# 目的

PICマイコンを用いて、計算機の構造ろ機能を理解し、計算機システムのハードウェアとソフトウェアの関わりを理解する。

# 原理

PICマイコンPIC16F873Aに搭載されている、演算機能、メモリ、AD変換機能、IO機能を用いて、FIRST PICK ROBOを動作させる。

# 使用器具

パソコン、ライター、FIRST PICK ROBO

# 方法

パソコン上でプログラム（アセンブラまたはC）を作成し
MPLABでコンパイルし、ライターを通して
FIRST PICK ROBOへ実行ファイルを書き込んだ。

**プログラムの説明に関してはコード内のコメントで行う。**

# 実験1

LED1を光らせるプログラムをアセンブリ言語で作成し、動作させる。

```
LIST P = 16F873A;
INCLUDE P16F873A.INC;
ORG 0;

BSF STATUS, 5; //STATUSの第５bitを１に

MOVLW B'11000001';  //Wレジスタに11000001b
MOVWF TRISB;  //TRISBに11000001b

BCF STATUS, 5;  //STATUSの第５bitを0に

MOVLW B'10';  //Wレジスタに10bを
```

**結果**

LED1が光った。

# 実験２

LED1を1.00感覚で点滅させるプログラムをアセンブリ言語で作成

```
        LIST   P=16F873A;
        INCLUDE     P16F873A.INC
        ORG  0

CNTA  EQU  H'20';  //ラベルCNTAを20hに設定
CNTB  EQU  H'21';  //ラベルCNTBを21hに設定
CNTC  EQU  H'22';  //ラベルCNTCを22hに設定

                MOVLW       D'255';     //Wレジスタに255
                MOVWF       CNTA;       //CTNAに255を書き込む
                MOVWF       CNTB;       //CTNBに255を書き込む
                MOVLW       D'24';      //Wレジスタに24
                MOVWF       CNTC;       //CNTCに24を書き込む

                BSF         STATUS,5;   //STATUSレジスタの第５bitを1に
                MOVLW       B'11000001';   //Wレジスタに11000001b
                MOVWF       TRISB;   //TRISBに11000001b
                BCF         STATUS,5;  //STATUSレジスタの第５bitを0に
                MOVLW       B'00000010';   //Wレジスタに00000010b
                MOVWF       PORTB;  //PORTBに00000010b

                CALL NLOOP1;   //NLOOP1開始

NLOOP1      MOVLW       D'255';   //Wレジスタを255に
                MOVWF       CNTB;   //CNTBを255に
                GOTO    NLOOP2;  //NLOOP2へ

NLOOP2      MOVLW       D'24';   //Wレジスタを24に
                MOVWF       CNTC;  //CNTCを24に
                GOTO    NLOOP3;  //NLOOP3へ
```

```
NLOOP3    NOP;  //何もしない
          DECFSZ     CNTC,F //CNTCを１減算,CNTCが0なら次命令無視
          GOTO  NLOOP3; //NLOOP3へ

          DECFSZ     CNTB,F; //   CNTBを1減算,０なら次命令無視
          GOTO  NLOOP2 ; //NLOOP2へ

          DECFSZ     CNTA,F;  //CNTAを1減算,０なら次命令無視
          GOTO  NLOOP1; //NLOOP1へ

          MOVLW      D'255';   //Wレジスタに255
          MOVWF      CNTA;   //CNTAに255
          MOVLW      B'00000000'; //Wレジスタリセット
          MOVWF      PORTB;   //PORTBリセット   →   消灯
          GOTO       LOOP1;   // LOOP1へ

LOOP1     MOVLW      D'255'; //Wレジスタに255
          MOVWF      CNTB;  //CNTB255
          GOTO  LOOP2;  //LOOP2へ

LOOP2     MOVLW      D'24';   //Wレジスタに24
          MOVWF      CNTC;  //CNTCに24
          GOTO   LOOP3;   //LOOP3へ

LOOP3     NOP;   //なにもしない
          DECFSZ     CNTC,F   //CNTCを１減算,CNTCが0なら次命令無視
          GOTO LOOP3;   //LOOP3へ

          DECFSZ     CNTB,F;  //   CNTBを1減算,０なら次命令無視
          GOTO LOOP2;  //NLOOP2へ

          DECFSZ     CNTA,F; //CNTAを1減算,０なら次命令無視
          GOTOLOOP1; //NLOOP1へ

          MOVLW      D'255'; //Wレジスタに255
          MOVWF      CNTA;  //CNTAに255
          MOVLW      B'00000010';//Wレジスタに00000010b
          MOVWF      PORTB;  //PORTBに00000010b   →点灯
          GOTONLOOP1;   //NLOOP1へ
```

## 結果

LED1を点滅させることができた。

## フローチャート

page(　)に示す

## 考察

しかしこのコードでは実際に 1 秒ではなく1.38秒程度の間隔であったと考えれられ、計算ミスしていた。

## 実験３

実験 1 と同じ動作をするコードをC言語で作成し、
アセンブリを出力し、実験 1 のコードと比較した。

```
1:              #include<pic.h>
2:
3:              void main(){
4:                      TRISB = 0b11000001;
        0005    30C1    MOVLW 0xc1
        0006    1683    BSF 0x3, 0x5
        0007    1303    BCF 0x3, 0x6
        0008    0086    MOVWF 0x6
5:                      RB1 = 1;
        0009    1283    BCF 0x3, 0x5
        000A    1486    BSF 0x6, 0x1
6:              }

        000B    0183    CLRF 0x3
```

## 結果

上図のコードが出力された。

## 考察

実験 1 よりもバンクの切り替えに行が増えたようである。

## 実験４-(1)

T01Fの立ち上がりを監視することでLED1を
1.0秒間隔で点滅させるプログラムを作成

```
#include<pic.h>
void main()
{
	int i = 0;   //初期設定
	int j = 0;
	int flag = 0;
	OPTION = 0b11010111;  //タイマ0に割り当てスケール分周比1:256に
	TRISB = 0b11000001;   //LEDの入出力設定
	PORTB = 0b10; //点灯
	T0IF = 0b00;  //T0IFを初期化
	flag = 1; //点灯しているかどうかのフラグ

	while（20 < j){
		if(T0IF == 0b01){    //T0IFが１にセットされたら
			i = i + 1;
			T0IF = 0b00;   //T0IFの初期化
			if(i == 76){  //13.1ms × 76 =  約１秒後
				if(flag == 1){
					PORTB = 0b00; //消灯
					j = j+1;
					flag = 0;
				}else{
					PORTB = 0b10;   //点灯
					j = j+1;
					flag = 1;
				}
				i =0;
			}
		}
	}
}
```

## 結果

期待通りの動作が確認できた。

## フローチャート

page(　)に示す

## 考察

loop分を書くのはアセンブリよりもCのほうがかなり簡潔である
処理速度の問題を考慮するとアセンブラのほうが高速で
あるとは考えられるがこの程度ならCでも十分であり
生産性を含めるとCのほうが優位である。

## 実験4-(2)

LEDを１→２→３→３→４→５に順に1.0秒間隔で点灯し
続いて同じ順に1.0秒間隔で消灯していくプログラム

```
#include<pic.h>
void main()
{
	int i = 0; //初期設定
	int j = 0;
	int flag = 0;
	int k = 2;
	OPTION = 0b11010111;  //タイマ0に割り当てスケール分周比1:256に
	TRISB = 0b11000001;  //LEDの入出力設定
	PORTB = 0b10; //点灯
	T0IF = 0b00;
	flag = 1;

	while(flag < 5){ //RB5まで点灯
		if(T0IF == 0b01){
			i = i + 1;
			T0IF = 0b00;
			if(i == 76){
				PORTB = PORTB * 2;   //隣のLED点灯
				i = 0;
				flag = flag + 1;
			}
		}
	}
```

```
        T0IF = 0b00;
        while(flag > -2){
                if(T0IF == 0b01){
                        i = i + 1;
                        T0IF = 0b00;
                        if(i == 76){
                                PORTB = 64 - k;   //隣のLED消灯
                                k = k * 2;
                                i = 0;
                                flag = flag -1;
                        }
                }
        }
}
```

## 結果

期待通りの結果が得られた。

## フローチャートとプログラムについて

page(　)に示す。

## 考察

本実験において重要なのはLEDを光らせる時PORTBに対し
どの値を入力すればどのLEDの組み合わせが点灯するか
それを試行錯誤で理解し簡潔なプログラムを書くかであった。
点灯時と消灯時のLEDの組み合わせが違うので同じように
演算を行うのではうまくいかなかったのはそのせいであった。

## 実験５

短形波で1.0kHzの音を鳴らすプログラムを作成

```
#include<pic.h>
int i = 0; //初期設定
int flag = 0;
void sw(){
	if(T0IF == 0b01){   //102.4μs経過
		i = i + 1;
		T0IF = 0b00;
		if(i == 5){    //102.4μs × 5 = 約0.5ms   2サイクルで1波長周期1ms
			if(flag == 1){
				PORTA = 0;   //電圧非印可
				flag = 0;
			}else{
				PORTA = 0b00100000; //電圧印可
				flag = 1;
			}
			i =0;
		}
	}
}
int main(){
	//初期設定
	OPTION = 0b11010000;   //タイマ0に割り当て分周比1;2に
	TRISA = 0b11000001;   //PORTAを出力に
	PORTA = 0b00100000;  //RA5 = 1(電圧印可)
	T0IF = 0b00;   //初期化
	while(1){   //ループ開始
		sw();
	}
}
```

## 結果

おそらく約1.0kHzであろう音が発生した。

## フローチャート

page(　)に示す。

## 考察

音を鳴らすときに考慮すべきところは音１波長に対し
T0IFの立ち上がりが一サイクル分ではなく
２サイクル分必要なことを踏まえた上で
かかる時間を計算に設定することであった。

## 課題２

音楽を鳴らすプログラムの作成

```
#include<pic.h>


ここで音の定義と長さの定義
#define B0 20      //ÉV
#define C  19      //Éh
#define Cs 18      //Éh#
#define D  17      //Éå
#define Ds 16      //Éå#
#define E  15      //É~
#define F  14      //ÉtÉ@
#define G  12      //É\
#define A  11      //Éâ
#define B  10      //ÉV

#define HACHI     9766          //     8/16
#define ROKU      7324          //     6/16
#define YON               4882          //     4/16
#define SAN               3662          //     3/16
#define NI          2441        //       2/16
#define ICHI        1221          //     1/16
```

```
//1kHzのときの応用で音階によって切り替えの早さを変える
//音の長さも同時に計測
void onpu(int onkai, int time){
	int i = 0;
	int j = 0;
	int flag = 0;

	for(j=0;j<time;j++){
		while(1){
			if(T0IF == 1){
				i++;
				T0IF = 0;
				if(i >= onkai){
					if(flag == 1){
						PORTA = 0;
						flag = 0;
					}else{
						PORTA = 0b00100000;
						flag = 1;
					}
					i =0;
				}
			break;
			}
		}
	}
}
```

```
void main(){

        OPTION = 0b11010000;
        TRISA = 0b11000001;
        PORTA = 0b00100000;
        T0IF = 0;

    while(1){


        onpu(Ds, YON);
        onpu(Cs, YON);
        onpu(B0, ROKU);
        onpu(Ds, NI);

        onpu(Cs, SAN);
        onpu(B0, ICHI);
        onpu(B0, NI);
        onpu(B0, HACHI);
        onpu(B0, ICHI);
        onpu(Cs, ICHI);

        onpu(Ds, SAN);
        onpu(Ds, ICHI);
        onpu(Ds, NI);
        onpu(E, NI);
        onpu(Ds, SAN);
        onpu(B0, ICHI);
        onpu(B0, NI);
        onpu(Ds, NI);

        onpu(Ds, SAN);
        onpu(Cs, ICHI);
        onpu(Cs, NI);
        onpu(B0, NI);
        onpu(B0, HACHI);

    }
}
```

## 結果

音楽を鳴らすことができた。

## フローチャート

フローチャートについてはpage(  )に示す
実験５の派生のため大部分を省略した。
プログラムについてはコメントと実験５の応用

## 考察

音楽を鳴らそうとした場合、始めにdefineで長さや音階を
定義しなければ、コンパイル不可能なことがあった。
おそらくあまりに長い配列をもちいたために
可能な長さの命令語長を超えてしまったためであったと考えられる。

## 課題５

コンパイラの働きを知るためにCで書いたコードを
逆アセンブラし、その解析を行った。
次ページ以降ににそのコードと説明をしたものを示す

## 結果

以下のアセンブラが出力された

## フローチャート

page(  )~(  )に示す

## 5－1

これから掛け算を足し算で実行していることがわかる。

```
1:              #include<pic.h>
2:

3:              void main(){

4:                 char a,b,c;

5:                 a = 11;

        07F3    300B     MOVLW 0xb
 //11をWにコピー
        07F4    1283     BCF 0x3, 0x5
 //bank0に切り替え
        07F5    1303     BCF 0x3, 0x6
 //bank0に切り替え STATUSのbit6,5を00に
        07F6    00A2     MOVWF 0x22   //Wの値を0x22にコピー

6:                 b = 13;

        07F7    300D     MOVLW 0xd

        07F8    00A0     MOVWF 0x20

7:                 a++;

        07F9    0AA2     INCF 0x22, F //0x22をインクリメント

8:                 c = a*b;
```

```
        07FA    00FF     MOVWF 0x7f          //bを0x7fにコピー

        07FB    0822     MOVF 0x22, W
  //0x22(a)をWにコピー
        07FC    27E7     CALL 0x7e7
    //0x7e7へ移動
        07FD    00A1     MOVWF 0x21

9:

10:

11:             }

        07FE    0183     CLRF 0x3
---  C:¥home¥user02¥bmul.as
--------------------------------------------------------------------

        07E7    00FE     MOVWF 0x7e
                //aの値を0x7eへコピー
        07E8  3000   MOVLW 0
        07E9    187F     BTFSC 0x7f, 0
                //bのbit0が0なら次命令無視
        07EA    077E     ADDWF 0x7e, W
        //bのbit0が1ならaと0を加算（一桁目の掛け算）
        07EB    1003     BCF 0x3, 0
                //キャリーをクリア
        07EC    0DFE     RLF 0x7e, F
                //aを一桁左にずらす
        07ED    1003     BCF 0x3, 0

        07EE    0CFF     RRF 0x7f, F
                //bを一桁右にずらす
        07EF    08FF     MOVF 0x7f, F

        07F0    1903     BTFSC 0x3, 0x2
        //先の演算結果が0以外なら次命令無視


        07F1    0008     RETURN
                        //サブルーチンから復帰
```

```
07F2    2FE9     GOTO 0x7e9
```

## 考察

掛け算を足し算、つまり、普段人が用いる掛け算の筆算と
同じ要領で計算されている。

## ５－２

配列のメモリへの格納の様子がわかる。

```
1:              #include<pic.h>

2:

3:              void main(){

4:                 int a[3] = {1000,2000,3000};

        07E4    30E8     MOVLW 0xe8
         //1000=0x3e8
        07E5    1283     BCF 0x3, 0x5
                //上位bitと下位
        07E6    1303     BCF 0x3, 0x6

        07E7    00A6     MOVWF 0x26

        07E8    3003     MOVLW 0x3
        //3
        07E9    00A7     MOVWF 0x27

        07EA    30D0     MOVLW 0xd0
        //2000=0x7d0
        07EB    00A8     MOVWF 0x28

        07EC    3007     MOVLW 0x7
        //7
        07ED    00A9     MOVWF 0x29
```

```
        07EE    30B8     MOVLW 0xb8
        //3000=0xbb8
        07EF    00AA     MOVWF 0x2a

        07F0    300B     MOVLW 0xb
        //11
        07F1    00AB     MOVWF 0x2b

5:              int x,y,z;

6:              x = a[0];

        07F2    0826     MOVF 0x26, W

        07F3    00A0     MOVWF 0x20

        07F4    0827     MOVF 0x27, W

        07F5    00A1     MOVWF 0x21

7:              y = a[2];

        07F6    082A     MOVF 0x2a, W

        07F7    00A2     MOVWF 0x22

        07F8    082B     MOVF 0x2b, W

        07F9    00A3     MOVWF 0x23

8:              z = a[3];

        07FA    082C     MOVF 0x2c, W

        07FB    00A4     MOVWF 0x24

        07FC    082D     MOVF 0x2d, W

        07FD    00A5     MOVWF 0x25

9:

10:             }
```

07FE　0183　CLRF 0x3

## 考察

int型は２byteのため、各値に対して上位8bitと下位8bitが
別々で挿入されていることがわかった。
また配列は先頭アドレスを情報として持つと考えられる。

## ５－３

ポインタとメモリ番地の関係

```
1:          #include<pic.h>
2:
3:          void main(){
4:             int a[2] = {1000,2000};

        07E5　30E8　MOVLW 0xe8

        07E6　1283　BCF 0x3, 0x5

        07E7　1303　BCF 0x3, 0x6

        07E8　00A6　MOVWF 0x26
              //a[0]の番地
        07E9　3003　MOVLW 0x3

        07EA　00A7　MOVWF 0x27

        07EB　30D0　MOVLW 0xd0

        07EC　00A8　MOVWF 0x28

        07ED　3007　MOVLW 0x7

        07EE　00A9　MOVWF 0x29

5:             char b[2] = {1,2};

        07EF　01A4　CLRF 0x24
              //0x24をクリア
        07F0　0AA4　INCF 0x24, F
```

```
            //0x24をインクリメント（b[0]）
        07F1    3002    MOVLW 0x2

        07F2    00A5    MOVWF 0x25          //b[1]

6:              char c =10;

        07F3    300A    MOVLW 0xa

        07F4    00A3    MOVWF 0x23          //cの番地
7:
8:              char *p,*q;
9:              int *r;
10:
11:             p = &c;

        07F5    3023    MOVLW 0x23
            // c の番地をWにコピー
        07F6    00A0    MOVWF 0x20
            //pにcの番地を代入
12:             q = &(a[0]);

        07F7    3026    MOVLW 0x26

        07F8    00A1    MOVWF 0x21

13:             r = b;
                    //bの配列の先頭アドレスをさす
        07F9    3024    MOVLW 0x24

        07FA    00A2    MOVWF 0x22
            //bの番地をrに代入
14:
15:             q++;

        07FB    0AA1    INCF 0x21, F

16:             r++;

        07FC    3002    MOVLW 0x2
```

07FD　07A2　ADDWF 0x22, F　　//intのポインタなので2バイト分インクリメント

17:
18:　　}

07FE　0183　CLRF 0x3
//charとintのバイト数の違いからくるポインタのずれについて考察

## 考察

charのポインタは1byteなので
インクリメントするのにポイントを
１byteインクリメントするだけで良いが
intのポインタは2byteなので
2byteインクリメントする必要があることがわかる。