Microprocessor Interfacing & Programming

LECTURE 17

Rotate instructions and Data Serialization

- In PIC18 the rotation instructions are RRCF, RRNCF, RLCF and RLNCF.
- They allow a program to rotate the file register right or left.
- They are widely used in many applications.

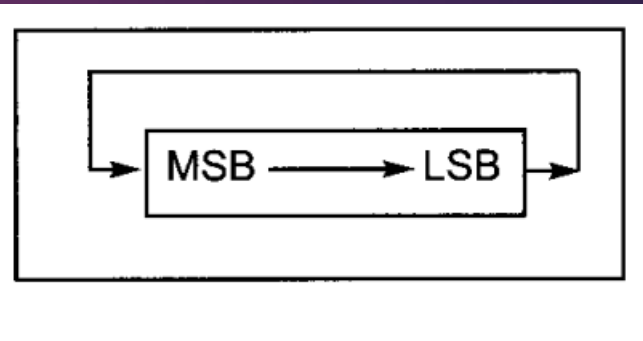
- Two types of rotations:

1. A simple rotation of the bits of the file register.
2. A rotation through the carry.

Rotating the bits of fileReg right

- RRNCF fileReg, d ; rotate fileReg right (no carry)
- In rotate right, the 8 bits of the fileReg are rotated right one bit, and bit D0 exists from the least significant bit and enters into D7 (MSB).
- After the rotation the result can be in fileReg or WREG, depending on the d bit.

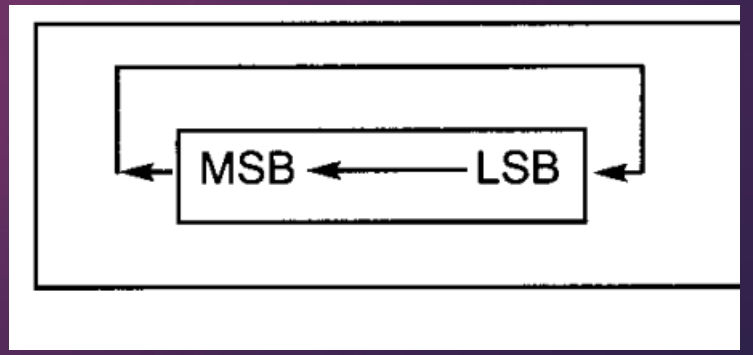
```
MREG EQU 0x20
      MOVLW 0x36              ;WREG = 0011 0110
      MOVWF MYREG
      RRNCF MYREG,F           ;MYREG = 0001 1011
      RRNCF MYREG,F           ;MYREG = 1000 1101
      RRNCF MYREG,F           ;MYREG = 1100 0110
      RRNCF MYREG,F           ;MYREG = 0110 0011
```



Rotating the bits of fileReg left

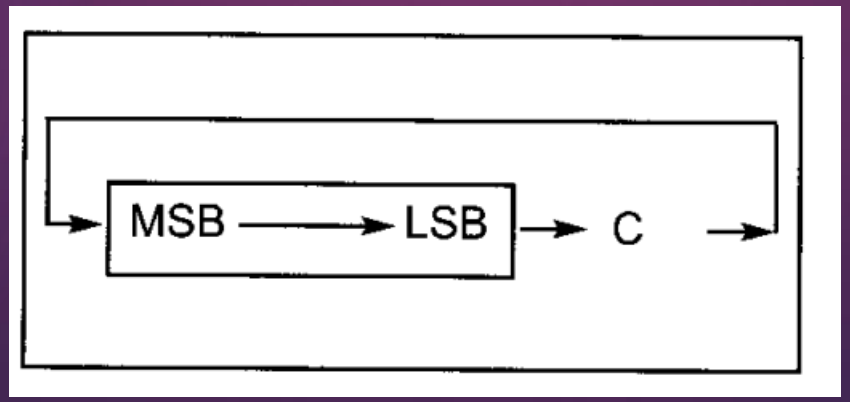
- RLNCF fileReg,d ; rotate fileReg left (no carry)
- In rotate left, the 8 bits of the fileReg are rotated left one bit, and bit D7 exists from the MSB and enters into D0 (LSB).
- Notice, in RRNCF and RLNCF instructions that both the Z and N flags are affected.

```
MREG EQU 0x20
      MOVLW 0x72              ;WREG = 0111 0010
      MOVWF MYREG
      RLNCF MYREG,F           ;MYREG = 1110 0100
      RLNCF MYREG,F           ;MYREG = 1100 1001
```



Rotating right through the carry

- RRCF fileReg,d ; rotate fileReg right through carry
- In RRCF, as bites are rotated from left to right, the carry flag enters the MSB and the LSB exits to the carry flag.
- In other words, in RRCF the C is moved to the MSB, and the LSB is moved to the C.
- In reality, the carry flags acts as if it is part of the register, making it a 9 bit register.

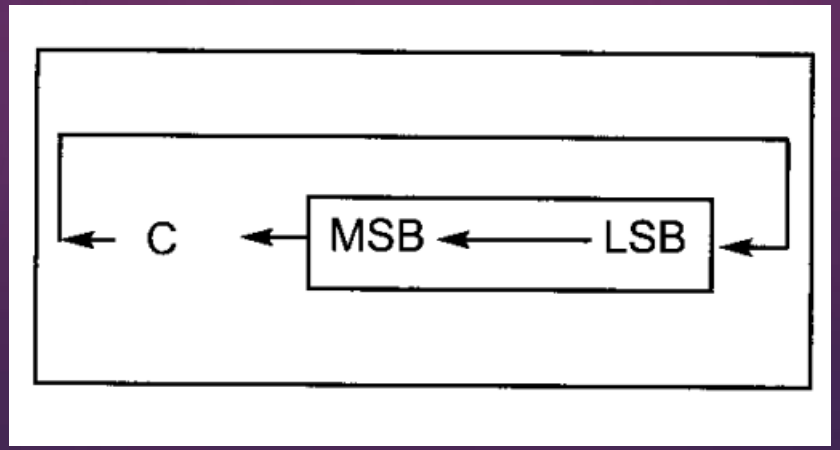


Example

```
MREG EQU 0x20
      BCF    STATUS,C    ;make C = 0 (carry is D0 of status)
      MOVLW  0x26        ;WREG = 0010 0110
      MOVWF  MYREG
      RRCF   MYREG,F     ;MYREG = 0001 0011 C = 0
      RRCF   MYREG,F     ;MYREG = 0000 1001 C = 1
      RRCF   MYREG,F     ;MYREG = 1000 0100 C = 1
```

Rotating left through the carry

- RLCF fileReg,d ; rotate fileReg left through carry
- In RLCF, as bits are shifted from right to left, the carry flag enters the LSB and the MSB exits to the carry flag.
- In other words, in RLCF the C is moved to the LSB, and the MSB is moved to the C.
- Again, the carry flag acts as if it is part of the register, making it a 9 bit register.



Example

```
MREG EQU 0x20
      BSF    STATUS,C    ;make C = 1 (carry is D0 of status)
      MOVLW  0x15        ;WREG = 0001 0101
      MOVWF  MYREG
      RLCF   MYREG,F     ;MYREG = 0010 1011 C = 0
      RLCF   MYREG,F     ;MYREG = 0101 0110 C = 0
      RLCF   MYREG,F     ;MYREG = 1010 1100 C = 0
      RLCF   MYREG,F     ;MYREG = 0101 1000 C = 1
```

Serializing data

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of the microcontroller. There are 2 ways to transfer a byte of data serially:
- Using the serial port. In using this approach, programmers have very limited control over the sequence of data transfer.
- To transfer data one bit at a time and control the sequence of data and spaces between them. In many devices, such as LCD, ADC and ROM, the serial versions are becoming popular as they take less space on a printed circuit board.
- We use rotate instructions in serializing data.

Write a program to transfer value 41H serially (one bit at a time) via pin RB1. Put one high at the start and end of the data. Send the LSB first.

Solution:

```
        RCNT    EQU   0x20          ;fileReg loc for counter
        MYREG   EQU   0x21          ;fileReg loc for rotate

        BCF     TRISB,1             ;make RB1 an output bit
        MOVLW   0x41                ;WREG = 41
        MOVWF   MYREG               ;value to be serialized
        BCF     STATUS,C            ;C = 0
        MOVLW   0x8                 ;counter
        MOVWF   RCNT                ;load the counter
        BSF     PORTB,1             ;RB1 = high
AGAIN   RRCF    MYREG,F             ;rotate right via carry
        BNC     OVER
        BSF     PORTB,1             ;set the carry bit to PB1
        BRA     NEXT
OVER    BCF     PORTB,1
NEXT    DECF    RCNT,F
        BNZ     AGAIN
        BSF     PORTB,1             ;RB1 = high
```

Write a program that finds the number of 1s in a given byte.

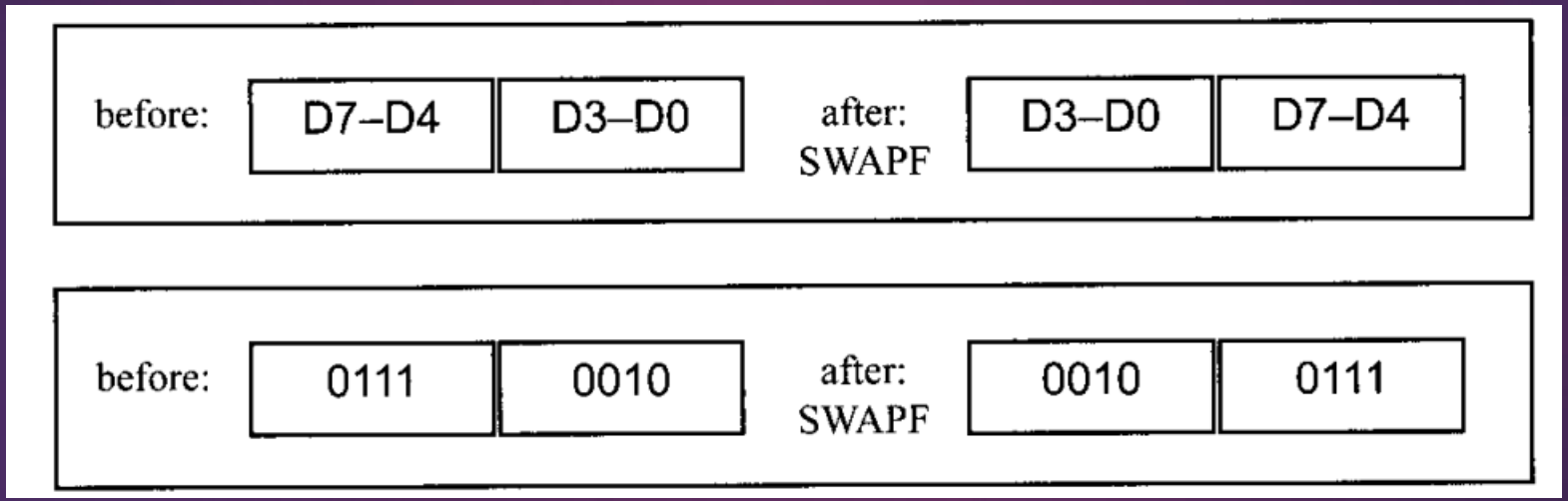
Solution:

```
R1      EQU   0x20   ;fileReg loc for number of 1s
COUNT   EQU   0x21   ;fileReg loc for counter
VALREG  EQU   0x22   ;fileReg loc for the byte

        BCF   STATUS,C      ;C = 0
        CLRF  R1            ;R1 keeps the number of 1s
        MOVLW 0x8           ;counter = 08 to rotate 8 times
        MOVWF COUNT
        MOVLW 0x97          ;find the number of 1s in 97H
        MOVWF VALREG
AGAIN   RLCF  VALREG,F      ;rotate it through the C once
        BNC   NEXT          ;check for C
        INCF  R1,F          ;if C = 1 then add one to R1 reg
NEXT    DECF  COUNT,F
        BNZ AGAIN           ;go through this 8 times
                            ;now loc 0x20 has the number of 1s
```

SWAPF fileReg, d

- It swaps the lower nibble and the higher nibble.



Example

(a) Find the contents of the MYREG register in the following code.
(b) In the absence of a SWAPF instruction, how would you exchange the nibbles? Write a simple program to show the process.

Solution:

(a)

```
MYREG  EQU  0x20
       MOVLW  0x72          ;WREG = 72H
       MOVWF  MYREG         ;MYREG = 72H
       SWAPF  MYREG,F       ;MYREG = 27H
```

(b)

```
MYREG  EQU  0x20
       MOVLW  0x72          ;WREG = 0111 0010
       MOVWF  MYREG         ;MYREG = 0111 0010
       RLNCF  MYREG,F       ;MYREG = 1110 0100
       RLNCF  MYREG,F       ;MYREG = 1100 1001
       RLNCF  MYREG,F       ;MYREG = 1001 0011
       RLNCF  MYREG,F       ;MYREG = 0010 0111
```

Addressing Modes

- The CPU can access data in various ways. The data could be in a register, or in memory, or as an immediate value.
- These various ways of accessing data are called **addressing modes.**
- PIC18 provides **4** distinct addressing modes.

1. Immediate
2. Direct
3. Register indirect
4. Indexed - ROM

Immediate addressing mode

- In this addressing mode, the operand is a literal constant.
- The operand comes immediately after the opcode when the instruction is assembled.
- **Notice** that immediate data is called a literal in the PIC.
- This addressing mode can be used to load information into WREG and selected registers, but not to any file register.
- The immediate addressing mode is also used for arithmetic and logic instructions.

```
MOVLW 0x25          ;load 25H into WREG
SUBLW D'62'         ;subtract WREG from 62
ANDLW B'01000000'   ;AND WREG with 40H
```

Immediate addressing mode

- We can also use immediate addressing mode to perform arithmetic and logic operations on WREG only.

  e.g; ADDLW 0x25 adds value 25 to WREG

- We can use the EQU directive to access immediate data as shown below:

```
COUNT EQU 0x30
...   ..
MOVLW COUNT          ;WREG = 30h
```

Direct addressing mode

- In direct addressing mode, the operand data is in a RAM memory location whose address is known, and this address is given as a part of the instruction.
- "L" in the instruction means literal (immediate), "F" in the instruction signifies the address of the file register location.

```
MOVLW 0x56        ;WREG = 56H (immediate addressing mode)
MOVWF 0x40        ;copy WREG into fileReg RAM location 40H
MOVFF 0x40,0x50   ;copy data from loc 40H to 50H.
```

0 ≤ ffff ffff ≤ FF

A – bank accessed for operation
A = 0, use default access bank
A = 1, use bank pointed to by
BSR (Bank Selector Register)

The difference between "INCF fileReg, W" and "INCF fileReg, F"

In direct addressing mode, when an operation is performed on a file register, we have the option of saving the result in the file register itself or in WREG. This option is a major source of errors in PIC programming and its correct use must be emphasized. The following code increments the contents of register file location 20H using the direct addressing mode, but the destination for the increment result is decided by the W or F parameter:

```
MOVLW 0          ;WREG = 0
MOVWF 0x20       ;loc 0x20 = (0), WREG = 0
INCF  0x20,W     ;loc 0x20 = (0), WREG = 1
INCF  0x20,W     ;loc 0x20 = (0), WREG = 1
INCF  0x20,W     ;loc 0x20 = (0), WREG = 1
INCF  0x20,F     ;loc 0x20 = (1), WREG = 1
INCF  0x20,F     ;loc 0x20 = (2), WREG = 1
INCF  0x20       ;loc 0x20 = (3), WREG = 1
INCF  0x20       ;loc 0x20 = (4), WREG = 1
INCF  0x20,W     ;loc 0x20 = (4), WREG = 5
```

Notice in the above code when the second parameter is not stated, it is assumed to be fileReg (F).

DECFSZ and DECF

Other instructions that need to be examined are DECFSZ and DECF. We can use either one for looping. In "DECFSZ fileReg, d" the fileReg is decremented, and the next instruction is skipped if the fileReg is zero. DECF does not skip the next instruction. Contrast the two codes complementing PORTB 5 times. The following code uses the DECFSZ instruction for looping:

```
        CLRF    TRISB       ;Port B as output
        MOVLW   5           ;WREG = 5
        MOVWF   MYREG       ;counter = 5
        CLRF    PORTB       ;clear Port B
B1      COMF    PORTB       ;complement Port B
        DECFSZ  MYREG,F     ;decrement and skip if MYREG = 0
        GOTO    B1          ;go back since it is not zero
        SETF    PORTB       ;make PB = FFH
```

while the code below uses the BNZ (branch not zero) instruction:

```
        CLRF    TRISB       ;Port B as output
        MOVLW   5           ;WREG = 5
        MOVWF   MYREG       ;counter = 5
        CLRF    PORTB       ;clear Port B
B2      COMF    PORTB       ;complement Port B
        DECF    MYREG,F     ;decrement counter
        BNZ     B2          ;go back if MYREG is not zero
        SETF    PORTB       ;make PB = FFH
```

Notice that if we use "DECF MYREG, W" instead of "DECF MYREG, F" in the BNZ program, we will never get out of the loop because the values of MYREG and WREG will remain 5 and 4 respectively forever.

Symbol	Name	Address
WREG	Working register	FE8H
PORTA	Port A	F80H
PORTB	Port B	F81H
PORTC	Port C	F82H
LATA	Output latch, Port A	F89H
LATB	Output latch, Port B	F8AH
LATC	Output latch, Port C	F8BH
TRISA	Data direction, Port A	F92H
TRISB	Data direction, Port B	F93H
TRISC	Data direction, Port C	F94H
INDF0	Indirect addressing register 0	FEFH
INDF1	Indirect addressing register 1	FE7H
FSR0L	Indirect data memory address pointer 0 low	FE9H
FSR0H	Indirect data memory address pointer 0 high	FEAH
FSR1L	Indirect data memory address pointer 1 low	FE1H
FSR1H	Indirect data memory address pointer 1 high	FE2H
PLUSW0	Indirect indexed address register	FEBH
PREINC0	Preincrement register 0	FECH
POSTDEC0	Post-decrement register 0	FEDH
POSTINC0	Post-increment register 0	FEEH
TBLPTRL	Table pointer, low byte	FF6H
TBLPTRH	Table pointer, high byte	FF7H
TBLPTRU	Table pointer, upper byte	FF8H
TABLAT	Program memory table latch	FF5H
STATUS	Status flag byte	FD8H

SFRs and their addresses

1. The special function registers have addresses between F80H and FFFH. These addresses are below FFFH, because the PIC18 starts assigning SFR addresses at FFFH and goes down until all SFRs supported by that chip are assigned. Not all the members of the PIC18 family have the same peripherals; therefore, the number of locations used for SFRs varies among the PIC18 family.
2. Not all the address space of F80H to FFFH is used by the SFR. The unused locations F80H to FFFH are reserved and must not be used by the PIC18 programmer.

```
MOVWF 0xF81        ;is the same as
MOVWF PORTB        ;which means copy WREG into Port B

CLRF 0xF82         ;is the same as
CLRF PORTC         ;which means clear Port C

BSF 0xFD8,0        ;is the same as
BSF STATUS,C       ;which make C = 1
```

Example

Write code to send 55H to Port B. Include

(a) The register names.
(b) Their addresses.

Solution:

(a)
```
CLRF   TRISB        ;Port B output
MOVLW  0x55         ;WREG = 55H
MOVWF  PORTB        ;Port B = 55H
```

(b) From Table 6-1, TRISB address = F93H and PORTB address = F81H

```
CLRF   0xF93        ;Port B output
MOVLW  0x55H        ;WREG = 55H
MOVWF  0xF81        ;Port B = 55H
```

Direct addressing mode in PIC18

Regarding direct addressing mode in the PIC18, notice the following points:

1. If you examine the .lst file for an Assembly language program, you will see that the SFR register names are replaced with their addresses as listed in Table 6-1.
2. The WREG register is one of the SFR registers and has address FE8H.
3. The direct addressing mode is also called *register direct* to contrast it with the register indirect addressing mode discussed in the next section.

Register indirect addressing mode

- In the register indirect addressing mode, a register is used as a pointer to the data RAM location.
- In PIC18, three registers are used for this purpose:

1. FSR0
2. FSR1
3. FSR2

- FSR stands for file select register.
- FSR is a 12 bit register allowing access to the entire 4096 bytes of data RAM space.
- We use LFSR (load FSR) to load the RAM address.

- When FSRx are used as pointers, they must be loaded first with the RAM addresses as shown below:

```
LFSR 0, 0x30      ;load FSR0 with 0x30
LFSR 1, 0x40      ;load FSR1 with 0x40
LFSR 2, 0x6F      ;load FSR2 with 0x6F
```

FSRx are 12 bit registers so they cannot fit into the SFR address space. So, they are split into 8 bit size, called FSRxL and FSRxH.

Another register associated with the register indirect addressing mose is the INDF (indirect register).

Each of FSRx registers has an INDf register, and these are INDF0, INDF1 and INDF2. When we move data into INDFx, we are moving data into a RAM location pointed to by the FSR.

```
LFSR   0, 0x30      ;FSR0 = 30H RAM location pointer
MOVWF  INDF0        ;copy contents of WREG into RAM
                    ;location whose address is held by
                    ;12-bit FSR0 register
```

1. One of the advantages of register indirect addressing mode is that it makes accessing data dynamic rather than static, as with direct addressing mode.

2. Allows access to various memory locations easily

3. Useful for arrays, tables and loops

4. Dynamic memory access. The address can be computed or changed during program execution, which is not possible with direct addressing mode.

```
LFSR   0, 0x20     ; Load address 0x20 into FSR0
MOVF   INDF0, W    ; Access data at address 0x20
INCF   FSR0L, F    ; Increment the address stored in FSR0
MOVF   INDF0, W    ; Now access data at address 0x21
```

Write a program to copy the value 55H into RAM memory locations 40H to 45H using
(a) Direct addressing mode.
(b) Register indirect addressing mode without a loop.
(c) A loop.

Solution:

(a)

```
        MOVLW   0x55        ;load WREG with value 55H
        MOVWF   0x40        ;copy WREG to RAM location 40H
        MOVWF   0x41        ;copy WREG to RAM location 41H
        MOVWF   0x42        ;copy WREG to RAM location 42H
        MOVWF   0x43        ;copy WREG to RAM location 43H
        MOVWF   0x44        ;copy WREG to RAM location 44H
```

(b)

```
        MOVLW   55H         ;load with value 55H
        LFSR    0,0x40      ;load the pointer. FSR0 = 40H
        MOVWF   INDF0       ;copy W to RAM loc FSR0 points to
        INCF    FSR0L,F     ;increment pointer. Now FSR0 = 41H
        MOVWF   INDF0       ;copy W to RAM loc FSR0 points to
        INCF    FSR0L,F     ;increment pointer. Now FSR0 = 42H
        MOVWF   INDF0       ;copy W to RAM loc FSR0 points to
        INCF    FSR0L,F     ;increment pointer. Now FSR0 = 43H
        MOVWF   INDF0       ;copy W to RAM loc FSR0 points to
        INCF    FSR0L,F     ;increment pointer. Now FSR0 = 44H
        MOVWF   INDF0       ;copy W to RAM loc FSR0 points to
```

(c)

```
        COUNT   EQU 0x10    ;location 10H for counter
        MOVLW   0x5         ;WREG = 5
        MOVWF   COUNT       ;load the counter, Count = 5
        LFSR    0,0x40      ;load pointer. FSR0 = 40H, RAM address
        MOVLW   0x55        ;WREG = 55h value to be copied
B1      MOVWF   INDF0       ;copy WREG to RAM loc SFR0 points to
        INCF    FSR0L,F     ;increment FSR0 pointer
        DECF    COUNT,F     ;decrement the counter
        BNZ     B1          ;loop until counter = zero
```

Use the MPLAB simulator to examine RAM contents after the above program is run.

40 = (55)
41 = (55)
42 = (55)
43 = (55)
44 = (55)

Assume that RAM locations 30–34H have a string of ASCII data, as shown below. Write a program to get each character and send it to Port B one byte at a time. Show the program using:

(a) Direct addressing mode.
(b) Register indirect addressing mode.

30 = ('H')
31 = ('E')
32 = ('L')
33 = ('L')
34 = ('O')

Solution:

(a) Using direct addressing mode

```
        CLRF   TRISB                ;make Port B an output
        MOVFF  0x30,  PORTB         ;copy contents of loc 0x30 to PB
        MOVFF  0x31,  PORTB
        MOVFF  0x32,  PORTB
        MOVFF  0x33,  PORTB
        MOVFF  0x34,  PORTB
```

(b) Using register indirect mode

```
        COUNTREG EQU 0x20           ;fileReg loc for counter
        CNTVAL EQU 5                ;counter value
        CLRF   TRISB                ;make Port B an output (TRSI = 0 = out)
        MOVLW  CNTVAL               ;WREG = 5
        MOVWF  COUNTREG             ;load the counter, Count = 5
        LFSR   2,0x30               ;load pointer. FSR2 = 30H, RAM address
B3      MOVF   INDF2,W              ;copy RAM loc FSR2 points at to WREG
        MOVWF  PORTB                ;copy WREG to PORTB
        INCF   FSR2L                ;increment FSR2 to point at next loc
        DECF   COUNTREG,F           ;decrement counter
        BNZ    B3                   ;loop until counter = zero
```

When simulating the above program on the MPLAB, make sure that RAM locations 30H–34H have the message "HELLO". Notice that "`MOVF INDF2, W`" moves data from INDF2 into WREG.

Auto increment option for FSR

Because the FSR is a 12-bit register, it can go from 000 to FFFH, which covers the entire 4K RAM space of the PIC18. Using the "`INCF FSR0L, F`" instruction to increment the pointer can cause a problem when an address such as 5FFH is incremented. The instruction "`INCF FSR0L, F`" will not propagate the carry into the FSR1H register. The PIC18 gives us the options of auto-increment and auto-decrement for FSRn to overcome this problem. The syntax used for such cases for the CLRF instruction is shown in Table 6-2.

Instruction	Function
CLRF INDFn	After clearing fileReg pointed to by FSRn, the FSRn stays the same.
CLRF POSTINCn	After clearing fileReg pointed to by FSRn, the FSRn is incremented.
CLRF PREINCn	The FSRn is incremented, then fileReg pointed to by FSRn is cleared
CLRF POSTDECn	After clearing fileReg pointed to by FSRn, the FSRn is decremented.
CLRF PLUSWn	Clears fileReg pointed to by (FSRn +WREG), FSRn & W unchanged

Note: This table shows the syntax for the CLRF instruction, it works for all such instructions. The auto-decrement or auto-increment affects the entire 12 bits of the FSRn and has no effect on status register. This means that FSR0 going from FFF to 000 will not raise any flag. The option of PLUSWn is widely used for a RAM-based look-up table. See Section

Write a program to clear 16 RAM locations starting at RAM address 60H.
Use the following:
(a) INCF FSRnL
(b) Auto-increment

Solution:

```
(a)
COUNTREG EQU 0x10          ;fileReg loc for counter
CNTVAL EQU D'16'           ;counter value
       MOVLW   CNTVAL      ;WREG = 16
       MOVWF   COUNTREG    ;load the counter, Count = 16
       LFSR    1,0x60      ;load pointer. FSR1 = 40H, RAM address
B2     CLRF    INDF1       ;clear RAM loc FSR1 points to
       INCF    FSR1L,F     ;increment FSR1L, point to next loc
       DECF    COUNTREG,F  ;decrement counter
       BNZ     B2          ;loop until counter = zero

(b)
COUNTREG EQU 0x10          ;fileReg loc for counter
CNTVAL EQU D'16'           ;counter value
       MOVLW   CNTVAL      ;WREG = 16
       MOVWF   COUNTREG    ;load the counter, Count = 16
       LFSR    1,0x60      ;load pointer. SFR0 = 40H, RAM address
B3     CLRF    POSTINC1    ;clear RAM, increment FSR1 pointer
       DECF    COUNTREG,F  ;decrement counter
       BNZ     B3          ;loop until counter = zero
```

Assume that RAM locations 40–43H have the following hex data. Write a program to add them together and place the result in locations 0x06 and 0x07.

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

Solution:

```
        COUNTREG EQU 0x20          ;fileReg loc for counter
        L_BYTE EQU 0x06            ;fileReg loc for L_Byte
        H_BYTE EQU 0x07            ;fileReg loc for L_Byte
        CNTVAL EQU 4               ;counter value
        MOVLW  CNTVAL              ;WREG = 4
        MOVWF  COUNTREG            ;load the counter
        LFSR   0,0x40              ;load pointer. FSR0 = 40H, RAM address
        CLRF   WREG                ;clear WREG
        CLRF   H_BYTE              ;clear H_BYTE
B5      ADDWF  POSTINC0, W         ;add RAM to WREG and increment FSR0
        BNC    OVER                ;if C = 0, go to next
        INCF   H_BYTE,F            ;C = 1, add 1 to high byte
OVER    DECF   COUNTREG,F          ;decrement counter
        BNZ    B5                  ;loop until counter = zero
        MOVWF  L_BYTE
```

The above is a register indirect version of Example 5-2 in Chapter 5 with a loop. Contrast them to see the difference.