 点灯プログラムは次のリスト 1 の様になる。

リスト 1、LED 点灯プログラム

```
        LIST        P=PIC16F84A
        INCLUDE     “P16F84A.INC”
        ORG         0               ; 0 番地からプログラム開始
        BSF         STATUS,RP0      ; バンク 1 を選択
        CLRF        TRISB           ; RB0～RB7 を出力ピンに設定
        BCF         STATUS,RP0      ; バンク 0 に戻す
        MOVLW       055H            ; 55H → W
        MOVWF       PORTB           ; W → PORTB（PORTB はバンク 0 のレジスタ）
WAIT    GOTO        WAIT            ; 無限ループにより停止
        END
```

教科書 P.112 のリスト 5.1 では、LED の点灯パターンを表わす 55H という定数に LEDD という名前をつけて、点灯パターンの変更を容易にしている。なお、リスト 5.1 では、CLRF PORTB により、一旦全部の LED を消灯しているが、ほとんど意味がない。

・LED 点滅プログラム(教科書 P.114 参照)

次に、LED を点滅させる事を考えてみる。全ての LED を点滅させるには、教科書 P.114 の図 5.14 のフローチャートに示すとおり、ポート B に 00H と FFH を交互に出力すればよい。(ただし、図 5.14 では、ポート B の全ピンを出力ピンに設定する処理が抜けている) これを素直にプログラムにすれば、次のリスト 2 の様になる。

リスト 2、最も単純な LED 点滅プログラム

```
        LIST        P=PIC16F84A
        INCLUDE     “P16F84A.INC”
        ORG         0               ; 0 番地からプログラム開始
        BSF         STATUS,RP0      ; バンク 1 を選択
        CLRF        TRISB           ; RB0～RB7 を出力ピンに設定
        BCF         STATUS,RP0      ; バンク 0 に戻す
LOOP    CLRF        PORTB           ; 0 → PORTB
        MOVLW       0FFH            ; FFH → W
        MOVWF       PORTB           ; W → PORTB
        GOTO        LOOP            ; ラベル LOOP にジャンプ
```

```
        END
```

ただし、これでは点滅が早すぎて、目で点滅を確認できない。

例えば LED が消えている期間の長さは、MOVLW 0FFH と MOVWF PORTB の 2 命令を実行する時間に相当するので、2 実行サイクルである。使用しているマイコンには 10[MHz]のセラロックが接続されているので、1 実行サイクルの時間は 1÷10×4=0.4[μs]となる。よって、LED が消灯している時間は 2×0.4=0.8[μs]しかない。

また、LED が点灯している期間の長さは、GOTO LOOP と CLRF PORTB の 2 命令を実行する時間に相当する。GOTO 命令の実行に 2 実行サイクルかかることを考慮すると、3 実行サイクルの間 LED が点灯している事になる。よって、0.4×3=1.2[μs]しか点灯しない。

・タイマプログラム(教科書 P.114 参照)

人間の目に、LED の点滅がはっきり認識されるためには、点灯時間、消灯時間共に、数百[ms]必要である。よって、空ループ(中で何も処理しないループ)を回して時間稼ぎをする事を考える。

そのためには、例えば教科書 P.114 のリスト 5.2 のプログラムを使えばよい。このプログラムを実行するには、図 5.16 に示す方法で実行サイクル数が計算でき、合計 249 実行サイクル必要なことが分かる。1 実行サイクルが 0.4[μs]であるから、0.4×249=99.6[μs]≒0.1[ms]の実行時間がかかる計算になる。

なお、後期中間試験には、次のリスト 3 の様なプログラムに関する出題があった。

リスト 3、試験に出たプログラム(コメント付き)

```
        LIST        P=PIC16F84A
        INCLUDE     "P16F84A.INC"
CNT     EQU         0CH     ; 変数 CNT のアドレスの宣言
NUM     EQU         0AH     ; 変数 CNT の初期値の宣言
        ORG         0       ; 0 番地よりプログラム開始
        MOVLW       NUM     ; NUM → W      (1 サイクル)
        MOVWF       CNT     ; W → CNT      (1 サイクル)
LOOP1   DECFSZ      CNT     ; CNT を 1 減らし、0 ならば次の命令をスキップ
                            ; (スキップしない場合 1 サイクル、スキップする場合 2 サイクル)
        GOTO        LOOP1   ; ラベル LOOP1 にジャンプ      (2 サイクル)
                    NOP     ; 何もしない    (1 サイクル)
LOOP2   GOTO        LOOP2   ; 無限ループにより CPU 停止
        END
```

このプログラムも、動作原理は教科書のリスト 5.2 と同様である。

リスト 3 において、MOVLW NUM を実行し始めてから NOP を実行し終わるまでの時間を求めてみよう。

変数 CNT は、MOVLW NUM と MOVWF CNT により 0AH(10 進数の 10)に初期化される。この初期化に 2 実行サイクルかかる。

DECFSZ CNT と GOTO LOOP1 の 2 命令が空ループを構成している。DECFSZ CNT が実行される度

に、CNT の内容は 09H、08H、07H、…、01H、00H と 1 つずつ減って行き、00H になったところで GOTO LOOP1 がスキップされループを抜ける。

DECFSZ は 10 回実行され、ループは 10 回まわるが、最初の 9 回のループは、DECFSZ CNT に 1 実行サイクル、GOTO LOOP1 に 2 実行サイクルの合計 3 実行サイクルかかる。また最後の 1 回のループは、GOTO LOOP1 がスキップされ、DECFSZ CNT だけが実行されるが、この場合 DECFSZ CNT の実行には 2 実行サイクル必要である。

ループを抜ければ NOP 命令の実行に 1 実行サイクル必要になる。

よって、必要な実行サイクルは、2＋3×9＋2＋1=32 実行サイクルになる。クロック周波数が 10[MHz]だと 1 実行サイクルは 0.4μs であるから、0.4×32=12.8[μs]の実行時間が必要である。

なお、CNT の初期値 NUM を変えれば、実行に必要な時間も変わる。一番実行に時間がかかるのは、初期値を 0 にした場合である。この場合、DECFSZ CNT を実行するたびに、CNT の内容は FFH、FEH、FDH、…、01H、00H と変化してゆき、256 回ループがまわる。最初の 255 回は 3 実行サイクル、最後の 1 回は 2 実行サイクルの実行時間が必要である。よって、全体では 2＋3×255＋2＋1=770 実行サイクルの時間が必要である。秒数に換算すると、0.4×770=307[μs]=0.307[ms]となる。

この様に、単純に空ループを回すだけでは、稼げる時間は 1[ms]を下回る。よって、もっと時間を稼ぐ工夫をしないと、LED の点滅は目に見えない。

・タイマサブルーチンのネスト(教科書 P.115 参照)

そこで、教科書の P.115 では、タイマサブルーチンをネスト(入れ子)にして、さらに時間を稼ぐ方法を紹介している。

教科書 P.114 のリスト 5.2 の下に RETURN 命令を付ければ、CALL TIMER1 で呼び出すと、約 0.1[ms]の時間が稼げるサブルーチンになる。さらにこのサブルーチンを 100 回呼び出すサブルーチンを作り、その開始アドレスのラベルを TIMER2 とすれば、CALL TIMER2 の実行で、約 10[ms]の時間を稼げる。さらに CALL TIMER2 を 100 回実行するサブルーチンを作り、その開始アドレスのラベルを TIMER3 とすれば、CALL TIMER3 で約 1 [s]の時間を稼げる。さらに CALL TIMER3 を 10 回実行するサブルーチンを作り、その開始アドレスのラベルを TIMER4 とすれば、CALL TIMER4 で約 10[s]の時間を稼げる。(教科書 P.115 の図 5.17 参照)

この様にして約 10 秒の時間を稼げるサブルーチン TIMER4 を作り、マイコンの起動直後に LED を消灯させ、TIMER4 で 10 秒待った後に LED を点灯させるようにしたのが、教科書 P.116 のリスト 5.3 に示すプログラムである。

また、時間を加減して 0.5 秒のタイマサブルーチンを作り、0.5 秒ごとにポートＢの値が AAH(10101010B)と 55H(01010101B)の間で交互に切り替わるようにしたプログラムが、教科書 P.117 のリスト 5.4 に示すプログラムである。

・多重ループによる長い時間のタイマ

教科書で説明されているように、タイマサブルーチンをネストすれば、確かに長い時間を稼ぐ事ができるが、大きな欠点がある。教科書 P.32 に説明してあるように、PIC16F84A のスタック領域は 8 領域しかない。よって、サブルーチンのネストは最大 8 レベルに制限される。教科書 P.114 のリスト 5.2 の TIMER4 というサブルーチンを呼べば、それだけでサブルーチンのネストが 4 レベルになる。つまり、タイマサブルーチンをネストする方法では、貴重なスタック領域を大量に消費し、ネストの深いサブルーチンからは、

タイマサブルーチンを呼び出せなくなってしまうのである。

このような場合は、多重ループを使えばよい。多重ループとはループの中に別のループを入れることで、C 言語で書けば、例えば次の様なプログラムになる。

リスト 4、C 言語による多重ループ(この場合は 3 重ループ)

```
for(int i=0; i<100; i++) {
        for(int j=0; j<100; j++) {
                for(int k=0; k<100; k++) {
                }
        }
}
```

実は、11 月 13 日に出したレポート課題の問題 3 は、2 重ループにより時間を稼ぐプログラムに関する問題であった。その問題のプログラムを、次に再掲する。

リスト 5、11 月 13 日の課題のプログラム(コメント付き)

```
        LIST    P=PIC16F84A
        INCLUDE "P16F84A.INC"
CNT1    EQU     0CH     ; 外側のループのループ変数、CNT1 のアドレスの宣言
CNT2    EQU     0DH     ; 内側のループのループ変数、CNT2 のアドレスの宣言
        ORG     0H      ; 0 番地よりプログラム開始
; 外側のループの開始
; CNT1 に初期値 05H を代入
        MOVLW   05H
        MOVWF   CNT1
; 内側のループの開始
; CNT2 に初期値 04H を代入
LOOP1   MOVLW   04H
        MOVWF   CNT2
LOOP2   DECFSZ  CNT2    ; この命令は 5×4=20 回実行される
        GOTO    LOOP2
; ここで内側のループが終了
        DECFSZ  CNT1
        GOTO    LOOP1
; ここで外側のループが終了
LOOP3   GOTO    LOOP3   ; 無限ループにより CPU 停止
```

このプログラムを、C 言語で書き直すと、以下のプログラムと等価である。

リスト 6、リスト 5 を C 言語に直したもの

```
void main(void)
{
        unsigned char CNT1,CNT2;                        /* unsigned char は
                                                           通常符号なし 8 ビット整数 */
        for(CNT1=0x05; CNT1>0; CNT1--) {                /* 外側のループの開始 */
                for(CNT2=0x04; CNT2>0; CNT2--) {        /* 内側のループの開始 */
                }                                       /* 内側のループの終了 */
        }                                               /* 外側のループの終了 */
        while(1);                                       /* 無限ループにより CPU 停止 */
}
```

この様に、サブルーチン呼び出しをネストする代わりに、ループをネストして、多重ループにすれば、無駄にスタック領域を使うことなく、長い時間を稼ぐ事ができる。