EC-310 Microprocessor and Microcontroller Based Design

Chapter - 5

Logic Instructions

Nazar Abbas Saqib
nazar.abbas@ceme.nust.edu.pk

Outline

1. Logical Operations
2. Compare Instructions

AND Instructions

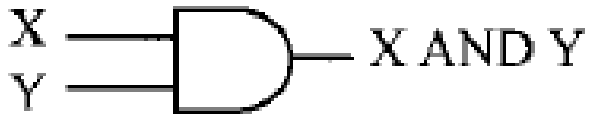
- Performs logical AND operation on two input operands
- ANDLW
- One operand is in WREG and other is a literal.
- The result is placed in WREG.
- Format:

  ANDLW K

- The AND instruction affects the Z and N flags.

Logical AND Function

Inputs		Output
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



AND Instructions

- ANDWF
- Format:

ANDWF fileReg, d

- One operand is placed in WREG and other is placed in fileReg.
- Result destination is based on value of 'd'.
  - For d = 0, the destination is WREG
  - For d = 1, the destination is fileReg
- AND instruction is mostly used for masking.

AND Instructions

Example 5-17

Show the results of the following.

```
MOVLW  0x35          ;WREG = 35H
ANDLW  0x0F          ;W = W AND 0FH (now W = 05)
```

Solution:

```
35H   0 0 1 1 0 1 0 1
0FH   0 0 0 0 1 1 1 1
05H   0 0 0 0 0 1 0 1    ;35H AND 0FH = 05H, Z = 0, N = 0
```

OR Instructions

- Performs logical OR operation on two input operands
- IORLW
- One operand is in WREG and other is a literal.
- The result is placed in WREG.
- Format:

  IORLW K

- The OR instruction affects the Z and N flags.

Logical OR Function

Inputs		Output
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

OR Instructions

- IORWF
- Format:

IORWF fileReg, d

- One operand is placed in WREG and other is placed in fileReg.
- Result destination is based on value of 'd'.
  - For d = 0, the destination is WREG
  - For d = 1, the destination is fileReg
- OR instruction is mostly used for setting bits.

Example 5-18

(a) Show the results of the following:

```
MOVLW 0x04                ;WREG = 04
IORLW 0x30                ;now WREG = 34H
```

(b) Assume that Port B bit RB2 is used to control an outdoor light, and bit RB5 to control a light inside a building. Show how to turn "on" the outdoor light and turn "off" the inside one.

Solution:

(a)

```
04H      0000 0100
30H      0011 0000
34H      0011 0100      04 OR 30 = 34H, Z = 0 and N = 0
```

(b)

```
BCF   TRISB,2           ;make RB2 an output
BCF   TRISB,5           ;make RB5 an output
MOVLW B'00000100'       ;D2 = 1
IORWF PORTB,F           ;make RB2 = 1 only
MOVLW B'11011111'       ;D5 = 0
ANDWF PORTB,F           ;mask RB5 = 0 only
```

Of course, the above method is unnecessary in PIC, since we can manipulate individual bits using bit-oriented operations. This is shown in Section 6.4.

XOR Instructions

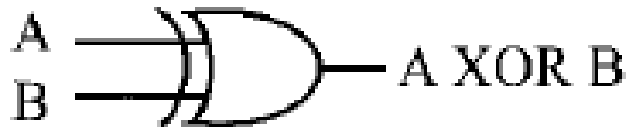
- Performs logical Exclusive OR operation on two input operands
- XORLW
- One operand is in WREG and other is a literal.
- The result is placed in WREG.
- Format:

  XORLW K

- The XOR instruction affects the Z and N flags.

Logical XOR Function

Inputs		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



XOR Instructions

- XORWF
- Format:

XORWF fileReg, d

- One operand is placed in WREG and other is placed in fileReg.
- Result destination is based on value of 'd'.
  - For d = 0, the destination is WREG
  - For d = 1, the destination is fileReg
- XOR instruction is mostly used for toggling bits.

```
MOVLW  0xFF          ;WREG = FFH
XORWF  PORTC,F       ;EX-OR PORTC with 1111 1111 will
                     ;change all the bits of Port C to
                     ;opposite
```

Example 5-19

Show the results of the following:

```
MOVLW 0x54
XORLW 0x78
```

Solution:

```
54H      0 1 0 1 0 1 0 0
78H      0 1 1 1 1 0 0 0
2CH      0 0 1 0 1 1 0 0   54H XOR 78H = 2CH, Z = 0, N = 0
```

Example 5-20

The EX-OR instruction can be used to test the contents of a register by EX-ORing it with a known value. In the following code, we show how EX-ORing value 45H with itself will raise the Z flag:

```
OVER   MOVF   PORTB,W      ;get a byte from PORTB into WREG
       XORLW  0x45
       BNZ    OVER        ;branch if not zero
```

Solution:

```
45H          01000101
45H          01000101
00           00000000
```

EX-ORing a number with itself sets it to zero with Z = 1. We can use the BNZ instruction to make the decision. EX-ORing with any other number will result in a non-zero value.

Example 5-21

Read and test PORTB to see whether it has value 45H. If it does, send 99H to PORTC; otherwise, it stays cleared.

Solution:

```
        CLRF   TRISC        ;Port C = output
        CLRF   PORTC        ;Port C = 00
        SETF   TRISB        ;Port B = input
        MOVLW  0x45
        XORWF  PORTB,W      ;EX-OR with 0x45, Z = 1 if yes
        BNZ    EXIT         ;branch if PORTB has value other than 0
        MOVLW  0x99
        MOVWF  PORTC        ;Port C = 99h
EXIT:...
```

COMF

- Complements the contents of *fileReg*.
- Format:

COMF fileReg, F

Logical Inverter

Input	Output
X	NOT X
0	1
1	0


```
CLRF   TRISB          ;Port B = Output
MOVLW  0x55
MOVWF  PORTB
COMF   PORTB,F        ;now PORTB = AAH
```

NEGF

- **Format:**

NEGF fileReg

Example 5-22

Find the 2's complement of the value 85H. Note that 85H is –123.

Solution:

```
MYREG EQU 0x10
      MOVLW 0x85          85H = 1000 0101
      MOVWF MYREG         1'S = 0111 1010
      NEGF  MYREG                    + 1
                    2's comp    0111 1011 = 7BH
```

COMF VS NEGF

- What is the difference between COMF and NEGF?

COMF computes the 1's Complement

NEGF computes the 2's Complement

Compare Instructions

- The PIC18 has three instructions for compare operations.
- These instructions compare the value in the fileReg with the contents of the WREG register and make decisions based on whether fileReg is greater than, equal to, or less than WREG.
- Compare instruction is a subtraction, except that the values of the operands do not change.
- The compare instructions don't even affect the flag bits.

Compare Instructions

Table 5-2: PIC18 Compare Instructions

CPFSGT	Compare FileReg with WREG, skip if greater than	FileReg > WREG
CPFSEQ	Compare FileReg with WREG, skip if equal	FileReg = WREG
CPFSLT	Compare fileReg with WREG, skip if less than	FileReg < WREG

Note: These instructions have no effect on the flag bits of the status register. Also the values in fileReg and WREG remain unchanged.

Compare Instructions

- **CPFSGT**
- The CPFSGT compares a fileReg with WREG and skips the next instruction if fileReg is greater than WREG.

$$F > W$$

- No flag bit is changed.
- No operand is changed.

Compare Instructions

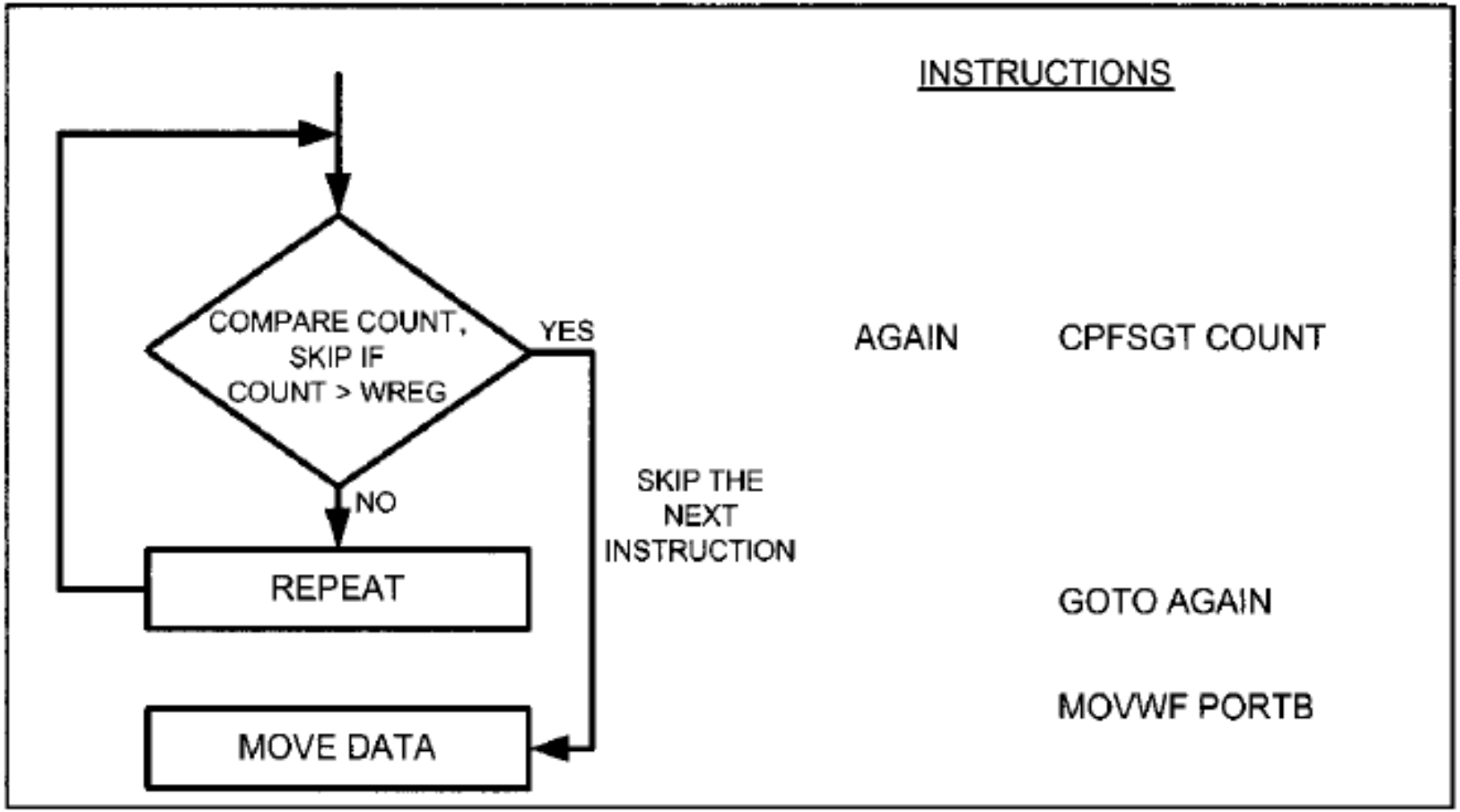


Figure 5-3. Flowchart for CPFSGT

Compare Instructions

Example 5-23

Write a program to find the greater of the two values 27 and 54, and place it in file register location 0x20.

Solution:

```
VAL_1  EQU   D'27'
VAL_2  EQU   D'54'
GREG   EQU   0x20

MOVLW  VAL_1       ;WREG = 27
MOVWF  GREG        ;GREG = 27
MOVLW  VAL_2       ;WREG = 54
CPFSGT GREG        ;skip if GREG > WREG
MOVWF  GREG        ;place the greater in GREG
```

Compare *Instructions*

- **CPFSEQ**
- The CPFSEQ compares a fileReg with WREG and skips the next instruction if fileReg is equal to WREG.

$$F = W$$

- No flag bit is changed.
- No operand is changed.

Compare Instructions

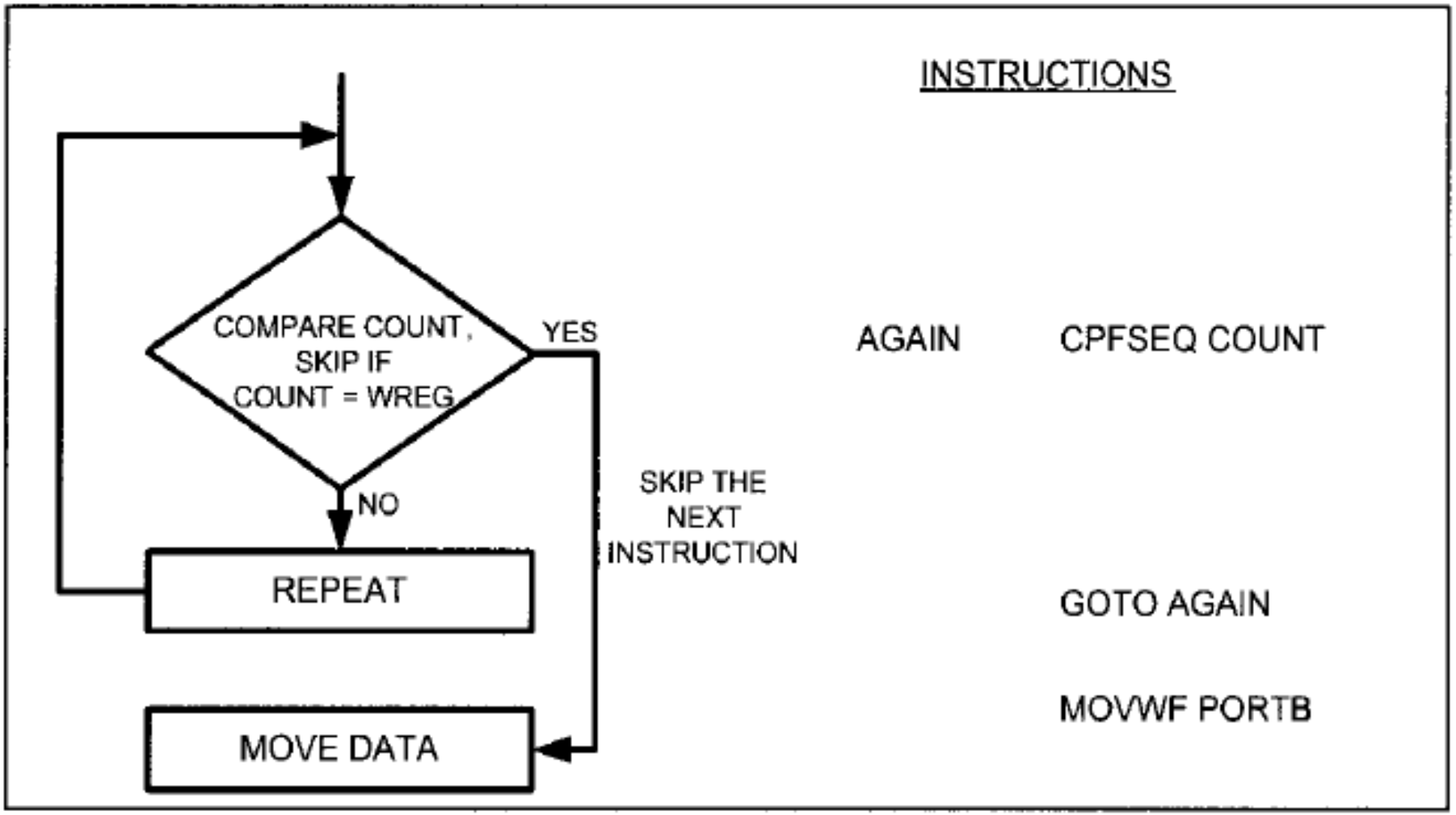


Figure 5-4. Flowchart for CPFSEQ

Example 5-27

Write code to determine if data on PORTB contains the value 99H. If so, write letter 'Y' to PORTC; otherwise, make PORTC = 'N'.

Solution:

```
        CLRF    TRISC       ;PORTC = output
        MOVLW   A'N'        ;WREG = 'N' (ASCII)
        MOVWF   PORTC       ;PORTC = 'N'
        SETF    TRISB       ;PORTB = input
        MOVLW   0x99        ;WREG = 99H
        CPFSEQ  PORTB       ;skip BRA instruction if PORTB = WREG
        BRA     OVER
        MOVLW   'Y'
        MOVWF   PORTC       ;PORTC = 'Y'
OVER    .....
```

Compare Instructions

- **CPFSLT**
- The CPFSLT compares a fileReg with WREG and skips the next instruction if fileReg is less than WREG.

$$F < W$$

- No flag bit is changed.
- No operand is changed.

Compare Instructions

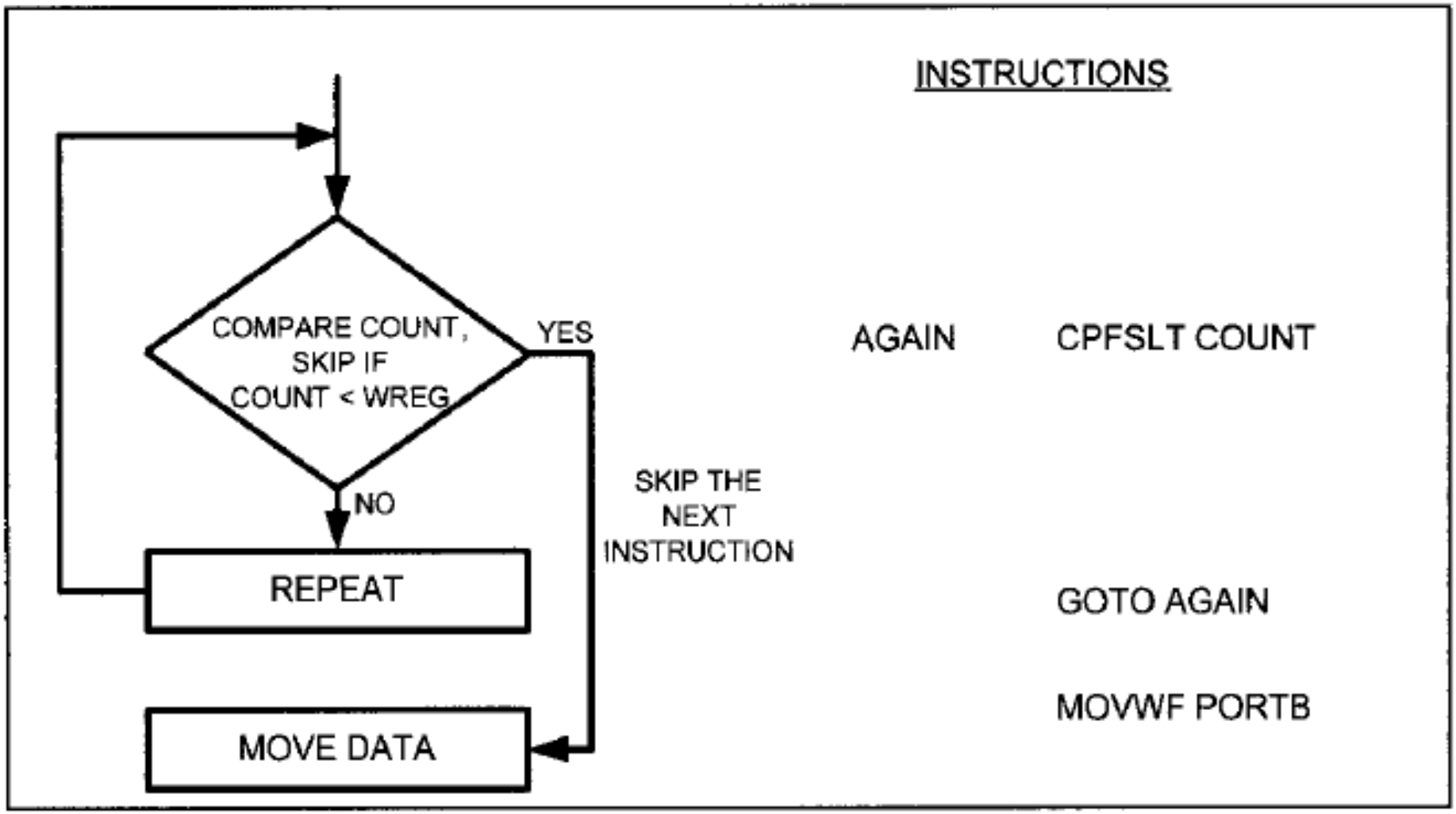


Figure 5-5. Flowchart for CPFSLT

Compare Instructions

Example 5-25

Write a program to find the smaller of the two values 27 and 54, and place it in file register location 0x20.

Solution:

```
VAL_1  EQU   D'27'
VAL_2  EQU   D'54'
LREG   EQU   0x20   ;location for smaller of two

MOVLW  VAL_1        ;WREG = 27
MOVWF  LREG         ;LREG = 27
MOVLW  VAL_2        ;WREG = 54
CPFSLT LREG         ;skip if LREG < WREG
MOVWF  LREG         ;place the smaller value in LREG
```

Example 5-26

Assume that Port D is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75	then WREG = 75
If T > 75	then GREG = T
If T < 75	then LREG = T

Solution:

```
LREG  EQU  0x20
GREG  EQU  0x21
      SETF    TRISD           ;PORTD = input
      MOVLW   D'75'           ;WREG = 75 decimal
      CPFSGT  PORTD           ;skip BRA instruction if PORTD > 75
      BRA     LEQ
      MOVFF   PORTD, GREG
      BRA     OVER
LEQ   CPFSLT  PORTD           ;skip if PORTD < 75
      BRA     OVER
      MOVFF   PORTD, LREG
OVER  .....                   ;it must be equal, WREG = 75
```

Rotate Instruction and Data Serialization

Rotate Instructions and Data Serialization

- In PIC18 the rotation instructions RRCF, RRNCF, RLCF and RLNCF are designed specifically for performing a bitwise rotation operation.

- These instructions allow us to rotate the file register left or right.

- There are two type of rotations:
  - Simple Rotation.
  - Rotation through carry.

Rotating the bits of fileReg right or left

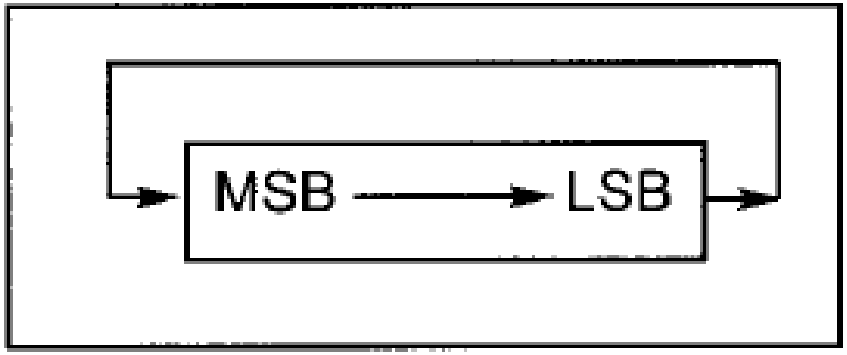
- RRNCF
- Rotate fileReg right with no carry.
- Format:

RRNCF fileReg, d

- In rotate right, the 8-bits of the fileReg are rotated right one bit.
- Bit D0 exits from the LSB and enters into D7 MSB.
- After the rotation the result can be fileReg or WREG, depending on the 'd' bit.
  - For d = 0, destination is WREG.
  - For d = 1, destination is fileReg.

Rotating the bits of fileReg right or left

- **RRNCF**




```
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 right or left

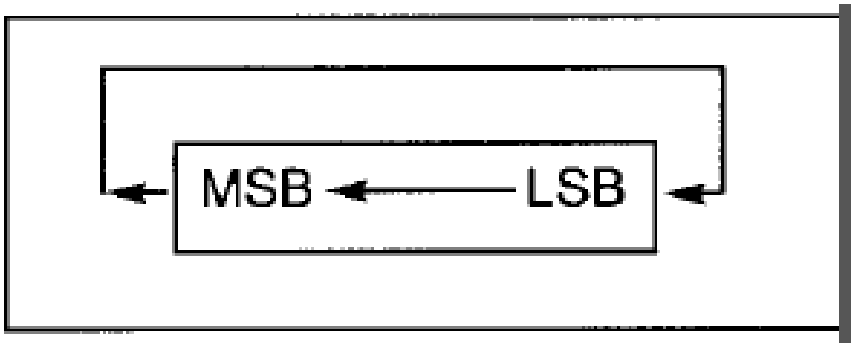
- **RLNCF**
- Rotate fileReg left with no carry.
- Format:

RLNCF fileReg, d

- In rotate right, the 8-bits of the fileReg are rotated left one bit.
- Bit D7 exits from the MSB and enters into D0 LSB.
- After the rotation the result can be fileReg or WREG, depending on the 'd' bit.
  - For d = 0, destination is WREG.
  - For d = 1, destination is fileReg.

Rotating the bits of fileReg right or left

- **RLNCF**




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

Rotating the bits of fileReg right or left through carry

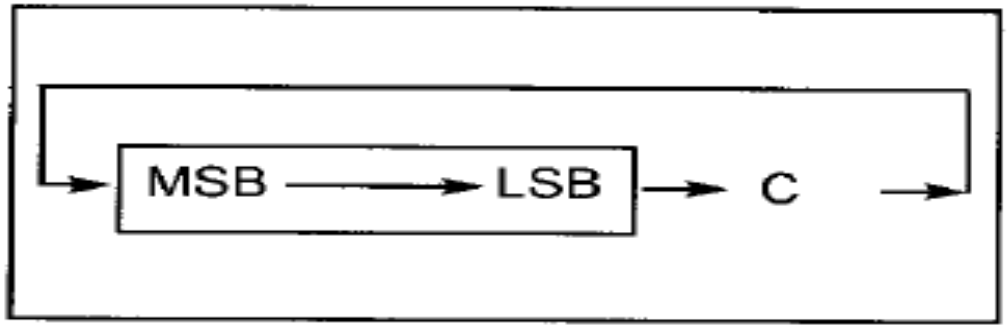
- **RRCF**
- Rotate fileReg right through carry.
- Format:

RRCF fileReg, d

- In rotate right through carry, the 8-bits of the fileReg are rotated right one bit through carry.
- Carry bit goes into the MSB and LSB exits into the carry flag.
- After the rotation the result can be fileReg or WREG, depending on the 'd' bit.
  - For d = 0, destination is WREG.
  - For d = 1, destination is fileReg.

Rotating the bits of fileReg right or left through carry

- **RRCF**




```
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 the bits of fileReg right or left through carry

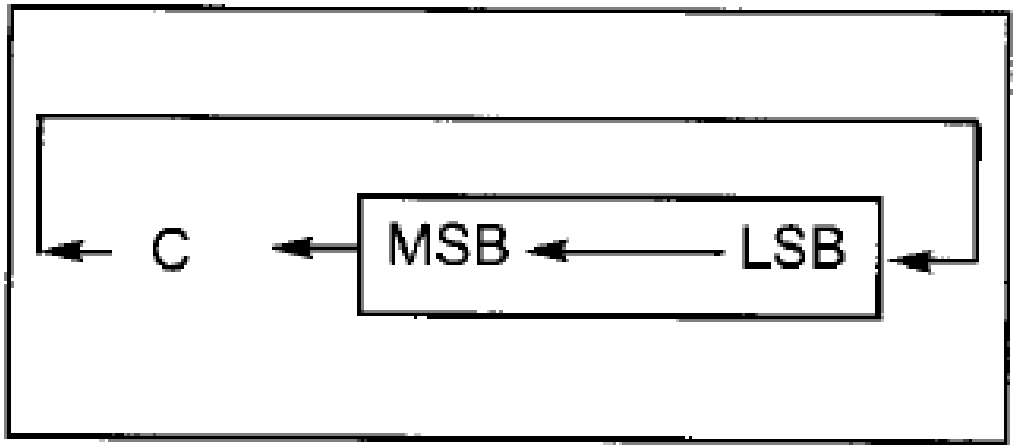
- **RLCF**
- Rotate fileReg left through carry.
- Format:

RLCF fileReg, d

- In rotate left through carry, the 8-bits of the fileReg are left right one bit through carry.
- Carry bit goes into the LSB and MSB exits into the carry flag.
- After the rotation the result can be fileReg or WREG, depending on the 'd' bit.
  - For d = 0, destination is WREG.
  - For d = 1, destination is fileReg.

Rotating the bits of fileReg right or left through carry

- **RLCF**




```
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 microcontroller.

- There are two ways of transfer a byte of data serially:
  - Using serial port (Discussed in chapter 10).
  - The second method of serializing data is to transfer one bit at a time.

- Rotate instruction can be used for serializing data.

Example 5-28

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
```

Example 5-29

Write a program to bring in a byte of data serially (one bit at a time) via pin RC7 and save it in file register location 0x21. The byte comes in with the LSB first.

Solution:

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

       BSF    TRISC,7             ;make RC7 an input bit
       MOVLW  0x8                 ;counter
       MOVWF  RCNT                ;load the counter
AGAIN  BTFSC  PORTC,7             ;skip if RC7 = 0
       BSF    STATUS,C            ;carry = 1
       BTFSS  PORTC,7             ;skip if RC7 = 1
       BCF    STATUS,C            ;otherwise carry = 0
       RRCF   MYREG,F             ;rotate right carry into MYREG
       DECF   RCNT,F              ;decrement the counter
       BNZ    AGAIN               ;repeat until RCNT = 0
                                  ;now loc 21H has the byte
```

Example 5-30

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

Solution:

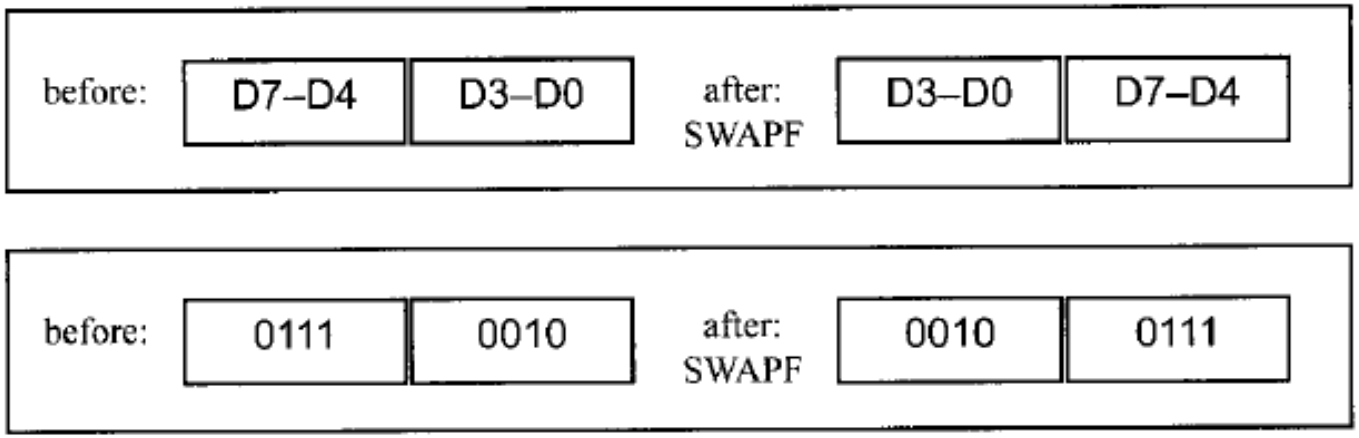
```
R1      EQU   0x20   ;fileReg loc for number of 1s
COUNT   EQU   0x21   ;fileReg loc for counter
VALREG  EQU   0x22   ;fileReg loc for the byte

        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
```

SWAP Instruction

- SWAPF works on fileReg.
- It swaps the contents of lower and higher nibble.
- This means, lower 4-bits are placed into higher 4-bits and higher 4-bits are placed in lower 4-bits.
- Format:

SWAPF fileReg, d



SWAP Instruction

Example 5-31

(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
```

BCD and ASCII Conversions

- **Packed BCD to ASCII Conversion**
- Steps:
  - Convert packed BCD to unpacked BCD.
  - Add 30H to both extractions.

Packed BCD	Unpacked BCD	ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

BCD and ASCII Conversions

Example 5-32

Assume that register WREG has packed BCD. Write a program to convert packed BCD to two ASCII numbers and place them in file register locations 6 and 7.

Solution:

```
BCD_VAL  EQU 0x29
L_ASC    EQU 0x06   ;set aside file register location
H_ASC    EQU 0x07   ;set aside file register location

        MOVLW BCD_VAL      ;WREG = 29H, packed BCD
        ANDLW 0x0F         ;mask the upper nibble (W = 09)
        IORLW 0x30         ;make it an ASCII, W = 39H ('9')
        MOVWF L_ASC        ;save it (L_ASC = 39H ASCII char)
        MOVLW BCD_VAL      ;W = 29H get BCD data once more
        ANDLW 0xF0         ;mask the lower nibble (W = 20H)
        SWAPF WREG,W       ;swap nibbles (WREG = 02H)
        IORLW 0x30         ;make it an ASCII, W = 32H ('2')
        MOVWF H_ASC        ;save it (H_ASC = 32H ASCII char)
```

BCD and ASCII Conversions

- **ASCII to Packed BCD Conversion**
- Steps:
  - Convert the ASCII into BCD by subtracting 30H.
  - Now combine the unpacked BCD into a packed BCD.

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 which is 47H

BCD and ASCII Conversions

```
MYBCD EQU 0x20      ;set aside location in file register

      MOVLW A'4'          ;WREG = 34H, hex for ASCII char 4
      ANDLW 0x0F          ;mask upper nibble (WREG = 04)
      MOVWF MYBCD         ;save it in MYBCD loc
      SWAPF MYBCD,F       ;MYBCD = 40H
      MOVLW A'7'          ;WREG = 37H, hex for ASCII char 7
      ANDLW 0x0F          ;mask upper nibble (WREG = 07)
      IORWF MYBCD,F       ;MYBCD = 47H, a packed BCD
```