실험 5. SWITCH와 KEYPAD 사용하기, 부저 울리기

1. 개별 스위치 읽기

가장 기본이 되는 I/O 소자 중의 하나가 스위치이다. 스위치의 형태는 다양하나, 프로세서의 입장에서는 '1'이나, '0'를 I/O pin에 가해 주는 역할만을 할 뿐이다. 따라서 스위치의 상태를 읽기 위해서는 먼저 I/O PORT를 INPUT으로 설정하고 PORT를 읽는 명령어를 사용하면 된다. ('마이크로프로세서로 LED 움직이기'에서 설명된 내용임)

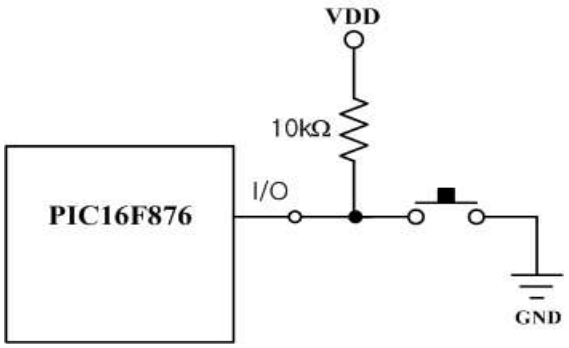

그림 E5-1. 개별 KEY 입력회로

그림 E5-1의 회로에서 스위치가 눌러지지 않으면 +5 *V*에 연결된 저항(pull up 저항이라 함)에 의해서 입력 신호는 +5 *V*가 되어 로직으로는 '1'이 들어오며, 스위치가 눌러지면 0 *V*가 되어 로직으로는 '0'이 들어온다.

2. MATRIX 구조의 keypad에서 key 읽기

스위치의 개수가 적을 경우에는 개별 스위치를 개별 I/O pin에 연결시키면 되나, 스위치가 많아지면 그렇게 할 수가 없다. 예로 PIC16F876는 I/O pin이 22개이므로 최대 22개의 스위치를 직접 읽을 수 있다. 하지만, 받은 키 정보를 숫자로 나타내려면 또 많은 출력 핀이 필요하므로 소자를 효율적으로 사용하기 위해서는 많은 스위치 신호를 받기 위한 다른 방법이 필요하며, 7-SEGMENT와 비슷한 방법으로 구현할 수 있다. 대표적인 방법이 그림

E5-2과 같은 회로를 사용하는 key matrix scanning 방법이다.

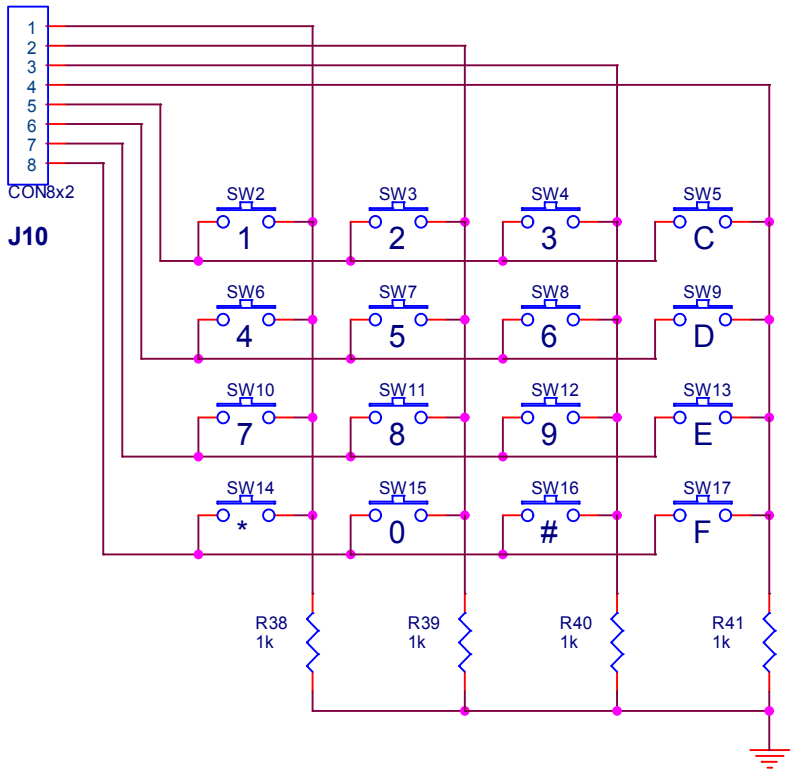

그림 E5-2. KEY MATRIX의 하드웨어 구조

그림 E5-2에서 보면 총 key 개수는 16개이나 외부로 연결되는 단자는 8개이고 이중 4개가 INPUT pin, 4개는 OUTPUT pin으로 사용된다.

☞ 회로를 보고 어떻게 알 수 있을까요? 개별 key 입력회로와 비교해 보세요. pull up된 곳이 입력 측입니다.

즉 INPUT pin 1개에 4개의 key 스위치가 병렬 연결된 것이며, 따라서 특정 key를 인식하기 위해서는 4개중 1개를 선택하기 위한 선택 신호가 필요하다. 이 신호가 OUTPUT PORT를 통해서 주어지게 된다. 즉 OUTPUT pin에 '0'을 출력하는 것만 선택되고, '1'을 출력하는 것은 선택되지 않는다. 즉, key를 누를 경우 입력 측 단자가 '0'으로 변화되는 것을 인식할 수 있기 때문이다. 따라서 14개의 key를 전부 인식하기 위해서는 OUTPUT을 4번 바꿔주고 INPUT을 4번 해 주어야 합니다.

예1) key 스위치 '2'가 눌러졌는가 아닌가를 구분하기 위한 방법
(회로 결선은 J10의 핀5,6,7,8를 RC3, 2, 1, 0에 연결하고, J10의 핀 4,3,2,1를 RA3, 0, 1, 2에 그리고 부저를 RA5에 연결한 경우)

```
;       I/O PORT 선택
;       RC PORT OUTPUT
;       RA5 PORT OUTPUT, RA3,2,1,0 PORT INPUT

        MOVLW   B'00000111'   ; RC3=0, RC2,1,0=1
        MOVWF   PORTC
LP      MOVF    PORTA,W
        ANDLW   B'00001111'
        SUBLW   B'00001101'   ; '2' key 만 눌러졌는가 확인
        BTFSC   STATUS,ZF
        GOTO    BUZZON
        GOTO    BUZZOFF
BUZZON
        BSF     PORTA,5       ; BUZZER ON
        GOTO    LP
BUZZOFF
        BCF     PORTA,5       ; BUZZER OFF
        GOTO    LP
```

☞ 참고: **위 프로그램은 하드웨어 결선 상태에 따라서 변경됩니다. 그러니 절대로 암기 하려고 하지 마시기 바랍니다.**

☞ 참고: 만약 RA4를 출력으로 사용하는 경우는 다음의 주의점이 있습니다. PIC16F876에서 PORTA의 RA4는 schmitt trigger input/open drain output입니다. 그리고 RA0~RA3은 TTL input/full CMOS output입니다. 따라서, 부저를 RA4에 연결할 때 단순히 선으로 두 개를 연결한다면 부저에서 소리가 나지 않습니다. RA4는 output pin으로 사용시 별도의 소자를 추가해야만 동작합니다. data sheet의 I/O 회로 부분을 검토하여 open drain이 무엇을 의미하며, 어떻게 사용하는지를 알아봅시다. 그리고 RA4를 RA5로 변경하여 동작시켜봅시다.

앞의 예는 특정 위치의 특정 key가 눌러졌는가를 인식하기 위한 프로그램이다. 이를 확장하여 key '1', '2', '3', '4', '5', '6' 중에서 어떤 key가 눌러졌는가를 찾아내는 프로그램을 작성해 보도록 하자. 먼저 key '1', '2', '3' 중에서 어떤 key가 눌러졌는가를 찾아내고, 다시 key '4', '5', '6' 중에서 어떤 key가 눌러졌는가를 찾는 것을 반복하면 된다.

그리고 편이성을 위해서 key-pad는 3*4(12개)만 있는 것으로 프로그램을 작성하고 정리한다.

예2)

```
KEY_IN
    MOVLW     0FH    ;초기값으로 key가 눌러지지 않음 설정
    MOVWF     KEY_DATA
LP  MOVLW     B'00000111'    ; RC3=0, RC2,1,0=1
    MOVWF     PORTC
    ;
    MOVF      PORTA,W     ; key 읽기
    ANDLW     B'00000111'
    SUBLW     B'00000011'    ; 1번 key가 눌러졌는가 확인
    BTFSC     STATUS,ZF
    GOTO      SW_1
    MOVF      PORTA,W
    ANDLW     B'00000111'
    SUBLW     B'00000101'    ; 2번 key가 눌러졌는가 확인
    BTFSC     STATUS,ZF
    GOTO      SW_2
    MOVF      PORTA,W
    ANDLW     B'00000111'
    SUBLW     B'00000110'    ; 3번 key가 눌러졌는가 확인
    BTFSC     STATUS,ZF
    GOTO      SW_3
    ;
    MOVLW     B'00001011'    ; RC3,1,0=1, RC2=0
    MOVWF     PORTC
    ;
```

```
    MOVF        PORTA,W     ; key 읽기
    ANDLW       B'00000111'
    SUBLW       B'00000011'   ; 4번 key가 눌러졌는가 확인
    BTFSC       STATUS,ZF
    GOTO        SW_4
    MOVF        PORTA,W
    ANDLW       B'00000111'
    SUBLW       B'00000101'   ; 5번 key가 눌러졌는가 확인
    BTFSC       STATUS,ZF
    GOTO        SW_5
    MOVF        PORTA,W
    ANDLW       B'00000111'
    SUBLW       B'00000110'   ; 6번 key가 눌러졌는가 확인
    BTFSC       STATUS,ZF
    GOTO        SW_6
    GOTO        LP
SW_1
    MOVLW       1
    GOTO        XLP
SW_2
    MOVLW       2
    GOTO        XLP
SW_3
    MOVLW       3
    GOTO        XLP
SW_4
    MOVLW       4
    GOTO        XLP
SW_5
    MOVLW       5
    GOTO        XLP
SW_6
    MOVLW       6
```

```
XLP
    MOVWF       KEY_DATA
    ; KEY_DATA 값이 눌러진 key 값이됨
    GOTO        LP
```

이렇게 프로그램을 나열하면 생각은 단순하나 너무 지저분하지요. 반복 부분을 변수를 사용하여 간단히 할 수 없을까요? 반복하는 부분이 OUTPUT과 INPUT 인데, OUTPUT은 '0111'에서 '1011', '1101', '1110' 다시 '0111'로 반복하며, INPUT는 '011', '101', '110'으로 반복한다. 먼저 OUTPUT는 4bit를 '0111'에서 오른쪽으로 shift(rotate 명령어 사용)하면 된다. 구체적으로는 rotate 명령어는 CARRY와 함께 동작하므로 CARRY에 대한 고려를 해야 한다. 그러나 RB3, 2, 1, 0를 사용하고, B'11110111'을 초기값으로 두고 RRF 명령어를 사용하면 들어오는 CARRY는 고려할 필요가 없으며, 끝은 CARRY가 발생하지 않을 때이다.

예3)

```
            MOVLW       0FH             ; key가 눌러지지 않음 설정
            MOVWF       KEY_DATA
    ;
    LP2     MOVLW       B'11110111'     ; RC3=0, RC2,1,0=1
            MOVWF       PORTC
    ;
    LP1     CALL        READ_KEY        ; key가 눌러짐 확인
            RRF         PORTC,F
            BTFSC       STATUS,CF
            GOTO        LP1
    ;   마지막 LOOP --> 초기화
            GOTO        LP2
    ; key가 눌러짐 확인 부 프로그램
    READ_KEY
            ; 여기서 확인
            RETURN
```

그러나 key가 눌러짐을 확인하는 부분은 아직도 복잡하지요. 이를 간단히 구현하는 방법으로 table lookup 방법을 사용할 수 있지요. 즉 OUTPUT 상태

와 INPUT 상태로 key의 동작상태가 결정되므로, 이 두 조건을 합하여 table 주소를 만들고 그 주소에 key 값을 넣으면 아주 단순해지고 key 값을 바꾸기도 편리하지요.

예4)

OUTPUT 값(PORTC의 BIT3,2,1,0) + INPUT 값(PORTA의 BIT2,1,0)
--> 주소 값('PORTC의 BIT3,2,1,0' +'PORTA의 BIT2,1,0')

이렇게 하면 주소 값이 7 bit가 되어 경우이 수가 2^7(=128개)이 되며, 이를 바로 table로 만들게 되면 table이 너무 커지게 되지요. 따라서 약간의 변형을 가하여 OUTPUT이나 INPUT 값을 그대로 사용하지 않고, 축소시켜서 사용하지요. 한 예로 OUTPUT는 1110, 1101, 1011, 0111로 4가지 경우만 가지므로 이를 구분하기 위해서는 2 bit만 있으면 됩니다. 따라서 table address는 출력과 입력의 조합으로 만들어진 5bit로 만들 수 있습니다.

TABLE ADDRESS =
BIT4,3,2(KEY_IN PORT BIT2,1,0) + BIT1,0(SCAN BIT 1,0)

예5) '0' key가 눌려지면 소리내기

```
LP      MOVLW       0FH
        MOVWF       KEY_DATA    ; 초기화
LP2     MOVLW       B'11110111'  ; RC3=0, RC2,1,0=1
        MOVWF       PORTC
        CLRF        KEY_T       ; SCAN 위치
;
LP1     CALL        READ_KEY    ; key가 눌러짐 확인
        INCF        KEY_T,F
        RRF         PORTC,F     ; 다음 위치 선택
        BTFSC       STATUS,CF   ; 마지막 위치 확인
        GOTO        LP1
;  마지막 위치 --> key 값에 따라서 주어진 일하기
        MOVF        KEY_DATA,W
        SUBLW       0           ; '0' key 인가 확인
        BTFSS       STATUS,ZF
        GOTO        LP3
; 특정시간 동안 소리내기
```

```
        BSF         PORTA,4     ; BUZZER ON
        CALL        DELAY
LP3     BCF         PORTA,4     ; BUZZER OFF
        GOTO        LP          ;   --> 초기화

; 스위치가 눌러짐 확인
READ_KEY
        MOVF        PORTA,W     ; 스위치 읽기
        ANDLW       B'00000111'
        SUBLW       B'00000111'
        BTFSC       STATUS,ZF   ; key 눌러짐 확인
        RETURN                  ;key가 눌러지지 않으면 그냥
                                 return
; key 값을 얻기 위한 TABLE ADDRESS 만듦
        MOVF        PORTA,W
        MOVWF       KEY_DATA
        RLF         KEY_DATA,F
        RLF         KEY_DATA,W
        ANDLW       B'00011100'
        IORWF       KEY_T,W
        ;ANDLW      B'00011111'
        CALL        KEY_TABLE
        MOVWF       KEY_DATA        ; 들어온 스위치 값
        RETURN
; KEY 값을 저장하는 TABLE  -- 32 개임
KEY_TABLE
        ADDWF       PCL,F
        RETLW       0FH   ; '000'+'00' 일때
        RETLW       0FH   ; '000'+'01' 일때
        RETLW       0FH   ; '000'+'10' 일때
        RETLW       0FH   ; '000'+'11' 일때
        RETLW       0FH   ; '001'+'00' 일때
        RETLW       0FH   ; '001'+'01' 일때
```

```
        RETLW   0FH     ; '001'+'10' 일때
        RETLW   0FH     ; '001'+'11' 일때
        RETLW   0FH     ; '010'+'00' 일때
        RETLW   0FH     ; '010'+'01' 일때
        RETLW   0FH     ; '010'+'10' 일때
        RETLW   0FH     ; '010'+'11' 일때
        RETLW   01H     ; '011'+'00' 일때
        RETLW   04H     ; '011'+'01' 일때
        RETLW   07H     ; '011'+'10' 일때
        RETLW   10H     ; '011'+'11' 일 때 -- '*' code
;
        RETLW   0FH     ; '100'+'00' 일때
        RETLW   0FH     ; '100'+'01' 일때
        RETLW   0FH     ; '100'+'10' 일때
        RETLW   0FH     ; '100'+'11' 일때
        RETLW   02H     ; '101'+'00' 일때
        RETLW   05H     ; '101'+'01' 일때
        RETLW   08H     ; '101'+'10' 일때
        RETLW   00H     ; '101'+'11' 일때
        RETLW   03H     ; '110'+'00' 일때
        RETLW   06H     ; '110'+'01' 일때
        RETLW   09H     ; '110'+'10' 일때
        RETLW   11H     ; '110'+'11' 일 때 -- '#' code
        RETLW   0FH     ; '111'+'00' 일때
        RETLW   0FH     ; '111'+'01' 일때
        RETLW   0FH     ; '111'+'10' 일때
        RETLW   0FH     ; '111'+'11' 일때
```

이해할 수 있습니까? 왜 스위치는 12개인데 table은 32개가 되는지? 지금까지 설명한 내용도 한 예 일뿐 프로그램에서는 이것만이 유일하다는 것은 없습니다. 동일한 방식으로 만약 key가 16개이면 table의 길이는 2^6(64)개가 되므로 앞에서 12개로 제한한 것입니다.

☞ 참고: 예2)와 같이 table을 사용하지 않고 개별 조건에 따른 key 값을 일

일이 프로그램에서 만들어 줄 수가 있습니다. 그러나 이 방법은 간단한 경우에만 사용합니다. 그 이유는 많은 key를 필요로 하거나, key 값을 변경하고자 할 경우에는 table 값만 변경하면 되기 때문입니다.

3. 스위치를 읽어서 숫자를 7-SEGMENT에 표시하기

앞에서 실습해 보았던 7-segment 와 스위치를 연결 동작 시켜 보도록 합시다. 여기서도 스위치를 읽는 동작과 숫자를 표시하는 동작을 병행처리 해야하므로 약간은 생각을 해보아야 합니다. PC에서 프로그램은 순서대로 처리되며, 한번에 두 가지 일을 동시에 처리할 수 없으므로 두 가지 프로그램을 완전히 별개로 작성하여 사용하기는 어렵지요. 따라서 2가지 일을 하나의 흐름으로 만들어 동작시켜야 합니다. 다시 말하면, 한번은 스위치를 읽고, 한번은 표시하고, 다시 스위치를 읽고 등으로 계속 반복시키는 것입니다. 만약 한 명령을 처리하는 속도가 아주 빠르다면, 이 반복과정이 동시에 처리되는 것처럼 보일 것입니다. 그래서 요즈음 컴퓨터의 clock 주파수가 계속 높아지고 있지요. 컴퓨터에서 한 두 가지의 일이 아닌 여러 가지 일을 시킨다면 각각의 일을 나누어 순차적으로 수행시켜야 하는데, 이 작업을 매번 프로그램마다 구현해 준다는 것은 여간 복잡하고도 어려운 일이 아니지요. 여러분 생각에도 이런 일을 용이하게 처리하도록 하는 기능이 필요할 것이라고 보이지요. 이런 일을 용이하게 처리하도록 하는 기능이 앞으로 배워야할 interrupt라는 기능입니다. 이 기능은 현재 수행되고 있는 프로그램에는 영향을 주지 않고 미리 설정된 다른 프로그램을 수행시키는 것으로 우리는 이 인터럽트 프로그램 영역을 interrupt service routine(ISR)이라고 합니다. 우리가 사용하는 PIC16F876도 이런 기능이 있습니다. data sheet의 memory map에 보면 04H번지가 interrupt vector라고 표현되어 있지요. 즉 04H번지가 ISR의 시작번지라는 것입니다. 여기서 말하는 현재 수행되고 있는 프로그램에는 영향을 주지 않고 라는 말은 기능적으로 그렇게 할 수 있다는 말이며, 구체적으로는 두 프로그램 사이에 중복 사용되는 변수 저장 영역 즉 register가 없도록 해 주어야 합니다. PC의 window에서는 여러 가지 일을 동시에 처리하는 것으로 보이지요. 이것이 interrupt 기능이며 여러분이 가장 쉽게 느낄 수 있는 예가 될 수 있습니다. 자세한 것은 뒤에서 다시 설명하겠습니다. 너무 많이 하려면 잊어 먹지요. 기초부터 확실히 합시다.

일단 interrupt를 사용하지 않으니, 여러분이 2가지 일을 순차적으로 수행되도록 프로그램을 작성해야 합니다. 즉 스위치를 읽어서 1자리 7-segment에 표시하기 위해서는 읽는 프로그램과 읽은 결과를 출력하는 프로그램 두 부분을 순서대로 나열하여 반복하게 하면 됩니다.

예6)

```
LP      CALL        KEY_IN
        MOVF        KEY_DATA, W
        CALL        CONV        ; 숫자를 7-segment 값으로 변
                                    경
        MOVWF       PORTC       ; 숫자 값 출력
        MOVLW       B'00001000'
        MOVWF       PORTA       ; DG1 위치 결정
        GOTO        LP
```

KEY_IN은 예5)의 key를 읽어서 결과를 KEY_DATA 변수에 저장하는 프로그램을 부 프로그램으로 만든 것이며, 단위 기능을 가지는 프로그램은 부 프로그램으로 만들어 사용하면 편리합니다. 여기서 예5)의 프로그램을 KEY_IN 부 프로그램으로 이용하기 위해서는 예5)의 프로그램을 조금 수정해야합니다. (부 프로그램에는 RETURN이 있어야 합니다.)

☞ 부 프로그램을 사용할 때는 이름을 'CALL 이름'으로 불러야 하며, 부 프로그램의 마지막(외형적인 끝이 아니고 수행될 때 마지막 위치)에는 반드시 원 위치로 되돌아오라는 의미의 명령어가 있어야 부 프로그램을 수행한 후 'CALL 이름' 다음의 명령어를 수행하게 됩니다. Return 의미로 사용되는 명령어는 'RETURN', 'RETLW K', 'RETFIE' 가 있습니다. 또 당연히 CALL 명령어는 부 프로그램으로 이동하기 전에 되돌아 올 위치를 어디엔가 저장해 주어야 합니다. 이것은 마이크로프로세서의 내부 기능으로 이루어지며, 저장 영역을 STACK, 저장되는 위치(주소)를 SP(stack pointer)라고 합니다. 명령어의 의미를 이해해 보도록 합시다.

Data sheet의 memory map을 보면 stack level 이라는 곳이 있습니다. PIC16F876는 level 1에서 8까지 되어 있으며, 이는 부 프로그램에서 부 프로

그램을 호출하는 것이 8단계까지 가능하다는 것입니다. 다시 말하면, CALL이 연속적으로 8번 이상 발생되면 안 된다는 것을 의미합니다. 왜냐하면 돌아올 주소가 더 이상 기억될 수 없기 때문이지요.

☞ 예와 같이 앞으로 작성된 프로그램은 약간씩 변경하여 반복 사용하기 되므로, 지워지지 않도록 본인이 관리합시다. (USB memory에 보관 필요)

이제부터는 명령어를 몰라서 프로그램을 작성하지 못하는 것이 아니고, 어떻게 구현할 것이냐가 더 중요한 문제로 보이지 않나요. 잘 하려면 물론 많은 노력과 경험이 필요하지요. 어떻게 하냐고요? 앞으로는 이렇게 시작해 봅시다.

1) 프로그램을 작성하기 전에 어떻게 해야 가장 간단하게 쉽게 그리고 변경이 용이하게 작성 할 것인가를 고민해야 한다.
2) 그리고 이것을 'flow chart'라는 것으로 만들어 먼저 프로그램의 흐름을 잡은 것이 좋습니다.
3) **여러분도 앞으로는 flow chart를 꼭 작성하고 coding하기 바랍니다.** --중요! 중요!

☞ flow chart를 작성할 때 사용되는 기호가 있으므로 참고 자료를 찾아봅시다.

☞ 일반적으로 가장 기본적이며 중요한 I/O가 key와 7-segment 표시기이며, 두 경우 모두 scanning 방법을 사용하므로 data port는 별개로 사용하더라도, 출력으로 위치를 선택하는 scanning port를 공용으로 사용함으로서 회로 및 프로그램을 최소화할 수 있습니다. 예로 4자리 숫자와 4개의 스위치를 가지는 회로를 구성해봅시다. (단순하게는 I/O pin이 8+4+4 개가 필요하나, 이를 최소의 I/O pin 만 사용하여 구현해 봅시다.)

4. SWITCH 사용 시 주의할 점

그리고 push 스위치를 사용하여 누른 횟수에 따라서 동작시키려고 할 때에는 다음을 명심해야 된다. 즉 스위치는 기계적인 구조를 가지므로, 여러분이 한번 ON/OFF 시키면 한번의 펄스만이 나와야 하나, 기계적인 떨림에 의하여 아주 짧은 순간에 여러 번 신호가 나온 후에 안정된 상태로 들어가게 된다.

이를 bounce, chattering 이라고 한다. 따라서 이러한 신호를 바로 읽어서 처리하면, 여러 번 스위치를 누른 것과 같은 현상이 나타나게 된다. 이러한 영향을 적게 받으려면, 또는 잡음의 영향을 적게 받으려면, INPUT 체크를 한번만 하지 않고 특정 시간 지연 후 한번 더 읽어서 두 개의 상태가 같을 때만 입력으로 인식하거나, 특정 시간 이상 계속 같은 신호가 들어올 경우에만 인식하도록 하는 등의 방법을 사용해야 한다.

■ 예비문제 5

1) PC의 keyboard에는 key가 몇 개 있으며, 많은 스위치를 어떻게 받아드릴까 설명해 보시오.

2) 54개의 INPUT 스위치를 최소의 I/O pin으로 구현하려고 한다. 최소 몇 개가 필요한가.

3) 그림 E5-2에서 2와 4 및 5를 동시에 누르면 어떤 현상 나타날까 설명해 보시오. 그리고 이러한 경우가 발생되는 것을 막으려면 회로를 어떻게 수정해야 하나요. 최소 소자로 구현해 보시오.
(☞ Sneak path 라는 용어를 참고하시오.)

4) 예1)에서 4와 6를 동시에 누를 경우만 소리가 나도록 만들어 보시오.

5) 프로그램을 작성하면서 CALL과 RETURN으로 된 부 프로그램을 많이 사용하게 되는데, 이것은 어떻게 동작할까요? 그리고 사용상의 제약점은 없나요? 확인해 봅시다.

6) Open Drain에 대해서 조사하시오.(구조, 사용방법, 이용분야 등)
(☞ data sheet의 I/O part 회로도 부분을 참고하십시오. 그리고 왜 다양한 형태의 출력회로가 필요한지도 검토해 보시오.)

■ 실 험 5. KEYPAD 사용하기와 소리내기

1) 예1)를 실험으로 확인하고, 의미를 이해합시다. 이때 RA4는 Open drain 출력이므로 이것을 고려하여 배선을 해야 합니다.

☞ Hint: Open drain는 출력에 1이 나오지 않는 회로 입니다. 따라서 1를 나오게 하려면 저항(약 1KΩ)으로 pull up 해야 합니다. 그러나 실험보드에는 이미 저항으로 PULL UP 되어 있습니다.

2) 예비문제 4).를 실험으로 확인합시다.

3) 예5)를 실험으로 확인하시오. 이때 프로그램 실행 초기에 부저가 울리게 됩니다. 그 이유를 data sheet의 RA4 PORT 회로도와 register file summary의 value on power on reset 조건과 연관시켜 설명하시오.

4) 초기에 부저가 울리지 않도록 프로그램적으로 해결할 수 있는가? 아니면 하드웨어가지 수정해야 하는가? 이유 및 방법을 설명하고 구현해 보시오.

5) '7'과 '8' 스위치를 동시에 누를 경우에 만 소리가 나도록 프로그램을 변경해 보시오.

6) 예6)을 실험으로 확인하여 봅시다.
(이 부분은 조금 어려운 내용으로 1주 실험내용에서는 생략해도 됩니다.)

① 예6)은 완전한 프로그램이 아닙니다. 단지 key input과 display를 나열한 것입니다.

② 하드웨어 측면에서 1개의 7-segment를 사용하기 위해서는 최소 8개 output pin(9개가 되어야 제어 가능함)이 필요하며, keypad 역시 7개의 I/O핀이 필요합니다. 그리고 부저에 1개의 output pin이 필요합니다. 이를 13개의 I/O 핀 만 사용하여 단순하게 해결할 수 있나요?

해결방법은...
- 외부에 별도의 회로를 추가한다. (앞으로 해야 할 일)
- I/O pin이 많은 PIC 소자를 사용한다.

- display output pin과 keypad output pin을 공통으로 사용한다. (이번 실험에서는 이 방법을 이용합니다.) 즉 1개의 output으로 한번은 표시데이터, 한번은 key 데이터로 순환시키면 되겠지요. 앞 실험의 7-segment 동작시키기와 같은 원리를 적용하면 됩니다.

③ PIC16F876와 7-segment, keypad, 부저의 연결 회로도를 작성하시오.
④ 회로를 동작시키기 위한 flow chart를 작성하시오.

7) 앞에서 사용했던 예)부 프로그램들을 사용하여 예6)의 프로그램을 실행하면 key를 누를 때만 7-segment에 숫자가 표시되고, key를 누르지 않으면 'F'가 표시됩니다. 프로그램을 수정하여 누른 숫자가 7-segment에 계속 나타나도록 하시오. 그리고, key를 누를 때만 key 입력 확인용으로 짧은 시간 동안 소리가 나도록 하시오.

■ HOME WORK 5

1) Stack이란 무엇인가? 그리고 어떻게 동작되는가. 주의점은 무엇인가 등등..

2) 예5) 보다 더 간단한 방법으로 프로그램 하여 보시요.

3) 4 자리 7-segment와 4개의 스위치를 동시에 동작시킬 수 있는 하드웨어를 그리고, 동작 순서를 설명하시오.

4) 4*4(16) keypad를 이상적으로 인식할 수 있는 table를 만들어 봅시다.