부록 1. 명령어 보는 법

1. 처음 보는 사람

- Mnemonic 과 operands를 보고 의미(명령어 설명부분)를 익힌다.
- Operand의 f,d,k 기호를 이해한다.

  f: data RAM을 의미한다. (RAM 주소를 가리킨다.)

  d: 결과가 저장될 위치를 지정한다.

  d가 '0'이면 W레지스터에 저장하고, "1"이면 f레지스터에 저장한다.)

  ☞ d가 생략되면, assembler가 d=1로 이해한다.

  k: literal(글자대로의) 값으로 상수를 의미한다.

  k 값이 데이터로 사용되는 경우는 8bit 범위(0~255 사이 값)의 값만 가질 수 있다.

  k 값을 프로그램의 label로 주면, 그 프로그램이 들어있는 주소가 된다. 11bit 범위(0000H~07FFH 사이 값)의 값만 가질 수 있다.

  ☞ 명령어 표에서 (XX)에서 ()는 XX의 내용이란 의미이다.

2. 중간 기능을 보는 사람

- Mnemonic 과 operands에 따른 status의 변화를 이해한다.

  (Status C,DC,Z 및 PD,TO는 file register 중 03번지에 해당하는 특수용도 data RAM(고유이름으로 STATUS라고 한다.)의 bit0,1,2,3,4를 나타낸다.)

- Cycle time을 보고 동작 시간을 이해한다.

  (DECFSZ,INCFSZ,BTFSC,BTFSS 명령어)

  ☞ 명령어 중에서 프로그램의 흐름을 변경하는(PC 값이 변경되는 경우) 명령어는 cycle time이 2가 되며, 나머지는 모두 1 cycle 만에 수행된다. 그리고 1cycle time은 마이크로프로세서의 clock 주파수의 1/4에 해당하는 시간이 된다.

3. 고급 기능을 보는 사람

- Mnemonic에 따른 기계어 코드(opcode)를 이해하고, f와 k의 사용 가능한 최대범위를 이해한다. (특히 GOTO k, CALL k 명령어)
- 특수한 목적의 하드웨어를 동작시키는 명령어를 이해한다.

(CLRWDT,SLEEP)

표 A2-1 PIC16CXXX 명령어 요약표

니모닉		플래그			사이클	14비트 OP코드		설명	주의
명령	오퍼랜드	C	DC	Z		msb	lsb		
MOVF	f,d	·	·	√	1	00 1000 dfff ffff		Move f	1,2
MOVWF	f	·	·	·	1	00 0000 1fff ffff		Move W to f	
MOVLW	k	·	·	·	1	11 00xx kkkk kkkk		Move literal to W	
SWAPF	f,d	·	·	·	1	00 1110 dfff ffff		Swap nibbles in f	1,2
CLRW		·	·	√	1	00 0001 0xxx xxxx		Clear w	
CLRF	f	·	·	√	1	00 0001 1fff ffff		Clear f	2
ADDWF	f,d	√	√	√	1	00 0111 dfff ffff		Add W and f	1,2
ADDLW	k	√	√	√	1	11 111x kkkk kkkk		Add literal to W	
SUBWF	f,d	√	√	√	1	00 0010 dfff ffff		Subtract W from f	1,2
SUBLW	k	√	√	√	1	11 110x kkkk kkkk		Subtract W from literal	
COMF	f,d	·	·	√	1	00 1001 dfff ffff		Complement	1,2
ANDWF	f,d	·	·	√	1	00 0101 dfff ffff		AND W and f	1,2
ANDLW	k	·	·	√	1	11 1001 kkkk kkkk		AND literal to W	
IORWF	f,d	·	·	√	1	00 0100 dfff ffff		Inclusive OR W and f	1,2
IORLW	k	·	·	√	1	11 1000 kkkk kkkk		Inclusive OR literal to W	
XORWF	f,d	·	·	√	1	00 0110 dfff ffff		Exclusive OR W and f	1,2
XORLW	k	·	·	√	1	11 1010 kkkk kkkk		Exclusive OR literal to W	
RLF	f,d	√	·	·	1	00 1101 dfff ffff		Rotate left throught carry	1,2
RRF	f,d	√	·	·	1	00 1100 dfff ffff		Rotate right throught carry	1,2
BCF	f,b	·	·	·	1	01 00bb bfff ffff		Bit clear f	1,2
BSF	f,b	·	·	·	1	01 01bb bfff ffff		Bit set f	1,2
BTFSC	f,b	·	·	·	1(2)	01 10bb bfff ffff		Bit test f, Skip if clear	3
BTFSS	f,b	·	·	·	1(2)	01 11bb bfff ffff		Bit test f, Skip if set	3
DECF	f,d	·	·	√	1	00 0011 dfff ffff		Decrement f	1,2
INCF	f,d	·	·	√	1	00 1010 dfff ffff		Iecrement f	1,2
DECFSZ	f,d	·	·	·	1(2)	00 1011 dfff ffff		Decrement f, skip if 0	1,2,3
INCFSZ	f,d	·	·	·	1(2)	01 1111 dfff ffff		Increment f, skip if 0	1,2,3
GOTO	k	·	·	·	2	10 1kkk kkkk kkkk		Go to address	
CALL	k	·	·	·	2	10 0kkk kkkk kkkk		Call subroutine	
RETURN		·	·	·	2	00 0000 0000 1000		Return from subroutine	
RETLW	k	·	·	·	2	11 01xx kkkk kkkk		Return with literal in W	
RETFIE		·	·	·	2	11 0000 0000 1001		Return from interrupt	
NOP		·	·	·	1	00 0000 0xx0 0000		No operation	
CLRWDT		·	·	·	1	00 0000 0110 0100		Clear watchdog timer, $\overline{TO}$=1, $\overline{PD}$=1	
SLEEP		·	·	·	1	00 0000 0110 0011		Go into standby mode, $\overline{TO}$=1, $\overline{PD}$=0	

☞ 오퍼랜드에 사용된 기호의 의미

f(file) : 레지스터 파일 어드레스 (최대 0~127), W : working register

b(bit) : 레지스터의 비트번호(0~7), k : 상수 혹은 라벨(주소가 됨)

d(destination) : 목적지 선택 코드(d=0: W, d=1: f 에 저장)

∨ : flag(status)의 표시는 이 명령어에 의해서 영향 받음을 나타낸다.

주의 ; 1 : 오퍼랜드로 I/O 레지스터를 사용할 경우, 현재 핀에 나타난 값이 사용된다.

2 : TMR0 레지스터를 사용하는 명령이면 프리스케일러가 클리어 된다.

3 : PC가 변경되거나, 조건이 참이면 2 사이틀이 필요하게 된다.

4. PIC 16F84의 명령어 사용 요약

1) 데이터의 이동을 관리하는 MOVE 명령어

MOVE 명령어는 **MOVF, MOVWF, MOVLW** 세가지 명령어가 준비되어 있다. 이 중 **MOVF와 MOVWF 은 W와 f 사이의 MOVE 이고, MOVLW 는 W에 k**라는 상수를 넣는 것이다. 만약 F에 k라는 상수를 바로 넣는 명령어를 만든다면, f와 k가 모두 변수가 되어 14bit 명령어로 만들 수 없다. (f: 7BIT, k: 8bit 가 되어 명령어의 최소 bit가 16bit 이상 되어야 한다.)

2) 결과를 저장하는 목적지를 결정하는 명령어

PIC 계열은 많은 명령어(file register를 조작하는 명령어)에서 결과를 저장하는 목적지를 사용자가 결정할 수 있도록 하고 있다. 명령어에서 보면, d 값이며, '0'이나 '1'의 값을 가진다. **d가 '0'이면** 결과를 **W레지스터에 저장하고, "1"이면 f레지스터에 저장한다.**

3) Bit를 지정하여 사용하는 명령어

PIC 계열에서는 임의 file register에 대하여 bit를 지정하여 '1' 또는 '0'으로 만들거나, '1' 또는 '0' 인가를 확인하는 명령어가 있다. 이 명령어들은 제어용 신호를 만들거나, 입력 신호에 따라서 여러 가지 일을 시키고자할 때에 아주 유용하게 사용된다.

4) 부 프로그램을 관리하는 명령어

부 프로그램을 호출하는 명령어는 CALL 한 개이나, 부프로그램에서 원래의 프로그램으로 되돌아오는 명령어는 방법은 **RETURN, RETLW, RETFIE 3 가지가 있다.**

첫 번째 RETURN 명령어는 단순 리턴 명령으로 가장 많이 사용하는 리턴 명령이다. 두 번째 RETLW 명령어는 리턴 동작과 동시에 W레지스터에 상

수를 넣을 수 있는 기능을 동시에 구현한 것이다. 이는 부 프로그램에서 주 프로그램으로 되돌아올 때, 되돌아오는 조건을 직접 가지고 올 수 있어 아주 유용성 있는 명령어 있다.(테이블을 만들어 조건에 따른 테이플 값을 가져오는데 아주 편리한 명령어이다.) 그리고 세 번째 RETFIE 명령은 일반 부 프로그램이 아닌 인터럽트 처리를 위한 인터럽트 서브루틴 (인터럽트를 처리하는 프로그램도 일반적으로 부 프로그램으로 작성된다.)에서 인터럽트의 처리를 끝내고자 할 때 사용하는 리턴 명령으로 RETRIE 명령 하나로 리턴 동작과 동시에 다시 새로운 인터럽트를 받아드리기 위해서 GIE (글로벌 인터럽트 Enable) 비트를 '1'로 설정하는 동작을 동시에 구현한 것이다.

RETLW 명령어는 다음의 두 개의 명령어로 구현할 수 있다.

```
MOVLW    K
RETURN
```

RETFIE 명령어는 다음의 두 개의 명령어로 구현할 수 있다.

```
BSF      INTCON, GIE
RETURN
```

(INTCON 레지스터에 있는 GIE 플래그는 인터럽트 전체 기능을 ON/OFF 하는 플래그이다.)

☞ RETFIE 명령어를 BSF INTCON, GIE와 RETURN으로 구현 가능하다고 하였으나, 엄밀히 따져보면, RETURN 되기 전에 BSF INTCON, GIE 명령어로 인터럽트를 다시 걸릴 수 있게 하므로 잘못하면, RETURN 없이 인터럽트 서비스루틴에 반복적으로 들어오는 사태가 발생하여 STACK over가 발생하여 프로그램이 정상적으로 동작되지 않을 수 있다. 따라서 일반적인 마이크로프로세서에서는 return interrupt라는 별도의 명령어를 가지고 있다.

☞ CALL 명령어를 사용하면, 되돌아올 주소를 저장하여야 한다. 이때 주소가 명령어에 의해서 자동으로 특정 기억 장소에 저장되며, 이 장소를 stack라고 하며, 저장되는 주소를 stack pointer(SP)라고 한다. Stack는 SP 한 개로 LIFO(last in first out) 메모리로 동작한다. 그러면 TOS(top of stack)라는 말이 이해되지요.

♣ **사용상의 주의점**은 stack 메모리의 개수가 몇 개인가를 확인하여야 한

다. 왜냐하면, 이 개수보다 더 많은 횟수를 CALL 하면, stack over가 발생하여 프로그램이 정상 동작할 수가 없다. 그러나 CALL과 RETURN이 쌍으로 사용되면, 사용되는 stack 메모리가 늘어나지 않으므로 여러번 사용해도 상관없다. PIC 계열에 따라서 이 stack 메모리의 개수가 많이 차이나며, **PIC16F84는 8개(8 level)이다.**