Microprocessor Interfacing & Programming

LECTURE 15

Checking an input pin

- To make decisions based on the status of a given bit in the file register, we use two instructions:

  BTFSC (bit test fileReg skip if clear)

  BTFSS (bit test fileReg skip if set)

- These instructions allow us to monitor a single pin and make decision depending on whether it is 0 or 1.

- To monitor the status of a single bit for HIGH, we use BTFSS instruction.
- To monitor the status of a single bit for LOW, we use BTFSC instruction.

Write a program to perform the following:
(a) Keep monitoring the RB2 bit until it becomes HIGH;
(b) When RB2 becomes HIGH, write value 45H to Port C, and also send a HIGH-to-LOW pulse to RD3.

Solution:

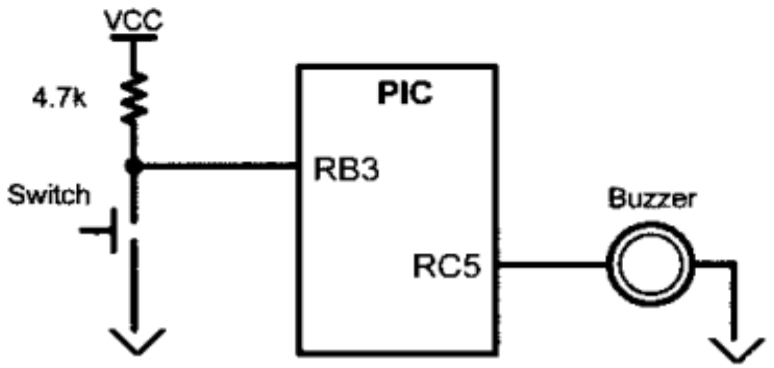
```
      BSF   TRISB,2      ;make RB2 an input
      CLRF  TRISC        ;make PORTC an output port
      BCF   PORTD,3      ;make RD3 an output
      MOVLW 0x45         ;WREG = 45h
AGAIN BTFSS PORTB,2      ;bit test RB2 for HIGH
      BRA   AGAIN        ;keep checking if LOW
      MOVWF PORTC        ;issue WREG to Port C
      BSF   PORTD,3      ;bit set fileReg RD3 (H-to-L)
      BCF   PORTD,3      ;bit clear fileReg RD3 (L)
```

In this program, instruction "`BTFSS  PORTB,2`" stays in the loop as long as RB2 is LOW. When RB2 becomes HIGH, it skips the branch instruction to get out of the loop, and writes the value 45H to Port C. It also sends a HIGH-to-LOW pulse to RD3.

Assume that bit RB3 is an input and represents the condition of a door alarm. If it goes LOW, it means that the door is open. Monitor the bit continuously. Whenever it goes LOW, send a HIGH-to-LOW pulse to port RC5 to turn on a buzzer.

Solution:

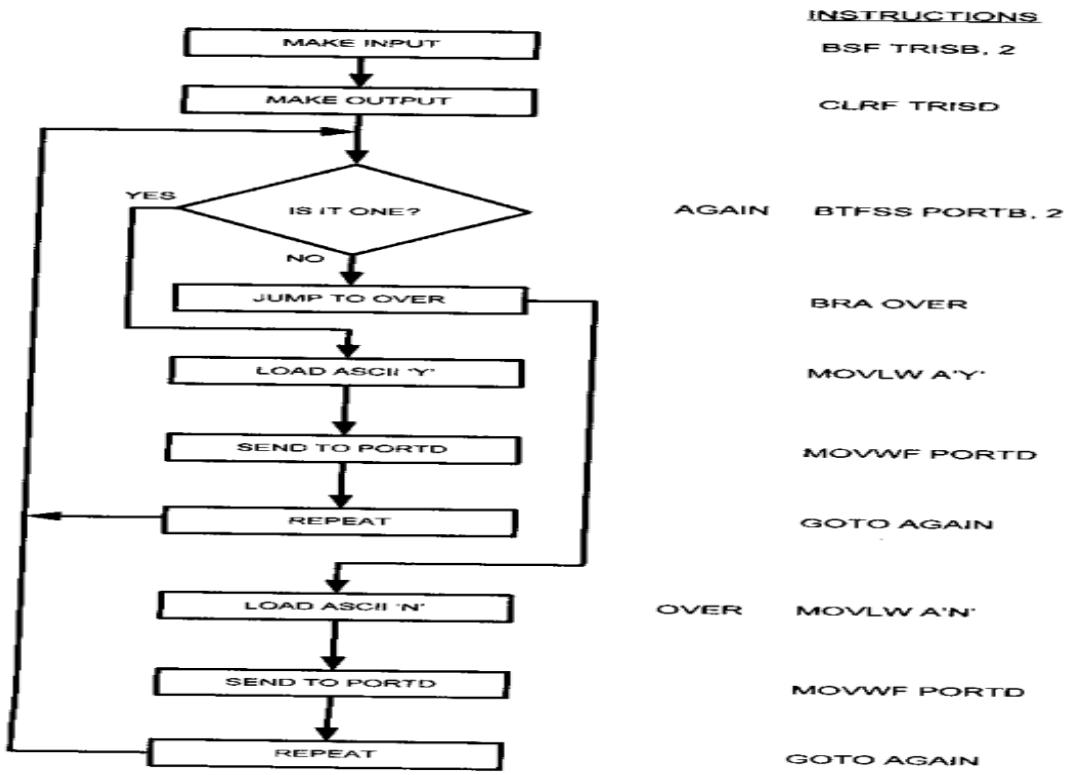
```
        BSF    TRISB,3      ;make RB3 an input
        BCF    TRISC,5      ;make RC5 an output
HERE    BTFSC  PORTB,3      ;keep monitoring RB3 for HIGH
        BRA    HERE         ;stay in the loop
        BSF    PORTC,5      ;make RC5 HIGH
        BCF    PORTC,5      ;make RC5 LOW for H-to-L
        BRA    HERE
```



A switch is connected to pin RB2. Write a program to check the status of SW and form the following:
(a) If SW = 0, send the letter 'N' to PORTD.
(b) If SW = 1, send the letter 'Y' to PORTD.

Solution:

```
        BSF     TRISB,2         ;make RB2 an input
        CLRF    TRISD           ;make PORTD an output port
AGAIN   BTFSS   PORTB,2         ;bit test RB2 for HIGH
        BRA     OVER            ;it must be LOW
        MOVLW   A'Y'            ;WREG = 'Y' ASCII letter Y
        MOVWF   PORTD           ;issue WREG to PORTD
        GOTO    AGAIN           ;we can use BRA too
OVER    MOVLW   A'N'            ;WREG = 'N' ASCII letter N
        MOVWF   PORTD           ;issue WREG to PORTD
        GOTO    AGAIN           ;we can use BRA too
```



	INSTRUCTIONS
	BSF TRISB, 2
	CLRF TRISD
AGAIN	BTFSS PORTB, 2
	BRA OVER
	MOVLW A'Y'
	MOVWF PORTD
	GOTO AGAIN
OVER	MOVLW A'N'
	MOVWF PORTD
	GOTO AGAIN

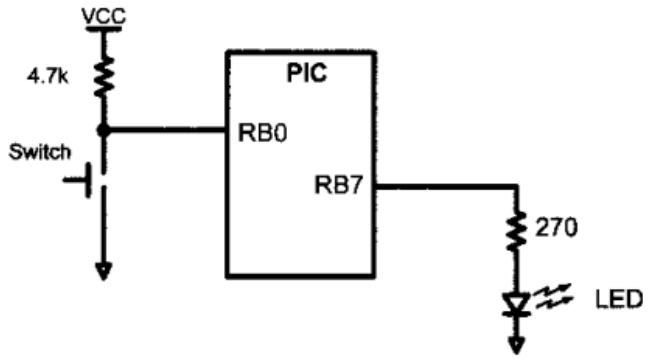
Reading a single bit

- We can also use the bit test instructions to read the status of a single bit and send it to another bit or save it.

A switch is connected to pin RB0 and an LED to pin RB7. Write a program to get the status of SW and send it to the LED.

Solution:

```
       BSF    TRISB,0      ;make RB0 an input
       BCF    TRISB,7      ;make RB7 an output
AGAIN  BTFSS  PORTB,0      ;bit test RB0 for HIGH
       GOTO   OVER         ;it must be LOW (BRA is OK too)
       BSF    PORTB,7
       GOTO   AGAIN        ;we can use BRA too
OVER   BCF    PORTB,7
       GOTO   AGAIN        ;we can use BRA too
```

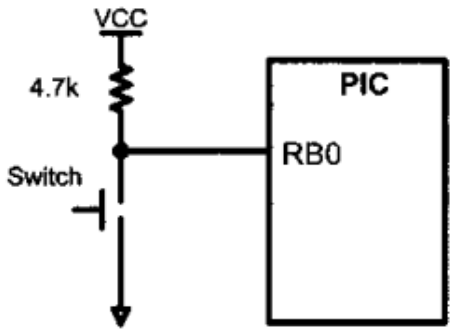


A switch is connected to pin RB0. Write a program to get the status of SW and save it in D0 of fileReg location 0x20.

Solution:

```
MYBITREG EQU 0x20 ;set aside loc 0x20 reg

       BSF   TRISB,0      ;make RB0 an input
AGAIN  BTFSS PORTB,0      ;bit test RB0 for HIGH
       GOTO  OVER         ;it must be LOW (BRA is OK too)
       BSF   MYBITREG,0   ;set bit 0 of fileReg
       GOTO  AGAIN        ;we can use BRA too
OVER   BCF   MYBITREG,0   ;clear bit 0 of fileReg
       GOTO  AGAIN        ;we can use BRA too
```



Reading input pins vs LATx port

- In reading a port, some instructions read the status of the port while others read the status of an internal port called LATx.
- When reading ports, two possibilities:

1. Read the status of the input pin.
2. Read the internal latch of the LAT register.

"COMF PORTB" is an example of the instruction that reads the contents of an internal port latch.

Sequence of actions in COMF

1. The instruction reads the internal latch of the LATB and brings that data into the CPU.
2. This data is complemented.
3. The result is rewritten back to the LATB latch.
4. The data on the pins are changed only if the TRISB bits are cleared to 0s.

NOTE: It is very rare that we use an instruction to read the latch register, such as "COMF LATB, F".

- The instructions that read the port latch normally read a latch value, perform an operation, then rewrite it back to the port latch. This **is read-modify-write.**

NOTE: To use read-modify-write, the port must be configured as output.

Table 4-10: Some of the Read-Modify-Write Instructions

Instruction		Function
ADDWF	fileReg,d	Add WREG to f
BSF	fileReg,bit	Bit Set fileReg (set the bit: bit = 1)
BCF	fileReg,bit	Bit Clear fileReg (clear the bit: bit = 0)
COMF	fileReg,d	Complement f
INCF	fileReg,d	Increment f
SUBWF	fileReg,d	Subtract WREG from f
XORWF	fileReg,d	Exclusive-OR WREG with f

Arithmetic instructions

- **Unsigned numbers:** Data in which all the bits are used to represent data, and no bits are set aside for the positive or negative sign.
- It means, the operand can be between 00 and FFH (0 to 255 decimal) for 8 bit data.

Addition of unsigned numbers: In addition, WREG register must be involved.

SYNTAX: ADDLW K ; WREG = WREG + K

The sum is stored in WREG register.

The instruction can change any of the C, DC, Z, N or OV bits of the status register.

Example

Show how the flag register is affected by the following instructions.

```
MOVLW 0xF5          ;WREG = F5 hex
ADDLW 0xB           ;WREG = F5 + 0B = 00 and C = 1
```

Solution:

```
  F5H          1111 0101
+ 0BH        + 0000 1011
 100H          0000 0000
```

After the addition, register WREG contains 00 and the flags are as follows:
C = 1 because there is a carry out from D7.
Z = 1 because the result in WREG is zero.
DC = 1 because there is a carry from D3 to D4.

ADDWF and addition of individual bytes

- "ADDWF fileReg, d" allows the addition of WREG and individual bytes residing in RAM locations of the file register.

NOTE: WREG must be involved because memory-to-memory arithmetic operations are never allowed.

Assume that file register RAM locations 40–43H have the following hex values. Write a program to find the sum of the values. At the end of the program, location 6 of the file register should contain the low byte and location 7 the high byte of the sum.

```
40 = (7D)
41 = (EB)
42 = (C5)
43 = (5B)
```

Solution:

```
L_Byte EQU  0x6          ;assign RAM location 6 to L_byte of sum
H_Byte EQU  0x7          ;assign RAM location 7 to H_byte of sum

       MOVLW 0           ;clear WREG (WREG = 0)
       MOVWF H_Byte      ;H_Byte = 0
       ADDWF 0x40,W      ;WREG = 0 + 7DH = 7DH , C = 0
       BNC   N_1         ;branch if C = 0
       INCF  H_Byte,F    ;increment (now H_Byte = 0)
N_1    ADDWF 0x41,W      ;WREG = 7D + EB = 68H and C = 1
       BNC   N_2         ;
       INCF  H_Byte,F    ;C = 1, increment (now H_Byte = 1)
N_2    ADDWF 0x42,W      ;WREG = 68 + C5 = 2D and C = 1
       BNC   N_3         ;
       INCF  H_Byte      ;C = 1, increment (now H_Byte = 2)
N_3    ADDWF 0x43,W      ;WREG = 2D + 5B = 88H and C = 0
       BNC   N_4         ;
       INCF  H_Byte,F    ;(H_Byte = 2)
N_4    MOVWF L_Byte      ;now L_Byte = 88h
```

At the end the fileReg location 6 = (8B), and location 7 = (02) because 7D + EB + C5 + 5B + 30 = 28BH. We can use the register indirect addressing mode to do this program much more efficiently. Chapter 6 shows how to do that.

ADDWFC, addition of 16-bit numbers

- When adding two 16-bit data operands, the propagation of a carry from lower byte to the higher byte is considered. This is called **multi-byte addition.**

```
      1
    3C E7
+   3B 8D
   ------
    78 74
```

Write a program to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Assume that fileReg location 6 = (8D) and location 7 = (3B). Place the sum in fileReg locations 6 and 7; location 6 should have the lower byte.

Solution:

```
        ;location 6 = (8D)
        ;location 7 = (3B)

        MOVLW 0xE7          ;load the low byte now (WREG = E7H)
        ADDWF 0x6,F         ;F = W + F = E7 + 8D = 74 and CY = 1
        MOVLW 0x3C          ;load the high byte (WREG = 3CH)
        ADDWFC 0x7,F        ;F = W + F + carry, adding the upper byte
                            ;with Carry from lower byte
                            ;F = 3C + 3B + 1 = 78H (all in hex)
```

Notice the use of ADDWF for the lower byte and ADDWFC for the higher byte.

BCD (binary coded decimal)

- BCD because in everyday life we use the digits 0 to 9, not binary or hex numbers.
- Binary representation of 0 to 9 is called BCD.

- Two terms for BCD:

1. Unpacked BCD
2. Packed BCD

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Unpacked BCD: the lower 4 bits of the number represents the BCD number, and the rest of the bits are 0.

e.g; 0000 1001 and 0000 0101 are unpacked BCD for 9 and 5

- Unpacked BCD requires 1 byte of memory, or an 8 bit register to contain it.

Packed BCD: a single byte has two BCD numbers in it: one in the lower 4 bits, and one in the upper 4 bits.

e.g; 0101 1001 is packed BCD for 59H

- Only 1 byte of memory is need to store the packed BCD operands.

BCD

- Packed BCD is twice as efficient in storing data.

Problem: After adding packed BCD numbers, the result is no longer BCD.

```
MOVLW   0x17
ADDLW   0x28
```

The result should have been 17 + 28= 45 (0100 0101).

Solution: Add 6 (0110) to the lower digit

If problem arises in upper digit, add 6 to the upper digit. (52H + 87H)

PIC18 have an instruction "DAW" to deal with this issue

DAW (decimal adjust WREG)

- This instruction is to correct the aforementioned problem associated with the BCD addition.
- DAW will add 6 to the lower nibble or higher nibble if needed, otherwise, it will leave the result alone.

```
MOVLW  0x47
ADDLW  0x25
DAW
```

Summary of DAW action

After any instruction,

1. If the lower nibble(4 bits) is greater than 9, or if DC=1, add 0110 to the lower 4 bits.
2. If the upper nibble is greater than 9, or if C=1, add 0110 to the upper 4 bits.

- In reality, there is no use for the DC (auxiliary carry) flag bit other than for BCD addition and correction.

```
MOVLW 0x00   ;WREG = 0
ADDLW 0x09   ;WREG = 0x09
ADDLW 0x08   ;WREG = 0x11, DC = 1
DAW          ;WREG = 0x17 (9 + 8 = 17)
```

As another example, examine the case of adding 55H and 77H.

Hex	BCD	
57	0101 0111	
+ 77	+ 0111 0111	
CE	1100 1110	
+ 66	+ 0110 0110	
134	1 0011 0100	Note C = 1

```
MOVLW 0x0C   ;WREG = 00001100
DAW          ;WREG = 00001100 + 00000110 = 00010010 = 0x12
```

Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD.

```
40 = (71)
41 = (88)
42 = (69)
43 = (97)
```

Solution:

```
L_Byte      EQU    0x6          ;assign RAM loc 6 to L_Byte of sum
H_Byte      EQU    0x7          ;assign RAM loc 7 to H_Byte of sum

        MOVLW       0           ;clear WREG (WREG = 0)
        MOVWF       H_Byte      ;H_Byte = 0
        ADDWF       0x40,W      ;WREG = 0 + 71H = 71H, C = 0
        DAW                     ;WREG = 71H
        BNC         N_1         ;branch if C = 0
        INCF        H_Byte,F    ;
N_1     ADDWF       0x41,W      ;WREG = 71 + 88 = F9H
        DAW                     ;WREG = 59H AND C = 1
        BNC         N_2         ;
        INCF        H_Byte,F    ;C = 1, increment (now H_Byte = 1)
N_2     ADDWF       0x42,W      ;WREG = 59 + 69 = C2 and Carry = 0
        DAW                     ;WREG = 28 and C = 1
        BNC         N_3         ;
        INCF        H_Byte      ;C = 1, increment (now H_Byte = 2)
N_3     ADDWF       0x43,W      ;WREG = 28 + 97 = BFH and C = 0
        DAW                     ;WREG = 25 and C = 1
        BNC         N_4         ;
        INCF        H_Byte,F    ;(now H_Byte = 3)
N_4     MOVWF       L_Byte      ;Now L_Byte = 25H
```

After this code executes, fileReg location 6 = (03), and WREG = 25 because 71 + 88 + 69 + 97 = 325H. We can use the register indirect addressing mode and looping to do this program much more efficiently. Chapter 6 shows how to do that.

Subtraction of unsigned numbers

- In PIC18, we have 4 instructions for subtraction:

1. SUBLW
2. SUBWF
3. SUBWFB
4. SUBFWB

NOTICE: We use the C (carry flag) for the borrow.

- In the subtraction, 2's complement method is used.
- PIC uses adder circuitry to perform the subtraction command.

SUBLW instruction

- SUBLW K (WREG = K - WREG)
- Steps involved in the execution of subtraction, assume C=0 prior to the execution of instruction:

1. Take 2's complement of the subtrahend (WREG operand)
2. Add it to the minuend (K operand)

After these 2 steps, the result is obtained and the flags are set.

Show the steps involved in the following.

```
MOVLW 0x23          ;load 23H into WREG (WREG = 23H)
SUBLW 0x3F          ;WREG = 3F - WREG
```

Solution:

```
   K     = 3F   0011 1111       0011 1111
- WREG  = 23   0010 0011   +   1101 1101  (2's complement)
          1C                  1 0001 1100
                          C = 1, D7 = N = 0 (result is positive)
```

The flags would be set as follows: C = 1, N = 0 (notice that D7 is the negative flag). The programmer must look at the N (or C) flag to determine if the result is positive or negative.

NOTE:

1. If N=0 or C=1, the result is positive.
2. If N=1 or C=0, the result is negative and the destination has the 2's complement of the result.
3. NEGF (negate, 2's complement) instruction can be used to change it.

SUBWF instruction

Write a program to subtract 4C – 6E.

Solution:

```
MYREG  EQU 0x20
       MOVLW 0x4C          ;load WREG (WREG = 4CH)
       MOVWF MYREG         ;MYREG = 4CH
       MOVLW 0x6E          ;WREG = 6EH
       SUBWF MYREG,W       ;WREG = MYREG - WREG. 4C - 6E = DE, N = 1
       BNN   NEXT          ;if N = 0 (C = 1), jump to NEXT target
       NEGF  WREG          ;take 2's complement of WREG
NEXT   MOVWF MYREG         ;save the result in MYREG
```

The following are the steps after the SUBWF instruction:

```
  4C     0100 1100                  0100 1100
 -6E     0110 1110   2's comp =     1001 0010
 -22                                1101 1110
```

After SUBWF, we have N = 1 (or C = 0), and the result is negative, in 2's complement. Then it falls through and NEGF will be executed. The NEGF instruction will take the 2's complement, and we have MYREG = 22H.

SUBWFB instruction

- destination = fileReg - Wreg - Borrow
- This instruction is used for multibyte numbers and will take care of the borrow of the lower byte.
- If C=0 prior to the execution of the SUBWFB instruction, it also subtracts 1 from the result.

Write a program to subtract two 16-bit numbers. The numbers are 2762H – 1296H. Assume fileReg location 6 = (62) and location 7 = (27). Place the difference in fileReg locations 6 and 7; loc 6 should have the lower byte.

Solution:

```
loc 6 = (62)
loc 7 = (27)

MOVLW 0x96          ;load the low byte (WREG = 96H)
SUBWF 0x6,F   ;F = F - W = 62 - 96 = CCH, C = borrow = 0, N = 1
MOVLW 0x12          ;load the high byte (WREG = 12H)
SUBWFB 0x7,F        ;F = F - W - b, sub byte with the borrow
                    ;F = 27 - 12 - 1 = 14H
```

After the SUBWF, loc 6 has = 62H – 96H = CCH and the carry flag is set to 0, indicating there is a borrow (notice, N = 1). Because C = 0, when SUBWFB is executed the fileReg location 7 has = 27H – 12H – 1 = 14H. Therefore, we have 2762H – 1296H = 14CCH.

SUBFWB instruction

- detsination = Wreg - fileReg - borrow
- It is also used for the multibyte numbers and takes care of the borrow of the lower byte.

Difference between SUBWFB and SUBFWB ????

Make any example, and write assembly code