실험 3. LED에 생명을 부여하다

1. 목 적

컴퓨터는 기억소자에 들어 있는 내용을 순차적으로 수행하며, 통상적으로 그 결과를 기억소자에 저장되게 된다. 따라서 기억소자에 들어있는 내용이나 연산 결과를 보기 위해서는 컴퓨터 내부 내용을 외부로 연결시켜주는 창구가 필요하며, 이러한 창구 역할을 용어로 OUTPUT PORT라고 한다. 그리고 그 반대로 컴퓨터에 데이터를 주거나 제어 신호를 넣어줄 때는 외부신호를 컴퓨터 내부로 연결시키는 부분이 필요하다. 이를 INPUT PORT라고 한다. 즉 I/O PORT는 컴퓨터와 외부를 연결하는 장소가 되며, 특히 제어 기능으로 많이 활용되는 마이크로프로세서에서는 이를 잘 활용하는 것이 매우 중요하다.

앞 실험에서도 확인하였겠지만 아무리 완벽한 프로그램을 작성하여도 그 결과를 활용할 수 없다면 의미가 없는 일이 된다. 특히 assembly language로 작성되는 프로그램은 하드웨어와 밀접한 관계를 가지며, 주변 회로에 따라서 변경되며, 그 의미도 다르게 된다. 따라서 명령어 자체로만 이해하려고 해서는 아무런 의미가 없음을 다시 강조한다.

2. I/O pin 설정

우리가 실험에 사용하는 마이크로프로세서는 1 chip 프로세서이다. 이는 제한된 크기(IC PIN)에서 최대의 효용성을 부여하려면 당연히 가장 많은 I/O를 접속할 수 있도록 만들어야 할 것이다. 따라서 내부회로의 복잡성은 별개로 두고(왜냐하면 IC의 가격은 회로의 복잡함보다는 얼마나 대량 생산하느냐가 좌우됨), IC pin 1개를 여러 용도로 사용하도록 하고 있으며, 우리가 사용하는 PIC16F876은 pin 한 개를 5가지 용도로 사용하고 있으며, 따라서 이 핀을 사용하려면 먼저 어떤 용도로 사용할 것인가를 선언해 주어야한다. 즉 I/O pin의 사용 용도의 결정은 사용자가 하며, 사용자는 특수한 register를 이용하여 명령어를 설정함으로써 그 사용 용도를 결정하게 된다. 따라서 I/O pin과 이를 제어하는 명령어 사이의 관계를 이해하야만 마이크로프로세서를 사용할 수 있

다.

PIC16F876는 8bit 프로세서로 28pin짜리 IC이다. 이 중 3pin은 5V 전원(VDD, VSS*2) 인가용으로 사용되고 2pin은 마이크로프로세서를 동작시키는 기준신호를 만들어 주는 CLOCK 신호(OSC1, OSC2 : 통상적으로는 oscillator를 연결함)용이며, 1pin(/MCLR)은 마이크로프로세서를 초기화 시켜주는 RESET 신호용이다. 따라서 나머지 22pin(RC7~RC0, RB7~RB0, RA5~RA0)이 마이크로프로세서와 외부를 연결하는 통로로 사용 가능하다. 단순하게 생각하면 22개의 PIN은 많은 것처럼 보이나, 우리가 사용하는 신호는 디지털 신호로 최소 8개를 가져야만 1byte가 만들어지므로 정보전달의 의미에서 본다면 아주 적은 개수이다. 만약 이 pin이 INPUT PORT 또는 OUTPUT PORT로 고정되어 있다면 신호를 주고받기 위해서는 최소 16개가 있어야 하므로 사용 효용성을 아주 떨어뜨리게 된다. 따라서 대부분의 1 chip 프로세서에서는 I/O 단자를 INPUT 이나 OUTPUT으로 고정시키지 않고 사용자가 정의하여 사용할 수 있도록 내부회로를 구성하게된다. 그러므로 22pin 모두를 INPUT PORT로 또는 OUTPUT PORT로 사용 가능하며, 이는 프로그램 초기뿐만 아니라 프로그램 중간에서도 마음대로 변경 가능하다.

I/O pin 설정과정은 프로세서 종류마다 약간씩 다르나, 기본적인 순서는 다음과 같다.

① 먼저 단순 I/O 설정보다 특수 기능을 먼저 설정해야 한다.
② 설정은 마이크로프로세서의 내부회로를 관리(제어)하는 특수용도 register(주소와 용도가 결정되어 있는 I/O PORT : PIC 계열에서는 register file 중에서 고유 이름이 명기된 것들)를 통하여 이루어지며, data sheet를 확인해야 한다.
③ 특수 기능에 따른 pin의 사용 용도가 INPUT인가 OUTPUT인가를 확인하여 I/O 기능을 설정한다.
④ 단순 I/O 기능만 가지는 pin은 그냥 설정만 하면 된다.

INPUT PORT는 외부 신호를 마이크로프로세서의 내부로 연결시켜 주는 기능이므로 자체만으로는 기억 기능이 없으나, OUTPUT PORT는 일종의 기억소자이므로 출력되는 내용을 읽어서 확인할 수도 있고, 바로 변경하여 출력할 수 있으므로 이를 잘 활용하면 프로그램을 아주 효율적으로 작성할 수 있다.

1) I/O 설정과 관련된 명령어

TRISA, TRISC: W REG.의 값으로 I/O PORTA, PORT C의 각 BIT를 INPUT PORT로 사용할 것이냐 OUTPUT PORT로 사용할 것인가를 결정하는 특수용도 REGISTER

- DATA BIT 위치의 값이 1 이면 INPUT PIN으로 선언
- DATA BIT 위치의 값이 0 이면 OUTPUT PIN으로 선언

```
예)  MOVLW   B'00001111'
     MOVWF   TRISA      ; PORTA의 BIT 4는 OUT, BIT3,2,1,0은 IN
          ; (PORTA는 8bit 전체가 존재하지 않고 bit4,3,2,1,0 만 존재함)
     MOVLW   B'11110000'
     MOVWF   TRISC      ; PORTC의 BIT 7,6,5,4는 IN, BIT3,2,1,0은
                        ;  OUT단자로 사용
```

☞ 주의점 : TRISA, TRISC REG.는 REGISTER FILE의 BANK 1에 속해 있으므로 위의 명령어를 사용하려면 BANK를 변경해 주어야 한다. BANK 0과 BANK 1를 보면 동일한 의미를 가지는 주소가 존재하며, 이는 BANK를 변경하여도 내용이 변경되지 않음을 나타낸다. 예로 PCL 과 STATUS는 현재 program과 동작 상태를 나타내는 것이므로 BANK 의 변경에는 영향을 받지 않아야 프로세서가 정상적으로 동작될 수 있다.

2) 단순 I/O를 동작시키는 명령어

I/O를 동작시키는 명령어는 다양하게 만들 수 있다. 왜냐하면 I/O는 하나의 memory로서 동작하기 때문이다. 즉 PIC16F876 chip의 2번 pin에 1을 출력하기 위해서는 2번 pin이 RA0로, PORTA의 BIT 0이므로 PORTA를 지정하는 주소의 데이터 값 중 BIT 0의 값이 1이 되도록 해주면 된다. 구체적인 방법의 한 예는 다음과 같다.

```
예1)    MOVLW     B'1'
        MOVWF     PORTA

예2)    MOVLW     1
        IORWF     PORTA,1

예3)    BSF       PORTA,0

예4)    BSF       5,0
.
.
```

예1)는 다른 BIT 값도 전부 0으로 변경하므로 바람직하지 못한 방법이며, 이를 해결하기 위하여 예2)와 같이 비트 연산 기능을 가진 AND나 OR 기능을 사용하면 특정 비트를 1또는 0으로 만들 수 있다. 즉 ???WF f,d 명령어를 사용하여 연산 결과를 바로 PORTA에 넣는 방법이다. 그러나 마이크로프로세서 응용에서는 이러한 특정 비트를 다루는 작업이 매우 빈번하게 사용되며, 따라서 예2) 보다 더 편리하게 사용할 수 있는 방법을 요구한다. 이를 위하여 만들어진 명령어가 예3,4)에서 사용한 BSF f,b 명령어이다. BSF는 bit set file-reg.로 file-reg.의 b 비트를 강제로 1로 만드는 명령어이다. 예4)는 예3)과 동일한 것으로 앞에서 선어된 PORTA의 구체적인 번지인 5번지를 직접 사용한 것이다.

예3)과 예4)를 비교하면 어느 것이 더 프로그램을 이해하기가 쉬울까요? 막연한 숫자보다는 물리적인 의미를 가지는 약어를 쓰는 것이 눈에 잘 들어오므로 프로그램 앞에서 미리 선언을 하고 프로그램 내부에서는 구체적인 숫자(주소 또는 비트값 등)를 사용하는 대신에 선언된 약어를 쓰는 것이다. 이렇게 하면 변수의 구체적인 주소를 변경하려고 하면, 선언문만 변경하면 되기 때문에 매우 편리하게 프로그램을 작성할 수 있다. 따라서 **앞으로 프로그램 작성에서 test1.asm으로 주어진 프로그램의 앞쪽 부분(선언문)은 모든 프로그램에서 공통**으로 사용된다.

INPUT인 경우는 입력에 해당하는 번지의 값을 읽으면 되며(읽기만 하면 새로운 입력이 기억번지 안으로 들어옴), 별도의 다른 조작은 필요가 없다. 앞

실험 2에서와 같이 pin 2(이 pin이 PORTA의 bit0과 연결되어 있음)에 연결된 s/w의 상태를 읽어오기 위해서는 명령어 상으로는 단순하게 PORTA의 내용을 읽어오면 된다.

예5) MOVF PORTA,W

예6) MOVF PORTA,0

예 5,6)는 s/w의 상태를 읽어서 W-register에 가져오며, 이 중 bit0의 값이 s/w의 현재 상태이다.

☞ 주의: 부록의 명령어를 보면, **MOVF f,d 에서 d를 생략하면 '1'로 인식**되므로 주의해야 한다. 즉 **MOVF PORTA는 MOVF PORTA,1과 같아서 MOVF PORTA,0 과는 완전히 다른 명령어**가 된다. 따라서 명령어 표를 항상 보고 사용하도록 하여야 하며, 이를 편리하고 확실히 하기 위하여 d 대신에 미리 선언된 W 또는 F를 사용한다.

3) I/O 설정과 I/O를 동작시키는 완전한 명령어

I/O를 정상적으로 사용하기 위해서는 먼저 I/O 단자를 어떤 용도로 사용할 것인가를 선언하여야 한다. 그러나 마이크로프로세서는 전원이 인가되면(RESET되면), REGISTER FILE 중에서 BANK 0이 자동 선택되므로 I/O를 설정하기 위해서는 사용자가 BANK를 1로 변경하여야 한다. BANK를 관장하는 register는 'STATUS' 라고 부르는 REGISTER FILE 03 번지이며, 그 중에서 BIT 6,5이다. 이렇게 BANK를 사용하는 이유는 주어진 조건에서 많은 개수의 메모리를 확보하기 위한 방법이다. 주의할 점은 한 상태에서는 한 BANK 만 접근이 가능하므로 사용하고자 하는 register가 위치한 BANK를 알고 미리 BANK 변경을 하여 주어야 한다. 이때 BANK가 변경되면 프로세서를 관리하는 register가 변경될 수 있으므로 오동작을 하게되며, 이를 막기 위하여 프로세서의 기본 동작에 관여하는 register는 BANK 변경 후에도 값이 변하지 않도록 BANK에 무관하게 공용으로 사용되도록 선언되어 있다. Data sheet의 Fig, 2-4의 REGISTER FILE MAP을 보면 PCL(02번지), STATUS(03번지), FSR(04번지), PCLATH(0A번지) 그리고 INTCON(0B번지)

가 동일한 이름으로 두 BANK에 선언되어 있음을 볼 수 있다.

☞ PIC16F876는 BANK가 0,1,3,4로 4개이므로 이를 선택하기 위한 bit는 2개가 있어야 한다. 실험에서는 BIT 6,5가 사용되며 BIT6,5가 00이면 BANK 0, 01이면 BANK1, 10이면 2, 11이면 3을 선택한다.

그리고 I/O 설정이 끝나면 반드시 BANK 0으로 되돌아와야만 I/O PORT를 동작시킬 수 있다.

다음 프로그램은 I/O를 설정하고 입/출력을 구현한 예이다.

예7)

```
        BCF     STATUS,RP1  ' BANK 1 선택
        BSF     STATUS,RP0
        ; PORTA를 analog input과 digital in/out 등의 여러 용도
        ;로 사용 가능한 단자이며, 따라서 DIGITAL I/O로 사용시
        ;에는 아래와 같이 선언을 해 주어야 한다
        MOVLW   B'00000111'
        MOVWF   ADCON1
        MOVLW   B'00000000'   ; PORTA를 전부 DO로 선언
        MOVWF   TRISA
        MOVLW   B'00001111'   ; PORTB를 DO/DI로 선언
        MOVWF   TRISB
        BCF     STATUS,RP1  ' BANK 0 선택
        BCF     STATUS,RP0

LOOP    MOVLW   00H
        MOVWF   PORTA         ; PORTA에 전부 '0'를 출력함
        MOVLW   0FFH
        MOVWF   PORTB         ; PORTB에 '1111????'를 출력
        ; 함, 이유는 선언 부분에서 bit7,6,5,4 만 출력으로 선언했기
        ; 때문임
        GOTO    LOOP
```

프로그램을 작성할 때 중요한 기능 중의 하나가 조건에 따라서 서로 다른

여러 가지 일을 하도록 하는 것이다. 이것을 가능하게 하기 위해서는 최소한 두 가지 기능을 가지는 명령어가 필요하다. 첫째는 조건을 만들어 주는 명령어이며, 둘째는 조건에 따라서 분지(branch) 하는 명령어이다. 첫째로 상태 기억 장소(flag 또는 status)를 만들어 두고, 조건을 만들어 주면 또는 조건이 되면 그 조건이 만들어졌는가 아닌가의 확인하여야 다음 명령어로 분지가 가능하다. 기억되는 상태의 종류는 마이크로프로세서에 따라서 다르며, PIC16F84에서는 status register에 C(carry /borrow), DC(digit carry/borrow), Z(zero), PD, TO로 5가지가 있다. 일반적으로 많이 사용되는 status는 C, Z이다.

C : STATUS REG.의 BIT0으로 ADD, SUB, rotate 명령어 사용 결과 CARRY가 발생하면 '1'로 SET되고 발생하지 않으면 '0'이 된다.

DC : STATUS REG.의 BIT1로 명령어 사용 결과로 bit3에서 bit4로 자리올림이 발생하면 '1'로 SET되고, 발생하지 않으면 '0'이 된다.
(BCD 연산에서 사용하며, 일반적인 경우는 거의 사용하지 않는다)

Z : STATUS REG.의 BIT2로 명령어 사용 결과 값이 ZERO 이면 '1'로 SET되고 아니면 '0'이 된다.

☞ 참고: 어떤 명령어가 어떤 STATUS에 영향을 주는가는 기본 원칙은 있으나, 마이크로프로세서 종류에 따라서 다르다. 일반적으로 ALU와 관련된 명령어(연산이 사용된 명령어)는 STATUS에 영향을 주며, 연산 결과에 따른 통상적인 의미로 STATUS가 정해진다. 그러므로 STATUS는 사용할 때마다 명령어 표를 참고하기 바란다.

☞ 참고: 비트 단위 파일 조작 명령어는 STATUS에 영향을 안 줌
BCF, BSF, BTFSC, BTFSS

☞ 참고: STATUS의 C, DC, Z flag을 프로그램에서 선언하여 사용하는 것이 편리하며, **본 책에서는 C는 CF, DC는 DC 그리고 Z는 ZF로 선언**하여 사용한다.

4) 'C' STATUS에 영향을 주는 명령어

STATUS reg.의 C bit(bit 0)가 연산결과 CARRY의 발생 유무에 따라서 결정되며, 기본 명령어는 ALU와 연관된 명령어 들이다.

ADDWF F,d : ADDLW K --- 연산 결과 그대로 영향 받음
SUBWF F,d : SUBLW K --- 아래 주의 참고
RLF F,d : RRF F,d --- 'C'를 만드는 것뿐만 아니라 'C'를 받아 들임으로 주의 요망

명령어 표에 의하면, 'ADDLW K' 명령어를 수행시키면 C, DC, Z STATUS가 영향을 받는다고 나와 있다. 즉 명령어는 (W) + K → (W) 의 동작을 하며(여기서 (W)는 W reg.의 내용이라는 의미임), 그때 STATUS 레지스터의 변화를 아래에 나타내었다.

****

(W + K) 연산을 수행했을 때 STATUS 레지스터의 변화

(1) 연산결과 255보다 큰 값이 되어 자리 올림이 발생된 경우 C=1
(2) 연산결과 W reg,의 결과가 0이 되는 경우 Z=1
(3) 연산결과 bit3에서 bit4로 자리 올림이 발생된 경우 DC=1

****

따라서 ADD 명령어는 일반적인 의미로 STATUS reg.가 설정된다. 그러나 'SUBLW K' 명령어는 K - (W) → (W) 의 동작을 하며, 그때 W reg.와 STATUS 레지스터의 변화는 아래 예와 같다.

예로 'W=05' 일 때 ' SUBLW 07' 를 수행시키면 07 - 05 로 결과는 W에 02 가 저장된다. 이때 STATUS의 발생 상황을 명확히 알려면 다음과 같이 계산해 보면 된다. PIC 계열의 마이크로프로세서에서 빼기 연산은 2의 보수를 이용한다. 따라서 07H-05H 연산을 이진수로 2의 보수를 이용하여 직접 계산을 해보면 07H+(100H-05H) = 07H+FBH = 02H가 되고 따라서 DC와 C가 발생하는 것을 알 수 있습니다. 즉 STATUS reg.의 값이 '?????011'로 설정된다.

또 'W=08' 일 때 ' SUBLW 07' 를 수행시키면 07 - 08 로 결과는 W=FFH, C=0, DC=0, Z=0으로 설정된다. 여러분이 설명을 읽어도 이해가 잘

안되는 부분이 있을 것으로 보이며, 실제 여러 번 여러 경우를 접해 보아야 완전히 이해가 가능할 것이다. 특히 subtraction 명령어를 사용 시 C는 항상 혼돈이 잘되므로 주의를 요한다.

☞ 참고: carry는 INTEL 계열의 기존 마이크로프로에서 사용하는 의미와 다르므로 혼돈이 생길 수 있다. INTEL 계열에서는 carry와 borrow를 구분하지 않고, 발생할 경우에 1로 set하고 있다. 그러나 PIC 계열에서는 뺄셈을 2의 보수로 연산하여 덧셈으로 변경한 후 더하여 나타나는 결과에 따라서 STATUS가 설정되도록 되어 있다.

☞ 참고: STATUS reg.에 영향을 주지 않는 명령어가 많이 있으며, 이 경우는 과거의 값이 그대로 유지된다.

5) 'Z' STATUS에 영향을 주는 명령어

Zero flag에 영향을 주는 명령어는 아래와 같은 명령어로 가장 많으며, 명령의 수행 상태가 0이 만들어지는 경우에 설정된다.

ADD, AND, CLR, COMF, DECF, INCF, IOR, MOVF, SUB, XOR

6) 조건에 따라서 분지(branch) 하는 명령어

조건에 따라서 분지(branch) 하는 명령어는 앞에서 설명한 것과 같이 조건에 따른 상태를 확인하여 분지가 일어나게 하는 명령어와 연산 결과에 따라서 바로 분지가 일어나도록 하는 명령어 2 종류가 있다.

- 상태를 확인하여 분지 하도록 하는 명령어: BTFSC f, b 와 BTFSS f, b
- 연산 결과로 바로 분지 하는 명령어 : DECFSZ f, d 와 INCFSZ f, d

① 상태를 확인하여 분지 하도록 하는 명령어는 register의 특정 bit가 0 이냐, 1 이냐에 따라서 분지가 결정되는 것으로 BTFSC f, b 와 BTFSS f, b 이 있다. 여기서 f 는 register를, b는 bit의 위치를(7에서 0까지임) 그리고 BTFSC(bit test file skip clear)는 f-file register의 bit b의 값이 0 이면 skip하라는 의미이고, BTFSS는 bit test 결과가 1 이면 skip하라는 것이다. 여기서 skip의 의미는 이 명령어 바로 다음 명령어를 수행하지 않고 그 다음 명령어를 수행하라는 것이다.

예8) REGISTER FILE 22H 번지의 내용이 00H 이면 XY 번지의 프로그램을 수행하고 아니면 WZ 번지의 프로그램을 수행하도록 하는 프로그램은 다음과 같다.

```
MOVLW     00H
SUBWF     22H,W          ; SUBWF    22H,0
BTFSS     STATUS,ZF      ; BTFSS    03H,2
GOTO      WZ
GOTO      XY
```

위 프로그램은 0인가를 확인하기 위하여 뺄셈 연산 명령어를 확인하였으며, 덧셈 명령어도 동일한 효과가 있다. 또 다른 명령어로 MOVF f,F 명령어를 사용해도 동일한 결과가 나온다. 어떠한 명령어를 사용하는 것이 편리하겠는가? 이것이 어셈블리 명령어의 묘미이다.

```
MOVF      22H, F         ; = MOVF   .34, 1
BTFSS     STATUS, ZF
GOTO      WZ
GOTO      XY
```

② 연산 결과로 바로 분지 하는 명령어는 **DECFSZ f, d 와 INCFSZ f, d** 이 있다. 이 명령어는 F-file register를 1씩 감소하거나 증가시키면서 그 값이 0이 되면 바로 아래 명령어를 SKIP 하는 것이다. 특히 **이 명령어는 STATUS에 영향을 전혀 주지 않는**다. 따라서 DECFSZ, INCFSZ 명령어는 STATUS flag에 영향을 주지 않고 프로그램을 분지할 수 있기 때문에 프로그램을 단순하게 구현할 수 있어, PIC 계열이 다른 종류의 프로세서에 비해서 좋은 기능을 가지게 된다.

☞ 참고: DECF f, d 및 INCF f, d 명령어는 ZERO flag에 영향을 준다.

3. 시간지연을 만드는 부 프로그램

앞 실험에서 RUN이라는 기능으로 프로그램을 수행시키면 LED 4개 전부가

불이 들어옴을 확인하였을 것이다. 이는 작성된 프로그램이 너무 빠른 속도로 LED에 1 과 0를 출력(on/off)하므로 우리가 눈으로 판단하기에는 잔상에 의해서 항상 불이 들어온 것으로 보이게 된다. 따라서 사람이 LED의 불이 on/off 됨을 인식하도록 하려면 프로그램에서 한번 on시킨 후 특정 시간 동안 다른 일을(특별한 일이 없을 경우에는 놀게 만듬) 하도록 하고 나서, LED을 off시키고 또 다른 일을 하고 on을 반복하면 된다. 이러한 용도로 사용되는 가장 간단한 기능(놀게 만듬: 지연시킴)을 DELAY PROGRAM 이라 한다. DELAY PROGRAM은 의미 없는 일을 반복시켜서 시간 지연을 얻는 것이다.

예9) 어떤 프로그램을 100번 반복할 경우의 프로그램은

```
        MOVLW   .100
        MOVWF   COUNT       ; 100번을 확인하기 위한 변수
LP      .                   ; 반복되는 PROGRAM 영역
        .
        DECFSZ  COUNT,F     ; 변수를 감소시켜 가면서 00
                            ; 이 되었나 확인
        GOTO    LP          ; zero가 아니면 여기에 들어옴.
        ???.                ; zero이면 NEXT PROGRAM 시작
```

☞ 당연히 COUNT는 사용자가 임의로 만든 변수이므로 FILE REGISTER 중 USER 영역 안의 번지로 선언해주어야지요. 또 BANK도 맞게 해주어야 합니다.

반복시키는 프로그램 양이 적으면 시간 지연이 적게 되므로 한 반복 LOOP를 다시 반복시키면 많은 시간 지연을 만들 수 있다.

예10) DELAY 부 프로그램 : 약 100 *ms* 지연

```
DELAY
        MOVLW   .125
        MOVWF   DBUF1       ; 긴 시간을 설정하기 위한 변수
LP1     MOVLW   .200
```

```
        MOVWF     DBUF2
LP2     NOP
        DECFSZ    DBUF2,F
        GOTO      LP2
        DECFSZ    DBUF1,F   ;변수를 감소시켜 가면서 00이 되었나 확인
        GOTO      LP1       ; ZERO가 아니면 여기에 들어옴.
        RETURN              ; 0이면 바로 이곳으로->부 프로그램 종료
```

현재 실험에서 사용하는 CLOCK는 4㎒(20㎒)이므로 1㎒(5㎒)로 한 명령어가 수행되므로(data sheet instruction 부분 참고) 반복되는 횟수×1 μs(0.2 μs)가 총 지연시간이 된다. 구체적으로 위 프로그램의 지연시간은 대략 LP2의 지연시간 * LP1의 loop 횟수로 아래와 같이 계산할 수 있다.

125*(200*(1+1+2))*1 μs(0.2 μs) = 100,000 μs(20,000 μs)

☞ 참고1: 이 프로그램은 부프로그램으로 작성되었으므로 필요한 위치에서 불러서 사용 하면 되고, 프로그램으로는 'CALL DELAY' 라고 해야지요.

☞ 참고2: loop 횟수를 이용한 시간 계산은 정밀하지 못하다. 따라서 정밀한 시간이 요구될 경우는 timer와 interrupt라는 기능을 사용하며, 다음에 배운다.

■ 예비문제 3

1) 앞에서 설명하지 않은 방법으로 PIN 11를 통하여 1을 출력하는 명령어를 만들어 보시오

2) 1)의 조건을 입력으로 변경하여 명령어를 만들어 보시오.

3) I/O 초기 설정까지 포함하여 PORTA, PORTB 전부를 OUTPUT으로 사용하기 위한 프로그램을 작성하시오.

4) 예8)를 다른 명령어를 사용하여 2개 이상 프로그램 하시오.

5) PIN 25(RB4)이 1이면 XY 번지로, 0이면 WZ 번지로 분지하는 프로그램을 작성하시오.

6) 명령어 표를 보면 어떤 명령어는 STATUS에 영향을 주지 않는 것으로 되어 있다. 이 의미가 프로그램 작성시 구체적으로 어떻다는 것인가를 설명해 보시오.

■ 실 험 3

LED 12개를 사용하여 ON/OFF 하여봅시다..

1) 다음 프로그램을 작성합시다.

```
; 앞의 HEADER에 해당하는 부분

; I/O를 설정하는 부분
; --> PORTA, PORTC를 전부 OUTPUT으로 선언

;  MAIN PROGRAM
        CLRF        PORTA
        MOVLW       01
        ADDLW       00              ; CF를 0으로 만듬
        MOVWF       LED1            ; 초기값 01를 넣음
LOOP
        MOVF        LED1,W
        XORLW       B'11111111'     ; NOT 시킴
        MOVWF       PORCT           ; W 을 PORTC에 출력
        RLF         LED1,F          ; rotate 시킴
        CALL        DELAY
        GOTO        LOOP

;  SUBROUTINE
DELAY
        MOVLW       .125
        MOVWF       DBUF1           ; 125번을 확인하기 위한 변수
```

```
LP1   MOVLW    .200
      MOVWF    DBUF2        ; 200번을 확인하기 위한 변수
LP2   NOP
      DECFSZ   DBUF2,F
      GOTO     LP2
      DECFSZ   DBUF1,F      ; 변수를 감소시켜 00이 되었나 확인
      GOTO     LP1          ; ZERO가 아니면 여기에 들어옴.
      RETURN

      END      ; 프로그램 끝을 알리는 가상 명령어
```

2) ERROR를 수정합시다.

☞ Hint: 프로그램에서 선언되지 않는 변수가 있으면 안되지요.

3) RC7~RC0를 LED L8~L1에 연결합시다. 그리고 수행시켜보고 무슨 일을 하는가 이해합시다.(LED 8개가 순환되면서 ON됨, 한번은 전부 OFF됨)

4) RA3,2,1,0를 LED L12,11,10,9에 연결하고, 위 프로그램을 기초로 LED 12개가 순환되도록 프로그램을 변경하시오.

☞ Hint: LED2 변수를 추가, 'RLF LED2,F' 추가
LED2의 BIT4가 1이면 MAIN PROGRAM 초기로 이동 등..

♥ 몇 주만에 수강 신청 안한 사람과 차이가 많이 남을 느낄 수 있지요!!!
기운 내도록 합시다. 너무 속도가 빠릅니까?
구체적인 의견 주세요. (의견은 인터넷으로 → **http://cac.knu.ac.kr**)

■ HOME WORK 3

1) 실험에서는 LED가 1개씩 ON되어 순환되었다. 이것을 2개씩 ON되어 순환되도록 작성하시오.

2) 0.5SEC DELAY PROGRAM을 작성하시오.

3) 명령어 전부를 쓰고 의미를 적어 봅시다.