실험 8.인터럽트 이해하기와 2자리 초시계 만들기

1. 인터럽트 이해하기

마이크로프로세서가 동일시간에 여러 가지 일을 할 수가 없으므로, 조금 복잡한 기능을 구현하기에는 어려움이 많다는 것을 체험하였을 것이다. 대표적으로 7-segment에 글자를 나타내면서 key를 읽어 오기 위해서는 2가지 일을 번갈아 가면서 수행되도록 프로그램을 작성해야 하는데 이것은 아주 귀찮은 일이 된다. 따라서 마이크로프로세서가 현재 수행되고 있는 프로그램에는 영향을 주지 않고 별개의 프로그램을 수행할 수 있는 기능을 가진다면, 앞의 예처럼 여러 가지 일을 번갈아 해야할 때 이 기능은 아주 유용하게 사용될 수 있다.

프로그램은 PC(Program Counter)의 내용에 따라서 순차적으로 수행되므로, 따라서 현재 흐름에서 벗어나 다른 프로그램을 수행한다는 것은 PC 값을 적절히 변경하는 일이 된다. 그러나 별개의 프로그램이 수행되어도 지금 수행되고 있었던 프로그램의 흐름에는 영향을 주지 않으려면 최소한 두 가지 조건을 만족해야한다.

☞ PC 값이 변경되어 다른 프로그램으로 이동하였다가 원래의 프로그램으로 정상적으로 복귀해야한다. 즉 원래 프로그램의 흐름상에서는 마치 PC가 변경되지 않고 흘러간 것처럼 보여야한다. 이러한 기능을 하는 명령어가 여러분이 알고 있는 CALL 명령어이다. CALL 명령어는 원 프로그램에서 벗어나 다른 프로그램을 수행하다가 RETURN 명령어를 만나면 원 프로그램으로 되돌아가 마치 PC 입장에서는 원 프로그램만 계속 수행한 것처럼 보인다.

☞ 또 원래의 프로그램에 영향을 주지 않기 위해서는 모든 register를 별개로 사용해야한다. 지금까지 여러분이 사용한 CALL 명령어는 프로그램은 별개로 작성하고 register를 공유하여 원 프로그램과 부 프로그램을 연결시켜 사용해 왔다. 그러나 지금 설명하고 있는 내용은, 두 프로그램이 완전히 별개의 프로그램이 되기 위해서는 두 프로그램에서 사용되고 있는

모든 register를 별도로 사용해야한다.

그러면 과연 모든 register를 별도로 사용 가능한 것인가요? 사용자가 정의하여 사용하는 변수 register(general purpose register)는 서로 다른 저장 번지를 설정하면 되겠지요. 그리고 만약 두 프로그램이 변수를 공유하려고 하면 같은 주소를 사용하면 되지요. 그러나 프로세서가 사용하는 변수 register(system register, special purpose register)는 사용자가 어떻게 할 수가 없을 뿐만 아니라, 계속 변하는 값이 되지요. 대표적으로 PC, W, STATUS 등이 있지요. 따라서 두 프로그램이 서로 영향을 받지 않고 별개로 동작되도록 하려면 최소한 PC, W, STATUS register는 별개로 만들어야하지요. 그러나 별개로 만들면 하드웨어도 복잡해지고 또 어려움이 있으므로, CALL 명령어와 같이 PC를 별도의 기억장소(stack)에 보관시키고 나중에 다시 불러내 사용한 것처럼(RETURN 명령어), W와 STATUS를 다른 프로그램으로 이동시 파괴가 되지 않은 저장 영역에 저장하였다가 프로그램 끝에서 저장된 내용을 복귀시키면 되지 않을까요?

☞ 우리가 사용하는 PIC에서는 1-chip으로 general purpose register도 별도의 주소 없이 바로 접근 가능하므로 general purpose register에 저장하나, 1-chip 프로세서가 아닌 경우는 general purpose register에 직접 접근하기가 어려워 CALL 명령어와 같이 stack에 저장하는 것이 원칙이다.

그리고 두 프로그램 사이를 옮겨가는 구체적인 방법으로는 사용자가 원할 때에 는 현재 수행되고 있는 프로그램으로부터 언제라도 벗어날 수가 있어야 하므로 CALL 명령어와는 다른 의미를 가지게 된다. 즉 CALL은 프로그램 내에서 호출해야 부 프로그램으로 이동되지만 위에서 설명한 기능은 CALL 명령어가 없어도 다른 프로그램으로 이동이 되어야한다.

일반적이 모든 프로세서는 앞에서 설명한 기능이 가능하도록 하드웨어적으로 구현해 놓았으며, 이를 interrupt라고 부른다. interrupt는 마이크로프로세서를 고급스럽게 활용하는 핵심적인 기능이므로 잘 이해해 두길 바란다. 사용자 입장에서 interrupt를 가장 쉽게 이해하는 방법은 '**외부 신호에 의해서 현재 수행 중인 프로그램 사이에 하드웨어적으로 CALL ISR 명령어가 삽입된다**' 라고 받아드리면 된다. 이때 이동해 가는 주소 ISR(interrupt service routine)은 주소가 고정될 수도 있고, 가변될 수도 있다.

☞ PIC 에서는 ISR 주소가 고정되어 있으며, program memory에서 interrupt vector 주소로 표현된 0004H 번지가 시작 번지이다. 주소가 여러 개이거나 가변되는 경우는 보다 많은 서로 다른 종류의 인터럽트를 처리할 수 있게 되므로 고급 기능이라고 생각하면 된다.

☞ 인터럽트 ISR 주소가 1개만 있는 경우에 여러 종류의 인터럽트를 받아들일 수가 있는가? 답은 '있다'다. 회로적으로는 여러 개의 인터럽트를 OR 로직으로 합하여 어느 하나라도 외부 신호가 들어오면, 인터럽트가 발생하도록 하면 된다. 그리고 ISR 프로그램에서 여러 입력 중 어느 것이 인터럽트를 발생했는가를 확인하여 그에 해당되는 프로그램으로 분지하면 된다. 이러한 기능을 가능하게 하기 위해서는 개별 인터럽트 발생원은 인터럽트 발생 유무를 기억시키는 기억소자(interrupt flag)를 가지고 있어야 하며, 또 여러 개의 인터럽트 발생원 중에서 사용 하고자 하는 것만을 선택하는 기능이 추가되어야 한다. 후자의 기능은 하드웨어적으로는 AND 로직으로 만들 수 있으며, 인터럽트 활성/비활성의 선택은 interrupt mask 기억소자를 사용하면 된다.

그리고 interrupt를 발생시키는 신호원으로는 여러 가지가 있다. PIC16F84에는 4가지가 있으며, 외부신호를 받아서 발생시키는 것과 내부회로로 발생시키는 것으로 나누어진다. 자세한 것은 data sheet 6장을 참고하기 바란다.

2. 인터럽트를 동작시키기

Interrupt를 동작시키기 위해서는 몇 가지 초기 조건을 만들어 주어야 한다.

1) 특정 interrupt를 살리기

CPU가 reset되면 초기 조건으로 모든 interrupt가 disable된다. 따라서 interrupt를 가능하게 하려면 interrupt를 enable 시켜야한다. Register 중에 interrupt를 제어하는 것이 있으며, INTCON(0BH)이다. INTCON은 8bit로 4종류의 interrupt를 선택 동작시키는 기능과 interrupt 전체를 동작시키는 기능으로 나누어진다.

인터럽트를 선택하는 기능은 GIE, EEIE, T0IE, INTE 및 RBIE로 끝이 E로 끝나는 bit이다. 여기서 E는 enable을 의미한다. 즉 기능을 가능하게 하겠다는 의미이다. 그리고 인터럽트가 발생되며 그것을 기억하는 기억장치가 있으며, T0IF, INTF 및 RBIF이다. 따라서 enable 된 상태에서 인터럽트 flag가 set 되어 있으며, 인터럽트가 발생한다.

표 E8-1. INTCON 기능도

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE**: Global Interrupt Enable bit
1=Enables all unmasked interrupts
0=Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1=Enables the EE Write Complete interrupts
0=Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1=Enables the TMR0 interrupt
0=Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1=Enables the RB0/INT external interrupt
0=Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1=Enables the RB port change interrupt
0=Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1=TMR0 register has overflowed (**must be cleared in software**)
0=TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1=The RB0/INT external interrupt occurred
(**must be cleared in software**)
0=The RB0/INT external interrupt did not occurred

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1=At least one of the RB7:RB4 pins changed state
(**must be cleared in software**)
0=None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

만약 timer 0의 overflow(timer 0라는 내부 계수회로가 있고 이것이 자리 올림이나 넘침이 발생한 경우)가 발생하면 interrupt를 발생시키려고 한다면 주프로그램에서 다음 명령어를 사용해야한다.

예1)

```
BSF    INTCON,5          ; TIMER INTERRUPT ENABLE
BSF    INTCON,7          ; GLOBAL INT. ENABLE

; 또는
T0IE   EQU   5
GIE    EQU   7
BSF    INTCON,T0IE       ; TIMER INTERRUPT ENABLE
BSF    INTCON,GIE        ; GLOBAL INT. ENABLE
```

2) ISR 위치에 프로그램을 작성하기

ISR 시작 번지가 0004H번지로 되어 있으므로 여기서부터 interrupt가 발생되었을 때 수행할 프로그램을 작성한다. interrupt가 발생되면 이때 PC는 자동적으로 stack에 저장되나 W와 STATUS register는 저장되지 않으므로 별도로 저장시켜 주어야 한다.

예2)

```
        ; ISR 시작 번지
        MOVWF     W_TEMP ; 현재 사용되고 있는 W reg.를 저장
        SWAPF     STATUS, W
        MOVWF     STATUS_TEMP
        ; 여기서부터 실질적인 ISR이 된다.
              .
              .
        ; 여기가 실질적인 ISR 끝 부분
        SWAPF     STATUS_TEMP, W   ; 저장된 내용으로 복원
        MOVWF     STATUS
        SWAPF     W_TEMP, F
```

```
SWAPF       W_TEMP, W
BCF         INTCON, 2
RETFIE
```

첫줄은 주프로그램에서 사용되던 W register가 변경되지 않도록 W_TEMP라는 변수에 저장하는 것이며, 2, 3줄은 STATUS를 보관하기 위한 것이다. 특정 번지의 내용을 다른 번지로 이동하기 위해서는 일단 register의 내용을 W로 이동해야한다. 이런 용도로 사용 가능한 명령어는 여러분이 많이 사용한 MOVF reg, W가 있다. 그러나 이 명령어는 STATUS의 Z flag에 영향을 주므로 여기에서는 사용할 수가 없다. 왜냐하면 STATUS가 변경될 수 있기 때문이다. 따라서 다른 명령어를 찾아보아야 한다. SWAPF reg, W 명령어가 있다. 이 명령어는 어떠한 flag에도 영향을 주지 않는다. 꼭 data sheet를 확인해 보기 바란다. 따라서 위 프로그램에서는 SWAPF STATUS, W 와 MOVWF STATUS_TEMP를 사용하여 저장하였다.

그리고 실질적인 ISR이 끝나면 저장된 W와 STATUS register를 다시 환원시켜 주어야 한다. 그런데 W_TEMP를 W로 다시 환원하기 위하여 SWAPF W_TEMP, F와 SWAPF W_TEMP, W를 사용한 것은 MOVF reg, W를 직접 사용할 수가 없기 때문에 두 번 돌려서 사용한 것이다.

그리고 마지막 두 명령어 중 BCF INTCON, 2는 timer 0의 overflow bit를 clear 시켜주는 명령어이다. (어려운 이야기이지만 이것은 timer 0에서 overflow가 발생하면 overflow가 발생되었음을 기억시키게 된다. 즉 timer 0에 overflow flag가 있다는 것이며, 기억되어야 할 이유는 timer 자체에서는 over가 되는 순간에만 아주 짧은 시간만 신호가 나오면, 이러한 짧은 신호는 interrupt를 발생시키지 못할 경우가 나타나게 되기 때문이다. 특히 ISR내에서 overflow가 발생되면 interrupt를 받아드리지 못하므로 overflow를 놓치게 된다. 이러한 사태를 막기 위하여 기억시키고 있으며, 따라서 인식되어 사용되면 다시 clear 해야 한다. Clear 하지 않으면 계속 interrupt를 발생시킨다.) 그리고 ISR내에서는 이중으로 interrupt를 받아드리지 않도록 하드웨어가 interrupt 전체를 disable 시키므로 ISR이 끝나서 다시 새로운 interrupt를 받아들이기 위해서는 interrupt를 enable 시켜야한다. 이것을 별도의 명령어로 만들면 흐름이 비정상적이 될 수 있으므로 RETURN과 동시에 interrupt를 enable 시켜야 한다. 이 명령어가 마지막 줄의 RETFIE이다.

☞ 조금 구체적인 설명이나 사용법을 이해하려면 기능의 의미와 하드웨어를 알아야 됩니다. Data sheet를 자주 확인하고 이해하기를 바랍니다.

3. 인터럽트를 주기적으로 발생시키기 위한 timer 동작시키기

Interrupt는 주프로그램이 어떤 일을 하면서 다른 일을 시킬 수 있는 기능이다. 우리가 시계를 만든다고 하면, 시간을 표시하는 기능과 시간을 만들어 내는 기능 그리고 여러 가지 부가적인 기능이 필요하다. 시간을 표시하는 기능은 7-segment를 사용하는 경우 scanning에 의해서 주기적으로 표시해야하며, 시간을 만드는 것도 아주 정확한 주기를 요구하게된다.(앞 실험에서는 delay program으로 적당히(?) 구현한 것임) 따라서 이 두 기능은 주기적으로 수행해야 하므로 interrupt 내에서 구현하는 것이 편리하다. 그 대신 interrupt를 주기적으로 발생시키기 위한 하드웨어와 이를 제어하기 위한 프로그램이 추가로 필요하다. 실제의 여러 응용에서 주기적인 처리가 많이 필요하며, 따라서 일반적인 마이크로프로세서는 내부에 timer를 가지고 있다. PIC16f84에는 timer가 한 개 있으나, 다른 칩에는 여러 개 있는 것도 있다.

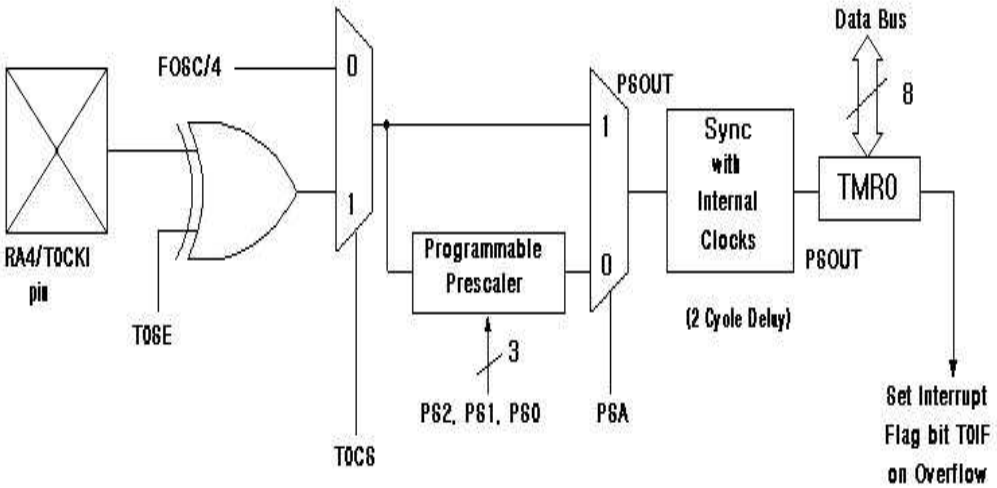

그림 E8-1. Timer0의 내부 구성도와 제어 신호

다음은 timer의 사용법에 대한 설명이다. PIC16F84에 내장된 timer는 다음과 같은 규격을 갖는다.

- 8bit 길이이다. read/write가 가능하다.
- 프로그램 가능한 8bit prescaler가 내장되어 있다.
- 입력 신호로는 내부 clock과 외부 pin 신호(RA4 pin)가 있고 선택 가능하다.
- 외부 신호를 사용 시는 동작되는 신호의 성질을 선택할 수 있다. (rising/falling edge에서 동작됨)
- Timer의 내용이 0FFH에서 00H로 변하면 overflow를 set시키고 interrupt를 발생하게된다.

표 E8-2. OPTIONR(주소 : 81H) 기능

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 $\overline{\text{RBPU}}$: PORTB Pull-up Enable bit
1 =PORTB pull-ups are disabled
0 =PORTB pull-ups enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 =Interrupt on rising edge of RB0/INT pin
0 =Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 =Transition on RA4/T0CKI pin
0 =Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 =Increment on high-to-low transition on RA4/T0CKI pin
0 =Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 =Prescaler is assigned to the WDT
0 =Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Legend:
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
- n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

Bit 2,1,0의 설정에 따른 Prescaler의 설정값

Bit Value	TMR0 Rate	WDT Rate
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Timer 전체의 구체적인 내용은 여러분이 data sheet를 보면서 이해하도록 하고, 우리가 사용하려는 경우만 설명한다. 우리는 주기적인 interrupt를 발생시키려고 하므로 신호원은 내부 clock을 사용할 것이다. 그런데 내부 clock 신호는 주파수가 높아서 8bit 계수기로는 너무 빨리 overflow가 발생되게 된다. 따라서 prescaler를 사용하여 발생 주기를 원하는 단위가 되도록 만들어야한다. Prescaler를 동작시키는 기능은 OPTIONR이라는 register에서 하게 된다.

이 기능을 이용하여 약 2msec의 주기를 갖는 interrupt를 만들어 보자. 먼저 timer 0의 내부 구성도를 보면 T0CS=0일 때 내부clock이 선택된다. 이 경우 주파수는 4MHz/4가 되어 1MHz가 된다. 이것을 바로 8bit timer에 넣으면 256/1MHz가 되어 0.25msec가 된다. 따라서 주기를 2msec가 되도록 하려면 prescaler를 8로 설정해야한다. 즉 8분주가 된다는 것이다. 즉 PS2, PS1, PS0=010을 선택하면 된다. 따라서 OPTIONR에 기억되는 제어 코드는 B'00000010'이 된다. 여기서 주의할 점은 분주비를 임의로 선택할 수 없다는 것이며, 이로 인하여 주어진 clock 주파수로는 원하는 시간을 정확하게 만들 수 없다는 것이다. 만약 아주 정확한 시간이 요구될 때는 clock 주파수를 역으로 분주비에 맞추어 만들어 주어야한다.

예 3)

```
;INTERRUPT 시간 설정 및 interrupt 초기화  --- 2.048msec 주기
        BSF     STATUS,RP0          ; RAM BANK 1 선택
        MOVLW   B'00000010' ; 2,048msec
        MOVWF   OPTIONR
        BCF     STATUS,RP0          ; RAM BANK 0 선택
```

```
BSF    INTCON,5          ; TIMER INTERRUPT ENABLE
BSF    INTCON,7          ; GLOBAL INT. ENABLE
GOTO   MAIN_ST
```

4. 두자리 초시계를 만들기 위한 하드웨어 구성

여러 번 설명하였지만 PIC16F84를 이용하여 기능을 구현하려면 I/O가 8+5로 총 13개이므로 이 범위 안에서 회로를 설계하는 것이 하드웨어를 최소화할 수 있다. 7-segment는 data용으로 8bit, digit 선택용으로 2bit 가 사용되어 총 10개가 사용된다. 따라서 남는 pin은 3개가된다. 이중 1개는 부저를 동작시키는 것으로 하면 2개가 남고, 이것을 입력으로 하여 push 스위치를 받아드려 시간을 설정하는 등의 제어 신호로 사용해 보자. 이 방법보다 더 좋은 간단한 방법이 있을까요? 생각해 보세요. 여분 1개는 만들 수 있습니다.

5. 초시간을 만들기 위한 프로그램 작성

(이 부분은 한가지 예이므로 변경해도 됨)

앞에서 약간 설명하였지만, 시계를 만들기 위해서는 기준시간을 만들어야한다. 편의상 앞의 interrupt 시간을 2msec라고 한다면 interrupt 가 발생한 회수를 500번 계수하면 1초가 얻어진다. 500의 값도 8bit를 넘은 값이므로 사용하기가 복잡하므로 이것을 나누어 생각하자. 즉 두자리 초시계를 만들려면 7-segment를 2개 사용하여야하고 scanning을 2번 해야한다. 따라서 interrupt가 한번 들어오면 한자리 숫자 표시를 하고 조작 스위치를 읽고, 다시 들어오면 다음 숫자를 표시하고 시간계수를 1씩 증가시키면 된다. 이것을 반복하다가 시간계수가 250이 되면 1초가 경과된 것이므로 시간계수를 0으로 만들고 초자리 시간 buffer를 1씩 증가시킨다. 초자리 시간 buffer가 9에서 10이 되면 초자리 시간 buffer는 0으로 만들고 10초 자리의 시간 buffer를 1씩 증가한다. 그리고 이 값이 10이 되면 0으로 만들어 준다. 그러면 00에서 99까지를 반복계수하는 두자리 초시계가 만들어지게 된다. 변수 선언은 다음과 같이한다.

```
INT_CNT: interrupt가 들어온 횟수 계수(시간계수)
D_10SEC, D_1SEC: 10자리 1자리 초시계 값
key_IN: 입력 key 값(00: key값없음, 01: 1번 key, 02: 2번 key)

예 4)
;
      LIST   P=16F84
;
;  REGISTER FILE 설정

;  USER REGISTER 변수 설정

;
            ORG         0
            GOTO        START_UP
            ORG         4
; ISR 시작 번지
            MOVWF       W_TEMP ; 현재 사용되고 있는 W REG를 저장

            SWAPF       STATUS,W
            MOVWF       STATUS_TEMP
            CALL        DISP          ; DISPLAY 부 프로그램
            SWAPF       STATUS_TEMP,W ; 저장된 내용으로 복원
            MOVWF       STATUS
            SWAPF       W_TEMP,F
            SWAPF       W_TEMP,W
            BCF         INTCON,2
            RETFIE


; DISPLAY ROUTINE
DISP; 7-segment의 표시 숫자가 2자리이므로 두 자리를 순차적으로 표시해
      ; 야 한다. 따라서 이 프로그램에 들어오는 횟수를 확인하여 처음 들
```

```
        ; 어오는 경우와 다음 들어오는 경우를 구분하여 동작해야한다.
        ; < 처음 들어올 때> 와 < 다음 들어올 때>를 구분하기

DISP1;< 처음 들어올 때>
        ; D_10SEC 변수 내용이 MSD 7-segment에 표시되도록 할 것

        ; key 상태에 따라서 key_IN 변수에 값 넣기
        ;           (00: key값 없음, 01: 1번 key, 02: 2번 key)
                RETURN

DISP2;< 다음 들어올 때>
        ; D_1SEC 변수 내용이 LSD 7-segment에 표시되도록 할 것  .
        ; INT_CNT: interrupt가 들어온 횟수 계수 증가시킬 것
                RETURN
;

; main program 시작

START_UP
                BSF         STATUS,RP0  ; RAM BANK 1 선택
                MOVLW       B'????????'    ; PORT I/O 선택
                MOVWF       TRISA
                MOVLW       B'????????'    ; PORT I/O 선택
                MOVWF       TRISB
; INTERRUPT 시간 설정      --- 2.048msec 주기
                MOVLW       B'00000010'    ; 2.048msec
                MOVWF       OPTIONR
                BCF         STATUS,RP0  ; RAM BANK 0 선택
                BSF         INTCON,5   ; TIMER INTERRUPT ENABLE
                BSF         INTCON,7       ; GLOBAL INT. ENABLE
                GPTO        MAIN_ST

MAIN_ST
```

```
;  여기서부터 USER PROGRAM 작성
;       변수 초기화
; INT_CNT=0, D_10SEC=0, D_1SEC=0, key_IN=0

M_LOOP
: interrupt가 들어온 횟수 확인(시간계수)
            MOVLW       .250
            SUBWF       INT_CNT,W
            BTFSS       STATUS,ZF
            GOTO        XLOOP

      ; 1sec 마다 들어오는 부분
      CK_LOOP
            CLRF        INT_CNT ; 다음 1초를 기다리기 위한 초기화
            INCF        D_1SEC   ; 1초 단위 변수 증가
            MOVLW       .10
            SUBWF       D_1SEC,W
            BTFSS       STATUS,ZF
            GOTO        XLOOP

      ; 10초마다 들어오는 부분
            CLRF        D_1SEC  ; 다음 10초를 기다리기 위한 초기화
            INCF        D_10SEC  ; 10초 단위 변수 증가
            MOVLW       .10
            SUBWF       D_10SEC,W
            BTFSC       STATUS,ZF
            CLRF        D_10SEC             ; 10초 단위를 초기화
            GOTO        XLOOP
      ; 나머지 시간 동안 사용자 기능을 수행하기 위한 프로그램 영역

      XLOOP
      ; key를 확인하여 key에 따른 기능 수행
      ; 기능에 따른 부저 울리기 등
```

```
        .
        .
GOTO  M_LOOP
```

1) DISPLAY 부분 프로그램 작성 요령

① < 처음 들어올 때> 와 < 다음 들어올 때>를 구분하는 방법

변수를 설정하여(DISP_CNT) bit 0이 0이면 처음으로, 1이면 다음이 되도록 한다. 이때 변수는 DISP 부프로그램에 들어올 때마다 1씩 증가시킨다.

② 설정이 가능하려면 표시가 두 종류가 되어야 한다. 즉 현재 시간을 나타내는 것과 설정되는 값을 나타내는 것이다. 이것을 따로 작성하려면 귀찮으므로 표시기는 DISP_1, DISP_2 변수값을 7-segment에 나타내도록 하고 주프로그램에서 현재 시간을 표시하려고 하면 현재 시간이 들어있는 변수를 DISP_1, DISP_2 변수로 이동시키고, 설정 시간을 나타내려면 설정 시간이 들어있는 변수를 읽어서 DISP_1, DISP_2 변수에 넣으면 된다.

2) 나머지 시간동안 사용자 기능을 수행하기 위한 프로그램 작성 요령

① key를 인식하여 설정과 시작 모드로 이동하기

앞 표시부분에서 설명한 것처럼 시작 key가 눌러지면 시계모드로 이동하고, 설정 key가 눌러지면 설정모드에 들어가게 하면 된다. 각 모드에서 처음 시작될 때는 초기값을 넣어 주어야 되지요.

② 설정 key를 인식하여 설정값을 증가시킬 때 아무런 조처 없이 그냥 프로그램하면 XLOOP에 들어오는 시간이 아주 짧음으로 아주 빠른 속도로 값이 증가되어 설정을 정상적으로 할 수 없다. 따라서 key의 동작 속도를 느리게 만들어야 한다. 이를 해결하기 위한 가장 이상적인 방법은 시간계수 buffer(INT_CNT)를 확인하여 특정시간이 경과한 다음에만 key를 확인하여 동작하도록 하며 이 문제를 해결할 수 있다. 약 100msec 단위로 key를 인식하도록 하면 좋지 않을까?

■ 예비문제

1) 왜 Interrupt에 들어가면 하드웨어가 interrupt를 자동으로 disable시켜야 할까요? 설명해 보시오.

2) Interrupt에서 RETFIE 명령어와 BCF INTCON,7 RETURN의 차이를 자세히 설명하시오.

3) 실험의 주 내용이 시계를 만드는 것이며, 이를 위해서는 기준 시간이 필요하다. 기준시간은 초시계인 경우 1sec이면 되는데 왜 interrupt의 발생 주기를 2msec로 하여 사용하고 있는가? 그 이유를 설명해 보시오.

4) 두자리 초시계에서 스위치를 3개로 만들 수 있는 회로를 그리고 기본 동작을 설명해 보시오.

■ 실 험

1) 예 4)를 작성하고 동작시켜 봅시다. 단 “사용자 기능을 수행하기 위한 프로그램 영역”은 편의상 생략합시다.

2) 앞 실험은 시간이 정확하지 않다고 하였으며, 정확한 시간을 얻으려면 발진기의 주파수를 바꾸어야 된다고 하였다. 그러나 프로그램으로도 오차를 최소화 할 수 있으며, 이를 구현해 봅시다.

3) 0.5 초 간격으로 dot가 깜박거리게 만들어 보시오.

☞ disp에서 INT_CNT 변수가 0이냐 125냐에 따라서 dot 데이터를 XORWF를 사용하여 toggle 시키면 된다.

4) 예 4)를 00에서 59까지 나타나도록 해 보시요.

5) 예 4)는 00에서 59까지를 반복 계수한다. 이것을 시작 키(push switch 1)를 누르면 설정된 값에서 00으로 감소하며, 00이 되면 부저를 3초간 계속 울린다. 이때 설정값은 설정 키(push switch 2)를 눌러서 만든다. 이를 구현하시오.