EC-310 Microprocessor and Microcontroller Based Design

Chapter - 5

Arithmetic (Unsigned)

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

Outline

1. *Arithmetic Instructions*
    - *Unsigned*

Arithmetic Instructions

- Addition of unsigned numbers:
- ADDLW
- To add two unsigned numbers in PIC, the WREG must be involved.

```
ADDLW    k    ;  WREG = WREG + K
```

- The instruction could change any of the C, DC, Z, N or OV bits of status register depending upon the operands.

Arithmetic Instructions

Example 5-1

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.

Arithmetic Instructions

- Addition of unsigned numbers:
- ADDWF
- ADDWF adds contents of WREG and fileReg register and places the result in location specified by the programmer.

ADDWF fileReg, d

For d = w WREG = WREG + fileReg

For d = f fileReg = WREG + fileReg

- Memory to memory arithmetic instructions are not allowed in PIC, hence WREG must be involved.
- Incase of adding multiple bytes, each time carry bit must looked after.

Solution:

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

Sum = 7D+EB+C5+58

```
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.

Arithmetic Instructions

- ADDWFC
- In addition of 16-bit numbers, we have to take care for the carry propagated from lower byte to higher byte.
- In PIC18, ADDWFC is used to cater for this issue of carry propagation from lower byte to higher byte.

ADDWFC fileReg, d

For d = w WREG = WREG + fileReg + C

For d = f fileReg = WREG + fileReg + C

- The lower bytes are added using instruction ADDWF and the higher bytes are added using instruction ADDWFC.

Example 5-3

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)
```

3CE7 H
3B8D H +

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

Binary Coded Decimals

- Binary Representation from 0-9 is called BCD (Binary Coded Decimal).
- Used since decimal numbers are used in day to day life.
- There are further two types of BCD numbers based on their placement in memory.
  - Packed BCD
  - Unpacked BCD.

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

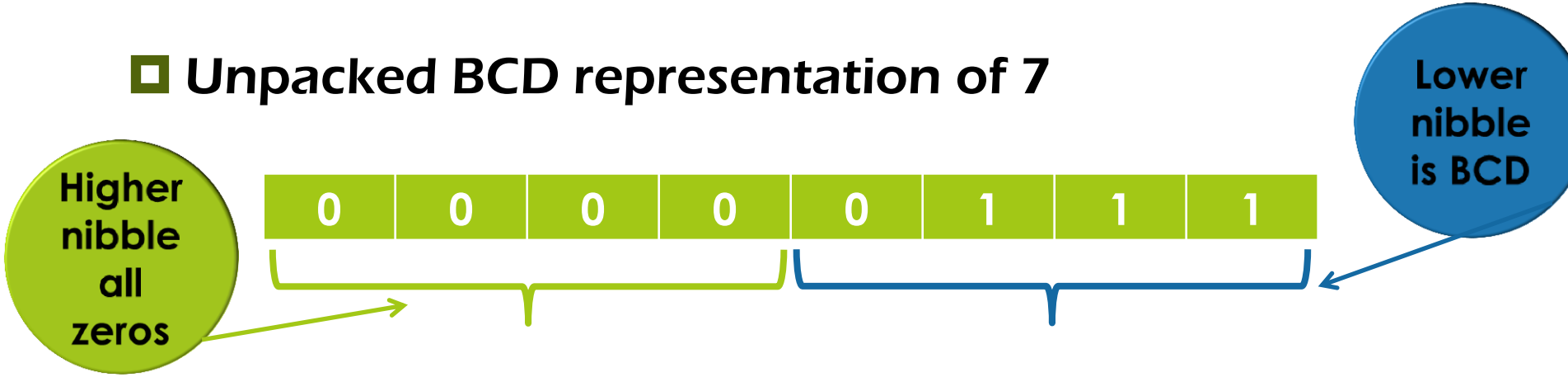
Binary Coded Decimals

- **Unpacked BCD:**

- In unpacked BCD representation of BCD numbers, the lower 4-bits represent the number and higher 4-bits are all zeros.

- 8-bits are utilized to represent a single BCD number.

- Unpacked BCD representation of 7



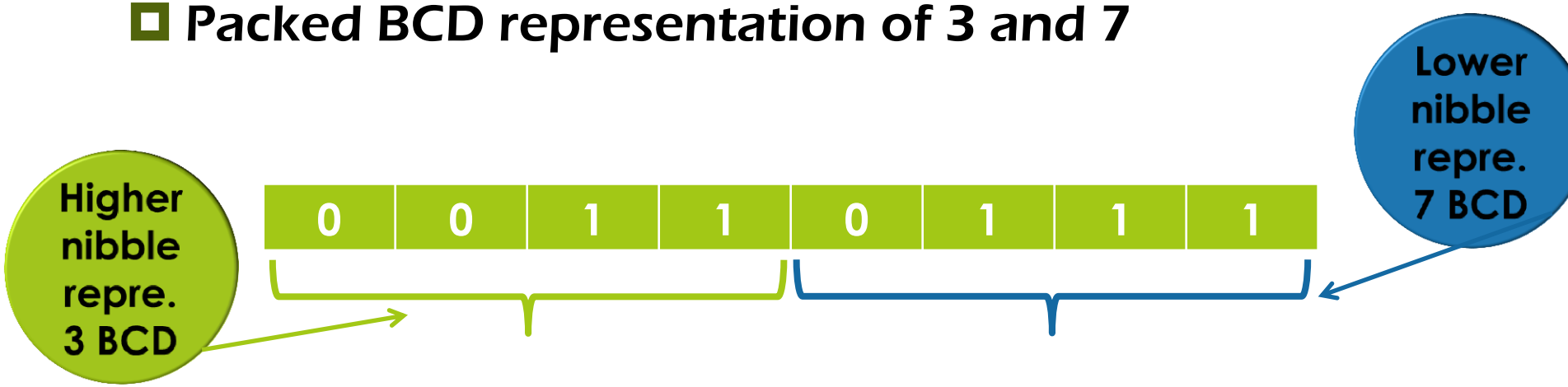
Binary Coded Decimals

- **Packed BCD:**

- In packed BCD representation of BCD numbers, the single byte represents two BCD numbers

- 8-bits are utilized to represent a two BCD number.

- Packed BCD representation of 3 and 7



Problem of adding 2 BCD Numbers

- Adding two BCD numbers sometime result in a number which itself is not BCD.

```
MOVLW    0 X17   = 0001  0111
ADDLW    0 X 28  = 0010  1000   +
                   0011  1111
```

- Adding these two numbers gives **3Fh**, which is not a BCD number.
- The result must have been **45**.
- To resolve this issue add **06h** to **3Fh**. This gives

3Fh + 06h = 45h

Problem of adding 2 BCD Numbers

- Now consider example of adding **52h** and **87h**.
- Adding these two numbers gives **D9h**, which is not a BCD number.
- The result must have been **139**.
- To resolve this issue add **60h** to **3Fh**. This gives

D9h + 60h = 139h

- **Conclusion:** If non-BCD digit appears in **lower nibble add 06H**, if non-BCD digit appears in **higher nibble add 60H**.

DAW Instruction

- DAW instruction is used in PIC to resolve the BCD addition problem.
- DAW: **D**ecimal **A**djust **W**REG.
- DAW works only on operand in WREG.
- DAW will add 6 to the lower nibble or higher nibble depending upon the requirement.
- In case if there is no need of correction factor, no factor will be added.

```
MOVLW   0 X 47      ; WREG=47H FIRST BCD OPERAND
ADDLW   0 X 25      ; HEX(BINARY) ADDITION (WREG=6CH)
DAW                 ; ADJUST FOR BCD ADDITION (WREG=72H)
```

DAW Instruction

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)
```

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
```

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

Sum = 7D+EB+C5+58
*all numbers BCD
* Sum must be BCD

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

- SUBLW
- There is no dedicated subtractor in PIC.
- The operation of subtraction is performed using two steps:
  - Computing two's compliment
  - Performing addition
- The format of SUBLW instruction is

```
SUBLW    K    ;WREG = K - WREG
```

- After execution of SUB if
  - N = 1 or C = 0; the result is negative
  - N = 0 or C = 1; the result is positive

Subtraction

Example 5-5

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.

Subtraction

- SUBWF
- Destination = fileReg – WREG
- Format:

SUBWF fileReg, d

for d = W, WREG = fileReg – WREG

for d = F, fileReg = fileReg - WREG

- In case of negative result, the destination location has 2's compliment of the result.
- In order to restore the result into original number from the 2's compliment form NEGF command is used.

Subtraction

Example 5-6

Write a program to subtract 4C – 6E.

Solution:

- SUBWF

```
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.

Subtraction

- **SUBWFB**
- This instruction is used for multibyte subtraction and takes care of borrow from lower byte.
- If C = 0 or N=1, prior to execution of SUBWFB, then it subtracts 1 from the result.
- Destination = fileReg – WREG – Borrow
- Format:

SUBWFB fileReg, d

for d = W, WREG = fileReg – WREG – Borrow'

for d = F, fileReg = fileReg - WREG – Borrow'

Subtraction

Example 5-7

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:

- **SUBWFB**

2762 H
1296 H −

```
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.

Subtraction

- SUBFWB
- This instruction is used for multibyte subtraction and takes care of borrow from lower byte.
- If C = 0 or N=1, prior to execution of SUBWFB, then it subtracts 1 from the result.
- Destination = WREG – fileReg – Borrow
- Format:

SUBFWB fileReg, d

for d = W, WREG = WREG – fileReg – Borrow

for d = F, fileReg = WREG - fileReg – Borrow

Multiplication of Unsigned Numbers

- In PIC only byte by byte multiplication is possible.
- These bytes are assumed to be unsigned bytes.
- For multiplication, one operand must be in WREG and other operand must be a literal.
- Format:

MULLW K

- The result goes into two 1-byte registers i.e PRODH and PRODL. Why?

Product of two 8-bit numbers can have a maximum value of FE01H which needs 16-bits.

Multiplication of Unsigned Numbers

- The lower byte of the result goes into PRODL.
- The higher byte of the result goes into PRODH.
- Both PRODL and PRODH are special function registers.

```
MOVLW   0 X 25          ; load 25H to WREG  ( WREG = 25H)
MULLW    0 X 65         ; 25H * 65H = E99 where
                         ; PRODH = 0EH  & PRODL = 99H
MOVLW   0 X 25          ; load 26H to WREG  ( WREG = 25H)

ADDWF   PRODH,  W
ADDWF   PRODL,   W
```

Division of Unsigned Numbers

- There is no instruction in PIC18 to perform division.
- Division can be performed using repetitive subtractions.

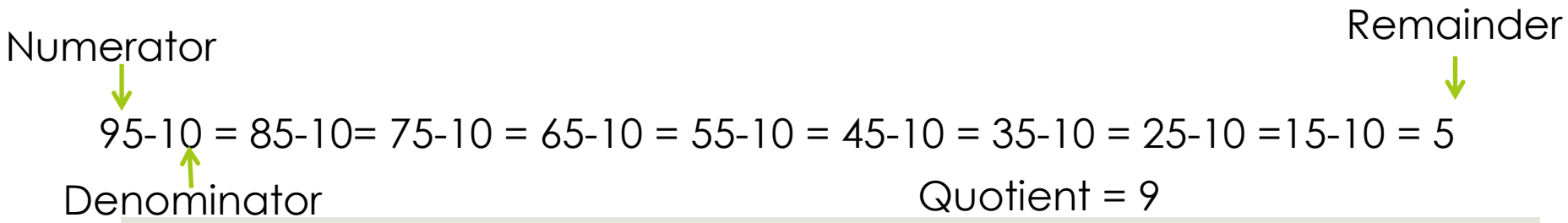
Multiplication: $5 \times 4 = 4 + 4 + 4 + 4 + 4 = 20$

Division(20/4): $20 - 4 - 4 - 4 - 4 - 4 = 0$

Division(20/5): $20 - 5 - 5 - 5 - 5 = 0$

Division of Unsigned Numbers

- Following steps are used to perform division in PIC18.
  - i. Place the numerator into a fileReg
  - ii. Subtract the denominator from numerator
  - iii. Keep on performing step 2, until the remainder (content of fileReg) becomes smaller than the denominator
  - iv. Now the number of times subtraction performed is the quotient and remainder is the number in fileReg



Division of Unsigned Numbers

```
NUM    EQU   0x19          ;set aside fileReg
MYQ    EQU   0x20
MYNMB  EQU   D'95'
MYDEN  EQU   D'10'
       CLRF  MYQ           ;quotient = 0
       MOVLW MYNMB         ;WREG = 95
       MOVWF NUM           ;numerator = 95
       MOVLW MYDEN         ;WREG = denominator = 10
B1     INCF  MYQ,F         ;increment quotient for every 10 subtr
       SUBWF NUM,F         ;subtract 10 (F = F - W)
       BC    B1            ;keep doing it until C = 0
       DECF  MYQ,F         ;once too many
       ADDWF NUM,F         ;add 10 back to get remainder
```

10 / 95

Application of Division for interfacing external ADC

Example 5-9

Analyze the program in Example 5-8 for a numerator of 253.

Solution:

To convert a binary (hex) value to decimal, we divide it by 10 repeatedly until the quotient is less than 10. After each division the remainder is saved. In the case of an 8-bit binary, such as FDH, we have 253 decimal, as shown below.

```
                 Quotient     Remainder
253/10  =        25           3  (low digit)
25/10   =        2            5  (middle digit)
                              2  (high digit)
```

Therefore, we have FDH = 253. In order to display this data, it must be converted to ASCII, which is described in a later section in this chapter.